

Layer-skipping connections facilitate training of layered networks using equilibrium propagation.

Jimmy Gammell^{1,*}, Sonia Buckley¹, Sae Woo Nam¹ and Adam N. McCaughan¹

¹*National Institute of Standards and Technology, Boulder, CO, United States*

Correspondence*:

Jimmy Gammell

jimmy.gammell@colorado.edu

2 ABSTRACT

3 Equilibrium propagation is a learning framework that marks a step forward in the search for a
4 biologically-plausible implementation of deep learning, and could be implemented efficiently in
5 neuromorphic hardware. Previous applications of this framework to layered networks encountered
6 a vanishing gradient problem that has not yet been solved in a simple, biologically-plausible way.
7 In this paper, we demonstrate that the vanishing gradient problem can be overcome by replacing
8 some of a layered network's connections with random layer-skipping connections. We additionally
9 compare the computational complexities of equilibrium propagation and backpropagation to show
10 that it would be easier to implement the former in neuromorphic hardware.

11 **Keywords:** equilibrium propagation, deep learning, small-world, layer-skipping connections, neuromorphic computing, biologically-
12 motivated

13 **Number of words:** 6417

14 **Number of figures:** 7

15 **Number of tables:** 1

1 INTRODUCTION

16 As research into neural networks grows, there has been increased interest in designing biologically-inspired
17 training algorithms, as they may offer insight into biological learning processes and also offer clues
18 towards developing energy-efficient neuromorphic systems [Wozniak et al., 2018; Crafton et al., 2019;
19 Ernout et al., 2020; Bartunov et al., 2018; Lillicrap et al., 2014; Bengio et al., 2015]. The equilibrium
20 propagation learning framework developed Scellier and Bengio [2016] is one such algorithm. It is a
21 method for training a class of energy-based networks, the prototype for which is the continuous Hopfield
22 network Hopfield [1984]. In particular, it addresses one of the major issues that prevent other training
23 algorithms (such as backpropagation) from being biologically-plausible, which is the requirement for
24 separate computation pathways for different phases of training. This also makes the algorithm appealing for

practical implementation into neuromorphic hardware, because only a single computation circuit is required within each (non-output) neuron, rather than multiple distinct circuits. However, current implementations of the algorithm still have a defect that diminishes its biological plausibility: they require hand-tuned per-layer hyperparameters to account for a vanishing gradient through the network. In addition to not being biologically plausible, these multiplicative hyperparameters would be difficult to implement in a neuromorphic hardware system with limited bit depth. In this work, we demonstrate that the vanishing gradient problem can instead be solved through topological means: by randomly replacing some of a layered network's connections with layer-skipping connections, we can generate a network that trains each layer more evenly and does not need per-layer hyperparameters.

Implementation of equilibrium propagation in [Scellier and Bengio, 2016] was hindered by a vanishing gradient problem whereby networks with as few as 3 hidden layers trained slowly on MNIST [LeCun and Cortes, 1998] – a serious issue given that network depth is critical to performance on difficult datasets [Simonyan and Zisserman, 2014; Srivastava et al., 2015b] and that convergence to a low error rate on MNIST is a low bar to meet. The problem was overcome in [Scellier and Bengio, 2016] by independently tuning a unique learning rate for each layer in the network. These learning rates were multiplicative factors that proportionally scaled the signals communicated between layers.

In our work, we have modified the strictly-layered topology of the original implementation by adding and removing connections to create a small-world network [Watts and Strogatz, 1998]. Through this modification we have eliminated the per-layer hyperparameters without degrading the algorithm's performance – the modified network produces 0% training error (out of 50,000 examples) and $\lesssim 2.5\%$ test error (out of 10,000 examples) on MNIST using a network with three hidden layers and no regularization term in its cost function. These error rates are comparable to those of other biologically-motivated networks [Bartunov et al., 2018] and are approximately the same as those of the layered network with unique, manually-tuned learning rates in [Scellier and Bengio, 2016]. Our method could be implemented with relative ease in any system with configurable connectivity, such as those already described in several neuromorphic hardware platforms [Davies et al., 2018; Schemmel et al., 2010; Shainline et al., 2019]. Layer-skipping connections have been observed in biological brains [Bullmore and Sporns, 2009], so the approach is biologically-plausible. Similar techniques have seen success in convolutional [He et al., 2015; Srivastava et al., 2015a] and multilayer feedforward [Xiaohu et al., 2011; Krishnan et al., 2019] networks. Our findings outlined in this paper suggest that layer-skipping connections are effective-enough to be appealing in contexts where simplicity and biological plausibility are important.

2 BACKGROUND

2.1 Equilibrium propagation

Similarly to backpropagation, the equilibrium propagation algorithm [Scellier and Bengio, 2016] trains networks by approximating gradient descent on a cost function. Equilibrium propagation is applicable to any network with dynamics characterized by evolution to a fixed point of an associated energy function; our implementation is a recreation of that in [Scellier and Bengio, 2016], which applies it to a continuous Hopfield network [Hopfield, 1984]. The mathematical formulation of the framework can be found in [Scellier and Bengio, 2016]. We discuss its appeal over backpropagation in section 5.1.

63 2.1.1 Implementation in a continuous Hopfield network

64 Here we summarize the equations through which a continuous Hopfield network is trained using
65 equilibrium propagation; this summary is based on the more-thorough and more-general treatment in
66 [Scellier and Bengio, 2016].

67 Consider a network with n neurons organized into an input layer with p neurons, hidden layers with q
68 neurons and an output layer with r neurons. Let the activations of these neurons be denoted respectively
69 by vectors $\mathbf{x} \in \mathbb{R}^p$, $\mathbf{h} \in \mathbb{R}^q$ and $\mathbf{y} \in \mathbb{R}^r$, and let $\mathbf{s} = (\mathbf{h}^T, \mathbf{y}^T)^T \in \mathbb{R}^{q+r}$ and $\mathbf{u} = (\mathbf{x}^T, \mathbf{s}^T)^T \in \mathbb{R}^n$ be
70 vectors of, respectively, the activations of non-fixed (non-input) neurons and of all neurons in the network.
71 Let $\mathbf{W} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$ denote the network's weights and biases where w_{ij} is the connection weight
72 between neurons i and j and b_i is the bias for neuron i ($\forall i$ $w_{ii} = 0$ to prevent self-connections), and let ρ
73 denote its activation function; here and in [Scellier and Bengio, 2016],

$$\rho(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x \leq 1 \\ 1 & x > 1 \end{cases} \quad (1)$$

74 is a hardened sigmoid function where $\rho'(0) = \rho'(1)$ is defined to be 1 to avoid neuron saturation. Let
75 $\rho((x_1, \dots, x_n)^T) = (\rho(x_1), \dots, \rho(x_n))^T$.

