Learning in the Machine: Recirculation is Random Backpropagation

P. Baldi and P. Sadowski

October 3, 2018

Abstract

Learning in physical neural systems must rely on learning rules that are local in both space and time. Optimal learning in deep neural architectures requires that non-local information be available to the deep synapses. Thus, in general, optimal learning in physical neural systems requires the presence of a deep learning channel to communicate non-local information to deep synapses, in a direction opposite to the forward propagation of the activities. Theoretical arguments suggest that for circular autoencoders, an important class of neural architectures where the output layer is identical to the input layer, alternative algorithms may exist that enable local learning without the need for additional learning channels, by using the forward activation channel as the deep learning channel. Here we systematically identify, classify, and study several such local learning algorithms, based on the general idea of recirculating information from the output layer to the hidden layers. We show through simulations and mathematical derivations that these algorithms are robust and converge to critical points of the global error function. In most cases, we show that these recirculation algorithms are very similar to an adaptive form of random backpropagation, where each hidden layer receives a linearly transformed, slowly-varying, version of the output error.

1 Introduction

Autoencoders play a central role in the theory and practice of neural networks, for instance by providing a bridge between supervised and unsupervised learning. Autoencoders can be trained by backpropagation using the input data as the target. While this approach is reasonable in a digital simulation, it faces serious challenges in a physical neural autoencoder due to the non-locality of the backpropagation algorithm. In a physical neural system, such as a brain or a neuromorphic chip, all learning rules must be local in both space and time, i.e. must depend only on variables that are available locally at, or near, each synapse. Thus, in a physical neural autoencoder, a deep learning channel must exist capable of communicating non-local spatio-temporal information, such as errors, to deep synapses [1].

In this work, we study various embodiments of deep learning channels for autoencoder architectures and the information they ought to carry, as well as the corresponding learning rules. In particular, the theory of local learning predicts the existence of algorithms for training autoencoders, alternative to backpropagation, in which non-local error information is provided to the deep weights by recirculating the autoencoder output back through the autoencoder itself. In these algorithms, differences of activation in time are used to replace backpropagated errors. Thus, in recirculation algorithms, the forward weights have a dual role as they embody both the forward channel and the learning channel. The recirculation process can be implemented in several ways which we study both through mathematical and simulation analyses.

The mathematical and simulation analyses reveal robustness across these variations and close connections between the class of recirculation learning algorithms and random backpropagation [2, 3, 4]. As in random backpropagation, the recirculation learning channel provides a linearly transformed version of the error information to the deep weights. And as in adaptive random backpropagation, this linear transformation evolves during learning.

This study falls under the broader theme of "learning in the machine", in contrast to machine learning, focused on studying the effects of physical constraints on physical neural systems, in contrast to digitally simulated neural systems. The key advantage of recirculation algorithms is that they are local. This locality property means that they could be used to train physical neural systems, for example in fast, power-efficient neuromorphic chips [5], and might provide a useful model of learning for biological neural systems. Recirculation algorithms are not meant to be practical in the sense of superseding backpropagation on digital computers and standard deep learning applications.

In Section 2 we first review autoencoders, backpropagation, and random backpropagation and formalize the notion of a circular autoencoder in which the output layer is identical or physically close to the input. In Section 3, we briefly review the theory of local learning and the problems it identifies with backpropagation applied to autoencoders in physical neural networks. This naturally leads in Section 4 to the definition of various recirculation algorithms, depending in particular on whether errors or output are being recirculated and how activities are combined across different cycles. Mathematical analyses of error and output recirculation, including connections to random backpropagation, are provided in Sections 5 and 6. Simulations further corroborating our results are reported in Section 7, followed by a short Discussion. Finally, recirculation algorithms for non-circular autoencoders are studied in the Appendix.

2 Autoencoders and Backpropagation

In a supervised learning framework for neural networks, data typically comes in the form of input/output-target pairs of the form $(I(k), T(k))$ for $k = 1, \ldots, K$. In the special case of autoencoders, the targets are equal to the inputs: $T(k) = I(k)$ for every k. We also assume that the learning task for the autoencoder is

to minimize a standard error function $\mathcal{E}$, typically square error loss in the case of real valued data, and sum of relative entropies in the case of binary data.

2.1 Standard Autoencoders

We begin by considering a Standard Autoeconder architecture $\mathcal{A}[N_0, N_1, N_2]$ with N_0 inputs in layer $l = 0$, N_1 hidden units in hidden layer $l = 1$, and $N_2 = N_0$ output units in output layer $l = 2$ (Figure 1). All the concepts presented here can be immediately extended to autoencoders architectures with additional hidden layers. We assume that there is an $N_0 \times N_1$ matrix of weights A connecting the hidden layer to the output layer, and an $N_1 \times N_0$ matrix of weight B connecting the input layer to the hidden layer (Figure 1). A unit i in layer $l = 1$ has a total input S_i^1 and produces an output O_i^1 given by:

$$O_i^1 = f(S_i^1) = f(\sum_j b_{ij} O_j^0) \tag{1}$$

or, in matrix notation,

$$O^1 = F \circ BO^0 = FBO^0 \tag{2}$$

where O^0 is the vector of activation in the input layer, initially equal to a training example ($O^0 = I$), and $F \circ BO^0$ denotes that the non linear function f is applied to each component of the vector BO^0 (it is also possible to have a different non-linear function f_i^1 for each neuron i in layer 1). Since there is no risk of confusion, we omit the symbol "$\circ$" in what follows. Likewise, a unit i in layer $l = 2$ has a total input S_i^2 and produces an output O_i^2 given by:

$$O_i^1 = g(S_i^2) = g(\sum_j a_{ij} O_j^1) \tag{3}$$

or, in matrix notation,

$$O^2 = G \circ AO^1 = GAO^1 = GAFBI \tag{4}$$

where G represents the non-linear functions of the output layer, with the same remarks as for the hidden layer.

We assume standard transfer functions where f (or g) is the identity ($f = Id$) if the corresponding unit is linear, or f is typically sigmoidal (e.g. logistic or hyperbolic tangent) or rectified linear in the case of non-linear units. When $A = B^t$ (where B^t denotes the transpose of the matrix B) we say that the standard autoencoder is symmetric. This symmetry relation can hold at initialization time only or, more strongly, at all times. When the term symmetric is used without any other qualifications, we will take it to mean symmetric at all times. The symmetric standard autoencoder architecture corresponds to a standard autoencoder architecture where $A = B^t$ at all times. Obviously in a physical implementation of a symmetric standard autoencoder, one must consider the computational and physical mechanisms that ensure the symmetry.

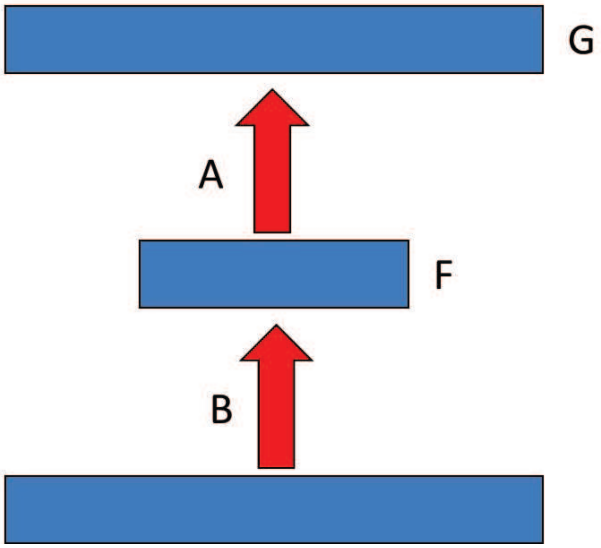

Figure 1: Standard autoencoder architecture with weight matrices A and B and non-linear transformations F and G.

2.2 Circular Autoencoders

For reasons that will become obvious in the coming sections, we are particularly interested in the study of circular autoencoders. In a physical implementation, in addition to the standard autoencoder architecture, it is also possible to consider a circular autoencoder architecture $\mathcal{A}^*[N_0, N_1, N_2, \ldots, N_{L-1}, N_L]$ with $N_0 = N_L$, shown in Figure 2, with multiple hidden layers numbered $0, 1, \ldots, L$. In a circular architecture, the output units are identical (or spatially very close) to the input units, thus here $0 = L$ corresponds to the input/output layer. We assume that there is a matrix of weights A^h connecting layer $h-1$ to layer h for $h = 1, \ldots, L$ (note: while we use $A^1, \ldots, A^L$ with multiple hidden layers, when only a single hidden layer is present we use A and B instead to improve readability). In addition, in the non-linear case, there is a vector of non-linear functions F^h associated with each layer h, $h = 1, \ldots, L$. Thus, with the same notation as above, we have:

$$O^h(t) = F^h A^h O^{h-1}(t) \tag{5}$$

for $h = 1, \ldots, L$ and $O^0(0) = I$. The time index t here is used to denote the cycle, with $t = 0$ corresponding to the activations generated by the original input I. In the coming sections, $t = 1$ will be used to index the first cycle of recirculation, and so forth.

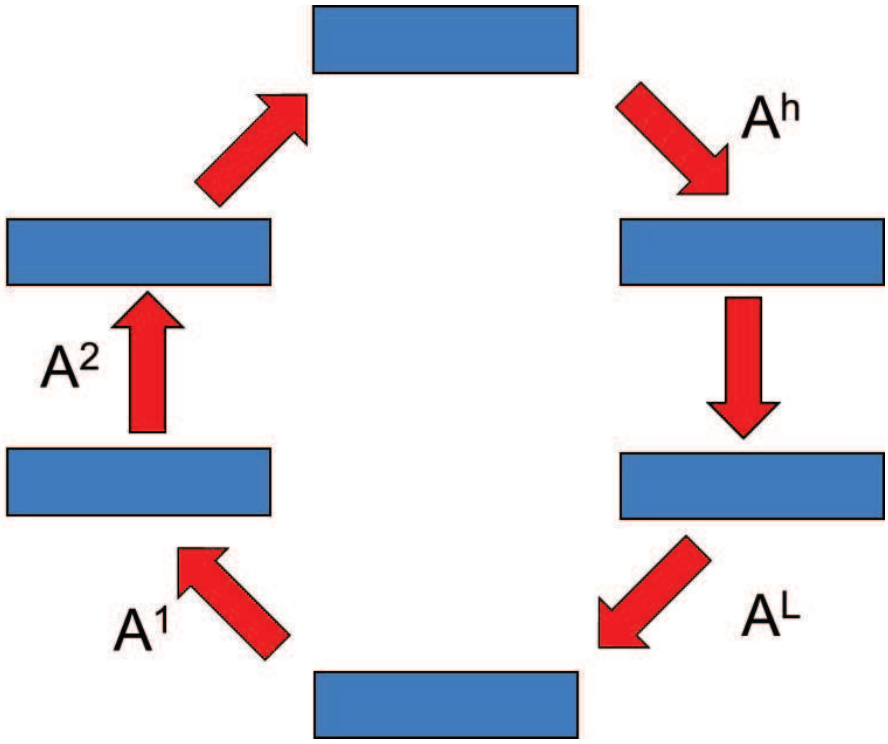

Figure 2: Circular autoencoder with multiple hidden layers and connection matrices $A^1, \ldots, A^L$. In the non-linear case, non-linear transformations $F^1, \ldots, F^L$ are associated with each layer.

