Chapter 7

The Local Learning Principle

This chapter and the following one leverage the principle of local learning [103] and use it to answer several fundamental questions, including: (1) what is the relationship between Hebbian learning and backpropagaton, in particular is backpropagation "Hebbian"? (2) what is the space of learning rules? (3) why has no one been able to carry out the Fukushima program of training convolutional neural networks using Hebbian learning?; (4) why is backpropagation challenging to implement in a physical system, and what are some possible alternatives?

7.1 Virtualization and Learning in the Machine

Although straightforward, the principle of local learning has remained somewhat hidden because of the ambiguitiy of Hebb's original formulation and because of virtualization issues. Virtualization is a fundamental concept in computer science which refers to the use of a computational machine or process to simulate another computational machine or process. Modern computing centers are full of virtual machines and, for instance, UNIX interfaces can be simulated on Windows machines and vice versa. However, the concept of virtualization goes back to the foundations of computer science, since the very concept of a computer relies on mapping computations or algorithms, such as adding two binary integers, from a Platonic world of mathematical abstractions to the state space dynamics of a physical system. Moreover, in the theory of Turing machines, one first develops special-purpose Turing machines dedicated to solving specific tasks, such as detecting whether a binary input represent an even or odd integer, or adding two binary inputs. This is followed by the realization that all such special-purpose Turing machines, or programs, can be simulated–or virtualized–on a single, general-purpose Turing machine, the so-called universal Turing machine. In addition, virtualization is in essence also what one uses to prove NP-completeness results by polynomially mapping one NP decision problem onto another NP decision problem. For instance, in the chapter on autoencoders, we showed that clustering on a 2D lattice with respect to the Manhattan distance can be "virtualized" to clustering on the hypercube with respect to the Hamming distance. Finally, one may suspect that several different forms of virtualization may be present in the brain. For instance, what is empathy if not the ability to virtualize the feelings of others, and feel the pain of others as if it were your own? This ability alone may have played a fundamental role in the evolution of not only morality, but also intelligence by enabling the transition from cold blooded to warm blooded animals that require far more food, hence far longer periods of learning, hence far longer periods of parental care [187].

But why is virtualization relevant to neural networks and deep learning? The reason is simple. Every time you run a neural network on a standard digital com-

puter, you are in fact running a virtual neural network, i.e. a fantasy of neurons and synapses stored in digital, Turing-tape-style, format. While virtualizations are useful, they can also lead to misleading illusions if one starts to believe in them, as if they were the real thing. An obvious example is the typical desktop interface which gives us the illusion of the existence of folders, of moving files to folders, and so forth. And these illusions can sometimes become hard to detect (see, for instance, Donald Hoffman's analogy between our perceptual systems and the Windows desktop interface in light of evolution[317]).

In the case of neural networks, the virtualization aspect is routinely forgotten; but recognizing it is going to enable us to answer the key questions mentioned above. To recognize it, you must use your imagination and do like Einstein who imagined how the world would look like if he were a ray of light. In the case of neural networks, this is what can be called "learning in the machine", as opposed to machine learning, i.e. trying to imagine how things work in a native, physical, neural system, as opposed to a simulated one. You must imagine how the world would look like if you were a neuron or, even better, a synapse.

7.2 The Neuronal View

Putting oneself in the shoes of a neuron reveals, for instance, why the CONNECTED problem[1] seen in a previous chapter is much harder than what it appears to be on the surface. At first sight, the problem seems easy, or at least much easier than the PARITY problem, as we humans can solve it effortlessly using our visual system. But the point is precisely that a neuron does not have a visual system, just a set of incoming connections and signals that can be permuted. Thus, to solve the problem, a neuron would have to learn the particular permutation associated with the ordering of the vector components, a task that is too hard for a single neuron.

In the same vein, imagining that neurons have a high rate of failure could lead immediately to the dropout learning algorithm [604, 91] where, for each training example, neurons are randomly dropped from the training procedure. As a side note, in standard dropout, neurons are unrealistically assumed to function perfectly at production time.