76 The behavior of the network is to perform gradient descent on a total energy function F that is modified
77 by a training example $(\mathbf{x}_d, \mathbf{y}_d)$. Consider energy function $E : \mathbb{R}^n \rightarrow \mathbb{R}$,

$$E(\mathbf{u}; \mathbf{W}, \mathbf{b}) = \frac{1}{2} \mathbf{u}^T \mathbf{u} - \frac{1}{2} \rho(\mathbf{u})^T \mathbf{W} \rho(\mathbf{u}) - \mathbf{b}^T \mathbf{u} \quad (2)$$

78 and arbitrary cost function $C : \mathbb{R}^r \rightarrow \mathbb{R}_+$; here and in [Scellier and Bengio, 2016] it is a quadratic cost
79 function given by

$$C(\mathbf{y}) = \frac{1}{2} \|\mathbf{y} - \mathbf{y}_d\|_2^2, \quad (3)$$

80 though the framework still works for cost functions incorporating a regularization term dependent on \mathbf{W}
81 and \mathbf{b} . The total energy function $F : \mathbb{R}^n \rightarrow \mathbb{R}$ is given by

$$F(\mathbf{u}; \beta, \mathbf{W}, \mathbf{b}) = E(\mathbf{u}; \mathbf{W}, \mathbf{b}) + \beta C(\mathbf{y}) \quad (4)$$

82 where the clamping factor β is a small constant. \mathbf{s} evolves over time t as

$$\frac{d\mathbf{s}}{dt} \propto -\frac{\partial F}{\partial \mathbf{s}}. \quad (5)$$

83 Equilibrium has been reached when $\frac{\partial F}{\partial \mathbf{s}} \approx 0$. This can be viewed as solving the optimization problem

$$\underset{\mathbf{s} \in \mathbb{R}^{q+r}}{\text{minimize}} F((\mathbf{x}_d^T, \mathbf{s}^T)^T; \beta, \mathbf{W}, \mathbf{b}) \quad (6)$$

84 by using gradient descent to find a local minimum of F .

85 The procedure for training on a single input-output pair $(\mathbf{x}_d, \mathbf{y}_d)$ is as follows:

- 86 1. Clamp \mathbf{x} to \mathbf{x}_d and perform the free-phase evolution: evolve to equilibrium on the energy function
87 $F(\mathbf{u}; 0, \mathbf{W}, \mathbf{b})$ in a manner dictated by equation 5. Record the equilibrium state \mathbf{u}^0 .
- 88 2. Perform the weakly-clamped evolution: evolve to equilibrium on the energy function $F(\mathbf{u}; \beta, \mathbf{W}, \mathbf{b})$
89 using \mathbf{u}^0 as a starting point. Record the equilibrium state \mathbf{u}^β .
- 90 3. Compute the correction to each weight in the network:

$$\Delta W_{ij} = \frac{1}{\beta}(\rho(u_i^\beta)\rho(u_j^\beta) - \rho(u_i^0)\rho(u_j^0)). \quad (7)$$

- 91 Adjust the weights using $W_{ij} \leftarrow W_{ij} + \alpha \Delta W_{ij}$ where the learning rate α is a positive constant.
- 92 4. Compute the correction to each bias in the network:

$$\Delta b_i = \frac{1}{\beta}(\rho(u_i^\beta) - \rho(u_i^0)) \quad (8)$$

93 and adjust the biases using $b_i \leftarrow b_i + \alpha \Delta b_i$.

94 This can be repeated on as many training examples as desired. Training can be done on batches by
95 computing ΔW_{ij} and Δb_i for each input-output pair in the batch, and correcting using the averages of
96 these values. Note that the correction to a weight is computed using only the activations of neurons it
97 directly affects, and the correction to a bias is computed using only the activation of the neuron it directly
98 affects. This contrasts with backpropagation, where to correct a weight or bias l layers from the output it is
99 necessary to know the activations, derivatives and weights of all neurons between 0 and $l - 1$ layers from
100 the output.

101 2.2 Vanishing gradient problem

102 Vanishing gradients are problematic because they reduce a network’s rate of training and could be
103 difficult to represent in neuromorphic analog hardware due to limited bit depth. As a simple example, the
104 multiplicative factor of 0.008 used in previous implementations would lead to significant precision errors
105 in a system with signals represented by integers from 0-16 (bit depth of 4).

106 The vanishing gradient problem is familiar in the context of conventional feedforward networks,
107 where techniques such as the weight initialization scheme in [Glorot and Bengio, 2010], the use of
108 activation functions with derivatives that do not lead to output saturation [Schmidhuber, 2015], and batch
109 normalization [Ioffe and Szegedy, 2015] have been effective at overcoming it. However, in the context of
110 the networks trained in [Scellier and Bengio, 2016], the vanishing gradient problem persists even when the
111 former two techniques are used. To our knowledge batch normalization has not been used in the context of
112 equilibrium propagation; however, it seems unlikely to be biologically-plausible.

3 IMPLEMENTATION

We recreated the equilibrium propagation implementation¹ in [Scellier and Bengio, 2016] using the Pytorch library. Like the networks in [Scellier and Bengio, 2016], our networks are continuous Hopfield networks

¹ https://github.com/jgammell/Equilibrium_Propagation_mobile.git

with a hardened sigmoid activation function

$$\sigma(x) = \text{Max}\{0, \text{Min}\{x, 1\}\}$$

and squared-error cost function with no regularization term

$$C = \|\mathbf{y} - \mathbf{y}_d\|_2^2,$$

113 where \mathbf{y} is the network's output and \mathbf{y}_d is the target output. Tests were run on MNIST [LeCun and Cortes,
114 1998] grouped into batches of 20 examples, with the 50,000 training examples used for training and the
115 10,000 validation examples used for computing test errors.

116 We use two performance-enhancing techniques that were used in [Scellier and Bengio, 2016]: we
117 randomize the sign of β before training on each batch, which has a regularization effect, and we use
118 persistent particles, where the state of the network after training on a given batch during epoch n is used as
119 the initial state for that batch during epoch $n + 1$. Persistent particles reduce the computational resources
120 needed to approximate the differential equation governing network evolution, and would be unnecessary
121 in an analog implementation that can approximate the equation efficiently. Note that this technique leads
122 to higher error rates early in training than would be present with a more-thorough approximation of the
123 differential equation.

124 3.1 Layered topology with per-layer rates

125 We recreated the 5-layer network evaluated in [Scellier and Bengio, 2016]. It has the standard layered
126 topology shown in figure 1, and consists of a 784-neuron input layer, 3 500-neuron hidden layers and
127 a 10-neuron output layer. Weights are initialized using the scheme from [Glorot and Bengio, 2010]. As
128 mentioned above, each layer has a unique learning rate; the rates are $\alpha_1 = .128$, $\alpha_2 = .032$, $\alpha_3 = .008$ and
129 $\alpha_4 = .002$ where α_i is the learning rate for the connection weights between layers i and $i + 1$ and for the
130 biases in layer i , and the input and output layers are denoted $i = 1$ and $i = 5$, respectively.

