A Theory of Local Learning, the Learning Channel, and the Optimality of Backpropagation

PIERRE BALDI* and PETER SADOWSKI
Department of Computer Science
University of California, Irvine
Irvine, CA 92697-3435

pfbaldi,psadowsk@uci.edu

Abstract

In a physical neural system, where storage and processing are intimately intertwined, the rules for adjusting the synaptic weights can only depend on variables that are available locally, such as the activity of the pre- and post-synaptic neurons, resulting in *local learning rules*. A systematic framework for studying the space of local learning rules is obtained by first specifying the nature of the local variables, and then the functional form that ties them together into each learning rule. Such a framework enables also the systematic discovery of new learning rules and exploration of relationships between learning rules and group symmetries. We study polynomial local learning rules stratified by their degree and analyze their behavior and capabilities in both linear and non-linear units and networks. Stacking local learning rules in deep feedforward networks leads to *deep local learning*. While deep local learning can learn interesting representations, it cannot learn complex input-output functions, even when targets are available for the top layer. Learning complex input-output functions requires *local deep learning* where target information is communicated to the deep layers through a backward *learning channel*. The nature of the communicated information about the targets and the structure of the learning channel partition the space of learning algorithms. For any learning algorithm, the *capacity* of the learning channel can be defined as the number of bits provided about the error gradient per weight, divided by the number of required operations per weight. We estimate the capacity associated with several learning algorithms and show that backpropagation outperforms them by simultaneously maximizing the information rate and minimizing the computational cost. This result is also shown to be true for recurrent networks, by unfolding them in time. The theory clarifies the concept of Hebbian learning, establishes the power and limitations of local learning rules, introduces the learning channel which enables a formal analysis of the optimality of backpropagation, and explains the sparsity of the space of learning rules discovered so far.

Keywords: machine learning; neural networks; deep learning; backpropagation; learning rules; Hebbian learning; learning channel; recurrent networks; recursive networks; supervised learning; unsupervised learning.

1 Introduction

The deep learning problem can be viewed as the problem of learning the connection weights of a large computational graphs, in particular the weights of the deep connections that are far away from the inputs or outputs [51]. In spite of decades of research, only very few algorithms have been proposed to try to address this task. Among the most important ones, and somewhat in opposition to each other, are backpropagation [50] and Hebbian learning [26]. Backpropagation has been the dominant algorithm, at least in terms of successful applications, which have ranged over the years from computer vision [35] to high-energy physics [10]. In spite of many attempts, no better algorithm has been found, at least within the standard supervised learning framework. In contrast to backpropagation which is a well defined algorithm–stochastic gradient descent–Hebbian learning has remained a more nebulous concept, often associated with notions of biological and unsupervised learning. While less successful than backpropagation in applications, it has periodically inspired the development of theories aimed at capturing the essence of neural learning [26, 22, 29]. Within this general context, the goal of this work is to create a precise framework to organize and study the space of learning rules and their properties and address several questions, in particular: (1) What is Hebbian learning? (2) What are the capabilities and limitations of Hebbian learning? (3) What are the connections between

*Contact author

Hebbian learning and backpropagation? (4) Are there other learning algorithms better than backpropagation? These questions are addressed in two parts: the first part focuses on Hebbian learning, the second part on backpropagation.

1.1 The Deep Learning Problem

At the core of many neural system models is the idea that information is stored in synapses and typically represented by a "synaptic weight". While synapses could conceivably be far more complex (e.g. [36]) and require multiple variables for describing their states, for simplicity here we will use the single synaptic weight framework, although the same ideas can readily be extended to more complex cases. In this framework, synapses are faced with the task of adjusting their individual weights in order to store relevant information and collectively organize in order to sustain neural activity leading to appropriately adapted behavior at the level of the organism. This is a daunting task if one thinks about the scale of synapses and how remote they can be from sensory inputs and motor outputs. Suffice it to say that when rescaled by a factor of 10^6, a synapse is the size of a fist and the bow of the violin, or the tennis racket, it ought to help control is 1,000 miles away. This is the core of the deep learning problem.

1.2 The Hebbian Learning Problem

Donald Hebb is credited with being among the first to think about this problem and attempt to come up with a plausible solution in his 1949 book *The Organization of Behavior* [26]. However, Hebb was primarily a psychologist and his ideas were stated in rather vague terms, such as: "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" often paraphrased as "Neurons that fire together wire together". Not a single equation can be found in his book.

While the concept of Hebbian learning has played an important role in the development of both neuroscience and machine learning, its lack of crispness becomes obvious as soon as one raises simple questions like: Is the backpropagation learning rule Hebbian? Is Oja's learning rule [44] Hebbian? Is a rule that depends on a function of the output [31] Hebbian? Is a learning rule that depends only on the input Hebbian? and so forth. This lack of crispness is more than a simple semantic issue. While it may have helped the field in its early stages–in the same way that vague concepts like "gene" or "consciousness" may have helped molecular biology or neuroscience, it has also prevented clear thinking to address basic questions regarding, for instance, the behavior of linear networks under Hebbian learning, or the capabilities and limitations of Hebbian learning in both shallow and deep networks.

At the same time, there have been several attempts at putting the concept of Hebbian learning at the center of biological learning [22, 29]. Hopfield proposed to use Hebbian learning to store memories in networks of symmetrically connected threshold gates. While the resulting model is elegant and amenable to interesting analyses, it oversimplifies the problem by considering only shallow networks, where all the units are visible and have targets. Fukushima proposed the neocognitron architecture for computer vision, inspired by the earlier neurophysiological work of Hubel and Wiesel [30], essentially in the form of a multi-layer convolutional neural network. Most importantly for the present work, Fukushima proposed to learn the parameters of the neocognitron architecture in a self-organized way using some kind of Hebbian mechanism. While the Fukushima program has remained a source of inspiration for several decades, a key result of this paper is to show that such a program *cannot* succeed at finding an optimal set of weights in a feedforward architecture, regardless of which specific form of Hebbian learning is being used.

1.3 The Space of Learning Rules and its Sparsity Problem

Partly related to the nebulous nature of Hebbian learning, is the observation that so far the entire machine learning field has been able to come up only with very few learning rules like the backpropagation rule and Hebb's rule. Other familiar rules, such as the perceptron learning rule [49], the delta learning rule [55], and Oja's rule [44], can be viewed as special cases of, or variations on, backpropagation or Hebb (Table 1). Additional variations are found also in, for instance, [13, 32, 37], and discussions of learning rules from a general standpoint in [4, 34]. This creates a potentially unsatisfactory situation given that of the two most important learning algorithms, the first one could have been derived by Newton or Leibniz, and the second one is shrouded in vagueness. Furthermore, this raises the broader question of the nature of the space of learning rules. In particular, why does the space seem so sparse? Are there new rules that remain to be discovered in this space?

Learning Rule	Expression
Simple Hebb	$\Delta w_{ij} \propto O_i O_j$
Oja	$\Delta w_{ij} \propto O_i O_j - O_i^2 w_{ij}$
Perceptron	$\Delta w_{ij} \propto (T - O_i) O_j$
Delta	$\Delta w_{ij} \propto (T - O_i) f'(S_i) O_j$
Backprogation	$\Delta w_{ij} \propto B_i O_j$

Table 1: Common learning rules and their on-line expressions. O_i represents the activity of the postsynaptic neuron, O_j the activity of the presynaptic neuron, and w_{ij} the synaptic strength of the corresponding connection. B_i represents the back-propagated error in the postsynaptic neuron. The perceptron and Delta learning rules were originally defined for a single unit (or single layer), in which case T is the readily available output target.

2 A Framework for Local Learning Rules

The origin of the vagueness of the Hebbian learning idea is that it indiscriminately mixes two fundamental but distinct ideas: (1) learning ought to depend on local information associated with the pre- and post-synaptic neurons; and (2) learning ought to depend on the correlation between the activities of these neurons, yielding a spectrum of possibilities on how these correlations are computed and used to change the synaptic weights. The concept of local learning rule, mentioned but not exploited in [4], is more fundamental than the concept of Hebbian learning rule, as it explicitly exposes the more general notion of locality, which is implicit but somehow hidden in the vagueness of the Hebbian concept.

2.1 The Concept of Locality

To address all the above issues, the first observation is that in a physical implementation a learning rule to adjust a synaptic weight can only include *local* variables. Thus to bring clarity to the computational models, *one must first define which variables are to be considered local in a given model*. Consider the backpropagation learning rule $\Delta w_{ij} \propto B_i O_j$ where B_i is the postsynaptic backpropagated error and O_j is the presynaptic activity. If the backpropagated error is not considered a local variable, then backpropagation is not a local learning rule, and thus is not Hebbian. If the backpropagated error is considered a local variable, then backpropagation may be Hebbian, both in the sense of being local and of being a simple product of local pre- and post-synaptic terms. [Note that, even if considered local, the backpropagated error may or may not be of the same nature (e.g. firing rate) as the presynaptic term, and this may invalidate its Hebbian character depending, again, on how one interprets the vague Hebbian concept.]

Once one has decided which variables are to be considered local in a given model, then one can generally express a learning rule as

$$\Delta w_{ij} = F(\text{local variables}) \tag{1}$$

for some function F. A systematic study of local learning requires a systematic analysis of many cases in terms of not only the functions F, but also in terms of the computing units and their transfer functions (e.g. linear, sigmoidal, threshold gates, rectified linear, stochastic, spiking), the network topologies (e.g. shallow/deep, autoencoders, feed-forward/recurrent), and other possible parameters (e.g. on-line vs batch). Here for simplicity we first consider single processing units with input-output functions of the form

$$O = f(S) = f(\sum_j w_j I_j) \tag{2}$$

where I is the input vector and the transfer function f is the identity in the linear case, or the [0,1] logistic function $\sigma_{[0,1]}(x) = 1/(1+e^{-x})$, or the [-1,1] hyperbolic tangent function $\sigma_{[-1,1]}(x) = (1-e^{-2x})/(1+e^{-2x})$, or the corresponding threshold functions $\tau_{[0,1]}$ and $\tau_{[-1,1]}$. When necessary, the bias is included in this framework by considering

that the input value I_0 is always set to 1 and the bias is provided by corresponding weight w_0. In the case of a network of N such units, we write

$$O_i = f(S_i) = f(\sum_j w_{ij} O_j) \tag{3}$$

where in general we assume that there are no self-connections (w_{ii}=0). In general, the computing units can be subdivided into three subsets corresponding to input units, output units, and hidden units. While this formalism includes both feedforward and recurrent networks, in the first part of the paper we will focus primarily on feedforward networks. However issues of feedback and recurrent networks will become important in the second part.

Within this general formalism, we typically consider first that the local variables are the presynaptic activity, the postynaptic activity, and w_{ij} so that

$$\Delta w_{ij} = F(O_i, O_j, w_{ij}) \tag{4}$$

In supervised learning, in a model where the target T_i is considered a local variable, the rule can have the more general form

$$\Delta w_{ij} = F(T_i, O_i, O_j, w_{ij}) \tag{5}$$

For instance, we will consider cases where the output is clamped to the value T_i, or where the error signal $T_i - O_i$ is a component of the learning rule. The latter is the case in the perceptron learning algorithm, or in the deep targets algorithm described below, with backpropagation as a special case. Equation 5 represents a local learning rule if one assumes that there is a target T_i that is locally available. Targets can be clearly available and local for the output layer. However the generation and local availability of targets for deep layers is a fundamental, but separate, question that will be addressed in later sections. *Thus it is essential to note that the concept of locality is orthogonal to the concept of unsupervised learning*. An unsupervised learning rule can be non-local if F depends on activities or synaptic weights that are far away in the network. Likewise a supervised learning rule can be local, if the target is assumed to be a local variable. Finally, we also assume that the learning rate η is a local variable contained in the function F. For simplicity, we will assume that the value of η is shared by all the units, although more general models are possible.

In short, it is time to move away from the vagueness of the term "Hebbian learning" and replace it with a clear definition, in each situation, of: (1) which variables are to be considered local; and (2) which functional form is used to combine the local variables into a local learning rule. A key goal is then to systematically study the properties of different local rules across different network types.

2.1.1 Spiking versus Non-Spiking Neurons

The concept of locality (Equation 1) is completely general and applies equally well to networks of spiking neurons and non-spiking neurons. The analyses of specific local learning rules in Sections 3-5 are conducted for non-spiking neurons, but some extensions to spiking neurons are possible (see, for instance, [59]). Most of the material in Sections 6-8 is general again and is applicable to networks of spiking units. The main reason is that these sections are concerned primarily with the propagation of information about the targets from the output layer back to the deeper layers, regardless of how this information is encoded, and regardless of whether non-spiking or spiking neurons are used in the forward, or backward, directions.

2.2 Coordinate Transformations and Symmetries

This subsection is not essential to follow the rest of this paper and can initially be skipped. When studying local learning rules, it is important to look at the effects of coordinate transformations and various symmetries on the learning rules. While a complete treatment of these operations is beyond our scope, we give several specific examples below. In general, applying coordinate changes or symmetries can bring to light some important properties of a learning rule, and shows in general that the function F should not be considered too narrowly, but rather as a member of a class.

2.2.1 Example 1: Range Transformation (Affine Transformation)

For instance, consider the narrow definition of Hebb's rule as $\Delta w_{ij} \propto O_i O_j$ applied to threshold gates with binary inputs. This definition makes some sense if the threshold gates are defined using a $[-1, 1]$ formalism, but is problematic over a $[0, 1]$ formalism because it results in $\Delta w_{ij} = O_i O_j$ being 0 in three out of four cases, and always positive and equal to 1 in the remaining fourth case. Thus the narrow definition of Hebb's rule over a $[0, 1]$ system should be modified using the corresponding affine transformation. However the new expression will have to be in the *same functional class*, i.e. in this case quadratic function over the activities. The same considerations apply when sigmoid transfer functions are used.

More specifically, [0,1] networks are transformed into [-1,1] networks through the transformation $x \rightarrow 2x - 1$ or vice versa through the transformation $x \rightarrow (x + 1)/2$. It is easy to show that a polynomial local rule in one type of network is transformed into a polynomial local rule of the same degree in the other type of network. For instance, a quadratic local rule with coefficients $\alpha_{[0,1]}, \beta_{[0,1]}, \gamma_{[0,1]}, \delta_{[0,1]}$ of the form

$$\Delta w_{ij} \propto \alpha_{[0,1]} O_i O_j + \beta_{[0,1]} O_i + \gamma_{[0,1]} O_j + \delta_{[0,1]} \tag{6}$$

is transformed into a rule with coefficients $\alpha_{[-1,1]}, \beta_{[-1,1]}, \gamma_{[-1,1]}, \delta_{[1,1]}$ through the homogeneous system:

$$\begin{aligned}
\alpha_{[-1,1]} &= 4\alpha_{[0,1]} && (7)\\
\beta_{[-1,1]} &= 2\beta_{[0,1]} - 2\alpha_{[0,1]} && (8)\\
\gamma_{[-1,1]} &= 2\gamma_{[0,1]} - 2\alpha_{[0,1]} && (9)\\
\delta_{[-1,1]} &= \delta_{[0,1]} + \alpha_{[0,1]} - \beta_{[0,1]} - \gamma_{[0,1]} && (10)
\end{aligned}$$

Note that no non-zero quadratic rule can have the same form in both systems, even when trivial multiplicative coefficients are absorbed into the learning rate.

2.2.2 Example 2: Permutation of Training Examples

Learning rules may be more or less sensitive to permutations in the order in which examples are presented. In order to analyze the behavior of most rules, here we will assume that they are not sensitive to the order in which the examples are presented, which is generally the case if all the training examples are treated equally, and the on-line learning rate is small and changes slowly so that averages can be computed over entire epochs (see below).

2.2.3 Example 3: Network Symmetries

When the same learning rule is applied isotropically, it is important to examine its behavior under the symmetries of the network architecture to which it is applied. This is the case, for instance, in Hopfield networks where all the units are connected symmetrically to each other (see next section), or between fully connected layers of a feedforward architecture. In particular, it is important to examine whether differences in inputs or in weight initializations can lead to symmetry breaking. It is also possible to consider models where different neurons, or different connections, use different rules, or rules in the same class (like Equation 6) but with different coefficients.

2.2.4 Example 4: Hypercube Isometries

As a fourth example [3] consider a Hopfield network [29] consisting of N threshold gates, with ± 1 outputs, connected symmetrically to each other ($w_{ij} = w_{ji}$). It is well known that such a system and its dynamics is characterized by the quadratic energy function $E = -(1/2) \sum_{i,j} w_{ij} O_i O_j$ (note that the linear terms of the quadratic energy function are taken into account by $O_0 = 1$). The quadratic function E induces an acyclic orientation $\mathcal{O}$ of the N-dimensional Hypercube $\mathcal{H}^N = [-1, 1]^N$ where the edge between two neighboring (i.e. at Hamming distance 1) state spaces x and y is oriented from x to y if and only if $E(x) > E(y)$. Patterns or "memories" are stored in the weights of the system by applying the simple Hebb rule $\Delta w_{ij} \propto O_i O_j$ to the memories. Thus a given training set S produces a corresponding set of weights, thus a corresponding energy function, and thus a corresponding acyclic orientation $\mathcal{O}(S)$ of the hypercube. Consider now an isometry h of the N-dimensional hypercube, i.e. a one-to-one function from $\mathcal{H}^N$ to $\mathcal{H}^N$ that preserves the Hamming distance. It is easy to see that all isometries can be generated by composing two kinds of elementary operations: (1) permuting two components; and (2) inverting the sign of a component (hence

the isometries are linear). It is then natural to ask what happens to $\mathcal{O}(S)$ when h is applied to $\mathcal{H}^N$ and thus to S. It can be shown that under the simple Hebb rule the "diagram commutes" (Figure 2.1). In other words, $h(S)$ is a new training set which leads to a new acyclic orientation $\mathcal{O}(h(S))$ and

$$h(\mathcal{O}(S)) = \mathcal{O}(h(S)) \tag{11}$$

Thus the simple Hebb rule is invariant under the action of the isometries of the hypercube. In Appendix A, we show it is the only rule with this property.

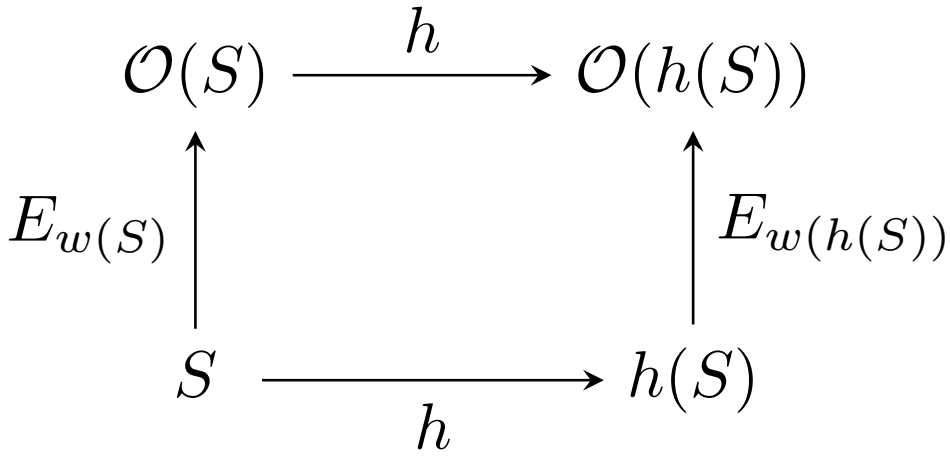

Figure 2.1: Commutative diagram for the simple Hebb rule in a Hopfield network. Application of the simple Hebb's rule to a set S binary vectors over the $[-1, 1]^N$ hypercube in a Hopfield network with N units yields a set of symmetric weights $w_{ij} = w_{ji}$ and a corresponding quadratic energy function $E_{w(S)}$, which ultimately produces a directed acyclic orientation of the hypercube $\mathcal{O}(S)$ directing the dynamics of the network towards minima of $E_{w(S)}$. An isometry h over the hypercube yields a new set of vectors $h(S)$ hence, by application of the same Hebb rule, a new set of weights, a new energy function $E_{w(h(S))}$, and a new acyclic orientation such that $h(\mathcal{O}(S)) = \mathcal{O}(h(S))$.

2.3 Functional Forms

Within the general assumption that $\Delta w_{ij} = F(O_i, O_j, w_{ij})$, or $\Delta w_{ij} = F(T_i, O_i, O_j, w_{ij})$ in the supervised case, one must consider next the functional form of F. Among other things, this allows one to organize and stratify the space of learning rules. As seen above, the function F cannot be defined too narrowly, as it must be invariant to certain changes, and thus one is primarily interested in classes of functions. In this paper, we focus exclusively on the case where F is a polynomial function of degree n (e.g. linear, quadratic, cubic) in the local variables, although other functional forms could be considered, such as rational functions or power functions with rational exponents. Most rules that are found in the neural network literature correspond to low degree polynomial rules. Thus we consider functions F comprising a sum of terms of the form $\alpha O_i^{n_i} O_j^{n_j} w_{ij}^{n_{w_{ij}}}$ (or $\alpha T_i^{n_{T_i}} O_i^{n_i} O_j^{n_j} w_{ij}^{n_{w_{ij}}}$) where α is a real coefficient [in this paper we assume that the constant α is the same for all the weights, but many of the analyses carry over to the case where different weights have different coefficients although such a system is not invariant under relabeling of the neurons]; n_{T_i}, n_i, n_j, and $n_{w_{ij}}$ are non-negative integers satisfying $n_{T_i} + n_i + n_j + n_{w_{ij}} \leq n$. In this term, the *apparent* degree of w is $n_{w_{ij}}$ but the *effective* degree of w may be higher because O_i depends also on w_{ij}, typically in a linear way at least around the current value of O_i. In this case, the effective degree of w_{ij} in this term is $n_i + n_{w_{ij}}$. [For instance, consider a rule of the form $\Delta w_{ij} = w_{ij} O_i^2 I_j$, with a linear unit $O_i = \sum_k w_{ik} I_k$. The apparent degree of the rule in w_{ij} is 1, but the effective degree is 3.] Finally, we let d ($d \leq n$) denote the highest effective degree of w, among all the terms in F. As we shall see, n and d are the two main numbers of interest used to stratify the polynomial learning rules.

