Abstract

Equilibrium propagation is a learning framework that marks a step forward in the search for a biologically-plausible implementation of deep learning, and is appealing for implementation in neuromorphic analog hardware. However, previous implementations on layered networks encountered a vanishing gradient problem that has not yet been solved in a simple, biologically-plausible way. In this paper, we demonstrate that the vanishing gradient problem can be overcome by replacing some of a layered network's connections with random layer-skipping connections. This approach could be conveniently implemented in neuromorphic analog hardware, and is biologically-plausible.

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

Jimmy Gammell Sae Woo Nam Adam N. McCaughan

September 10, 2020

1 Introduction

Equilibrium propagation [16] is a learning framework for energy-based networks such as the continuous Hopfield network [8]. It is appealing relative to backpropagation because it is more biologically-plausible, and as a side-effect could be implemented more-easily in neuromorphic analog hardware.

Implementation of equilibrium propagation in [16] was hindered by a vanishing gradient problem whereby networks with as few as 3 hidden layers trained slowly on MNIST [11] - a serious issue given that network depth is critical to performance on difficult datasets [19, 20] and that convergence to a low error rate on MNIST is a low bar to meet. The problem was overcome in [16] by independently tuning a unique learning rate for each layer in the network; however, this approach is unappealing because (1) it introduces additional hyperparameters to tune, (2) it would be inconvenient to implement in neuromorphic analog hardware, and (3) it has not been observed in biological systems.

The purpose of this paper is to demonstrate that in this context the vanishing gradient problem can instead be solved by randomly replacing some of a layered network's connections with layer-skipping connections. [1] Through this modification we have achieved 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 [1] and are roughly the same as those of the layered network with unique, manually-tuned learning rates in [16]. Our method could be implemented with relative ease in any system with configurable connectivity. Layer-skipping connections have been observed in biological brains [3], so the approach is biologically-plausible. Similar techniques have seen success in convolutional [7, 21] and multilayer feedforward [23, 10] 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 and Theory

2.1 Equilibrium propagation

Similarly to backpropagation, equilibrium propagation [16] 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 [16], which applies it to a continuous Hopfield network [8]. The mathematical formulation of the framework can be found in [16].

A major reason backpropagation is not biologically-plausible is that to implement it, each neuron would need two distinct mechanisms for information transmission: one to transmit its activation to shallower neurons during the forward-propagation phase, and another to transmit error-correction information to deeper neurons during the backward-propagation phase [2]. While this is easy in a digital computer that can oversee and manipulate an entire network, it would be cumbersome in hardware (biological or otherwise) consisting

[1]This modification was inspired by small-world topology [22]; however, we have not observed a strong correlation between network performance and common metrics of small-worldness (characteristic path length, clustering coefficient, small-world coefficient).

of many simple, independent computational nodes with limited ability to share information. In contrast, equilibrium propagation consists of a free phase (comparable to forward-propagation) and a weakly-clamped phase (comparable to backward-propagation) during which each neuron only needs to know the activations of neighboring neurons, so only one mechanism for information transmission is needed [16]. In a similar vein, to implement backpropagation each neuron would need mechanisms to compute both an activation value using activations of deeper neurons and error-correction information using that of shallower neurons. For equilibrium propagation a neuron would only need the ability to compute an activation given those of adjacent neurons [16].

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 [16].

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 $\boldsymbol{x} \in \mathbb{R}^p$, $\boldsymbol{h} \in \mathbb{R}^q$ and $\boldsymbol{y} \in \mathbb{R}^r$, and let $\boldsymbol{s} = (\boldsymbol{h}^T, \boldsymbol{y}^T)^T \in \mathbb{R}^{q+r}$ and $\boldsymbol{u} = (\boldsymbol{x}^T, \boldsymbol{s}^T)^T \in \mathbb{R}^n$ be vectors of, respectively, the activations of non-fixed (non-input) neurons and of all neurons in the network. Let $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^n$ 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 ρ denote its activation function; here and in [16],

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

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

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

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

and arbitrary cost function $C : \mathbb{R}^r \to \mathbb{R}_+$; here and in [16] it is a quadratic cost function given by

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

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

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

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

$$\frac{d\boldsymbol{s}}{dt} \propto -\frac{\partial F}{\partial \boldsymbol{s}}. \tag{5}$$

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

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

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

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

1. Clamp $\boldsymbol{x}$ to $\boldsymbol{x}_d$ and perform the free-phase evolution: evolve to equilibrium on the energy function $F(\boldsymbol{u}; 0, \boldsymbol{W}, \boldsymbol{b})$ in a manner dictated by equation 5. Record the equilibrium state $\boldsymbol{u}^0$.

2. Perform the weakly-clamped evolution: evolve to equilibrium on the energy function $F(\boldsymbol{u}; \beta, \boldsymbol{W}, \boldsymbol{b})$ using $\boldsymbol{u}^0$ as a starting point. Record the equilibrium state $\boldsymbol{u}^\beta$.

