Deep Learning without Weight Transport

Mohamed Akrout
University of Toronto, Triage

Collin Wilson
University of Toronto

Peter C. Humphreys
DeepMind

Timothy Lillicrap
DeepMind, University College London

Douglas Tweed
University of Toronto, York University

Abstract

Current algorithms for deep learning probably cannot run in the brain because they rely on *weight transport*, where forward-path neurons transmit their synaptic weights to a feedback path, in a way that is likely impossible biologically. An algorithm called feedback alignment achieves deep learning without weight transport by using random feedback weights, but it performs poorly on hard visual-recognition tasks. Here we describe two mechanisms — a neural circuit called a *weight mirror* and a modification of an algorithm proposed by Kolen and Pollack in 1994 — both of which let the feedback path learn appropriate synaptic weights quickly and accurately even in large networks, without weight transport or complex wiring. Tested on the ImageNet visual-recognition task, these mechanisms outperform both feedback alignment and the newer sign-symmetry method, and nearly match backprop, the standard algorithm of deep learning, which uses weight transport.

1 Introduction

The algorithms of deep learning were devised to run on computers, yet in many ways they seem suitable for brains as well; for instance, they use multilayer networks of processing units, each with many inputs and a single output, like networks of neurons. But current algorithms can't quite work in the brain because they rely on the error-backpropagation algorithm, or backprop, which uses *weight transport*: each unit multiplies its incoming signals by numbers called weights, and some units transmit their weights to other units. In the brain, it is the synapses that perform this weighting, but there is no known pathway by which they can transmit their weights to other neurons or to other synapses in the same neuron [1, 2].

Lillicrap et al. [3] offered a solution in the form of feedback alignment, a mechanism that lets deep networks learn without weight transport, and they reported good results on several tasks. But Bartunov et al. [4] and Moskovitz et al. [5] have found that feedback alignment does not scale to hard visual recognition problems such as ImageNet [6].

Xiao et al. [7] achieved good performance on ImageNet using a *sign-symmetry* algorithm in which only the signs of the forward and feedback weights, not necessarily their values, must correspond, and they suggested a mechanism by which that correspondence might be set up during brain development. Krotov and Hopfield [8] and Guerguiev et al. [9] have explored other approaches to deep learning without weight transport, but so far only in smaller networks and tasks.

Here we propose two different approaches that learn ImageNet about as well as backprop does, with no need to initialize forward and feedback matrices so their signs agree. We describe a circuit called a *weight mirror* and a version of an algorithm proposed by Kolen and Pollack in 1994 [10], both of which let initially random feedback weights learn appropriate values without weight transport.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

There are of course other questions about the biological implications of deep-learning algorithms, some of which we touch on in Appendix C, but in this paper our main concern is with weight transport.

2 The weight-transport problem

In a typical deep-learning network, some signals flow along a *forward path* through multiple layers of processing units from the input layer to the output, while other signals flow back from the output layer along a *feedback path*. Forward-path signals perform inference (e.g. they try to infer what objects are depicted in a visual input) while the feedback path conveys error signals that guide learning. In the forward path, signals flow according to the equation

$$\mathbf{y}_{l+1} = \phi(\mathbf{W}_{l+1}\ \mathbf{y}_l + \mathbf{b}_{l+1}) \tag{1}$$

Here $\mathbf{y}_l$ is the output signal of layer l, i.e. a vector whose *i*-th element is the activity of unit i in layer l. Equation 1 shows how the next layer $l+1$ processes its input $\mathbf{y}_l$: it multiplies $\mathbf{y}_l$ by the forward weight matrix $\mathbf{W}_{l+1}$, adds a bias vector $\mathbf{b}_{l+1}$, and puts the sum through an activation function ϕ. Interpreted as parts of a real neuronal network in the brain, the $\mathbf{y}$'s might be vectors of neuronal firing rates, or some function of those rates, $\mathbf{W}_{l+1}$ might be arrays of synaptic weights, and $\mathbf{b}_{l+1}$ and ϕ bias currents and nonlinearities in the neurons.

In the feedback path, error signals $\boldsymbol{\delta}$ flow through the network from its output layer according to the error-backpropagation [11] or *backprop* equation:

$$\boldsymbol{\delta}_l = \phi'(\mathbf{y}_l)\ \mathbf{W}_{l+1}^T\ \boldsymbol{\delta}_{l+1} \tag{2}$$

Here ϕ' is the derivative of the activation function ϕ from equation (1), which can be computed from $\mathbf{y}_l$. So feedback signals pass layer by layer through weights $\mathbf{W}_l^T$. Interpreted as a structure in the brain, the feedback path might be another set of neurons, distinct from those in the forward path, or the same set of neurons might carry inference signals in one direction and errors in the other [12, 13].

Either way, we have the problem that the same weight matrix $\mathbf{W}_l$ appears in the forward equation (1) and then again, transposed, in the feedback equation (2), whereas in the brain, the synapses in the forward and feedback paths are physically distinct, with no known way to coordinate themselves so one set is always the transpose of the other [1, 2].

3 Feedback alignment

In feedback alignment, the problem is avoided by replacing the transposed $\mathbf{W}_l$'s in the feedback path by random, fixed (non-learning) weight matrices $\mathbf{B}_l$,

$$\boldsymbol{\delta}_l = \phi'(\boldsymbol{y}_l)\ \mathbf{B}_{l+1}\ \boldsymbol{\delta}_{l+1} \tag{3}$$

These feedback signals $\boldsymbol{\delta}$ drive learning in the forward weights $\mathbf{W}$ by the rule

$$\Delta\mathbf{W}_{l+1} = -\eta_W\ \boldsymbol{\delta}_{l+1}\ \mathbf{y}_l^T \tag{4}$$

where η_W is a learning-rate factor. As shown in [3], equations (1), (3), and (4) together drive the forward matrices $\mathbf{W}_l$ to become roughly proportional to transposes of the feedback matrices $\mathbf{B}_l$. That rough transposition makes equation (3) similar enough to the backprop equation (2) that the network can learn simple tasks as well as backprop does.

Can feedback alignment be augmented to handle harder tasks? One approach is to adjust the feedback weights $\mathbf{B}_l$ as well as the forward weights $\mathbf{W}_l$, to improve their agreement. Here we show two mechanisms by which that adjustment can be achieved quickly and accurately in large networks without weight transport.