2.4 Terminology

We do not expect to be able to change how the word Hebbian is used but the recommendation, used in the rest of this paper, is to replace Hebbian with the more precise concept of local learning rule, which assumes a pre-existing definition of which variables are to be considered local. Within the set of local learning rules, it is easy to see that in general linear ($n = 1$) learning rules of the form $\Delta w_{ij} \propto \alpha O_i + \beta O_j + \gamma w_{ij}$ are not very useful (in fact the same is true also for rules of the form $\Delta w_{ij} \propto h(O_i) + g(O_j) + k(w_{ij})$ for any functions h, g, and k). Thus, for a local learning rule to be interesting it must be at least quadratic.

Within quadratic learning rules, one could adopt the position that only $\Delta w_{ij} \propto O_i O_j$ should be called Hebbian. At the other extreme, one could adopt the position that all quadratic rules with $n = 2$ should be called Hebbian. This would include the correlation rule

$$\Delta w_{ij} \propto (O_i - E(O_i))(O_j - E(O_j)) \tag{12}$$

which requires information about the averages $E(O_i)$ and $E(O_j)$ over the training examples, and other rules of the form

$$\Delta w_{ij} \propto \alpha O_i O_j + \beta O_i + \gamma O_j + \delta \tag{13}$$

Note that this is not the most general possible form since other terms can also be included (i.e. terms in O_i^2, O_j^2, w_{ij}, w_{ij}^2, $w_{ij} O_i$, and $w_{ij} O_j$) and will be considered below. Note also that under any of these definitions of Hebbian, Oja's rule

$$\Delta w_{ij} \propto O_i O_j - O_i^2 w_{ij} \tag{14}$$

is local, but not Hebbian since it is not quadratic in the local variables O_i, O_j, and w_{ij}. Rather it is a cubic rule with $n = 3$ and $d = 3$.

In any case, to avoid these terminological complexities, which result from the vagueness of the Hebbian concept, we will: (1) focus on the concept of locality; (2) stratify local rules by their degrees n and d; and (3) reserve "simple Hebb" for the rule $\Delta w_{ij} \propto O_i O_j$, avoiding to use "Hebb" in any other context.

2.5 Time-scales and Averaging

We assume a training set consisting of M inputs $I(t)$ for $t = 1, \ldots, M$ in the unsupervised case, and M input-target pairs $(I(t), T(t))$ in the supervised case. On-line learning with local rules will exhibit stochastic fluctuations and the weights will change at each on-line presentation. However, with a small learning rate and randomized order of example presentation, we expect the long term behavior to be dominated by the average values of the weight changes computed over one epoch. Thus we assume that O varies rapidly over the training data, compared to the synaptic weights w which are assumed to remain constant over an epoch. The difference in time-scales is what enables the analysis since we assume the weight w_{ij} remains essentially constant throughout an epoch and we can compute the average of the changes induced by the training data over an entire epoch. While the instantaneous evolution of the weights is governed by the relationship

$$w_{ij}(t+1) = w_{ij}(t) + \eta \Delta w_{ij}(t) \tag{15}$$

the assumption of small η allow us to average this relation over an entire epoch and write

$$w_{ij}(k+1) = w_{ij}(k) + \eta E(\Delta w_{ij}) \tag{16}$$

where the index k is now over entire epochs, and E is the expectation taken over the corresponding epoch. Thus, in the analyses, we must first compute the expectation E and then solve the recurrence relation (Equation 16), or the corresponding differential equation.

2.6 Initial Roadmap

The article is subdivided into two main parts. In the first part, the focus is on Hebbian learning, or more precisely on local learning rules. Because we are restricting ourselves to learning rules with a polynomial form, the initial goal is to estimate expectations of the form $E(O_i^{n_i} O_j^{n_j} w_{ij}^{n_{w_{ij}}})$ in the unsupervised case, or $E(T_i^{n_{T_i}} O_i^{n_i} O_j^{n_j} w_{ij}^{n_{w_{ij}}})$ in the supervised case. Because of the time-scale assumption, within an epoch we can assume that w_{ij} is constant and therefore the corresponding term factors out of the expectation. Thus we are left with estimating terms of the form $E(O_i^{n_i} O_j^{n_j})$ in the unsupervised case, or $E(T_i^{n_{T_i}} O_i^{n_i} O_j^{n_j})$ in the supervised case.

In terms of architectures, we are primarily interested in deep feedforward architectures and thus we focus first on layered feedforward networks, with local supervised or unsupervised learning rules, where local learning is applied layer by layer in batch mode, starting from the layer closest to the inputs. *In this feedforward framework, within any single layer of units, all the units learn independently of each other given the inputs provided by the previous layer. Thus in essence the entire problem reduces to understanding learning in a single unit and, using the notation of Equation 2, to estimating the expectations $E(O^{n_O} I_j^{n_j})$ in the unsupervised case, or $E(T^{n_T} O^{n_O} I_j^{n_j})$ in the supervised case, where I_j are the inputs and O is the output of the unit being considered.* In what follows, we first consider the linear case (Section 3) and then the non-linear case (Section 4). We then give examples of how new learning rules can be derived (Section 5).

In the second part, the focus is on backpropagation. We first study the limitations of purely local learning in shallow or deep networks, also called deep local learning (Section 6). To go beyond these limitations, naturally leads to the introduction of local deep learning algorithms and deep targets algorithms, and the study of the properties of the backward learning channel and the optimality of backpropagation (Sections 7 and 8).

3 Local Learning in the Linear Case

The study of feedforward layered linear networks is thus reduced to the study of a single linear unit of the form $O = \sum_{i=0}^{N} w_i I_i$. In this case, to understand the behavior of any local learning rule, one must compute expectations of the form

$$E(T^{n_T} O^{n_O} I_i^{n_{I_i}} w_i^{n_i}) = w_i^{n_i} E\left[T^{n_T} (\sum_k w_k I_k)^{n_O} I_i^{n_{I_i}}\right] \tag{17}$$

This encompasses also the unsupervised case by letting $n_T = 0$. Thus this expectation is a polynomial in the weights, with coefficients that correspond to the statistical moments of the training data of the form $E(T^{n_T} I_i^{n_\alpha} I_k^{n_\beta})$. When this polynomial is linear in the weights ($d \leq 1$), the learning equation can be solved exactly using standard methods. When the effective degree is greater than 1 ($d > 1$), then the learning equation can be solved in some special cases, but not in the general case.

To look at this analysis more precisely, here we assume that the learning rule only uses data terms of order two or less. Thus only the means, variances, and covariances of I and T are necessary to compute the expectations in the learning rule. For example, a term $w_i TO$ is acceptable, but not $w_i TO^2$ which requires third-order moments of the data of the form $E(TI_iI_j)$ to compute its expectation. To compute all the necessary expectations systematically, we will use the following notations.

3.1 Notations

- All vectors are column vectors.
- A' denotes the transpose of the matrix A, and similarly for vectors.
- u is the N dimensional vector of all ones: $u' = (1, 1, \ldots, 1)$.
- $\circ$ is the Hadamard or Schur product, i.e. the component-wise product of matrices or vectors of the same size. We denote by $v^{(k)}$ the Schur product of v with itself k times, i.e. $v^{(k)} = v \circ v \ldots \circ v$.
- $\mathrm{diag}M$ is an operator that creates a vector whose components are the diagonal entries of the square matrix M.
- When applied to a vector $\mathrm{Diag}v$ represents the square diagonal matrix whose components are the components of the vector M.

- $\mathrm{Diag}M$ represents the square diagonal matrix whose entries on the diagonal are identical to those of M (and 0 elsewhere), when M is a square matrix.
- For the first order moments, we let $E(I_i) = \mu_i$ and $E(T) = \mu_T$. In vector form, $\mu = (E(I_i))$.
- For the second order moments, we introduce the matrix $\Sigma_{II'} = (E(I_i I_j)) = (\mathrm{Cov}(I_i, I_j) + \mu_i \mu_j)$

3.2 Computation of the Expectations

With these notations, we can compute all the necessary expectations. Thus:

- In Table 2, we list all the possible terms with $n = 0$ or $n = 1$ and their expectations.
- In Table 3, we list all the possible quadratic terms with $n = 2$ and their expectations.
- In Table 4, we list all the possible cubic terms with $n = 3$, requiring only first and second moments of the data, and their expectations.
- In Table 5, we list all the possible terms of order n, requiring only first and second moments of the data, and their expectations.

Note that in Table 5, for the term $w_i^{n-2} I_i^2$ the expectation in matrix form can be written as $w^{(n-2)} \circ \mathrm{Diag}\Sigma_{II'} = \mathrm{Diag}(\Sigma_{II'})w \circ w^{(n-3)}$. Thus in the cubic case where $n = 3$, the expectation has the form $\mathrm{Diag}(\Sigma_{II'})w$. Likewise, for $w_i^{n-2} I_i T$ the expectation in matrix form can also be written as $w^{(n-2)} \circ \Sigma_{IT'} = w^{(n-3)} \circ (\mathrm{diag}\Sigma_{IT'})w$. Thus in the cubic case where $n = 3$, the expectation has the form $(\mathrm{diag}\Sigma_{IT'})w$.

Note also that when there is a bias term, we consider that the corresponding input I_0 is constant and clamped to 1, so that $E(I_0^n) = 1$ for any n, and I_0 can simply be ignored in any product expression.

3.3 Solving the Learning Recurrence Relation in the Linear Case ($d \leq 1$)

When the effective degree d satisfies $d \leq 1$, then the recurrence relation provided by Equation 18 is linear for any value of the overall degree n. Thus it can be solved by standard methods provided all the necessary data statistics are available to compute the expectations. More precisely, computing the expectation over one epoch leads to the relation

$$w(k+1) = Aw(k) + b \tag{18}$$

Starting from $w(0)$ and iterating this relation, the solution can be written as

$$w(k) = A^k w(0) + A^{k-1} b + A^{k-2} b + \ldots + Ab + b = A^k w(0) + [\mathrm{I} + A + A^2 + \ldots + A^{k-1}]b \tag{19}$$

where I denotes the identity matrix. Furthermore, if $A - \mathrm{I}$ is an invertible matrix, this expression can be written as

$$w(k) = A^k w(0) + (A^k - \mathrm{I})(A - \mathrm{I})^{-1} b = A^k w(0) + (A - \mathrm{I})^{-1}(A^k - \mathrm{I})b \tag{20}$$

When A is symmetric, there is an orthonormal matrix C such that $A = CDC^{-1} = CDC'$, where $D = \mathrm{Diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix and $\lambda_1, \ldots, \lambda_N$ are the real eigenvalues of A. Then for any power k we have $A^k = CD^k C^{-1} = C\mathrm{Diag}(\lambda_1^k, \ldots, \lambda_N^k)C^{-1}$ and $A^k - \mathrm{I} = C(D^k - \mathrm{I})C^{-1} = C\mathrm{diag}(\lambda_1^k - 1, \ldots, \lambda_N^k - 1)C^{-1}$ so that Equation 19 becomes

$$w(k) = CD^k C^{-1} w(0) + C[\mathrm{I} + D + D^2 + \ldots + D^{k-1}]C^{-1} b = CD^k C^{-1} w(0) + CEC^{-1} b \tag{21}$$

where $E = \mathrm{Diag}(\xi_1, \ldots, \xi_N)$ is a diagonal matrix with $\xi_i = (\lambda_i^n - 1)/(\lambda_i - 1)$ if $\lambda_i \neq 1$, and $\xi_i = k$ if $\lambda_i = 1$. If all the eigenvalues of A are between 0 and 1 ($0 < \lambda_i < 1$ for every i) then the vector $w(k)$ converges to the vector $C\mathrm{Diag}(1/(1-\lambda_1), \ldots, 1/(1-\lambda_\mathrm{N}))\mathrm{C'b}$. If all the eigenvalues of A are 1, then $w(k) = w(0) + kb$.

3.4 Examples

We now give a few examples of learning equations with $d \leq 1$.

Constant and Linear Terms	Expectation	Matrix Form
c_i $(0,0)$	c_i	$c = (c_i)$
I_i $(1,0)$	μ_i	$\mu = (\mu_i)$
O $(1,1)$	$\sum_j w_j \mu_j$	$(w'\mu)u = (\mu' w)u = (\mathrm{Diag}\mu)w$
w_i $(1,1)$	w_i	$w = (w_i)$
T $(1,0)$	μ_T	$\mu_T u$

Table 2: Constant and Linear Terms and their Expectations in Scalar and Vector Form. The table contains all the constant and linear terms of degree (n, d) equal to $(0, 0)$, $(1, 0)$, and $(1, 1)$ depending only on first order statistics of the data. The horizontal double line separates unsupervised terms (top) from supervised terms (bottom). The terms are sorted by increasing values of the effective degree (d), and then by increasing values of the apparent degree of w_i.

Quadratic Terms	Expectation	Vector Form
I_i^2 $(2,0)$	$\mathrm{Var}I_i + \mu_i^2$	$\mathrm{diag}(\Sigma_{II'})$
$I_i O$ $(2,1)$	$w_i(\mathrm{Var}I_i + \mu_i^2) + \sum_{j \neq i} w_j(\mathrm{Cov}(I_i, I_j) + \mu_i \mu_j)$	$(\mathrm{Cov}I)w + (\mu\mu')w = \Sigma_{II'}w$
$w_i I_i$ $(2,1)$	$w_i \mu_i$	$w \circ \mu = (\mathrm{Diag}\mu)w$
O^2 $(2,2)$	$\sum_i w_i^2(\mathrm{Var}I_i + \mu_i^2) + \sum_{i<j} 2w_i w_j(\mathrm{Cov}(I_i, I_j) + \mu_i \mu_j)$	$(w'\Sigma_{II'}w)u$
$w_i O$ $(2,2)$	$w_i \sum_j w_j \mu_j$	$(w'\mu)w = (\mu' w)w$
w_i^2 $(2,2)$	w_i^2	$w^{(2)} = (w_i^2) = w \circ w$
$I_i T$ $(2,0)$	$\mathrm{Cov}(I_i, T) + \mu_i \mu_T$	$\mathrm{Cov}(I, T) + \mu_T \mu = \Sigma_{IT'}$
T^2 $(2,0)$	$\mathrm{Var}T + \mu_T^2$	$(\mathrm{Var}T + \mu_T^2)u$
OT $(2,1)$	$\sum_i w_i[\mathrm{Cov}(I_i, T) + \mu_i \mu_T]$	$w'\Sigma_{IT'}$
$w_i T$ $(2,1)$	$w_i \mu_T$	$\mu_T w$

Table 3: Quadratic Terms and their Expectations in Scalar and Vector Form. The table contains all the quadratic terms (n, d) where $n = 2$ and $d = 0, 1$, or 2. These terms depend only on the first and second order statistics of the data. The horizontal double line separates unsupervised terms (top) from supervised terms (bottom). The terms are sorted by increasing values of the effective degree (d), and then by increasing values of the apparent degree of w_i.

3.4.1 Unsupervised Simple Hebbian Rule

As an example, consider the simple Hebb rule with $\Delta w_i = \eta I_i O$ ($n = 2$, $d = 1$). Using Table 3 we get in vector form $E(\Delta w) = \eta \Sigma_{II'} w$ and thus

$$w(k) = (\mathrm{I} + \eta \Sigma_{II'})^k w(0) \tag{22}$$

In general, this will lead to weights that grow in magnitude exponentially with the number of epochs. For instance, if all the inputs have mean 0 ($\mu = 0$), variance σ_i^2, and are independent of each other, then

$$w_i(k) = (1 + \eta \sigma_i^2)^k w_i(0) \tag{23}$$

Alternatively, we can use the independence approximation to write

$$w_i(k) = w_i(k-1) + \eta E(O(k-1)I_i) \approx w_i(k-1) + \eta \mu_i E(O(k-1)) = w_i(k-1) + \eta \mu_i \sum_j w_j(k-1)\mu_j \tag{24}$$

Simple Cubic Terms	Expectation	Vector Form
$w_i I_i^2$ $(3,1)$	$w_i(\mathrm{Var}I_i + \mu_i^2)$	$w \circ \mathrm{diag}\Sigma_{II'} = \mathrm{Diag}(\Sigma_{II'})w$
$w_i I_i O$ $(3,2)$	$w_i[w_i(\mathrm{Var}I_i + \mu_i^2) + \sum_{j\neq i} w_j(\mathrm{Cov}(I_i, I_j) + \mu_i\mu_j)]$	$w \circ \Sigma_{II'}w$
$w_i^2 I_i$ $(3,2)$	$w_i^2\mu_i$	$w^{(2)} \circ \mu = w \circ (\mathrm{Diag}\mu)w$
$w_i O^2$ $(3,3)$	$w_i[\sum_i w_i(\mathrm{Var}I_i + \mu_i^2) + \sum_{i<j} 2w_iw_j(\mathrm{Cov}(I_i, I_j) + \mu_i\mu_j)]$	$(w'\Sigma_{II'}w)w$
$w_i^2 O$ $(3,3)$	$w_i^2 \sum_j w_j\mu_j$	$(\mathrm{Diag}\mu w) \circ w^{(2)}$
w_i^3 $(3,3)$	w_i^3	$w^{(3)} = (w_i^3) = w \circ w \circ w$
$w_i I_i T$ $(3,1)$	$w_i(\mathrm{Cov}(I_i, T) + \mu_i\mu_T)$	$w \circ \Sigma_{IT'} = \mathrm{diag}\Sigma_{IT'}w$
$w_i T^2$ $(3,1)$	$w_i(\mathrm{Var}T + \mu_T^2)$	$(\mathrm{Var}T + \mu_T^2)w$
$w_i OT$ $(3,2)$	$w_i^2 E(I_iT) + \sum_{j\neq i} w_iw_j E(I_jT)$	$w \circ (w'\Sigma_{IT'})$
$w_i^2 T$ $(3,2)$	$w_i^2\mu_T$	$\mu_T w^{(2)} = \mu_T w \circ w$

Table 4: Cubic Terms and their Expectations in Scalar and Vector From. The table contains all the terms of degree (n, r) with $n = 3$ and $r = 0, 1, 2$ or 3 that depend only on the first and second order statistics of the data. The horizontal double line separates unsupervised terms (top) from supervised terms (bottom). The terms are sorted by increasing values of the effective degree (d), and then by increasing values of the apparent degree of w_i.

Simple n-th Terms	Expectation	Vector Form
$w_i^{n-2} I_i^2$ $(n, n-2)$	$w_i^{n-2}(\mathrm{Var}I_i + \mu_i^2)$	$w^{(n-2)} \circ \mathrm{diag}\Sigma_{II'}$
$w_i^{n-2} I_i O$ $(n, n-1)$	$w_i^{n-2}[w_i(\mathrm{Var}I_i + \mu_i^2) + \sum_{j\neq i} w_j(\mathrm{Cov}(I_i, I_j) + \mu_i\mu_j)]$	$w^{(n-2)} \circ \Sigma_{II'}w$
$w_i^{n-1} I_i$ $(n, n-1)$	$w_i^{n-1}\mu_i$	$w^{(n-1)} \circ \mu = w^{n-2} \circ (\mathrm{Diag}\mu)w$
$w_i^{n-2} O^2$ (n, n)	$w_i^{n-2}[\sum_i w_i(\mathrm{Var}I_i + \mu_i^2) + \sum_{i<j} 2w_iw_j(\mathrm{Cov}(I_i, I_j) + \mu_i\mu_j)]$	$w'\Sigma_{II'}ww^{(n-2)}$
$w_i^{n-1} O$ (n, n)	$w_i^{n-1} \sum_j w_j\mu_j$	$w^{(n-1)} \circ (\mathrm{Diag}\mu)w$
w_i^n (n, n)	w_i^n	$w^{(n)} = (w_i^n) = w \circ \ldots \circ w$
$w_i^{n-2} I_i T$ $(n, n-2)$	$w_i^{(n-2)}(\mathrm{Cov}(I_i, T) + \mu_i\mu_T)$	$w^{(n-2)} \circ \Sigma_{IT'}$
$w_i^{n-2} T^2$ $(n, n-2)$	$w_i^{n-2}(\mathrm{Var}T + \mu_T^2)$	$(\mathrm{Var}T + \mu_T^2)w^{(n-2)}$
$w_i^{n-2} OT$ $(n, n-1)$	$w_i^{n-1} E(I_iT) + \sum_{j\neq i} w_i^{n-2} w_j E(I_jT)$	$w^{(n-2)} \circ (w'\Sigma_{IT'})$
$w_i^{n-1} T$ $(n, n-1)$	$w_i^{n-1}\mu_T$	$\mu_T w^{(n-1)} = \mu_T w \circ \ldots \circ w$

Table 5: Simple Terms of Order n and their Expectations in Scalar and Vector Form. The table contains all the terms of degree (n, d) with $d = n-2, n-1$, or n that depend only on the first and second order statistics of the data. The horizontal double line separates unsupervised terms (top) from supervised terms (bottom). The terms are sorted by increasing values of the effective degree (d), and then by increasing values of the apparent degree.

which, in vector form, gives the approximation

$$w(k) = w(k-1) + \eta\mu' w(k-1)\mu = (\mathrm{I} + \eta A)w(k-1) \quad \text{or} \quad w(k) = (\mathrm{I} + \eta A)^k w(0) \tag{25}$$

where $A = \mu\mu'$.

