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

Jimmy Gammell, Adam McCaughan

February 24, 2020

Abstract

Equilibrium propagation is a learning framework for energy-based networks that is an appealing candidate for implementation through neuromorphic analog hardware and is a step forward in the search for a biologically-plausible way to implement deep learning, because it can be implemented with neurons that need perform only one type of computation in the prediction and correction phases of training and because a correction to a parameter of a neuron can be computed with knowledge only of the activations of neurons to which it is directly connected. However, networks of any depth suffer a vanishing gradient problem that has not yet been solved in a simple or biologically-plausible way. In this paper we present a modification that can be made to a network's topology that counteracts the vanishing gradient problem to facilitate training, that is biologically-plausible and would be convenient to implement in analog hardware.

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 [8]. It is appealing as a framework that could be implemented in neuromorphic analog hardware because unlike in backpropagation, neurons are required to perform only one type of computation in the prediction of correction phases of training, and the parameters of a neuron can be updated using only the activations of neurons to which it is directly connected - there is no need for a feedback path through which to transmit information about the parameters and states of neurons across the entire network. These traits also address several of the problems [3] that make backpropagation biologically-implausible.

It has been demonstrated that a continuous Hopfield network with a basic multilayer topology can be trained on MNIST [11]. However, previous implementations encountered a vanishing gradient problem that significantly impedes training of networks with several hidden layers. Given that network depth has been critical to recent advancements in other fields of deep learning, and that MNIST is an easy dataset to train on, this is a significant issue with the learning framework. It has been demonstrated that the problem can be solved by using a unique learning rate for weight parameters at different depths in the network, with deeper parameters trained to a greater extent, to counteract attenuation of the gradient with network depth and to make its layers train uniformly. This approach is unappealing because (1) it introduces additional hyperparameters that must be tuned, (2) it would be inconvenient to implement and tune unique learning rates in analog hardware, and (3) it seems unlikely that the approach is biologically plausible.

The purpose of this paper is to introduce a modification to the basic multilayer topology that can counteract the vanishing gradient problem in the context of energy-based networks trained using equilibrium propagation. We replace a small portion of a network's connections by random layer-skipping connections in a manner inspired by small-world network theory [20]. We achieve 0% training error and under 2.5% test error on MNIST using a network with 3 hidden layers and no regularization term in its cost function. These error rates are competitive (**this may be too generous) in the context of similar biologically-motivated networks [2] and are the same as those achieved by a similar network in the original paper using a basic topology and unique learning rates, albiet they are achieved after around 25% more epochs. The method requires only one additional hyperparameter - the number of connections to randomly replace - and could

be implemented with relative ease in analog hardware. Layer-skipping connections are also documented in biological brains [4], so it is biologically-plausible. Similar techniques have been used with some success in convolutional ([7], [19]) and multilayer feedforward ([21], [10]) networks with varying degrees of success. Our findings here show that layer-skipping connections are effective enough to be appealing in contexts where simplicity and biological plausibility are important, but would likely not make sense for use in a digital implementation where performance is a primary consideration.

2 Background and Theory

2.1 Equilibrium propagation

Equilibrium propagation [17] is a learning framework for energy-based networks that approximates gradient descent on an 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 the original paper, it is implemented on a continuous Hopfield network [8]. For a variety of reasons outlined in [3], backpropagation is not biologically plausible. One of the major reasons is that to correct the parameters of a given neuron, it would be necessary to have a feedback mechanism providing specific information about the activations and parameters of all neurons between that neuron and the output layer, and no such path has been observed. Equilibrium propagation avoids this problem by approximating the gradient of an arbitrary cost function with respect to a neuron's parameters using only the activations of neurons to which it directly connects. It also requires neurons to perform only a single type of cpmputation in the prediction and correction phases of training. In addition to making equilibrium propagation more biologically-plausible, these characteristics also make it appealing as a candidate to implement in neuromorphic analog hardware by limiting the complexity and scope required of it.

2.1.1 Implementation in a continuous Hopfield network

Here we summarize the dynamics of a continuous Hopfield network trained using equilibrium propagation; a thorough and more-general treatment is given in the original paper [17].

