

# Layer-skipping connections improve the effectiveness of equilibrium propagation on layered networks

Jimmy Gammell<sup>1,\*</sup>, Sonia M. Buckley<sup>1</sup>, Sae Woo Nam<sup>1</sup> and Adam N. McCaughan<sup>1</sup>

<sup>1</sup>*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  
7 way. In this paper, we demonstrate that the vanishing gradient problem can be mitigated by  
8 replacing some of a layered network's connections with random layer-skipping connections in a  
9 manner inspired by small-world networks. This approach would be convenient to implement in  
10 neuromorphic hardware, and is biologically-plausible.

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

13 **Number of words:** 3152

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 Ernault 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 addressed 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. This solution is biologically-plausible and would be easier to implement in a neuromorphic system; additionally, it entails hand-tuning only two new hyperparameters (the number of layer-skipping connections and their initial weights), whereas the original solution adds a new hyperparameter for each pair of layers in a network.

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-like network [Watts and Strogatz, 1998]. Through this modification we have eliminated the per-layer hyperparameters without substantially degrading the algorithm’s performance – after 250 epochs the modified network produces .0117% training error (out of 50,000 examples) and 2.55% 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. While small-world networks are not a novel concept, to our knowledge our work is the first to train small-world-like networks using the Equilibrium Propagation learning framework.

## 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 A.

### 2.1.1 Implementation in a continuous Hopfield network

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

Consider a network with  $n$  neurons organized into an input layer with  $p$  neurons, hidden layers with  $q$  neurons and an output layer with  $r$  neurons. Let the activations of these neurons be denoted respectively 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 vectors of, respectively, the activations of non-fixed (non-input) neurons and of all neurons in the network. 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 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  $\rho$  denote its activation function; here and in [Scellier and Bengio, 2016],

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

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

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

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

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

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

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

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

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

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

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

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

by using gradient descent to find a local minimum of  $F$ . Parameters  $\theta$  can then be updated using the rule

$$\Delta\theta \propto -\frac{1}{\beta} \left( \frac{\partial F}{\partial \theta}((\mathbf{x}_d^T, \mathbf{s}^T)^T; \beta, \mathbf{W}, \mathbf{b}) - \frac{\partial F}{\partial \theta}((\mathbf{x}_d^T, \mathbf{s}^T)^T; 0, \mathbf{W}, \mathbf{b}) \right). \quad (7)$$

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

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

$$\Delta W_{ij} = \frac{1}{\beta} (\rho(u_i^\beta) \rho(u_j^\beta) - \rho(u_i^0) \rho(u_j^0)). \quad (8)$$

Adjust the weights using  $W_{ij} \leftarrow W_{ij} + \alpha \Delta W_{ij}$  where the learning rate  $\alpha$  is a positive constant.

4. Compute the correction to each bias in the network:

$$\Delta b_i = \frac{1}{\beta} (\rho(u_i^\beta) - \rho(u_i^0)) \quad (9)$$

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

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

## 2.2 Vanishing gradient problem

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

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

## 2.3 Small-world networks

The topology presented in this paper is inspired by small-world networks described in [Watts and Strogatz, 1998]. In that paper the authors consider ring lattice networks which are highly-cliquish in the sense that

the neighbors of a vertex, defined as the set of vertices with which the vertex shares an edge, are very likely to share edges with one-another. The typical distance between a pair of vertices in such a network, defined as the minimum number of edges that must be traversed in order to move between them, tends to be large as well. The authors demonstrate that when each edge in the network is randomly re-wired with some probability  $p$ , the typical distance is reduced at a much higher rate than the cliquishness as  $p$  increases, and one can thereby create a network with a high degree of cliquishness yet a short typical path length between vertices. They call such a network a small-world network.

Since in our context networks with multilayer feedforward topology have gradients that attenuate by a multiplicative factor of distance from the output layer, it seems reasonable to expect attenuation to be reduced when the typical number of connections between a given neuron and the output layer is reduced. Based on this we were motivated to explore the small-world-inspired topology described in subsequent sections as a way to reduce the typical number of connections between a pair of neurons and therefore between a neuron and the output layer, while largely preserving the layered structure of a network. It is worth noting that multilayer feedforward topology is not a perfect analogy for a circular lattice network because it lacks connections within layers and is therefore not cliquish based on the definition used in [Watts and Strogatz, 1998]; nonetheless, we have seen empirically that our small-world-inspired topology does mitigate the vanishing gradient problem and generally improve network performance.

## 2.4 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. Reference [Pedroni et al., 2019] discusses various means of storing weights in memory in the context of neuromorphic implementations of spike timing dependent plasticity, which is similar to equilibrium propagation in that it is a biologically-inspired learning algorithm that performs weight updates using local information. 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.

## 3 IMPLEMENTATION

We implemented<sup>1</sup> the Equilibrium Propagation framework described in [Scellier and Bengio, 2016] using Pytorch [Paszke et al., 2019]. Like the networks in [Scellier and Bengio, 2016], our networks are continuous Hopfield networks with a hard sigmoid activation function

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

and squared-error cost function with no regularization term

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

<sup>1</sup> [https://github.com/jgammell/equilibrium\\_propagation](https://github.com/jgammell/equilibrium_propagation)

where  $\mathbf{y}$  is the network’s output and  $\mathbf{y}_d$  is the target output. As described in section 5.1 of [Scellier and Bengio, 2016], we numerically approximate the dynamics described by equation 5 through a forward Euler approximation with step size  $\epsilon$ ,  $N_{free}$  iterations of the free phase of training and  $N_{weakly-clamped}$  iterations of the weakly-clamped phase of training.