3.4.2 Supervised Simple Hebbian Rule (Clamped Case)

As a second example, consider the supervised version of the simple Hebb rule where the output is clamped to some target value T with $\Delta w_i = \eta I_i T$ ($n = 2$, $d = 0$). Using Table 3 we get in vector form $E(\Delta w) = \eta \Sigma_{IT'}$ and thus

$$w(k) = w(0) + \eta k \Sigma_{IT'} \tag{26}$$

In general the weights will grow in magnitude linearly with the number k of epochs, unless $E(I_i T) = 0$ in which case the corresponding weight remains constant $w_i(k) = w_i(0)$.

Note that in some cases it is possible to assume, as a quick approximation, that the targets are independent of the inputs so that $E(\Delta w_i) = \eta E(T) E(I_i) = \eta E(T) \mu_i$. This simple approximation gives

$$w(k) = w(0) + \eta k E(T) \mu \tag{27}$$

Thus the weights are growing linearly in the direction of the center of gravity of the input data.

Thus, in the linear setting, many local learning rules lead to divergent weights. There are notable exceptions, however, in particular when the learning rule is performing some form of (stochastic) gradient descent on a convex objective function.

3.4.3 Simple Anti-Hebbian Rule

The anti-Hebbian quadratic learning rule $\Delta w_i = -\eta I_i O$ ($n = 2$, $d = 1$) performs gradient descent on the objective function $\frac{1}{2} \sum_t O^2(t)$ and will tend to converge to the uninteresting solution where all weights (bias included) are equal to zero.

3.4.4 Gradient Descent Rule

A more interesting example is provided by the rule $\Delta w_i = \eta (T - O) I_i$ ($n = 2$, $d = 1$). Using Table 3 we get in vector form $E(\Delta w) = \eta (\Sigma_{IT'} - \Sigma_{II'})$. The rule is convergent (with properly decreasing learning rate η) because it performs gradient descent on the quadratic error function $\frac{1}{2} \sum_{t=1}^{M} (T(t) - O(t))^2$ converging in general to the linear regression solution.

In summary, when $d \leq 1$ the dynamics of the learning rule can be solved exactly in the linear case and it is entirely determined by the statistical moments of the data, in particular by the means, variances, and covariances of the inputs and targets (e.g. when $n \leq 2$).

3.5 The Case $d \geq 2$

When the effective degree of the weights is greater than one in the learning rule, the recurrence relation is not linear and there is no systematic solution in the general case. It must be noted however that in some special cases, this can result in a Bernoulli or Riccati ($d = 2$) differential equation for the evolution of each weight which can be solved (e.g. [47]). For reasons that will become clear in later sections, let us for instance consider the learning equation

$$\Delta w_i = \eta (1 - w_i^2) I_i \tag{28}$$

with $n = 3$ and $d = 2$. We have

$$E(\Delta w_i) = \eta (1 - w_i^2) \mu_i \tag{29}$$

Dropping the index i, the corresponding Riccati differential equation is given by

$$\frac{dw}{dt} = \eta \mu - \eta \mu w^2 \tag{30}$$

The intuitive behavior of this equation is clear. Suppose we start at, or near, $w = 0$. Then the sign of the derivative at the origin is determined by the sign of μ, and w will either increase and asymptotically converge towards +1

when $\mu > 0$, or decrease and asymptotically converge towards -1 when $\mu < 0$. Note also that $w_{obv1}(t) = 1$ and $w_{obv2}(t) = -1$ are two obvious constant solutions of the differential equation.

To solve the Riccati equation more formally we use the known obvious solutions and introduce the new variable $z = 1/(w - w_{obv}) = 1/(w + 1)$ (and similarly one can introduce the new variable $z = 1/(w - 1)$ to get a different solution). As a result, $w = (1 - z)/z$. It is then easy to see that the new variable z satisfies a linear differential equation. More precisely, a simple calculation gives

$$\frac{dz}{dt} = -2\eta\mu z + \eta\mu \tag{31}$$

resulting in

$$z(t) = Ce^{-2\eta\mu t} + \frac{1}{2} \tag{32}$$

and thus

$$w(t) = \frac{1 - 2Ce^{-2\eta\mu t}}{1 + 2Ce^{-2\eta\mu t}} \quad \text{with} \quad w(0) = \frac{1 - 2C}{1 + 2C} \quad \text{or} \quad C = \frac{1 - w(0)}{2(1 + w(0))} \tag{33}$$

Simulations are shown in Figure 5.2 for the unsupervised case, and Figure 5.3 for the supervised case.

3.5.1 Oja's Rule

An important example of a rule with $r > 1$ is provided by Oja's rule [44]

$$\Delta w_i = \eta(OI_i - O^2 w_i) \tag{34}$$

with $d = 3$, originally derived for a linear neuron. The idea behind this rule is to control the growth of the weights induced by the simple Hebb rule by adding a decay term. The form of the decay term can easily be obtained by requiring the weights to have constant norm and expanding the corresponding constraint in Taylor series with respect to η. It can be shown under reasonable assumptions that the weight vector will converge towards the principal eigenvector of the input correlation matrix. Converging learning rules are discussed more broadly in Section 5.

4 Local Learning in the Non-Linear Case

To extend the theory to the non-linear case, we consider a non-linear unit $O = f(S) = f(\sum_0^N w_i I_i)$ where f is a transfer function that is logistic (0,1) or hyperbolic tangent (-1,1) in the differentiable case, or the corresponding threshold functions. All the expectations computed in the linear case that do not involve the variable O can be computed exactly as in the linear case. Furthermore, at least in the case of threshold gates, we can easily deal with powers of O because $O^2 = O$ in the (0,1) case, and $O^2 = 1$ in the (-1,1) case. Thus, in essence, the main challenge is to compute terms of the form $E(O)$ and $E(OI_i)$ when O is non-linear. We next show how these expectations can be approximated.

4.1 Terms in $E(O)$ and the Dropout Approximation

When the transfer function is a sigmoidal logistic function $\sigma = \sigma_{[0,1]}$, we can use the approximation [9]

$$E(O) = E(\sigma(S)) \approx \sigma(E(S)) \quad \text{with} \quad |E(\sigma(S)) - \sigma(E(S))| \approx \frac{V|1 - 2E|}{1 - 2V} \leq 2E(1 - E)|1 - 2E| \tag{35}$$

where $E = E(O) = E(\sigma(S))$ and $V = \mathrm{Var}(O) = \mathrm{Var}(\sigma(S))$. Thus $E(O) \approx \sigma(\sum_i w_i \mu_i)$. During learning, as the w_i vary, this term could fluctuate. Note however that if the data is centered ($\mu_i = 0$ for every i), which is often done in practice, then we can approximate the term $E(O)$ by a constant equal to $\sigma(0)$ across all epochs. Although there are cases where the approximation of Equation 35 is not precise, in most reasonable cases it is quite good.

This approximation has its origin in the dropout approximation $E(O) \approx NWGM(O) = \sigma(E(S))$ where $NWGM$ represents the normalized geometric mean. These and several other related results are proven in [9].

When the transfer function is a hyperbolic tangent function we can use the same approximation

$$E(O) = E(\tanh(S)) \approx \tanh(E(S)) \tag{36}$$

This is simply because

$$\tanh(S) = 2\sigma(2S) - 1 \tag{37}$$

Equation 35 is valid not only for the standard logistic function, but also for any logistic function with slope λ of the form $\sigma(S) = 1/(1 + ce^{-\lambda S})$. Threshold functions are approximated by sigmoidal functions with $\lambda \to +\infty$. Thus the approximation can be used also for threshold functions with $\lambda \to +\infty$, with similar caveats. More generally, if the transfer function f is differentiable and can be expanded as a Taylor series around the mean $E(S)$, we always have: $f(S) \approx f(E(S)) + f'(E(S))(S - E(S)) + \frac{1}{2}f''(E(S))(S - E(S))^2$ and thus $E(f(S)) \approx f(E(S)) + \frac{1}{2}f''(E(S))VarS$. Thus if $VarS$ is small or $f''(E(S))$ is small, then $E(f(S)) \approx f(E(S)) = f(\sum_i w_i\mu_i)$. The approximations can often be used also for other functions (e.g. rectified linear), as discussed in [9].

4.2 Terms in $E(OI_i)$

Next, in the analysis of learning rules in the non-linear case, we must deal with expectations of the form $E(OI_i)$. A first simple approximation is to assume that O and I_i are almost independent and therefore

$$E(OI_i) \approx E(O)E(I_i) = E(O)\mu_i \tag{38}$$

In this expression, $E(O)$ can in turn be approximated using the method above. For instance, in the case of a logistic or $\tanh$ transfer function

$$E(OI_i) \approx E(O)E(I_i) = E(O)\mu_i \approx \mu_i\sigma(E(S)) = \mu_i\sigma(\sum_{i=1}^{N} w_i\mu_i) \tag{39}$$

If the data is centered, the approximation reduces to 0.

A second possible approximation is obtained by expanding the sigmoidal transfer function into a Taylor series. To a first order, this gives

$$E(OI_i) = E\left[\sigma(\sum_j w_jI_j)I_i\right] = E\left[\sigma(\sum_{j\neq i} w_jI_j + w_iI_i)I_i\right] \approx E\left[\sigma(\sum_{j\neq i} w_jI_j)I_i + \sigma'(\sum_{j\neq i} w_jI_j)w_iI_iI_i\right] \tag{40}$$

with the approximation quality of a first-order Taylor approximation to σ. To further estimate this term we need to assume that the terms depending on j but not on i are independent of the terms dependent on i, i.e. that the data covariances are 0. In this case,

$$E(OI_i) \approx E(\sigma(\sum_{j\neq i} w_jI_j))E(I_i) + E(\sigma'(\sum_{j\neq i} w_jI_j)w_iE(I_i^2)) \approx \mu_i\sigma(\sum_{j\neq i} w_j\mu_j) + E(\sigma'(\sum_{j\neq i} w_jI_j))w_i(\mu_i^2 + \sigma_i^2) \tag{41}$$

where $\sigma_i^2 = VarI_i$ and the latter approximation uses again the dropout approximation. If in addition the data is centered ($\mu_i = 0$ for every i) we have

$$E(OI_i) \approx E(\sigma'(\sum_{j\neq i} w_jI_j))w_i\sigma_i^2 \tag{42}$$

which reduces back to a linear term in w

$$E(OI_i) \approx E(\sigma'(0))w_i\sigma_i^2 \tag{43}$$

when the weights are small, the typical case at the beginning of learning.

In summary, when $n \leq 2$ and $r \leq 1$ the dynamics of the learning rule can be solved exactly or approximately, even in the non-linear case, and it is entirely determined by the statistical moments of the data.

4.3 Examples

In this section we consider simple local learning rules applied to a single sigmoidal or threshold unit.

4.3.1 Unsupervised Simple Hebb Rule

We first consider the simple Hebb rule $\Delta w_i = \eta I_i O$. Using the approximations described above we obtain

$$E(\Delta w_i) \approx \eta\mu_i E(O) \approx \eta\mu_i\sigma(\sum_i w_i\mu_i) \quad \text{thus} \quad w_i(k) = w_i(0) + \eta\mu_i \left[\sum_{l=0}^{k-1} \sigma(\sum_j w_j(l)\mu_j)\right] \tag{44}$$

Thus the weight vector tends to align itself with the center of gravity of the data. However, this provides only a direction for the weight vector which continues to grow to infinity along that direction, as demonstrated in Figure 4.1.

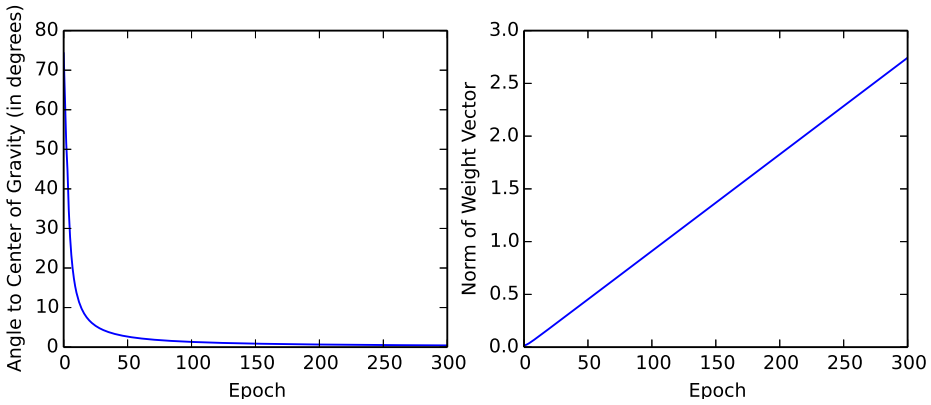

Figure 4.1: Single unit trained on the MNIST data set (60,000 examples) for 500 epochs, with a learning rate of 0.001 using the simple Hebb rule in unsupervised fashion. The fan-in is 784 (28×28). The weights are initialized from a normal distribution with standard deviation 0.01. Left: Angle of the weight vector to the center of gravity. Right: Norm of the weight vector.

4.3.2 Supervised Simple Hebb Rule

Here we consider a single [-1,1] sigmoidal or threshold unit trained using a set of M training input-output pairs $I(t), T(t)$ where $T(t) = \pm 1$. In supervised mode, here the output is clamped to the target value (note in general this is different from the perceptron or backpropagation rule).

Here we apply the simple Hebb rule with the output clamped to the target so that $\Delta w_i = \eta I_i T$. Thus the expectation $E(\Delta w_i) = \eta E(I_i T)$ is constant across all the epochs and depends only on the data moments. In general the weights will grow linearly with the number of epochs, unless $E(\Delta w_i) = 0$ in which case the w_i will remain constant and equal to the initial value $w_i(0)$. In short,

$$w_i(k) = w_i(0) + k\eta E(I_i T) \tag{45}$$

If the targets are essentially independent of the inputs, we have $E(\Delta w_i) \approx \eta E(I_i)E(T) = \eta\mu_i E(T)$, and thus after k learning epochs the weights are given by

$$w_i(k) = w_i(0) + k\eta\mu_i\mu_T \tag{46}$$

In this case we see again that the weight vector tends to be co-linear with the center of gravity of the data, with a sign that depends on the average target, and a norm that grows linearly with the number of epochs.

4.3.3 Gradient Descent Rule

A last example of a convergent rule is provided by $\Delta w_i = \eta(T - O)I_i$ with the logistic transfer function. The rule is convergent (with properly decreasing learning rate η) because it performs gradient descent on the relative entropy error function $E_{err}(w) = -\sum_{t=1}^{M} T(t)\log O(t) + (1 - T(t))\log(1 - O(t))$. Remarkably, up to a trivial scaling factor of two that can be absorbed into the learning rate, this learning rule has exactly the same form when the $\tanh$ function is used over the $[-1, 1]$ range (Appendix B).

5 Derivation of New Learning Rules

The local learning framework is also helpful for discovering new learning rules. In principle, one could recursively enumerate all polynomial learning rules with rational coefficients and search for rules satisfying particular properties. However this is not necessary for several reasons. In practice, we are only interested in polynomial learning rules with relatively small degree (e.g. $n \leq 5$) and more direct approaches are possible. To provide an example, here we consider the issue of convergence and derive new convergent learning rules.

We first note that a major concern with a Hebbian rule, even in the simple case $\Delta w_{ij} \propto O_i O_j$, is that the weights tend to diverge over time towards very large positive or very large negative values. To ensure that the weights remain within a finite range, it is natural to introduce a decay term so that $\Delta w_{ij} \propto O_i O_j - C w_{ij}$ with $C > 0$. The decay coefficient can also be adaptive as long as it remains positive. This is exactly what happens in Oja's cubic learning rule [44]

$$\Delta w_{ij} \propto O_i O_j - O_i^2 w_{ij} \tag{47}$$

which has a weight decay term $O_i^2 w_{ij}$ proportional to the square of the output and is known to extract the principal component of the data. Using different adaptive terms, we immediately get new rules such as:

$$\Delta w_{ij} \propto O_i O_j - O_j^2 w_{ij} \tag{48}$$

and

$$\Delta w_{ij} \propto O_i O_j - (O_i O_j)^2 w_{ij} = O_i O_j (1 - O_i O_j w_{ij}) \tag{49}$$

And when the postsynaptic neuron has a target T_i, we can consider the clamped or gradient descent version of these rules. In the clamped cases, some or all the occurrences of O_i in Equations 47, 48, and 49 are to be replaced by the target T_i. In the gradient descent version, some or all the occurrences of O_i in Equations 47, 48, and 49 are to be replaced by $(T_i - O_i)$. The corresponding list of rules is given in Appendix C.

To derive additional convergent learning rules, we can take yet a different approach by introducing a saturation effect on the weights. To ensure that the weights remain in the $[-1, 1]$ range, we can assume that the weights are calculated by applying a hyperbolic tangent function.

Thus consider a [-1,1] system trained using the simple Hebb rule $\Delta w_{ij} \propto O_i O_j$. To keep the weights in the [-1,1] range throughout learning, we can write:

$$w_{ij}(t+1) = \tanh[w_{ij}(0) + \eta O_i(1)O_j(1) + \ldots \eta O_i(t)O_j(t) + \eta O_i(t+1)O_j(t+1)] \tag{50}$$

where η is the learning rate. By taking a first order Taylor expansion and using the fact that $\tanh(x)' = 1 - \tanh^2(x)$, we obtain the new rule

$$w_{ij}(t+1) = w(t) + \eta(1 - w_{ij}^2)O_i(t)O_j(t) \quad \text{or} \quad \Delta w_{ij} \propto (1 - w_{ij}^2)O_iO_j \tag{51}$$

Note that while simple, this is a quartic learning rule in the local variables with $n = 4$ and $d = 3$. The rule forces $\Delta w_{ij} \to 0$ as $|w_{ij}| \to 1$. In the supervised case, this rule becomes

$$\Delta w_{ij} \propto (1 - w_{ij}^2)T_iO_j \tag{52}$$

in the clamped setting, and

$$\Delta w_{ij} \propto (1 - w_{ij}^2)(T_i - O_i)O_j \tag{53}$$

in the gradient descent setting.

To further analyze the behavior of this rule in the clamped setting, for instance, let us consider a single tanh or threshold [-1,1] unit with $\Delta w_i = \eta(1 - w^2)TI_i$. In the regime where the independence approximation is acceptable, this yields $E(\Delta w_i) = \eta(1 - w^2)E(T)\mu_i$ which is associated with the Riccati differential equation that we already solved in the linear case. One of the solutions (converging to +1) is given by

$$w(k) = \frac{1 - 2Ce^{-2\eta\mu E(t)k}}{1 + 2Ce^{-2\eta\mu E(t)k}} \quad \text{with} \quad C = \frac{1 - w(0)}{2(1 + w(0))} \tag{54}$$

Simulations of these new rules demonstrating how they effectively control the magnitude of the weights and how well the theory fits the empirical data are shown in Figures 5.1, 5.2, 5.3, and 5.4.

Finally, another alternative mechanism for preventing unlimited growth of the weights is to reduce the learning rate as learning progresses, for instance using a linear decay schedule.

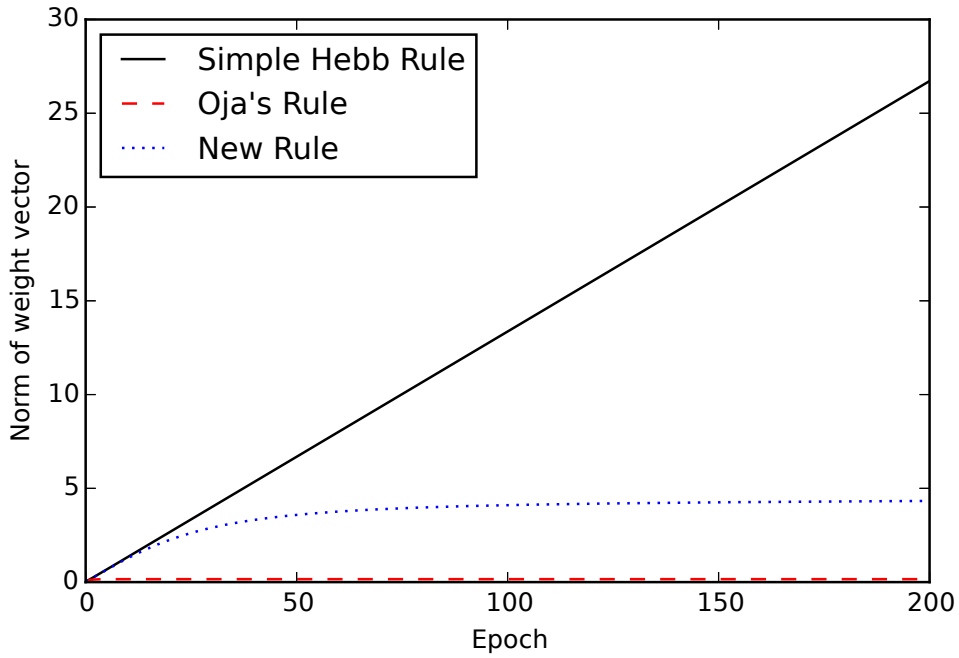

Figure 5.1: Temporal evolution of the norm of the weight vector of a single threshold gate with 20 inputs and a bias trained in supervised mode using 500 randomly generated training examples using three different learning rules: Basic Hebb, Oja, and the New Rule. Oja and the New Rule gracefully prevent the unbounded growth of the weights. The New Rule produces a weight vector whose component are fairly saturated (close to -1 or 1) with a total norm close to $\sqrt{21}$.