4 Weight mirrors

4.1 Learning the transpose

The aim here is to adjust an initially random matrix $\mathbf{B}$ so it becomes proportional to the transpose of another matrix $\mathbf{W}$ without weight transport, i.e. given only the input and output vectors $\mathbf{x}$ and

$\mathbf{y} = \mathbf{W}\mathbf{x}$ (for this explanation, we neglect the activation function ϕ). We observe that $E\left[\mathbf{x}\,\mathbf{y}^T\right] = E\left[\mathbf{x}\,\mathbf{x}^T\mathbf{W}^T\right] = E\left[\mathbf{x}\,\mathbf{x}^T\right]\mathbf{W}^T$. In the simplest case, if the elements of $\mathbf{x}$ are independent and zero-mean with equal variance, σ^2 , it follows that $E\left[\mathbf{x}\,\mathbf{y}^T\right] = \sigma^2\mathbf{W}^T$. Therefore we can push $\mathbf{B}$ steadily in the direction $\sigma^2\mathbf{W}$ using this *transposing rule*,

$$\Delta\mathbf{B} = \eta_B\ \mathbf{x}\,\mathbf{y}^T \tag{5}$$

So $\mathbf{B}$ integrates a signal that is proportional to $\mathbf{W}^T$ on average. Over time, $\mathbf{B}$ may grow large, but if we add a mechanism such as weight decay to keep $\|\mathbf{B}\|$ small [14–16], then the initial, random values in $\mathbf{B}$ shrink away, and $\mathbf{B}$ converges to a scalar multiple of $\mathbf{W}^T$ (see Appendix A for an account of this learning rule in terms of gradient descent).

4.2 A circuit for transposition

Figure 1 shows one way the learning rule (5) might be implemented in a neural network. This network alternates between two modes: an *engaged mode*, where it receives sensory inputs and adjusts its forward weights to improve its inference, and a *mirror mode*, where its neurons discharge noisily and adjust the feedback weights so they mimic the forward ones. Biologically, these two modes may correspond to wakefulness and sleep, or simply to practicing a task and then setting it aside for a moment.

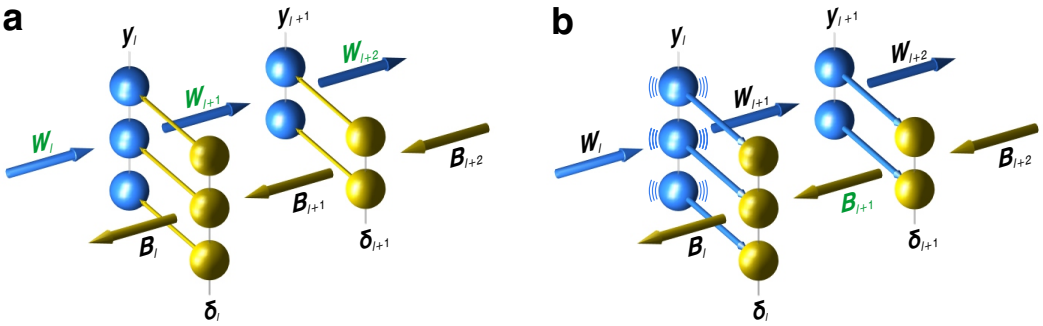

Figure 1: Network modes for weight mirroring. Both panels show the same two-layer section of a network. In both modes, the three neurons in layer l of the forward path (■) send their output signal $\mathbf{y}_l$ through the weight array $\mathbf{W}_{l+1}$ (and other processing shown in equation (1)) to yield the next-layer signal $\mathbf{y}_{l+1}$. And in the feedback path (■), the two neurons in layer $l+1$ send their signal $\boldsymbol{\delta}_{l+1}$ through weight array $\mathbf{B}_{l+1}$ to yield $\boldsymbol{\delta}_l$, as in (3). The figure omits the biases $\mathbf{b}$, nonlinearities ϕ, and, in the top panel, the projections that convey $\mathbf{y}_l$ to the $\boldsymbol{\delta}_l$ cells, allowing them to compute the factor $\phi'(\mathbf{y}_l)$ in equation (3). **a)** In *engaged mode*, cross-projections (▬) convey the feedback signals $\boldsymbol{\delta}$ to the forward-path cells, so they can adjust the forward weights $\mathbf{W}$ using learning rule (4). **b)** In *mirror mode*, one layer of forward cells, say layer l, fires noisily. Its signal $\mathbf{y}_l$ still passes through $\mathbf{W}_{l+1}$ to yield $\mathbf{y}_{l+1}$, but now the blue cross-projections (▬) control firing in the feedback path, so $\boldsymbol{\delta}_l = \mathbf{y}_l$ and $\boldsymbol{\delta}_{l+1} = \mathbf{y}_{l+1}$, and the $\boldsymbol{\delta}_l$ neurons adjust the feedback weights $\mathbf{B}_{l+1}$ using learning rule (7). We call the circuit $\mathbf{y}_l$, $\mathbf{y}_{l+1}$, $\boldsymbol{\delta}_{l+1}$, $\boldsymbol{\delta}_l$ a *weight mirror* because it makes the weight array $\mathbf{B}_{l+1}$ resemble $\mathbf{W}_{l+1}^T$.

In mirror mode, the forward-path neurons in each layer l, carrying the signal $\mathbf{y}_l$, project strongly to layer l of the feedback path — strongly enough that each signal $\boldsymbol{\delta}_l$ of the feedback path faithfully mimics $\mathbf{y}_l$, i.e.

$$\boldsymbol{\delta}_l = \boldsymbol{y}_l \tag{6}$$

Also in mirror mode, those forward-path signals $\mathbf{y}_l$ are noisy. Multiple layers may fire at once, but the process is simpler to explain in the case where they take turns, with just one layer l driving forward-path activity at any one time. In that case, all the cells of layer l fire randomly and independently, so their output signal $\mathbf{y}_l$ has zero-mean and equal variance σ^2. That signal passes through forward weight matrix $\mathbf{W}_{l+1}$ and activation function ϕ to yield $\mathbf{y}_{l+1} = \phi(\mathbf{W}_{l+1}\ \mathbf{y}_l + b_l)$. By equation (6), signals $\mathbf{y}_l$ and $\mathbf{y}_{l+1}$ are transmitted to the feedback path. Then the layer-l feedback cells adjust their

weights $\mathbf{B}_{l+1}$ by Hebbian learning,

$$\Delta \mathbf{B}_{l+1} = \eta_B \; \boldsymbol{\delta}_l \; \boldsymbol{\delta}_{l+1}^T \tag{7}$$

This circuitry and learning rule together constitute the weight mirror.