2.3 Backpropagation

2.3.1 Standard Backpropagation

Backpropagation has been the most successful algorithm for training neural networks. In standard backpropagation in a layered feedforward network with L layers, the learning rule for a deep weight w_{ij}^h connecting neuron j in layer $h-1$ to neuron i in layer h is given by:

$$\Delta w_{ij}^h = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}^h} = \eta (BP)_i^h O_j^{h-1} \tag{6}$$

where O_j^{h-1} is the presynaptic activity of neuron j in layer $h-1$, $(BP)_i^h$ is the postsynaptic backpropagated error, which is computed recursively by propagating the error $T - O^L$ from the top layer to the bottom layer using the chain rule, and T is the target. These equations can be applied in batch mode (gradient descent), or online (stochastic gradient descent). Using the chain rule, one sees immediately that the backpropagated error satisfies the recurrence relation:

$$(BP)_i^h = \frac{\partial \mathcal{E}}{\partial S_i^h} = (f_i^h)' \sum_k (BP)_k^{h+1} w_{ki}^{h+1} \tag{7}$$

with the typical boundary condition:

$$(BP)_i^L = \frac{\partial \mathcal{E}_i}{\partial S_i^L} = T_i - O_i^L \tag{8}$$

This is the boundary condition for the three most common types of neural network outputs and error functions: (1) linear outputs with squared error for regression; (2) logistic outputs with relative entropy error for binary classification; and (3) softmax outputs with relative entropy error for classification into k classes ($k > 2$).

Thus, in short, the errors are propagated backwards in an essentially linear fashion using the transpose of the forward matrices, with a multiplication by the derivative of the corresponding forward activations every time a layer is traversed. *It is essential to note that at any critical point where the gradient is zero, the deep weights will depend on many quantities, such as the output error which, in a physical feed-forward network, are not available locally at each deep synapse. Thus, in a physical feed-forward neural system, for any kind of optimal learning (i.e. any learning capable of reaching critical points) there must exist physical channels capable of transmitting non-local information to the deep synapses (see Section 3 on Local Learning).*

2.3.2 Random Backpropagation

An important result, supported by simulations and theoretical analyses [2, 3, 6, 4], is that the weight matrices in the backpropagation process do not need to be the transpose of the forward matrices. In fact they can be chosen randomly and most of the time the network will be able to learn almost as well as if the transpose matrices had been used. Furthermore this random backpropagation process is robust in the sense that significant variations can be incorporated into the backward channel without interfering with the learning process, including: (1) sparse random matrices; (2) skip connections that directly connect the output layer to the deep layers; (3) non-linear operations; and (4) adaptation, such that the matrices in the learning channel change slowly during the training. It is best if the random matrices are full rank, as determined by the corresponding layers, in order to transmit the maximum amount of information about the error to the deep layers. However even that is not absolutely mandatory in the sense that graceful, as opposed to catastrophic, degradation is observed in the case of non-maximal ranks. Finally, as in all the simulations in [3, 4] the matrices should not contain any bias terms.

Thus, in short, what seems to be important are three things [3, 4]. First, gradient descent should be used in the top layer (i.e. no random matrices for the top layer). Second for learning in the deep layers, the deep layers should receive some information of the form $f(T - O)$ where the function f can be random

as long as it is smooth (e.g. continuously differentiable almost everywhere) and with no bias ($f(0) = 0$) so that locally, to a first-order approximation, f corresponds to a matrix multiplication that preserves information about $T - O$ (high-rank matrix). In addition, it is possible for each f to change slowly during learning. And thirdly, when updating the weights of a given layer, it is important to include the derivatives of the activations of the corresponding layer—all derivatives in the layers above can be ignored.

While random backpropagation and its variants are studied in [2, 3, 4], these references do not study random backpropagation in autoencoders or any recirculation algorithms.

2.3.3 Standard Backpropagation in the Standard Autoencoder

When the backpropagation equations are applied to the standard autoencoder, one immediately gets:

$$\Delta a_{ij} = \eta(T_i - O_i^2)O_j^1 \tag{9}$$

and :

$$\Delta b_{ij} = \eta[\sum_k (T_k - O_k^2)a_{ki}]f'(S_i^1)I_j \tag{10}$$

where here f' is the derivative of forward transfer function f. In vector-matrix form:

$$\Delta A = \eta(I - O^2)(O^1)^t \tag{11}$$

and:

$$\Delta B = \eta F' A^t (I - O^2)(I^t) \tag{12}$$

where M^t denotes the transpose of the matrix (or vector) M, F denotes the vector of derivatives f' at each hidden neuron, and η is the learning rate. By default, all vectors are column vectors.

3 Local Learning and Locality Problems

3.1 Local Learning

A physical neural system, such as a brain or a neuromorphic chip, must obey the laws of physics and in particular any learning rule for adjusting synaptic weights must be *local* in both space and time, i.e. it must depend on variables that are available locally at the synapse in both space and time [1]. With the formalism used here, a reasonable notion of spatial locality is to require that a synaptic learning rule depend only on the pre- and post-synaptic activities of its neurons, and the synaptic weight itself:

$$\Delta w_{ij}^h = \mathcal{F}(O_i, O_j, w_{ij}) \tag{13}$$

for some function $\mathcal{F}$. To extend the concept to include locality in time, one can write:

$$\Delta w_{ij}^h(t) = \mathcal{F}\left(O_i(s), O_j(s), w_{ij}(s)\right) \quad \text{for all} \quad t-u \leq s \leq t \tag{14}$$

where u is the window of time over which spatially local information can be integrated. One of the key ideas behind the recirculation algorithms to be described is to tradeoff space for time, using differences in the activations of the same units at different times to drive the learning. Thus we will be looking for learning algorithms that typically are polynomial in the local variables, in fact the product of a presynaptic and a postsynaptic term, and where the postsynaptic term may be a difference between two activations taken over a relatively short time window. Thus typically during recirculation we will consider rules of the form:

$$\Delta w_{ij}^h(t) = \mathcal{F}\left(O_i(s) - O_i(t), O_j(s), w_{ij}(s)\right) \quad \text{for} \quad t-u \leq s \leq t \tag{15}$$

These learning rules rely on the product of the presynaptic activity times some measure of change in the postsynaptic activity, used to communicate error information. Although in this work we are not using spiking neurons, such learning rules are closely related to the concept of spike time dependent synaptic plasticity (STDP). STDP Hebbian or anti-Hebbian learning rules have been proposed using the temporal derivative of the activity of the postsynaptic neuron [7] to encode error derivatives.

$$\Delta w_{ij} = \eta(\Delta O_i^{post})(O_j^{pre}) \tag{16}$$

with a negative sign in the anti-Hebbian case. In a layered architecture, for a deep weight w_{ij}^h we can write:

$$\Delta w_{ij}^h = \eta(\Delta O_i^h)(O_j^{h-1}) \tag{17}$$

3.2 Locality in the Standard Autoencoder

The fundamental starting point for this work is the observation that in the standard autoencoder, the backpropagation equations (Equations 9-12) violate the locality principle in three major ways:

1. **Locality of Targets:** In order to learn A, the error $(I - O^2)$ must be computed and I is not a local variable at the output layer of the standard feedforward autoencoder (Equation 11).

2. **Locality of Outputs or Errors:** In order to learn B, O^2 (or $I - O^2$) must be a local variable for the hidden layer of the standard feedforward autoencoder, and this is not the case (Equation 12).

3. **Locality of Weights:** In order to learn B, the synaptic weights associated with A^t must be local variables for the synapses connecting the input layer to the hidden layer of the standard feedforward autoencoder, and this is not the case (Equation 12).

These problems are magnified for autoencoders with several hidden layers.

We know from the study of random backpropagation [2, 3, 4] that the locality of the weights may not be a crucial problem. [Alternatively, one could imagine a mechanism that couples the weights in A and B and forces them to be symmetric, or approximately symmetric, at the beginning of learning, or throughout learning $A \approx B^t$. For instance, in the linear compressive case, it is known that an optimal solution satisfying $A = B^t$ exists [8]. However the other two problems are crucial and significantly more complex in a standard autoencoder versus a circular autoencoder. In a circular autoencoder, the problem of the Locality of Targets vanishes, because the output is spatially close or identical to the target, which is equal to the input. Thus the main problem in a circular autoencoder is the locality of the outputs or the errors. One physically plausible way to provide information about the outputs or the errors to the deep layers is by recirculating this information through the circular autoencoder.

3.3 Locality in the Circular Autoencoder

Remarkably, in the circular autoencoder two of the three locality problems vanish entirely: the Locality of Targets and the Locality of Outputs. Thus, in principle, no additional deep learning channel is needed. First, because the output layer is equal to the input layer, it is possible to locally compute the error, if the error is needed. Second this error, or the output of the last layer, can be communicated to the other layers by recirculating the corresponding information in the network using the existing connections of the circular autoencoder. In this way, necessary information about both inputs and targets becomes locally available at each deep synapse. *Thus, by recirculating information from the output of the circular autoencoder, one can predict that learning algorithms ought to exist that are both entirely local in space and time and enable global learning from the training data (i.e. convergence to critical points of the global error function). Furthermore, this recirculation process enables the communication of an error signal $f(I-O^L)$ at each deep layer that is a function of the error $I-O^L$ observed at the output layer, as in the the case of random backpropagation. Therefore the corresponding learning rules of these algorithms can take the same general form as the learning rules of BP/RPB in terms of a product of a presynaptic activity and a postsynaptic "error" term.*

4 Recirculation Algorithms in the Circular Autoencoder

Within the general concept of recirculating information in the circular autoencoder there are a number of variants depending on: (1) the nature of the information that is being recirculated; (2) how the neural activity in one cycle combines with the neural activity in the previous cycle(s); and (3) the specific local learning rule used to update the weights.

4.1 Recirculated Information

In terms of which kind of information is being recirculated, we will consider two different cases, although they turn out to be similar, at least in the case where the biases are set to 0.

1. **Recirculation of Errors:** where the error $I - O^L$ is computed in the input layer and then recirculated to the other layers.

2. **Recirculation of Outputs:** where the output O^L is recirculated to the other layers. While we will focus primarily on the first two cycles indexed by $t = 0$ and $t = 1$, it is also possible to recirculate output information over multiple cycles.

When the error $I - O^L$ is computed in the circular autoencoder and then recirculated, each deep layer receives some function $f(I - O^L)$ of the error, and this is exactly the situation one has in random backpropagation, provided there are no biases in the units to ensure that $f = 0$ when there is no error. The main difference is that f is computed in the forward direction of the architecture, rather than the reverse one, but this turns out to be a minor difference. When the outputs are recirculated, and this is perhaps the most important and interesting case, the difference in activity in time between two different cycles, is used to compute again some function $f(I - O^L)$ of the error for each layer and each unit. In this case, units can have biases since the effect of biases may cancel when the difference is taken. Note that in both cases of recirculation, for each layer the function f slowly varies in time as the learning progresses, which corresponds to an adaptive variant of random backpropagation (ARBP) [3, 4].

4.2 Combination of Neural Activity Across Cycles

When information is recirculated, one must define how the activity resulting from the recirculation process combines with any previous activity. This is most relevant for the recirculation of outputs. Here we will consider up to four main cases, described below for the circular autoencoder but the same ideas apply to the standard autoencoder. In the tables below, for simplicity, we show the activity for layers 0, 1, and 2 only, but the same idea extends to all the layers. These cases contain and generalize the single case considered in [9]. As we shall see in the mathematical derivations and simulations, all these cases turn out

to be fairly similar and there is nothing fundamental about any one of them, which is satisfactory since each one of the cases corresponds to different low-level assumptions about how the underlying hardware, and its time constants, operate.

4.2.1 Plain Recirculation (PR)

In this case, recirculation is treated as a fast process with no memory: there is no combination of activities.

	t=0	**t=1**
$O^2(t)$	$F^2A^2F^1A^1I$	$F^2A^2F^1A^1O^L(0)$
$O^1(t)$	F^1A^1I	$F^1A^1O^L(0)$
$O^0(t)$	I	$O^0(1) = O^L(0) = F^LA^L \dots F^1A^1I$

Table 1: Plain Recirculation Algorithm (PR).