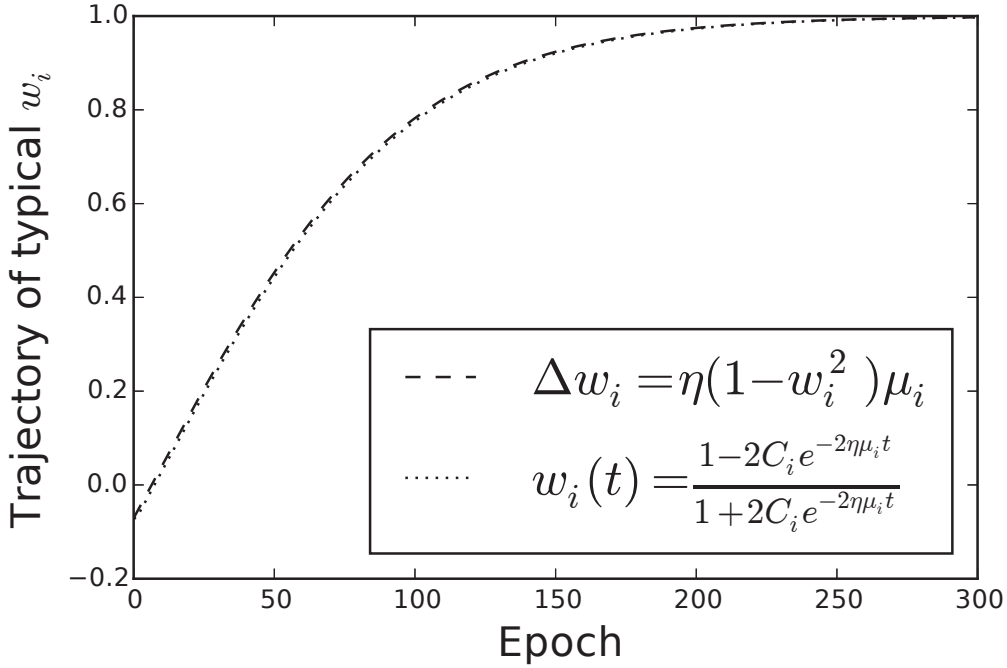

Figure 5.2: A learning rule that results in a Riccati differential equation. The solution to this Riccati equation tells us that all the weights will converge to 1. A typical weight is is shown. It is initialized randomly from $N(0, 0.1)$ and trained on 1000 MNIST resulting in a fan-in of 784 (28×28). There is almost perfect agreement between the theoretical and empirical curve.

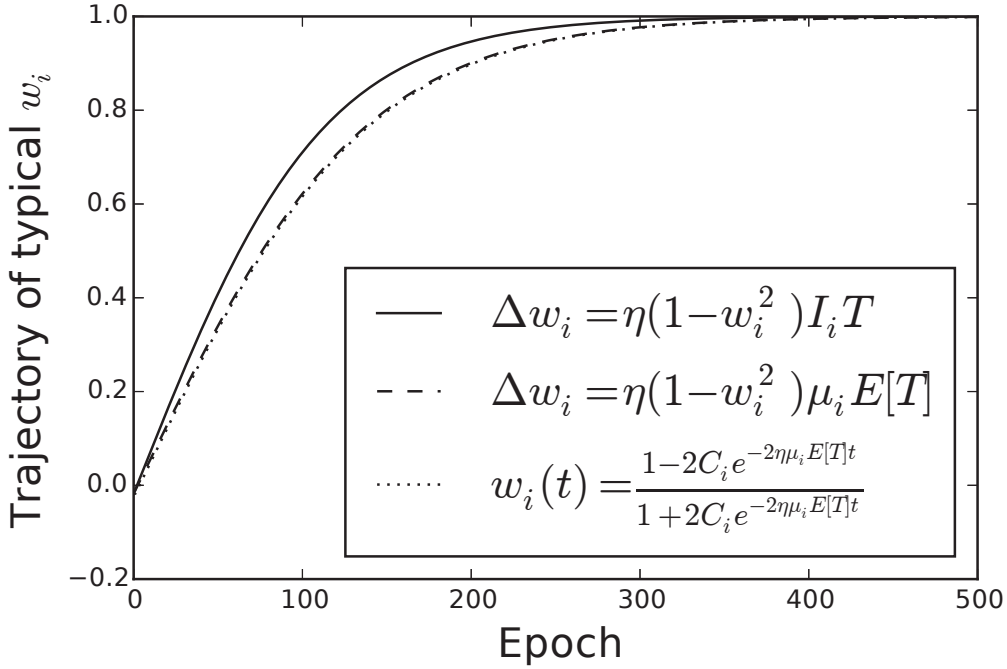

Figure 5.3: When the independence assumption is reasonable, the Riccati equation describes the dynamics of learning and can be used to find the exact solution. The typical weight shown here is randomly initialized from $N(0, 0.1)$ and is trained on $M = 1000$ MNIST samples to recognize digits 0-8 vs 9 classes. $N = 784$.

6 What is Learnable by Shallow or Deep Local Learning

The previous sections have focused on the study of local learning rules, stratified by their degree, in shallow networks. In this section, we begin to look at local learning rules applied to deep feedforward networks and partially address the

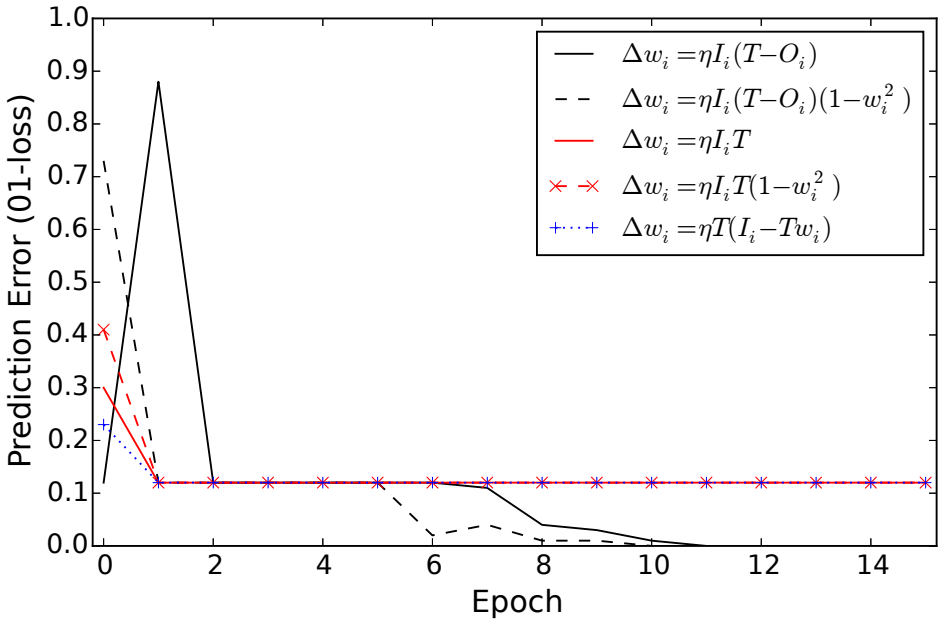

Figure 5.4: A single neuron with tanh activation trained to recognize the handwritten digit nine with five supervised learning rules. The input data is 100 MNIST images (made binary by setting pixels to +1 if the greyscale value surpassed a threshold of 0.2, and -1 otherwise), and binary -1,+1 targets. Weights were initialized independently from N(0, 0.1), and updated with learning rate $\eta = 0.1$.

question of what is locally learnable in feedforward networks.

Specifically we want to consider shallow (single adaptive layer) local learning, or *deep local learning* defined as learning in deep feedforward layered architectures with learning rules of the form $\Delta w_{ij} = F(O_i, O_j, w_{ij})$ applied successively to all the layers, starting from the input layer, possibly followed by supervised learning rule of the form $\Delta w_{ij} = F(O_i, O_j, T_i, w_{ij})$ in the top layer alone, when targets are available for the top layer (Figure 6.2). One would like to understand what input-output functions can be learnt from examples using this strategy and whether this provides a viable alternative to back-propagation. We begin with a few simulation experiments to further motivate the analyses.

6.1 Simulation Experiments: Learning Boolean Functions

We conduct experiments using various local learning rules to try to learn Boolean functions with *small* fan in architectures with one or two adaptive layers. These experiments are purposely carried to show how simulations run in simple cases can raise false hopes of learnability by local rules that do not extend to large fan in and more complex functions, as shown in a later section. Specifically, we train binary [-1,1] threshold gates to learn Boolean functions of up to 4 inputs, using the simple Hebb rule, the Oja rule, and the new rule corresponding to Equation 51 and its supervised version. Sometimes multiple random initializations of the weights are tried and the function is considered to be learnable if it is learnt in at least one case. [Note: The number of trials needed is not important since the ultimate goal is to show that this local learning strategy cannot work for more complex functions, even when a very large number of trials is used.] In Tables 6 and 7, we report the results obtained both in the shallow case (single adaptive layer) trained in a supervised manner, and the results obtained in the deep case (two adaptive layers) where the adaptive input layer is trained in unsupervised manner and the adaptive output layer is trained in supervised manner. In the experiments, all inputs and targets are binary (-1,1), all the units have a bias, and the learning rate decays linearly.

As shown in Table 6, 14 of the 16 possible Boolean functions of two variables ($N = 2$) can be learnt using the Simple Hebb, Oja, and new rules. The two Boolean functions that cannot be learnt are of course XOR and its converse which cannot be implemented by a single layer network. Using deep local learning in two-layer networks, then all three rules are able to learn all the Boolean functions with $N = 2$, demonstrating that at least some complex functions can be learnt by combining unsupervised learning in the lower layer with supervised learning in the top layer. Similar results are also seen for $N = 3$, where 104 Boolean functions, out of a total of 256, are learnable in a shallow network. And all 256 functions are learnable by a two-layer network by any of the three learning rules.

Table 7 shows similar results on the subset of *monotone* Boolean functions. As a reminder, a Boolean function

is said to be monotone if increasing the total number of +1 in the input vector can only leave the value of the output unchanged or increase its value from -1 to +1. Equivalently, it is the set of Boolean functions with a circuit comprising only AND and OR gates. There are recursive methods for generating monotone Boolean functions and the total number of monotone Boolean functions is known as the Dedekind number. For instance, there are 168 monotone Boolean functions with $N = 4$ inputs. Of these, 150 are learnable by a single unit trained in supervised fashion, and all 168 are learnable by a two-layer network trained with a combination of unsupervised (input layer) and supervised (output layer) application of the three local rules.

Fan In	**Functions Learnt**		**Total Number of Functions**	**Rule**
	Shallow	**Deep**		
2	14	16	16	Simple Hebb
2	14	16	16	Oja
2	14	16	16	New
3	104	256	256	Simple Hebb
3	104	256	256	Oja
3	104	256	256	New

Table 6: Small fan-on Boolean functions learnt by deep local learning.

Fan In	**Functions Learnt**		**Total Number of Functions**	**Rule**
	Shallow	**Deep**		
2	6	6	6	Simple Hebb
2	6	6	6	Oja
2	6	6	6	New
3	20	20	20	Simple Hebb
3	20	20	20	Oja
3	20	20	20	New
4	150	168	168	Simple Hebb
4	150	168	168	Oja
4	150	168	168	New

Table 7: Small fan-in monotone Boolean functions learnt by deep local learning.

In combination, these simulations raise the question of what are the classes of functions learnable by shallow or deep local learning, and raise the (false) hope that purely local learning may be able to replace backpropagation.

6.2 Learnability in Shallow Networks

Here we consider in more detail the learning problem for a single [-1,1] threshold gate, or perceptron.

6.2.1 Perceptron Rule

In this setting, the problem has already been solved at least in one setting by the perceptron learning algorithm and theorem [49, 42]. Obviously, by definition, a threshold gate can only implement in an exact way functions (Boolean or continuous) that are *linearly separable*. The perceptron learning algorithm simply states that if the data is linearly

separable, the local gradient descent learning rule $\Delta w_i = \eta(T - O)I_i$ will converge to such a separating hyperplane. Note that this is true also in the case of [0,1] gates as the gradient descent rule as the same form in both systems. When the training data is not linearly separable, the perceptron algorithm is still well behaved in the sense that algorithm converges to a relatively small compact region [42, 14, 24]. Here we consider similar results for a slightly different supervised rule, the clamped form of the simple Hebb rule: $\Delta w_i = \eta T I_i$.

6.2.2 Supervised Simple Hebb Rule

Here we consider a supervised training set consisting of input-target pairs of the form $\mathcal{S} = \{(I(t), T(t)) : t = 1, \ldots, M\}$ where the input vectors $I(t)$ are N-dimensional (not-necessarily binary) vectors with corresponding targets $T(t) = \pm 1$ for every t (Figure 6.1). $\mathcal{S}$ is linearly separable (with or without bias) if there is a separating hyperplane, i.e. set of weights w such that $\tau(I(t)) = \tau(\sum w_i I_i(t)) = T(t)$ (with or without bias) for every t, where τ is the ± 1 threshold function. To slightly simplify the notation and analysis, throughout this section, we do not allow ambiguous cases where $\tau(I) = 0$ for any I of interest. In this framework, the linearly separable set $\mathcal{S}$ is learnable by a given learning rule R (R-learnable) if the rule can find a separating hyperplane.

The Case Without Bias: When there is no bias ($w_0 = 0$), then $\tau(-I) = -\tau(I)$ for every I. In this case, a set $\mathcal{S}$ is *consistent* if for every t_1 and t_2: $I(t_1) = -I(t_2) \implies T(t_1) = -T(t_2)$. Obviously consistency is a necessary condition for separability and learnability in the case of 0 bias. When the bias is 0, the training set S can be put into its *canonical* form $\mathcal{S}^c$ by ensuring that all targets are set to $+1$, replacing any training pair of the form $(I(t), -1)$ by the equivalent pair $(T(t)I(t), +1) = (-I(t), +1)$. Thus the size of a learnable canonical training set in the binary case, where $I_i(t) = \pm 1$ for every i and t, is at most 2^{N-1}.

We now consider whether $\mathcal{S}$ is learnable by the supervised simple Hebb rule (SSH-learnable) corresponding to clamped outputs $\Delta w_i = \eta I_i T$, first in the case where there is no bias, i.e. $w_0 = 0$. We let Cos denote the $M \times M$ symmetric square matrix of cosine values $Cos = (Cos_{uv}) = (\cos(T(u)I(u), T(v)I(v))) = (\cos(I^c(u), I^c(v)))$. It is easy to see that applying the supervised simple Hebb rule with the vectors in $\mathcal{S}$ is equivalent to applying the supervised simple Hebb rule with the vectors in $\mathcal{S}^c$, both leading to the same weights. If $\mathcal{S}$ is in canonical form and there is no bias, we have the following properties.

Theorem:

1. The supervised simple Hebb rule leads to $\Delta w_i = \eta E(I_i T) = \eta E(I_i^c) = \eta \mu_i^c$ and thus $w(k) = w(0) + \eta k \mu^c$.
2. A necessary condition for $\mathcal{S}$ to be SSH-learnable is that $\mathcal{S}$ (and equivalently $\mathcal{S}^c$) be linearly separable by a hyperplane going through the origin.
3. A sufficient condition for $\mathcal{S}$ to be SSH-learnable from any set of starting weights is that all the vectors in $\mathcal{S}^c$ be in a common orthant, i.e. that the angle between any $I^c(u)$ and $I^c(c)$ lie between 0 and $\pi/2$ or, equivalently, that $0 \leq \cos(I^c(u), I^c(v)) \leq 1$ for any u and v.
4. A sufficient condition for $\mathcal{S}$ to be SSH-learnable from any set of starting weights is that all the vectors in $\mathcal{S}$(or equivalently in $\mathcal{S}^c$) be orthogonal to each other, i.e. $I(u)I(v) = 0$ for any $u \neq v$.
5. If all the vectors $I(t)$ have the same length, in particular in the binary ± 1 case, S is SSH-learnable from any set of initial weights if and only if the sum of any row or column of the cosine matrix associated with $\mathcal{S}^c$ is strictly positive.

Proof:
1) Since $\mathcal{S}^c$ is in canonical form, all the targets are equal to +1, and thus $E(I_i(t)T(t)) = E(I_i^c(t)) = \mu_i^c$. After k learning epochs, with a constant learning rate, the weight vector is given by $w(k) = w(0) + \eta k \mu^c$.
2) This is obvious since the unit is a threshold gate.
3) For any u, the vector $I^(u)$ has been learnt after k epochs if and only if

$$\sum_{i=1}^{N} [w_i(0) + \eta k E(I_i(t)T(t))] I_i(u) = \sum_{i=1}^{N} \left(w_i(0) I_i^c(u) + \eta k \frac{1}{M} \sum_{t=1}^{M} I_i^c(t) I_i^c(u) \right) > 0 \tag{55}$$

Here we assume a constant positive learning rate, so after a sufficient number of epochs the effect of the initial conditions on this inequality can be ignored. Alternatively one can examine the regime of decreasing learning rates

using initial conditions close to 0. Thus ignoring the transient effect caused by the initial conditions, and separating the terms corresponding to u, $I(u)$ will be learnt after a sufficient number of epochs if and only if

$$\sum_{i=1}^{N}\sum_{t=1}^{M} I_i^c(t)I_i^c(u) = \sum_{t=1}^{M} I^c(t)I^c(u) = \sum_{t=1}^{M} ||I^c(t)||||I^c(u)|| \cos(I^c(t), I^c(u))$$
$$= |I^c(u)|^2 + \sum_{t\neq u} ||I^c(t)||||I^c(u)|| \cos(I^c(t), I^c(u)) > 0 \quad (56)$$

Thus if all the cosines are between 0 and 1 this sum is strictly positive (note that we do not allow $I(u) = 0$ in the training set). Since the training set is finite, we simply take the maximum number of epochs over all training examples where this inequality is satisfied, to offset the initial conditions. Note that the expression in Equation 56 is invariant with respect to any transformation that preserves vector lengths and angles, or changes the sign of all or some of the angles. Thus it is invariant with respect to any rotations, or symmetries.
4) This is a special case of 3, also obvious from Equation 56. Note in particular that a set of αN ($0 < \alpha \leq 1$) vectors chosen randomly (e.g. uniformly over the sphere or with fair coin flips) will be essentially orthogonal and thus learnable with high probability when N is large.
5) If all the training vectors have the same length A (with $A = \sqrt{N}$ in the binary case), Equation 56 simply becomes

$$A^2 \sum_{t=1}^{M} \cos(I^c(u), I^c(t)) > 0 \quad (57)$$

and the property is then obvious. Note that it is easy to construct counterexamples where this property is not true if the training vectors do not have the same length. Take, for instance, $\mathcal{S} = \{(I(1), +1), (I(2), +1)\}$ with $I(1) = (1, 0, 0, \ldots, 0)$ and $I(2) = (-\epsilon, 0, 0 \ldots, 0)$ for some small $\epsilon > 0$.

The Case With Adaptive Bias: When there is a bias (w_0 is not necessarily 0), starting from the training set $\mathcal{S} = \{(I(t), T(t))\}$ we first modify each vector $I(t)$ into a vector $I'(t)$ by adding a zero-th component equal to +1, so that $I'_0(t) = +1$, and $I'_i(t) = I_i(t)$ otherwise. Finally, we construct the corresponding canonical set $\mathcal{S}^c$ as in the case of 0 bias by letting $\mathcal{S}^c = \{(I^c(t), +1)\} = \{(T(t)I'(t), +1)\}$ and apply the previous results to $\mathcal{S}^c$. It is easy to check that applying the supervised simple Hebb rule with the vectors in $\mathcal{S}$ is equivalent to applying the supervised simple Hebb rule with the vectors in $\mathcal{S}^c$, both leading to the same weights.

Theorem:

1. The supervised simple Hebb rule applied to $\mathcal{S}^c$ leads to $\Delta w_i = \eta E(I'_i T) = \eta E(I_i^c) = \eta \mu_i^c$ and thus $w(k) = w(0) + \eta k \mu^c$. The component μ_0^c is equal to the proportion of vectors in $\mathcal{S}$ with a target equal to +1.

2. A necessary condition for $\mathcal{S}$ to be SSH-learnable is that $\mathcal{S}^c$ be linearly separable by a hyperplane going through the origin in $N + 1$ dimensional space.

