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

Jimmy Gammell
Sae Woo Nam
Adam N. McCaughan

ABSTRACT

Equilibrium propagation is a learning framework for energy-based networks that can be implemented by neurons that perform only one type of computation in the prediction and correction phases of training, and that computes parameter corrections for a given neuron using only the activation values of directly-connected neurons. This makes it an appealing candidate for implementation in neuromorphic analog hardware, and marks a step forward in the search for a biologically-plausible implementation of deep learning. However, in previous implementations of equilibrium propagation, layered networks suffered from 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 counteracted by replacing some of a layered network's connections with random layer-skipping connections in a manner inspired by small-world networks. This approach could be conveniently implemented in neuromorphic analog hardware and is biologically-plausible.

CCS CONCEPTS

• **Computing methodologies** → **Bio-inspired approaches**; **Neural networks**.

KEYWORDS

deep learning, neural network, biologically-motivated, equilibrium propagation, vanishing gradient, small-world, neuromorphic hardware

ACM Reference Format:
Jimmy Gammell, Sae Woo Nam, and Adam N. McCaughan. 2020. Layer-skipping connections facilitate training of layered networks using equilibrium propagation.. In *Proceedings of International Conference on Neuromorphic Systems (ICONS 2020).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The equilibrium propagation learning framework [17] is a method for training a class of energy-based networks, the prototype for which is the continuous Hopfield network [7]. It is appealing because it addresses some major issues that prevent backpropagation from being biologically-plausible, and as a side-effect could be implemented in neuromorphic analog hardware more-easily than backpropagation.

A major reason backpropagation is not biologically-plausible is that to implement it in a system lacking an omniscient supervisor (i.e. code that can easily access all parameters and activations in the network), each neuron would require one mechanism by which to pass its activation to shallower neurons during the forward-propagation phase and a second mechanism by which to pass deeper neurons the necessary information to correct their parameters during the backpropagation phase (including precise information about the manner in which their activations are propagated to the network's output) [2]. In contrast, equilibrium propagation allows a neuron to compute corrections to its parameters using only the activations of neighboring neurons, allowing both the prediction and correction phases of training to be carried out using a single information-transfer mechanism [17].

Another issue with backpropagation is that neurons are required to perform distinct operations during the forward and backward propagation phases - in the forward propagation phase they pass forward an activation value that is a function of the activations of deeper neurons, whereas in the backward propagation phase they pass backwards an error term that is a different function of error terms from shallower neurons. This implies two distinct sets of computational circuitry in each neuron. In equilibrium propagation the only difference between the prediction and correction phases of training is that in the correction phase output neurons are weakly-clamped towards the target output; the dynamics by which the network evolves are otherwise the same [17].

Because of these differences, relative to backpropagation, equilibrium propagation could be implemented more-easily in neuromorphic analog hardware, and it is more-plausible that it could be carried out by observed architecture in biological brains.

It has been demonstrated [17] that a continuous Hopfield network with a basic layered topology can be trained on MNIST [11] through the equilibrium propagation framework. However, previous implementations encountered a vanishing gradient problem (section 2.2) that significantly impedes training of networks with several hidden layers. Given that network depth is critical for performance on difficult datasets [20, 21], and that convergence to a low error rate on MNIST is a low bar for a network to meet, this is a nontrivial issue. It has been demonstrated [17] that the problem can be solved by using a unique learning rate for parameters at different depths in the network, with deeper parameters trained with larger learning rates to counteract gradient attenuation with depth. This approach is unappealing because (1) it introduces additional

ICONS 2020, July 2020, Chicago, Illinois

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

hyperparameters that must be tuned, (2) it would be inconvenient to implement and tune unique learning rates in analog hardware, and (3) it has not been observed in biological systems.