4.2.2 Convex Combination of Recirculation (CCR)

In this case, recirculation is still a fast process but the units are assumed to have some memory of the original activity so that the original activity and the recirculated activity are combined through a convex combination.

	t=0	**t=1**
$O^2(t)$	$F^2A^2F^1A^1I$	$\lambda F^2A^2F^1A^1I + (1-\lambda)F^2A^2F^1A^1O^L(0)$
$O^1(t)$	F^1A^1I	$\lambda F^1A^1I + (1-\lambda)F^1A^1O^L(0)$
$O^0(t)$	I	$O^0(1) = \lambda I + (1-\lambda)O^L(0)$

Table 2: Convex Combination of Recirculation Algorithm (CCR).

4.2.3 Recirculation of Convex Combination (RCC)

In this case, recirculation is a slower process in the sense that the recirculated activity and the original activity are first combined in a convex manner in layer 0, and then this convex combination is propagated. Note that CCR and RCC are identical for linear autoencoders, where all the units are linear.

	t=0	t=1
$O^2(t)$	$F^2A^2F^1A^1I$	$F^2A^2F^1A^1[\lambda I + (1-\lambda)O^L(0)]$
$O^1(t)$	F^1A^1I	$F^1A^1[\lambda I + (1-\lambda)O^L(0)]$
$O^0(t)$	I	$O^0(1) = \lambda I + (1-\lambda)O^L(0)$

Table 3: Recirculation of Convex Combination (RCC).

4.2.4 Convex Combination of Recirculation of Convex Combination (CCRCC)

This case combines the two previous cases together and assumes both relatively slow recirculation and units with a memory. Here the recirculated activity and the original activity are first combined in a convex manner in layer 0, then this convex combination is propagated, and then the activity of the units equilibrates to a convex combination of the previous activity and the activity resulting from the recirculation process. For simplicity, we will use the same coefficient λ in the two kinds of convex combinations, and across all layers, but the results remain similar if, for instance, two different coefficients λ_1 and λ_2 are used for the two different convex combinations.

	t=0	t=1
$O^2(t)$	$F^2A^2F^1A^1I$	$\lambda F^2A^2F^1A^1I + (1-\lambda)F^2A^2F^1A^1[\lambda I + (1-\lambda)O^L(0)]$
$O^1(t)$	F^1A^1I	$\lambda F^1A^1I + (1-\lambda)F^1A^1[\lambda I + (1-\lambda)O^L(0)]$
$O^0(t)$	I	$O^0(1) = \lambda I + (1-\lambda)O^L(0)$

Table 4: Convex Combination of Recirculation of Convex Combination Algorithm (CCRCC).

Yet another possibility is to have each layer take a combination of its original and current activity, and pass it forward. We call this recirculation with convex combination (RWCC). All these variations correspond to slightly different modalities in the ways the underlying neurons operate and the time scales of how they integrate and communicate information. As we shall see, in the big picture of recirculation, these differences turn out to be relatively minor.

4.3 Local Learning Rules

In general, we will use local learning rules that are typically Hebbian in the form of a product of a presynaptic activity and a postsynaptic term containing error information—as we know from BP/RBP that such rules are both sufficient

and effective. In addition, adopting rules with such form automatically ensures locality in space. On the other hand, locality in time requires that all the terms in the learning rule be computed within a few cycles–usually two, but we will consider also larger intervals—of recirculation. With output recirculation, we will typically use differences in activity between cycle 0 and cycle 1 to derive an error signal. In this way, the same local learning rule can be applied to each layer, and the rule for the weights in the top layer corresponds exactly to gradient descent.

To the best of our knowledge, the term recirculation was introduced in [9] together with one recirculation algorithm. However, this work had several limitations: (1) no theoretical motivation was provided for the necessary existence of such an algorithm; (2) only one special version of the algorithm (Convex Combination of Recirculation) was presented, without noticing that the convex combination is not necessary (see below); and (3) the formal analysis was incomplete and unnecessarily limited to the very restrictive case of a circular autoencoder with a single hidden layer, and symmetric weight matrices, and linear output units with "high regression"; (4) no connection to RBP (which was not known at the time) was presented. Here we look at more general classes of algorithms and address all these points.

5 Circular Autoencoder: Analysis of Error Recirculation

5.1 Error Recirculation and Random Backpropagation

As previously discussed, either the error information $I-O^L(0)$ is first computed and then recirculated (Error Recirculation) so that $O^0(1) = I - O^L(0)$, or the output $O^L(0)$ is recirculated (Output Recirculation) so that $O^0(1) = O^L(0)$. In the case of error recirculation, it is natural to focus on plain recirculation (PR) only, as the original information and the recirculated information are not of the same kind. In this case, we have: $O^0(1) = I - O^L(0)$ with the learning rules:

$$\begin{cases} \Delta A^h = \eta O^h(1)(O^{h-1}(0))^t & \text{for} \quad h = 1, \ldots, L-1 \\ \Delta A^L = \eta O^0(1)(O^{L-1}(0))^t \end{cases} \tag{18}$$

Clearly A^L is being updated by gradient descent in a local way, as long as there is a memory for the pass at $t = 0$ in the presynaptic units. For the other layers, $O^h(1) = F^h A^h O^{h-1}(1) = F^h A^h [F^{h-1} A^{h-1} [\ldots F^1 A^1 [I - O^L(0)] \ldots]]$. Thus the error $[I-O^L(0)]$ is being communicated to each layer through a series of matrix multiplications and non-linear transformations, where the matrices themselves are evolving (slowly) in time. The matrices A^h are typically initialized randomly, and thus likely to be full rank, at the beginning of learning and likely to remain full rank during learning of any challenging data set. *Thus, in short, this can be viewed as a form of adaptive non-linear random backpropagation and thus in general can be expected to converge [3, 4].* This is confirmed in the mathematical

derivations and simulations below. Note that the main difference with standard RBP is that the error is being propagated in the forward direction, from the input layer to the target layer, rather than in the reverse direction.

5.2 Error Recirculation: the Linear Case

In the linear case, this can be written as:

$$\begin{cases} \Delta A^h = \eta P^h (I - P^L(I))[P^{h-1} I]^t = \eta P^h (Id - P^L) I I^t (P^{h-1})^t \\ \Delta A^L = \eta (I - P^L I)[P^{L-1} I]^t = \eta (Id - P^L) I I^t (P^{L-1})^t \end{cases} \tag{19}$$

for $h = 1, \ldots, L-1$ where $P^h = A^h A^{h-1} \ldots A^1$. Here Id denotes the Identity matrix associated with the size of the input vector I, $P^0 = Id$.

As usual, in the linear case, we can obtain a system of differential equations by taking expectations over the training data while assuming that the learning rate is small [3, 4]. As a result, we get the polynomial system of ordinary differential equations:

$$\begin{cases} \frac{dA^h}{dt} = P^h (Id - P) \Sigma (P^{h-1})^t \quad \text{for} \quad h = 1, \ldots, L-1 \\ \frac{dA^L}{dt} = (Id - P) \Sigma (P^{L-1})^t \end{cases} \tag{20}$$

where $P = P^L$, $P^0 = Id$, and $\Sigma = E(II^t)$. Thus consecutive matrices are deterministically coupled by the system of differential equations:

$$\frac{dA^{h+1}}{dt} = A^{h+1} \frac{dA^h}{dt} (A^h)^t \tag{21}$$

In the case of a circular autoencoder with a single hidden layer ($L = 2$), the system becomes:

$$\begin{cases} \frac{dA^1}{dt} = A^1 (Id - P) \Sigma \\ \frac{dA^2}{dt} = (Id - P) \Sigma (A^1)^t \end{cases} \tag{22}$$

In general, even in the linear case, these systems are not easy to analyze. However complete analysis, showing that the system converges to optimal solutions, are possible in some important special cases. Since in the linear case error and output recirculation are similar, we prove the corresponding theorems below in the section on output recirculation in circular autoencoders.

6 Circular Autoencoder: Analysis of Output Recirculation

This section contains the main mathematical results.

6.1 Output Recirculation and Random Backpropagation

In the case of Plain Recirculation, we have:

$$\begin{cases} \Delta A^h = \eta(O^h(0) - O^h(1))(O^{h-1}(0))^t & \text{for} \quad h = 1, \ldots, L-1 \\ \Delta A^L = \eta(O^0(0) - O^0(1))(O^{L-1}(0))^t \end{cases} \tag{23}$$

Again A^L is being updated by gradient descent. For the other layers, assuming that all the transfer functions are continuous, we can use the mean value theorem to write:

$$O^h(0) - O^h(1) = F^h A^h O^{h-1}(0) - F^h A^h O^{h-1}(1) = [F^h]'(R^h) A^h \cdot (O^{h-1}(0) - O^{h-1}(1)) \tag{24}$$

where $[F^h]'(R^h)$ represents the vector of derivatives of the activation functions taken at the appropriate intermediate vector of coordinates R^h. The right hand side corresponds to the dot product of this vector with the difference of the outputs in layer $h-1$. One key point, is that the right hand side is 0 when $(O^{h-1}(0) - O^{h-1}(1)) = 0$. Equation 24 holds even when biases are present and, as we shall see, a similar relationship holds when activities are combined in different ways (CCR, RCC, etc.) This equation can be iterated layer-by-layer to produce:

$$O^h(0) - O^h(1) = [F^h]'(R^h) \cdot A^h [[F^{h-1}]'(R^{h-1}) \cdot A^{h-1} \ldots [F^1]'(R^1) \cdot A^1 (I - O^L(0)) \tag{25}$$

Thus again the error $[I - O^L(0)]$ is being communicated to each layer through a series of matrix and scalar multiplications, where the matrices and the scalars themselves are evolving smoothly in time. This can be viewed as a form of non-linear adaptive random backpropagation and thus in general can be expected to converge [3, 4].

In the case of CCR, the difference in the left hand side of Equation 24 is replaced by:

$$O^h(0) - \lambda O^h(0) - (1-\lambda) O^h(1)) = (1-\lambda)(O^h(0) - O^h(1)) \tag{26}$$

thus except for a rescaling of the learning rate by $1 - \lambda$ everything else is the same.

In the case of RCC, if we let G^h be the function $G^h = F^h A^h F^{h-1} A^{h-1} \ldots F^1 A^1$ then, using again the mean value theorem:

$$O^h(0) - O^h(1) = G^h(I) - G^h(\lambda I + (1-\lambda) O^L(0)) = [G^h]'(R^h) \cdot ((1-\lambda) I - (1-\lambda) O^L(0)) \tag{27}$$

where $[G^h]'(R^h)$ represents the vector of derivatives of the activation functions G^h taken at the appropriate intermediate vector of coordinates R^h. Thus again the hidden layer h receives a transformed version of the error.

In the case of CCRCC, using the same notation as above, we have:

$$O^h(0) - O^h(1) = G^h(I) - (\lambda G^h(I) + (1-\lambda)G^h(\lambda I + (1-\lambda)O^L(0)) \quad (28)$$

and thus:

$$O^h(0) - O^h(1) = (1-\lambda)(G^h(I) - G^h(\lambda I + (1-\lambda)O^L(0)) \quad (29)$$

Again, using the mean value theorem, this leads to:

$$O^h(0) - O^h(1) = (1-\lambda)[G^h]'(R^h)\cdot(I - \lambda I - (1-\lambda)O^L(0)) = (1-\lambda)^2[G^h]'(R^h)\cdot(I - O^L(0)) \quad (30)$$

where $[G^h]'(R^h)$ represents the vector of derivatives of the activation functions G^h taken at the appropriate intermediate vector of coordinates R^h. Thus again the hidden layer h receives a transformed version of the error, with a rescaling of the learning rate by a factor $(1-\lambda)^2$ (or $1-\lambda_1)(1-\lambda_2)$ in the case of two different convex combination coefficients). Similar observations can be made with the RWCC algorithm.

In summary, in all cases (PR, CCR, RCC, CCRCC, RWCC) application of the mean value theorem shows that the difference of the corresponding activities during cycle $t = 0$ and cycle $t = 1$ can be written as a function of the error $I - O^L(0)$ and, to a first order, this function is linear with no constant terms, as in random backpropagation and its variants. Furthermore, this remains true when biases are present.