3. A sufficient condition for $\mathcal{S}$ to be SSH-learnable from any set of starting weights is that all the vectors in $\mathcal{S}^c$ be in a common orthant, i.e. that the angle between any $I^c(u)$ and $I^c(v)$ lie between 0 and $\pi/2$ or, equivalently, that $0 \leq \cos(I^c(u), I^c(v)) \leq 1$ for any u and v.

4. A sufficient condition for $\mathcal{S}$ to be SSH-learnable from any set of starting weights is that all the vectors in $\mathcal{S}^c$ be orthogonal to each other, i.e. $I^c(u)I^c(v) = 0$ for any $u \neq v$.

5. If all the vectors $I^c(t)$ have the same length, in particular in the binary ± 1 case, S is SSH-learnable from any set of initial weights if and only if the sum of any row or column of the cosine matrix $Cos = (\cos(I^c(u), I^c(v)))$ is strictly positive.

Proof:
The proofs are the same as above. Note that $\mu_0^c = E(I_0^c(t)) = E(I'_0(t)T(t)) = E(T(t))$.

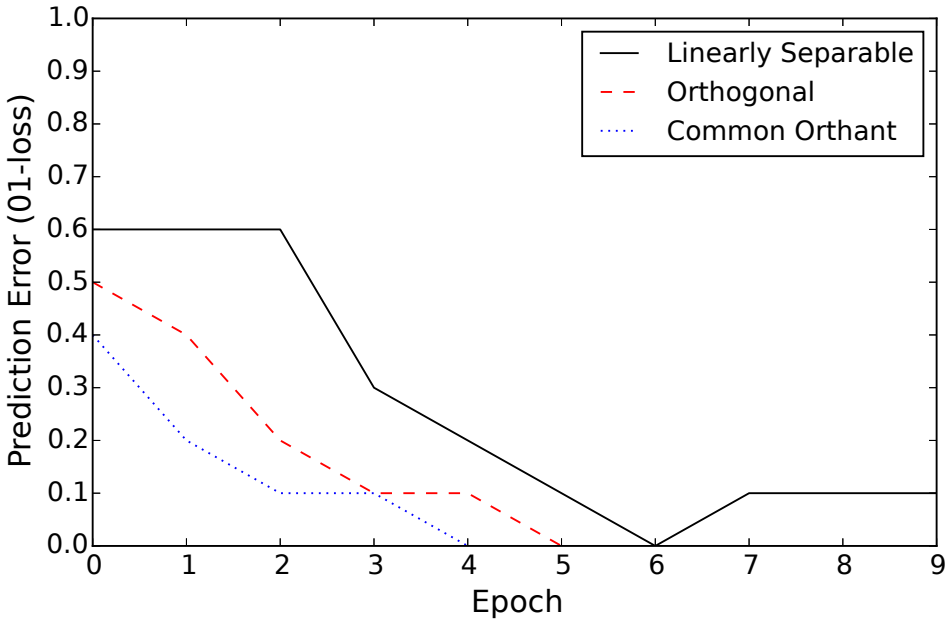

Figure 6.1: Examples of supervised simple Hebb learning with different training set properties. The linearly separable data is a random matrix of binary -1,+1 values (p=0.5) of shape M=10, N=10, with binary -1,1 targets determined by a random hyperplane. The orthogonal dataset is simply the identity matrix (multiplied by the scalar $\sqrt{10}$) and random binary -1,+1 targets. The common orthant dataset was created by sampling the features in each column from either $[-1, 0)$ or $(0, 1]$, then setting all the targets to +1. The weights were initialized independently from N(0,1), and weights were updated with the learning rate $\eta = 0.1$.

6.3 Limitations of Shallow Local Learning

In summary, strictly local learning in a single threshold gate or sigmoidal function can learn any linearly separable function. While it is as powerful as the unit allows it to be, this form of learning is limited in the sense that it can learn only a very small fraction of all possible functions. This is because the logarithm of the size of the set of all possible Boolean functions of N variables is exponential and equal to 2^N, whereas the logarithm of the size of the total number of linearly separable Boolean functions scales polynomially like N^2. Indeed, the total number T_N of threshold functions of N variables satisfies

$$N(N-1)/2 \leq \log_2 T_N \leq N^2 \tag{58}$$

(see [61, 18, 43, 5] and references therein). The same negative result holds also for the more restricted class of monotone Boolean functions, or any other class of exponential size. Most monotone Boolean functions cannot be learnt by a single linear threshold unit because the number M_N of monotone Boolean functions of N variables, known as the Dedekind number, satisfies [33]

$$\binom{N}{\lfloor N/2 \rfloor} \leq \log_2 M_N \leq \binom{N}{\lfloor N/2 \rfloor} (1 + O(\log N/N)) \tag{59}$$

These results are immediately true also for polynomial threshold functions, where the polynomials have bounded degree, by similar counting arguments [5]. *In short, linear or bounded-polynomial threshold functions can at best learn a vanishingly small fraction of all Boolean functions, or any subclass of exponential size, regardless of the learning rule used for learning.*

The fact that local learning in shallow networks has significant limitations seems to be a consequence of the limitations of shallow networks, which are simply not able to implement complex function. This alone, does not preclude the possibility that iterated shallow learning applied to deep architectures, i.e. deep local learning, may be able to learn complex functions. After all this would be consistent with what is observed in the simple simulations described above where the XOR function, which is not learnable by a shallow networks, becomes learnable by local rules in a network of depth two. Thus over the years many attempts have been made to seek efficient, and perhaps

more biologically plausible, alternatives to backpropagation for learning complex data using *only local rules*. For example, in one of the simplest cases, one could try to learn a simple two-layer autoencoder using unsupervised local learning in the first layer and supervised local learning in the top layer. More broadly, one could for example try to learn the MNIST benchmark [38] data using purely local learning. Simulations show (data not shown) however that such schemes fail regardless of which local learning rules are used, how the learning rates and other hyperparameters are tuned, and so forth. In the next section we show why all the attempts that have been made in this direction are bound to fail.

6.4 Limitations of Deep Local Learning

Consider now deep local learning in a deep layered feedforward architecture (Figure 6.2) with $L+1$ layers of size $N_0, N_1, \ldots N_L$ where layer 0 is the input layer, and layer L is the output layer. We let O_i^h denote the activity of unit i in layer h with $O_i^h = f(S_i^h) = f(\sum_j w_{ij}^h O_j^{h-1})$. The non-linear processing units can be fairly arbitrary. For this section, it will be sufficient to assume that the functions f be differentiable functions of their synaptic weights and inputs. It is also possible to extend the analysis to, for instance, threshold gates by taking the limit of very steep differentiable sigmoidal functions. We consider the supervised learning framework with a training set of input-output vector pairs of the form $(I(t), T(t))$ for $t = 1, \ldots, M$ and the goal is to minimize a differentiable error function E_{err}. The main learning constraint is that we can only use deep local learning (Figure 6.2).

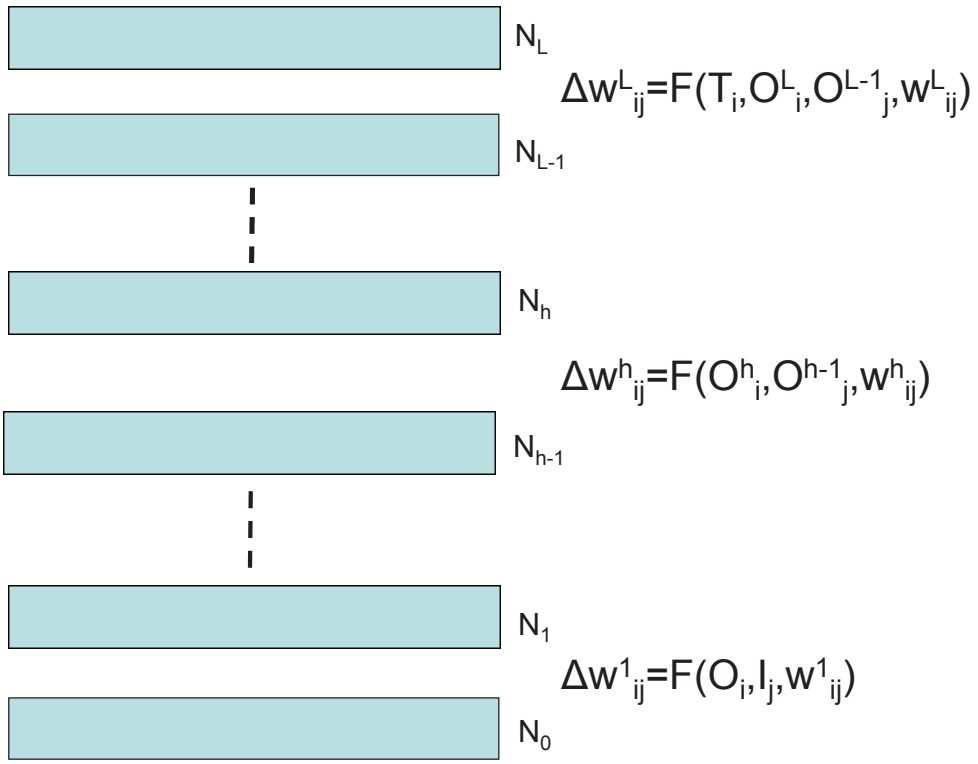

Figure 6.2: Deep local learning. Local learning rules are used for each unit. For all the hidden units, the local learning rules are unsupervised and thus of the form $\Delta w_{ij}^h = F(O_i^h, O_j^{h-1}, w_{ij}^h)$. For all the output units, the local learning rules can be supervised since the targets are considered as local variables and thus of the form $\Delta w_{ij}^L = F(T, O_i^L, O_j^{L-1}, w_{ij}^L)$.

Fact: Consider the supervised learning problem in a deep feedforward architecture with differentiable error function and transfer functions. Then in most cases deep local learning cannot find weights associated with critical points of the error functions, and thus it cannot find locally or globally optimal weights.

Proof: If we consider any weight w_{ij}^h in a deep layer h (i.e. $0 < h < l$), a simple application of the chain rule (or the

backpropagation equations) shows that

$$\frac{\partial E_{err}}{\partial w_{ij}^h} = E\left[B_i^h(t)O_j^{h-1}(t)\right] = \frac{1}{M}\sum_{t=1}^{M} B_i^h(t)O_j^{h-1}(t) \tag{60}$$

where $B_i^h(t)$ is the backpropagated error of unit i in layer h, which depends in particular on the targets $T(t)$ and the weights in the layers above layer h. Likewise, $O_j^{h-1}(t)$ is the presynaptic activity of unit j in layer $h-1$ which depends on the inputs $I(t)$ and the weights in the layers below layer $h-1$. *In short, the gradient is a sum over all training examples of product terms, each product term being the product of a target-dependent term with an input-dependent term.* [*The target-dependent term depends explicitly also on all the descendant weights of unit i in layer h, and the input-dependent term depends also on all the ancestors weights of unit j in layer $h-1$.*] As a result, in most cases, the deep weights w_{ij}^h, which correspond to a critical point where $\partial E_{err}/\partial w_{ij}^h = 0$, must depend *on both the inputs and the targets*, as well as all the other weights. In particular, this must be true at any local or global optimum. However, using any strictly local learning scheme all the deep weights w_{ij}^h ($h < L$) depend on the *inputs only*, and thus cannot correspond to a critical point.

In particular, this shows that applying local Hebbian learning to a feedforward architecture, whether a simple autoencoder architecture or Fukushima's complex neocognitron architecture, cannot achieve optimal weights, regardless of which kind of local Hebbian rule is being used. For the same reasons, an architecture consisting of a stack of autoencoders trained using unlabeled data only [27, 28, 12, 11, 20] cannot be optimal in general, even when the top layer is trained by gradient descent. It is of course possible to use local learning, shallow or deep autoencoders, Restricted Boltzmann Machines, and so forth to compress data, or to it initialize the weights of a deep architecture. However, these steps alone cannot learn complex functions optimally because learning a complex function optimally necessitates the reverse propagation of information from the targets back to the deep layers.

The Fact above is correct at a level that would satisfy a physicist and is consistent with empirical evidence. It is not completely tight from a mathematical standpoint due to the phrase "in most cases". This expression is meant to exclude trivial cases that are not important in practice, but which would be difficult to capture exhaustively with mathematical precision. These include the case when the training data is trivial with respect to the architecture (e.g. $M = 1$) and can be loaded entirely in the weights of the top layer, even with random weights in the lower layers, or when the data is generated precisely with an artificially constructed architecture where the deep weights depend only on the input data, or are selected at random.

This simple result has significant consequences. In particular, if a constrained feedforward architecture is to be trained on a complex task in some optimal way, the deep weights of the architecture must depend on both the training inputs and the target outputs. Thus in any physical implementation, in order to be able to reach a locally optimal architecture there *must exist a physical learning channel that conveys information about the targets back to the deep weights*. This raises three sets of questions regarding: (1) the nature of the backward learning channel; and (2) the nature of the information being transmitted through this channel; and (3) the rate of the backward learning channel. These questions will be addressed in Section 8. We now focus on the information about the targets that is being transmitted to the deep layers.

7 Local Deep Learning and Deep Targets Algorithms

7.1 Definitions and their Equivalence

We have seen in the previous section that in general in an optimal implementation each weight w_{ij}^h must depend on both the inputs I and the targets T. In order for learning to remain local, we let $I_{ij}^h(T)$ denote the information about the targets that is transmitted from the output layer to the weight w_{ij}^h for its update by a corresponding local learning rule of the form

$$\Delta w_{ij}^h = F(I_{ij}^h, O_i^h, O_j^{h-1}, w_{ij}^h) \tag{61}$$

[The upper and lower indexes on I distinguish it clearly from the inputs in the 0-th layer. We call this local deep learning (Figure 7.1) in contrast with deep local learning. The main point of the previous section was to show that local deep learning is more powerful than deep local learning, and local deep learning is necessary for reaching optimal weights.

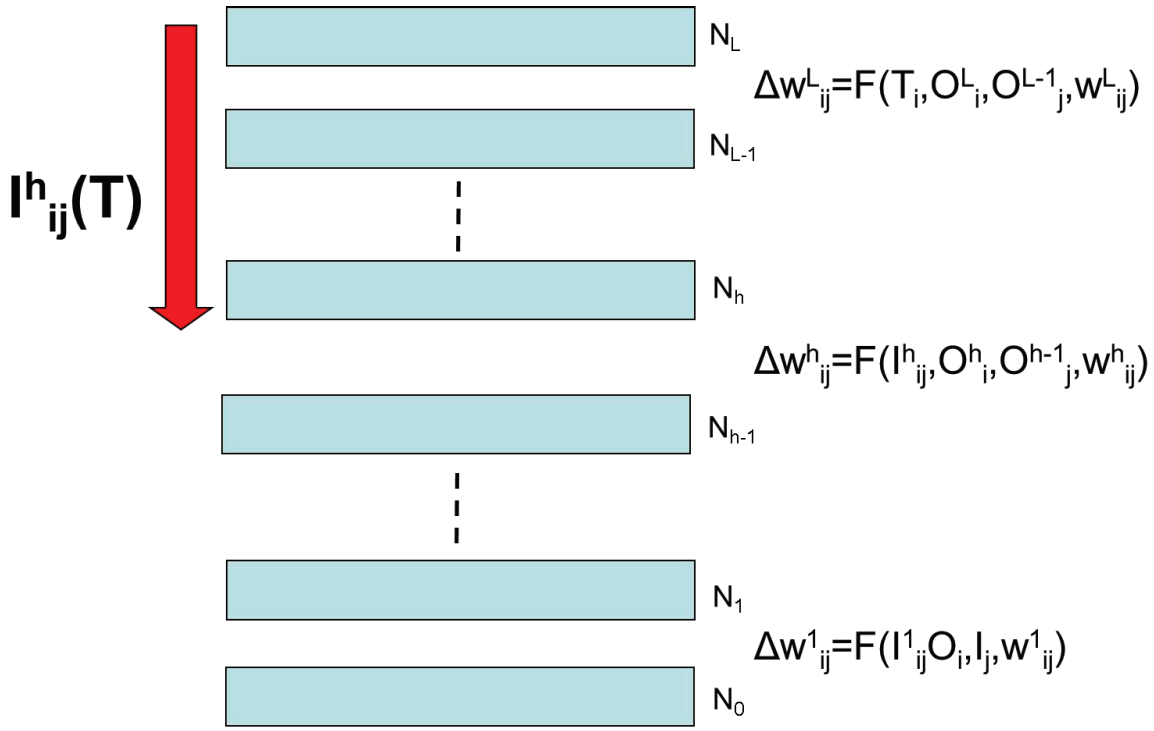

Figure 7.1: Local deep learning. In general, deep local learning cannot learn complex functions optimally since it leads to architectures where only the weights in the top layer depend on the targets. For optimal learning, some information $I^h_{ij}(T)$ about the targets must be transmitted to each synapse associated with any deep layer h, so that it becomes a local variable that can be incorporated into the corresponding local learning rule $\Delta w^h_{ij} = F(I^h_{ij}, O^h_i, O^{h-1}_j, w^h_{ij})$.

Definition 1: Within the class of local deep learning algorithms, we define the subclass of deep targets local learning algorithms as those for which the information I^h_{ij} transmitted about the targets depends only on the postsynaptic unit, in other words $I^h_{ij}(T) = I^h_i(T)$. Thus in a deep targets learning algorithm we have

$$\Delta w^h_{ij} = F(I^h_i, O^h_i, O^{h-1}_j, w^h_{ij}) \tag{62}$$

for some function F (Figure 7.2) .

We have also seen that when proper targets are available, there are efficient local learning rules for adapting the weights of a unit. In particular, the rule $\Delta w = \eta(T - O)I$ works well in practice for both sigmoidal and threshold transfer functions. Thus the deep learning problem can in principle be solved by providing good targets for the deep layers. We can introduce a second definition of deep targets algorithms:

Definition 2: A learning algorithm is a deep targets learning algorithm if it provides targets for all the trainable units.

Theorem: Definition 1 is equivalent to Definition 2. Furthermore, backpropagation can be viewed as a deep targets algorithm.

Proof: Starting from Definition 2, if some target T^h_i is available for unit i in layer h, then we can set $I^h_i = T^h_i$ in Definition 1. Conversely, starting from Definition 1, consider a deep targets algorithm of the form

$$\Delta w^h_{ij} = F(I^h_i, O^h_i, O^{h-1}_j, w^h_{ij}) \tag{63}$$

If we had a corresponding target T^h_i for this unit, it would be able to learn by gradient descent in the form

$$\Delta w^h_{ij} = \eta(T^h_i - O^h_j)O^{h-1}_j \tag{64}$$

This is true both for sigmoidal transfer functions and for threshold gates, otherwise the rule should be slightly modified to accommodate other transfer functions accordingly. By combining Equations 63 and 64, we can solve for the target

$$T_i^h = \frac{F(I_i^h, O_i^h, O_j^{h-1}, w_{ij}^h)}{\eta O_j^{h-1}} + O_i^h \tag{65}$$

assuming the presynaptic activity $O_j^{h-1} \neq 0$ (note that $T^L = T$) (Figure 7.2). In particular, we see that backpropagation can be viewed as a deep targets algorithm providing targets for the hidden layers according to Equation 65 in the form:

$$T_i^h = I_i^h + O_i^h \tag{66}$$

where $I_i^h = B_i^h = \partial E_{err}/\partial S_I^h$ is exactly the backpropagated error.

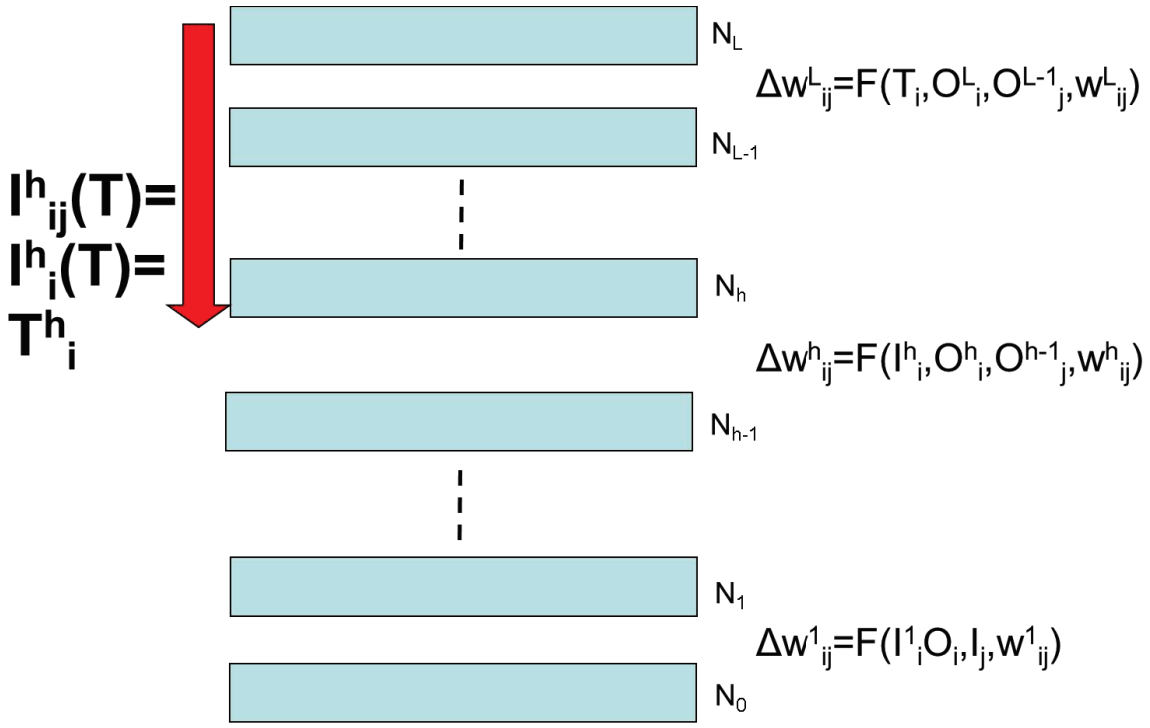

Figure 7.2: Deep targets learning. This is a special case of local deep learning, where the transmitted information $I_{ij}^h(T)$ about the targets does not depend on the presynaptic unit $I_{ij}^h(T) = I_i^h(T)$. It can be shown that this is equivalent to transmitting a deep target T_i^h for training any unit i in any deep layer h by a local supervised rule of the form $\Delta w_{ij}^h = F(T_i^h, O_i^h, O_j^{h-1}, w_{ij}^h)$. In typical cases (linear, threshold, or sigmoidal units), this rule is $\Delta w_{ij}^h = \eta(T_i^h - O_i^h)O_j^{h-1}$.