131 3.2 Layered topology with global learning rate

132 To illustrate the vanishing gradient problem and provide a point of reference, we also tested the network
133 in section 3.1 with a single global learning rate of .02.

134 3.3 Our topology

135 To generate a network with our topology, we use algorithm 1. This topology is illustrated in figure 2.
136 The above algorithm is approximately equivalent to the algorithm for generating a small-world network
137 described in [Watts and Strogatz, 1998] with $p = 1 - (\frac{N_o - 1}{N_o})^n$ for $p \lesssim .2$, where N_o is the number
138 of connections in the network; to contextualize the number of replaced connections we will henceforth
139 describe networks with our topology in terms of p instead of n . We have seen good results with $p \approx 8\%$.
140 We have seen similar results when connections are added to the network, rather than randomly replaced
141 (algorithm 1, without removing pre-existing connections).

142 For these networks we use a global learning rate of .02 and, as in the networks from sections 3.1 and
143 3.2, initialize connections between neurons in adjacent layers using the scheme from [Glorot and Bengio,
144 2010]. For all other connections we draw initial weights from the uniform distribution $U[-.05, .05]$ where
145 the value .05 was determined empirically to yield good results.

Algorithm 1: Algorithm to produce our topology

Input: Layered network from section 3.2
Input: Integer n , giving number of connections to replace
Output: A network with our modified topology
for hidden layer in network **do**
 Add edge between each pair of neurons in layer
for $i \leftarrow 1$ to n **do**
 Randomly select pre-existing connection in network;
 Add connection between random unconnected pair of
 neurons in network;
 // Do not allow self connections
 // Do not allow connections between two input neurons or
 between two output neurons
 Remove pre-existing connection;
return modified network

4 RESULTS

146 We compared the networks described in section 3 by observing their behavior while training on MNIST
 147 [LeCun and Cortes, 1998]. All networks used $\epsilon = .5$, $\beta = 1.0$, 500 free-phase iterations, 8 weakly-clamped-
 148 phase iterations, and were trained for 250 epochs.

149 4.1 Network performance comparison

150 Figure 3 illustrates that our network significantly outperforms one with a global learning rate, and
 151 achieves close to the same training and test error rates as one with unique learning rates, albeit after
 152 around 25% more epochs. Both our network and the layered network with unique learning rates achieve
 153 approximately a 2.5% test error and 0% training error, whereas the layered network with a global learning
 154 rate has test and training error rates around .5% higher than the other two networks.

155 4.2 Training rates of individual pairs of layers

156 To observe the extent of the vanishing gradient problem, for each network we tracked the root-mean-
 157 square correction to weights in each of its layers during training on MNIST [LeCun and Cortes, 1998].
 158 Figure 4 shows an 11-point centered moving average of these values (without averaging the values are
 159 very volatile). It can be seen that for the layered network with a global learning rate, the magnitude of the
 160 correction to a typical neuron vanishes with depth relative to the output, with the shallowest weights training
 161 around 100 times faster than the deepest weights - this illustrates the vanishing gradient problem. The use
 162 of unique learning rates is very effective at making corrections uniform. Our topology with $p = 7.56\%$ is
 163 effective at making deeper layers train in a uniform way, but the output layer still trains around 10 times
 164 faster than deeper layers; nonetheless, figure 3 suggests that this imperfect solution still yields a significant
 165 performance benefit.

166 The fast training of the output layer in the network with our topology is probably because no layer-
 167 skipping connections attach directly to the target output, so for any value of p the shortest path between a
 168 deep neuron and the target layer is at least 2 connections long, whereas the path between an output neuron
 169 and the target layer is only 1 connection long.

170 4.3 Effect of p

171 We tracked the training error after one epoch of a network with our topology while varying p ; the results
 172 are shown in figure 5. For $p < .1\%$, there is little improvement in the error rate as p is increased, but there
 173 is substantial improvement in the uniformity of the training rates of deep layers. When $p > .1\%$, the deep
 174 layers are very uniform, and the error rate starts decreasing with p at a rate that is slightly slower than
 175 exponential; at this point there is little improvement in the uniformity of deep layers, but the rate of the
 176 shallowest layer appears to move closer to those of the deeper layers.

177 We found that our topology performs significantly worse than the basic topology with one learning rate
 178 when few connections are replaced. This could be due to a poor weight initialization scheme for the added
 179 intralayer connections; we have noticed anecdotally that networks appear to be less-sensitive to their weight
 180 initialization scheme as connections are replaced. We have found that networks perform poorly relative to a
 181 basic network with one learning rate until p is in the ballpark of 7%. This experiment suggests that training
 182 rate will keep improving long after that, but does not show long-term performance or test performance; we
 183 suspect that a network's generalization ability will suffer for large p as it loses its regimented nature.

5 DISCUSSION

184 5.1 Comparing the computational complexity of equilibrium propagation and 185 backpropagation

186 The main motivations for using equilibrium propagation instead of an alternative machine learning
 187 technique (such as deep learning using backpropagation for training) are 1) to gain insight into the
 188 operation of the brain by developing target-based learning approaches in biologically plausible networks
 189 and 2) to develop algorithms that are more easily implemented in hardware. Below we qualitatively
 190 compare the hardware that would be needed for implementation of equilibrium propagation versus for
 191 backpropagation on a standard feedforward network to gain insight into the utility of these networks.

192 5.1.1 Requirements of equilibrium propagation

Just as in the algorithm, to implement equilibrium propagation in hardware, three different phases of hardware operation are required. In the first (free) phase, it follows from equations 2, 4 and 5 that to determine its state, the i -th neuron in a network must compute

$$\frac{\partial F}{\partial u_i} = u_i - \frac{1}{2}\rho'(u_i)\left[\sum_{i \neq j} W_{ij}\rho(u_j) + b_i\right],$$

193 plus the term $\beta(u_i - y_i^{target})$ for output neurons when using a squared-error cost function, and then integrate
 194 the result over time. Parameter correction rules are given by equations 7 and 8. This is exactly the operation
 195 of an analog leaky integrate and fire neuron, as for example implemented in the neuromorphic hardware
 196 platforms of references [Indiveri et al., 2011; Schemmel et al., 2010] among others. A qualitative diagram
 197 of potential neuron and synapse devices and their output and read/write to memory operations are shown in
 198 the diagram in figure 6 (a). At each neuron N_i^l the value of $U_{i,0}^{l+1}$ is written to memory and the nonlinear
 199 function ρ is applied before sending to the synapse device, where it is multiplied by the weight w_{ij}^{l+1} which
 200 is read from memory by the synapse device. These weighted outputs are summed at the input of the next
 201 neuron device (N_j^{l+1}) and added to a bias value b_j^{l+1} that is read from memory to generate U_{l+1} .

202 In the second (weakly-clamped) phase, shown in figure 6 (b), the operation of the hardware is exactly
 203 the same as in (a), with the value U_β written to memory at each neuron. Not shown in figure 6 is the
 204 functionality at the output neurons which are weakly clamped and have a new function in this phase.