	Backpropagation	Equilibrium Propagation
Memory	Space to store activation and error term for each neuron	Space to store free and weakly-clamped activations for each neuron
Nonlinear activation function	Yes	Yes
Derivative of nonlinear activation function	Yes	Yes
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
Correction computation	Corrections require dedicated circuitry unique from that implementing propagation	Corrections require dedicated circuitry unique from that implementing evolution
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

Table 1

3. Compute the correction to each weight in the network:

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

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

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

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

   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 ΔW_{ij} and Δb_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.

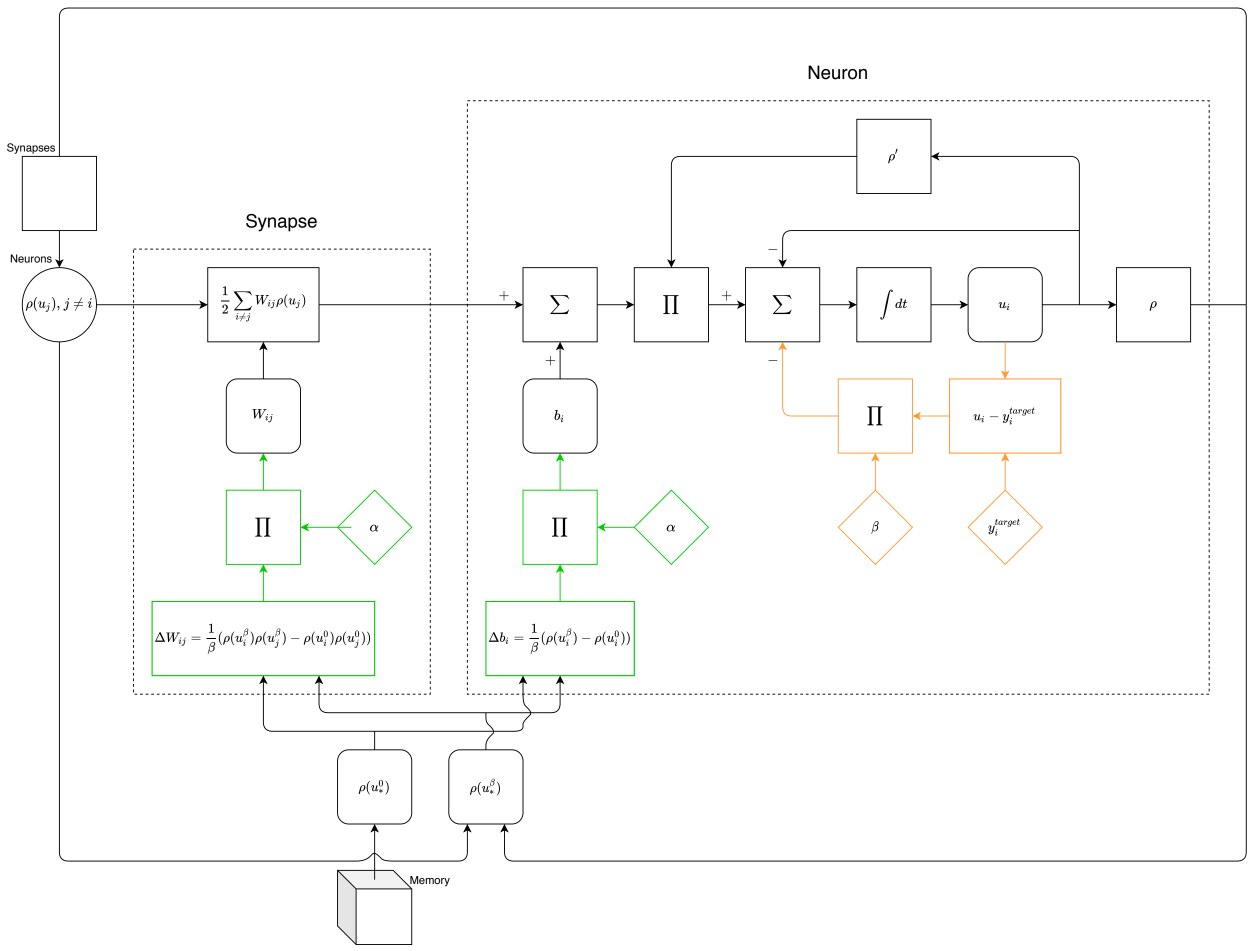

Figure 1: Illustration of the functionality needed to implement equilibrium propagation in hardware. Black lines denote functionality needed in the free phase. Green lines denote functionality to correct parameters. Orange lines denote functionality needed only by output neurons, that is unique to the weakly-clamped phase. There is no functionality unique to the weakly-clamped phase that is needed by all neurons.