6.2 The Special $L = 2$ Case

For better readability, we will use the notation $A^1 = B$ and $A^2 = A$ in the $L = 2$ case which depends only on two matrices. The recirculation learning rules in the PR case are:

$$\begin{cases} \Delta B = \eta(O^1(0) - O^1(1))(O^0(0))^t \\ \Delta A = \eta(O^0(0) - O^0(1))(O^1(0))^t \end{cases} \quad (31)$$

The learning equations of A and B correspond to Equation 15 taken over one cycle. The learning equation for A corresponds exactly to gradient descent. The learning equation for B can be written as:

$$\Delta B = \eta(FS^1(0) - FS^1(1))(O^0(0))^t = \eta F'(R)(S^1(0) - S^1(1))(O^0(0))^t \quad (32)$$

where F represents the vector transfer function of the hidden layer, and S represents the linear activity coming into the layer. The second equality is obtained by applying the mean value theorem. If the transfer functions are sigmoidal (or increasing) all the components of F' are positive, and if these transfer functions are logistic, then all the components of F' are between 0

and 1. For comparison, purposes, the gradient descent equation for B is easily derived simply by applying the backpropogation algorithm. The output error $(O^0(0) - O^0(1))$ must be multiplied first by the transpose of the forward matrix A, then by the vector of derivatives of the activation functions $F'(S^1(0))$, to yield the backpropagated error. The backpropagated error in turn is multiplied by the presynaptic activity $O^0(0))$. Thus, using matrix notation, the gradient descent learning rule for B is given by:

$$\Delta' B = \eta F'(S^1(0)) A^t (O^0(0) - O^0(1))(O^0(0))^t \tag{33}$$

If we assume that the autoencoder is symmetric, i.e. $B = A^t$, then the gradient descent equation can be rewritten as:

$$\Delta' B = \eta F'(S^1(0)) B (O^0(0) - O^0(1))(O^0(0))^t = \eta F'(S^1(0))(S^1(0) - S^1(1))(O^0(0))^t \tag{34}$$

As all the components of $F'(R)$ have the same sign as the components of $F'(S^1(0))$, it is clear in this case that ΔB is similar to $\Delta' B$. Thus we see that at least in the symmetric case $B = A^t$ the recirculation learning equation is very similar to gradient descent. In particular, the signs of the weight updates are identical. Thus starting with $B(0) \approx A^t(0)$ may help the algorithm converge although the weight updates for A and B do not guarantee that the symmetry will be maintained throughout the learning. In the symmetric linear case, Equations 32 and 34 are the same and recirculation is identical to backpropagation. Furthermore, if the initialization is symmetric $B(0) = A^t(0)$, then symmetry is preserved at all times during learning, i.e. $B = A^t$ at all times.

6.3 Output Recirculation: the Linear Case

In the linear case, taking as usual expectations over the training set with a small learning rate, one obtains a polynomial system of ordinary differential equations of the form:

$$\begin{cases} \frac{dA^h}{dt} = P^h (Id - P) \Sigma (P^{h-1})^t & \text{for} \quad h = 1, \ldots, L-1 \\ \frac{dA^L}{dt} = (Id - P) \Sigma (P^{L-1})^t \end{cases} \tag{35}$$

where $P^h = A^h A^{h-1} \ldots A^1$, $P = P^L$, and $\Sigma = E(II^t)$. Thus in the linear case, plain recirculation of error or plain recirculation of output are the same. For consecutive matrices one again has the relationship:

$$\frac{dA^{h+1}}{dt} = A^{h+1} \frac{dA^h}{dt} [A^h]^t \tag{36}$$

As for the similar case in RBP [3, 4], we see that there is a deterministic coupling between consecutive layers, and the top layer follows gradient descent.

It is instructive to look at the case of $L = 2$ and see how it illuminates the connection to both BP and RBP. When $L = 2$, letting $A_1 = B$ and $A_2 = A$, the recirculation learning differential equations are given by:

$$\begin{cases} \frac{dA}{dt} = (Id - AB)\Sigma B^t \\ \frac{dB}{dt} = B(Id - AB)\Sigma \end{cases} \tag{37}$$

In contrast, the usual gradient descent differential equations are given by:

$$\begin{cases} \frac{dA}{dt} = (Id - AB)\Sigma B^t \\ \frac{dB}{dt} = A^t(Id - AB)\Sigma \end{cases} \tag{38}$$

and the RBP differential equations by:

$$\begin{cases} \frac{dA}{dt} = (Id - AB)\Sigma B^t \\ \frac{dB}{dt} = C(Id - AB)\Sigma \end{cases} \tag{39}$$

where C is a random matrix. *Notice that the equation for dA/dt is exactly the same in all three cases. The difference is that BP uses A^t to send information about the error, RBP replaces it with a fixed random matrix C, and recirculation uses B itself as the "random matrix".*

6.3.1 The Linear $\mathcal{A}^*[1, 1, \ldots, 1]$ Case

In the circular autoencoder $\mathcal{A}^*[1, 1, \ldots, 1]$ case, where each layer contains only one linear neuron, the system becomes:

$$\begin{cases} \frac{da_h}{dt} = \beta a_1^2 \ldots a_{h-1}^2 a_h (1 - P) \quad \text{for} \quad h = 1, \ldots, L-1 \\ \frac{da_L}{dt} = \beta a_1 \ldots a_{L-1}(1 - P) \end{cases} \tag{40}$$

where $\beta = E(I^2)$ and $P = a_1 \ldots a_L$. Clearly $P = 1$ corresponds to the optimum. For consecutive weights, we have:

$$\frac{da_{h+1}}{dt} = a_{h+1} a_h \frac{da_h}{dt} \tag{41}$$

for $h = 1, \ldots, L-2$. Here the coupling can be solved, yielding:

$$\log |a_{h+1}| = \frac{1}{2} a_h^2 + K_h \quad \text{or} \quad |a_{h+1}| = K_h e^{a_h^2/2} \tag{42}$$

for $h = 1, \ldots, L-2$ where in either case K_h is a constant that depends only on the initial conditions. The following theorem states that in most cases the system will be able to learn and converge to an optimal set of weights for which $P = 1$.

Theorem 6.1 *If $\beta > 0$, then for any set of initial conditions satisfying the condition $a_1(0)a_2(0)\ldots a_{L-1}(0) \neq 0$, the system always learn and the weights a_i converge to an optimal equilibrium where $P = 1$. If the initial conditions satisfy $a_1(0)a_2(0)\ldots a_{L-1}(0) = 0$ then $P(t) = 0$ at all times and the system cannot learn. If $\beta = 0$, then for every i $a_i(t) = a_i(0)$ and any choice of initial conditions $a_i(0)$ is optimal.*

Proof. First, if $\beta = 0$, then $I = 0$ and $a_i(t) = a_i(0)$ for every i. Thus any set of initial weights provides an optimal solution, including 0 weights.

Thus, in the rest of the proof, let us assume that $\beta >= 0$. Let $Q(t) = a_1(t)a_2(t)\dots a_{L-1}(t)$. Let us assume that $Q(0) = 0$ (rare set of initial conditions). In this case $P(0) = 0$. Let i_0 be the smallest index for which $a_{i_0}(0) = 0$. Then the variables $a_1(t), \dots, a_{i_0-1}(t)$ evolve in time, but the variables $a_{i_0}(t), \dots, a_L(t)$ remain constant and equal to 0. Thus $P(t) = 0$ at all times and the system cannot learn.

In the main case of interest, let us assume that $Q(0) \neq 0$, which is easily achieved by any random initialization of the weights.

From the system in Equation 40, we immediately have:

$$\begin{cases} \frac{dQ}{dt} = \beta Q(1-P)(1 + a_1^2 + a_1^2 a_2^2 + \dots a_1^2 a_2^2 ... a_{L-2}^2) \\ \frac{da_L}{dt} = \beta Q(1-P) \end{cases} \tag{43}$$

As a result:

$$\begin{cases} \frac{dQ}{dt} = \beta Q(1-P)f \\ \frac{da_L}{dt} = \beta Q(1-P) \\ P = Qa_L \end{cases} \tag{44}$$

where the function f satisfies $f > 1$ at all times. Thus $dQ/dt = f da_L/dt$ and thus the two derivatives always have the same sign. Furthermore note that if $P(0) = 1$, the system is already at an optimal point and all derivatives are 0. Assuming $P(0) \neq 1$, we can examine four possible cases of initial conditions depending on the sign of $Q(0)$ and $a_L(0)$:

1. $Q(0) > 0$ and $a_L(0) \geq 0$: If $P(0) > 1$, then both a_L and Q decrease until they converge to an optimal equilibrium where $P = 1$. If $P(0) < 1$, then both a_L and Q increase until they converge to an optimal equilibrium where $P = 1$.

2. $Q(0) < 0$ and $a_L(0) \leq 0$: If $P(0) > 1$, then both a_L and Q increase and P decreases and converges to an optimal equilibrium where $P = 1$. If $P(0) < 1$, then both a_L and Q decrease, and P increases and converges to an optimal equilibrium where $P = 1$.

3. $Q(0) > 0$ and $a_L(0) \leq 0$: Then initially $P(0) < 1$. Thus both a_L and Q increase, a_L crosses 0 (corresponding to $P = 0$) and then a_L and Q and P keep increasing until P converges to an optimal equilibrium where $P = 1$.

4. $Q(0) < 0$ and $a_L(0) \geq 0$: Then $P(0) < 1$. Thus Thus both a_L and Q decrease, a_L crosses 0 (corresponding to $P = 0$) and then a_L and Q keep decreasing while P increases until P converges to an optimal equilibrium where $P = 1$.

As an example, consider the simple case where $L = 2$. Thus we have:

$$\begin{cases} \frac{da_1}{dt} = \beta a_1(1-P) \\ \frac{da_2}{dt} = \beta a_1(1-P) \end{cases} \tag{45}$$

and thus: $a_2 = a_1 + K$ where $K = a_2(0) - a_1(0)$ is a constant that depends only on the initial conditions. As a result, we have the polynomial ordinary differential equation:

$$\frac{da_1}{dt} = \beta a_1(1 - a_1(a_1 + K)) \tag{46}$$

The polynomial is of degree 3 with a negative leading coefficient and thus the ODE is always convergent to a fixed point. The fixed points are given by:

$$a_1 = 0 \quad \text{and} \quad a_1 = \frac{-K \pm \sqrt{K^2+4}}{2} \tag{47}$$

By simple inspection of the cubic function, the fixed point $a_1 = 0$ is repulsive and the other two fixed points are attractive. Thus for any initial condition $a_1(0) \neq 0$, a_1 will converge to a non-zero fixed point, and so will a_2. At this fixed point, one must have $P = 1$ corresponding to an optimal solution.

Similar results can be obtained with various variants of the algorithms (number of cycles, presynaptic value, combination of activities).