The purpose of this paper is to introduce a modification to the layered topology that can counteract the vanishing gradient problem in the context of energy-based networks trained using equilibrium propagation. The modification is inspired by small-world topology [23], and consists of first fully-connecting a network's layers, then replacing some of its connections with random layer-skipping connections. We achieve 0% training error (out of 50,000 examples) and under 2.5% test error (out of 10,000 examples) on MNIST using a network with 3 hidden layers and no regularization term in its cost function. These error rates are comparable to those achieved by other biologically-motivated networks [1] and are the same as those achieved in the original paper using a layered topology and manually-tuned per-layer learning rates [17], albeit they are achieved after around 25% more epochs. Our method adds only one additional hyperparameter - the number of connections to randomly replace - while removing $L-2$ parameters, where L is the number of layers in the network. It could be implemented with relative ease in any system with configurable connectivity. [1] Layer-skipping connections have been observed in biological brains [3], so this approach is biologically-plausible. Similar techniques have been used with some success to modify convolutional [6, 22] and multilayer feedforward [10, 24] networks with varying degrees of success. Our findings in this paper show 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

Equilibrium propagation [17] is a learning framework for energy-based networks that trains their parameters by approximating gradient descent on some arbitrary cost function. It is applicable to any network with dynamics characterized by evolution to a fixed point of an associated energy function; in this and in [17], it is implemented on a continuous Hopfield network [7]. For a variety of reasons outlined in [2], backpropagation is not biologically-plausible. One of the major reasons is that to correct the parameters of a given neuron, backpropagation requires precise information about the activations and nonlinearities of all neurons in the corresponding feedforward path; equilibrium propagation avoids this problem by correcting a neuron's parameters using only the activations of neurons to which it directly connects. Another major reason backpropagation is not biologically-plausible is that it requires neurons to perform distinct computations (implemented using distinct circuitry) in the forward and backward propagation phases of training; in contrast, equilibrium propagation requires only one behavior of neurons in both the prediction (free) and correction (weakly-clamped) phases of training. In addition to enhancing its biological plausibility, these traits make equilibrium propagation appealing as a framework to implement on neuromorphic analog hardware because they would limit the complexity required of neurons and the amount of infrastructure needed to operate and train them.

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 [17].

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 by vectors $\boldsymbol{x} \in \mathbb{R}^p$, $\boldsymbol{y} \in \mathbb{R}^q$ and $\boldsymbol{h} \in \mathbb{R}^r$, 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 [17],

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

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 $E : \mathbb{R}^n \rightarrow \mathbb{R}$,

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

and arbitrary cost function $C : \mathbb{R}^r \rightarrow \mathbb{R}_+$; here and in [17] 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 $F : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 $\boldsymbol{u}^\beta$.

[1] Our current implementation also requires manual tuning of the initial weights of intralayer and layer-skipping connections, but we have found that networks are not very sensitive to these initializations and that performance is good as long as they are in the same ballpark as the rest of the network's initializations.

(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.

2.1.2 Approximation of equation of motion. In analog hardware the dynamics described by equation 5 could be implemented efficiently using leaky integrator neurons, but on digital hardware it is necessary to discretize and approximate these dynamics; we now describe the approximation used here and in [17]. Let $\boldsymbol{s}[n]$ denote the state of the network after n iterations of the approximation, N denote the total number of iterations and ϵ the size of each iteration. $\boldsymbol{s}[0]$ is the initial state of the network; here and in [17], $\boldsymbol{s}[0] = \mathbf{0}$ before the first epoch, and in subsequent epochs $\boldsymbol{s}[0]$ is the equilibrium state of the network from the last time it executed a free phase on the same batch. Let

$$s_i[n] = s_i[n-1] - \epsilon\frac{\partial F}{\partial s_i},\ n = 1, \ldots, N \tag{9}$$

(where $\beta = 0$ in the free phase); then $\boldsymbol{s}[N]$ approximates the state after time ϵN that would be given by equation 5. We denote the number of iterations in the free and weakly-clamped phases N_{free} and $N_{weakly-clamped}$, respectively. It is only necessary to run the weakly-clamped phase long enough to observe the initial direction in which the network evolves and to allow the perturbation of the output to influence all layers in the network, so typically for a network with L layers, $N_{free} >> N_{weakly-clamped} > L$.