4.3 Why it works

To see that (7) approximates the transposing rule (5), notice first that

$$\boldsymbol{\delta}_l \; \boldsymbol{\delta}_{l+1}^T = \mathbf{y}_l \; \mathbf{y}_{l+1}^T = \mathbf{y}_l \; \phi(\mathbf{W}_{l+1} \; \mathbf{y}_l + \mathbf{b}_{l+1})^T \tag{8}$$

If we assume, for now, that ϕ is everywhere differentiable, with derivatives everywhere positive, and if we make the variance σ^2 of $\mathbf{y}_l$ small enough that $\mathbf{W}_{l+1} \; \mathbf{y}_l + \mathbf{b}_{l+1}$ falls in a roughly affine range of ϕ, then

$$\phi(\mathbf{W}_{l+1} \; \mathbf{y}_l + \mathbf{b}_{l+1}) \approx \phi'(\mathbf{b}_{l+1}) \; (\mathbf{W}_{l+1} \; \mathbf{y}_l) + \phi(\mathbf{b}_{l+1}) \tag{9}$$

where $\phi'(\mathbf{b}_{l+1})$ is a positive-definite, diagonal matrix. Therefore

$$\boldsymbol{\delta}_l \; \boldsymbol{\delta}_{l+1}^T \approx \mathbf{y}_l \big[\; \mathbf{y}_l^T \; \mathbf{W}_{l+1}^T \; \phi'(\mathbf{b}_{l+1})^T + \phi(\mathbf{b}_{l+1})^T \; \big] \tag{10}$$

and so

$$\begin{aligned} E\big[\Delta \mathbf{B}_{l+1}\big] &\approx \eta_B \left(E\big[\mathbf{y}_l \mathbf{y}_l^T\big] \mathbf{W}_{l+1}^T \phi'(\mathbf{b}_{l+1})^T + E\big[\mathbf{y}_l\big] \phi(\mathbf{b}_{l+1})^T \right) \\ &= \eta_B \, E\big[\mathbf{y}_l \mathbf{y}_l^T\big] \mathbf{W}_{l+1}^T \phi'(\mathbf{b}_{l+1})^T \\ &= \eta_B \, \sigma^2 \phi'(\mathbf{b}_{l+1}) \mathbf{W}_{l+1}^T \end{aligned} \tag{11}$$

Hence the weight matrix $\mathbf{B}_{l+1}$ integrates a teaching signal (7) which is related to $\mathbf{W}_{l+1}^T$ on average by a positive-definite, diagonal matrix $\eta_B \sigma^2 \phi'(\mathbf{b}_{l+1})$. Over time, this integration may drive up the matrix norm $\|\mathbf{B}_{l+1}\|$, but if we add a mechanism to keep the norm small — such as weight decay or synaptic scaling [15, 16] — then (7) makes $\mathbf{B}_{l+1}$ approximately positive-definitely related to $\mathbf{W}_{l+1}^T$.

We get a stronger result if we assume the biases $\mathbf{b}_{l+1}$ are small, or if we suppose that neurons are capable of *bias-blocking* — of closing off their bias currents when in mirror mode, or preventing their influence on the axon hillock. Then

$$E\big[\Delta \mathbf{B}_{l+1}\big] \approx \eta_B \, \sigma^2 \phi'(\mathbf{0}) \mathbf{W}_{l+1}^T \tag{12}$$

So long as all neurons in forward layer $l+1$ have the same activation function, (12) implies that $\mathbf{B}_{l+1}$ will come to approximate a positive *scalar* multiple of $\mathbf{W}_{l+1}^T$.

And with bias-blocking we can also drop the requirement that ϕ have a positive derivative everywhere. Now it need only have a positive derivative near 0 (see Appendix C.2 for a more general formulation that further relaxes the requirements on ϕ).

In one respect the weight mirror resembles difference target propagation [4], because both mechanisms shape the feedback path layer by layer, but target propagation learns layer-wise autoencoders (though see [17]), and uses feedback weights to propagate targets rather than gradients.

5 The Kolen-Pollack algorithm

5.1 Convergence through weight decay

Kolen and Pollack [10] observed that we don't have to transport *weights* if we can transport *changes* in weights. Consider two synapses, W in the forward path and B in the feedback path (written without boldface because for now we are considering individual synapses, not matrices). Suppose W and B are initially unequal, but at each time step t they undergo identical adjustments $A(t)$ and apply identical weight-decay factors λ, so

$$\Delta W(t) = A(t) - \lambda W(t) \tag{13}$$

and

$$\Delta B(t) = A(t) - \lambda B(t) \tag{14}$$

Then $W(t+1)-B(t+1) = W(t)+\Delta W(t)-B(t)-\Delta B(t) = W(t)-B(t)-\lambda[W(t)-B(t)] = (1-\lambda)[W(t)-B(t)] = (1-\lambda)^{t+1}[W(0)-B(0)]$, and so with time, if $0<\lambda<1$, W and B will converge.

But biologically, it is no more feasible to transport weight changes than weights, and Kolen and Pollack do not say how their algorithm might run in the brain. Their flow diagram (Figure 2 in their paper) is not at all biological: it shows weight changes being calculated at one locus and then traveling to distinct synapses in the forward and feedback paths. In the brain, changes to different synapses are almost certainly calculated separately, within the synapses themselves. But it *is* possible to implement Kolen and Pollack's method in a network without transporting weights or weight changes.

5.2 A circuit for Kolen-Pollack learning

The standard, forward-path learning rule (4) says that the matrix $\mathbf{W}_{l+1}$ adjusts itself based on a product of its input vector $\mathbf{y}_l$ and a teaching vector $\boldsymbol{\delta}_{l+1}$. More specifically, each synapse $W_{l+1,ij}$ adjusts itself based on its own scalar input $y_{l,j}$ and the scalar teaching signal $\delta_{l+1,i}$ sent to its neuron from the feedback path.