205 Finally, in the third phase, the weights and biases are updated as shown in figure 6 (c). At each neuron
 206 device the values of U_0 and U_β are read from memory and $\rho(U_0)$ and $\rho(U_\beta)$ are calculated. At the synapse
 207 device, the computation of equation 7 is performed using these values from the pre- and post- synaptic
 208 neurons to calculate the weight update Δw , and the value of the weight in memory is updated to $w + \alpha \Delta w$.
 209 Similar updates are applied to the bias b at every neuron according to equation 8. [Brief sentence on number](#)
 210 [of neurons versus synapses for the read/write to memory operations.](#)

211 5.1.2 Requirements of backpropagation

Backpropagation is an algorithm for training networks using gradient descent. It is most typically applied to feedforward neural networks, in which the activation value of a neuron i in layer l is given by

$$\rho(u_i^l) = \rho\left(\sum_j W_{ij}^l u_j^{l-1} + b_i^l\right).$$

This is very similar to the free running situation in equilibrium propagation, with the main difference being that the connections are unidirectional. The qualitative implementation of this inference phase in hardware is shown in Fig. 7 (a). Using backpropagation, the parameters are then updated by computing error correction terms δ_i^l for each neuron i in layer l ; for the output layer L the correction is

$$\delta_i^L = \rho'(u_i^L)(\rho(u_i^L) - y_i^{target})$$

and for deeper layers it is

$$\delta_i^l = \rho'(u_i^l) \sum_j W_{ij}^{l+1} \delta_j^{l+1}.$$

The implementation of this in hardware is shown in figure 7 (b) (excluding layer L). Note that the data is now moving in the opposite (backwards) direction, and unlike in the case of equilibrium propagation, the functions implemented by the neurons are entirely different to the operation in the forward phase shown in (a). In a final phase, weights are corrected using

$$\Delta W_{ij}^l = \rho(u_i^{l-1}) \delta_j^l$$

and biases using

$$\Delta b_i^l = \delta_i^l.$$

212 This is shown in figure 7 (c).

213 5.1.3 Comparison

214 The most-significant difference between the algorithms is that in equilibrium propagation, the free
 215 and weakly-clamped phases of training are identical for most neurons and the weakly-clamped phase
 216 requires only slight modification to output neurons, whereas in backpropagation these phases demand
 217 significantly-different functionality from essentially all neurons. There are two other differences that we do
 218 not believe to be significant in terms of ease of implementation in hardware. One is that in equilibrium
 219 propagation each pair of neurons is joined by a bidirectional synapse, whereas in backpropagation each pair

is joined by two unidirectional synapses; we expect both cases to be equally easy to implement. The other is that in equilibrium propagation, each neuron must remember its equilibrium state after the free phase while it executes the weakly-clamped phase; since backpropagation implies a state variable for the activation and error term of each neuron, the memory requirement of each neuron should be the same in both cases. For a hardware implementation, the need for distinct free and weakly-clamped phases (temporally non-local credit assignment) significantly reduces the advantages associated with the spatially local credit assignment. Recently there has been new work that indicates that the algorithm can be modified to eliminate the need for both phases [Ernault et al., 2020]. This would significantly reduce the memory requirements of the algorithm. Various characteristics of both algorithms are compared side-by-side in table 1. [We should also discuss the layer skipping connections with respect to the implementation of this algorithm in hardware - just a sentence about it](#)

5.2 Related work

References [Lee et al., 2015; Xie and Seung, 2003; Pineda, 1987] describe other approaches to locally approximating the gradient of a cost function. References [Lillicrap et al., 2014; Crafton et al., 2019] explore the use of a random feedback matrix for backwards connections that is more biologically-plausible than identical forwards and backwards connections. Reference [Bartunov et al., 2018] explores the present state of biologically-motivated deep learning, and [Bengio et al., 2015] discusses the criteria a biologically-plausible network would need to satisfy. References [Shainline et al., 2019; Davies et al., 2018; Nahmias et al., 2013] discuss analog hardware that could potentially implement equilibrium propagation. References [He et al., 2015; Srivastava et al., 2015a; Xiaohu et al., 2011; Krishnan et al., 2019] use layer-skipping connections for other types of networks and learning frameworks. References [Ioffe and Szegedy, 2015; Glorot and Bengio, 2010] give approaches to solving vanishing gradient problems.

5.3 Directions for Future Research

There are several directions in which future research could be taken:

- Evaluating the effectiveness of this approach on hard datasets, such as CIFAR and ImageNet.
- Evaluating the effect of p on a network's test error in the long term.
- Exploring the effectiveness of layer-skipping connections on deeper networks.
- Exploring the effectiveness of a network when layer-skipping connections are used during training and removed afterwards.

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon.

REFERENCES

- Bartunov, S., Santoro, A., Richards, B. A., Hinton, G. E., and Lillicrap, T. P. (2018). Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *CoRR* abs/1807.04587

- 255 Bengio, Y., Lee, D., Bornschein, J., and Lin, Z. (2015). Towards biologically plausible deep learning.
256 *CoRR* abs/1502.04156
- 257 Bullmore, E. and Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and
258 functional systems. *Nature*
- 259 Crafton, B., Parihar, A., Gebhardt, E., and Raychowdhury, A. (2019). Direct feedback alignment with
260 sparse connections for local learning. *CoRR* abs/1903.02083
- 261 Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Joshi, P., Lines, A., et al. (2018). Loihi: A neuromorphic
262 manycore processor with on-chip learning. *IEEE Micro* PP, 1–1. doi:10.1109/MM.2018.112130359
- 263 Ernoult, M., Grollier, J., Querlioz, D., Bengio, Y., and Scellier, B. (2020). Equilibrium propagation with
264 continual weight updates
- 265 Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural
266 networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and*
267 *Statistics*, eds. Y. W. Teh and M. Titterton (Chia Laguna Resort, Sardinia, Italy: PMLR), vol. 9 of
268 *Proceedings of Machine Learning Research*, 249–256
- 269 He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*
270 abs/1512.03385
- 271 Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of
272 two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America* 81,
273 3088–92. doi:10.1073/pnas.81.10.3088
- 274 Indiveri, G., Linares-Barranco, B., Hamilton, T., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al.
275 (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience* 5, 73. doi:10.3389/fnins.2011.
276 00073
- 277 Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing
278 internal covariate shift. *CoRR* abs/1502.03167
- 279 Krishnan, G., Du, X., and Cao, Y. (2019). Structural pruning in deep neural networks: A small-world
280 approach
- 281 [Dataset] LeCun, Y. and Cortes, C. (1998). The mnist database of handwritten digits
- 282 Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. 498–515.
283 doi:10.1007/978-3-319-23528-8_31
- 284 Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support
285 learning in deep neural networks
- 286 Nahmias, M., Shastri, B., Tait, A., and Prucnal, P. (2013). A leaky integrate-and-fire laser neuron for
287 ultrafast cognitive computing. *Selected Topics in Quantum Electronics, IEEE Journal of* 19, 1–12.
288 doi:10.1109/JSTQE.2013.2257700
- 289 Pineda, F. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review*
290 *Letters* 59, 2229–2232
- 291 Scellier, B. and Bengio, Y. (2016). Equilibrium propagation: Bridging the gap between energy-based
292 models and backpropagation
- 293 Schemmel, J., Brüderle, D., Grünbl, A., Hock, M., Meier, K., and Millner, S. (2010). A wafer-scale
294 neuromorphic hardware system for large-scale neural modeling. *Proceedings of 2010 IEEE International*
295 *Symposium on Circuits and Systems* , 1947–1950
- 296 Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* 61, 85–117.
297 doi:10.1016/j.neunet.2014.09.003