2.2 Vanishing gradient problem

It has been observed [17] that energy-based networks with a layered topology trained using equilibrium propagation suffer from a vanishing gradient problem whereby the rate of training of a weight decreases approximately exponentially with the number of layers between that weight and the output layer. This is problematic because it has been observed to increase the time needed to train a network, and gradients spanning several orders of magnitude could be hard to represent in analog implementations of the framework with limited bit depth. The vanishing gradient problem is familiar in the context of conventional networks trained through backpropagation, where the composition of many layers can impede training by amplifying or attenuating the effect of changes to deep neurons as these changes propagate to the output layer, leading to a vanishing or exploding gradient. For those networks the problem can be effectively addressed by initializing weights to make activation variances and backpropagated gradient variances approximately uniform with respect to depth as described in [5], and by using activation functions with derivatives that do not cause neurons to saturate [18], such as rectified linear units $\rho(x) = \text{Max}\{0, x\}$ or hardened sigmoids (equation 1). Batch normalization [9] is another effective technique for promoting uniform training in a network; inputs to each layer are whitened (the input distribution $\mathbf{X}$ to each layer is transformed to have $E[\mathbf{X}] = \mathbf{0}$ and $Var[\mathbf{X}] = \mathbf{1}$) during each batch to prevent changes to deep parameters from changing the input distribution to shallower layers and requiring adjustment to compensate.

The weight initialization scheme from [5] and the use of hardened sigmoid activation functions do not solve the vanishing gradient problem in the context of equilibrium propagation. Batch normalization has not been tried in this context, but is not biologically-plausible. This suggests that the cause of the vanishing gradient problem here is different than in a conventional network. The situation in energy-based networks is more-complicated because there is no straightforward relationship between the activations of two given neurons; instead, neurons evolve as a coupled dynamical system.

In [17] the vanishing gradient problem was addressed by using unique learning rates for each layer, chosen to make the magnitudes of corrections to neurons' parameters uniform regardless of depth in the network. While this method is effective, it is unappealing because it seems unlikely to take place in a biological brain, it would add complexity to an analog implementation of the framework, and it introduces more hyperparameters that must be tuned to train a network. Our topology effectively addresses the problem without these unappealing characteristics.

2.3 Small-world networks

Our topology was inspired by small-world graph topology as described in [23]. We have observed that energy-based networks with layered topology experience exponential attenuation of the gradient of a given parameter as its depth in the network increases, so it seems reasonable to expect that the vanishing gradient problem can be reduced by decreasing the number of connections needed to move between a given neuron and the output layer. Small-world topology offers a way to do this while largely preserving clustering between neurons within the same layer. It is also a topology that has been observed in biological brains [3]. Essentially, a layered network with fully-connected layers can be viewed as a network with a large typical path length between pairs of neurons and a large amount of clustering between nearby neurons. By randomly replacing a small proportion of its connections, its typical path length can be greatly reduced without significantly reducing its clustering. Quantitative metrics of a network's small-worldness are described in appendix A, but in our experiments these metrics have little correlation with network behavior (we discuss this further in section 6.2).

Algorithm 1: Algorithm to generate small-world graph

Input: A regimented graph G
Input: A probability p
// Probability with which to replace a given preexisting edge
Output: Resulting modified graph
for *edge* $e \in G$ **do**
  $p' \sim U[0,1]$;
  if $p' \leq p$ **then**
    Add edge between random unconnected pair of vertices;
    Remove e;
return G

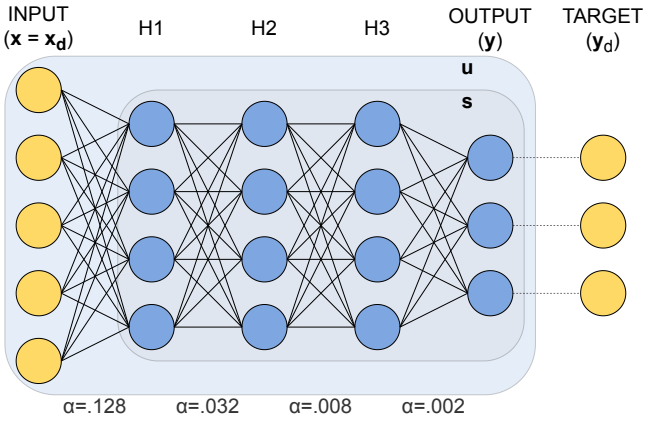

Figure 1: Topology of the layered network tested in [17]. All pairs of neurons in adjacent layers are connected, and there are no additional connections. All connections are bidirectional. The learning rate for weights is reduced by a factor of 4 each time distance from the output decreases by one layer, to compensate for the vanishing gradient problem.