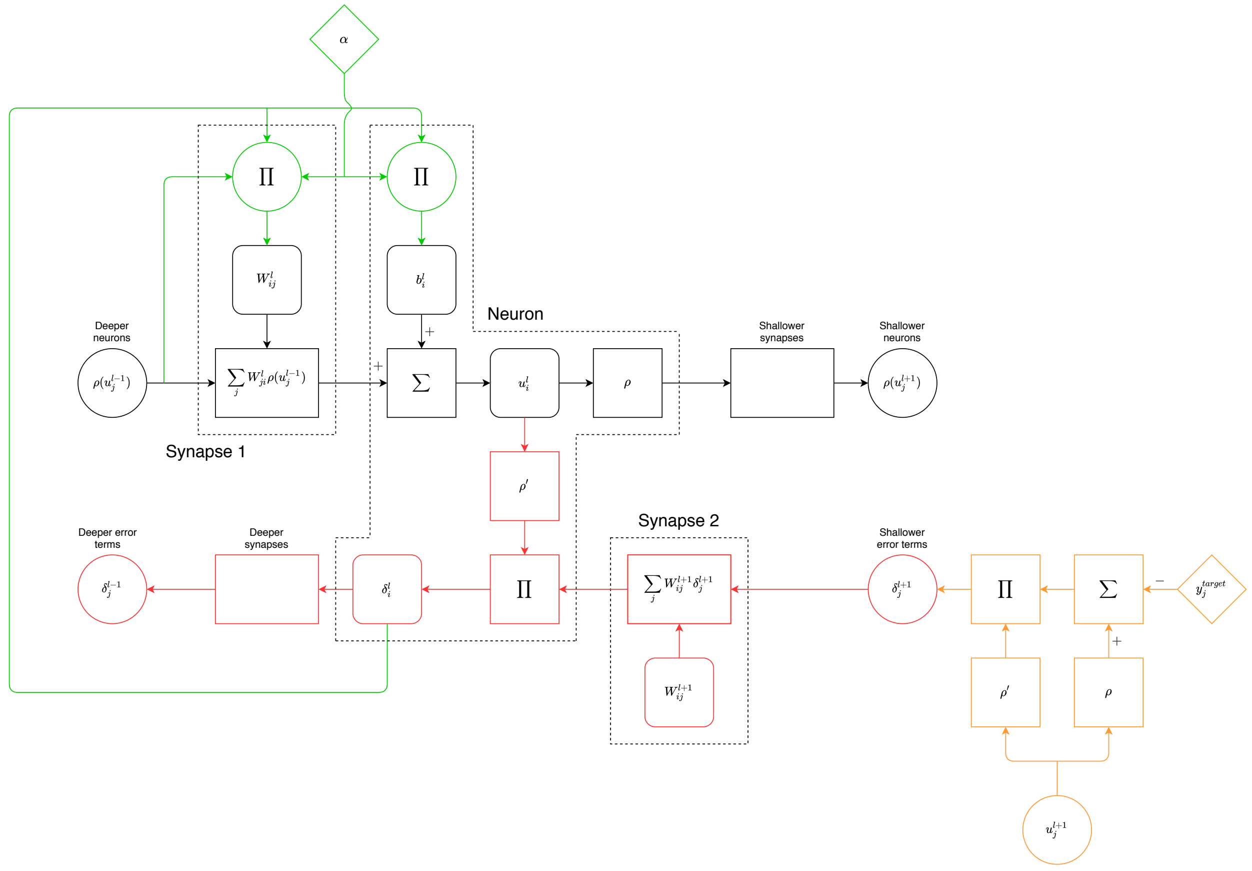

Figure 2: Illustration of the functionality needed to implement backpropagation in hardware. Black lines denote functionality needed in the forwards phase. Green lines denote functionality to correct parameters. Red lines denote functionality unique to the backwards phase that is needed by all neurons. Orange lines denote functionality needed only by output neurons, that is unique to the backwards phase.

2.2 Comparing the computational complexity of equilibrium propagation and backpropagation

2.2.1 Requirements of equilibrium propagation

It follows from equations 2, 4 and 5 that to determine its state, the i-th neuron in a network must compute

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

plus the term $\beta(u_i - y_i^{target})$ for output neurons when using a squared-error cost function, and then integrate the result over time. Parameter correction rules are given by equations 7 and 8. A block diagram of this process is shown in figure 1 and qualitatively describes one way equilibrium propagation could potentially be implemented in hardware.

2.2.2 Requirements of backpropagation

In backpropagation, the activation value of a neuron i in layer l is given by

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

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

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

Weights are corrected using

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

and biases using

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

The block diagram in figure 2 qualitatively describes a way this algorithm could potentially be implemented in hardware.

2.2.3 Comparison

The most-significant difference between the algorithms is that in equilibrium propagation, the free and weakly-clamped phases of training are identical for most neurons and the weakly-clamped phase requires only slight modification to output neurons, whereas in backpropagation these phases very different functionality from essentially all neurons. Another visible difference is that in equilibrium propagation each neuron corresponds to a single synapse whereas in backpropagation a neuron corresponds to two synapses; we do not expect this difference to be significant because a synapse in the former case a synapse takes as inputs the outputs of all neighboring neurons, whereas in the latter case each has inputs from either shallower or deeper neurons (about half as many). While neurons in equilibrium propagation explicitly write their states to memory after the free phase, in backpropagation the need for distinct state variables to hold the activation and error term of each neurons implies a need for the same amount of memory. Various characteristics of both algorithms are compared side-by-side in table 1.

