

Random feedback weights support learning in deep neural networks

Timothy P. Lillicrap^{1*}, Daniel Cownden², Douglas B. Tweed^{3,4}, Colin J. Akerman¹

¹Department of Pharmacology, University of Oxford, Oxford, United Kingdom

²Centre for the Study of Cultural Evolution, Stockholm University, Stockholm, Sweden

³Departments of Physiology and Medicine, University of Toronto, Toronto, Canada

⁴Centre for Vision Research, York University, Toronto, Canada

*To whom correspondence should be addressed:

timothy.lillicrap@pharm.ox.ac.uk

colin.akerman@pharm.ox.ac.uk

Abstract

The brain processes information through many layers of neurons. This deep architecture is representationally powerful^{1,2,3,4}, but it complicates learning by making it hard to identify the responsible neurons when a mistake is made^{1,5}. In machine learning, the backpropagation algorithm¹ assigns blame to a neuron by computing exactly how it contributed to an error. To do this, it multiplies error signals by matrices consisting of all the synaptic weights on the neuron's axon and farther downstream. This operation requires a precisely choreographed transport of synaptic weight information, which is thought to be impossible in the brain^{1,6,7,8,9,10,11,12,13,14}. Here we present a surprisingly simple algorithm for deep learning, which assigns blame by multiplying error signals by *random* synaptic weights. We show that a network can learn to extract useful information from signals sent through these random feedback connections. In essence, the network learns to learn. We demonstrate that this new mechanism performs as quickly and accurately as backpropagation on a variety of problems and describe the principles which underlie its function. Our demonstration provides a plausible basis for how a neuron can be adapted using error signals generated at distal locations in the brain, and thus dispels long-held assumptions about the algorithmic constraints on learning in neural circuits.

Networks in the brain compute via many layers of interconnected neurons^{15,16}. To work properly neurons must adjust their synapses so that the network’s outputs are appropriate for its tasks. A longstanding mystery is how upstream synapses (e.g. the synapse between α and β in Fig. 1a) are adjusted on the basis of downstream errors (e.g. e in Fig. 1a). In artificial intelligence this problem is solved by an algorithm called backpropagation of error¹. Backprop works well in real-world applications^{17,18,19}, and networks trained with it can account for cell response properties in some areas of cortex^{20,21}. But it is biologically implausible because it requires that neurons send each other precise information about large numbers of synaptic weights — i.e. it needs *weight transport*^{1,6,7,8,12,14,22} (Fig. 1a, b). Specifically, backprop multiplies error signals e by the matrix W^T , the transpose of the forward synaptic connections, W (Fig. 1b). This implies that feedback is computed using knowledge of all the synaptic weights W in the forward path.

For this reason, current theories of biological learning have turned to simpler schemes such as reinforcement learning²³, and “shallow” mechanisms which use errors to adjust only the final layer of a network^{4,11}. But reinforcement learning, which delivers the same reward signal to each neuron, is slow and scales poorly with network size^{5,13,24}. And shallow mechanisms waste the representational power of deep networks^{3,4,25}.

Here we describe a new deep-learning algorithm that is as fast and accurate as backprop, but much simpler, avoiding all transport of synaptic weight information. This makes it a mechanism the brain could easily exploit. It is based on three insights: (i) The feedback weights need not be exactly W^T . In fact, any matrix B will suffice, so long as on average,

$$e^T W B e > 0 \tag{1}$$

where e is the error in the network’s output (Fig. 1a). Geometrically, this means the teaching signal sent by the matrix, $B e$, lies within 90° of the signal used by backprop, $W^T e$, i.e. B pushes the network in roughly the same direction as backprop would. (ii) Even if the network doesn’t have this property initially, it can acquire it through learning. The obvious option is to adjust B to make equation (1) true, but (iii) another possibility is to do the same by adjusting W . We will show this can be done very simply, even with a fixed, random B (Fig. 1c).

We first demonstrate that this mechanism works for a variety of tasks, and then explain why it works. For clarity we consider a three-layer network of linear neurons (see

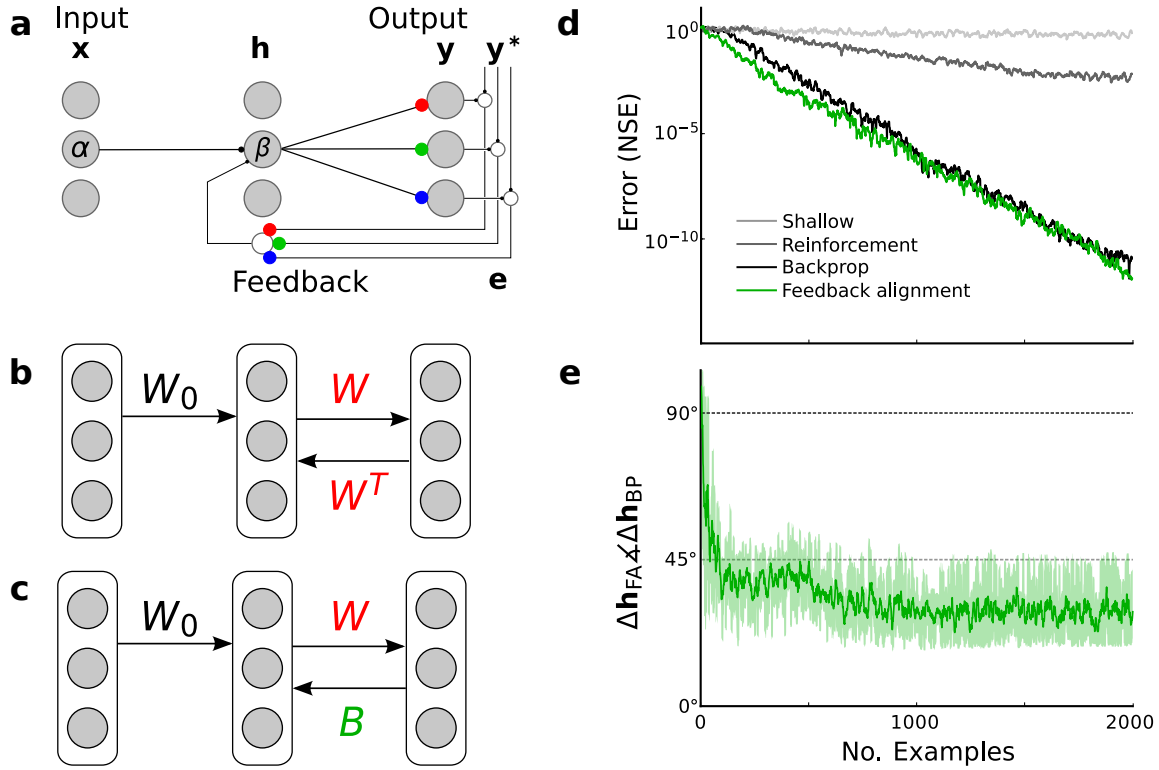


Figure 1: Random feedback weights can deliver useful teaching signals to preceding layers of a neural network. **a**, The backprop learning algorithm is powerful, but requires biologically implausible transport of individual synaptic weight information. For backprop, neurons must know each other's synaptic weights, e.g. the three coloured synapses on the feedback cell at bottom must have weights equal to those of the corresponding coloured synapses on three cells in the forward path. On a computer it is simple to use the synaptic weights in both forward and backward computations, but synapses in the brain communicate information unidirectionally. **b**, Implemented in a computer, backprop computes how to change hidden unit activities by multiplying the error vector, $e = y^* - y$, by the transpose matrix of the forward weights, i.e., $\Delta h_{BP} = W^T e$. **c**, Our *feedback alignment* method uses the counterintuitive observation that learning is still effective if W^T is replaced by a matrix of fixed random weights, B , so that $\Delta h_{FA} = B e$. **d**, Four algorithms learn to mimic a linear function: 'shallow' learning (light gray), reinforcement learning (dark gray), backprop (black), and feedback alignment (green). NSE is normalized squared error. **e**, Angle between the hidden-unit update vector prescribed by feedback alignment and that prescribed by backprop, i.e., $\Delta h_{FA} \angle \Delta h_{BP}$. Error bars are two standard deviations for a sliding window of 10 examples.