2.3.1 Algorithm for generating a small-world graph. Algorithm 1 was introduced in [23] to convert a regimented graph into a small-world graph. It was implemented on a lattice graph but works equally-well for the topology of a layered neural network. As $p \approx 0$ is increased, the mean shortest path between a pair of vertices in the graph (L from appendix A) decreases rapidly while preexisting clusters remain largely intact (C from appendix A does not significantly decrease). To generate a small-world graph, p is chosen so that the graph has a large amount of clustering and a small mean shortest path, relative to a random graph with the same number of vertices and edges (σ from appendix A is large).

3 IMPLEMENTATION

We implemented [2] the equilibrium propagation learning framework [17] using the Pytorch library, and verified that we could

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

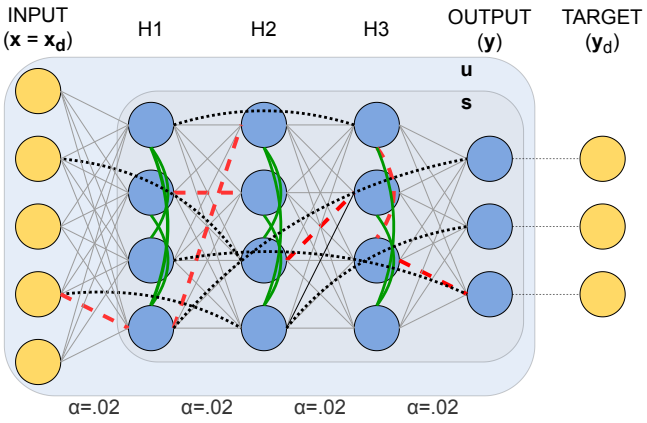

Figure 2: Changes to the layered topology to compensate for the vanishing gradient problem while using a single learning rate for all weights. Red dotted lines denote connections that have been removed and black dotted lines denote their replacements. Green solid lines denote added intralayer connections. In this illustration connections have been replaced by a random layer-skipping connection with probability $p \approx 8\%$. All connections are bidirectional.

recreate the experiments ran in [17]. We follow the procedure described in section 2.1; we use a hardened sigmoid activation function (equation 1) and a squared-error cost function with no regularization term (equation 3). Testing was done on the MNIST dataset [11] with input-output pairs grouped into batches of 20; training was done on the 50,000 training pairs and testing on the 10,000 validation pairs.

We use two performance-boosting techniques described in [17]. We randomize the sign of β at the outset of each batch, which provides a regularization effect (note that parameter corrections are divided by β, so are changed to minimize cost regardless of its sign). We use persistent particles: after the first epoch, before training on a batch the network's state is initialized to its free equilibrium state from the last time it trained on the same batch. This reduces computational resources needed for training by reducing the number of iterations of equation 5 needed to reach equilibrium, and would be unnecessary in an analog implementation; note that training and test error rates during early epochs are artificially inflated as a result of this technique.

3.1 Layered topology with unique learning rates

We recreated the 5-layer network used in [17]. It has a standard layered topology shown in 1.

The network consists of an input layer with 784 neurons, 3 hidden layers with 500 neurons each, and an output layer with 10 neurons. There are connections between every pair of neurons in adjacent layers, no connections between neurons in the same layer, and no layer-skipping connections.

The weight matrix was initialized using the Glorot-Bengio initialization scheme [5]: a weight connecting layer j with n_j neurons to

layer $j+1$ with n_{j+1} neurons is drawn from a uniform distribution

$$U[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}]. \quad (10)$$

For effective training it is necessary to use unique learning rates to train the weights connecting each pair of layers. We use the same learning rates that were used for experiments in [17]: .128, .032, .008, .002, in order of the deepest to the shallowest layer; to counter the vanishing gradient problem the learning rate is reduced by a factor of 4 each time the distance from the output layer decreases by one layer. Each neuron has a bias term that is trained using the learning rate corresponding to the weights of the neuron's input connections, e.g. the biases of the first hidden layer are trained with a learning rate of .128.