- 298 Shainline, J. M., Buckley, S. M., McCaughan, A. N., Chiles, J. T., Jafari Salim, A., Castellanos-Beltran,
299 M., et al. (2019). Superconducting optoelectronic loop neurons. *Journal of Applied Physics* 126, 044902.
300 doi:10.1063/1.5096403
- 301 Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image
302 recognition
- 303 Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015a). Highway networks. *CoRR* abs/1505.00387
- 304 Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015b). Training very deep networks
- 305 Watts, D. and Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *Nature*
- 306 Wozniak, S., Pantazi, A., and Eleftheriou, E. (2018). Deep networks incorporating spiking neural dynamics.
307 *CoRR* abs/1812.07040
- 308 Xiaohu, L., Xiaoling, L., Jinhua, Z., Yulin, Z., and Maolin, L. (2011). A new multilayer feedforward small-
309 world neural network with its performances on function approximation. In *2011 IEEE International*
310 *Conference on Computer Science and Automation Engineering*
- 311 Xie, X. and Seung, H. (2003). Equivalence of backpropagation and contrastive hebbian learning in a
312 layered network. *Neural computation* 15, 441–54. doi:10.1162/089976603762552988

FIGURES

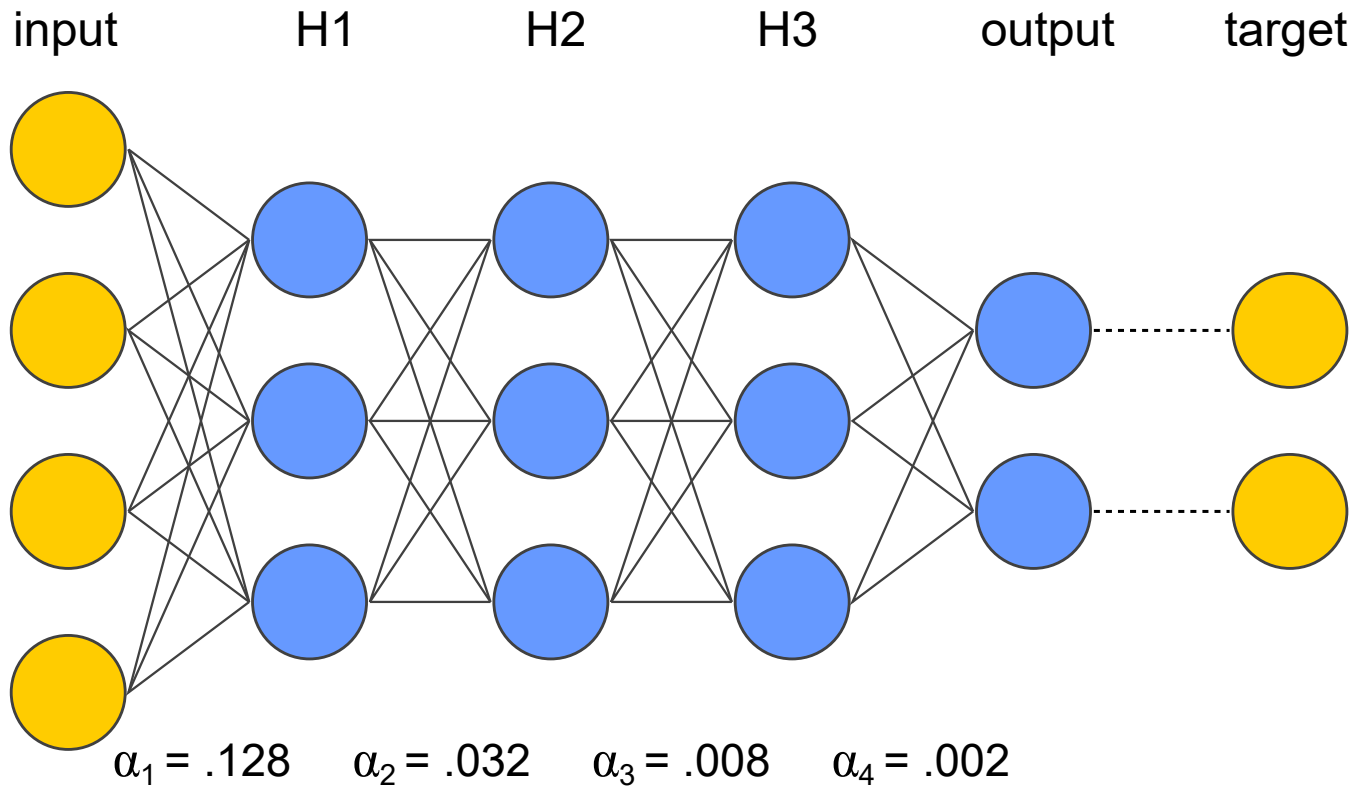


Figure 1. Topology of the layered network tested in [Scellier and Bengio, 2016]. All pairs of neurons in adjacent layers are connected. All connections are bidirectional. To compensate for the vanishing gradient problem, the learning rate α is reduced by a factor of 4 each time distance from the output decreases by one layer.