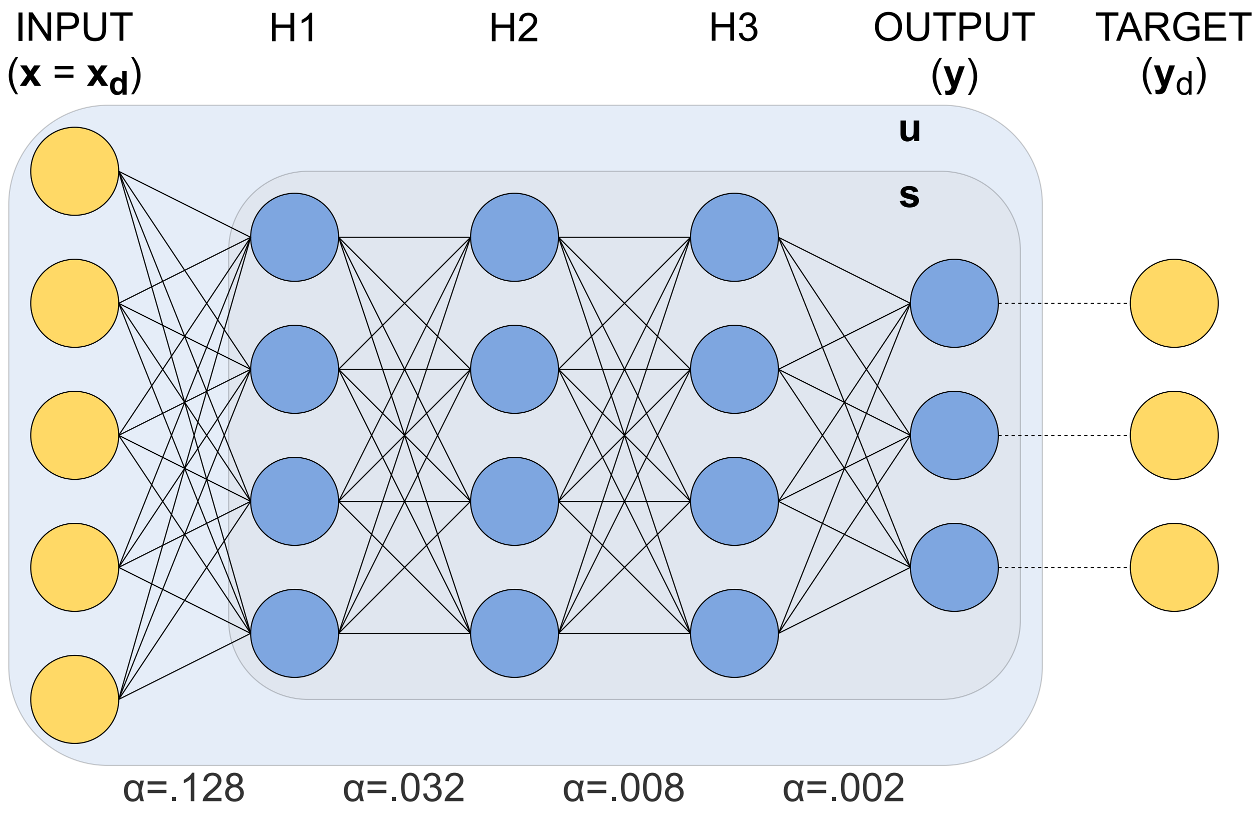

Figure 3: Topology of the layered network tested in [16]. All pairs of neurons in adjacent layers are connected. All connections are bidirectional. To compensate for the vanishing gradient problem, the learning rate is reduced by a factor of 4 each time distance from the output decreases by one layer.

2.3 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.

The vanishing gradient problem is familiar in the context of conventional feedforward networks, where techniques such as the weight initialization scheme in [6], the use of activation functions with derivatives that do not lead to output saturation [17], and batch normalization [9] have been effective at overcoming it. However, in the context of the networks trained in [16], 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.

3 Implementation

We recreated the equilibrium propagation implementation in [16] using the Pytorch library. [2] Like the networks in [16], our networks are continuous Hopfield networks with a hardened sigmoid activation function