3.2 Layered topology with single learning rate

To provide a point of reference, we ran tests on a network that is identical to that in section 3.1 except with a single learning rate of .02 across the entire network. It performs poorly due to the vanishing gradient problem.

3.3 Our topology

Algorithm 2: Algorithm to produce our topology

Input: A network N with layered topology
`// described in section 3.2`
Input: An integer n
`// number of connections to replace`
Output: A network with our modified topology
for *hidden layer* $l \in N$ **do**
  Add edge between each pair of neurons in l
for $i \leftarrow 1$ **to** n **do**
  Randomly select connection $c \in N$;
  Add connection between random unconnected pair of neurons $\in N$;
  `// Do not allow self connections`
  `// Do not allow connections between two input neurons or between two output neurons`
  Remove c;
return N

To generate a network with our topology, we use algorithm 2. The topology is illustrated in figure 2. For $p \lesssim .2$, algorithm 2 is roughly equivalent to algorithm 1 with $p = 1 - (\frac{N_o-1}{N_o})^n$, where N_o is the number of connections in the network [3] ; to contextualize the number of replaced connections, we will henceforth describe networks with our topology in terms of p and not n. We have also tried adding new connections without removing existing ones, and observed roughly the same performance; in this paper we replace connections to limit the number of parameters added to the network. We do not add connections within the input or output layers and we do not allow neurons to connect to themselves. We have seen good results with $p \approx 8\%$.

This topology allows us to train the network with a uniform learning rate of .02 across the entire network. Weights of connections between pairs of neurons in adjacent layers are still initialized based on equation 10. Weights of intralayer connections and layer-skipping connections are drawn from the uniform distribution $U[-.05, .05]$, where the value .05 was chosen empirically to optimize network performance.

3.4 Tracking the training rates of individual pairs of layers

We find it informative to track the root-mean-square magnitude of corrections to weights in individual layers; if $\Delta w_{ij}^l(b)$ denotes the correction to the connection weight between the i^{th} neuron in layer l and the j^{th} neuron in layer $l+1$ in response to batch b and N^l denotes the number of neurons in layer l, we define

$$\Delta w^l(b) := \sqrt{\frac{\sum_{i=1}^{N^l}\sum_{j=1}^{N^{l+1}}(\Delta w_{ij}^l(b))^2}{N^l N^{l+1}}} \quad (11)$$

as a metric of the extent to which the weights between layers l and $l+1$ trained in response to batch b. We ignore the training of the weights of intralayer and layer-skipping connections.

This measurement is volatile, so for clarity we plot the average of $\Delta w^l(b')$ for b' in a neighborhood of batch b: for some n, we define

$$\Delta \hat{w}^l(b) := \frac{1}{2n+1}\sum_{b'=b-n}^{b+n}\Delta w^l(b'). \quad (12)$$

Comparing the traces $\Delta \hat{w}^l(b)$ vs. b for different layers reveals the extent of the vanishing gradient problem.

4 RESULTS

We compared the three network topologies described in sections 3.1, 3.2, and 3.3 on the MNIST dataset [11]. The only hyperparameter that differed between networks was the learning rate(s); the layered network with unique learning rates used rates of .128, .032, .008, and .002 for layers from output to input, and the layered network with one learning rate and the network with our topology used a learning rate of .02. All networks used $\epsilon = .5$, $\beta = 1.0$, $N_{free} = 500$, $N_{weakly-clamped} = 8$ and were trained for 250 epochs with 50,000 training examples, 10,000 test examples and a batch size of 20. To generate our topology we used algorithm 2 with $p = 7.56\%$ (100,000 replacements made).

4.1 Network performance comparison

We tracked the error rates of the three networks as they were trained and found that the network with our topology significantly outperforms the basic network with one learning rate, and achieves the same error rates as the basic network with unique learning rates after around 25% more epochs. These results are shown in figure 3.