7.2 Deep Targets Algorithms: the Sampling Approach

In the search for alternative to backpropagation, one can thus investigate whether there exists alternative deep targets algorithms [8]. More generally, deep targets algorithms rely on two key assumptions: (1) the availability of an algorithm Θ for optimizing any layer or unit, while holding the rest of the architecture fixed, once a target is provided; and (2) the availability of an algorithm for providing deep targets. The maximization by Θ may be complete or partial, this optimization taking place with respect to an error measure Δ^h that can be specific to layer h in the architecture (or even specific to a subset of units in the case of an architecture where different units are found in the same layer). For instance, an exact optimization algorithm Θ is obvious in the unconstrained Boolean case [7, 6]. For a layer of threshold gates, Θ can be the perceptron algorithm, which is exact in the linearly separable case. For a layer of artificial neurons with differentiable transfer functions, Θ can be the delta rule or gradient descent, which in general perform only partial optimization. Thus deep targets algorithms proceed according to two loops: an outer loop and an inner loop. The inner loop is used to find suitable targets. The outer loop uses these targets to optimize the weights, as it cycles through the units and the layers of the architecture.

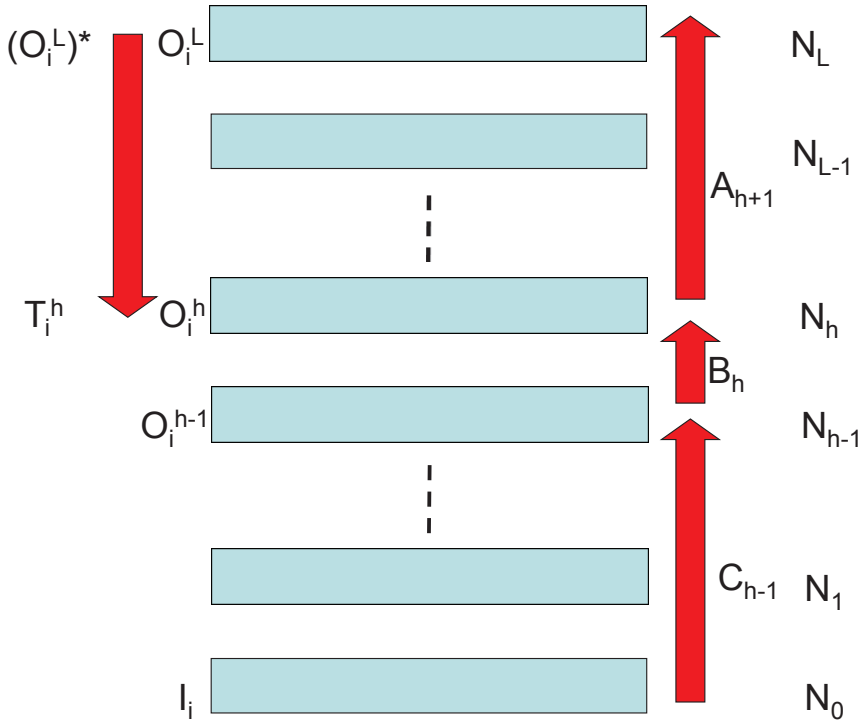

Figure 7.3: Deep architecture and deep targets algorithm. The algorithm visits the various layers according to some schedule and optimizes each one of them. This is achieved by the deep targets algorithm which is capable of providing suitable targets (T_i^h) for any layer h and any input I, assuming that the rest of the architecture is fixed. The targets are used to modify the weights associated with the function B_h. The targets can be found using a sampling strategy: sampling the activities in layer h, propagating them forward to the output layer, selecting the best output sample, and backtracking it to the best sample in layer h.

7.2.1 Outer Loop

The outerloop is used to cycle through, and progressively modify, the weights of a deep feedforward architecture:

1. Cycle through the layers and possibly the individual units in each layer according to some schedule. Examples of relevant schedules include successively sweeping through the architecture layer by layer from the first layer to the top layer.

2. During the cycling process, for a given layer or unit, identify suitable targets, while holding the rest of the architecture fixed.

3. Use the algorithm Θ to optimize the corresponding weights.

Step 2 is addressed by the following inner loop.

7.2.2 Inner Loop: the Sampling Approach

The key question of course is whether one can find ways for identifying deep targets T_i^h, other than backpropagation, which is available only in differentiable networks. It is possible to identify targets by using a sampling strategy in both differentiable and non-differentiable networks.

In the online layered version, consider an input vector $I = I(t)$ and its target $T = T(t)$ and any adaptive layer h, with $1 \leq h \leq L$. We can write the overall input-output function W as $W = A_{h+1}B_hC_{h-1}$ (Figure 7.3). We assume that both A_{h+1} and C_{h-1} are fixed. The input I produces an activation vector O_i^{h-1} and our goal is to find a suitable vector target T^h for layer h. For this we generate a sample $\mathcal{S}^h$ of activity vectors S^h in layer h. This sampling can be carried in different ways, for instance: (1) by sampling the values O^h over the training set; (2) by small random perturbations, i.e. using random vectors sampled in the proximity of the vector $O^h = B_hC_{h-1}(I)$;(3) by large random perturbation (e.g. in the case of logistic transfer functions by tossing dies with probabilities equal to the activations) or by sampling uniformly; and (4) exhaustively (e.g. in the case of a short binary layer). Finally, each sample S^h can

be propagated forward to the output layer and produce a corresponding output $A_{h+1}(S^h)$. We then select as the target vector T^h the sample that produces the output closest to the true target T. Thus

$$T^h = \arg \min_{S^h \in \mathcal{S}^h} \Delta^L(T, A_{h+1}(S^h)) \tag{67}$$

If there are several optimal vectors in $\mathcal{S}^h j$, then one can select one of them at random, or use Δ^h to control the size of the learning step. For instance, by selecting a vector S^h that not only minimizes the output error Δ^L but also minimizes the error $\Delta^h(S^h, O^h)$, one can ensure that the target vector is as close as possible to the current layer activity, and hence minimizes the corresponding perturbation. As with other learning and optimization algorithms, these algorithmic details can be varied during training, for instance by progressively reducing the size of the learning steps as learning progresses (see Appendix D for additional remarks on deep targets algorithms). Note that the algorithm described above is the natural generalization of the algorithm introduced in [7] for the unrestricted Boolean autoencoder, specifically for the optimization of the lower layer. Related reinforcement learning [52] algorithms for connectionist networks of stochastic units can be found in [56].]

7.3 Simulation

Here we present the result of a simulation to show that sampling deep target algorithms can work and can even be applied to the case of non-differentiable networks where back propagation cannot be applied directly. A different application of the deep targets idea is developed in [19]. We use a four-adjustable-layer perceptron autoencoder with threshold gate units and Hamming distance error in all the layers. The input and output layers have $N_0 = N_4 = 100$ units each, and there are three hidden layers with $N_1 = 30$, $N_2 = 10$, and $N_3 = 30$ units. All units in any layer h are fully connected to the N_{h-1} units in the layer below, plus a bias term. The weights are initialized randomly from the uniform distribution $U(-\frac{1}{\sqrt{N_{h-1}}}, \frac{1}{\sqrt{N_{h-1}}})$ except for the bias terms which are all zero.

The training data consists of 10 clusters of 100 binary examples each for a total of $M = 1000$. The centroid of each cluster is a random 100-bit binary vector with each bit drawn independently from the binomial distribution with $p = 0.5$. An example from a particular cluster is generated by starting from the centroid and introducing noise – each bit has an independent probability 0.05 of being flipped. The test data consists of an additional 100 examples drawn from each of the 10 clusters. The distortion function Δ^h for all layers is the Hamming distance, and the optimization algorithm Θ is 10 iterations of the perceptron algorithm with a learning rate of 1. The gradient is calculated in batch mode using all 1000 training examples at once. For the second layer with $N_2 = 10$, we use exhaustive sampling since there are only $2^{10} = 1024$ possible activation values. For other layers where $N_h > 10$, the sample comprises all the 1000 activation vectors of the corresponding layer over the training set, plus a set of 1000 random binary vectors where each bit is independent and 1 with probability 0.5. Updates to the layers are made on a schedule that cycles through the layers in sequential order: $1, 2, 3, 4$. One cycle of updates constitutes an epoch. The trajectory of the training and test errors are shown in Figure 7.4 demonstrating that this sampling deep targets algorithm is capable of training this non-differentiable network reasonably well.

8 The Learning Channel and the Optimality of Backpropagation

Armed with the understanding that in order to implement learning capable of reaching minima of the error function there must be a channel conveying information about the targets to the deep weights, we can now examine the three key questions about the channel: its nature, its semantics, and its rate.

8.1 The Nature of the Channel

In terms of the nature of the channel, regardless of the hardware embodiment, there are two main possibilities. Information about the targets can travel to the deep weights either by: (1) traveling along the physical forward connections but in the reverse direction; or (2) using a separate different channel.

8.1.1 Using the Forward Channel in the Backward Direction

In essence, this is the implementation that is typically emulated on digital computers using the transpose of the forward matrices in the backpropagation algorithm. However, the first thing to observe, is that even when the same channel is being used in the forward and backward direction, the signal itself does not need to be of the same nature.

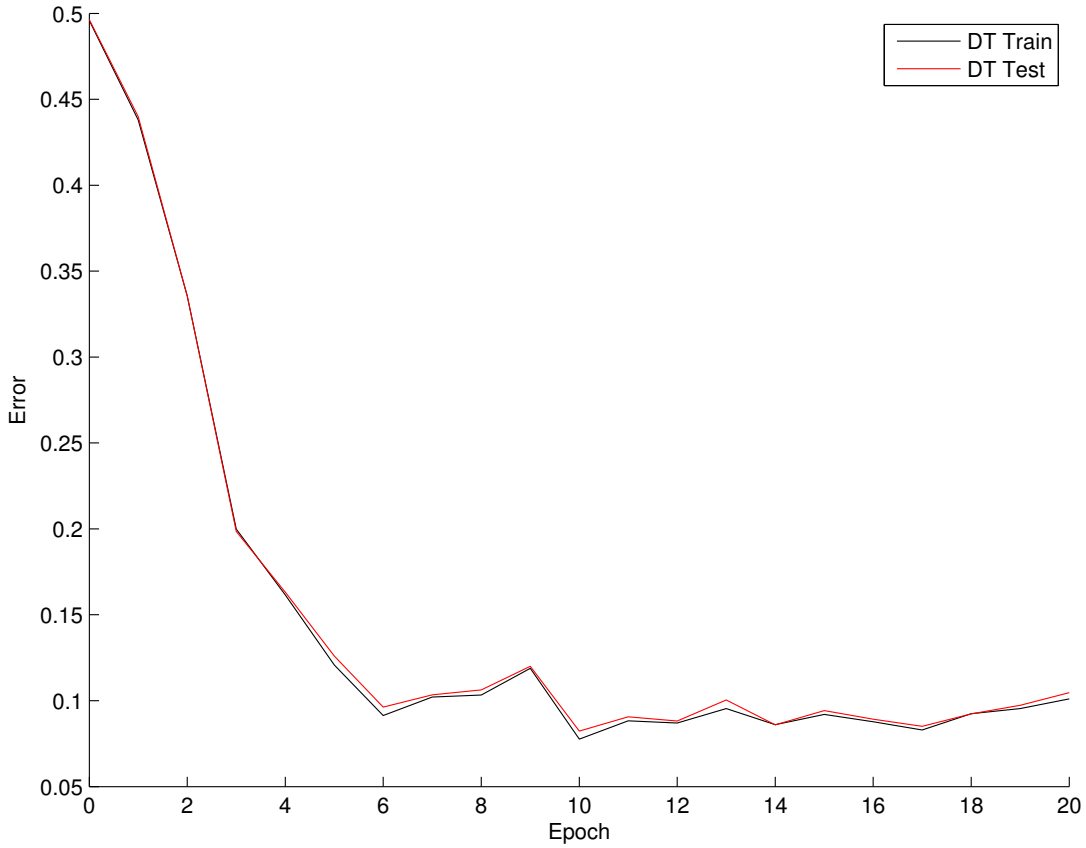

Figure 7.4: A sampling deep target (DT) algorithm is used to train a simple autoencoder network of threshold gates thus purely comprised of non-differentiable transfer functions. The y axis correspond to the average Hamming error per component and per example.

For instance the forward propagation could be electrical while the backward propagation could be chemical. This is congruent with the observation that the two signals can have very different time scales with the forward propagation being fast compared to learning which can occur over longer time scales. In biological neural networks, there is evidence for the existence of complex molecular signaling cascades traveling from the synapses of a neuron to the DNA in its nucleus, capable of activating epigenetics modifications and gene expression, and conversely for molecular signals traveling from the DNA to the synapses. At least in principle, chemical or electrical signals could traverse synaptic clefts in both directions. In short, while there is no direct evidence supporting the use of the same physical connections in both directions in biological neural systems, this possibility cannot be ruled out entirely at this time and conceivably it could be used in other hardware embodiments. It should be noted that a deep targets algorithm in which the feedback reaches the soma of a unit leads to a simpler feedback channel but puts the burden on the unit to propagate the central message from the soma to the synapses.

8.1.2 Using a Separate Backward Learning Channel

If a separate channel is used, the channel must allow the transfer of information from the output layer and the targets to the deep weights. This transfer of information could occur through direct connections from the output layer to each deep layer, or through a staged process of propagation through each level (as in backpropagation). Obviously combination of both processes are also possible. In either case, the new channel implements some form of feedback. Note again that the learning feedback can be slow and distinct in terms of signal, if not also in terms of channel, from the usual feedback of recurrent networks that is typically used in rapid dynamic mode to fine tune a rapid response, e.g. by helping combine a top down generative model with a bottom up recognition model.

In biological neuronal circuits, there are plenty of feedback connections between different processing stages (e.g. [21]) and some of these connections could serve as the primary channel for carrying the feedback signal necessary for learning. It must be noted, however, that given a synaptic weight w_{ij}, the feedback could typically reach either the dendrites of neuron i, or the dendrites of neuron j or both. In general, the dendrites of the presynaptic neuron j are physically far away from the synapse associated with w_{ij} which is located on the dendritic tree of the post-synaptic

neuron i, raising again a problem of information transmission within neuron j from its incoming synapses on its dendritic tree to the synapses associated with its output at the end of the arborization of its axon. The feedback reaching the dendrites of neuron i could in principle be much closer to the site where w_{ij} is implemented, although this requires a substantial degree of organization of the synapses along the dendrites of neuron i, spatially combining synapses originating from feedforward neurons with synapses originating from feedback neurons, together with the biochemical mechanisms required for the local transmission of the feedback information between spatially close synapses. In short, the nature of the feedback channel depends on the physical implementation. While a clear understanding of how biological neural systems implement learning is still out of reach, the framework presented here–in particular the notions of local learning and deep targets algorithms–clarifies some of the aspects and potential challenges associated with the feedback channel and the complex geometry of neurons.

An important related issue is the issue of the symmetry of the weights. Backpropagation uses symmetric (transposed) weights in the forward and backward directions in order to compute exact gradients. In a physical implementation, especially one that uses different channels for the forward and backward propagation of information, it may be difficult to instantiate weights that are precisely symmetric. However simulations [40] seem to indicate that, for instance, random weights can be used in the backward direction without affecting too much the speed of learning, or the quality of the solutions. Random weights in general result in matrices that have the maximum rank allowed by the size of the layers and thus transmit as much information as possible in the reverse direction, at least globally at the level of entire layers. How this global transmission of information allow precise learning is not entirely clear. But at least in the simple case of a network with one hidden layer and one output unit, it is easy to give a mathematical proof that random weights will support convergence and learning, provided the random weights have the same sign as the forward weights. It is plausible that biological networks could use non-symmetric connections, and that these connections could possibly be random, or random but with the same sign as the forward connections.

8.2 The Semantics of the Channel

Regardless of the nature of the channel, next one must consider the meaning of the information that is being transmitted to the deep weights, as well as its amount. Whatever information about the targets is fed back, it is ultimately used within each epoch to change the weights in the form

$$\Delta w_{ij}^h = \eta_{ij}^h E[F(I_{ij}^h, O_i^h, O_j^{h-1}, w_{ij}^h)] \tag{68}$$

so that with small learning rates η_{ij}^h a Taylor expansion leads to

$$\begin{aligned} E_{err}(w_{ij}^h + \Delta w_{ij}^h) &= E_{err}(w_{ij}^h) + \sum_{w_{ijh}} \frac{\partial E_{err}}{\partial w_{ij}^h} \Delta w_{ij}^h + \frac{1}{2}(\Delta w_{ij}^h)^t H(\Delta w_{ij}^h) + R & (69) \\ &= E_{err}(w_{ij}^h) + G \cdot (\Delta w_{ij}^h) + \frac{1}{2}(\Delta w_{ij}^h)^t \cdot H(\Delta w_{ij}^h) + R & (70) \end{aligned}$$

where G is the gradient, H is the Hessian, and R is the higher order remainder. If we let W denote the total number of weights in the system, the full Hessian has W^2 entries and thus in general is not computable for large W, which is the case of interest here. Thus limiting the expansion to the first order:

$$E_{err}(w_{ij}^h + \Delta w_{ij}^h) \approx E_{err}(w_{ij}^h) + G \cdot (\Delta w_{ij}^h) = E_{err}(w_{ij}^h) + \eta ||G|| u \cdot g = E_{err}(w_{ij}^h) + \eta ||G|| \mathcal{O} \tag{71}$$

where u is the unit vector associated with the weight adjustments ($\eta u = (\Delta w_{ij}^h)$), g is the unit vector associated with the gradient ($g = G/||G||$), and $\mathcal{O} = g \cdot u$. Thus to a first order approximation, the information that is sent back to the deep weights can be interpreted in terms of how well it approximates the gradient G, or how many bits of information it provides about the gradient. With W weights and a precision level of D-bits for any real number, the gradient contains WD bits of information. These can in turn be split into $D-1$ bits for the magnitude $||G||$ of the gradient (a single positive real number), and $(W-1)D+1$ bits to specify the direction by the corresponding unit vector ($g = G/||G||$), using $W-1$ real numbers plus one bit to determine the sign of the remaining component. Thus most of the information of the gradient in a high-dimensional space is contained in its *direction*. The information I_{ij}^h determines Δw_{ij}^h and thus the main questions is how many bits the vector (Δw_{ij}^h) conveys about G, which is essentially how many bits the vector u conveys about g, or how close is u to g? With a full budget of B bits per

weight, the gradient can be computed with B bits of precision, which defines a box around the true gradient vector, or a cone of possible unitary directions u. Thus the expectation of $\mathcal{O}$ provides a measure of how well the gradient is being approximated and how good is the corresponding optimization step (see next section).

Conceivably, one can also look at regimes where even *more* information than the gradient is transmitted through the backward channel. This could include, for instance, second order information about the curvature of the error function. However, as mentioned above, in the case of large deep networks this seems problematic since with W weights, this procedure would scale like W^2. Approximations that essentially compute only the diagonal of the Hessian matrix, and thus only W additional numbers, have been considered [39], using a procedure similar to back-propagation that scales like W operations. These methods were introduced for other purposes (e.g. network pruning) and do not seem to have led to significant or practically useful improvements to deep learning methods. Furthermore, they do not change the essence of following scaling computations.

8.3 The Rate and other Properties of the Channel

Here we want to compare several on-line learning algorithms where information about the targets is transmitted back to the deep weights and define and compute a notion of transmission rate for the backward channel. We are interested in estimating a number of important quantities in the limit of large networks. The estimates do not need to be very precise, we are primarily interested in expectations and scaling behavior. Here all the estimates are computed for the adjustment of all the weights on a given training example and thus would have to be multiplied by a factor M for a complete epoch. In particular, given a training example, we want to estimate the scaling of:

- The number $\mathcal{C}_{\mathcal{W}}$ of computations required to transmit the backward information per network weight. The estimates are computed in terms of number of elementary operations which are assumed to have a fixed unit cost. Elementary operations include addition, multiplication, computing the value of a transfer function, and computing the value of the derivative of the transfer function. We also assume the same costs for the forward or backward propagation of information. Obviously these assumptions in essence capture the implementation of neural networks on digital computers but could be revised when considering completely different physical implementations. With these assumptions, the total number of computations required by a forward pass or a backpropagation through the network scales like W, and thus $\mathcal{C}_{\mathcal{W}} = 1$ for a forward or backward pass.