Another example of learning-in-the-machine thinking applied at the neuronal level leads one to think about patterns of connectivity between neuronal layers that are local, as opposed to full, as well as receptive fields that are not perfectly rectangular and perfectly arranged on a square lattice, as is routinely done in convolutional

[1] Given a binary input vector, determine whether the 0's are all adjacent to each other (with or without wrap around).

neural networks and computer vision applications.

In fact, the whole notion of exact weight sharing at the center of convolutional architectures may not be realistic for certain machines, including the brain. Thus one may want to consider convolutional architectures with a relaxed form of weight sharing, where neurons may have similar but not identical receptive fields, and similar but not identical connections at the beginning of learning [493]. An argument can be made that as long as the system is presented with video data, i.e. essentially with images of identical objects translated at different positions, then the approximate weight sharing should be preserved throughout learning by stochastic gradient descent. Indeed the corresponding connections of two neurons located in the same array would be receiving similar initialization at the beginning of learning, up to small random noise, and evolve along parallel paths during learning as the corresponding neurons would see roughly the same stimuli and the same backpropagated errors at almost the same times.

These are examples of insights that can be gained from thinking about the physical neural level. However, for the purpose of understanding learning and answering the questions posed at the beginning of this chapter, it is even more productive to imagine the world from the point of view of a physical synapse, which yields the local learning principle.

7.3 The Synaptic View: the Local Learning Principle

As we have seen in the Introduction, if we resize a synapse by a factor of 10^6, it becomes the size of a fist. And if the synapse is in Los Angeles, the tennis racket it must learn to control is in San Francisco. This highlights the fundamental problem of deep learning and leads to the local learning principle.

Local Learning Principle: *In a physical neural system, the learning rule of a synapse can only depend on quantities available locally at the synapse, in both space and time.*

The local learning principle can easily be stated in layman terms and may seem obvious, but it has significant technical consequences. A simple analogy, with all the necessary caveats to keep things in proportion, is the relativity principle, originally introduced by Galilei, which states that the laws of physics are the same in all frames of reference. Again this statement is simple and understandable by a layman. However, when applied to inertial frames together with the invariance of the speed of light in vacuum, it leads immediately to the Lorentz transformation

and special relativity. When applied to non-inertial frames, allowing for rotations and accelerations, it leads to general relativity.

To study the consequences of this principle, in this Chapter, we focus primarily on locality in space (see Appendix in [64]). Locality in time is sparsely covered in the next two chapters. However, it should be obvious that even for the backpropagation algorithm, issues of what is local in time come up immediately: for instance, the backward pass must remember the derivatives associated with the outputs of each layer during the forward pass. More broadly, for a deep weight w_{ij}^h the presynaptic output O_j^{h-1} and the postsynaptic backpropagated error B_i^h must be close in time, where the definition of closeness depends on the kind of physical neural system.

To begin with, it is worth to ask which known learning rules are local in space. The space of learning rules found in the literature is rather sparse. In fact most familiar rules, including for instance the perceptron learning rule [545], the delta learning rule [679], and Oja's rule [487], can be viewed as special cases of, or variations on, backpropagation or simple Hebbian learning (Table 7.1).

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 7.1: 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 locally available output target.

To understand which rules are local, we must provide a precise definition of locality. In a model, one is of course free to decide which variables are local. Considering first locality in space, using the formalism of this book, it is reasonable to assume that the local variables of a deep synapse with strength w_{ij}^h are the pre- and post-synaptic outputs (or activations), as well as the synaptic strength itself. Thus typically, in the present formalism, a local learning rule for a deep layer of a feedforward architecture should have the functional form:

$$\Delta w_{ij}^h = F(O_i^h, O_j^{h-1}, w_{ij}^h) \tag{7.1}$$

where O_i^h and O_j^{h-1} represent the outputs of the post- and pre-synaptic neurons respectively. Assuming that the targets are local variables at the output layer, local learning rules for the output layer L must have the functional form:

$$\Delta w_{ij}^L = F(T_i, O_i^L, O_j^{L-1}, w_{ij}^L) \tag{7.2}$$

Under these definitions of spatial locality, we see that the simple Hebb rule, Oja's rule, the perceptron rule, and the Delta rule are local learning rules. It should also be obvious that in any $A(n, m)$ architecture with local learning, all the units learn independently of each other. However the backpropagation rule is *not local*.