6.3.2 The Linear $\mathcal{A}^*[1, 1, \ldots, 1]$ Case with Multiple Cycles of Recirculation

If we recirculate the output n times through the $\mathcal{A}^*[1, 1, \ldots, 1]$ circular autoencoder and take the difference between the activities in first circulation and the n-th recirculation, the differential equation for learning are given by

$$\begin{cases} \frac{da_h}{dt} = \beta a_1^2 \ldots a_{h-1}^2 a_h(1 - P^n) \quad \text{for} \quad h = 1, \ldots, L-1 \\ \frac{da_L}{dt} = \beta a_1 \ldots a_{L-1}(1 - P^n) \end{cases} \tag{48}$$

where $\beta = E(I^2)$ and $P = a_1 \ldots a_L$. Again we get the same deterministic coupling between consecutive weights and thus:

$$\log |a_{h+1}| = \frac{1}{2} a_h^2 + K_h \quad \text{or} \quad |a_{h+1}| = K_h e^{a_h^2/2} \tag{49}$$

for $h = 1, \ldots, L-2$ where in either case K_h is a constant that depends only on the initial conditions. The following theorem states that in most cases the system will be able to learn and converge to an optimal set of weights for which $P = 1$. It is identical to the above case corresponding to $n = 1$

Theorem 6.2 *If $\beta > 0$, then for any set of initial conditions satisfying the condition $a_1(0)a_2(0)\ldots a_{L-1}(0) \neq 0$, the system always learn and the weights a_i converge to an optimal equilibrium where $P = 1$. If the initial conditions satisfy $a_1(0)a_2(0)\ldots a_{L-1}(0) = 0$ then $P(t) = 0$ at all times and the system*

cannot learn. If $\beta = 0$, then for every i $a_i(t) = a_i(0)$ and any choice of initial conditions $a_i(0)$ is optimal.

Proof. The proof is almost identical to the proof for the case $n = 1$. Again letting $Q(t) = a_1(t) \dots a_{L-1}(t)$, we have:

$$\begin{cases} \frac{dQ}{dt} = \beta Q(1 - P^n)f \\ \frac{da_L}{dt} = \beta Q(1 - P^n) \\ P = Qa_L \end{cases} \tag{50}$$

where the function f satisfies $f > 1$ at all times. The rest of the proof is identical with the obvious adjustments.

One can also recirculate the output m times and compare the activities between cycle m and any earlier cycle n ($n < m$). Specifically, in the learning rate of each weight one can multiply: (1) the difference in post-synaptic activity between cycle n and cycle m (as the "error" term); with (2) the presynaptic activity at cycle n. This yields the system of differential equations:

$$\frac{da_h}{dt} = \beta a_1^2 \dots a_{h-1}^2 a_h P^{2n}(1 - P^{m-n}) \quad \text{for} \quad h = 1, \dots, L \tag{51}$$

Theorem 6.3 *If $\beta > 0$, then for any set of initial conditions satisfying the condition $a_1(0)a_2(0) \dots a_L(0) \neq 0$, the system always learn and the weights a_i converge to an optimal equilibrium where $P = 1$. If the initial conditions satisfy $a_1(0)a_2(0) \dots a_L(0) = 0$ then $P(t) = 0$ at all times and the system cannot learn. If $\beta = 0$, then for every i $a_i(t) = a_i(0)$ and any choice of initial conditions $a_i(0)$ is optimal.*

Proof. The same deterministic coupling between consecutive weights remains and the proof is similar to the previous two proofs.

6.3.3 The Linear $\mathcal{A}^*[1, 1, \dots, 1]$ Case with Different Activity Combination

The previous analyses were carried with plain recirculation (PR). But similar results are obtained with the other ways (e.g. CCR, RCC) of combining activities. For instance, in the CCR case, the learning equations are identical to the PR case except for an additional multiplicative factor of $1 - \lambda$ which simply rescales the learning rate. Thus there is the same convergence to the same optimal solution. And in the linear case, the equations for RCC are identical to the equations of CCR.

Thus in short, in the $\mathcal{A}[1, \dots, 1]$ linear case, for almost all initial conditions and all variations of recirculation algorithms, learning converges to the optimal solution.

6.3.4 The Linear $\mathcal{A}^*[1, N, 1]$ Case

Consider the case of a linear circular autoencoder with expansive architecture $\mathcal{A}^*[1, N, 1]$ with a vector of weights $a = (a_i)$ between the hidden layer and the output layer, and a vector of weights $b = (b_i)$ between the input and the hidden layer. The system of differential equations associated with PR learning is given by:

$$\begin{cases} \frac{da_i}{dt} = \beta b_i(1 - \sum a_i b_i) = \beta b_i(1 - P) \\ \frac{db_i}{dt} = \beta b_i(1 - \sum a_i b_i) = \beta b_i(1 - P) \end{cases} \tag{52}$$

where we let $P = \sum_i a_i b_i$ be the global multiplier of the system. Obviously the derivative of a_i is equal to the derivative of b_i and thus any solution satisfies $a_i(t) = b_i(t) + c_i$, where the constant vector c satisfies: $c_i = a_i(0) - b_i(0)$. We let $A = ||a||^2 = \sum_i a_i^2$, $B = ||b||^2 = \sum_i b_i^2$, and $C = ||c||^2 = \sum_i c_i^2$. We immediately have:

$$\frac{dP}{dt} = \beta(1 - P) \sum_i b_i(a_i + b_i) = \beta(1 - P)(P + B) \tag{53}$$

$$\frac{dA}{dt} = 2\beta(1 - P)P \tag{54}$$

$$\frac{dB}{dt} = 2\beta(1 - P)B \tag{55}$$

The following theorem states that in most cases the system will be able to learn and converge to an optimal set of weights for which $P = 1$.

Theorem 6.4 *If $\beta > 0$, then for any set of initial conditions satisfying the condition $b(0) \neq 0$, the system always learn and the weights converge to an optimal equilibrium where $P = 1$. If the initial conditions satisfy $b(0) = 0$ then all the weights remain constant, $P(t) = 0$ at all times, and the system cannot learn. If $\beta = 0$, then for every i $a_i(t) = a_i(0)$ and $b_i(t) = b_i(0)$ and any choice of initial conditions is optimal.*

Proof. First, if $\beta = 0$, then $I = 0$ and $a_i(t) = a_i(0)$ for every i, and similarly $b_i(t) = b_i(0)$ for every i. Thus any set of initial weights provides an optimal solution, including the case $b(0) = 0$.

Thus, in the rest of the proof, let us assume that $\beta > 0$. Let us assume $b(0) = 0$ (rare set of initial conditions). In this case all derivatives are equal to 0 and, for every i, $a_i(t) = a_i(0)$ and $b_i(t) = b_i(0) = 0$. Thus $P(t) = 0$ at all times and the system cannot learn.

In the main case of interest, let us assume that $b(0) \neq 0$, which is easily achieved by any random initialization of the weights. Note that if there are some values of i for which $b_i(0) = 0$ the corresponding $a_i(t)$ and $b_i(t)$ remain constant, as above. Thus the evolution of the weights is concentrated on those indices i

for which $b_i(0) \neq 0$. We have the obvious bounds on P, and the vectors a, b, and c: $|P^2| = |ab|^2 \leq AB$ and $A = B + C + 2bc \leq B + C + 2\sqrt{BC}$. Together with the equations above, this shows that A, B, and P must be bounded. $P(0) = 1$ is a stable equilibrium. By taking the derivative of AB, we get:

$$\frac{dAB}{dt} = 2\beta P(1 - P)(1 + B) \tag{56}$$

Note that $AB \geq 0$ and $AB = 0$ if and only if $A = 0$ since here we are excluding the case $B = 0$. If $P(0) < 0$, the derivative of AB is negative and AB decreases until $A = P = 0$ after which the derivative of AB becomes positive, and both AB and P grow until $P = 1$. Likewise, if $0 \leq P(0) \leq 1$ the system converges (or remains at) $P = 1$. If $P(0) > 1$, the derivative of AB is negative and AB and P decrease until $P = 1$.

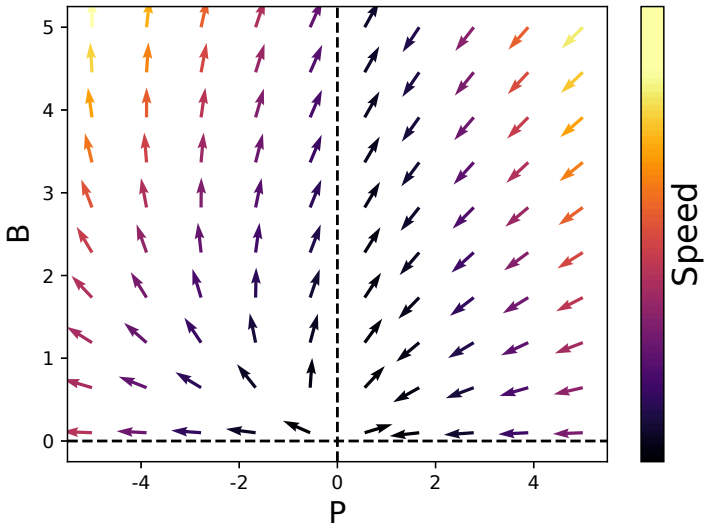

Figure 3: Vector flow in the P, B plane for the system defined by Equations 53 and 55 with $\beta = 1$. The global optimal equilibrium correspond to the vertical line $P = 1$.

The vector flow of the system in the (P, B) plane is shown in Figure 3. Assuming $P(0) \neq 1$, we can examine four possible cases of initial conditions depending on the size of $P(0)$ and $B(0)$:

1. $P(0) > 1$: In this case, initially dA/dt and dB/dt are negative and, from the ODEs above, A and B and P will decrease until $P = 1$.

2. $0 \leq P(0) < 1$: In this case, initially dA/dt and dB/dt are positive and, from the ODEs above, A and B and P will increase until $P = 1$.

3. $-B(0) \leq P(0) < 0$: In this case, initially $dA/dt < 0$, $dB/dt > 0$ and $dP/dt > 0$, thus P will increase and A decrease until $A = P = 0$, after which the dynamics is as in case 2 above. In particular, A and B will continue to increase until $P = 1$.

4. $P(0) \leq -B(0)$: In this case, initially $dA/dt < 0$, $dB/dt > 0$ and $dP/dt < 0$, thus P decreases until $P = B$, after which the dynamics is as in case 3 above. In particular, P will continue to increase and and A to decreases, until $A = P = 0$, after which A and B will increase until $P = 1$.

7 Simulations

7.1 Data Sets

Simulation experiments were performed on the following three data sets. For each data set, a training epoch consisted of 60,000 examples, with another 10,000 examples used for testing.

1. *Lowrank*: We simulated data of dimension d with low rank r by creating an $r \times d$ random matrix V with each element sampled independently from $\mathcal{N}(0, 1)$. Each mini-batch of size n was then created by sampling a new $n \times r$ matrix U, again with each element from $\mathcal{N}(0, 1)$, and computing $U \cdot V$. The data set used in these experiments has $d = 100$ dimensions and rank $r = 20$.

2. *MNIST*: This well-known benchmark data set contains 70,000 28-by-28 grayscale images of handwritten digits, with real pixel values in the range $[0, 1]$. In experiments with linear autoencoders, we use a linear output layer with squared error loss, while in the non-linear architectures we use a sigmoid output layer with cross-entropy loss.

3. *Fashion*: This data set contains 70,000 28-by-28 grayscale images of *Fashion* accessories [10], with real pixel values in the range $[0, 1]$. In experiments with linear autoencoders, we use a linear output layer with squared error loss, while in the non-linear architectures we use a sigmoid output layer with cross-entropy loss.

7.2 Comparison to Backpropagation and Random BP

Output recirculation was compared to backpropagation and random backpropagation using compressive autoencoders on three different data sets, in which the high-dimensional data ($d = 100$ for *Lowrank*, $d = 784$ for *MNIST* and *Fashion*) is compressed down to only 20 dimensions. In order to achieve good reconstruction performance on these tasks, deep learning is needed to find good low-dimensional representations of the data. Facilitating the comparison is the fact that all three learning rules perform gradient descent in the last layer, and differ only in how the intermediate-layer weights are updated.

The experiments demonstrate that recirculation leads to good intermediate representations with small reconstruction error on a variety of data sets and architecture configurations. Figure 4 shows that recirculation quickly learns the relevant latent subspace in the synthetic *Lowrank* task and achieves near-zero reconstruction error. Figure 5 and 6 show that recirculation learns useful intermediate representations on real images, in both linear and non-linear architectures, with one or three hidden layers. Examples of reconstructed *MNIST* digits are shown in Figure 7.