Notice that the layered network with unique learning rates converges to 0% training error and 2.5% test error in around 150 epochs and the network with our topology converges to 0% training error and 2.5% test error in around 190 epochs, whereas the layered

[3] Algorithm 1 with large p produces more layer-skipping connections than algorithm 2 with large n; it can be shown that for any n, algorithm 2 is equivalent to algorithm 1 with $p = N_l - N_l \sqrt[N_o]{\frac{N_l}{N_o+N_l} + \frac{N_o}{N_o+N_l}(1 - \frac{N_o+N_l}{N_o N_l})^n}$ where N_l is the number of unconnected pairs of neurons that are candidates to share a layer-skipping connection.

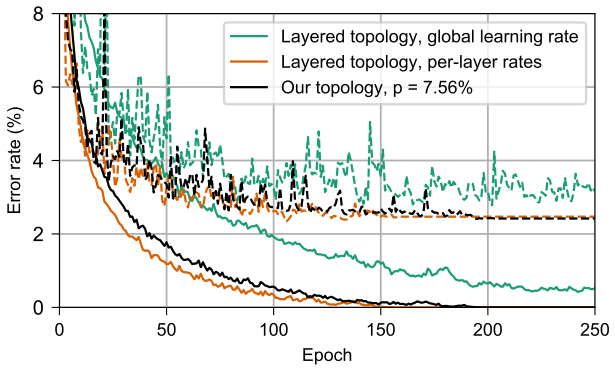

Figure 3: Comparison of performance of network topologies on MNIST dataset. Dotted lines show test error and solid lines show training error. In green is a network with the layered topology and a single learning rate (section 3.2). In orange is a network with the layered topology and unique learning rates (section 3.1), tuned to counter the vanishing gradient problem; this is a recreation of the 5-layer network in [17]. In black is a network with our topology, $p = 7.56\%$ (section 3.3). The network with a layered topology and a single learning rate performs poorly because it suffers from the vanishing gradient problem. The problem can be solved by introducing unique learning rates, or by implementing our topology.

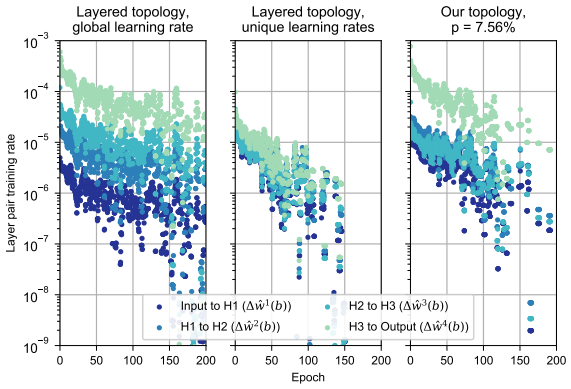

Figure 4: Extent of the vanishing gradient problem in different network topologies while training on MNIST dataset. (left) A network with the layered topology and a single learning rate. The training rate of a layer decreases by a factor of ~ 4 with each additional layer. (center) A network with the layered topology and unique per-layer learning rates. The vanishing gradient is counteracted by hand-tuning a learning rate hyperparameter for each layer. (right) A network with our topology, $p = 7.56\%$. The new topology eliminates the need for hand-tuning per-layer learning rate hyperparameters.

network with one learning rate fails to converge in 250 epochs and has training and test error around .5% higher than the other two networks. While it is possible that the basic network with one learning rate would converge given enough time, it is clearly inferior to the other two networks. It is also apparent that on the MNIST dataset, our network with a single learning rate is practically interchangeable with the layered network with unique learning rates.

4.2 Training rates of individual pairs of layers

We tracked the training rates of individual pairs of layers as described in equations 11 and 12 and found evidence that the extent of the vanishing gradient problem is the primary cause of the performance disparity seen in section 4.1. These results are shown in figure 4.

Notice that in the layered network with one learning rate there is a significant spread in the training rates of pairs of layers, with the deepest pair training at around 1% of the rate of the shallowest pair. This problem is solved very effectively in the layered network with unique learning rates. The problem appears to be solved effectively for the deepest 3 pairs in the network with our topology, but the output layer still trains significantly faster than the deeper 3 layers. This makes sense if we assume that the important factor in a layer's training rate is its expected path length to the target layer, because every neuron in the output layer connects to the target layer through a single connection, whereas paths starting at deeper neurons must first pass through the output layer before connecting to the target layer.