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

and squared-error cost function with no regularization term

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

[2]`https://github.com/jgammell/Equilibrium_Propagation_mobile.git`

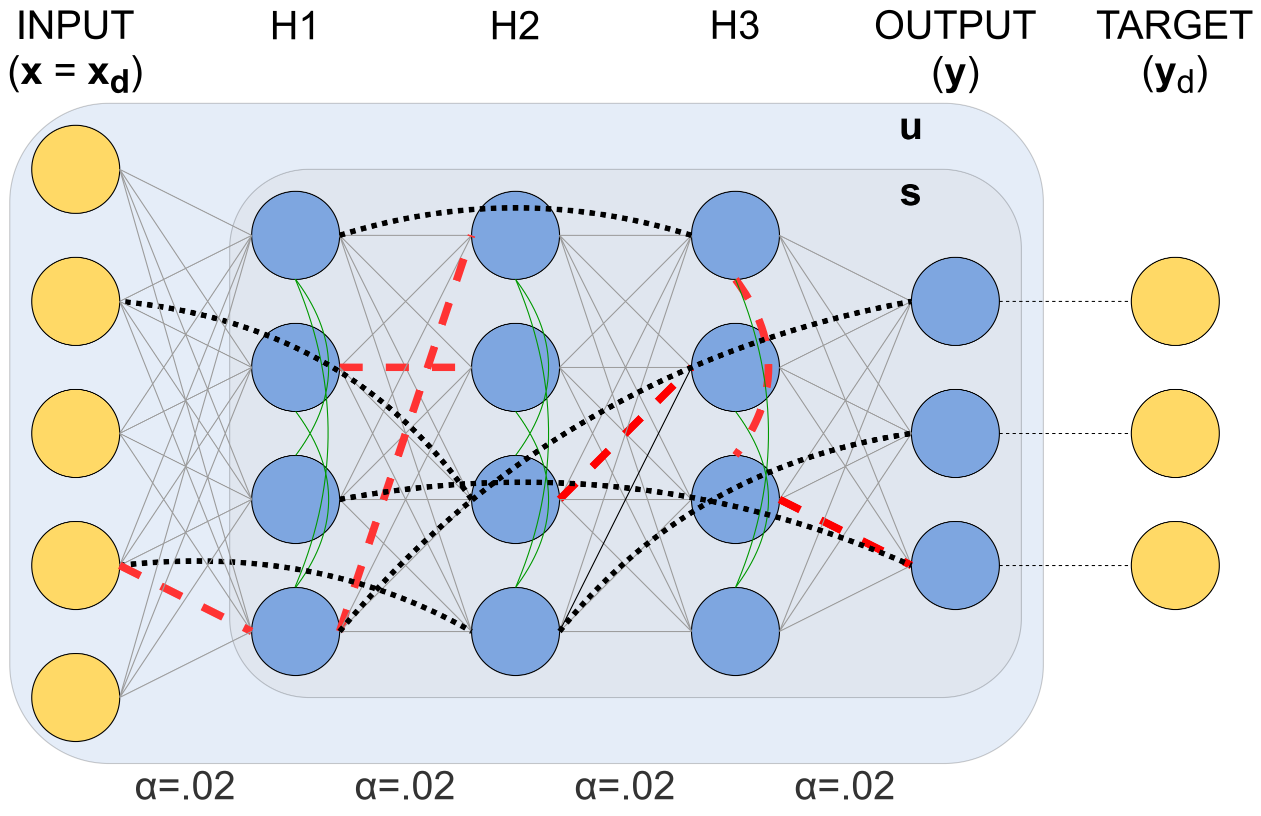

Figure 4: Our modifications to the topology of figure 3 to avoid a vanishing gradient while using a global learning rate. Red dotted lines denote connections that have been removed, black dotted lines denote their replacements, and green solid lines denote added intralayer connections. All connections are bidirectional. This illustration shows a network with $p = 8\%$.

where $\boldsymbol{y}$ is the network's output and $\boldsymbol{y}_d$ is the target output. Tests were run on MNIST [11] grouped into batches of 20 examples, with the 50,000 training examples used for training and the 10,000 validation examples used for computing test errors.

We use two performance-enhancing techniques that were used in [16]: we randomize the sign of β before training on each batch, which has 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 reduce the computational resources needed to approximate the differential equation governing network evolution, and would be unnecessary in an analog implementation that can approximate the equation efficiently. 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.

3.1 Layered topology with per-layer rates

We recreated the 5-layer network evaluated in [16]. It has the standard layered topology shown in 3, and consists of a 784-neuron input layer, 3 500-neuron hidden layers and a 10-neuron output layer. Weights are initialized using the scheme from [6]. As mentioned above, each layer has a unique learning rate; the rates are $\alpha_1 = .128$, $\alpha_2 = .032$, $\alpha_3 = .008$ and $\alpha_4 = .002$ where α_i is the learning rate for the connection weights between layers i and $i+1$ and for the biases in layer i, and the input and output layers are denoted $i = 1$ and $i = 5$, respectively.