Methods). The network’s output is $\mathbf{y} = W\mathbf{h}$, where \mathbf{h} is the hidden-unit activity vector, given by $\mathbf{h} = W_0\mathbf{x}$, where \mathbf{x} is the input to the network. W_0 is the matrix of synaptic weights from \mathbf{x} to \mathbf{h} and W is the weights from \mathbf{h} to \mathbf{y} . The network learns to approximate a linear function, T (for “target”). Its goal is to reduce the squared error, or loss, $\mathcal{L} = \frac{1}{2}\mathbf{e}^T\mathbf{e}$, where the error $\mathbf{e} = \mathbf{y}^* - \mathbf{y} = T\mathbf{x} - \mathbf{y}$.

We trained the network using four algorithms (Fig. 1d). If only the output weights, W , are adjusted, as in shallow methods, then the loss hardly decreases. If a fast variant of reinforcement learning is used to adjust the hidden weights, W_0 , then there is some progress but it is slow. In contrast, backprop sends the loss rapidly towards zero. It adjusts the hidden-unit weights according to the gradient of the loss, $\Delta W_0 \propto (\partial\mathcal{L}/\partial W_0) = (\partial\mathcal{L}/\partial\mathbf{h})(\partial\mathbf{h}/\partial W_0) = -(W^T\mathbf{e})\mathbf{x}^T$. Thus, backprop adjusts the hidden units according to the vector: $\Delta\mathbf{h}_{\text{BP}} = W^T\mathbf{e}$. Here and throughout $\Delta\mathbf{h}$ denotes the update sent to the hidden layer, rather than the change in the hidden units. Our new algorithm adjusts W in the same way as backprop ($\Delta W \propto (\partial\mathcal{L}/\partial W) = -\mathbf{e}\mathbf{h}^T$), but for $\Delta\mathbf{h}$ it uses a much simpler formula, which needs no information about W or any other synapses but instead sends \mathbf{e} through a fixed random matrix B ,

$$\Delta\mathbf{h} = B\mathbf{e} \tag{2}$$

This algorithm, which we call *feedback alignment* drives down the loss as quickly as backprop does (Fig. 1d).

The network learns how to learn — it gradually discovers how to use B , which then allows effective modification of the hidden units. At first, the updates to the hidden layer are not helpful, but they quickly improve by an implicit feedback process that alters W so that $\mathbf{e}^T W B \mathbf{e} > 0$. To reveal this, we plot the angle between the hidden-unit updates prescribed by feedback alignment and backprop, $\Delta\mathbf{h}_{\text{FA}} \angle \Delta\mathbf{h}_{\text{BP}}$ (Fig. 1e; see Methods). Initially the angles average about 90° . But they soon shrink, as the algorithm begins to take steps that are closer to those of backprop. This alignment of the $\Delta\mathbf{h}$ ’s implies that B has begun to act like W^T . And because B is fixed, the alignment is driven by changes in the forward weights W . In this way, random feedback weights come to transmit useful teaching signals to neurons deep in the network.

Feedback alignment learning also solves nonlinear, real-world problems. We ran it on a benchmark classification problem (Fig. 2a), learning to recognize handwritten digits¹⁷ (see Methods). On this task, backprop brings the mean error on the test set to 2.4% (average of $n=20$ runs). Feedback alignment learns as quickly as backprop, reaches 2.1% mean error ($n=20$), and develops similar feature detectors (Supplemen-

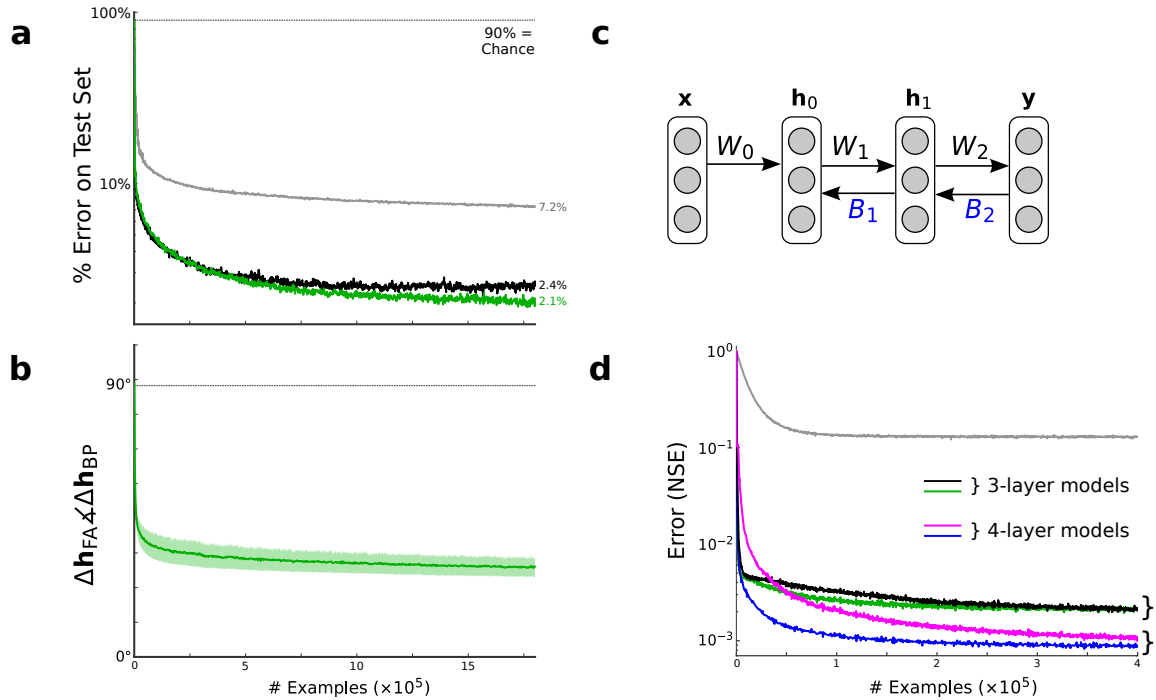


Figure 2: Feedback alignment solves nonlinear, real-world problems. **a**, A 784–1000–10 network of logistic units learns to recognize handwritten digits. Representative performance curves for backprop (black), feedback alignment (green), and shallow learning (light grey) on 10,000 test images. **b**, Angle between the hidden-unit update made by feedback alignment and that prescribed by backprop, i.e., $\Delta \mathbf{h}_{FA} \angle \Delta \mathbf{h}_{BP}$. Error bars are one standard deviation around the time-averaged mean. **c**, Feedback alignment can train deeper layers via random weights, e.g. B_1 and B_2 . **d**, Normalized squared error curves, each an average over 20 trials, for a nonlinear function-approximation task; three-layer network trained with shallow learning (grey), backprop (black), and feedback alignment (green); four-layer network trained with backprop (magenta) and feedback alignment (blue).

tary Fig. S1). As measured by the angle, $\Delta \mathbf{h}_{FA} \angle \Delta \mathbf{h}_{BP}$, feedback alignment quickly learns to use the random weights to transmit useful error information to the hidden units (Fig. 2b). Even when we randomly remove 50% of the elements of the W and B matrices, so that neurons in \mathbf{h} and \mathbf{y} have a 25% chance of reciprocal connection, feedback alignment still matches backprop (2.4% mean error; $n=20$).