4.3 Error rate after one epoch as connections are added

We tracked the training error after one epoch of a network with our topology while varying p; the results are shown in figure 5.

Three regimes are evident in the figure. 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.

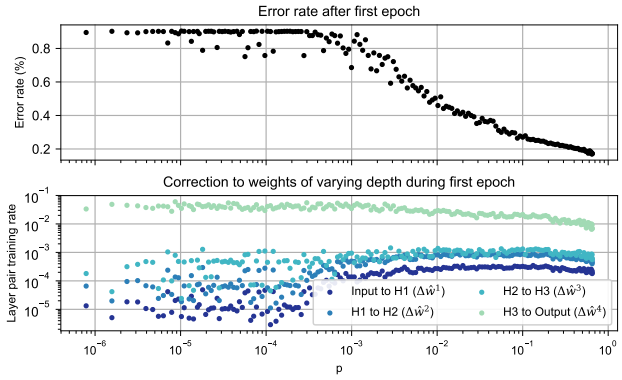

Figure 5: Performance of a network with varying p during the initial stages of training. (top) The training error after one epoch. (bottom) Training rate for each layer after one epoch. There is little improvement for $p < .1\%$, after which the error rate decreases at a rate that is slightly slower than exponential with p. Improvement begins after the training rates of non-output layers converge, and improvement beyond that point is associated with convergence between the rates of the output layer and the non-output layers.

5 RELATED WORK

Much research has been done in pursuit of a biologically-plausible deep learning algorithm. References [12, 16, 25] describe other algorithms that approximate the gradient of a cost function locally. Reference [13] demonstrates that networks can train through backpropagation with randomly-initialized feedback weights between neurons; the need for symmetric connection weights is not solved by equilibrium propagation and is not biologically-plausible, so this finding could potentially be applied here as well. Reference [15] implements equilibrium propagation using spiking neurons; this type of activation function is believed to be more biologically-plausible than a hardened sigmoid function. Reference [2] discusses the criteria a biologically-plausible algorithm would need to satisfy. Reference [1] surveys promising biologically-motivated algorithms and evaluates their effectiveness on hard datasets.

References [4, 14, 19] discuss neuromorphic architecture that could potentially implement equilibrium propagation.

Layer-skipping connections have been explored in other contexts. References [6, 22] use layer-skipping connections as a linear transformation in parallel with nonlinear layers to great effect in very-deep convolutional networks. References [10, 24] apply small-world topology to networks trained using backpropagation.

References [5, 9] effectively counter the vanishing gradient problem in the context of deep networks trained using backpropagation.

6 DISCUSSION AND FUTURE RESEARCH

6.1 Characteristics of an effective network

Our experiments indicate that networks with our topology perform poorly relative to a layered network with a single learning rate until p is in the ballpark of 8%. It seems likely that the training rate will continue to improve as p is increased to around 70%; however, we suspect this improvement would be at the cost of a higher test error rate, because as p increases a network's topology looks less-layered and more like a sparsely-connected single-layer network. We did not observe deterioration of a network's generalization ability for $p = 7.56\%$; it is possible that such deterioration is not evident when training on a dataset as simple as MNIST, but would become apparent on a more-difficult dataset such as CIFAR or ImageNet.

6.2 Factors influencing network performance

Contrary to our expectations, we found that a network's performance did not track closely with any of the small-world metrics (Appendix A). Good performance was achieved when algorithm 2 was executed with $p \approx 8\%$, which leads to a network with small L and C that was therefore not small-world. In retrospect we believe these metrics to be poor predictors of network performance. One reason for this may be that the characteristic path length L quantifies the shortest path between two neurons, but the total number of paths is also important because a neuron's behavior is dictated by an affine transformation of all its inputs. Similarly, it can be important for a network to retain its layered nature in order to generalize well, but the clustering coefficient C does not appear to be a good metric for this - e.g. a layered network without intralayer connections has a small clustering coefficient because the neighbors of a given neuron are not connected to one-another.