3.2 Layered topology with global learning rate

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

3.3 Our topology

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
```

To generate a network with our topology, we use algorithm 1. This topology is illustrated in figure 4. The above algorithm is approximately equivalent to the algorithm for generating a small-world network described in [22] with $p = 1 - (\frac{N_o - 1}{N_o})^n$ for $p \lesssim .2$, where N_o is the number of connections in the network; to contextualize the number of replaced connections we will henceforth describe networks with our topology in terms of p instead of n. We have seen good results with $p \approx 8\%$. We have seen similar results when connections are added to the network, rather than randomly replaced (algorithm 1, without removing pre-existing connections).

For these networks we use a global learning rate of .02 and, as in the networks from sections 3.1 and 3.2, initialize connections between neurons in adjacent layers using the scheme from [6]. For all other

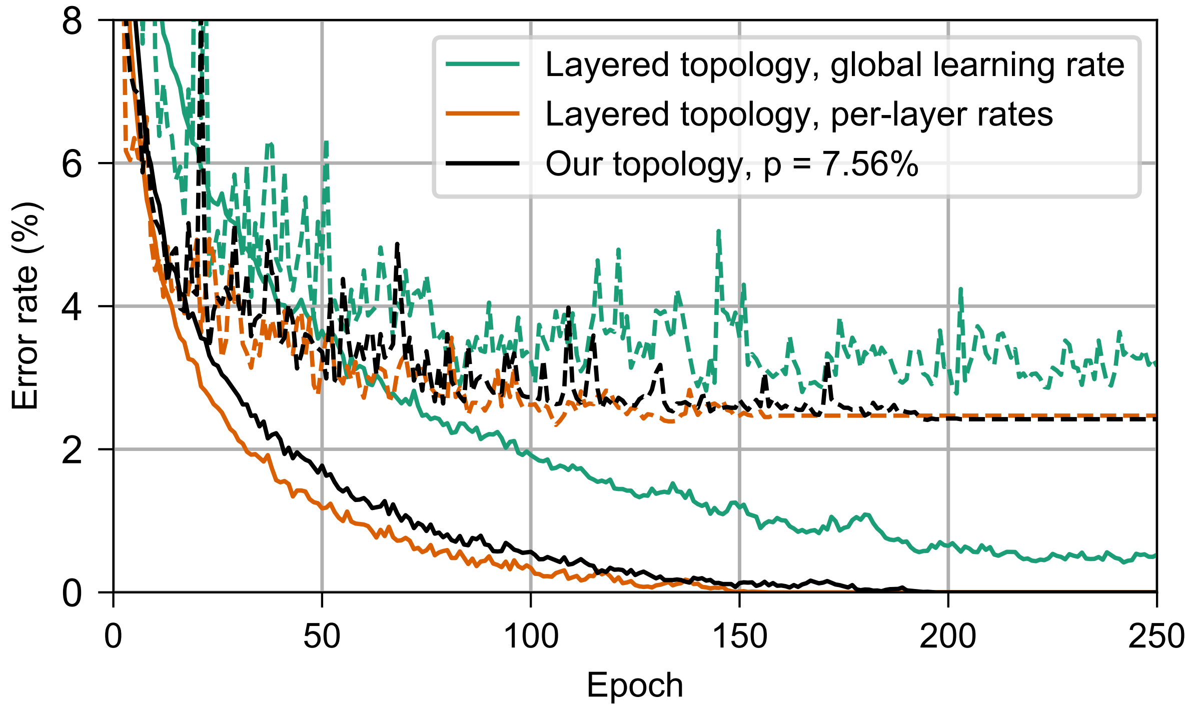

Figure 5: Performance on MNIST of the networks in section 3. Dashed lines show the test error and solid lines show the training error. In green is a layered network with a global learning rate (section 3.2), in orange is a layered network with per-layer rates individually tuned to counter the vanishing 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.

connections we draw initial weights from the uniform distribution $U[-.05, .05]$ where the value .05 was determined empirically to yield good results.

3.4 Deeper networks

To verify that trends hold for deeper networks, we did some limited experimentation on a network with 5 100-neuron hidden layers, with a single global learning rate as in section 3.2 and with our topology, $p = 7.56\%$, as in section 3.3. For these experiments we drew added connections from $U[-.1, .1]$ and used a learning rate of .01; settings were otherwise the same as in preceding sections. These hyperparameters are likely sub-optimal.

4 Results

We compared the networks described in section 3 by observing their behavior while training on MNIST [11]. All networks used $\epsilon = .5$, $\beta = 1.0$, 500 free-phase iterations, 8 weakly-clamped-phase iterations, and were trained for 250 epochs.

4.1 Network performance comparison

Figure 5 illustrates that our network significantly outperforms one with a global learning rate, and achieves close to the same training and test error rates as one with unique learning rates, albeit after around 25%

more epochs. Both our network and the layered network with unique learning rates achieve approximately a 2.5% test error and 0% training error, whereas the layered network with a global learning rate has test and training error rates around .5% higher than the other two networks; it is unclear whether it would converge given enough time, but it is clear that it is inferior.

4.2 Training rates of individual pairs of layers