We use two performance-enhancing techniques that were used in [Scellier and Bengio, 2016]: we randomize the sign of  $\beta$  before training on each batch, which was found in the original paper to have a regularization effect, and we use persistent particles, where the state of the network after training on a given batch during epoch  $n$  is used as the initial state for that batch during epoch  $n + 1$ . Persistent particles are useful when simulating equilibrium propagation on a digital computer because they allow the network to start an epoch closer to the equilibrium state, thereby reducing the number of iterations of the forward Euler approximation that are necessary to get sufficiently close to equilibrium. Note that this technique leads to higher error rates early in training than would be present with a more-thorough approximation of the differential equation. For purposes of computational efficiency we compute training error throughout training by recording the classification error on each training batch prior to correcting the network’s parameters, and we compute the test error by evolving to equilibrium and evaluating classification error for each batch in the test dataset after each full epoch of training. In some of our trials (e.g. figure 4) this approach causes the training error to exceed the test error early on in training because the network has undergone only part of an epoch of training prior to evaluating error on each training batch, but a full epoch prior to evaluating error on each test batch.

In all networks we use the weight initialization scheme in [Glorot and Bengio, 2010] for the weights of interlayer connections; weights connecting a pair of layers with  $n_1$  and  $n_2$  neurons are taken from the uniform distribution  $U[-\sqrt{\frac{6}{n_1+n_2}}, \sqrt{\frac{6}{n_1+n_2}}]$ . We have found empirically that for new connections added in our topology, for a network with  $N$  layers it works reasonably well to draw all initial weights from  $U[-a, a]$  where  $a = \frac{1}{N} \sum_{i=1}^{N-1} \sqrt{\frac{6}{n_i+n_{i+1}}}$  and  $n_i$  denotes the number of neurons in layer  $i$ .

### 3.1 Multilayer feedforward (MLFF) topology

The purpose of this paper is to address the vanishing gradient problem that is present in networks with a multilayer feedforward (MLFF) topology in [Scellier and Bengio, 2016]. Therefore, we have done a variety of trials on networks with MLFF topology to provide points of reference. The MLFF topology is illustrated to the left in figure 1; it consists of layers of neurons with connections between every pair of neurons in adjacent layers, no connections within layers, and no connections between neurons in non-adjacent layers.

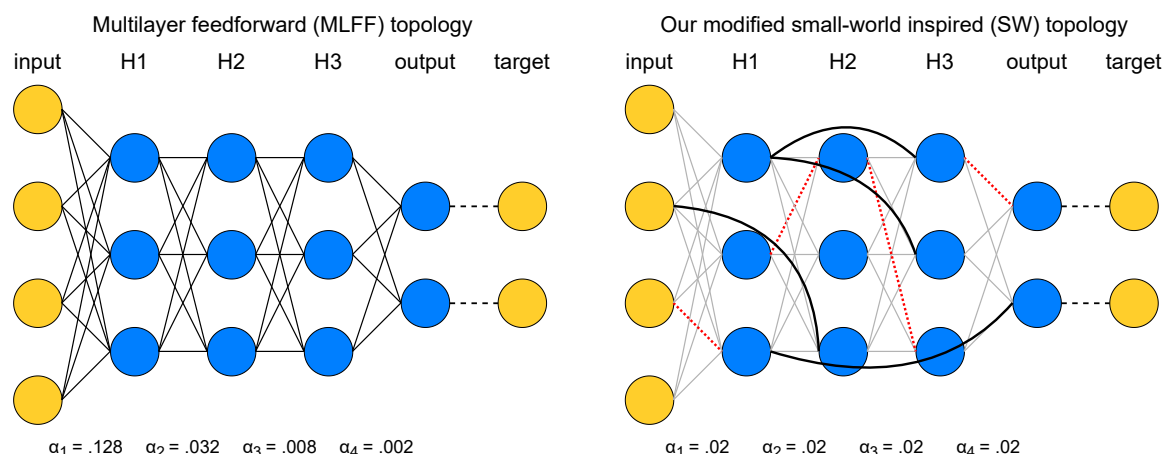
In some trials we use a single global learning rate  $\alpha$ , and in other trials we use per-layer rates individually tuned to counter the vanishing gradient problem. The latter case entails  $N + 1$  unique learning rates  $\alpha_i$ ,  $i = 1, \dots, N + 1$  for a network with  $N$  hidden layers, where the weights connecting layers  $i$  and  $i + 1$  and the biases in layer  $i$  have learning rate  $\alpha_i$ .

### 3.2 Small-world inspired (SW) topology

We generate a network with small-world inspired (SW) topology by first starting with a MLFF topology as described above in section 3.1, then applying an algorithm similar to the one described in [Watts and Strogatz, 1998] to randomly replace <sup>2</sup> existing connections by random layer-skipping connections with some probability  $p$ . This is done using algorithm 1, and the resulting SW topology is illustrated to the right in figure 1. In all of our trials networks with SW topology are trained using a single global learning rate  $\alpha$ .