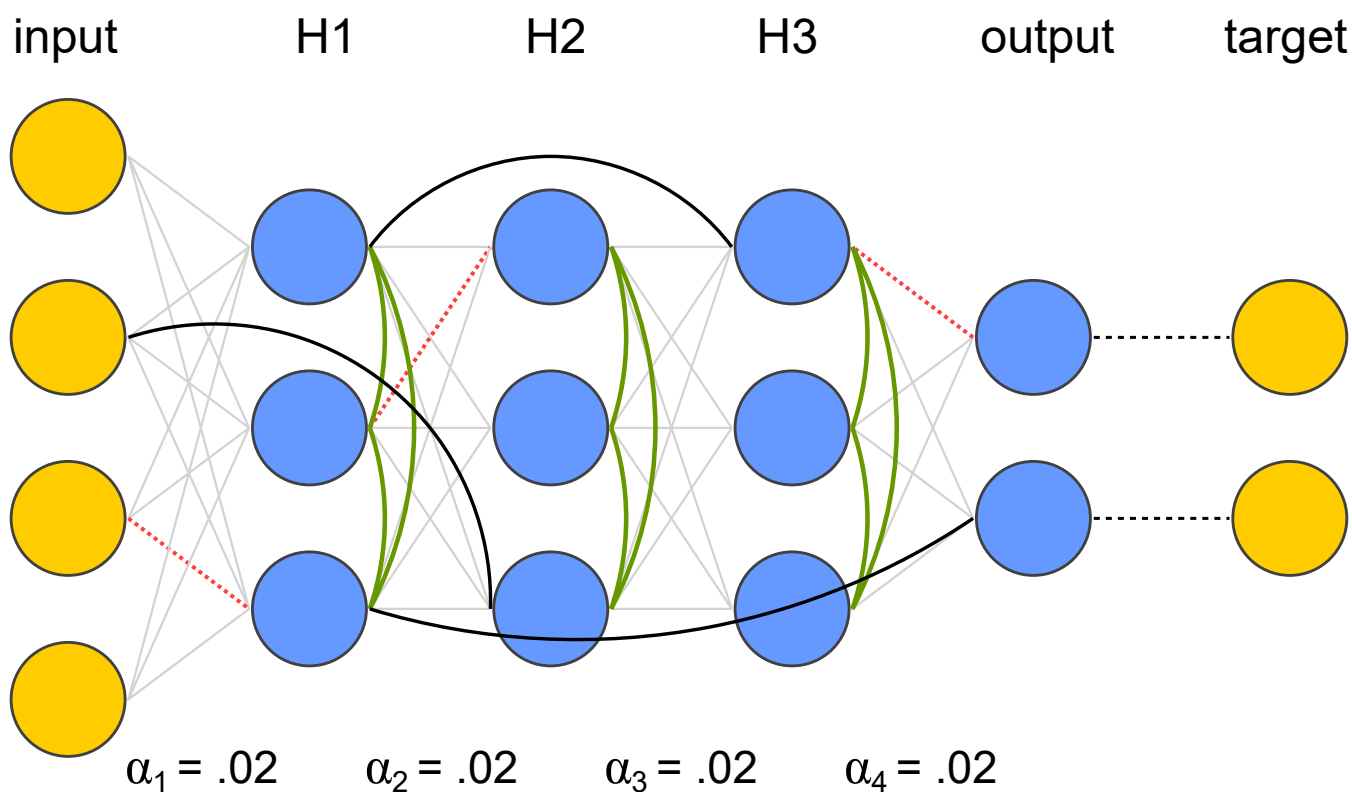


Figure 2. Our modifications to the topology of figure 1 to avoid a vanishing gradient while using a global learning rate. Red dotted lines denote connections that have been removed, black lines denote their replacements, and green solid lines denote added intralayer connections. All connections are bidirectional.

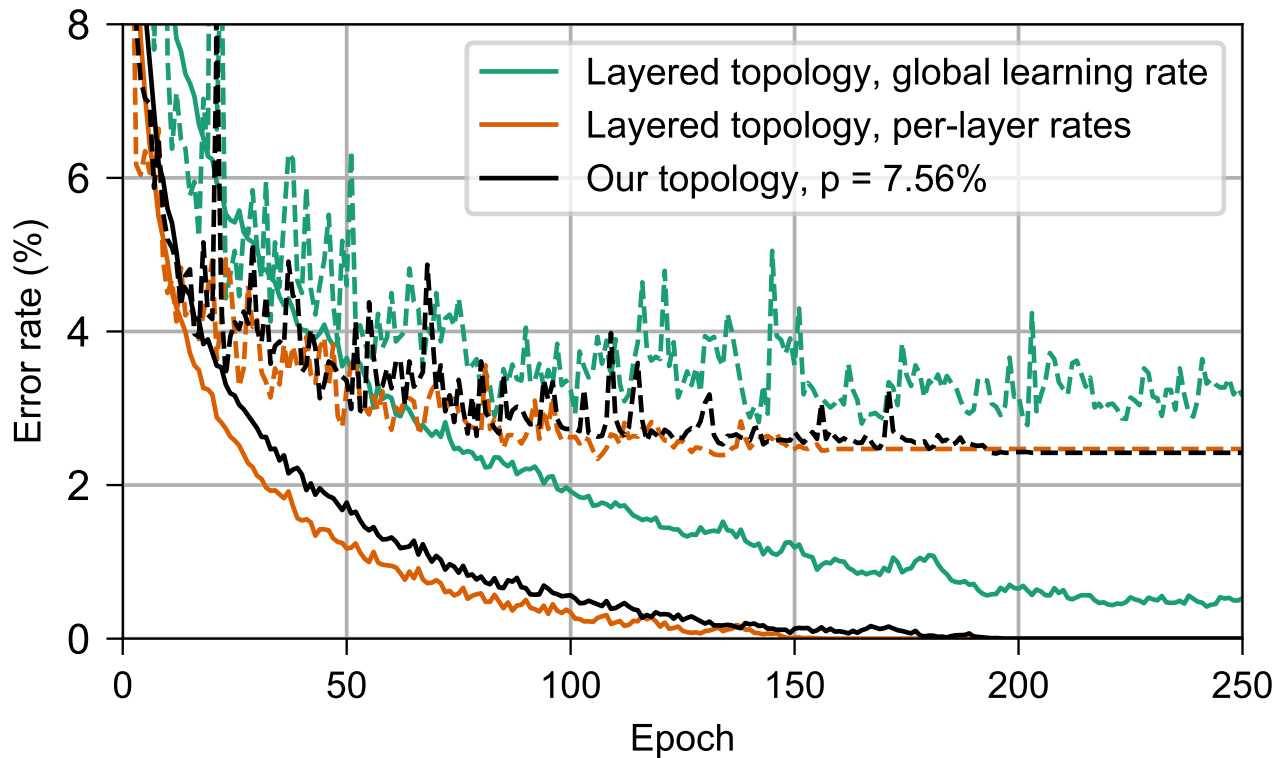


Figure 3. Performance on MNIST of the networks in section 3. Dashed lines show the test error and solid lines show the training error. In green is a layered network with a global learning rate (section 3.2), in orange is a layered network with per-layer rates individually tuned to counter the vanishing gradient problem (section 3.1), and in green is a network with our topology, $p = 7.56\%$ (section 3.3). Observe that our topology is almost as effective as per-layer rates at countering the vanishing gradient problem that impedes training of the layered network with a global learning rate.