We propose a reciprocal arrangement, where synapses in the feedback path adjust themselves based on their own inputs and cell-specific, scalar teaching signals from the forward path,

$$\Delta \mathbf{B}_{l+1} = -\eta\, \mathbf{y}_l\, \boldsymbol{\delta}_{l+1}^T \tag{15}$$

If learning rates and weight decay agree in the forward and feedback paths, we get

$$\Delta \mathbf{W}_{l+1} = -\eta_W\, \boldsymbol{\delta}_{l+1}\, \mathbf{y}_l^T - \lambda\, \mathbf{W}_{l+1} \tag{16}$$

and

$$\Delta \mathbf{B}_{l+1} = -\eta_W\, \mathbf{y}_l\, \boldsymbol{\delta}_{l+1}^T - \lambda\, \mathbf{B}_{l+1} \tag{17}$$

i.e.

$$\Delta \mathbf{B}_{l+1}^T = -\eta_W\, \boldsymbol{\delta}_{l+1}\, \mathbf{y}_l^T - \lambda\, \mathbf{B}_{l+1}^T \tag{18}$$

In this network (drawn in Figure 2), the only variables transmitted between cells are the activity vectors $\mathbf{y}_l$ and $\boldsymbol{\delta}_{l+1}$, and each synapse computes its own adjustment locally, but (16) and (18) have the form of the Kolen-Pollack equations (13) and (14), and therefore the forward and feedback weight matrices converge to transposes of each other.

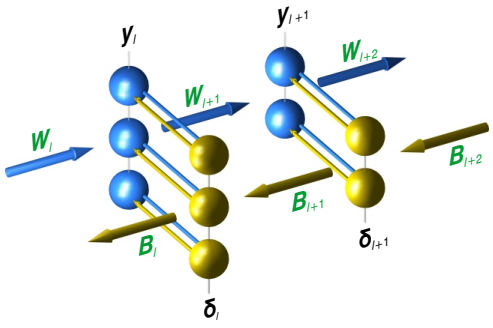

Figure 2: Reciprocal network for Kolen-Pollack learning. There is a single mode of operation. Gold-colored cross-projections (—) convey feedback signals $\boldsymbol{\delta}$ to forward-path cells, so they can adjust the forward weights $\mathbf{W}$ using learning rule (16). Blue cross-projections (—) convey the signals $\mathbf{y}$ to the feedback cells, so they can adjust the feedback weights $\mathbf{B}$ using (17).

We have released a Python version of the proprietary TensorFlow/TPU code for the weight mirror and the KP reciprocal network that we used in our tests; see `github.com/makrout/Deep-Learning-without-Weight-Transport`.

6 Experiments

We compared our weight-mirror and Kolen-Pollack networks to backprop, plain feedback alignment, and the sign-symmetry method [5, 7]. For easier comparison with recent papers on biologically-motivated algorithms [4, 5, 7], we used the same types of networks they did, with convolution [18], batch normalization (BatchNorm) [19], and rectified linear units (ReLUs) without bias-blocking. In most experiments, we used a ResNet block variant where signals were normalized by BatchNorm after the ReLU nonlinearity, rather than before (see Appendix D.3). More brain-like implementations would have to replace BatchNorm with some kind of synaptic scaling [15, 16], ReLU with a bounded function such as rectified tanh, and convolution with non-weight-sharing local connections.

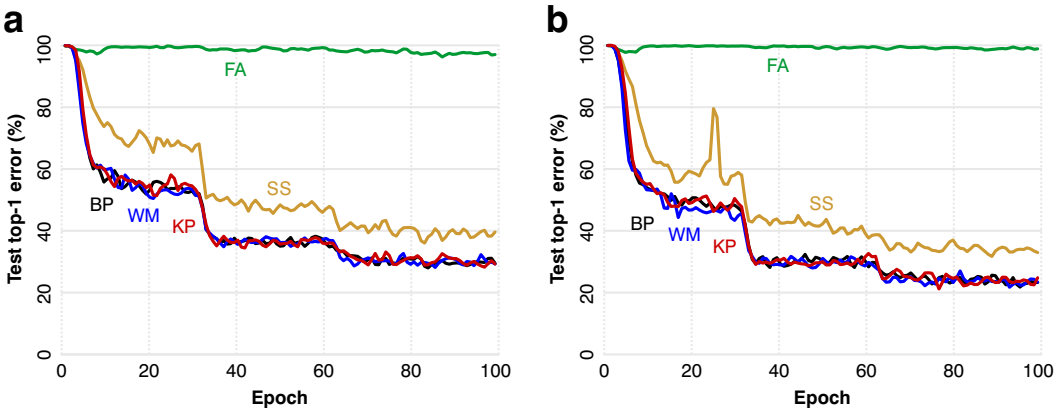

Figure 3: ImageNet results. **a)** With ResNet-18 architecture, the weight-mirror network (— WM) and Kolen-Pollack (— KP) outperformed plain feedback alignment (— FA) and the sign-symmetry algorithm (— SS), and nearly matched backprop (— BP). **b)** With the larger ResNet-50 architecture, results were similar.

Run on the ImageNet visual-recognition task [6] with the ResNet-18 network (Figure 5a), weight mirrors managed a final top-1 test error of $30.2(7)\%$, and Kolen-Pollack reached $29.2(4)\%$, versus $97.4(2)\%$ for plain feedback alignment, $39.2(4)\%$ for sign-symmetry, and $30.1(4)\%$ for backprop. With ResNet-50 (Figure 5b), the scores were: weight mirrors $23.4(5)\%$, Kolen-Pollack $23.9(7)\%$, feedback alignment $98.9(1)\%$, sign-symmetry $33.8(3)\%$, and backprop $22.9(4)\%$. (Digits in parentheses are standard errors).

Sign-symmetry did better in other experiments where batch normalization was applied *before* the ReLU nonlinearity. In those runs, it achieved top-1 test errors of $37.8(4)\%$ with ResNet-18 (close to the 37.91% reported in [7] for the same network) and $32.6(6)\%$ with ResNet-50 (see Appendix D.1 for details of our hyperparameter selection, and Appendix D.3 for a figure of the best result attained by sign-symmetry on our tests). The same change in BatchNorm made little difference to the other four methods — backprop, feedback alignment, Kolen-Pollack, and the weight mirror.

Weight mirroring kept the forward and feedback matrices in agreement throughout training, as shown in Figure 4. One way to measure this agreement is by matrix angles: in each layer of the networks, we took the feedback matrix $\mathbf{B}_l$ and the transpose of the forward matrix, $\mathbf{W}_l^T$, and reshaped them into vectors. With backprop, the angle between those vectors was of course always 0. With weight mirrors (Figure 4a), the angle stayed $< 12^\circ$ in all layers, and $< 6^\circ$ later in the run for all layers except the final one. That final layer was fully connected, and therefore its $\mathbf{W}_l$ received more inputs than those of the other, convolutional layers, making its $\mathbf{W}_l^T$ harder to deduce. For closer alignment, we would have needed longer mirroring with more examples.