<sup>2</sup> We have done a small number of trials in which layer skipping connections are added without removing an equal number of preexisting connections, and have observed behavior similar to that resulting from using algorithm 1.





**Figure 1.** Illustration of our topological modifications to mitigate the vanishing gradient problem while using a global learning rate. (left) A network with MLFF topology as tested in [Scellier and Bengio, 2016]. Observe that the learning rate increases by a factor of 4 each time the distance from the output increases by one layer. (right) A network with SW topology, where a subset of connections have been replaced by random layer-skipping connections and per-layer learning rates have been replaced by a single global learning rate. Red dotted lines denote removed connections and solid black lines denote the layer-skipping connections replacing them.

---

**Algorithm 1:** Algorithm to generate SW topology starting with a network with MLFF topology

---

**Input:** Network with MLFF topology to modify

**Parameter:** Probability  $p$  with which to replace a given preexisting connection

**Output:** Modified network with SW topology

network  $\leftarrow$  input network with MLFF topology

potentialConns  $\leftarrow$  {pairs of distinct neurons in distinct layers that do not share a connection}

**for** existing connection  $ec$  in network **do**

$pp \leftarrow$  value drawn from uniform distribution  $U[0, 1]$

**if**  $pp < p$  **then**

        remove connection  $ec$  in network

        randomly draw connection  $pc$  from potentialConns

        make connection  $pc$  in network

        remove  $pc$  from potentialConns

        add  $ec$  to potentialConns

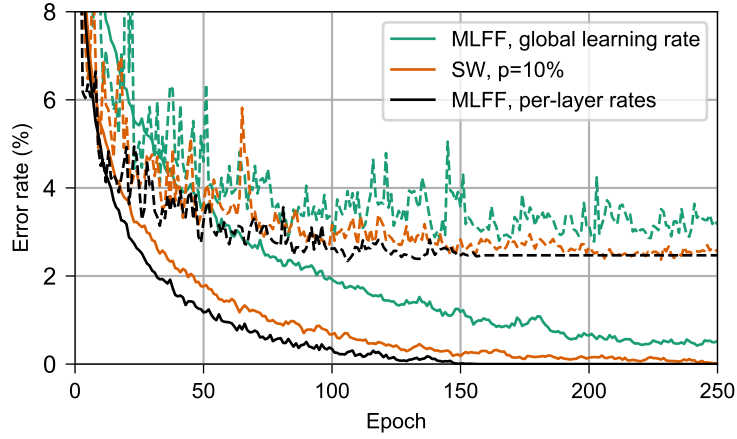
## 4 RESULTS

### 192 4.1 Evaluation on MNIST dataset

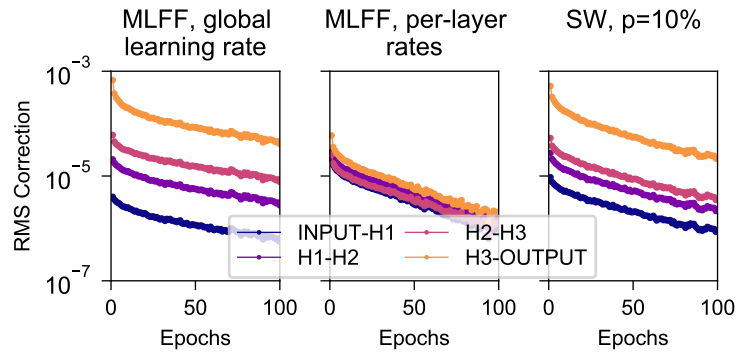
193 Here we compare the behavior of networks with the SW topology presented here to those with the MLFF  
 194 topology used in [Scellier and Bengio, 2016] when training on the MNIST dataset. We are using this  
 195 dataset because it allows us to reproduce and extend the trials in [Scellier and Bengio, 2016], and because  
 196 it is non-trivial to effectively train on yet small enough to allow trials to complete in a reasonably-short  
 197 amount of time.

#### 198 4.1.1 Classification error

199 Figure 2 shows the results of comparing the classification error on MNIST of a network with SW topology  
 200 to that of a MLFF network with individually-tuned per-layer learning rates, as in [Scellier and Bengio,  
 201 2016], and to that of a MLFF network with a single global learning rate. For all networks we use 3



**Figure 2.** Performance on MNIST of networks with 3 500-neuron hidden layers. Dashed lines show the test error and solid lines show the training error. In black is a MLFF network with per-layer rates individually tuned to counter the vanishing gradient problem. In green is the same MLFF network but with a single global learning rate. In orange is a network with SW topology,  $p = 10\%$ . Observe that the network with our topology trains almost as quickly as a network with per-layer rates, and significantly more-quickly than a network with a single learning rate.