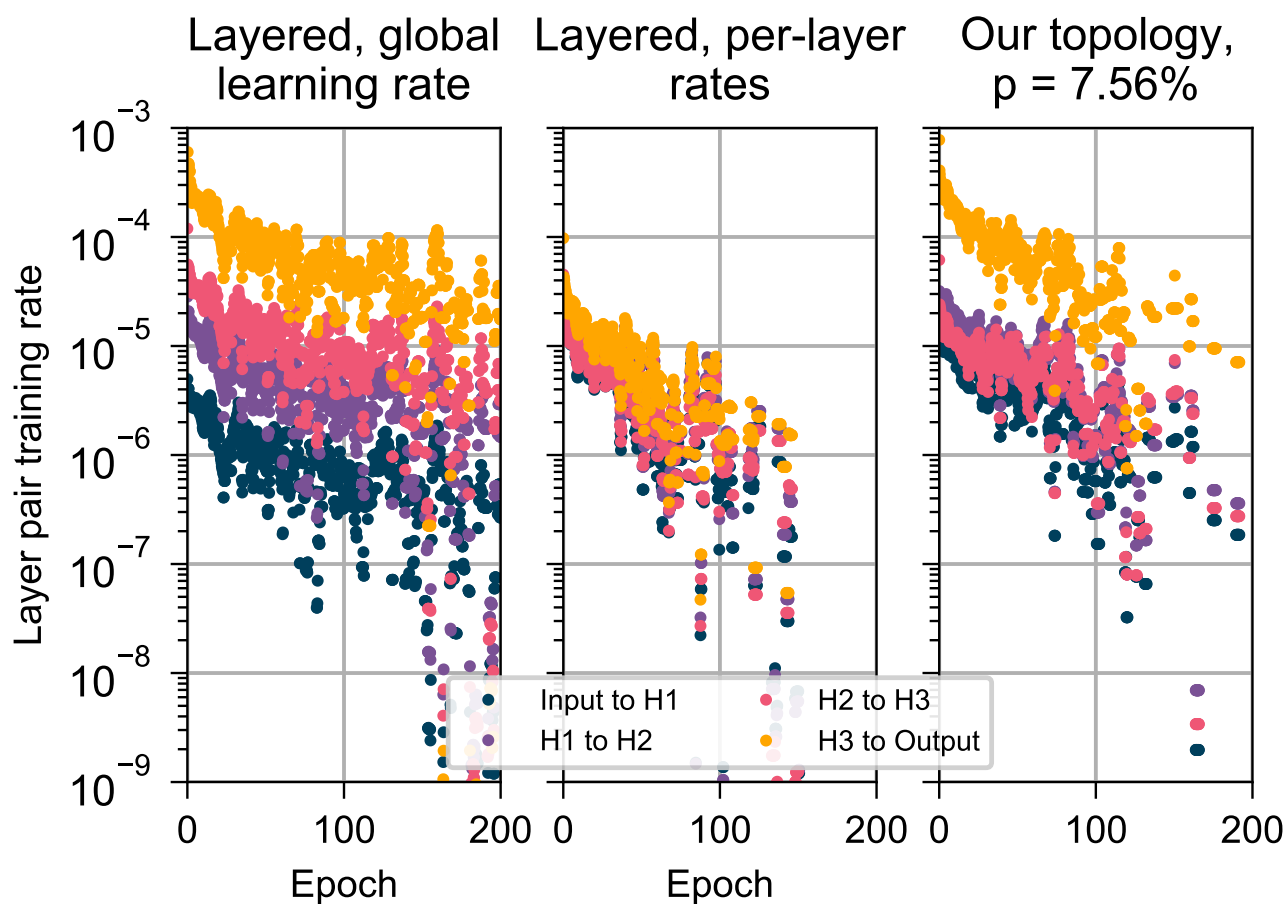


Figure 4. Root-mean-square corrections to weights in different layers while training on MNIST, for the networks in section 3. For clarity, values were subjected to an 11-point centered moving average. (left) A layered network with a single global learning rate (section 3.2). (center) A layered network a unique, individually-tuned learning rate for each layer (section 3.1). (right) A network with our topology, $p = 7.56\%$ (section 3.3). Observe that the layered topology with a global learning rate has a vanishing gradient problem, which is almost completely solved by tuning an individual learning rate for each layer. Our topology improves the situation by making training uniform among the deeper layers, although the shallowest layer still trains more-quickly than the deeper layers.

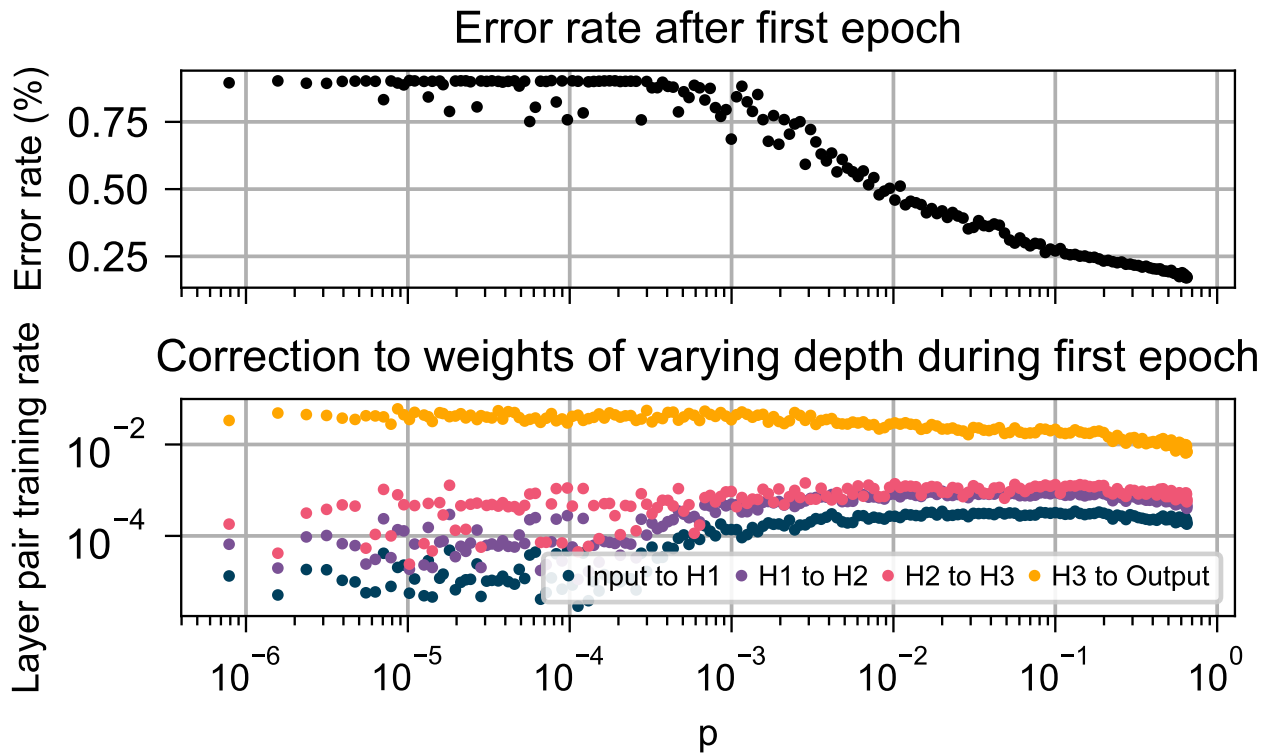


Figure 5. Behavior of our network (section 3.3) with varying p , during the first epoch of training. (top) The training error after one epoch. (bottom) Root-mean-square correction to weights in different layers during the first epoch. Observe that as p is increased, the error rate decreases and the root-mean-square corrections to each layer become more-uniform.