The matrix angles grew between epochs 2 and 10 and then held steady at relatively high levels till epoch 32 because during this period the learning rate η_W was large (see Appendix D.1), and mirroring didn't keep the $\mathbf{B}_l$'s matched to the fast-changing $\mathbf{W}_l^T$'s. That problem could also have been solved with more mirroring, but it did no harm because at epoch 32, η_W shrank by 90%, and from then on, the $\mathbf{B}_l$'s and $\mathbf{W}_l^T$'s stayed better aligned.

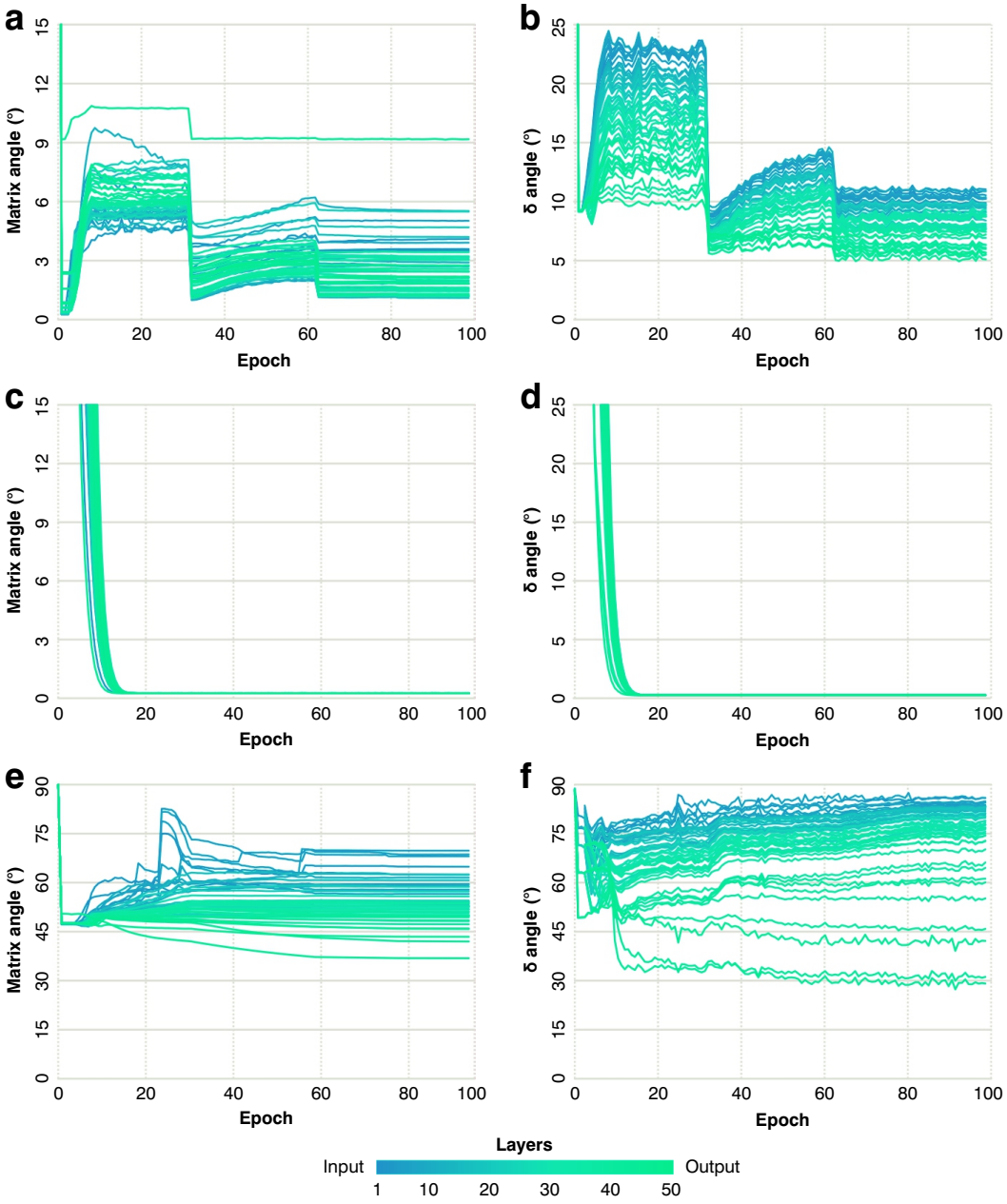

Figure 4: Agreement of forward and feedback matrices in the ResNet-50 from Figure 5b. **a)** Weight mirrors kept the angles between the matrices $\mathbf{B}_l$ and $\mathbf{W}_l^T$ small in all layers, from the input layer (—) to the output (—). **b)** Feedback vectors $\boldsymbol{\delta}_l$ computed by the weight-mirror network were also well aligned with those that would have been computed by backprop. **c, d)** The Kolen-Pollack network kept the matrix and $\boldsymbol{\delta}$ angles even smaller. **e, f)** The sign-symmetry method was less accurate.

We also computed the $\boldsymbol{\delta}$ angles between the feedback vectors $\boldsymbol{\delta}_l$ computed by the weight-mirror network (using $\mathbf{B}_l$'s) and those that would have been computed by backprop (using $\mathbf{W}_l^T$'s). Weight mirrors kept these angles $< 25°$ in all layers (Figure 4b), with worse alignment farther upstream, because $\boldsymbol{\delta}$ angles depend on the accumulated small discrepancies between all the $\mathbf{B}_l$'s and $\mathbf{W}_l^T$'s in all downstream layers.

The Kolen-Pollack network was even more accurate, bringing the matrix and $\boldsymbol{\delta}$ angles to near zero within 20 epochs and holding them there, as shown in Figures 4c and 4d.

The sign-symmetry method aligned matrices and $\boldsymbol{\delta}$'s less accurately (Figures 4e and 4f), while with feedback alignment (not shown), both angles stayed > 80° for most layers in both the ResNet-18 and ResNet-50 architectures.

7 Discussion

Both the weight mirror and the Kolen-Pollack network outperformed feedback alignment and the sign-symmetry algorithm, and both kept pace, at least roughly, with backprop. Kolen-Pollack has some advantages over weight mirrors, as it doesn't call for separate modes of operation and needn't proceed layer by layer. Conversely, weight mirrors don't need sensory input but learn from noise, so they could tune feedback paths in sleep or in utero. And while KP kept matrix and δ angles smaller than WM did in Figure 4, that may not be the case in all learning tasks. With KP, the matrix $\mathbf{B}$ converges to $\mathbf{W}^T$ at a rate that depends on λ, the weight-decay factor in equation (17). A big λ speeds up alignment, but may hamper learning, and at present we have no proof that a good balance can always be found between λ and learning rate η_W. In this respect, WM may be more versatile than KP, because if mirroring ever fails to yield small enough angles, we can simply do more mirroring, e.g. in sleep. More tests are needed to assess the two mechanisms' aptitude for different tasks, their sensitivity to hyperparameters, and their effectiveness in non-convolutional networks and other architectures.