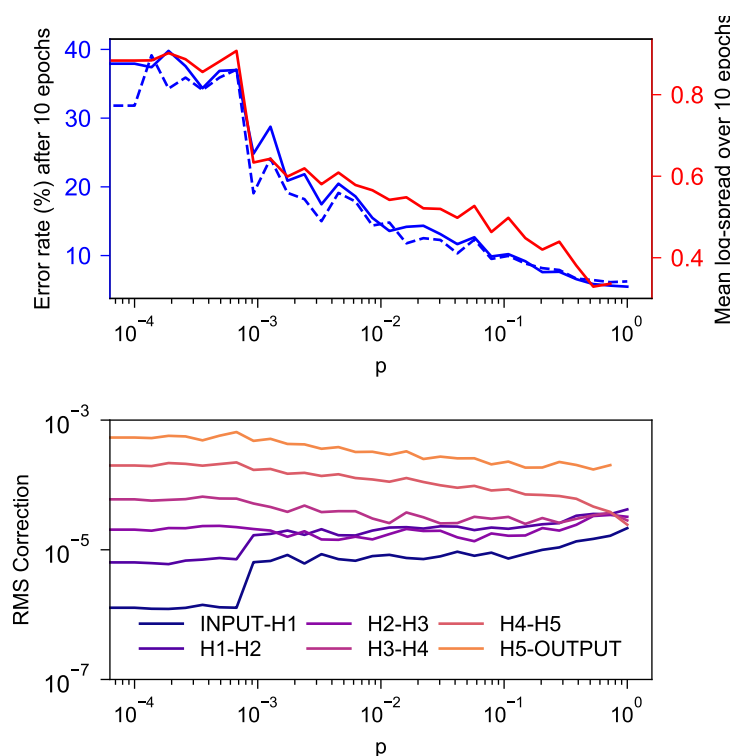


**Figure 3.** Root mean square corrections to weights in different layers while training on MNIST, for networks with 3 500-neuron hidden layers. Measurements were taken after each batch, and averaged over each epoch. (left) A MLFF network with a single global learning rate. (center) A MLFF network with per-layer rates individually tuned to counter the vanishing gradient problem. (right) A network with SW topology,  $p = 10\%$ . Observe that the correction magnitudes attenuate significantly with depth in the MLFF network with a single global learning rate, and that a network with SW topology reduces the severity of the issue, albeit less-effectively than individually tuning a learning rate for each layer.

202 500-neuron hidden layers,  $\epsilon = .5$ ,  $\beta = 1.0$ , 500 free-phase iterations, 8 weakly-clamped-phase iterations,  
 203 and train for 250 epochs. For the SW network we use  $p = 10\%$  and a global learning rate  $\alpha = .02$ . For the  
 204 MLFF network with per-layer rates we use learning rates  $\alpha_1 = .128$ ,  $\alpha_2 = .032$ ,  $\alpha_3 = .008$  and  $\alpha_4 = .002$ .  
 205 For the MLFF network with a single global learning rate, we use learning rate  $\alpha = .02$ .

206 We find that both the SW network and the MLFF network with per-layer rates significantly outperform  
 207 the MLFF network with a single global learning rate during the first 250 epochs of training. The SW  
 208 network achieves training and test error rates similar to those of the MLFF network with per-layer rates,  
 209 albeit after around 100 additional epochs of training.





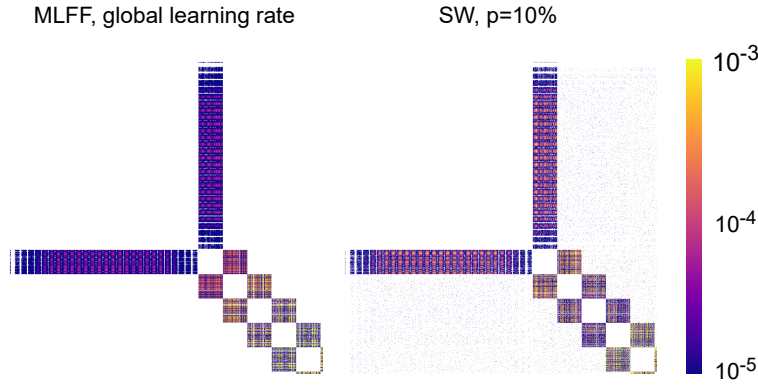
**Figure 4.** Behavior during the first 10 epochs of training on MNIST for a SW network with 5 100-neuron hidden layers for various values of  $p$ . (top) In solid and dashed blue are the training and test error rates after 10 epochs and in red is the log-spread (equation 10) averaged over the 10 epochs. As expected, both values decrease as  $p$  increases. There appears to be a strong linear correlation between the two, with coefficient of determination  $r^2 = .970$ . This is consistent with our suspicion that mitigation of the vanishing gradient problem is the reason our topology tends to increase the rate at which layered networks train. (bottom) Root mean square corrections to weights in different layers, averaged over the 10 epochs. As expected, the spread of these values decreases as  $p$  increases.

#### 210 4.1.2 Training rates of individual pairs of layers

211 Here we consider the first 100 epochs of the trials described in section 4.1.1 above and track the root-  
 212 mean-square correction to weights connecting each pair of adjacent layers. Figure 3 shows these values for  
 213 the 3 network topologies, recorded after each batch and averaged over each epoch.