Some tasks are better performed by networks with more than one hidden layer, but for that we need a learning algorithm that can exploit the extra power of a deeper network^{2,3,4} (Fig. 2c). Backprop assigns blame to a neuron by taking into account *all* of its downstream synapses. Thus, the update for the first hidden layer in a four-layer network (Fig. 2c) is $\Delta \mathbf{h}_{BP}^0 = W_1^T ((W_2^T \mathbf{e}) \circ \mathbf{h}'_1)$, where \circ is element-wise multiplication,

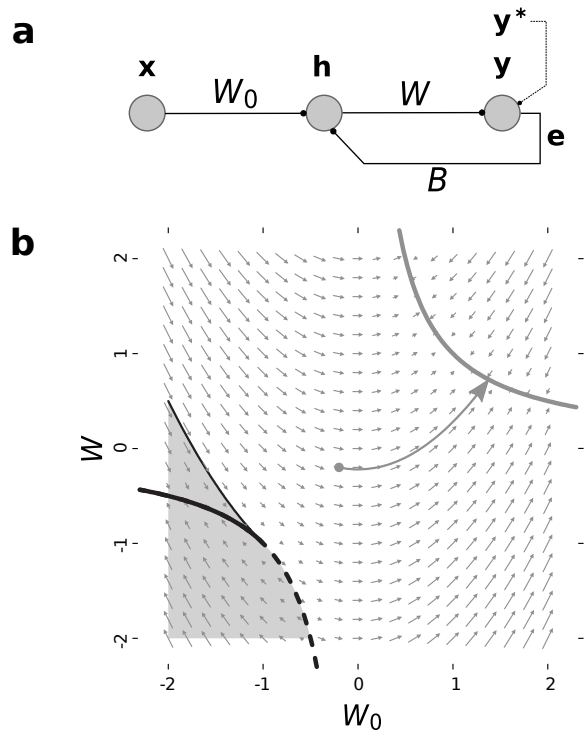


Figure 3: Network dynamics underlying feedback alignment. **a**, Three-neuron network learning to match a linear function, $\mathbf{y}^* = T\mathbf{x}$, with $T = 1$ and B ‘randomly’ chosen to be 1. **b**, Vector flow field (small arrows) demonstrates the evolution of W_0 and W during feedback alignment. Thick lines are solution manifolds (i.e. $W_0 W = 1 = T$) where: $eWB e > 0$ (grey), $eWB e < 0$ (black), or unstable solutions (dashed black). There is a small region of weight space (shaded grey) from which the system travels to the “bad” hyperbola at lower left, but this is simply avoided by starting near 0. Large arrow traces the trajectory for the initial condition $W_0 = W = 0$.

and \mathbf{h}'_1 is the derivative of the \mathbf{h}_1 activation function. With feedback alignment the update is instead, $\Delta \mathbf{h}_{\text{FA}}^0 = B_1((B_2 \mathbf{e}) \circ \mathbf{h}'_1)$, where B_1 and B_2 are random matrices (Fig. 2c). On a non-linear function-fitting task (see Methods), both backprop (t-test, $n = 20, p = 3 \times 10^{-12}$; Fig. 2d) and feedback alignment (t-test, $n = 20, p = 9 \times 10^{-13}$; Fig. 2d) deliver better performance with a four-layer network than with a three-layer network. Thus, the algorithm builds useful feature detectors in the deepest layers of a network by relaying errors via random connections. This flexibility makes feedback alignment more plausible for the brain: learning proceeds even when error vectors are indiscriminately broadcast via random feedback weights to multiple layers of cells.

Why does feedback alignment work? Its mechanism arises from certain novel network

dynamics. To explain them, we consider a minimal network with just one linear neuron in each layer (Fig. 3a, see Methods). We visualize (Fig. 3b) how the network’s two weights, W_0 and W , evolve when the feedback weight B is set to 1. The flow field shows that the system moves along parabolic paths. From most starting points the network weights travel to the hyperbola at upper right (Fig. 3b). This hyperbola is a set of stable equilibria solutions where $W > 0$ and therefore $e^T W B e > 0$ for all e – that is, equation (1) is satisfied, which means W has evolved so that the feedback matrix B is delivering useful teaching signals.

In higher dimensions we can identify conditions under which feedback alignment is guaranteed to reduce errors to zero (Supplementary Proof 1). Importantly, the proof holds for cases where the error can reach zero only if B transmits useful information to the hidden neurons. The proof also demonstrates that high-dimensional analogues of the pattern of parabolic paths seen in the minimal network (Fig. 3a, b), also hold for networks with large numbers of units. Indeed, the proof hinges on the fact that feedback alignment yields the relation $BW + W^T B^T = W_0 W_0^T + C$, where C is a constant, i.e. the left-hand side is a quadratic function of W_0 .

Feedback alignment updates do not converge with backprop (Fig. 1e, Fig. 2b), superficially suggesting that they are merely suboptimal approximations of $\Delta \mathbf{h}_{\text{BP}}$. Further analysis shows this view is too simplistic. Our proof says that weights W_0 and W evolve to equilibrium manifolds, but simulations (Fig. 4) and analytic results (Supplementary Proof 2) hint at something more specific: that when the weights begin near 0, feedback alignment encourages W to act like a local pseudoinverse of B around the error manifold. This fact is important because if B were *exactly* W^+ (the Moore-Penrose pseudoinverse of W), then the network would be performing Gauss-Newton optimization (Supplementary Proof 3). We call this update rule for the hidden units *pseudobackprop* and denote it by $\Delta \mathbf{h}_{\text{PBP}} = W^+ e$. Experiments with the linear network show that the angle, $\Delta \mathbf{h}_{\text{FA}} \angle \Delta \mathbf{h}_{\text{PBP}}$ quickly becomes smaller than $\Delta \mathbf{h}_{\text{FA}} \angle \Delta \mathbf{h}_{\text{BP}}$ (Fig. 4b, c; see Methods). In other words feedback alignment, despite its simplicity, displays elements of second-order learning.

In the 1980’s, new artificial network learning algorithms promised to provide insight into brain function¹. But the most powerful class of algorithms use error signals tailored to each neuron and seemed impossible to implement in the brain because they required weight transport^{6,7}. More-plausible algorithms have been devised^{9,10,13,14,22,26,27,28,29,30}, but these either fall far short of backprop’s speed or call for a lot of additional processing^{10,11,12,13,22}. In marked contrast, the mechanism developed here is much *simpler* than backprop, but still matches its speed and accuracy. Feedback alignment dispels the central assumption of previous neuron-specific algorithms - that error information

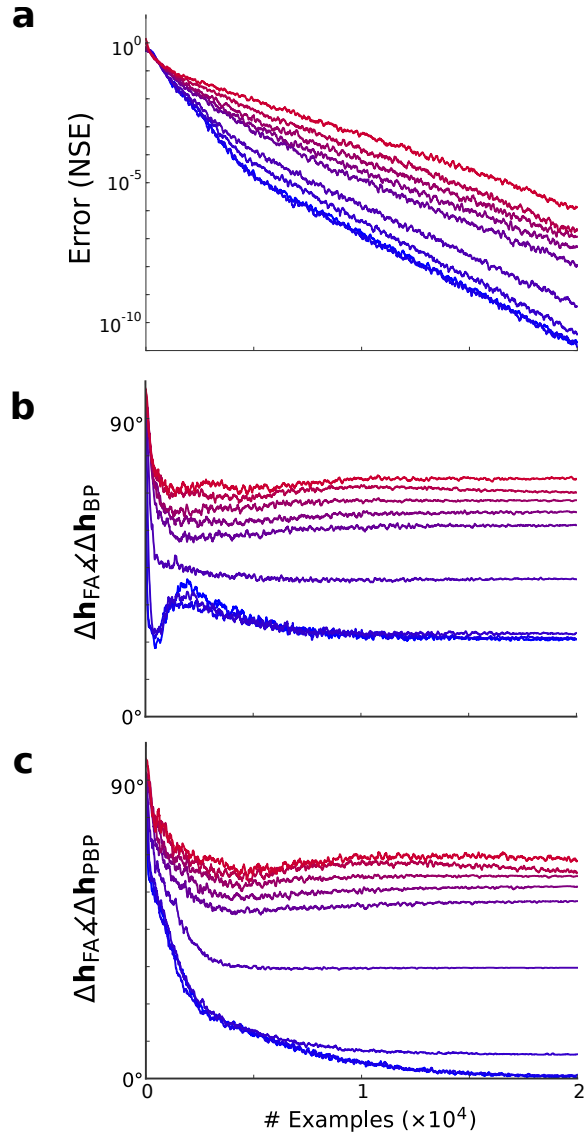


Figure 4: If W_0 and W start small then W learns to act like a local pseudoinverse of B . **a**, Each trace is a single run of feedback alignment learning with the elements of W_0 and W drawn uniformly from $[-\omega, \omega]$, where $\omega = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.125, 0.15, 0.2, 0.25]$, corresponding to blue through red, respectively. Loss is normalized squared error (NSE). **b-c**, Angle between the hidden unit changes prescribed by feedback alignment versus backprop (panel b) and versus pseudobackprop (panel c).

must be precisely tailored for each neuron. Our work shows that it is far easier than previously thought to send neuron-specific teaching signals through a deep network: all you need is random feedback connections. Thus, the principles underlying feedback alignment learning are compatible with many brain circuits in which reciprocal feedback connections exist, such as occur within, and between regions of the neocortex^{15,16}. This makes it an attractive basis for understanding various forms of learning in deep networks, including the integration of sensory information and motor adaptation processes. Finally, feedback alignment may offer new opportunities to integrate neuroscience with recent advances in machine learning which have highlighted the power of deep architectures^{2,3,18,19,25}.