To observe the extent of the vanishing gradient problem, for each network we tracked the root-mean-square correction to weights in each of its layers during training on MNIST [11]. Figure 6 shows an 11-point centered moving average of these values (without averaging the values are very volatile). It can be seen that for the layered network with a global learning rate, the magnitude of the correction to a typical neuron vanishes with depth relative to the output, with the shallowest weights training around 100 times faster than the deepest weights - this illustrates the vanishing gradient problem. The use of unique learning rates is very effective at making corrections uniform. Our topology with $p = 7.56\%$ is effective at making deeper layers train in a uniform way, but the output layer still trains around 10 times faster than deeper layers; nonetheless, figure 5 suggests that this imperfect solution still yields a significant performance benefit.

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

4.3 Effect of p

We tracked the training error after one epoch of a network with our topology while varying p; the results are shown in figure 7. For $p < .1\%$, there is little improvement in the error rate as p is increased, but there is substantial improvement in the uniformity of the training rates of deep layers. When $p > .1\%$, the deep layers are very uniform, and the error rate starts decreasing with p at a rate that is slightly slower than exponential; at this point there is little improvement in the uniformity of deep layers, but the rate of the shallowest layer appears to move closer to those of the deeper layers.

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

5 Related work

References [12, 24, 15] describe other approaches to locally approximating the gradient of a cost function. References [13, 4] explore the use of a random feedback matrix for backwards connections that is more biologically-plausible than identical forwards and backwards connections. Reference [1] explores the present state of biologically-motivated deep learning, and [2] discusses the criteria a biologically-plausible network would need to satisfy. References [18, 5, 14] discuss analog hardware that could potentially implement equilibrium propagation. References [7, 21, 23, 10] use layer-skipping connections for other types of networks and learning frameworks. References [9, 6] give approaches to solving vanishing gradient problems.

6 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.

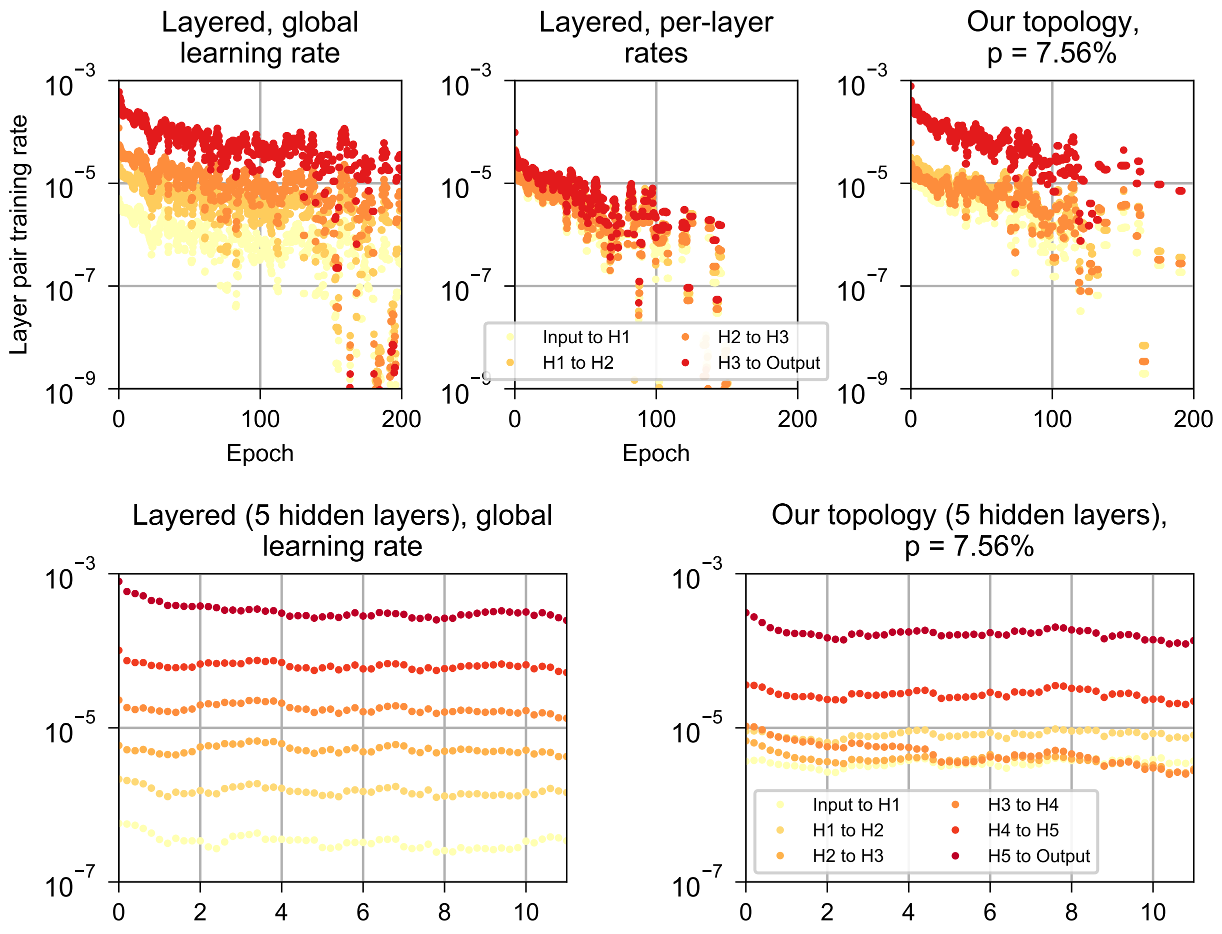

Figure 6: Root-mean-square corrections to weights in different layers while training on MNIST, for the networks in section 3. For clarity, values were subjected to an 11-point centered moving average. (top left) A layered network with a single global learning rate (section 3.2). (top center) A layered network a unique, individually-tuned learning rate for each layer (section 3.1). (top right) A network with our topology, $p = 7.56\%$ (section 3.3). (bottom left) A layered network with 5 100-neuron hidden layers and a single global learning rate (section 3.4). (bottom right) A network with our topology, $p = 7.56\%$, and 5 100-neuron hidden layers (section 3.4). Observe that for the shallower networks 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. For the deeper networks, the same trend is apparent but not as strong; we believe this is due to sub-optimal hyperparameter settings.