214 We can clearly see the vanishing gradient problem for the MLFF network with a single global learning  
 215 rate (left), manifesting as attenuation with depth of the root-mean-square corrections to weights. The  
 216 problem is addressed very-effectively by the use of manually-tuned per-layer learning rates (center), and is  
 217 mitigated to a more-modest extent when we use SW topology with a global learning rate. It is noteworthy  
 218 that the speed of training of these networks as shown in figure 2 is commensurate with the uniformity with  
 219 which their layers train as shown in figure 3.

220 It can be seen that in the network with SW topology the weights connecting to the output layer train  
 221 significantly faster than deeper weights, which cluster together; we have observed similar behavior in  
 222 a variety of datasets and network dimensions. We suspect it has to do with the fact that output neurons  
 223 connect directly to the target output, whereas because layer-skipping connections are not attached to the  
 224 target output the other layers must connect indirectly through at least 2 connections.



**Figure 5.** Mean absolute value correction matrix over 100 epochs for networks with 5 100-neuron hidden layers. A pixel at position  $(i, j)$  corresponds to the magnitude of the correction to the connection weight between neurons  $i$  and  $j$ . (left) A network with MLFF topology and a single global learning rate. (right) A network with SW topology,  $p = 10\%$ . Observe that attenuation of these values with depth is less-significant in the latter than in the former.

#### 225 4.1.3 Behavior for varying $p$

226 Figure 4 shows the behavior during the first 10 epochs of training on MNIST for a network with SW  
 227 topology as  $p$  is increased from 0 to .727. The network being tested has 5 100-neuron hidden layers and is  
 228 trained with learning rate  $\alpha = .015$ ,  $\epsilon = .5$ ,  $\beta = 1.0$ , 500 free-phase iterations and 8 weakly-clamped-phase  
 229 iterations.

230 We see in the top graph that the training and test error rates after 10 epochs decay exponentially as  $p$   
 231 increases. The bottom graph indicates that the RMS corrections to weights become more-uniform with  
 232 depth as  $p$  increases. Corrections to layers that do not connect directly to the output layer cluster closer  
 233 together as  $p$  increases, but it appears that the corrections to weights connecting directly to the output layer  
 234 train faster than deeper weights with a gap that does not appear to decrease with  $p$ ; this is similar to the  
 235 behavior seen in section 4.1.2.

236 To quantify the spread of the RMS corrections as a single scalar, we introduce the statistic

$$\text{log-spread} = \text{Std. dev}\{\log_{10}(w_l), l = 1, \dots, N + 1\} \quad (10)$$

237 where  $N$  denotes the number of hidden layers in a network and  $w_l$  denotes the root mean square magnitude  
 238 of corrections to weights connecting the  $l^{\text{th}}$  and  $l + 1^{\text{th}}$  layers from the input, averaged over all epochs of  
 239 training. The log-spread is plotted in the top graph alongside the error rate, and these values can be seen to  
 240 have a strong linear correlation. The training error and the log-spread have a coefficient of determination  
 241  $r^2 = .970$ .

#### 242 4.1.4 Weight correction matrix

243 In figure 5 we visualize the mean weight correction matrices resulting from training a MLFF network and  
 244 a SW network with 5 100-neuron hidden layers as described in section 4.1.3. Training on a batch yields a  
 245 correction matrix  $d\mathbf{W}$  where element  $d\mathbf{W}_{ij}$  denotes the change to the connection weight between neurons  
 246  $i$  and  $j$ ; here we have recorded a matrix with element  $(i, j)$  containing the average of  $|d\mathbf{W}_{ij}|$  over 100  
 247 epochs of training and displayed it as an image with the color of a pixel at position  $(i, j)$  encoding the  
 248 magnitude of  $|d\mathbf{W}_{ij}|$  as indicated by the legend.

As expected, there is attenuation with depth to weight correction magnitudes in a MLFF network with a single global learning rate, and transitioning to a network with SW topology reduces the severity of the problem. An interesting feature of the SW matrix that is visible upon close inspection is that layer-skipping connections tend to receive larger correction magnitudes when they are closer to the output of the network. Striations can be seen in correction magnitudes, which we believe correspond to sections of images in the MNIST dataset that contain varying amounts of information; for example, for corrections connecting to the input layer, there are 28 striations which likely correspond to the 28 rows of pixels in an image of a digit, with pixels closer to the edges of the images typically blank and pixels closer to the centers of the images typically containing most of the variation that encodes a digit.

## 4.2 Evaluation on various datasets and topologies

Here we evaluate the presence of the vanishing gradient problem and the effectiveness of our topology at addressing it on MNIST [LeCun and Cortes, 1998], Fashion MNIST (FMNIST) [Xiao et al., 2017], and the diabetes and wine toy datasets distributed in scikit-learn [Pedregosa et al., 2011] with various network architectures. Our results are shown in table 1. For all of these trials we use  $\beta = 1.0$  and  $\epsilon = .5$ . While hyperparameters in these trials have not been thoroughly optimized, the trials serve to illustrate that the trends we saw on the MNIST dataset in section 4.1 hold for deeper networks and for networks training on 4 different datasets.

We report the training and test error after 100 epochs, as well as the log-spread (equation 10). We see that networks with SW topology,  $p = 10\%$ , have a consistently smaller log-spread than networks with a MLFF topology and a single global learning rate. Here this associated with smaller training and test error rates, though we have seen for some less-optimized combinations of hyperparameters that the error rates do not significantly change. We have observed that all of these networks behave in ways that are qualitatively similar to the networks explored in section 4.1.