Both methods may have applications outside biology, because the brain is not the only computing device that lacks weight transport. Abstractly, the issue is that the brain represents information in two different forms: some is coded in action potentials, which are energetically expensive but rapidly transmissible to other parts of the brain, while other information is stored in synaptic weights, which are cheap and compact but localized — they influence the transmissible signals but are not themselves transmitted. Similar issues arise in certain kinds of technology, such as application-specific integrated circuits (ASICs). Here as in the brain, mechanisms like weight mirroring and Kolen-Pollack could allow forward and feedback weights to live locally, saving time and energy [20–22].

References

[1] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, 1987.

[2] Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

[3] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7:13276, 2016.

[4] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9390–9400, 2018.

[5] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[6] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[7] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*, 2018.

[8] Dmitry Krotov and John Hopfield. Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, 116(16):7723–7731, 2019.

[9] Jordan Guerguiev, Konrad Kording, and Blake Richards. Spike-based causal inference for weight alignment. *arXiv preprint arXiv:1910.01689*, 2019.

[10] John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1375–1380. IEEE, 1994.

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

[12] Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

[13] Jordan Guergiuev, Timothy P Lillicrap, and Blake A Richards. Biologically feasible deep learning with segregated dendrites. *arXiv preprint arXiv:1610.00161*, 2016.

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

[15] Gina G Turrigiano. The self-tuning neuron: synaptic scaling of excitatory synapses. *Cell*, 135(3):422–435, 2008.

[16] Dhrubajyoti Chowdhury and Johannes W Hell. Homeostatic synaptic scaling: Molecular regulators of synaptic AMPA-type glutamate receptors. *F1000Research*, 7, 2018.

[17] Daniel Kunin, Jonathan M Bloom, Aleksandrina Goeva, and Cotton Seed. Loss landscapes of regularized linear autoencoders. *arXiv preprint arXiv:1901.08168*, 2019.

[18] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[20] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[21] Hyoukjun Kwon, Michael Pellauer, and Tushar Krishna. Maestro: An open-source infrastructure for modeling dataflows within deep learning accelerators. *arXiv preprint arXiv:1805.02566*, 2018.

[22] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *arXiv preprint arXiv:1903.02083*, 2019.

[23] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017.

[24] Richard Naud and Henning Sprekeler. Sparse bursts optimize information transmission in a multiplexed neural code. *Proceedings of the National Academy of Sciences*, 115(27):E6329–E6338, 2018.

[25] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, 2004.

[26] R.j Leigh and D.S. Zee. *The Neurology of Eye Movements*. New York: Oxford University Press, 3rd ed. edition, 1999.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[28] Peter Buchlovsky, David Budden, Dominik Grewe, Chris Jones, John Aslanides, Frederic Besse, Andy Brock, Aidan Clark, Sergio Gómez Colmenarejo, Aedan Pope, et al. Tf-replicator: Distributed machine learning for researchers. *arXiv preprint arXiv:1902.00465*, 2019.

[29] Ilya Sutskever, James Martens, George E Dahl, and Geoffrey E Hinton. On the importance of initialization and momentum in deep learning. *ICML (3)*, 28(1139-1147):5, 2013.

Appendices

A The transposing rule as gradient descent

The learning rule (5) can be expressed as a form of gradient descent,

$$\Delta \mathbf{B} = \eta \, \mathbf{x} \, \mathbf{y}^T = -\eta \, \frac{\partial f}{\partial \mathbf{B}} \tag{19}$$

where

$$f(\mathbf{x}, \mathbf{y}, \mathbf{B}) = -sum(\mathbf{x} \, \mathbf{y}^T \odot \mathbf{B}) = -\sum_{i,j} x_i \, y_j \, B_{ij} \tag{20}$$

This function f is not a *loss* or *objective* function, as it has no minimum for any fixed, non-zero $\mathbf{x}$ and $\mathbf{y}$, and neither is it a quantity we would *wish* to minimize, because it can be pushed farther and farther below zero by making $\mathbf{B}$ larger and larger. But if we combine (5) with weight decay

$$\Delta \mathbf{B} = \eta_B \, \mathbf{x} \, \mathbf{y}^T - \lambda_{WM} \, \mathbf{B} \tag{21}$$

then we do descend the gradient of a loss

$$\mathcal{L} = f + \frac{\lambda_{WM}}{2 \, \eta_B} \, \|\mathbf{B}\|^2 \tag{22}$$

B Computational costs

Weight mirroring is slightly more expensive computationally than is Kolen-Pollack learning. Suppose layers l and $l+1$ of a learning network are fully connected, with n_l and n_{l+1} forward units (and the same numbers of feedback units if they are separate from the forward ones), and let $n = min(n_l, n_{l+1})$. Then for each training example, KP does $n + 4 n_l n_{l+1}$ flops to adjust $\mathbf{B}_{l+1}$ using equation (17). WM does the same number to adjust $\mathbf{B}_{l+1}$ using equation (7) and weight decay. But WM also has to generate a random vector y_l and then perform about $2 n_l n_{l+1}$ flops to compute y_{l+1} from y_l using equation (1), whereas KP uses the same y_l and y_{l+1} that train the forward matrices. In short, WM needs twice as many forward passes as KP does to collect as many training examples for its $\mathbf{B}$ matrices (whether the net is fully-connected or not).

C Biological interpretations

The variables in the weight-mirror and Kolen-Pollack equations can be interpreted physically in several different ways. Here we describe some issues and options:

C.1 Distinct feedback neurons?

Figures 1 and 2 show learning networks where inference signals $\mathbf{y}$ and error signals $\boldsymbol{\delta}$ are carried by distinct sets of neurons. But their equations work just as well if the same neurons carry inference signals in one direction and errors in the other.

If the forward and feedback paths are *distinct* sets of neurons, then our proposed methods call for a one-to-one pairing, connecting each forward-path cell with a partner cell in the feedback path. Such connections might arise during development. We know that very precise and consistent neuronal wiring is found in simple organisms such as C. elegans and in the compound eyes of insects, while in primate cerebellum each Purkinje cell connects with exactly one appropriate climbing fiber. Alternatively, something less than strict one-to-one wiring may suffice for effective learning, and may itself be learned.