This alone reveals one of the main reasons for the lack of clarity in Hebb's original formulation, with its implicit focus on the neuronal level ("neurons that fire together, wire together") rather than the synaptic level. Hebb implicitly combined two different concepts that ought to be clearly separated: (1) the locality of synaptic learning rules; and (2) the functional form of synaptic learning rules (e.g. quadratic form). Among other things, separating these two concepts, immediately suggest many possible different learning rules, and ways to stratify them by their complexity. It is, for instance, reasonable to consider learning rules that are polynomial in the local variables, and stratify these rules by their degree [64, 103].

7.4 Stratification of Learning Rules

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 next consider the functional form of F. Here we consider only the case where F is a polynomial function of degree n (e.g. linear, quadratic, cubic) in the local variables, although other functional forms can easily be considered within this framework. Indeed, 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; 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 d of w may be higher because O_i depends also on w_{ij}, typically in a linear way at least around the current value. 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 $\Delta w_{ij} = \eta w_{ij} O_i^2 I_j$, 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.

As we have seen, one of the best ways to understand the behavior of learning rules is to convert them to difference, or differential, equations by taking averages over the training set and assuming the learning rate is small. Because we are restricting ourselves to learning rules with a polynomial form, the initial goal is thus 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. Estimation of these terms can be carried out, especially in the linear case [103] where, by averaging over the training set, one can derive a differential equation for the learning dynamics (see exercises).

In the linear case, to understand the behavior of any local learning rule, one must compute expectations over the training data of the form

$$E(T^{n_T} O^{n_O} I_i^{n_{I_i}} w_i^{n_i}) = w_i^{n_i} E\left[T^{n_T} (\sum_k w_k I_k)^{n_O} I_i^{n_{I_i}}\right] \tag{7.3}$$

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 differential equation associated with the learning dynamics 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. Systematic examples can be found in the exercises at the end of this chapter and in [103].

So far, the principle of local learning has allowed us to precisely define and stratify the space of learning rules. The study of local learning rules in single units is interesting per se, and some rules may lead to useful statistical functions, such as PCA. However, this can only go so far, because the use of local learning rules in feedforward networks faces fundamental limitations.

7.5 Deep Local Learning and its Fundamental Limitations

Here we consider deep layered feedforward networks, where local learning is applied layer-by-layer, starting from the layer closest to the inputs (asynchronous local learning would not change much of the main conclusions). We call this form of learning, which stacks local learning rules in a deep architecture, deep local learning (Figure 7.1). As we have seen in the introduction, deep local learning is what had been proposed for instance by Fukushima [266] to train the neocognitron architecture using Hebbian learning.

In principle such networks can be analyzed since they reduce to the study done above, and in the exercises, of local learning in single units. This is because within any single layer of units, all the units learn independently of each other given the inputs provided by the previous layer. Deep local learning may be able to perform simple statistical operations on the data (e.g. PCA) and learn simple functions. However, in deep local learning, information about the targets is not propagated to the deep layers and therefore in general deep local learning cannot learn complex functions.

To see this more precisely, consider deep local learning in a deep layered feedforward architecture (Figure 7.1) $A(n_0, n_1, \ldots, n_L)$ with $L+1$ layers of size $n_0, n_1, \ldots n_L$ where here 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 transfer functions can be arbitrary, as long as they are continuous and differentiable (except possibly for a finite number of points pf non-differentiability, e.g. ReLU transfer functions). It is also possible to extend the analysis to threshold functions 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 $(I(k), T(k))$ for $k = 1, \ldots, K$ and the goal is to minimize a differentiable error function $\mathcal{E}(w)$. The main learning constraint is that we can only use local learning rules in each adaptive layer (Figure 7.1).