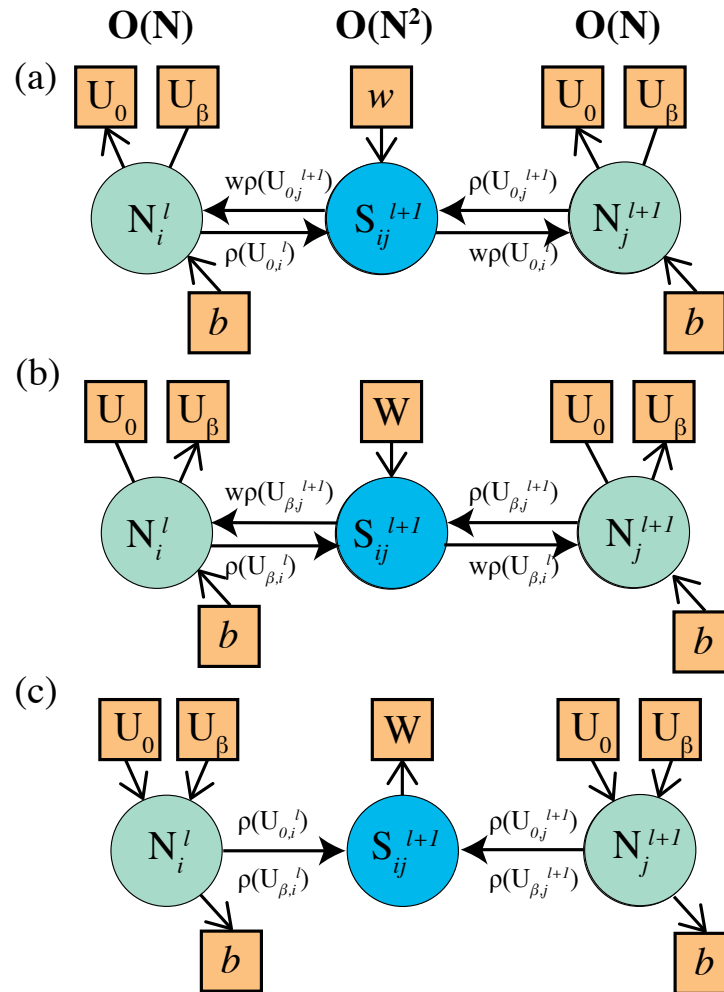


Figure 6. Illustration of the functionality needed to implement equilibrium propagation in hardware. Yellow squares indicate a value that must be stored in memory for a subsequent phase. The circles indicate (N) neuron and (S) synapse devices with the associated functions described in the text. (a) The functionality required by the neurons and synapses in the free running phase. (b) The functionality of the neurons and synapses (except output neurons) in the weakly clamped phase. (c) The functionality of the neurons and synapses in the weight and bias update phase.

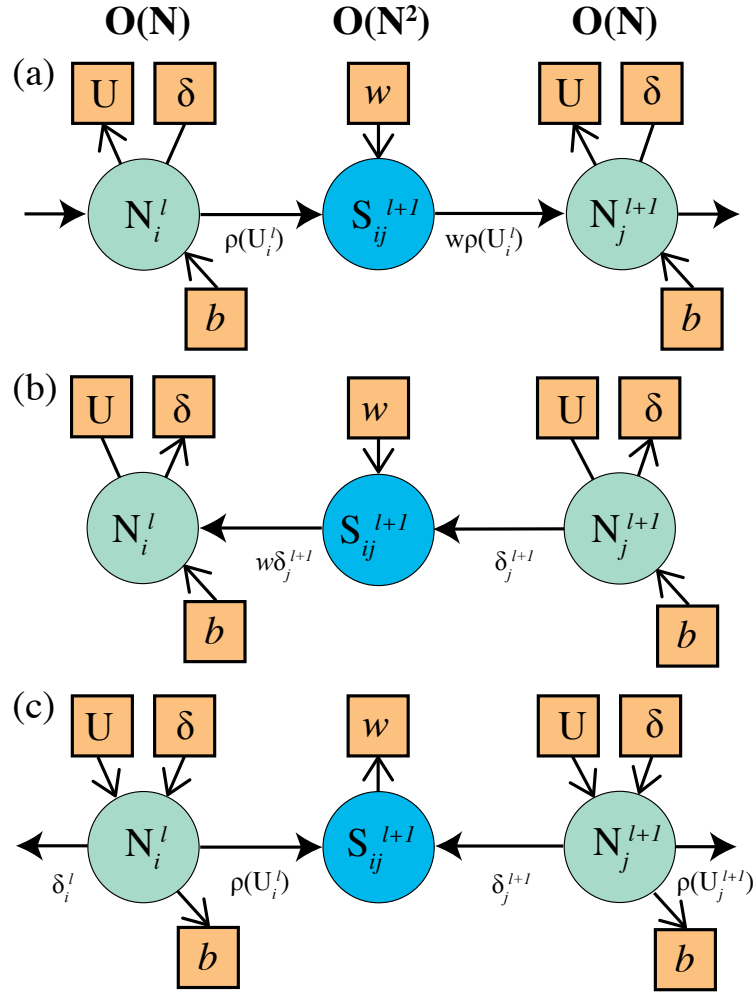


Figure 7. Illustration of the functionality needed to implement backpropagation in hardware. Yellow squares indicate a value that must be stored in memory for a subsequent phase. The circles indicate (N) neuron and (S) synapse devices with the associated functions described in the text. (a) The functionality required by the neurons and synapses in the forward pass phase. (b) The functionality of the neurons and synapses (except the last layer of neurons) in the backpropagation phase. (c) The functionality of the neurons and synapses in the weight and bias update phase.

TABLES

| | Backpropagation | Equilibrium Propagation |
|---|---|--|
| Number of distinct computations | 2 – computations during forwards and backwards phases are distinct | ≈ 1 – hidden neurons perform same computation in both phases. Output neurons perform a similar but modified version of the same computation. |
| Types of connections | Unidirectional to transmit activation to shallower neighbors and error to deeper neighbors | Bidirectional to each neighbor |
| Memory | Space to store activation and error term for each neuron | Space to store free and weakly-clamped activations for each neuron |
| Order of computations | Forwards propagation phase where layers are computed from deepest to shallowest; backwards propagation phase where layers are computed from shallowest to deepest; parameter update phase | Free phase where all neurons evolve simultaneously; weakly-clamped phase where all neurons evolve simultaneously; parameter update phase |
| Nonlinear activation function | Yes | Yes |
| Derivative of nonlinear activation function | Yes | Yes |
| Correction computation | Corrections require dedicated circuitry unique from that implementing propagation | Corrections require dedicated circuitry unique from that implementing evolution |

Table 1. Comparison of the capabilities a hardware neuron would need in order to implement backpropagation and equilibrium propagation.