- The amount of information $\mathcal{I}_{\mathcal{W}}$ that is sent back to each weight. In the case of deep targets algorithms, we can also consider the amount of information $\mathcal{I}_{\mathcal{N}}$ that is sent back to each hidden unit, from which a value $\mathcal{I}_{\mathcal{W}}$ can also be derived. We let D (for double precision) denote the number of bits used to represent a real number in a given implementation. Thus, for instance, the backpropagation algorithm provides D bits of information to each unit and each weight ($\mathcal{I}_{\mathcal{W}} = \mathcal{I}_{\mathcal{N}} = D$) for each training example, associated with the corresponding derivative.

- We define the *rate* $\mathcal{R}$ of the backward channel of a learning algorithm by $\mathcal{R} = \mathcal{I}_{\mathcal{W}}/\mathcal{C}_{\mathcal{W}}$. It is the number of bits (about the gradient) transmitted to each weight through the backward channel divided by the number of operations required to compute/transmit this information per weight. Note that the rate is bounded by D: $\mathcal{R} \leq D$. This is because the maximal information that can be transmitted is the actual gradient corresponding to D bits per weight, and the minimal computational/transmission cost must be at least one operation per weight.

- It is also useful to consider the improvement or expected improvement $\mathcal{O}'$, or its normalized version $\mathcal{O}$. All the algorithms to be considered ultimately lead to a learning step ηu where η is the global learning rate and u is the vector of weight changes. To a first order of approximation, the corresponding improvement is computed by taking the dot product with the gradient so that $\mathcal{O}' = \eta u \cdot G$. In the case of (stochastic) gradient descent we have $\mathcal{O}' = \eta G \cdot G = \eta ||G||^2$. In gradient descent, the gradient provides both a direction and a magnitude of the corresponding optimization step. In the perturbation algorithms to be described, the perturbation stochastically produces a direction but there is no natural notion of magnitude. Since when W is large most of the information about the gradient is in its direction (and not its magnitude), to compare the various algorithms we can simply compare the directions of the vector being produced,in particular in relation to the direction of the gradient. Thus we will assume that all the algorithms produce a step of the form ηu where $||u|| = 1$ and thus $\mathcal{O}' = \eta u \cdot G = \eta ||G|| u \cdot g = \eta ||G|| \mathcal{O}$. Note that the maximum possible value of $\mathcal{O} = u \cdot g$ is one, and corresponds to $\mathcal{O}' = \eta ||G||$.

To avoid unnecessary mathematical complications associated with the generation of random vectors of unit length uniformly distributed over a high-dimensional sphere, we will approximate this process by assuming in some of

the calculations that the components of u are i.i.d. Gaussian with mean 0 and variance $1/W$ (when perturbing the weights). Equivalently for our purposes, we can alternatively assume the components of u to be i.i.d. uniform over $[-a, a]$, with $a = \sqrt{3/W}$, so that the mean is also 0 and variance $1/W$. In either case, the square of the norm u^2 tends to be normally distributed by the central limit theorem, with expectation 1. A simple calculation shows that the variance of u^2 is given by $2/W$ in the Gaussian case, and by $4/5W$ in the uniform case. Thus in this case $G \cdot u$ tends to be normally distributed with mean 0 and variance $C||G||^2/W$ (for some constant $C > 0$) so that $\mathcal{O}' \approx \eta\sqrt{C}||G||/\sqrt{W}$.

In some calculations, we will also require all the components of u to be positive. In this case, it is easier to assume that the components of u are i.i.d. uniform over $[0, a]$ with $a = \sqrt{3/W}$, so that u^2 tends to be normally distributed by the central limit theorem, with expectation 1 and variance $4/5W$. Thus in this case $G \cdot u$ tends to be normally distributed with mean $(\sqrt{3/W}/2)\sum_i G_i$ and variance $||G||^2/(4W)$ so that $\mathcal{O}' \approx \eta(\sqrt{3/W}/2)\sum_i G_i$.

8.4 A Spectrum of Descent Algorithms

In addition to backpropagation (BP) which is one way of implementing stochastic gradient descent, we consider stochastic descent algorithms associated with small perturbations of the weights or the activities. These algorithms can be identified by a name of the form P{W or A}{L or G}{B or R}{$\emptyset$ or K}. The perturbation (P) can be applied to the weights (W) or, in deep targets algorithms, to the activities (A). The perturbation can be either local (L) when applied to a single weight or activity, or global (G) when applied to all the weights or activities. The feedback provided to the network can be either binary (B) indicating whether the perturbation leads to an improvement or not, or real (R) indicating the magnitude of the improvement. Finally, the presence of K indicates that the corresponding perturbation is repeated K times. For brevity, we focus on the following main cases (other cases, including intermediary cases between local and global where, for instance, perturbations are applied layerwise can be analyzed in similar ways and do not offer additional insights or improvements):

- PWGB is the stochastic descent algorithm where all the weights are perturbed by a small amount. If the error decreases the perturbation is accepted. If the error increases the perturbation is rejected. Alternatively the opposite perturbation can be accepted, since it will decrease the error (in the case of differentiable error function and small perturbations), however this is detail since at best it speeds things up by a factor of two.
- PWLR is the stochastic descent algorithm where each weight in turn is perturbed by a small amount and the feedback provided is a real number representing the change in the error. Thus this algorithm corresponds to the computation of the derivative of the error with respect to each weight using the definition of the derivative. In short, it corresponds also to stochastic gradient descent but provides a different mechanism for computing the derivative. It is not a deep targets algorithm.
- PWLB is the binary version of PWLR where only one bit, whether the error increases or decreases, is transmitted back to each weight upon its small perturbation. Thus in essence this algorithms provides the sign of each component of the gradient, or the orthant in which the gradient is located, but not its magnitude. After cycling once through all the weights, a random descent unit vector can be generated in the corresponding orthant (each component of u_i has the sign of g_i).
- PALR is the deep targets version of PWLR where the activity of each unit in turn is perturbed by a small amount, thus providing the derivative of the error with respect to each activity, which in turn can be used to compute the derivative of the error with respect to each weight.
- PWGBK is similar to PWGB, except that K small global perturbations are produced, rather than a single one. In this case, the binary feedback provides information about which perturbation leads to the largest decrease in error.
- PWGRK is similar to PWGBK except that for each perturbation a real number, corresponding to the change in the error, is fed back. This corresponds to providing the value of the dot product of the gradient with K different unit vector directions.

8.5 Analysis of the Algorithms: the Optimality of Backpropagation

8.5.1 Global Weight Perturbation with Binary Feeback (PWGB)

For each small global perturbation of the weights, this algorithm transmits a single bit back to all the weights, corresponding to whether the error increases or decreases. This is not a deep targets algorithm. The perturbation itself

requires one forward propagation, leading to $\mathcal{I}_{\mathcal{W}} = 1/W$ and $\mathcal{C}_{\mathcal{W}} = 1$. Thus:

- $\mathcal{I}_{\mathcal{W}} = 1/W$
- $\mathcal{C}_{\mathcal{W}} = 1$
- $\mathcal{R} = 1/W$
- $\mathcal{O}' = \eta C ||G|| / \sqrt{W}$ for some constant $C > 0$, so that $\mathcal{O} = C/\sqrt{W}$

8.5.2 Local Weight Perturbation with Real Feedback (PWLR)

This is the definition of the derivative. The derivative $\partial E_{err}/\partial w_{ij}^h$ can also be computed directly by first perturbing w_{ij}^h by a small amount ϵ, propagating forward, measuring $\Delta E_{err} = E_{err}(w_{ij} + \epsilon) - E_{err}(w_{ij})$ and then using $\partial E_{err}/\partial w_{ij} \approx \Delta E_{err}/\epsilon$. This is not a deep target algorithm. The algorithm computes the gradient and thus propagates D bits back to each weight, at a total computational cost that scales like W per weight, since it essentially requires one forward propagation for each weight. Thus:

- $\mathcal{I}_{\mathcal{W}} = D$
- $\mathcal{C}_{\mathcal{W}} = W$
- $\mathcal{R} = D/W$
- $\mathcal{O}' = \eta ||G||$ for a step ηg, and thus $\mathcal{O} = 1$

8.5.3 Local Weight Perturbation with Binary Feedback (PWLB)

This is not a deep target algorithm. The algorithm provides a single bit of information back to each weigh and requires a forward propagation to do so. Without any loss of generality, we can assume that all the components of the final random descent vector u_i must be positive. Thus:

- $\mathcal{I}_{\mathcal{W}} = 1$
- $\mathcal{C}_{\mathcal{W}} = W$
- $\mathcal{R} = 1/W$
- $\mathcal{O}' = \eta(\sqrt{3/W}/2) \sum_i G_i$, and thus $\mathcal{O} = (\sqrt{3/W}/2) \sum_i g_i$

8.5.4 Local Activity Perturbation with Real Feedback (PALR)

This is a deep target algorithm and from the computation of the derivative with respect to the activity of each unit, one can derive the gradient. So it provides D bits of feedback to each unit, as well as to each weight. The algorithm requires in total N forward propagations, one for each unit, resulting in a total computational cost of NW or N per weight. Thus:

- $\mathcal{I}_{\mathcal{W}} = \mathcal{I}_{\mathcal{N}} = D$
- $\mathcal{C}_{\mathcal{W}} = N$
- $\mathcal{R} = D/N$
- $\mathcal{O}' = \eta ||G||$ for a step ηg, and thus $\mathcal{O} = 1$

8.5.5 Global Weight Perturbation with Binary Feedback K Times (PWGBK)

In this version of the algorithm, the information backpropagated is which of the K perturbation leads to the best improvement, corresponding to the same $\log K$ bits for all the weights. The total cost is K forward propagations. Note that the K perturbations constrain the gradient to be in the intersection of K hyperplanes, and this corresponds to more information than retaining only the best perturbation. However the gain is small enough that a more refined version of the algorithm and the corresponding calculations are not worth the effort. Thus here we just use the best perturbation. We have seen that for each perturbation the dot product of the corresponding unit vector u with G is essentially normally distributed with mean 0 and variance $C||G||^2/W$. The maximum of K samples of a normal distribution (or the absolute value of the samples if random ascending directions are inverted into descending directions) follows an extreme value distribution [17, 23] and the average of the maximum will scale like the standard deviation times a factor $\sqrt{\log K}$ up to a multiplicative constant. Thus:

- $\mathcal{I}_{\mathcal{W}} = \log K/W$
- $\mathcal{C}_{\mathcal{W}} = K$
- $\mathcal{R} = (\log K/W)/K$
- $\mathcal{O}' = \eta C||G||\sqrt{\log K}/\sqrt{W}$ for some constant $C > 0$, and thus $\mathcal{O} = C\sqrt{\log K}/\sqrt{W}$

8.5.6 Global Weight Perturbation with Real Feedback K Times (PWGRK)

This algorithm provides KD bits of feedback in total, or KD/W per weight and requires K forward propagations. In terms of improvements, let us consider that the algorithms generates K random unit vector directions $u^{(1)}, \ldots, u^{(K)}$ and produces the K dot products $u^{(1)} \cdot G, \ldots, u^{(K)} \cdot G$. In high dimensions ($W$ large), the K random directions are approximately orthogonal As a result of this information, one can select the unit descent direction given by

$$u = \frac{\sum_{k=1}^{K}(u^{(k)} \cdot G)u^{(k)}}{||\sum_{k=1}^{K}(u^{(k)} \cdot G)u^{(k)}||} \tag{72}$$

Now we have

$$||\sum_{k=1}^{K}(u^{(k)} \cdot G)u^{(k)}||^2 \approx \sum_{k=1}^{K}(u^{(k)} \cdot G)^2 \approx CK\frac{||G||^2}{W} \tag{73}$$

for some constant $C > 0$. The first approximation is because the vectors $u^{(k)}$ are roughly orthogonal, and the second is simply by taking the expectation. As a result, $\mathcal{O} = \eta u \cdot G = \eta C\sqrt{K}||G||/\sqrt{W}$ for some constant $C > 0$. Thus:

- $\mathcal{I}_{\mathcal{W}} = KD/W$
- $\mathcal{C}_{\mathcal{W}} = K$
- $\mathcal{R} = D/W$
- $\mathcal{O}' = \eta C\sqrt{K}||G||/\sqrt{W}$, and thus $\mathcal{O} = C\sqrt{K}/\sqrt{W}$

8.5.7 Backpropagation (BP)

As we have already seen:

- $\mathcal{I}_{\mathcal{W}} = \mathcal{I}_{\mathcal{N}} = D$
- $\mathcal{C} = 1$
- $\mathcal{R} = D$
- $\mathcal{O}' = \eta||G||$ for a step ηg, and thus $\mathcal{O} = 1$

Algorithm	Information $\mathcal{I}_{\mathcal{W}}$	Computation $\mathcal{C}_{\mathcal{W}}$	Rate $\mathcal{R}$	Improvement $\mathcal{O}$
PWGB	$1/W$	1	$1/W$	$C/\sqrt{W}$
PWLR	D	W	D/W	1
PWLB	1	W	$1/W$	$(\sqrt{3/W}/2)\sum_i g_i$
PALR	D	N	D/N	1
PWGBK	$\log K/W$	K	$(\log K/W)/K$	$C\sqrt{\log K}/\sqrt{W}$
PWGRK	KD/W	K	D/W	$C\sqrt{K}/\sqrt{W}$
BP	D	1	D	1

Table 8: The rate $\mathcal{R}$ and improvement $\mathcal{O}$ of several optimization algorithms.

The reason that no algorithms better than backpropagation has been found is that the rate $\mathcal{R}$ of backpropagation is greater or equal to that of all the alternatives considered here (Table 8). This is true also for the improvement $\mathcal{O}$. Furthermore, there is no close second: all the other algorithms discussed in this section fall considerably behind backpropagation in at least one dimension. And finally, it is unlikely that an algorithm exists with a rate or improvement higher than backpropagation, because backpropagation achieves both the maximal possible rate, and maximal possible improvement (Figure 8.1), up to multiplicative constants. Thus in conclusion we have the following theorem:

Theorem: The rate $\mathcal{R}$ of backpropagation is above or equal to the rate of all the other algorithms described here and it achieves the maximum possible value $\mathcal{R} = D$. The improvement $\mathcal{O}$ of backpropagation is above or equal to the improvement of all the other algorithms described here and it achieves the maximum possible value $\mathcal{O} = 1$ (or $\mathcal{O}' = \eta||G||$).

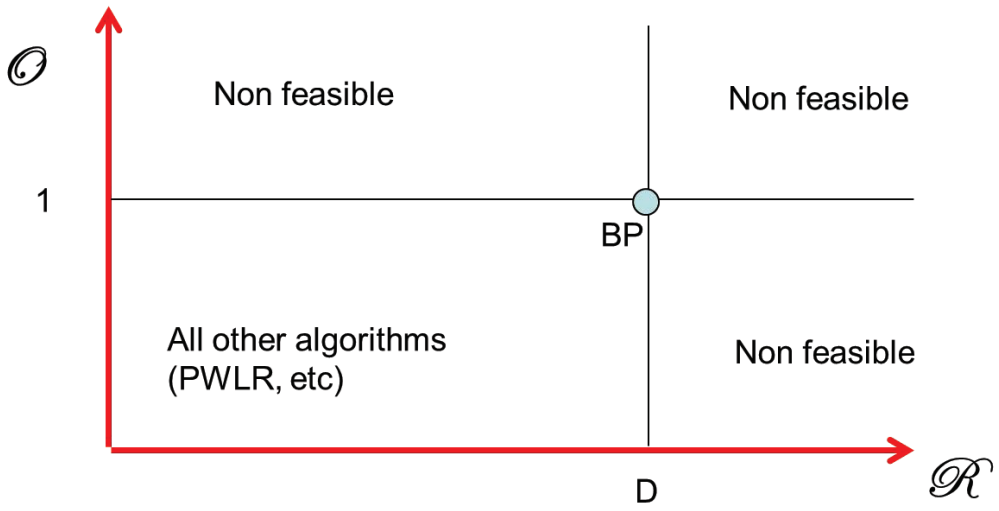

Figure 8.1: Backpropagation is optimal in the space of possible learning algorithms, achieving both the maximal possible rate $\mathcal{R} = D$ and maximal expected improvement $\mathcal{O} = 1$.

8.6 Recurrent Networks

Remarkably, the results of the previous sections can be extended to recurrent, as well as recursive, networks (see [57] for an attempt at implementing backpropagation in recurrent networks using Hebbian learning). To see this, consider a recurrent network with W connection weights, where the connections can form directed cycles. If the network is unfolded in time over L time steps, one obtains a deep feedforward network (Figure 8.2), where the same sets of original weights from the recurrent networks is used to update all the unit activations, from one layer (or time step)

to the next. Thus the unfolded version has a set of W weights that are shared L times. In the recurrent case, one may have targets for all the units at all the time steps, or more often, targets may be available only at some time steps, and possibly only for some of the units. Regardless of the pattern of available targets, the same argument used in Section 6.4 to expose the limitations of deep local learning, exposes the limitations of local learning in recurrent networks. More precisely, under the assumption that the error function is a differentiable function of the weights, any algorithm capable of reaching an optimal set of weights– where all the partial derivatives are zero–must be capable of "backpropagating" the information provided by any target at any time step to all the weights capable of influencing the corresponding activation. This is because a target T_i^l for unit i at time l will appear in the partial derivative of any weight present in the recurrent network that is capable of influencing the activity of unity i in l steps or less. Thus, in general, an implementation capable of reaching an optimal set of weights must have a "channel" in the unfolded network capable of transmitting information from T_i^L back to all the weights in all the layers up to l that can influence the activity of unity i in layer l. Again in a large recurrent network the maximal amount of information that can be sent back is the full gradient and the minimal number of operations required typically scales like WL. Thus this shows that the backpropagation through time algorithm is optimal in the sense of providing the most information, i.e. the full gradient, for the least number of computations (WL).

Boltzmann machines [1], which can be viewed as a particular class of recurrent networks with symmetric connections, can have hidden nodes and thus be considered deep. Although their main learning algorithm can be viewed as a form of simple Hebbian learning ($\Delta w_{ij} \propto < O_i O_j >_{clamped} - < O_i O_j >_{free}$), they are no exception to the previous analyses. This is because the connections of a Boltzmann machines provide a channel allowing information about the targets obtained at the visible units to propagate back towards the deep units. Furthermore, it is well known that this learning rule precisely implements gradient descent with respect to the relative divergence between the true and observed distributions of the data, measured at the visible units. Thus the Hebbian learning rule for Boltzmann machines implements a form of local deep learning which in principle is capable of transmitting the maximal amount of information, from the visible units to the deep units, equal to the gradient of the error function. What is perhaps less clear is the computational cost and how it scales with the total number of weights W, since the learning rule in principle requires the achievement of equilibrium distributions.

Finally, the nature of the learning channel, and its temporal dynamics, in physical recurrent networks, including biological neural networks, are important but beyond the scope of this paper. However, the analysis provided is already useful in clarifying that the backward recurrent connections could serve at least three different roles: (1) a fast role to dynamically combine bottom-up and top-down activity, for instance during sensory processing; (2) a slower role to help carry signals for learning the feedforward connections; and (3) a slower role to help carry signals for learning the backward connections.

9 Conclusion

The concept of Hebbian learning has played an important role in computational neuroscience, neural networks, and machine learning for over six decades. However, the vagueness of the concept has hampered systematic investigations and overall progress. To redress this situation, it is beneficial to expose two separate notions: the locality of learning rules and their functional form. Learning rules can be viewed as mathematical expressions for computing the adjustment of variables describing synapses during learning, as a function of variables which, in a physical system, must be local. Within this framework, we have studied the space of polynomial learning rules in linear and non-linear feedforward neural networks. In many cases, the behavior of these rules can be estimated analytically and reveals how these rules are capable of extracting relevant statistical information from the data. However, in general, deep local learning associated with the stacking of local learning rules in deep feedforward networks is not sufficient to learn complex input-output functions, even when targets are available for the top layer.