## 5 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 a network when layer-skipping connections are used during training and removed afterwards.
- Devise and mathematically justify a weight initialization scheme for layer-skipping connections.
- Mathematically justify the empirical results we have seen when using SW topology.

## 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

Dataset	Layer sizes	Topology	L.R.	Iterations	Error (train/test)	log-spread
Diabetes	10-10-10-10-10-10-1	MLFF	.01	1000/12	.0326/.0441	1.085
Diabetes	10-10-10-10-10-10-1	SW, p=10%	.01	1000/12	.0302/.0394	<b>.557</b>
Diabetes	10-10-10-10-10-10-10-10-10-1	MLFF	.01	5000/18	.0318/.0425	2.490
Diabetes	10-10-10-10-10-10-10-10-10-1	SW, p=10%	.01	5000/18	.0230/.0339	<b>.442</b>
Wine	13-10-10-10-10-10-3	MLFF	.01	1000/12	.680/.760	1.216
Wine	13-10-10-10-10-10-3	SW, p=10%	.01	1000/12	.467/.280	<b>.689</b>
Wine	13-10-10-10-10-10-10-10-10-3	MLFF	.01	5000/18	.653/.720	1.734
Wine	13-10-10-10-10-10-10-10-10-3	SW, p=10%	.01	5000/18	.253/.560	<b>.471</b>
MNIST	784-500-500-500-10	MLFF	.02	500/8	.0170/.0310	.689
MNIST	784-500-500-500-10	SW, p=10%	.02	500/8	.00675/.0272	<b>.545</b>
MNIST	784-100-100-100-100-100-10	MLFF	.015	1000/12	.156/.131	.867
MNIST	784-100-100-100-100-100-10	SW, p=10%	.015	1000/12	.0407/.0540	<b>.450</b>
FMNIST	784-100-100-100-100-100-10	MLFF	.015	1000/12	.266/.255	.862
FMNIST	784-100-100-100-100-100-10	SW, p=10%	.015	1000/12	.152/.164	<b>.484</b>

**Table 1.** Comparison of MLFF and SW topologies with various datasets and network architectures. Trials were run for 100 epochs with batch size 20 on MNIST and FMNIST and for 10 epochs with batch size 5 on Diabetes and Wine. Observe that the vanishing gradient problem, quantified by the log-spread (equation 10), is consistently less-severe when using the SW topology over the MLFF topology. Error is given as proportion of classes incorrectly identified for MNIST, FMNIST and Wine (classification problems) and as mean absolute value of error for Diabetes (regression problem).

- 285 Bengio, Y., Lee, D., Bornschein, J., and Lin, Z. (2015). Towards biologically plausible deep learning.  
286 *CoRR* abs/1502.04156
- 287 Bullmore, E. and Sporns, O. (2009). Complex brain networks: graph theoretical analysis of structural and  
288 functional systems. *Nature*
- 289 Crafton, B., Parihar, A., Gebhardt, E., and Raychowdhury, A. (2019). Direct feedback alignment with  
290 sparse connections for local learning. *CoRR* abs/1903.02083
- 291 Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Joshi, P., Lines, A., et al. (2018). Loihi: A neuromorphic  
292 manycore processor with on-chip learning. *IEEE Micro* PP, 1–1. doi:10.1109/MM.2018.112130359
- 293 Ernoult, M., Grollier, J., Querlioz, D., Bengio, Y., and Scellier, B. (2020). Equilibrium propagation with  
294 continual weight updates
- 295 Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural  
296 networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and*  
297 *Statistics*, eds. Y. W. Teh and M. Titterton (Chia Laguna Resort, Sardinia, Italy: PMLR), vol. 9 of  
298 *Proceedings of Machine Learning Research*, 249–256
- 299 He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*  
300 abs/1512.03385
- 301 Hopfield, J. (1984). Neurons with graded response have collective computational properties like those of  
302 two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America* 81,  
303 3088–92. doi:10.1073/pnas.81.10.3088
- 304 Indiveri, G., Linares-Barranco, B., Hamilton, T., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al.  
305 (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience* 5, 73. doi:10.3389/fnins.2011.  
306 00073

- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR* abs/1502.03167
- Krishnan, G., Du, X., and Cao, Y. (2019). Structural pruning in deep neural networks: A small-world approach
- [Dataset] LeCun, Y. and Cortes, C. (1998). The mnist database of handwritten digits
- Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. 498–515. doi:10.1007/978-3-319-23528-8\_31
- Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support learning in deep neural networks
- Nahmias, M., Shastri, B., Tait, A., and Prucnal, P. (2013). A leaky integrate-and-fire laser neuron for ultrafast cognitive computing. *Selected Topics in Quantum Electronics, IEEE Journal of* 19, 1–12. doi:10.1109/JSTQE.2013.2257700
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* 32, eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc.). 8024–8035
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12, 2825–2830
- Pedroni, B. U., Joshi, S., Deiss, S. R., Sheik, S., Detorakis, G., Paul, S., et al. (2019). Memory-efficient synaptic connectivity for spike-timing- dependent plasticity. *Frontiers in Neuroscience* 13, 357. doi:10.3389/fnins.2019.00357
- Pineda, F. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters* 59, 2229–2232
- Scellier, B. and Bengio, Y. (2016). Equilibrium propagation: Bridging the gap between energy-based models and backpropagation
- Schemmel, J., Brüderle, D., Grünbl, A., Hock, M., Meier, K., and Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 1947–1950
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks* 61, 85–117. doi:10.1016/j.neunet.2014.09.003
- Shainline, J. M., Buckley, S. M., McCaughan, A. N., Chiles, J. T., Jafari Salim, A., Castellanos-Beltran, M., et al. (2019). Superconducting optoelectronic loop neurons. *Journal of Applied Physics* 126, 044902. doi:10.1063/1.5096403
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015a). Highway networks. *CoRR* abs/1505.00387
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015b). Training very deep networks
- Watts, D. and Strogatz, S. (1998). Collective dynamics of 'small-world' networks. *Nature*
- Wozniak, S., Pantazi, A., and Eleftheriou, E. (2018). Deep networks incorporating spiking neural dynamics. *CoRR* abs/1812.07040
- [Dataset] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms
- Xiaohu, L., Xiaoling, L., Jinhua, Z., Yulin, Z., and Maolin, L. (2011). A new multilayer feedforward small-world neural network with its performances on function approximation. In *2011 IEEE International Conference on Computer Science and Automation Engineering*

- 352 Xie, X. and Seung, H. (2003). Equivalence of backpropagation and contrastive hebbian learning in a  
353 layered network. *Neural computation* 15, 441–54. doi:10.1162/089976603762552988



TABLES

	Backpropagation	Equilibrium Propagation
Number of distinct computations	2 – computations during forwards and backwards phases are distinct	$\approx 1$ – 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 2.** Comparison of the capabilities a hardware neuron would need in order to implement backpropagation and equilibrium propagation.

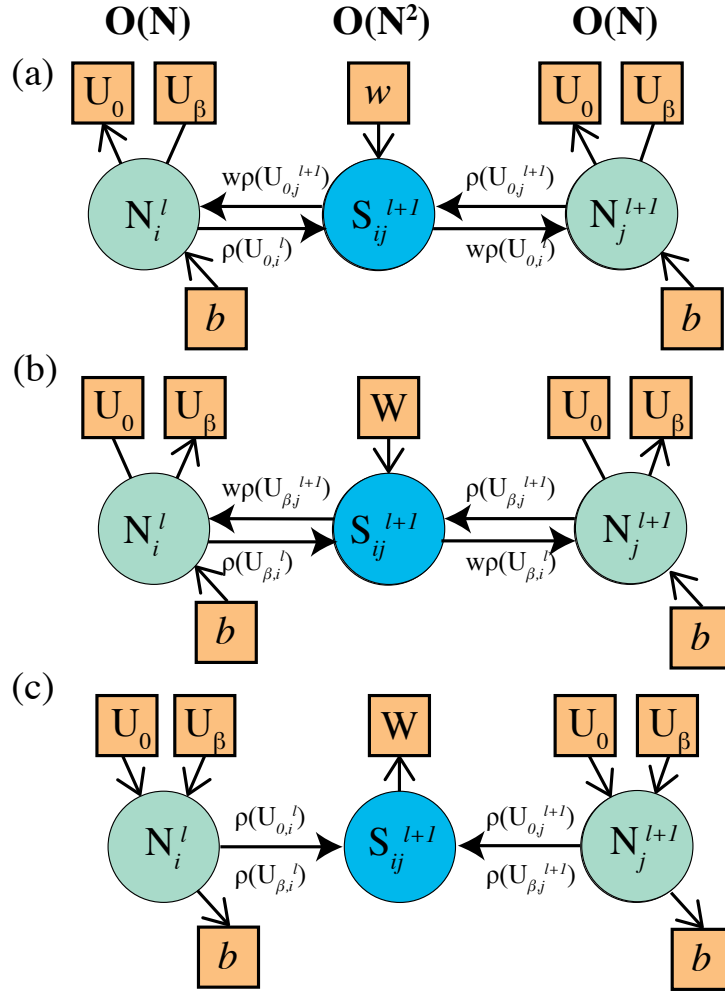
**A COMPARING THE COMPUTATIONAL COMPLEXITY OF EQUILIBRIUM PROPAGATION AND BACKPROPAGATION**