Getting a one-to-one correspondence is of course trivial if the *same* neurons make up the forward and feedback paths, though then we face the new problem of signal segregation — explaining how signals $\mathbf{y}$ and δ can flow through the same cells without interfering. Some possibilities are that neurons segregate $\mathbf{y}$ and δ by conveying them with different intracellular messengers or computing them in different parts of the cell [23] or by multiplexing [24], or cells may take turns carrying one or the other signal.

C.2 Zero-mean signals

In equation (11), we provided a rationale for our learning rule (7), but to do it we had to assume that the signal $\mathbf{y}_l$ had a zero mean. That assumption is awkward if we interpret the signals in our equations as firing rates of neurons, because neurons can't have negative rates, and so can't have zero means except by remaining utterly silent. But we can drop the zero-mean requirement if we suppose that neurons convey positive and negative values by modulating about a baseline rate β. For instance, we might have

$$\mathbf{y}_{l+1} = \phi\big(\mathbf{W}_{l+1}(\mathbf{y}_l - \beta) + \mathbf{b}_{l+1}\big) \tag{23}$$

where ϕ is a non-negative activation function and $\mathbf{y}_l - \beta$ means that the same scalar β is subtracted from each element of the signal vector $\mathbf{y}_l$. In engaged mode, forward matrices are adjusted by the learning rule

$$\Delta\mathbf{W}_{l+1} = -\eta_W\, \boldsymbol{\delta}_{l+1}(\mathbf{y}_l - \beta)^T \tag{24}$$

whereas in mirror mode, $\mathbf{y}_l$ fires noisily with a mean of β rather than 0, and $\mathbf{B}_{l+1}$ is adjusted by the rule

$$\Delta\mathbf{B}_{l+1} = \eta_B\, (\boldsymbol{\delta}_l - \beta)(\boldsymbol{\delta}_{l+1} - \beta)^T \tag{25}$$

This baseline parameter β may be built into forward and feedback neurons by the genome or it may be estimated locally, for example by taking the average of the firing rates over a period of mirroring, as we did in our experiments. And it is easy to show that a slight generalization of bias blocking lets us work with any activation function ϕ so long as there is some neighborhood where it is positive and has a positive derivative.

Another way to get positive and negative signals in the brain is to think of each processing unit not as a single neuron but as a group of cells acting in push-pull, some carrying positive signals and others negative [25]. Both mechanisms — baselines and push-pull — operate in the brain, for instance in the vestibulo-ocular reflex [26].

C.3 Multipurpose projections?

To avoid clutter, Figure 1a omitted the cross-projections which convey $\mathbf{y}_l$ to the feedback cells, allowing them to compute the factor $\phi'(\mathbf{y}_l)$ in equation (3). Figure 1b does show cross-projections from forward to feedback cells, carrying the same signal, $\mathbf{y}_l$, but having a different effect on the target cells, setting $\boldsymbol{\delta}_l = \mathbf{y}_l$. We may interpret these two sets of projections — the ones omitted from Figure 1a and the ones drawn as thin blue arrows in 1b — as two distinct sets of axons carrying the same signal, or as a single set of axons whose effects on their targets differ in the two modes. Maybe these axons form two types of synapses, some onto ionotropic receptors and some onto metabotropic, or maybe some switch in intracellular signaling within the feedback cells makes them respond differently to identical signals. Similar issues arise in Figure 2, where blue cross-projections convey the signals $\mathbf{y}$ for use in both (3) and (17).

C.4 Multilayer mirroring

In Figure 1b and accompanying text, we assumed that just one forward layer $\mathbf{y}_l$ was noisy, and just one feedback array $\mathbf{B}_{l+1}$ was adjusted, at any one time. Why not adjust all the $\mathbf{B}$'s at once? The problem is, when we adjust $\mathbf{B}_{l+1}$ we need a zero-mean (or β-mean), uncorrelated, equal-variance signal $\mathbf{y}_l$, which drives $\mathbf{y}_{l+1}$. But the resulting $\mathbf{y}_{l+1}$ generally will not be zero-mean, uncorrelated, or equal-variance, and so may not be effective at driving $\mathbf{B}_{l+2}$ toward $\mathbf{W}_{l+2}^T$. We can of course apply noise simultaneously to every *second* layer — for instance to $\mathbf{y}_2$, $\mathbf{y}_4$, $\mathbf{y}_6$, etc. — and adjust as many as half the $\mathbf{B}$'s at any one moment, so the mirroring does not really have to proceed one layer at a time. And there may be other options in networks with batch normalization or synaptic scaling [19]. Those mechanisms tend to keep all the forward signals approximately zero-mean and equal-variance (though not uncorrelated), and in that case it may be possible to adjust all the $\mathbf{B}$'s at once, driving the entire network with a single noisy input vector $\mathbf{y}_l$, though we haven't tested that idea here.

D Experimental details

D.1 Architecture and training

We ran our experiments using 18- and 50-layer deep-residual networks ResNet-18 and ResNet-50 [27]. These networks consisted of sequences of sub-blocks, each made up of two (ResNet-18) or three (ResNet-50) convolutional layers in series. In parallel with these layers was a shortcut connection whose output was added to the output from the convolutional layers. The output of the network passed through a final fully-connected layer followed by a softmax.

For the sign-symmetry algorithm, we carried out grid searches of the learning rate over the range [0.01, 2.0] while running training out to 140 epochs to ensure convergence. We found that 0.5 gave the lowest top-1 errors with both ResNet-18 and ResNet-50, and so we used that value for all sign-symmetry experiments. Otherwise, all hyperparameters in all algorithms (except those for mirror mode) were taken from [28], including forward-path Nesterov momentum [29] 0.9 and a weight decay factor (L2 regularizer) λ of 10^{-4}. We used TF-Replicator [28] to distribute training across 32 TPU v2 workers, for a total mini-batch of 2048 images. And we applied the annealing schedule from [28], i.e. η_W grew linearly over the first 6 epochs (or over epochs 3 to 8 for weight-mirror networks, see below), and shrank 10-fold after epochs 32, 62, and 82.

D.2 Mirroring

Each weight-mirror network spent its first two epochs entirely in mirror mode, bringing its initial, random weights into alignment. Thereafter, it did a small amount of mirroring after each mini-batch of engaged-mode learning. It mirrored layer-wise: it created a new mini-batch of noisy activity in layer *l* (independent Gaussian signals with zero mean and unit variance across the 2048 examples in the mirroring mini-batch) and it sent those signals through the convolutional layer and then the ReLU function. It computed the means of the post-ReLU outputs across the mini-batch and subtracted them to give zero-mean outputs in each layer.