Methods Summary

We trained feedforward networks on three tasks. In all cases the goal was to minimize the square of the error, $L = (1/2)\mathbf{e}^T\mathbf{e}$, where $\mathbf{e} = \mathbf{y}^* - \mathbf{y}$ is the difference between the desired and actual output. *Task (1)*: A 30–20–10 linear network learned to approximate a linear function, T . Input/output training pairs were produced via, $\mathbf{y}^* = T\mathbf{x}$, with $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu} = 0, \Sigma = I)$. Output weights were adjusted via, $\Delta W \propto \mathbf{e}^T\mathbf{h}$. Hidden weights were adjusted according to: (a) backprop: $\Delta W_0 \propto (W^T\mathbf{e})\mathbf{x}^T = \Delta\mathbf{h}_{\text{BP}}\mathbf{x}^T$, (b) feedback alignment: $\Delta W_0 \propto (B\mathbf{e})\mathbf{x}^T = \Delta\mathbf{h}_{\text{FA}}\mathbf{x}^T$ where the elements of B were drawn from the uniform distribution over $[-0.5, 0.5]$, (c) a variant of reinforcement learning called node perturbation^{23,24}. We chose the learning rate η for each algorithm via manual search³¹ in order to optimize learning speed. *Task (2)*: A 784–1000–10 network with standard sigmoidal hidden and output units (i.e., $\sigma(x) = 1/(1+\exp(-x))$) was trained to classify images of handwritten digits, 0–9. Each unit had an adjustable input bias. Standard 1-hot representation was used to code desired output. The network was trained with 60,000 images from the standard MNIST dataset¹⁷, and performance was measured as the percentage of errors made on a held aside test set of 10,000 images. Both algorithms used a learning rate of, $\eta = 10^{-3}$, and weight decay, $\alpha = 10^{-6}$. Parameter updates were the same as those used in the linear case, but with $\Delta\mathbf{h}_{\text{FA}} = (B\mathbf{e}) \circ \boldsymbol{\sigma}'$, where \circ is element-wise multiplication and $\boldsymbol{\sigma}'$ is the derivative of the output activations. *Task (3)*: A 30–20–10 and 30–20–10–10 network were trained to approximate the output of a 30–20–10–10 target network. All three networks had $\tanh(\cdot)$ hidden units, linear output units, and an adjustable input bias for each unit. Input/output training pairs were produced via, $\mathbf{y}^* = W_2 \tanh(W_1 \tanh(W_0\mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2$, with $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu} = 0, \Sigma = I)$. The angle between two vectors, e.g. $\mathbf{a} \angle \mathbf{b}$, was computed as: $\theta = \cos^{-1}(\|\mathbf{a}^T\mathbf{b}\|/(\|\mathbf{a}\| \cdot \|\mathbf{b}\|))$.

Acknowledgements This project was supported by the European Community’s Seventh Framework Programme (FP7/2007-2013), NSERC, and the Swedish Research Council (grant 2009-2390).

Author Contributions T.L., C.A. conceived the project; T.L., D.C., D.T. ran the simulations; T.L., D.C., D.T. wrote the Supplementary Information; T.L., D.C., D.T., C.A. wrote the manuscript.

Full Methods

Comparing the performance of different learning algorithms is notoriously tricky. Indeed, the no-free-lunch theorems remind us that any comparison tells only one part of the story³². We have used straightforward methodological approaches, allowing us to focus on the novel aspects of our observation. Thus, fixed learning rates, and simple methods for selecting hyperparameters have been used throughout. Performance may be improved by more complicated schemes, but our simple approach ensures a clear view of the fundamental ideas of the main text.

Task (1) Linear function approximation: The target linear function T mapped vectors in a 30 dimensional space to 10 dimensions. The elements of T were drawn at random, i.e. uniformly from the range $[-1, 1]$. Once chosen, the target matrix was fixed, so that each algorithm tried to learn the same function. The sequence of data points learned on was also fixed for each algorithm. That is, the dataset $\mathcal{D} = \{(x_1, y_1^*), \dots, (x_N, y_N^*)\}$ was generated once according to: $\mathbf{y}_i^* = T\mathbf{x}_i$, with $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu} = 0, \Sigma = I)$. The elements of the network weight matrices, W_0, W , were initialized by drawing uniformly from the range $[-0.01, 0.01]$. For node perturbation reinforcement learning we optimized the scale of the perturbation variance by manual search^{23,24,25}. Simulations for Fig. 4 were essentially the same as those for Fig. 1 except that the learning rate for the runs was set to $\eta = 10^{-3}$.

Task (2) MNIST dataset: We manually optimized the initial scale of the W_0 and W weight matrices and the learning rate, η , to give good performance with the backprop algorithm. That is, the elements of W_0 and W were drawn from the uniform distribution over $[-\omega, \omega]$ where ω was selected by looking at final performance on the test set. The same scale for the forward matrices and learning rate were used with the feedback alignment algorithm. In a similar fashion, the elements of the B matrix were drawn from a uniform distribution over $[-\beta, \beta]$ with β chosen by manual search. Empirically, we found that many scale parameters for B worked well. In practice it required 5

restarts to select the scale used for B in the simulations presented here. Once a scale for B was chosen, a new B matrix was drawn for each of the $n=20$ simulations. In the experiments where 50% of the weights in W and B were removed, we drew the remaining elements from the same uniform distributions as above (i.e. using ω and β). Learning was terminated after the same number of iterations for each simulation and for each algorithm. We selected the termination time by observing when backprop began to overfit on the test set.

Task (3) Nonlinear function approximation: The parameters for the 30–20–10–10 target network, $T(\cdot)$, were chosen at random and then fixed for all of the corresponding simulations. We sought a parameter regime for the target network in which backprop gained an unambiguous advantage from having an additional hidden layer. The sequence of data points learned on was fixed for each algorithm. The dataset $\mathcal{D} = \{(x_1, y_1^*), \dots, (x_N, y_N^*)\}$ was generated once according to: $y_i^* = T(\mathbf{x}_i)$, with $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu} = 0, \Sigma = I)$. A new set of random matrices were chosen for each of the $n=20$ simulations, both for the forward synaptic weights and biases and the backward matrices. 5000 data points were held aside as a test-set. Network performance was evaluated as the normalized squared error on these points. Hidden unit updates with feedback alignment were $\Delta \mathbf{h}_{\text{FA}}^1 = (B_2 \mathbf{e})$, and $\Delta \mathbf{h}_{\text{FA}}^0 = B_1((B_2 \mathbf{e}) \circ \mathbf{h}'_1)$ for the deeper hidden layer, where \mathbf{h}'_1 is the derivative of the \mathbf{h}_1 activities and \circ is element-wise multiplication. The elements of B_1 and B_2 were drawn from uniform distributions with scale parameters selected manually. Learning was terminated after the same number of iterations for each simulation and for each algorithm. We selected the termination time by observing when backprop made negligible gains on the training error.

Flow Field in Fig. 3: To produce the flow fields in Fig. 3, we computed the *expected* updates made by feedback alignment, rendering deterministic dynamics. Details for the deterministic dynamics can be found in Supplementary Proof 1.