Fundamental Limitation of Deep Local Learning: *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 function where the gradient is zero. Thus, in particular, it cannot find locally or globally optimal weights.*

Proof: If we consider any weight w_{ij}^h in a deep layer h (i.e. $0 < h < L$), by the backpropagation algorithm we know that:

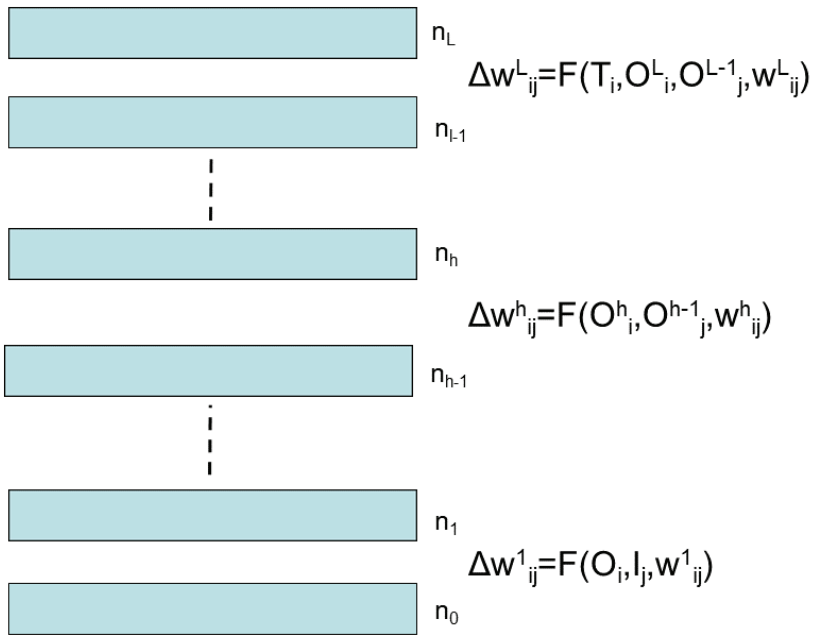

Figure 7.1: Deep Local Learning. In this case, local learning rules, such as the simple Hebb's rule, are applied to each synaptic weight in each adaptive layer of a deep feedforward architecture. For all the hidden units, the local learning rules are unsupervised and thus of the form: $\Delta w_{ij}^h = F(O_i^h, O_j^{h-1}, w_{ij}^h)$. For all the output units, the local learning rules can be supervised since the targets are considered to be local variables for the output layer, and thus can have the form: $\Delta w_{ij}^L = F(T, O_i^L, O_j^{L-1}, w_{ij}^L)$.

$$\frac{\partial \mathcal{E}}{\partial w_{ij}^h} = E\left[B_i^h(t) O_j^{h-1}(t)\right] = \frac{1}{K}\sum_{k=1}^{K} B_i^h(k) O_j^{h-1}(k) \tag{7.4}$$

where $B_i^h(k)$ is the backpropagated error of unit i in layer h, which depends in particular on the targets $T(k)$ and the weights in the layers above layer h. Likewise, $O_j^{h-1}(k)$ is the presynaptic activity of unit j in layer $h-1$ which depends on the inputs $I(k)$ and the weights in the layers below layer $h-1$. In short, the gradient is a large sum over all training examples of product terms, each product term being the product of a target/output-dependent term with an input-dependent term. 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*. In particular, this

must be true at any local or global optimum, as well as any saddle point, including low-error saddle points. However, by 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 such a critical point.

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 local learning rules, including all possible Hebbian rules, are used. For the same reasons, an architecture consisting of a stack of autoencoders trained using unlabeled data only [310, 239] 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 other related techniques to compress data, or to 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 where a majority of the weights may not depend on the targets. These cases are not important in practice, but they are not easy to capture exhaustively with mathematical precision. For example, they include the case when the training set is too small with respect to the architecture (e.g. $K = 1$). In this case, the training set can be loaded trivially into the architecture by using random weights in all the lower layers (so they are independent of the targets) and adjusting just the top layer.

Use of the local learning principle has allowed us to stratify the space of local learning rules. Then it has allowed us to provide an explanation for the Fukishima-Hebb problem and why local learning rules alone cannot do much in a feedforward architecture. Together with the relative optimality of backpropagation seen in the previous chapter, this also explains the sparsity of the space of learning rules as there is little incentive to search for new local earning rules, outside of biological or other hardware specific settings. Next, the local learning principle leads to the deep learning channel.