As in equation (21), the covariance matrix of these zero-mean signals was estimated by multiplying them and averaging over the mini-batch. In convolutional layers, because of weight sharing, each weight connected multiple sets of inputs and outputs, and so we estimated the covariance associated with any given weight by averaging the estimated covariances over the pairs of inputs and outputs it connected.

We used these covariance estimates to train the feedback weights, as in (21), with a learning-rate factor η_B of 0.1 and a weight decay λ_{WM} of 0.5.

D.3 Batch normalization

The weight mirror and Kolen-Pollack learned to match feedback matrices $\mathbf{B}_l$ to forward matrices $\mathbf{W}_l$, but didn't try to reproduce the batch normalization parameter vectors $\boldsymbol{\mu}$ or $\boldsymbol{\sigma}$ used in the forward path. In fact no $\mathbf{B}_l$ could have mirrored the combined effects of $\mathbf{W}_l$, $\boldsymbol{\mu}$, and $\boldsymbol{\sigma}$ in our convolutional networks, because the $\mathbf{B}_l$ matrices had the same convolutional, weight-sharing structure as the $\mathbf{W}_l$'s did — a structure which $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ ignored. Therefore we simply passed the scaling parameter $\boldsymbol{\sigma}$ from the forward to the feedback path ($\boldsymbol{\mu}$ was not needed). This transfer involved very little information — just one scalar variable per feedback neuron — and could be avoided if we replaced convolution by more biological local connections *without* weight sharing.

In most of our experiments we applied batch normalization after the activation function, but the sign-symmetry method learned slightly better the other way, with normalization applied *before* the activation, as in [27]. We gave the numerical results for both cases in Section 6, but here we provide also a figure of the best result achieved by sign-symmetry in our tests, its $32.6(6)\%$ final error with the ResNet-50 architecture:

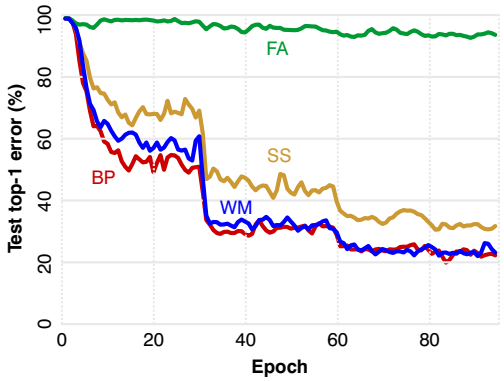

Figure 5: ResNet-50 ImageNet results with batchnorm applied before ReLU.

E Pseudo Code

Algorithm 1 Weight Mirrors

procedure WEIGHT MIRRORS($network$, $data$)
  ▷ $network$ has L layers
  for each epoch **do**
    for each batch = $(\mathbf{y}_0,\ \mathbf{y}^*) \in$ data **do**
    ▷ Engaged mode
      compute the batch prediction $\mathbf{y}_L$ ▷ forward pass
      $\boldsymbol{\delta}_L = \mathbf{y}_L - \mathbf{y}^*$ ▷ compute error
      for layer l **from** L-1 **to** 0 **do**
        $\mathbf{W}_{l+1} = (1-\lambda)\,\mathbf{W}_{l+1} - \eta_W\,\boldsymbol{\delta}_{l+1}\,\mathbf{y}_l^T$ ▷ equation (4), with weight decay
        $\mathbf{b}_{l+1} = (1-\lambda)\,\mathbf{b}_{l+1} - \eta_W\,\boldsymbol{\delta}_{l+1}$
        $\boldsymbol{\delta}_l = \phi'(\mathbf{y}_l)\,\mathbf{B}_{l+1}\,\boldsymbol{\delta}_{l+1}$ ▷ compute error gradients using $\mathbf{B}$, equation (3)
      end for
    ▷ Mirror mode
      for layer l **from** 1 **to** L-1 **do**
        sample $\mathbf{y}_l \sim \mathcal{N}(\mu,\,\sigma^2)$ ▷ ideally zero-mean, small-variance
        $\mathbf{y}_{l+1} = \phi(\mathbf{W}_l\,\mathbf{y}_l + \mathbf{b}_l)$
        $\boldsymbol{\delta}_l = \mathbf{y}_l - \bar{\mathbf{y}}_l$ ▷ subtract batch average
        $\boldsymbol{\delta}_{l+1} = \mathbf{y}_{l+1} - \bar{\mathbf{y}}_{l+1}$ ▷ forward cells drive feedback cells
        $\mathbf{B}_{l+1} = (1-\lambda_{WM})\mathbf{B}_{l+1} + \eta_B\,\boldsymbol{\delta}_l\,\boldsymbol{\delta}_{l+1}^T$ ▷ equations (7) and (21)
      end for
    end for
  end for
end procedure

Algorithm 2 Kolen-Pollack algorithm

procedure KOLEN-POLLACK ALGORITHM($network$, $data$)
    ▷ $network$ has L layers
    for each epoch **do**
        for each batch = $(\mathbf{y}_0,\ \mathbf{y}^*) \in$ data **do**
            compute the batch prediction $\mathbf{y}_L$ ▷ forward pass
            $\boldsymbol{\delta}_L = \mathbf{y}_L - \mathbf{y}^*$ ▷ compute error
            for layer l **from** L-1 **to** 0 **do**
                $\mathbf{W}_{l+1} = (1 - \lambda)\, \mathbf{W}_{l+1} - \eta_W\, \boldsymbol{\delta}_{l+1}\, \mathbf{y}_l^T$ ▷ equation (16)
                $\mathbf{b}_{l+1} = (1 - \lambda)\, \mathbf{b}_{l+1} - \eta_W\, \boldsymbol{\delta}_{l+1}$
                $\mathbf{B}_{l+1} = (1 - \lambda)\, \mathbf{B}_{l+1} - \eta_B\, \mathbf{y}_l\, \boldsymbol{\delta}_{l+1}^T$ ▷ equation (17)
                $\boldsymbol{\delta}_l = \phi'(\mathbf{y}_l)\, \mathbf{B}_{l+1}\, \boldsymbol{\delta}_{l+1}$ ▷ compute error gradients using $\mathbf{B}$, equation (3)
            end for
        end for
    end for
end procedure