Computational details:

All learning experiments were run using custom built code in Python with the Numpy library. MNIST experiments were sped up using a GPU card with the Cudamat and Gnumpy libraries^{33,34}. The dynamics in Fig. 3b were simulated with custom-built code in Matlab.

References

- [1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.
- [2] G.E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.
- [3] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [4] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In B. Léon, C. Olivier, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, 2007.
- [5] H. Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40:1063–1073, 2003.
- [6] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11:23–63, 1987.
- [7] Francis Crick. The recent excitement about neural networks. *Nature*, 337(12):129–132, 1989.
- [8] David G. Stork. Is backpropagation biologically plausible? In *International Joint Conference on Neural Networks*, volume 2, pages 241–246, 1989.
- [9] Pietro Mazzoni, Richard A. Anderson, and Michael I. Jordan. A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences*, 88:4433–4437, 1991.
- [10] Xiaohui Xie and H. Sebastian Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural Computation*, 15(2):441–454, 2003.
- [11] Alexandre Pouget and Lawrence H. Snyder. Computational approaches to sensorimotor transformations. *Nature Neuroscience*, 3:1192–1198, 2000.
- [12] Kenneth D. Harris. Stability of the fittest: organizing learning through retroaxonal signals. *Trends in Neurosciences*, 31(3):130–136, 2008.
- [13] Robert Urbanczik and Walter Senn. Reinforcement learning in populations of spiking neurons. *Nature Neuroscience*, 12(3):250–252, 2009.

- [14] L.V. Chintala and D.B. Tweed. Adaptive optimal control without weight transport. *Neural Computation*, 24(6):1487–1518, 2012.
- [15] D.C. Van Essen, C.H. Anderson, and D.J. Felleman. Information processing in the primate visual system: An integrated systems perspective. *Science*, 255:419–423, 1992.
- [16] R.J. Douglas and K.A. Martin. Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.*, 27:419–451, 2004.
- [17] Yann LeCun, Leon Bottou, Yoshua. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, November 1998.
- [18] D.C. Ciresan, U. Meier, L.M. Gambardella, and J. Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.
- [19] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, 1207.0580, 2012.
- [20] David Zipser and Richard A. Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331:679–684, 1988.
- [21] Timothy P. Lillicrap and Stephen H. Scott. Preference distributions of primary motor cortex neurons reflect control solutions optimized for limb biomechanics. *Neuron*, 77(1):168–179, 2013.
- [22] M.N. Abdelghani, T.P. Lillicrap, and D.B. Tweed. Sensitivity derivatives for flexible sensorimotor learning. *Neural Computation*, 20:2085–2111, 2008.
- [23] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.
- [24] Justin Werfel, Xiaohui Xie, and H. Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. *Neural Computation*, 17:2699–2718, 2005.
- [25] Pascal Lamblin and Yoshua Bengio. Important gains from supervised fine-tuning of deep architectures on large labeled sets. In *Neural Information Processing Systems*, volume 24th of *Deep Learning and Unsupervised Feature Learning Workshop*, pages 1–8, 2010.

- [26] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.
- [27] G.E. Hinton and J.L. McClelland. Learning representations by recirculation. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 358–366, New York, 1988.
- [28] John F. Kolen and Jordan B. Pollack. Back-propagation without weight transport. In *IEEE World Congress on Computational Intelligence*, volume 3, pages 1375–1380, Orlando, Florida, 1994.
- [29] Randall C. O’Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8:895–938, 1996.
- [30] Konrad P. Körding and Peter König. Supervised and unsupervised learning with two sites of synaptic integration. *Journal of Computational Neuroscience*, 11:207–215, 2001.
- [31] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.
- [32] D.H. Wolpert. The lack of *a priori* distinction between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.
- [33] Volodymyr Mnih. Cudamat: A cuda-based matrix class for python. Technical Report UTML TR 2009-004, University of Toronto, November 2009.
- [34] Tieleman. Gnumpy: an easy way to use GPU boards in python. Technical Report UTML TR 2010-002, University of Toronto, July 2010.

Supplementary Information for “Random feedback weights support learning in deep neural networks”

Supplementary Figures.

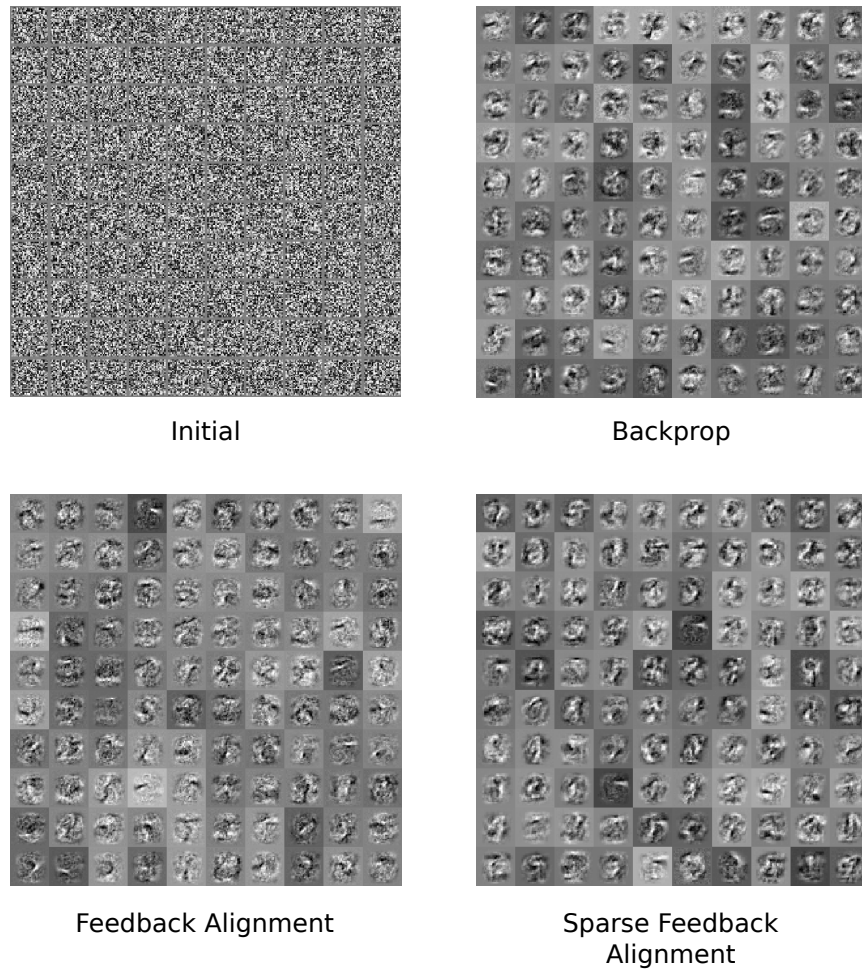


Figure S1: Receptive fields for 100 randomly selected hidden units shown at the beginning of learning (top left) and for the three learning variants discussed in the main text. Grey scale indicates the strength of connection from each of 28×28 pixels in MNIST images (white denotes strong positive, black denotes strong negative).

Introduction to analytic results.

Here we present three analytical results which provide insight into the efficacy of feedback alignment. The first result gives conditions under which feedback alignment is guaranteed to reduce the error of a network function to 0. The second result demonstrates that the backprop algorithm can be modified in a simple way to implement the second order Gauss-Newton method of error minimization, as contrasted with gradient descent method employed by standard backprop. The third result hints at a possible connection between feedback alignment and this Gauss-Newton modification of backprop.

Proof #1: Condition for alignment to reduce error to zero.

Although the empirical results presented in the main text suggest that feedback alignment is effective across a broad range of problems, we cannot, at this point, sharply delineate the space of learning problems where feedback alignment is guaranteed to work. We are, however, able to establish a class of problems where feedback alignment is guaranteed to reduce training error to 0. Importantly this class of problems contains cases where useful modifications must be made to downstream synaptic weights to achieve this error reduction. Thus, this theorem establishes that alignment does indeed succeed in transmitting useful error information to neurons deep within the network.