7.6 Local Deep Learning: the Deep Learning Channel

The fundamental limitation above 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 targets. Thus:

Deep Learning Channel(DLC): *In a physical feedforward neural system, in order to be able to reach a locally optimal configuration, where* $\partial\mathcal{E}/\partial w_{ij}^h = 0$ *for all the weights, there must exist a backward physical channel that conveys information about the targets back to the deep synapses so that they can be updated using a local learning algorithm depending on this information. We call this channel the Deep Learning Channel.*

The necessary existence of the DLC immediately leads to fundamental predictions for biology. First, if the learning frameworks discussed in this book have any relevance to biology, then the prediction is that there must exist backward channels, implemented as chains of neurons and axons, going from peripheral neurons where some notion of target may be available all the way back to almost every single deep neuron in the corresponding circuit. Second, one may think of the signal carried by the DLC as some form of "feedback". However, one must be careful to distinguish different kinds of feedback signals, such as dynamic and learning feedback. These different feedback signals may, or may not (or only partially), use the same physical connections. For instance, in the visual of sensory-motor systems one may consider fast dynamic feedback signals for vision or motor control. These feedback signals are different from the learning feedback signals in the DLC which may include fast electrical components, but also much slower bio-chemical components.

The corresponding DLC-based learning is called local deep learning (Figure 7.2, as opposed to deep local learning which simply stacks local learning rules (Figure 7.1). Backpropagation is a local deep learning algorithm but, as we shall see in the next chapter, it is not the only one. In a physical neural system, deep local learning raises several questions regarding the DLC: (1) what is the physical nature of the backward channel; (2)what is the nature of the information being transmitted through the DLC; and (3) what is the rate at which this information can be transmitted and its maximum (the capacity of the DLC).

The physical nature of the DLC depends on the physical nature of the neural system and, as such, it will not be our focus here. However it should at least be noted that there are a number of different possibilities, such as: (1) the forward and backward channels use different kinds of physical channels (e.g. electrical in

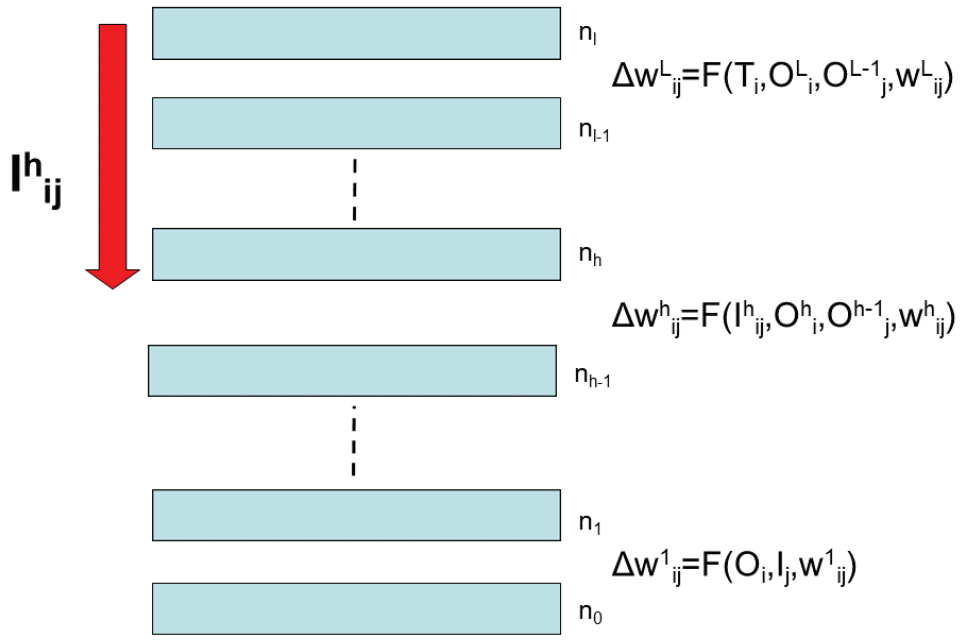

Figure 7.2: Local Deep Learning. In general, deep local learning in a deep feedforward architecture cannot learn complex functions optimally since it leads to architectures where only the weights in the top layer depend on the targets. For optimal learning, some information I^h_{ij} about the targets must be transmitted to each corresponding synapse, so that it becomes a local variable that can be incorporated into the corresponding local learning rule $\Delta w^h_{ij} = F(I^h_{ij}, O^h_i, O^{h-1}_j, w^h_{ij})$.