For each experiment, network weights were initialized from a scaled uniform distribution [11], except for the bias terms at the output layer which was set to match the expectation of the training data. The random backpropagation matrices were constructed using the same initialization scheme. Weight updates were made using a mini-batch size of 100 and a learning rate of 0.01, which was decreased by a factor of 10 for architectures with three hidden layers, and another factor of 10 for experiments on the *Lowrank* data set since the errors were larger. In the linear architectures, the incoming weights to every neuron were not allowed to exceed 2.0 for numerical stability (a constraint that only requires local information). Also in the linear setting, all neurons had linear activation and a mean squared error (MSE) objective, while in the non-linear setting the hidden neurons had tanh activation while the output neurons had the logistic activation with binary cross-entropy objective, or equivalently the Kullback–Leibler divergence (KL).

Deep learning with stochastic gradient descent can lead to non-monotonic trajectories of the training loss, and the non gradient descent learning rules are even more susceptible to this. In recirculation, this is sometimes due to the hidden layer weights growing very large, suggesting the need for regularization, weight constraints, or different learning rates for each layer. Figure 8 gives an example of this behavior in an *MNIST* compressive autoencoder with ten hidden layers of 100 tanh units each, where using smaller learning rates in the deep layers stabilizes learning. These details are similarly important for RBP, SRBP, and ARBP algorithms, along with the initialization of the (fixed) backwards weights.

7.3 Variants of the Recirculation Algorithm

Using the same *MNIST* autoencoder architectures discussed above, several variants on the plain recirculation algorithm were explored. First, Figure 9 compares the five proposed approaches (PR, CCR, RCC, CCRCC, RWCC) to combining the initial and recirculated output activity, which perform very similarly in experiments. Second, in Figure 10, the neuron activity is recirculated multiple times, from one to five, through the network, then the difference with the initial activity is used in the recirculation learning rule. The top layer is updated with gradient descent in every case. Again all variants seem to perform well.

Thus, in short, simulations on three data sets corroborate the mathematical analyses. In particular, they show that recirculation can be used to train autoencoders, robustly with respect to its many variants. Thus recirculation can

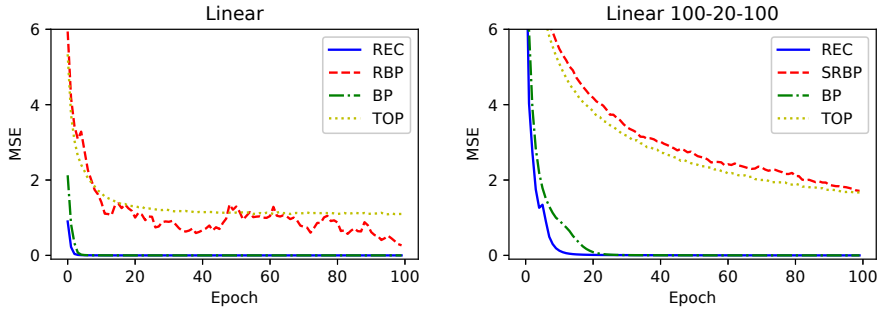

Figure 4: *Lowrank* test set performance of linear compressive autoencoders, with either a single hidden layer of 20 units (left) or three hidden layers of shape 100-20-100 (right), trained with recirculation (REC), random backpropagation (RBP), skip random backpropagation (SRBP), backpropagation (BP), and only training the top layer (TOP).

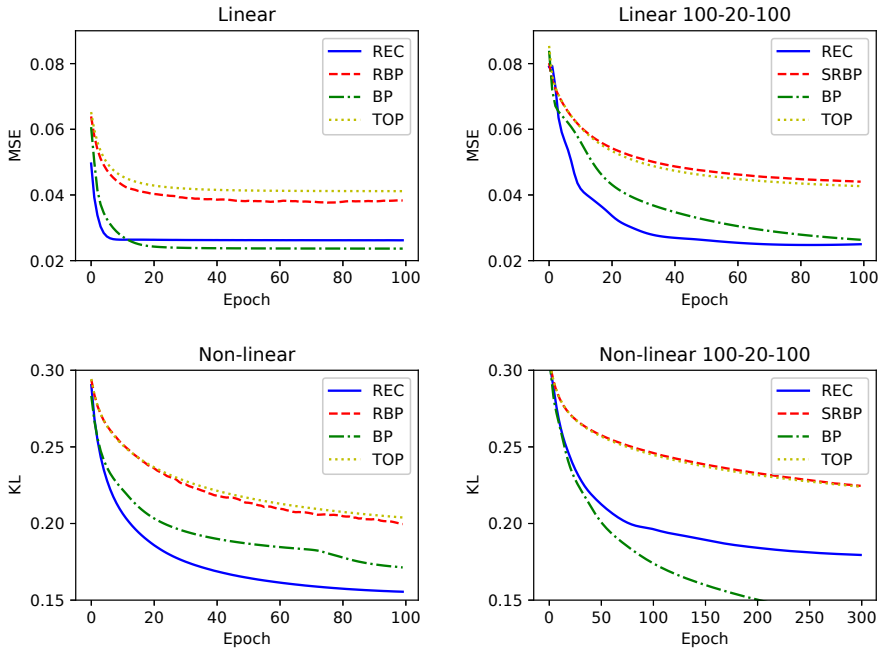

Figure 5: *MNIST* test set performance of linear and non-linear compressive autoencoders, with either a single hidden layer of 20 units (left) or three hidden layers of shape 100-20-100 (right), trained with recirculation (REC), random backpropagation (RBP), skip random backpropagation (SRBP), backpropagation (BP), and only training the top layer (TOP).

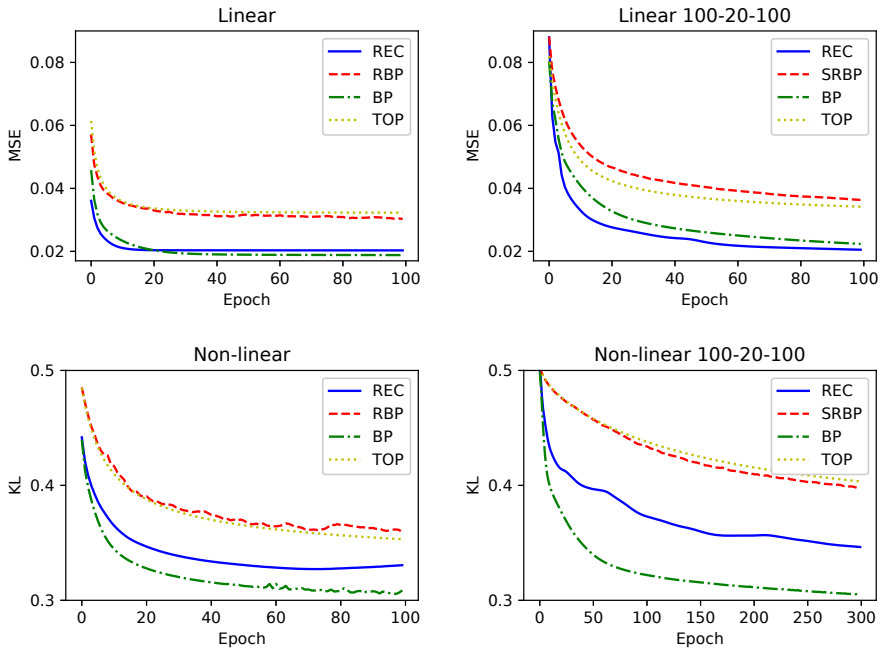

Figure 6: *Fashion* test set performance of linear and non-linear compressive autoencoders, with either a single hidden layer of 20 units (left) or three hidden layers of shape 100-20-100 (right), trained with recirculation (REC), random backpropagation (RBP), skip random backpropagation (SRBP), backpropagation (BP), and only training the top layer (TOP).

Figure 7: Reconstructed *MNIST* digits from the compressive autoencoder network with 100-20-100 hidden linear units, trained with backpropagation (BP), skip random backpropagation (SRBP), and recirculation (REC).

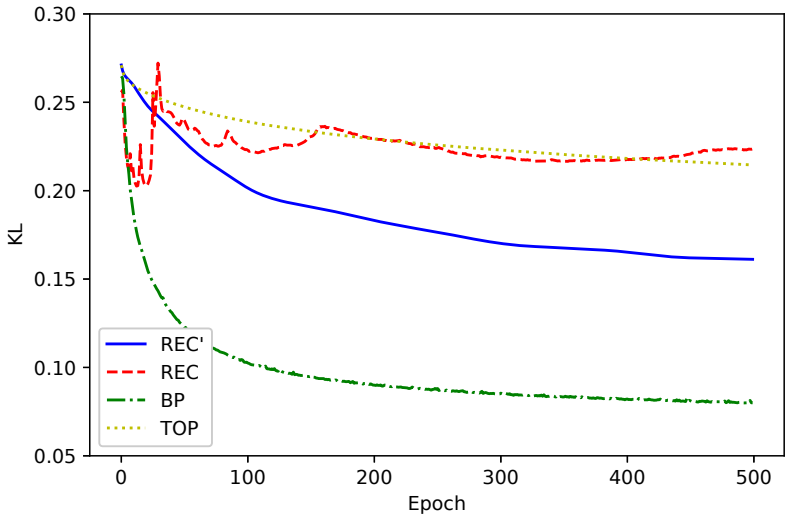

Figure 8: *MNIST* test set performance for a non-linear compressive autoencoder with ten hidden tanh layers, trained using recirculation with layer-specific learning rates (REC′), recirculation (REC), backpropagation (BP), and only training the top layer (TOP). The same learning rate of 0.01 is used in each algorithm, except for REC′, which uses a smaller learning rate in the lower layers: $\frac{0.01}{2^n}$ for the n-th hidden layer away from the output ($n \in [0, 10]$). At the beginning of learning, the loss trajectory of REC closely tracks that of BP, but then the loss increases and doesn't recover. Using smaller learning rates in the deeper layers alleviates this problem.

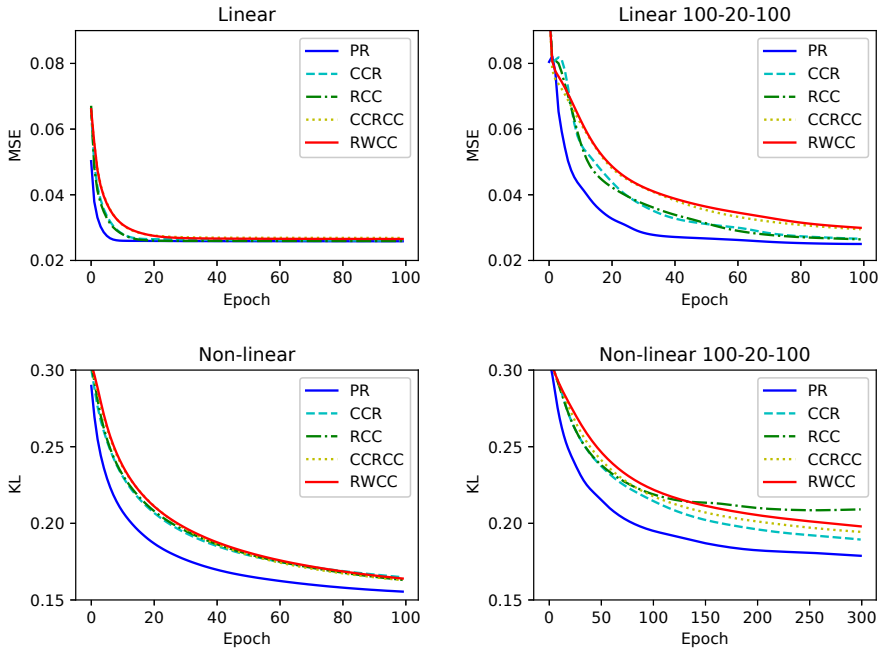

Figure 9: *MNIST* test set performance of linear and non-linear compressive autoencoders, with either a single hidden layer of 20 units (left) or three hidden layers of shape 100-20-100 (right), trained with recirculation where the recirculated activity is combined with the initial activity using plain recirculation (PR), convex combination of recirculation (CCR), recirculation of convex combination (RCC), convex combination of recirculation of convex combination (CCRCC), or recirculation with convex combination (RWCC).

be used to derive effective hidden representations of the data in an unsupervised fashion, without the need for additional deep learning channels. There is room left to further explore how to best optimize hyperparameters for recirculation and the related RBP algorithms in future studies.