We consider a linear network which generates output \mathbf{y} , from input \mathbf{x} according to

$$\mathbf{h} = A\mathbf{x} \tag{3}$$

$$\mathbf{y} = W\mathbf{h} \tag{4}$$

For each data point \mathbf{x} presented to the network, the desired output, $\tilde{\mathbf{y}}$, is given by a linear transformation T so that $\tilde{\mathbf{y}} = T\mathbf{x}$, (T for target). Our goal is to modify the elements of A and W , so that the network is functionally equivalent to T .

Some comments on notation. Vectors \mathbf{x} , \mathbf{h} , \mathbf{y} , etc. are column vectors, and we use standard matrix multiplication throughout. For example $\mathbf{x}^T\mathbf{x}$ is the inner product of \mathbf{x} with itself (resulting in a scalar) and $\mathbf{x}\mathbf{x}^T$ is the outer product of \mathbf{x} with itself (resulting in a matrix). For brevity and clarity the matrices of synaptic weights referred to as W_0 and W in the main text are here referred to simply as A and W respectively. When referring to the specific elements of A or W , we take A_i^j to be the weight from the i^{th}

input element to the j^{th} hidden element, and similarly we take W_j^k to be the weight from the j^{th} hidden element to the k^{th} output element.

Importantly, the transport of error problem still applies even for a linear network, with a linear target function T , provided the number of output units is less than the number of hidden units which is less than the number of input units, i.e. $n_o < n_h < n_i$. In this case the null space of A (those input vectors which A maps to zero) must be a subspace of the null space of T if the network function is to perfectly match the target function. The probability of a randomly initialized A having this property is effectively zero. Thus, if alignment is able to reduce error to zero, we can conclude that useful modifications have been made to A . Presumably, such modifications are only possible if useful information concerning the errors is employed when modifying A . In this section we prove that transmitting errors to hidden neurons via a fixed arbitrary matrix, B , provides sufficiently useful information for updating A , and reducing error to zero.

For convenience we define:

$$E := T - WA, \quad (5)$$

so that our error is $e = Ex$. Then the parameter updates can be written as

$$\Delta W = \eta E \mathbf{x} \mathbf{x}^T A^T \quad (6)$$

$$\Delta A = \eta B E \mathbf{x} \mathbf{x}^T. \quad (7)$$

Here, η is a small positive constant referred to as the learning rate.

Instead of modifying the parameters A and W after experiencing a single training pair $(\mathbf{x}, T\mathbf{x})$, it is possible to expose the network to many training examples, and then make a single parameter change proportional to the average of the parameter changes prescribed by each training pair. Learning in this way is referred to as batch-learning. In the limit as batch size becomes large the change in the parameters becomes deterministic and proportional to the expected change from a data point.

$$\Delta W = \eta [E \mathbf{x} \mathbf{x}^T A^T] \quad (8)$$

$$\Delta A = \eta [B E \mathbf{x} \mathbf{x}^T] \quad (9)$$

Here $[\cdot]$, denotes the expected value of a random variable. Under the assumption that the elements of \mathbf{x} are i.i.d. standard normal random variables, i.e. mean 0 and standard deviation 1, then $[\mathbf{x} \mathbf{x}^T] = I$. Here and throughout I denotes an identity matrix. Thus, under this normality assumption, in the limit as batch size becomes

large the learning dynamics simplify to

$$\Delta W = \eta EA^T \quad (10)$$

$$\Delta A = \eta BE. \quad (11)$$

In the limit as the learning rate, η becomes small these discrete time learning dynamics described converge to the continuous time dynamical system

$$\dot{W} = EA^T \quad (12)$$

$$\dot{A} = BE. \quad (13)$$

We will work within the context of this continuous time dynamical system when proving theorem 1.

Throughout the proof of theorem 1 we will use the following relation.

$$BW + W^T B^T = AA^T + C \quad (14)$$

To see why equation 14 holds note that if we multiply equation 12 by B on the left and multiply equation 13 by A^T on the right, we have

$$\dot{A}A^T = B\dot{W} \quad (15)$$

$$\int \dot{A}A^T dt = \int B\dot{W} dt + C_1. \quad (16)$$

Transposing this we have

$$\int A\dot{A}^T dt = \int \dot{W}^T B^T dt + C_1^T. \quad (17)$$

Then since

$$\int \dot{A}A^T dt = \int A\dot{A}^T dt = \frac{1}{2}AA^T + C \quad (18)$$

equation 14 follows. Note that $C = C_1 + C_1^T$, so C is symmetric and constant.

We are now in a position to state and prove theorem 1.

Theorem 1. *Given the learning dynamics*

$$\dot{W} = EA^T \quad (19)$$

$$\dot{A} = BE, \quad (20)$$

and assuming that the constant C in equation 14 is zero and that the matrix B satisfies

$$B^+B = I \quad (21)$$

then

$$\lim_{t \rightarrow \infty} E = 0. \quad (22)$$

Some notes on the conditions of the theorem. Here and throughout B^+ denotes the Moore-Penrose pseudoinverse of B , hereafter referred to simply as the pseudoinverse. The condition $B^+B = I$ holds when the columns of B are linearly independent, and B has at least as many rows as columns, i.e. $n_o \leq n_h$. Note that if the elements of B are chosen uniformly at random then the columns of B will be linearly independent with probability 1. The condition $C = 0$ is met when $AA^T = BW + W^TB^T$. While there are many initializations of W , A and B which satisfy this condition, the only way to ensure that the $C = 0$ condition is satisfied for all possible B is for W and A to be initialized as zero matrices.

Proof. Our proof is loosely inspired by Lyapunov's method, and makes use of Barbalat's lemma. Consider the quantity

$$V := \text{tr}(BEE^TB^T). \quad (23)$$

We will use Barbalat's lemma to show that $\dot{V} \rightarrow 0$.

Lemma 1 (Barbalat's Lemma). *If V satisfies:*

1. V is lower bounded,
2. \dot{V} is negative semi-definite,
3. \dot{V} is uniformly continuous in time, which is satisfied if \ddot{V} is finite,

then $\dot{V} \rightarrow 0$ as $t \rightarrow \infty$.

Because B and E are real valued V is equivalent to $\|BE\|^2$. Here and throughout $\|\cdot\|$ refers to the Frobenius norm. Consequently V is bounded below by zero, and so satisfies the first condition of lemma 1.

Lemma 2. \dot{V} is negative semi-definite.

$$\frac{d}{dt} \text{tr}(BEE^T B^T) = \text{tr}(B\dot{E}E^T B^T + BE\dot{E}^T B^T) \quad (24)$$

$$= \text{tr}(B\dot{E}E^T B^T) + \text{tr}(BE\dot{E}^T B^T) \quad (25)$$

$$= 2\text{tr}(B\dot{E}E^T B^T) \quad (26)$$

$$= 2\text{tr}(B(-\dot{W}A - W\dot{A})E^T B^T) \quad (27)$$

$$= -2\text{tr}(BEA^T AE^T B^T) - 2\text{tr}(BWBEE^T B^T) \quad (28)$$

Now,

$$2\text{tr}(BWBEE^T B^T) = \text{tr}(BWBEE^T B^T) + \text{tr}(BWBEE^T B^T) \quad (29)$$

$$= \text{tr}(BWBEE^T B^T) + \text{tr}(BEE^T B^T W^T B^T) \quad (30)$$

$$= \text{tr}(BWBEE^T B^T) + \text{tr}(W^T B^T BEE^T B^T) \quad (31)$$

$$= \text{tr}(AA^T (BEE^T B^T)) \quad (32)$$

$$= \text{tr}(A^T BEE^T B^T A) \quad (33)$$

which gives us that

$$\frac{d}{dt} \text{tr}(BEE^T B^T) = -2\text{tr}(BEA^T AE^T B^T) - \text{tr}(A^T BEE^T B^T A) \leq 0 \quad (34)$$

since each of these terms is of the form $\text{tr}(XX^T)$, i.e. the inner product of a vector with itself.