one direction and optical in the other one); (2) the forward and backward channels use exactly the same physical channel (e.g. wires), but this channel is bidirectional and, at least for backpropagation, it is also symmetric with the same weights in both directions; (3) the forward and backward channels are similar, but physically separate (e.g. different wires) and symmetric, i.e. with the same weights in both directions, as in backpropagation; and (4) the forward and backward channels are similar, but physically separate (e.g. different wires) and non-symmetric, i.e. with different weights in both directions. For biological neural systems, the last case seems the most plausible, as it is hard to imagine having connections with exactly the same synaptic weights in both directions. Finally, although we call it the DLC, in principle there could be on the order of $W \times n_L$ different DLCs, one for each synapse and each output component. In general this is not efficient and some form

of multiplexing is preferable and possible, as exemplified by the backpropagation algorithm. Backpropagation also shows that it can be enough to backpropagate the error signal to the postsynaptic neuron, rather than each one of its impinging synapses.

Next we turn to the question of the nature of the information that must be transmitted through the DLC. This question will also occupy most of the next chapter. In particular, we are interested in identifying which information is absolutely necessary, and which is not. Let us denote by I_{ij}^h the information that needs to be transmitted by the DLC to the synapse w_{ij}^h in order to enable any effective local deep learning of the form (Figure 7.2):

$$\Delta w_{ij}^h = F(I_{ij}^h, O_i^h, O_j^{h-1}, w_{ij}^h) \tag{7.5}$$

The upper and lower indexes on I distinguish it clearly from the inputs associated with the input layer. From equation 7.4, we know that in principle at a critical point I_{ij}^h could depend on the inputs, the outputs, the targets, and all the weights of the entire architecture. However, backpropagations shows that this can be simplified in two ways: (1) all the necessary information about the lower layers of the architecture and its weights is contained in O_j^{h-1}; and (2) the information carried by the DLC can be written as:

$$I_{ij}^h = I_i^h(T, O^L, w_{rs}^l(l > h), f'(l \geq h)) \tag{7.6}$$

In other words, rather than having to send a signal through the DLC to each individual synapse impinging onto neuron i in layer h, it is possible to multiplex the signals and, in particular, send a single error signal to neuron i. Neuron i can then broadcast it to all its impinging synapses. This broadcasting operation will not concern us here: it is essentially invisible when we think about the backpropagation algorithm; however, in a biological neuron it requires a major undertaking. In addition, backpropagation, shows that one additional level of simplification is possible: the dependence on both T and O^L can be mediated by the single error term $T - O^L = -\partial\mathcal{E}/\partial S^L$. Therefore:

$$I_{ij}^h = I_i^h(T - O^L, w_{rs}^l(l > h), f'(l \geq h)) \tag{7.7}$$

under the usual assumptions on the error function and the transfer functions for the top layer. Additional simplifications of I_{ij}^h will be examined in the next chapter.

7.7 Local Deep Learning and Deep Targets Equivalence

Since I_{ij}^h need not depend directly on j, we can define the corresponding class of local deep learning algorithms and show a basic equivalence between having post-synaptic backpropagated errors and deep targets that are local.