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

**A.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

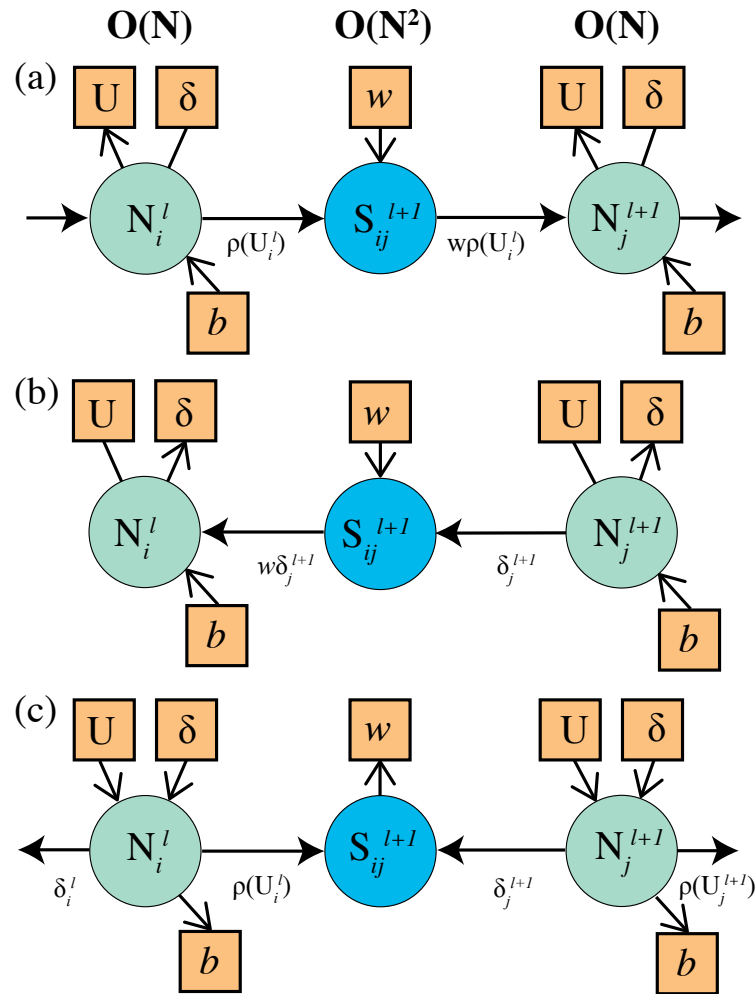
$$\frac{\partial F}{\partial u_i} = u_i - \frac{1}{2}\rho'(u_i)[\sum_{i \neq j} W_{ij}\rho(u_j) + b_i],$$



**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.

361 plus the term  $\beta(u_i - y_i^{target})$  for output neurons when using a squared-error cost function, and then integrate  
 362 the result over time. Parameter correction rules are given by equations 8 and 9. This is exactly the operation  
 363 of an analog leaky integrate and fire neuron, as for example implemented in the neuromorphic hardware  
 364 platforms of references [Indiveri et al., 2011; Schemmel et al., 2010] among others. A qualitative diagram  
 365 of potential neuron and synapse devices and their output and read/write to memory operations are shown in  
 366 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  
 367 function  $\rho$  is applied before sending to the synapse device, where it is multiplied by the weight  $w_{ij}^{l+1}$  which  
 368 is read from memory by the synapse device. These weighted outputs are summed at the input of the next  
 369 neuron device ( $N_j^{l+1}$ ) and added to a bias value  $b_j^{l+1}$  that is read from memory to generate  $U_{l+1}$ .

370 In the second (weakly-clamped) phase, shown in figure 6 (b), the operation of the hardware is exactly  
 371 the same as in (a), with the value  $U_\beta$  written to memory at each neuron. Not shown in figure 6 is the  
 372 functionality at the output neurons which are weakly clamped and have a new function in this 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.

Finally, in the third phase, the weights and biases are updated as shown in figure 6 (c). At each neuron device the values of  $U_0$  and  $U_\beta$  are read from memory and  $\rho(U_0)$  and  $\rho(U_\beta)$  are calculated. At the synapse device, the computation of equation 8 is performed using these values from the pre- and post-synaptic neurons to calculate the weight update  $\Delta w$ , and the value of the weight in memory is updated to  $w + \alpha\Delta w$ . Similar updates are applied to the bias  $b$  at every neuron according to equation 9. As shown at the top of 6, for  $N$  such neurons per layer, there will be  $N^2$  synapses.

## A.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  $\delta_i^l$  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

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

380 This is shown in figure 7 (c).

### 381 A.3 Comparison

382 The most-significant difference between the algorithms is that in equilibrium propagation, the free  
 383 and weakly-clamped phases of training are identical for most neurons and the weakly-clamped phase  
 384 requires only slight modification to output neurons, whereas in backpropagation these phases demand  
 385 significantly-different functionality from essentially all neurons. There are two other differences that we do  
 386 not believe to be significant in terms of ease of implementation in hardware. One is that in equilibrium  
 387 propagation each pair of neurons is joined by a bidirectional synapse, whereas in backpropagation each pair  
 388 is joined by two unidirectional synapses; we expect both cases to be equally easy to implement. The other is  
 389 that in equilibrium propagation, each neuron must remember its equilibrium state after the free phase while  
 390 it executes the weakly-clamped phase; since backpropagation implies a state variable for the activation and  
 391 error term of each neuron, the memory requirement of each neuron should be the same in both cases. For  
 392 a hardware implementation, the need for distinct free and weakly-clamped phases (temporally non-local  
 393 credit assignment) significantly reduces the advantages associated with the spatially local credit assignment.  
 394 Recently there has been new work that indicates that the algorithm can be modified to eliminate the need  
 395 for both phases [Ernault et al., 2020]. This would significantly reduce the memory requirements of the  
 396 algorithm. Various characteristics of both algorithms are compared side-by-side in table 2.

## FIGURES