8 Conclusion

Most applications of neural networks today are carried by simulating neural networks in digital machines. The usefulness of these applications often leads one to forget that in these simulations there are no neurons or synaptic weights, only their fantasies stored in digital memory format. Forgetting the reality of neural computations comes at a price, if nothing else energetic (a supercomputer can consume $\tilde{1}$00,000 Watts while a brain consumes only 40 Watts), and new insights can be gained by thinking about learning in the machine, i.e. learning

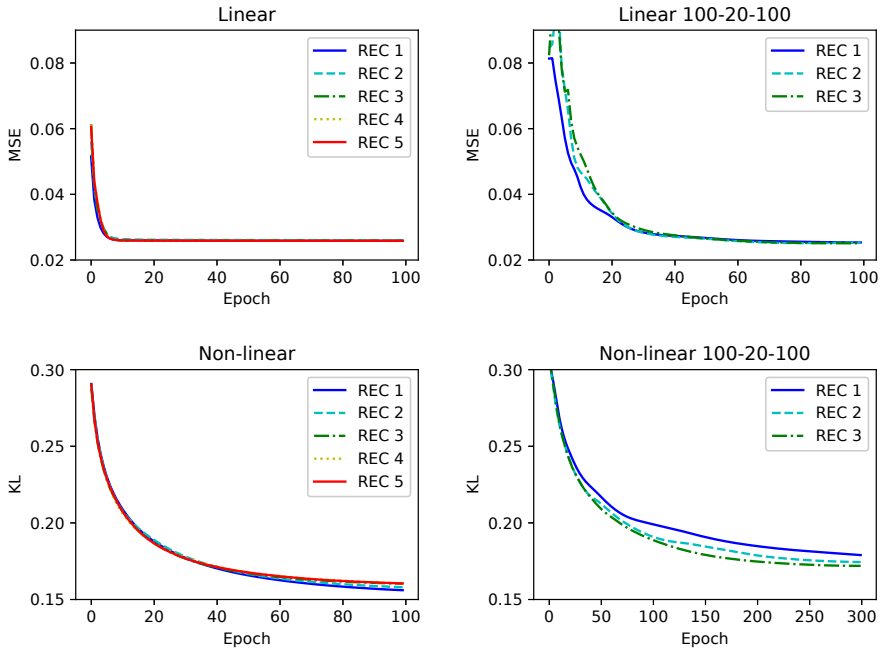

Figure 10: *MNIST* test set performance of linear and non-linear compressive autoencoders, with either a single hidden layer of 20 units (left) or three hidden layers of shape 100-20-100 (right), trained with the plain recirculation algorithm, but where the learning rule in the deep weights uses the difference between the initial activity and the activity after multiple (1-5) recirculations through the circular autoencoder. A single recirculation (REC 1) is equivalent to the plain recirculation algorithm.

in native—as opposed to simulated—neural networks implemented in carbon, silicon, or other substrates.

In essence, learning in the machine requires putting oneself in the shoes of a physical learning system and its components, such as its neurons and its synapses. Putting oneself in the shoes of a neuron, for instance, reveals why the CONNECTED problem[1] is much harder than it appears to be to a human observer. Likewise, imagining that neurons have a high rate of failure leads immediately to the dropout learning algorithm [12, 13] where, for each training example, neurons are randomly dropped from the training procedure. Other examples of "in the machine" thinking at the neuronal level include using local connectivity as opposed to full connectivity, or relaxing the exact weight sharing assumption behind convolutional neural networks.

More importantly, putting oneself in the shoes of a synapse provides a better appreciation of the deep learning problem, and leads to the notions of local learning, the stratification of learning rules by their functional complexity, the identification of the fundamental limitations of deep local learning—why Hebbian learning cannot train a feedforward convolutional neural network—and to local deep learning and the learning channel [1].

Here we have studied autoencoders from the learning-in-the-machine point of view and shown that learning algorithms must exist that do not require a separate learning channels running in the revers direction. This class of algorithms, broadly named recirculation algorithms, contains several variations which we have derived and studied through mathematical analyses and simulations. These analyses have revealed a remarkable connection with random backpropagation, namely that recirculation algorithms can be viewed as forms of adaptive random backpropagation. Recirculation algorithms provide some additional plausibility for the use of autoencoders in machines, perhaps including the brain, for extracting new, compressed or expanded, representations of the data without any supervision.

In terms of biological neural networks, there are two key ingredients of recirculation that increase its potential plausibility. First, it removes the need for symmetric connections, required by backpropagation, which may be challenging for biological neurons. In fact, recirculation goes beyond that. In principle, the problem of symmetric weights could be addressed by random backpropagation, i.e. by using random weights in the learning channel. Recirculation removes even the need for a separate learning channel. Second recirculation takes advantage of time differences, or signal derivatives, in the postsynaptic neurons to encode error information, in a way that is not inconsistent with biological observations and theories of learning, including STDP.

Using time differences in learning rules is just one example of a broader theme of research focused on the role of time in physical neural systems. In digitally simulated neural networks, most issues related to time are taken care of

[1]Given a binary input vector, determine whether the 0's are all adjacent to each other (with or without wrap around). The connectedness makes the problem easy to solve for the human visual system. However, a neuron must learn the particular permutation associated with the ordering of the coordinates.

in a manner that is external to the neural networks being simulated. Typically, activities in neurons and layers are updated sequentially according to a predetermined schedule. Likewise the timing and sequence of example presentations, weight updates, hyperparameter optimizations and so forth are all handled by external programs, using the digital computer clock. Obviously physical neurons must operate in "real time", with limited storage capacity, in an organic, largely asynchronous, fashion and within larger networks. Operating in real time must be true not only for learning, but also for all other activities neurons partake to. The time dimension of neural computations was very much present in some of the neuromorphic thinking of the 1980s [14, 15, 16, 17, 18] and revisiting some of these issues, in particular by looking at the role of time while learning in the machine, may lead to new insights.

Acknowledgement

Work in part supported by DARPA grant D17AP00002 and NIH grant NIH GM123558 to PB.

Appendix A. Physical Circular Autoencoders

When considering physical implementations of circular architectures with identical input and output units, additional issues need to be considered regarding the nature of the connections and the corresponding physical constraints. For simplicity, consider the case of a circular autoencoder with a single hidden layer ($L = 2$). In the bidirectional case (circular bidirectional) (Figure 11, left), the same connections between the input layer and the hidden layer are used bidirectionally. If the connections are isotropic (circular bidirectional symmetric), the corresponding synaptic weight is the same in both directions, otherwise it could differ (circular bidirectional asymmetric). If the connections between the layers are different, then we obtain a circular conjoined architecture, which again could be symmetric (Figure 11, middle) or asymmetric (Figure 11, right). At the computational level considered here, the bidirectional and conjoined architectures cannot be distinguished.

Appendix B. Recirculation Algorithms in the Standard Autoencoder

This section considers learning algorithms for standard autoencoders. It can be skipped by readers not interested in the technicalities that arise from the physical layout of the standard autoencoder.

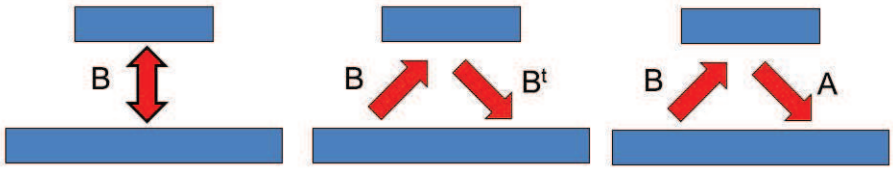

Figure 11: Circular autoencoder architectures, where the neurons in the input and output layer are identical. Left: circular bidirectional architecture, where signals can be sent bidirectionally along the same connection presumably, but not necessarily, with the same weight in both directions (symmetric vs asymmetric cases). Middle: circular conjoined symmetric architecture with symmetric connections ($A = B^t$). Right: circular conjoined asymmetric architecture with asymmetric connections.

B1. Three Deep Learning Channels

Recall that the deep learning channels are the channels used by a physical neural system to communicate non local information to its synapses [1]. For the standard autoencoder, there are three deep channels, and their combinations, to be considered (Figure 12): (1) a deep learning channel from the input layer to the output layer through a matrix $C_1 = Id$ to communicate the input I to the output layer in order to enable the computation of the error $T - O^2 = I - O^2$ locally at the output layer; (2) a deep learning channel from the output layer to the hidden layer to communicate outputs (O^2), or errors ($T-O^2 = I-O^2$) when the first deep learning channel is present, to the hidden layer through a matrix C_2; and (3) a deep learning channel from the output layer to the input layer to communicate outputs (O^2), or errors ($T - O^2 = I - O^2$) when the first deep channel is present, to the input layer through a matrix C_3. The information communicated by the third channel can be recirculated through the network.

B2. Algorithmic Variations

To develop learning algorithms for the standard autoencoder several possibilities arise depending on:

1. The combination of deep learning channels that are available.
2. The nature of the matrices associated with the deep learning channels.
3. The nature of the information communicated through the channels, in particular the output O^2 versus the error $T - O^2 = I - O^2$.

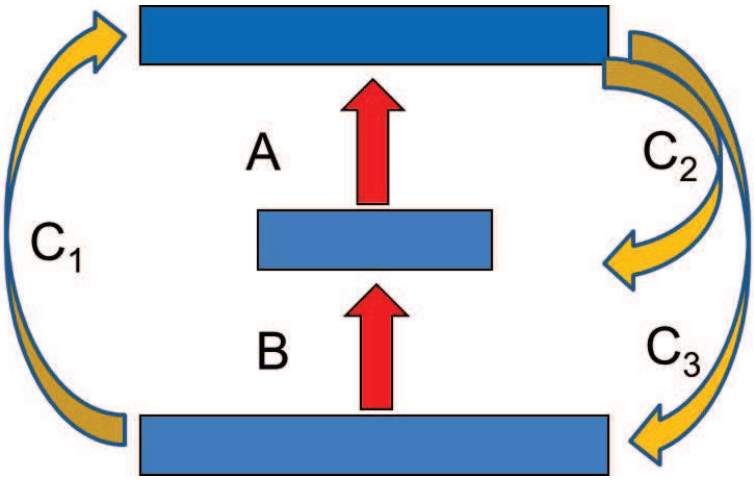

Figure 12: A standard autoencoder and its possible deep learning channels. A combination of these channels is necessary for the autoencoder to be able to reach optima of the error function using only local learning rules.

4. How the recirculated neural activity combines with previous cycles, and over which time scales and number of cycles (PR, CCR, RCC, CCRCC, RWCC).

5. The local learning rules used to update the weights, whether learning rules are applied also to the matrices in the deep learning channels (adaptive case), and whether non-linear transfer functions are used in the deep learning channels.

To streamline the analysis and reduce the number of possibilities here we focus on the main and most interesting cases in terms of channels and information being communicated. In particular we assume that $C_1 = Id$ and that $C_3 = Id$ unless otherwise stated. We also exclude the case where both C_2 and C_3 are present for several reasons. First this possibility is not necessary since several useful algorithms can be found using only C_2 or C_3. Furthermore, from a physical neural machine standpoint, having both C_2 and C_3 incurs additional connectivity costs. And finally, having both C_2 and C_3 leads to a combinatorial explosion of cases depending on whether output or errors are communicated over each channel, and this problem is compounded as the number of hidden layers is increased. In terms of unit activities, we consider primarily the PR case but, as for the case of the circulat autoencoder, it should be clear how to extend the results to other cases. In terms of channels this leaves the combination listed in Table 5 organized into three groups:

1. **Recirculation of Ouputs Only:** corresponding to C_2 alone to recirculate the output to the hidden layer or, C_3 alone to recirculate the output to the input layer.