Lemma 3. *A is bounded.*

Consider

$$s := \text{tr}(AA^T). \quad (35)$$

Then

$$\dot{s} = 2\text{tr}(BEA^T) \quad (36)$$

$$= 2\text{tr}(BTA^T - BWAA^T) \quad (37)$$

$$= 2\text{tr}(BTA^T) - \text{tr}(AA^T AA^T). \quad (38)$$

Now AA^T is an $n_h \times n_h$ symmetric matrix and hence diagonalizable, therefore

$$s \leq n_h \lambda \quad (39)$$

where λ is the dominant eigenvalue of AA^T . Then

$$\text{tr}(AA^T AA^T) = \|AA^T\| \quad (40)$$

$$\geq \lambda^2 \quad (41)$$

$$\geq \left(\frac{s}{n_h}\right)^2. \quad (42)$$

It follows that

$$\dot{s} \leq 2\text{tr}(BTA^T) - \left(\frac{s}{n_h}\right)^2. \quad (43)$$

Using the Cauchy-Schwarz inequality we have that

$$\text{tr}(BTA^T)^2 \leq \text{tr}(AA^T) \cdot \text{tr}(BTT^T B^T) = s\|BT\|^2, \quad (44)$$

so that when $s > \|BT\|^2$, then $\text{tr}(BTA^T) \leq s$. Therefore

$$\dot{s} < 2s - \frac{s^2}{n_h^2} \quad (45)$$

when $s > \|BT\|^2$. This implies that

$$\dot{s} < 0 \quad (46)$$

when $s > \|BT\|^2$ and $s > 2n_h$. We can conclude that

$$s \leq \|BT\|^2 + 2n_h \quad (47)$$

for all time.

Lemma 4. \ddot{V} is bounded.

Differentiating equation 34 we have that

$$\ddot{V} = -4\text{tr}(B\dot{E}A^T AE^T B^T) - 4\text{tr}(BE\dot{A}^T AE^T B^T) - 2\text{tr}(\dot{A}^T BEE^T B^T A) - 2\text{tr}(A^T B\dot{E}E^T B^T A) \quad (48)$$

$$\begin{aligned} &= 4\text{tr}(BEA^T AA^T AE^T B^T) + 4\text{tr}(BWBEA^T AE^T B^T) - 4\text{tr}(BEBEAE^T B^T) \\ &\quad - 2\text{tr}(BEBEE^T B^T A) + 2\text{tr}(A^T BEA^T AE^T B^T A) + 2\text{tr}(A^T BWBEE^T B^T A) \end{aligned} \quad (49)$$

Thus \dot{V} can be expressed in terms of the traces of products of the matrices B , E , A , and BW , and the transposes of these matrices. B is constant so it is bounded, V is bounded below by zero, and $\dot{V} \leq 0$, so V must converge to some value, implying the E is bounded. Lemma 3 shows that A is bounded. Recall that $AA^T = BW + W^T B^T$, and so A being bounded implies that BW and $W^T B^T$ are also bounded. Taken together we have that \dot{V} is bounded.

Thus the conditions of lemma 1 hold and in the limit as $t \rightarrow \infty$, $\dot{V} \rightarrow 0$. Since both addends of \dot{V} have the same sign, in the limit both must be identically zero. In particular $\text{tr}(BEA^T AE^T B^T) = 0$, therefore $BEA^T = 0$. Here and for the remainder of this proof when use W , A , T and E to refer to the value of these matrices in the limit as $t \rightarrow \infty$. Since B is constant we have,

$$EA^T = 0. \quad (50)$$

Recall that $\dot{W} = EA^T$, and so W is constant. Together with B being constant this implies that $AA^T = WB + B^T W^T$ is also constant. By definition, $BEA^T = BTA^T - BWAA^T$. Recall that $BEA^T = 0$, and that B , W and AA^T are all constant, and so BTA^T must also be constant. Note that $\dot{A}^T = E^T B^T$, so a constant BTA^T implies that $BTE^T B^T = 0$. Then we have

$$0 = BTE^T B^T = B^+ T E^T B^T (B^+)^T = TE^T = ET^T. \quad (51)$$

By definition $EE^T = ET^T - EA^T W^T$, and since both addends are zero $EE^T = 0$. Thus $\text{tr}(EE^T) = \|E\|^2 = 0$ and that E is identically zero. \square

Proof #2: Gauss-Newton modification of backprop

Here we will show that replacing the transpose matrix, W^T , with the Moore-Penrose pseudoinverse matrix, W^+ , in the backprop algorithm renders an update rule which approximates Gauss-Newton optimization. That is, the pseudoinverse of the forward matrix, W^+ , not only satisfies the first condition from the main text, i.e. $e^T W W^+ e > 0$, it prescribes second order updates for the hidden units.

The Gauss-Newton method is a way of minimizing squared error: it finds the vector x^* that minimizes a scalar valued function $L(x)$ of the form $L(x) = \frac{1}{2} e(x)^T e(x)$. It does this by starting with a guess of the value x^* and iteratively improving this guess. When L has this quadratic form its second derivative, with respect to x , or Hessian L_{xx} is

$e_x^T e_x + e^T e_{xx}$. When e is small this is close to $e_x^T e_x$. Therefore the Δx prescribed by Newton's method, $-L_{xx}^{-1} L_x^T$, is roughly $-(e_x^T e_x)^{-1} e_x^T e$, which may be written as, $e_x^+ e$, where e_x^+ is the Moore-Penrose inverse of e_x .

Now suppose we have a 3-layer network with input signal x , weight matrices A and W , monotonic squashing function σ , hidden-layer activity vector $h = \sigma(Ax)$, and linear output cells with activity,

$$y = Wh = W\sigma(Ax) \quad (52)$$

If we want to adjust h using the Gauss-Newton method, the formula is

$$\Delta h_{\text{GN}} = -e_h^+ e = -W^+ e \quad (53)$$

Most learning networks don't adjust activity vectors like h but rather synaptic weight matrices like A and W . Computing the Gauss-Newton adjustment to A is complicated, but a good approximation is obtained by replacing W^T with W^+ in the backprop formula. That is, backprop says

$$\begin{aligned} \Delta A_j^i \text{ BP} &= -\eta \sum_k (\partial L / \partial e^k) (\partial e^k / \partial A_j^i) = -\eta \sum_k e^k \partial y^k / \partial A_j^i \\ &= -\eta \sum_k e^k \partial (\sum_l W_l^k h^l) / \partial A_j^i = -\eta \sum_k e^k \sum_l W_l^k \partial h^l / \partial A_j^i \\ &= -\eta \sum_k e^k \sum_l W_l^k D\sigma^l \partial (\sum_m A_m^l x^m) / \partial A_j^i = -\eta \sum_k e^k \sum_l W_l^k D\sigma \partial A_j^i x^j / \partial A_j^i \\ &= -\eta \sum_k e^k \sum_l W_l^k D\sigma^l \delta^{il} x^j = -\eta \sum_k e^k W_l^k D\sigma^i x^j \\ &= -\eta \sum_k e^k W_k^{T^i} D\sigma^i x^j \end{aligned} \quad (54)$$

where δ^{il} is the Kronecker delta and $D\sigma^i$ is the derivative of the i 'th element of $\sigma(Ax)$ with respect to its argument, the i 'th element of Ax .