Learning complex input-output functions requires a learning channel capable of propagating information about the targets to the deep weights and resulting in local deep learning. In a physical implementation, this learning channel can use either the forward connections in the reverse direction, or a separate set of connections. Furthermore, for large networks, all the information carried by the feedback channel can be interpreted in terms of the number of bits of information about the gradient provided to each weight. The capacity of the feedback channel can be defined in terms of the number of bits provided about the gradient per weight, divided by the number of required operations per weight. The capacity of many possible algorithms can be calculated, and the calculations show that backpropagation outperforms all other algorithms as it achieves the maximum possible capacity. This is true in both feedforward and recurrent networks. It must be noted, however, that these results are obtained using somewhat rough estimates–up to multiplicative constants–and there may be other interesting algorithms that scale similarly to backpropagation. In

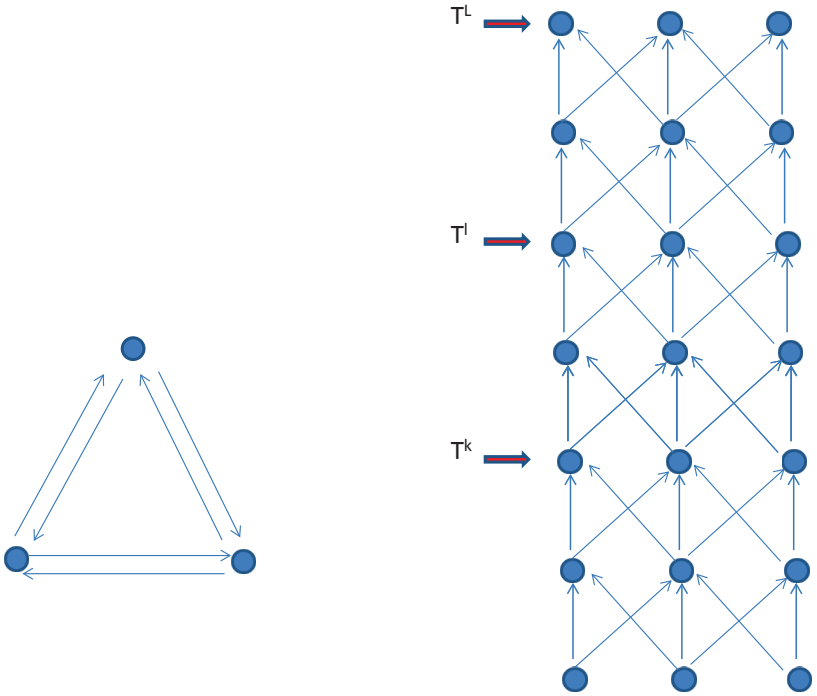

Figure 8.2: Left: Recurrent neural network with three neurons and $W = 6$ connection weights. Right: The same network unfolded through time for L steps, producing a deep feedforward network where the weights are shared between time steps. Each original weight is replicated L times . Targets are available for a subset of the layers (i.e. time steps). In order to reach an optimal set of weights, a learning algorithm must allow each individual target to influence all the copies of all the weights leading to the corresponding unit.

particular, we are investigating the use of random, as opposed to symmetric, weights in the learning channel, which seems to work in practice [40].

The remarkable optimality of backpropagation suggests that when deploying learning systems in computer environments with specific constraints and budgets (in terms of big data, local storage, parallel implementations, communication bandwidth, etc) backpropagation provides the upper bound of what can be achieved, and the model that should be emulated or approximated by other systems.

Likewise, the optimality of backpropgation leads one to wonder also whether, by necessity, biological neural systems must have discovered some form of stochastic gradient descent during the course of evolution. While the question of whether the results presented here have some biological relevance is interesting, several other points must be taken into consideration. First, the analyses have been carried in the simplified supervised learning setting, which is not meant to closely match how biological systems learn. Whether the supervised learning setting can approximate at least some essential aspects of biological learning is an open question, and so is the related question of extending the theory of local learning to other forms of learning, such as reinforcement learning [52].

Second, the analyses have been carried using artificial neural network models. Again the question of whether these networks capture some essential properties of biological networks is not settled. Obviously biological neurons are very complex biophysical information processing machines, far more complex than the neurons used here. On the other hand, there are several examples in the literature (see, for instance, [60, 45, 46, 2, 58]) where important biological properties seem to be captured by artificial neural network models. [In fact these results, taken together with the sometimes superhuman performance of backpropagation and the optimality results presented here, lead us to conjecture paradoxically that biological neurons may be trying to approximate artificial neurons, and not the other way around, as has been assumed for decades.] But even if they were substantially unrelated to biology, artificial neural networks still provide the best simple model we have of a connectionist style of computation and information storage, entirely different from the style of digital computers, where information is both scattered and superimposed across synapses and intertwined with processing, rather than stored at specific memory addresses and segregated from processing.

In any case, for realistic biological modeling, the complex geometry of neurons and their dendritic trees must be taken into consideration. For instance, there is a significant gap between having a feedback error signal B_i arrive at the soma of neuron i, and having B_i available as a local variable in a far away synapse located in the dentritic tree of neuron i. In other words, B_i must become a local variable at the synapse w_{ij}. Using the same factor of 10^6 from the Introduction, which rescales a synapse to the size of a fist, this gap could correspond to tens or even hundreds of meters. Furthermore, in a biological or other physical system, one must worry about locality not only in space, but also in *time*, e.g. how close must B_i and O_j be in time?

Third, issues of coordination of learning across different brain components and regions must also be taken into consideration (e.g. [53]). And finally, a more complete model of biological learning would have to include not only target signals that are backpropagated electrically, but ultimately also the complex and slower biochemical processes involved in synaptic modification, including gene expression and epigenetic modifications, and the complex production, transport, sequestration, and degradation of protein, RNA, and other molecular species (e.g. [25, 41, 54]).

However, while there is no definitive evidence in favor or against the use of stochastic gradient descent in biological neural systems, and obtaining such evidence remains a challenge, biological deep learning must follow the locality principle and thus the theory of local learning provides a framework for investigating this fundamental question.

Appendix A: Uniqueness of Simple Hebb in Hopfield Networks

A Hopfield model can be viewed as a network of N $[-1, 1]$ threshold gates connected symmetrically ($w_{ij} = w_{ji}$) with no self-connections ($w_{ii} = 0$). As a result the network has a quadratic energy function $E = -(1/2)\sum_{ij} w_{ij}O_iO_j$ and the dynamics of the network under stochastic asynchronous updates converges to local minima of the energy function. Given a set $\mathcal{S}$ of M memory vectors $M^{(1)}, M^{(2)}, \dots M^{(M)}$, the simple Hebb rule is used to produce an energy function $E_{\mathcal{S}}$ to try to store these memories as local minima of the energy function so that $w_{ij} = \sum_k M_i^{(k)} M_j^{(k)}$. $E_{\mathcal{S}}$ induces an acyclic orientation $\mathcal{O}(\mathcal{S})$ of the N dimensional hypercube $\mathcal{H}$. If h is an isometry of $\mathcal{H}$ for the Hamming distance, then for the simple Hebb rule we have $\mathcal{O}(h(\mathcal{S})) = h(\mathcal{O}(\mathcal{S}))$. Are there any other learning rules with the same property?

We consider here learning rules with $d = 0$. Thus we must have $\Delta w_{ij} = F(O_i, O_j)$ where F is a polynomial function. On the [-1,1] hypercube, we have $O_i^2 = O_j^2 = 1$ and thus we only need to consider the case $n = 2$ with $F(O_i, O_j) = \alpha O_iO_j + \beta O_i + \gamma O_j + \delta$. However the learning rule must be symmetric in i and j to preserve the symmetry $w_{ij} = w_{ji}$. Therefore F can only have the form $F(O_i, O_j) = \alpha O_iO_j + \beta(O_i + O_j) + \gamma$. Finally, the isometric invariance must be true for *any* set of memories $\mathcal{S}$. It is easy to construct examples, with specific sets $\mathcal{S}$, that force β and γ to be 0. Thus in this sense the simple Hebb rule $\Delta w_{ij} = \alpha O_iO_j$ is the only isometric invariant learning rule for the Hopfield model. A similar results can be derived for spin models with higher-order interactions where the energy function is a polynomial of degree $n > 2$ in the spin variables [5].

Appendix B: Invariance of the Gradient Descent Rule

In the $[0, 1]$ case with the logistic transfer function, the goal is to minimize the relative entropy error

$$E = -\left[T \log O + (1 - T) \log(1 - O)\right] \tag{74}$$

Therefore

$$\frac{\partial E}{\partial O} = \frac{T - O}{O(1 - O)} \quad \text{and} \quad \frac{\partial O}{\partial S} = O(1 - O) \quad \text{and thus} \quad \Delta w_i = \eta(T - O)I_i \tag{75}$$

In the $[-1, 1]$ case with the $\tanh$ transfer function, the equivalent goal is to minimize

$$E' = -\left[\frac{T' + 1}{2} \log \frac{O' + 1}{2} + \frac{1 - T'}{2} \log \frac{1 - O'}{2}\right] \tag{76}$$

where $T = \frac{T'1+1}{2}$ and $O = \frac{O'+1}{2}$. Therefore

$$\frac{\partial E'}{\partial O'} = \frac{2(T' - O')}{1 - O'^2} \quad \text{and} \quad \frac{\partial O'}{\partial S'} = 1 - O'^2 \quad \text{and thus} \quad \Delta w_i' = \eta 2(T' - O')I_i' \tag{77}$$

Thus the gradient descent learning rule is the same, up to a factor of 2 which can be absorbed by the learning rule. The origin of this factor lies in the fact that $\tanh(x) = (1-e^{-2x})/(1+e^{-2x})$ is actually *not* the natural $[-1,1]$ equivalent of the logistic function $\sigma(x)$. The natural equivalent is

$$\tanh\frac{x}{2} = 2\sigma(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}} \tag{78}$$

Appendix C: List of New Convergent Learning Rules

All the rules are based on adding a simple decay term to the simple Hebb rule and its supervised variants.

Fixed Decay

$$\Delta w_{ij} \propto O_i O_j - C w_{ij} \quad \text{with} \quad C > 0 \tag{79}$$

with the supervised clamped version

$$\Delta w_{ij} \propto T_i O_j - C w_{ij} \tag{80}$$

and the gradient descent version

$$\Delta w_{ij} \propto (T_i - O_i) O_j - C w_{ij} \tag{81}$$

Adaptive Decay Depending on the Presynaptic Term

$$\Delta w_{ij} \propto O_i O_j - O_j^2 w_{ij} \tag{82}$$

with the supervised clamped version

$$\Delta w_{ij} \propto T_i O_j - O_j^2 w_{ij} \tag{83}$$

and the gradient descent version

$$\Delta w_{ij} \propto (T_i - O_i) O_j - O_j^2 w_{ij} \tag{84}$$

Adaptive Decay Depending on the Postsynaptic Term

$$\Delta w_{ij} \propto O_i O_j - O_i^2 w_{ij} \tag{85}$$

This is Oja's rule, which yields the supervised clamped versions

$$\Delta w_{ij} \propto T_i O_j - O_i^2 w_{ij} \quad \text{and} \quad \Delta w_{ij} \propto T_i O_j - T_i^2 w_{ij} \tag{86}$$

and the gradient descent versions

$$\Delta w_{ij} \propto (T_i - O_i) O_j - O_i^2 w_{ij} \quad \text{and} \Delta w_{ij} \propto (T_i - O_i) O_j - (T_i - O_i)^2 w_{ij} \tag{87}$$

Adaptive Decay Depending on the Pre- and Post-Synaptic (Simple Hebb) Terms

$$\Delta w_{ij} \propto O_i O_j - (O_i O_j)^2 w_{ij} = O_i O_j (1 - O_i O_j w_{ij}) \tag{88}$$

with the clamped versions

$$\Delta w_{ij} \propto T_i O_j - (O_i O_j)^2 w_{ij} \quad \text{and} \Delta w_{ij} \propto T_i O_j - (T_i O_j)^2 w_{ij} \tag{89}$$

and gradient descent versions

$$\Delta w_{ij} \propto (T_i - O_i) O_j - (O_i O_j)^2 w_{ij} \quad \text{and} \Delta w_{ij} \propto (T_i - O_i) O_j - ((T_i - O_i) O_j)^2 w_{ij} \tag{90}$$

We now consider the alternative approach which bounds the weights in a $[-C, C]$ range for some $C < 0$. The initial values of the weights are assumed to be small or 0.

Bounded Weights

$$\Delta w_{ij} \propto O_i O_j (C - w_{ij}^2) \tag{91}$$

with the clamped version

$$\Delta w_{ij} \propto T_i O_j (C - w_{ij}^2) \tag{92}$$

and the gradient descent version

$$\Delta w_{ij} \propto (T_i - O_i) O_j (C - w_{ij}^2) \tag{93}$$

Appendix D: Additional Remarks on Deep Targets Algorithms

1) In many situations, for a given input vector I there will be a corresponding and distinct activity vector O^{h-1}. However, sometimes the function C_{h-1} may not be injective in which cases several input vectors $I(t_1), \ldots, I(t_k)$, with final targets $T(t_1), \ldots, T(t_k)$, may get mapped onto the same activity vector $O^{h-1} = C_{h-1}(I(t_1)) = \ldots = C_{h-1}(I(t_k))$. In this case, the procedure for determining the target vector T^h may need to be adjusted slightly as follows. First, the sample of activity $\mathcal{S}^h$ is generated and propagated forward using the function A_{h+1}, as in the non-injective case. However the selection of the best output vector over the sample may take into consideration all the targets $T(t_1), \ldots, T(t_k)$ rather than the isolated target associated with the current input example. For instance, the best output vector may be chosen as to minimize the sum of the errors with respect to all these targets. This procedure is the generalization of the procedure used to train an unrestricted Boolean autoencoder [7].
2) Depending on the schedule in the outer loop, the sampling approach, and the optimization algorithm used in the inner loop, as well as other implementation details, the description above provides a family of algorithms, rather than a single algorithm. Examples of schedules for the outerloop include a single pass from layer 1 to layer L, alternating up-and down passes along the architecture, cycling through the layers in the order 1,2,1,2,3,1,2,3,4, etc, and their variations.
3) The sampling deep targets approach can be combined with all the other "tricks" of backpropagation such as weight sharing and convolutional architectures, momentum, dropout, and so forth. Adjustable learning rates can be used with different adjustment rules for different learning phases [15, 16].
4) The sampling deep targets approach can be easily combined also with backpropagation. For instance, targets can be provided for every other layer, rather than for every layer, and backpropagation used to train pairs of adjacent layers. It is also possible to interleave the layers over which backpropagations is applied to better stitch the shallow components together (e.g. use backpropagations for layers 3,2,1 then 4,3,2, etc).
5) When sampling from a layer, here we have focused on using the optimal output sample to derive the target. It may be possible instead to leverage additional information contained in the entire distribution of samples.
6) In practice the algorithm converges, at least to a local minima of the error function. In general the convergence is not monotonic (Figure 7.4), with occasional uphill jumps that can be beneficial in avoiding poor local minima. Convergence can be proved mathematically in several cases. For instance, if the optimization procedure can map each

hidden activity to each corresponding target over the entire training set, then the overall training error is guaranteed to decrease or stay constant at each optimization step and hence it will converge to a stable value. In the unrestricted Boolean case (or in the Boolean case with perfect optimization), with exhaustive sampling of each hidden layer the algorithm can also be shown to be convergent. Finally, it can also be shown to be convergent in the framework of stochastic learning and stochastic component optimization [48, 16].
7 A different kind of deep targets algorithm, where the output targets are used as targets for all the hidden layers, is described in [19]. The goal in this case is to force successive hidden layers to refine their predictions towards the final target.

Acknowledgments Work supported in part by NSF grant IIS-1550705 and a Google Faculty Research Award to PB. We are also grateful for a hardware gift from NVDIA Corporation. This work was presented as a keynote talk at the 2015 ICLR Conference and a preliminary version was posted on ArXiv under the title "The Ebb and Flow of Deep Learning".

References

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] F. Anselmi, J. Leibo, L. Rosasco, J. Mutch, A. Tacchetti, , and T. Poggio. Unsupervised learning of invariant representations with low sample complexity: the magic of sensory cortex or a new framework for machine learning? *arXiv:1311.4158v5*, 2014.

[3] P. Baldi. Symmetries and learning in neural network models. *Physical Review Letters*, 59(17):1976–1978, 1987.

[4] P. Baldi. Group actions and learning for a family of automata. *Journal of Computer and System Sciences*, 36(2):1–15, 1988.

[5] P. Baldi. Neural networks, orientations of the hypercube and algebraic threshold functions. *IEEE Transactions on Information Theory*, 34(3):523–530, 1988.

[6] P. Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. *Journal of Machine Learning Research. Proceedings of 2011 ICML Workshop on Unsupervised and Transfer Learning*, 27:37–50, 2012.

[7] P. Baldi. Boolean autoencoders and hypercube clustering complexity. *Designs, Codes, and Cryptography*, 65(3):383–403, 2012.

[8] P. Baldi and P. Sadowski. Deep targets algorithms for deep learning. In *NIPS 2012: Workshop on Deep Learning and Unsupervised Feature Learning*, 2012.

[9] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial Intelligence*, 210C:78–122, 2014.

[10] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, 2014.

[11] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[12] Y. Bengio and Y. LeCun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007.

[13] E. L. Bienenstock, L. N. Cooper, and P. W. Munro. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *The Journal of Neuroscience*, 2(1):32–48, 1982.

[14] H. Block, S. Levin, and A. M. Society. *On the boundedness of an iterative procedure for solving a system of linear inequalities*. American Mathematical Society, 1970.

[15] L. Bottou. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[16] L. Bottou. Stochastic learning. In O. Bousquet and U. von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[17] S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer-Verlag, London, 2001.

[18] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on*, (3):326–334, 1965.

[19] P. Di Lena, K. Nagata, and P. Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28:2449–2457, 2012. doi: 10.1093/bioinformatics/bts475. First published online: July 30, 2012.

[20] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, Feb. 2010.

[21] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47, 1991.

[22] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[23] J. Galambos. *The Asymptotic Theory of Extreme Order Statistics*. Robert E. Krieger Publishing Company, Malabar, FL, 1987. Second Edition.

[24] A. Gelfand, L. van der Maaten, Y. Chen, and M. Welling. On herding and the perceptron cycling theorem. *Advances of Neural Information Processing Systems (NIPS)*, 23:694–702, 2010.

[25] Z. Guan, M. Giustetto, S. Lomvardas, J.-H. Kim, M. C. Miniaci, J. H. Schwartz, D. Thanos, and E. R. Kandel. Integration of long-term-memory-related synaptic plasticity involves bidirectional regulation of gene expression and chromatin structure. *Cell*, 111(4):483–493, 2002.

[26] D. Hebb. The organization of behavior: A neurophychological study. *Wiley Interscience, New York*, 1949.

[27] G. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[28] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504, 2006.

[29] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[30] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[31] A. Hyvärinen and E. Oja. Independent component analysis by general nonlinear hebbian-like learning rules. *Signal Processing*, 64(3):301–313, 1998.

[32] N. Intrator and L. N. Cooper. Objective function formulation of the bcm theory of visual cortical plasticity: Statistical connections, stability conditions. *Neural Networks*, 5(1):3–17, 1992.

[33] D. Kleitman. On Dedekind's problem: the number of monotone boolean functions. *Proceedings of the American Mathematical Society*, pages 677–682, 1969.

[34] T. Kohonen. Self-organizing maps. 1995, 1995.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[36] S. Lahiri and S. Ganguli. A memory frontier for complex synapses. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1034–1042. Curran Associates, Inc., 2013.

[37] C. C. Law and L. N. Cooper. Formation of receptive fields in realistic visual environments according to the bienenstock, cooper, and munro (bcm) theory. *Proceedings of the National Academy of Sciences*, 91(16):7797–7801, 1994.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 –2324, Nov. 1998.

[39] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, 1990.

[40] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[41] M. Mayford, S. A. Siegelbaum, and E. R. Kandel. Synapses and memory storage. *Cold Spring Harbor perspectives in biology*, 4(6):a005751, 2012.

[42] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[43] S. Muroga. Lower bounds of the number of threshold functions and a maximum weight. *Electronic Computers, IEEE Transactions on*, (2):136–148, 1965.

[44] E. Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[45] B. A. Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[46] A. Polsky, B. W. Mel, and J. Schiller. Computational subunits in thin dendrites of pyramidal cells. *Nature neuroscience*, 7(6):621–627, 2004.

[47] A. D. Polyanin and V. E. Zaitsev. *Handbook of Exact Solutions for Ordinary Differential Equations*.

[48] H. Robbins and D. Siegmund. A convergence theorem for non negative almost supermartingales and some applications. *Optimizing methods in statistics*, pages 233–257, 1971.

[49] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[50] D. Rumelhart, G. Hintont, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[51] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[53] L. G. Valiant. The hippocampus as a stable memory allocator for cortex. *Neural computation*, 24(11):2873–2899, 2012.

[54] A. Vogel-Ciernia, R. M. Barrett, D. P. Matheos, E. Kramar, S. Azzawi, Y. Chen, C. N. Magnan, M. Zeller, A. Sylvain, J. Haettig, Y. Jia, A. Tran, R. Dang, R. J. Post, M. Chabrier, A. abayan, J. I. Wu, G. R. Crabtree, P. Baldi, T. Z. Baram, G. Lynch, and M. A. Wood. The neuron-specific chromatin regulatory subunit BAF53b is necessary for synaptic plasticity and memory. *Nature Neuroscience*, 16:552–561, 2013.

[55] B. Widrow and M. Hoff. Adaptive switching circuits. In *Institute of Radio Engineers, Western Electronic Show and Converntion, Convention Record, Part 4*, pages 96–104, 1960.

[56] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[57] X. Xie and H. S. Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15(2):441–454, 2003.

[58] D. L. Yamins and J. J. DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

[59] F. Zenke, E. J. Agnes, and W. Gerstner. Diverse synaptic plasticity mechanisms orchestrated to form and retrieve memories in spiking neural networks. *Nature communications*, 6, 2015.

[60] D. Zipser and R. A. Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679–684, 1988.

[61] Y. A. Zuev. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Mathematics Doklady*, 39(3):512–513, 1989.