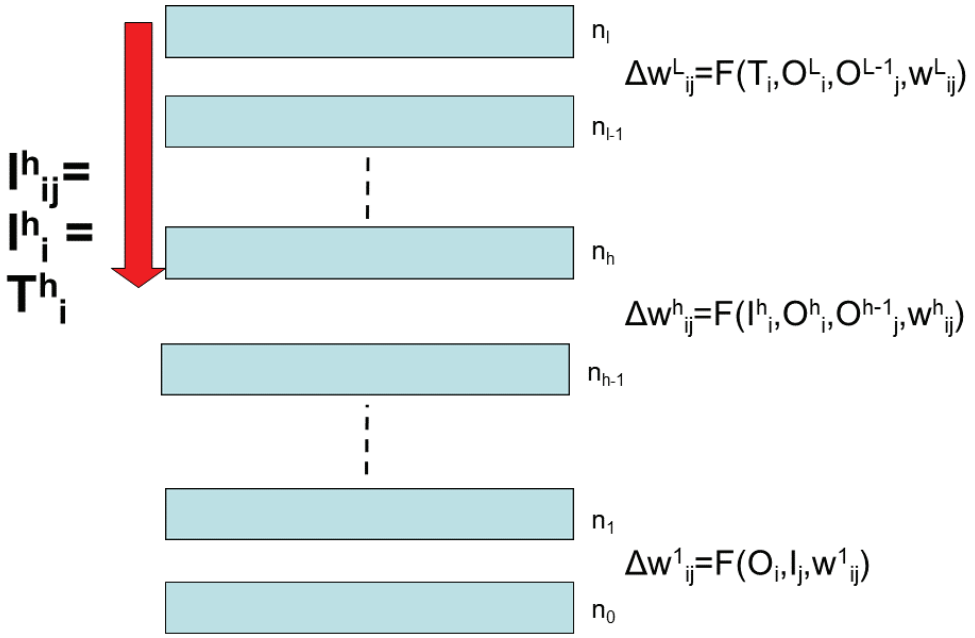

Figure 7.3: Deep Targets Learning. This is a special case of local deep learning, where the transmitted information I_{ij}^h about the targets does not depend on the presynaptic unit $I_{ij}^h = I_i^h$. In this chapter, it is shown that this is equivalent to providing a deep target T_i^h for training the corresponding unit 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 $\Delta w_{ij}^h = \eta(T_i^h - O_i^h)O_j^{h-1}$.

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_{ij}^h transmitted about the targets depends only on the postsynaptic unit, in other words $I_{ij}^h = I_i^h$. Thus in a deep targets learning algorithm we have

$$\Delta w_{ij}^h = F(I_i^h, O_i^h, O_j^{h-1}, w_{ij}^h) \tag{7.8}$$

for some function F (Figure 7.3).

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_i^h is available for unit i in layer h, then we can set $I_i^h = T_i^h$ in Definition 1. Conversely, starting from Definition 1, consider a deep targets algorithm of the form

$$\Delta w_{ij}^h = F(I_i^h, O_i^h, O_j^{h-1}, w_{ij}^h) \tag{7.9}$$

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

$$\Delta w_{ij}^h = \eta(T_i^h - O_j^h)O_j^{h-1} \tag{7.10}$$

This is true both for sigmoidal transfer functions and for threshold gates, otherwise the rule should be modified to accommodate other transfer functions accordingly. By combining Equations 7.9 and 7.10, we can solve for the target

$$T_i^h = \frac{F(I_i^h, O_i^h, O_j^{h-1}, w_{ij}^h)}{\eta O_j^{h-1}} + O_i^h \tag{7.11}$$

assuming the presynaptic activity $O_j^{h-1} \neq 0$ (note that $T^L = T$) (Figure 7.3). As seen in the previous chapter, as a special case, backpropagation can be viewed as a

deep targets algorithm providing targets for the hidden layers according to Equation 7.11 in the form:

$$T_i^h = I_i^h + O_i^h \tag{7.12}$$

where $I_i^h = B_i^h = -\partial\mathcal{E}/\partial S_i^h$ is the backpropagated error.

In summary, we started this chapter by introducing the notion of local learning, noticing that Hebbian learning is a special case of local learning. This notion allows us to get a clear view of the space of learning rules, and its stratification by complexity. Backpropagation is not a local learning rule, and therefore it is not Hebbian in this simple sense. By stacking local learning rules in a feedforward architecture, one obtains deep local learning. Deep local learning however is unable to learn complex functions. In order to learn complex functions, local deep learning is necessary. Local deep learning requires sending information about the targets back to the deep synapses through a channel, called the DLC. With the introduction of a DLC, backpropagation is a deep local learning algorithm, and in that sense the resulting learning rule is local. In the next chapter, we take a closer look at the information that must be sent over the DLC and at alternative local deep learning algorithms that do not require symmetric weights in the forward and backward directions.