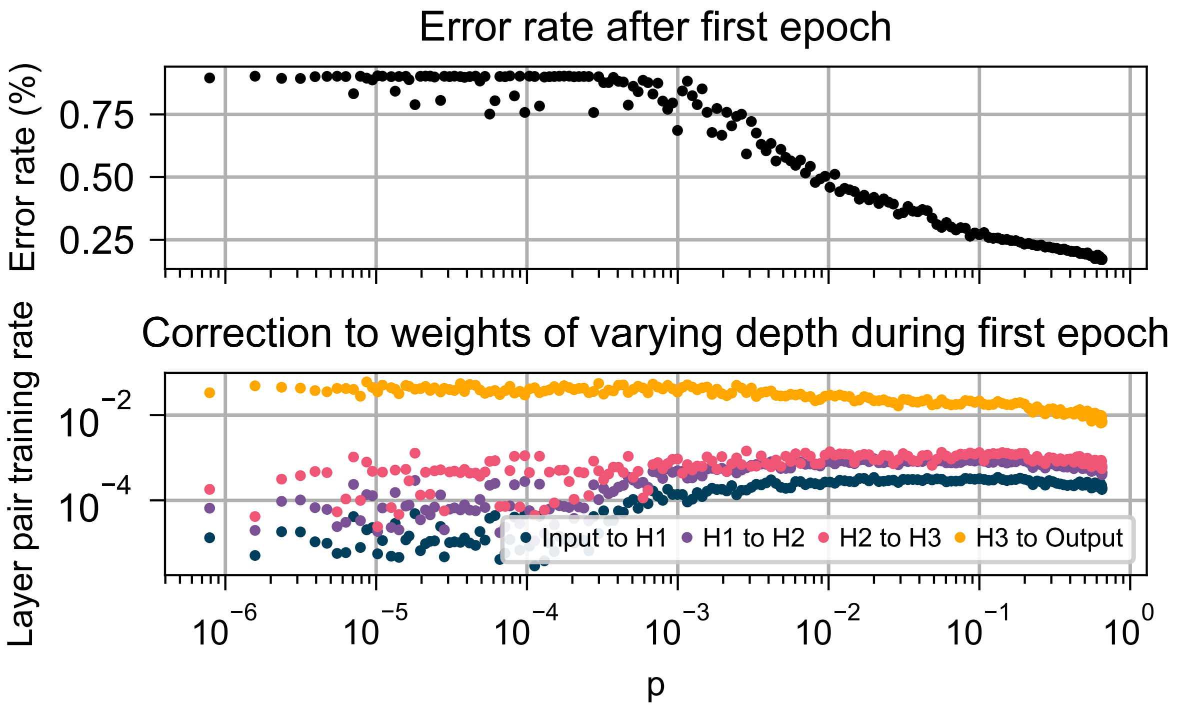

Figure 7: Behavior of our network (section 3.3) with varying p, during the first epoch of training. (top) The training error after one epoch. (bottom) Root-mean-square correction to weights in different layers during the first epoch. Observe that as p is increased, the error rate decreases and the root-mean-square corrections to each layer become more-uniform.

- 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.

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

References

[1] Sergey Bartunov, Adam Santoro, Blake A. Richards, Geoffrey E. Hinton, and Timothy P. Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *CoRR*, abs/1807.04587, 2018.

[2] Yoshua Bengio, Dong-Hyun Lee, Jörg Bornschein, and Zhouhan Lin. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015.

[3] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature*, 2009.

[4] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *CoRR*, abs/1903.02083, 2019.

[5] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Prasad Joshi, Andrew Lines, Andreas Wild, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, PP:1–1, 01 2018.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] John Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the United States of America*, 81:3088–92, 06 1984.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[10] Gokul Krishnan, Xiaocong Du, and Yu Cao. Structural pruning in deep neural networks: A small-world approach, 2019.

[11] Y. LeCun and C. Cortes. The mnist database of handwritten digits, 1998.

[12] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Y. Bengio. Difference target propagation. pages 498–515, 08 2015.

[13] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks, 2014.

[14] Mitchell Nahmias, Bhavin Shastri, A.N. Tait, and P.R. Prucnal. A leaky integrate-and-fire laser neuron for ultrafast cognitive computing. *Selected Topics in Quantum Electronics, IEEE Journal of*, 19:1–12, 09 2013.

[15] Fernando J Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19):2229–2232, 1987.

[16] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation, 2016.

[17] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[18] Jeffrey M. Shainline, Sonia M. Buckley, Adam N. McCaughan, Jeffrey T. Chiles, Amir Jafari Salim, Manuel Castellanos-Beltran, Christine A. Donnelly, Michael L. Schneider, Richard P. Mirin, and Sae Woo Nam. Superconducting optoelectronic loop neurons. *Journal of Applied Physics*, 126(4):044902, Jul 2019.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[20] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks, 2015.

[21] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[22] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 1998.

[23] L. Xiaohu, L. Xiaoling, Z. Jinhua, Z. Yulin, and L. Maolin. 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*, 2011.

[24] Xiaohui Xie and Hyunjune Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15:441–54, 03 2003.