2. **Recirculation of Errors Only:** corresponding to $C_1 = Id$ to compute the error, with C_2 to recirculate it to the hidden layer, or C_3 to recirculated it to the input layer (but not both).

3. **Mixed Recirculation (Error Calculation with Recirculation of Outputs):** corresponding to $C_1 = Id$ to compute the error, with C_2 to recirculate the output to the hidden layer, or C_3 to recirculate the output to the input layer (but not both).

C_1	C_2	C_3	
0	0	0	optimal learning impossible
0	1	0	optimal learning impossible or problematic
0	0	1	circular autoencoder
1	0	0	optimal learning impossible (A can be trained but not B)
1	1	0	backpropagation of errors (BP with $C2 = A^t$; RBP with $C2 = C$)
1	0	1	circular autoencoder
1	1	0	mixed recirculation problematic ($C = B^t$)
1	0	1	mixed recirculation (circular autoencoder)

Table 5: Learning channel combinations in the standard autoencoder. The table is further subdivided into three groups. In the first "output only" group (top 3 rows), the error is not computed in the output layer and only the output activity is communicated over the remaining channels (C_2 or C_3). In the second "error only" group (middle 3 rows) the error is computed in the output layer and communicated over the remaining channels. In the third "mixed" group (bottom 2 rows), the error is computed in the output layer and used to train A, but the output activity is communicated instead over the remaining channels. See text for detailed explanation of each row.

B3. Recirculation of Outputs Only

This case corresponds to the first three rows of Table 5 and Figure 13. If there are no deep learning channels at all (row 1), optimal learning is not possible since no information about the error is available in order to train A or B [1]. If channel C_2 is implemented by a matrix C connecting the output layer to the hidden layer (row 2), it is still not clear how A could be trained since no error information is available. [As far as learning B, if $C = B$ then the hidden layer can compute $BI - BO^2 = B(I - O^2)$. So a transformed version of the error could be available in the hidden layer, but this would not remain true as B evolves

(C would have to evolve identically violating locality again)]. Thus, in short, optimal learning seems problematic. Finally, if the output is recirculated to the input layer (row 3) we are back to the case of a circular autoencoder with one hidden layer if $C_3 = Id$, with a choice of recirculating the error or the output. If $C_3 = C$ for some $C \neq Id$, this could still correspond to a circular autoencoder with two hidden layers, provided the output is defined as $O^3 = CABI$.

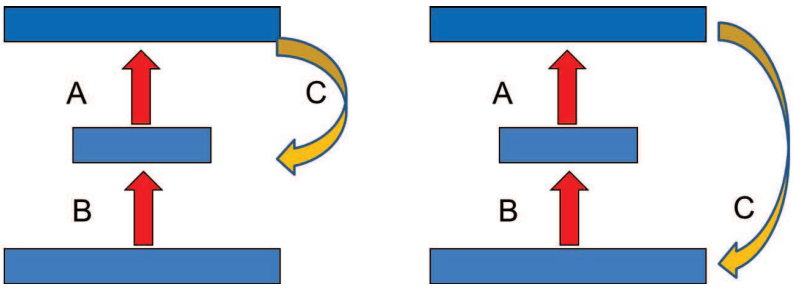

Figure 13: Recirculations of Outputs Only. Left: The output O^2 is fed back to the hidden layer through a matrix $C_2 = C$. Right: The output O^2 is fed back to the input layer through a matrix $C_3 = C$, with most often $C_3 = Id$.

B4. Recirculation of Errors Only

This case corresponds to the middle three rows of Table 5 and Figure 14. In this case, we assume that $C_1 = Id$ is present, allowing the computation of the error $T - O^2 = I - O^2$ locally at the output layer. This error can be fed back to the hidden layer using a matrix C_2, or to the input layer using a matrix C_3. Because the error is available at the output layer, the matrix A can be adjusted using gradient descent as the local learning algorithm, as usual. However if only $C_1 = Id$ is available (row 4), then there is no mechanism for providing error information to the hidden layer in order to train the matrix B. And thus optimal learning is not possible. If C_2 is present (row 5), then if $C_2 = A^t$ this corresponds to backpropagation. However this requires postulating some physical mechanism to ensure that $C_2 = A^t$ at all times during learning. More elegantly, if $C_2 = C$ for some fixed random matrix C, learning can proceed by random backpropagation and no mechanism to ensure weight symmetry is needed (Figure 15). Finally, if $C_1 = Id$ and $C_3 = Id$ (row 6), we are back to the case of a circular autoencoder with one hidden layer with error recirculation. The hidden layer receives a transformed version of the error equal to $B(I - O^2)$. If $C_3 = C \neq Id$, learning is still possible since the hidden layer receives a transformed version of the error equal to $BC(I - O^2)$. As usual, using the mean value theorem this remains true if there is a non-linearity F in the hidden layer (the transformed version of the error is of the form $F'(R)BC(I - O^2)$. Because B evolves in time, this is a form of ARBP.

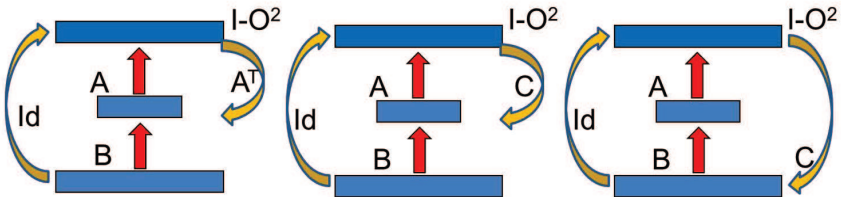

Figure 14: Recirculation of Errors Only. The channel with the identity matrix ($C_1 = Id$) enables the local computation of the error $T - O^2 = I - O^2$ at the output layer. This error is used to adapt the matrix A. This error is also fed back to the hidden layer or to the input layer. Left: Backpropagation ($C_2 = A^t$). Middle: Random backpropagation using a random matrix $C_2 = C$. Right: Error recirculation using a matrix $C_3 = C$.

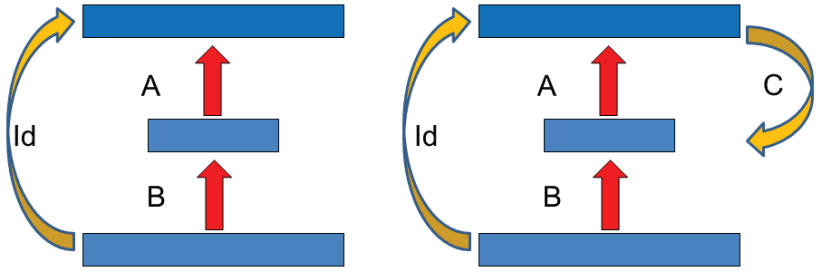

Figure 15: Standard backpropagation or random backpropagation, architecture for the standard autoencoder. Left: This requires first communicating the inputs to the output layer ($C_1 = Id$), so that the error $T - O^2 = I - O^2$ can be computed locally. Right: Then the error is backpropagated through a matrix $C = A^t$ in the case of backpropagation, or a random matrix C in the case of random backpropagation. The deep learning channels are shown in orange.

B5. Mixed Recirculation: Recirculation of Outputs with Error

This case correspond to the last two rows of Table 5 and Figure 16. In this case, we assume that $C_1 = Id$ is present, allowing the computation of the error $T - O^2 = I - O^2$ locally at the output layer. This error can be used again to

update the matrix A by gradient descent. In addition the output O^2 is fed back to the hidden layer using a matrix C_2 (row 7), or to the input layer using a matrix C_3 (row 8). When the output is recirculated to the hidden layer (row 7), it is not clear that this case can be made to work with a generic matrix C. However, if $C = B$, then the hidden layer sees BI in the initial forward pass and BO^2 in the recirculation pass. Thus the hidden layer can locally compute $B(I - O^2)$ (assuming the two passes are local in time), which is again the error transformed by an adaptive matrix B. Thus again this case falls under ARBP and is convergent under the present assumptions. Even if one takes into account a non-linearity F in the hidden layer, then the layer sees $O^1(0) = FBI$ in the initial forward pass, and $O^1(1) = FBO^2(0)$ in the recirculation pass. Assuming the two passes are local in time, the layer can locally compute the difference: $O^1(0) - O^1(1)$. By the mean value theorem, we have: $O^1(0) - O^1(1) = FBI - FBO^2(0) = F'(R)B(I - O^2(0))$ where F' is a vector of derivatives and R is a vector of intermediary values. Thus even in this case the hidden layer receives a noisy, but in general full-rank, transformed version of the error and thus it should be able to learn as in ARBP. The limitation, however, is that in a physical neural system this case requires a physical mechanism for ensuring that $C_2 = B$ at all times. Finally, when the output is recirculated to the input layer (row 8) with $C_3 = Id$, B can be trained as in a circular autoencoder with a single hidden layer.

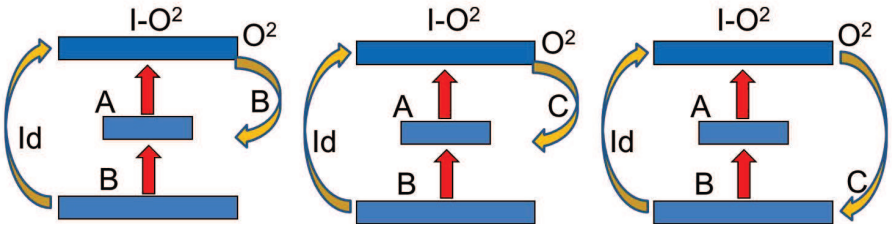

Figure 16: Mixed Recirculation: The channel with the identity matrix ($C_1 = Id$) enables the local computation of the error $T - O^2 = I - O^2$ at the output layer. This error is used to adapt the matrix A by gradient descent. The output O^2 is fed back to the hidden layer or to the input layer. Left: ($C_2 = A^t$). Middle: Mixed recirculation using a matrix $C_2 = C$. Right: Mixed recirculation using a matrix $C_3 = C$.

References

[1] Pierre Baldi and Peter Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:61–74, 2016.

[2] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7, 2016.

[3] P. Baldi, Z. Lu, and P. Sadowski. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial Intelligence*, 2018. In press. Also: arXiv:1612.02734.

[4] P. Baldi, Z. Lu, and P. Sadowski. Learning in the machine: the symmetries of the deep learning channel. *Neural Networks*, 95:110–133, 2017.

[5] Emre O. Neftci, Somnath Paul, Charles Augustine, and Georgios Detorakis. Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines. *Frontiers in Neuroscience*, 11, 2017.

[6] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1037–1045, 2016.

[7] Xiaohui Xie and H. Sebastian Seung. Spike-based learning rules and stabilization of persistent neural activity. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 199–208. MIT Press, 2000.

[8] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1988.

[9] Geoffrey E Hinton and James L McClelland. Learning representations by recirculation. In *Neural information processing systems*, pages 358–366. New York: American Institute of Physics, 1988.

[10] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

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

[14] John Hopfield and David Tank. Computing with neural circuits: A model. *Science*, 233(4764):625–633, 1986.

[15] David Tank and John Hopfield. Neural computation by concentrating information in time. *Proceedings of the National Academy of Sciences*, 84(7):1896–1900, 1987.

[16] Carver Mead and Misha Mahowald. A silicon model of early visual processing. *Neural Networks*, 1(1):91–97, 1988.

[17] Richard Lyon and Carver Mead. An analog electronic cochlea. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1119–1134, 1988.

[18] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*, volume 80. Springer Science & Business Media, 2012.