Replacing W^T by W^+ in the last line of equation 54, we get what we will call the *pseudobackprop* adjustment:

$$\Delta A_j^i \text{ PBP} = -\eta \sum_k e^k W_k^{+i} D\sigma^i x^j \quad (55)$$

This adjustment yields a change in \mathbf{h} that approximates the Gauss-Newton one, $\Delta\mathbf{h}_{\text{GN}}$ from equation 53. To see this, compute the first order approximation to the change in \mathbf{h} ,

$$\begin{aligned}\Delta\mathbf{h}_{\text{PBP}}^i &= D\boldsymbol{\sigma}^i \sum_j \Delta A_j^i \text{PBP} \mathbf{x}^j + o((x^j)^2)) \\ &\approx -\eta D\boldsymbol{\sigma}^i \sum_j \sum_k \mathbf{e}^k W_k^{+j} D\boldsymbol{\sigma}^i \mathbf{x}^j \mathbf{x}^j\end{aligned}\quad (56)$$

$$= -\eta (D\boldsymbol{\sigma}^i)^2 \sum_j (\mathbf{x}^j)^2 \sum_k \mathbf{e}^k W_k^{+i} \quad (57)$$

$$= \eta (D\boldsymbol{\sigma}^i)^2 \sum_j (\mathbf{x}^j)^2 \Delta\mathbf{h}_{\text{GN}}^i \quad (58)$$

That is, each element of the pseudobackprop (PBP) alteration to \mathbf{h} approximates the Gauss-Newton adjustment times a positive number. And that positive number is 1 if we choose $\eta = 1/(D\boldsymbol{\sigma}^i)^2 \mathbf{x}^T \mathbf{x}$. If instead we want a constant η then we can choose one that keeps $\Delta\mathbf{h}_{\text{PBP}}^i \leq \Delta\mathbf{h}_{\text{GN}}^i$, so we don't step too far.

In the context of training an artificial network pseudobackprop may be of little interest. The pseudoinverse is expensive to compute, and computational resources can be better spent either by simply taking more steps using the transpose matrix, or by using other, more efficient, second order methods. Pseudobackprop does, however, bear upon the results of the main text. We find experimentally that the alignment algorithm encourages W to act like B^+ , so that B begins to act like W^+ on the error vectors. Thus alignment may be understood as an approximate implimentation of psuedobackprop.

Proof #3: B acts like the pseudoinverse of W

Here we will prove that, under fairly restrictive conditions, feedback alignment prescribes hidden unit updates which are in the same direction as those prescribed by psuedobackprop, i.e. $\Delta\mathbf{h}_{\text{FA}} \angle \Delta\mathbf{h}_{\text{PBP}} = 0$. Again we take a linear network which generates output \mathbf{y} , from input \mathbf{x} according to

$$\mathbf{h} = A\mathbf{x} \quad (59)$$

$$\mathbf{y} = W\mathbf{h}. \quad (60)$$

We consider the dynamics of the parameters for this network when it is trained on a single input-output pair, $(\mathbf{x}, \tilde{\mathbf{y}})$, using the forward alignment algorithm.

The dynamics of the network parameters under this training regime are

$$W_{t+1} = W_t + \Delta W_t \quad (61)$$

$$A_{t+1} = A_t + \Delta A_t, \quad (62)$$

with

$$\Delta W = \eta_W e \mathbf{h}^T \quad (63)$$

$$\Delta A = \eta_A B e \mathbf{x}^T. \quad (64)$$

Here, as in proof #1, B is a random, fixed, matrix of full rank. η_W and η_A are small positive learning rates.

Because we will only present the network with a single input, \mathbf{x} , we have that

$$\begin{aligned} \mathbf{h}_{t+1} &= A_{t+1} \mathbf{x} \\ &= (A_t + \Delta A_t) \mathbf{x} \\ &= \mathbf{h}_t + \eta_A B e \mathbf{x}^T \mathbf{x} \\ &= \mathbf{h}_t + \eta_h B e. \end{aligned} \quad (65)$$

Here, $\eta_h = \mathbf{x}^T \mathbf{x} \eta_A$. For a judicious choice of η_A , namely $\eta_A = \eta_W / (\mathbf{x}^T \mathbf{x})$, we have $\eta_h = \eta_W = \eta$. For this choice of η_A it suffices to consider the simpler dynamics

$$W_{t+1} = W_t + \Delta W_t \quad (66)$$

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta \mathbf{h}_t \quad (67)$$

with

$$\Delta W = \eta e \mathbf{h}^T \quad (68)$$

$$\Delta \mathbf{h} = \eta B e. \quad (69)$$

We now establish a lemma concerning these simplified dynamics.

Lemma 5. *In the special case of W and A initialized to zero at every time step there is a scalar s_h such that*

$$\mathbf{h} = s_h B \tilde{\mathbf{y}} \quad (70)$$

and a scalar s_w such that

$$W = s_w \tilde{\mathbf{y}} (B \tilde{\mathbf{y}})^T. \quad (71)$$

Proof. In the first time step, when $\mathbf{h} = 0$ and $W = 0$, the conditions 70 and 71 are trivially satisfied with $s_h = 0$ and $s_w = 0$. We note that when conditions 70 and 71 hold we have that

$$\mathbf{y} = W\mathbf{h} = s_w s_h \tilde{\mathbf{y}}(B\tilde{\mathbf{y}})^T(B\tilde{\mathbf{y}}) = s_y \tilde{\mathbf{y}}. \quad (72)$$

Here $s_y := s_w s_h (B\tilde{\mathbf{y}})^T(B\tilde{\mathbf{y}})$. Now,

$$\mathbf{e} = \tilde{\mathbf{y}} - \mathbf{y} = \tilde{\mathbf{y}} - s_y \tilde{\mathbf{y}} = (1 - s_y) \tilde{\mathbf{y}}. \quad (73)$$

Then

$$\Delta W = \eta \mathbf{e} \mathbf{h}^T = \eta (1 - s_y) s_h \tilde{\mathbf{y}}(B\tilde{\mathbf{y}})^T \quad (74)$$

and

$$\Delta \mathbf{h} = \eta B \mathbf{e} = \eta (1 - s_y) B \tilde{\mathbf{y}}. \quad (75)$$

This yeilds

$$s_h^{t+1} = s_h^t + \eta (1 - s_y) \quad (76)$$

and

$$s_w^{t+1} = s_w^t + \eta (1 - s_y) s_h^t. \quad (77)$$

By induction we can conclude that equations 70 and 71 hold for every time step. \square

With this lemma we are now able to state and prove theorem 2.

Theorem 2. *Under the same conditions as Lemma 5, for the simplified dynamics described in equations 66 through 69 we have that the hidden unit updates prescribed by the the forward alignment algorithm, $\Delta_{\text{FA}} \mathbf{h}$, are always a positive scalar multiple of the hidden unit updates prescribed by the pseudobackprop algorithm, $\Delta_{\text{PBP}} \mathbf{h}$. That is*

$$\Delta_{\text{FA}} \mathbf{h} = s \Delta_{\text{PBP}} \mathbf{h} \quad (78)$$

where s is a positive scalar.

Proof. By lemma 5 we have that $W = s_w \tilde{\mathbf{y}}(B\tilde{\mathbf{y}})^T$, with s_w a positive scalar, and that $\mathbf{e} = (1 - s_y) \tilde{\mathbf{y}}$, with $(1 - s_y)$ a positive scalar. Thus, since $\Delta_{\text{FA}} \mathbf{h} = \eta (1 - s_y) B \tilde{\mathbf{y}}$ (equation 75) and $\Delta_{\text{FA}} \mathbf{h} = \eta (1 - s_y) W^+ \tilde{\mathbf{y}}$ it suffices to show that

$$s B \tilde{\mathbf{y}} = (\tilde{\mathbf{y}}(B\tilde{\mathbf{y}})^T)^+ \tilde{\mathbf{y}}, \quad (79)$$

with s a positive scalar. We show this by manipulating the left hand side of equation 79.

$$\begin{aligned}
 (\tilde{\mathbf{y}}(B\tilde{\mathbf{y}})^T)^+ \tilde{\mathbf{y}} &= (B\tilde{\mathbf{y}})^{T+} \tilde{\mathbf{y}}^+ \tilde{\mathbf{y}} \\
 &= B^{T+} \tilde{\mathbf{y}}^{T+} \tilde{\mathbf{y}}^+ \tilde{\mathbf{y}} \\
 &= B^{T+} \tilde{\mathbf{y}}^{T+} \tilde{\mathbf{y}}^T \tilde{\mathbf{y}}^{T+} \tilde{\mathbf{y}} \\
 &= B^{T+} \tilde{\mathbf{y}}^{T+} \\
 &= (B\tilde{\mathbf{y}})^{T+} \\
 &= sB\tilde{\mathbf{y}}
 \end{aligned} \tag{80}$$

Here $s = (B\tilde{\mathbf{y}})^T (B\tilde{\mathbf{y}})$.

□