7.8 Exercises

7.1 Compile a list, as exhaustive as possible, of examples of the concept of virtualization.

7.2 Suppose you have two Windows computers A and B connected to the Internet. You can run the Windows Remote Desktop application on computer B to view the computer screen of A, as if you were sitting in front of computer A. Which command(s) should you type on computer B in order to execute the command 'Ctrl-Alt-Del" on computer A?

7.3 (1) List all possible terms of degree $d = 1$ of a local polynomial learning rule, in both the supervised and unsupervised cases. (2) Compute their expectations. (3) Solve the corresponding differential equations in the linear case. (4) Study non-linear cases, analytically whenever possible, or through simulations.

7.4 (1) List all possible terms of degree $d = 2$ of a local polynomial learning rule, in both the supervised and unsupervised cases. (2) Compute their expectations. (3)

Solve the corresponding differential equations in the linear case. (4) Study non-linear cases, analytically whenever possible, or through simulations.

7.5 (1) List all possible terms of degree $d = 3$ of a local polynomial learning rule, in both the supervised and unsupervised cases. (2) Compute their expectations. (3) Solve the corresponding differential equations in the linear case. (4) Study non-linear cases, analytically whenever possible, or through simulations.

7.6 Learning rules often lead to synaptic weights that diverge over time to infinity. Find at least two different ways for creating polynomial learning rules of arbitrary degree that will keep the synaptic weights bounded.

7.7 Study the simple Hebbian learning rule: $\Delta w_i = \eta O I_i$ with $n = 2$ and $d = 1$ analytically, or by simulations, in both linear and non-linear (threshold gate, sigmoidal, ReLU) neurons. Study the supervised version of the rule, where O is replaced by T.

7.8 Study the simple anti-Hebbian learning rule: $\Delta w_i = -\eta O I_i$ with $n = 2$ and $d = 1$ analytically, or by simulations, in both linear and non-linear (threshold gate, sigmoidal, ReLU) neurons. Study the supervised version of the rule, where O is replaced by T.

7.9 Study the learning rule: $\Delta w_i = \eta(1 - w_i^2) I_i$ with $n = 3$ and $d = 2$ analytically, or by simulations, in both linear and non-linear (threshold gate, sigmoidal, ReLU) neurons.

7.10 Study the gradient descent learning rule: $\Delta w_i = \eta(T - O) I_i$ with $n = 2$ and $d = 1$ analytically, or by simulations, in both linear and non-linear (threshold gate, sigmoidal, ReLU) neurons.

7.11 Study Oja's learning rule: $\Delta w_i = \eta(O I_i - O^2 w_i)$ with $n = 3$ and $d = 3$ analytically, or by simulations, in both linear and non-linear (threshold gate, sigmoidal, ReLU) neurons.

7.12 Could batch or layer normalization be easily implemented in a physical neural system?

7.13 If the DLC theory can be applied to biological neural systems, what is its main prediction? Can you state the prediction in terms of the analogy made in the

Introduction where, by rescaling lengths by a factor of one million, a synapse is the size of a fist? Does such a prediction violate any known facts about the anatomy of the brain? Which assumptions in the DLC theory are most questionable in relation to biology?

7.14
1) Recall the analogy made in the Introduction where, by rescaling lengths by a factor of one million, a synapse is the size of a fist. In this rescaling, what would correspond to a neuron? Use this to provide a corresponding analogy for deep learning algorithms, where $I^h_{ij} = I^h_i$, for instance by equating the function of the DLC to mail delivery.
2) Likewise what would correspond to an entire brain under this rescaling by a factor of 10^6. Assuming there are on the order of 10^{14} synapses in the brain, explain the fundamental role of the locality principle within this rescaled model.

7.15 Consider a multilayer, feedforward architectures $A(n_1, \ldots, n_L)$ and a training set of input-output target pairs of the form (I, T). Consider two learning rules: backpropagation (local deep learning) and the simple Hebb rule (deep local learning). Using a standard training set (e.g. MNIST or CIFAR) plot in each case the trajectories of the hidden layers in the information plane. What are the main differences?