There are several factors other than path length that we believe affect performance with respect to p, and that may disguise relationships between performance and the small-world metrics. It has been shown [6, 9] that in deep convolutional networks, performance can be improved by using layer-skipping connections to create an identity mapping or dimension-preserving linear transformation in parallel with groups of nonlinear layers to reframe their task as learning the residual of the output of earlier layers; in our implementation layer-skipping connections form a linear transformation of earlier layers, and this fact may improve performance even though only some of the dimensions of the deeper layers are transferred. Due to the discrete approximation of the network's dynamics whereby a neuron l layers from the output is not influenced by a perturbation of the output until iteration $l + 1$ of the approximation, deep layers experience less evolution than shallow layers during the weakly-clamped phase; since layer-skipping connections allow deeper layers to begin evolving sooner, they probably reduce this effect. As connections are replaced, a network looks less layered and more like a sparsely-connected single-layer network, so we expect that there is a tradeoff between the rate at which a network trains and its ability to generalize. In this paper our topology did not deteriorate test error relative to the original layered network, but it is possible that a network on a harder dataset such as CIFAR or ImageNet or with a higher value of p would see deterioration as its topology deviates from a layered topology.

6.3 Directions for Future Research

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

- Finding and mathematically-justifying a weight initialization scheme for added and replaced connections.

- Exploring the effect of adding (instead of replacing) layer-skipping connections, training a network, then removing the connections and training further, to see if there is a lasting benefit after removal; if so, connections could be used to quickly train a network in analog hardware, then removed to reduce the network's power consumption and heat load.
- Exploring the effect of allowing non-output connections to be directly affected in the weakly-clamped phase of training, in effect allowing neurons outside the output layer to connect to the target layer; this would likely reduce the gap between the training rates of output and hidden layers seen in figure 4.
- Exploring a network's test error as p is increased, especially on difficult datasets such as CIFAR or ImageNet where generalization is very important.

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] 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.

[5] 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.

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

[7] 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.

[8] Mark Humphries and Kevin Gurney. Network 'small-world-ness': A quantitative method for determining canonical network equivalence. *PloS one*, 3:e0002051, 02 2008.

[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] Peter O'Connor, Efstratios Gavves, and Max Welling. Training a network of spiking neurons with equilibrium propagation. 2018.

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

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

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

[19] 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.

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

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

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

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

[24] 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.

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

Appendix A METRICS OF A GRAPH'S SMALL-WORLDNESS

A graph's small-worldness can be characterized by its characteristic path length L and its clustering coefficient C [23]. A graph is small-world if L is small relative to that of a random network with the same number of vertices and edges and C is not substantially smaller than that of the random network; this situation can be quantified by a small-world coefficient σ [8]. The characteristic path length is the minimum number of edges in a path joining a pair of vertices, averaged over all pairs of vertices in the network. For a graph with N vertices where $l(v_i, v_j)$ denotes the smallest number of edges needed to connect vertex v_i to vertex v_j,

$$L = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} l(v_i, v_j). \tag{13}$$

The neighborhood of a vertex is the set of vertices with which it shares an edge. The clustering coefficient is the proportion of pairs of vertices in the neighborhood of a given vertex that share an edge, averaged over all vertices in the network. For a graph with N vertices, if $\mathrm{N}(v_i)$ denotes the neighborhood of vertex v_i and

$$c(v_j, v_k) = \begin{cases} 1 & v_k \in \mathrm{N}(v_j) \\ 0 & \text{else} \end{cases},$$

$$C = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|\mathrm{N}(v_i)|} \sum_{\substack{v_j \in \mathrm{N}(v_i) \\ v_k \in \mathrm{N}(v_i) \\ j \neq k}} c(v_j, v_k) \tag{14}$$

The small-worldness of a graph can be quantified by a small-world coefficient

$$\sigma = \frac{C/C_r}{L/L_r} \tag{15}$$

where C_r and L_r are the expected clustering coefficient and characteristic path length of a random graph with the same number of vertices and edges as the graph under consideration.