Consider an arbitrary network with two subsets of its neurons designated as an input and output layer. Let $\boldsymbol{x}$, $\boldsymbol{h}$, and $\boldsymbol{y}$ denote, respectively, vectors representing the activations of the input, hidden and output layers of a network. Let $\boldsymbol{s} = \{\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{y}\}$ denote the state of the full network and $\boldsymbol{u} = \{\boldsymbol{h}, \boldsymbol{y}\}$ denote the state of its non-input neurons. Let $\boldsymbol{W}$ and $\boldsymbol{b}$ denote the network's weights and biases and $\boldsymbol{v}$ the set of external influences to the network (inputs $\boldsymbol{x}$ and target outputs $\boldsymbol{y}_{target}$ in a training dataset). Let ρ denote the activation function of a neuron; in this paper it is a hardened sigmoid function $\rho(x) = \text{Max}\{0, \text{Min}\{x, 1\}\}$.

A prediction of $\boldsymbol{y}_{target}$ given $\boldsymbol{x}$ is generated by clamping the network's input layer to $\boldsymbol{x}$, then evolving to a local minimum of an energy function

$$E(\boldsymbol{u}) = \frac{1}{2}\sum_i u_i^2 - \frac{1}{2}\sum_{i \neq j} W_{ij}\rho(u_i)\rho(u_j) - \sum_i b_i \rho(u_i). \tag{1}$$

This evolution is denoted the free phase. Consider a cost function C; in this and in the original paper it is a quadratic error term

$$C(\boldsymbol{y}, \boldsymbol{y}_{target}) = \frac{1}{2}||\boldsymbol{y} - \boldsymbol{y}_{target}||^2, \tag{2}$$

but the algorithm is applicable to an arbitrary cost function - for example, one with a regularization term. After the free phase, the network is then evolved to a local minimum of a new energy function

$$F(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{y}_{target}, \beta) = E(\boldsymbol{u}) + \beta C(\boldsymbol{y}, \boldsymbol{y}_{target}) \tag{3}$$

where β, called the clamping factor, is a small constant with a random sign. This is called the weakly-clamped phase and can be interpreted as weakly-clamping the network's output neurons to the target output [1]. Note

[1] For a cost function dependent on neurons outside of the output layer, this interpretation is complicated.

that $E(\boldsymbol{u}) = F(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{y}_{target}, 0)$. The network evolves to local minima of F by performing gradient descent on it through the equation of motion

$$\frac{d\boldsymbol{s}}{dt} = -\frac{\partial F}{\partial \boldsymbol{s}}. \tag{4}$$

The states of the network after the free and weakly-clamped phases are used to compute correction terms so must be saved; after evolution on some batch we will denote these states, respectively, as $\boldsymbol{s}^0$ and $\boldsymbol{s}^\beta$. The process for training over one batch is as follows:

1. Perform the free-phase evolution; evolve to equilibrium on energy function $F(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{y}_{target}, 0)$ by updating $\boldsymbol{s}$ based on equation 4. Record the state $\boldsymbol{s}^0$.

2. Perform the weakly-clamped evolution; evolve for a short amount of time to equilibrium on $F(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{y}_{target}, \beta)$, again based on equation 4, using $\boldsymbol{s}^0$ as a starting point. Record the state $\boldsymbol{s}^\beta$.

3. Compute the correction to each weight in the network using the function

$$\Delta W_{ij} = \alpha \frac{1}{\beta}(\rho(u_i^\beta)\rho(u_j^\beta) - \rho(u_i^0)\rho(u_j^0)) \tag{5}$$

   where α, the learning rate, is a positive constant. Adjust the weights as $W_{ij} := W_{ij} + \Delta W_{ij}$. (Recall that $\boldsymbol{u}$ is contained in $\boldsymbol{s}$).

4. Compute the correction to each bias in the network using the function

$$\Delta b_i = \alpha \frac{1}{\beta}(\rho(u_i^\beta) - \rho(u_i^0)) \tag{6}$$

   and adjust the biases as $b_i := b_i + \Delta b_i$.

Note that the correction to a weight depends only on the activations of neurons to which it connects, and the correction to a bias depends only on the activation of the neuron it affects. This is in contrast to backpropagation, where information would be required about the parameters and error terms of all neurons in shallower layers.

2.1.2 Approximation of the equation of motion

On analog hardware, the dynamics described by equation 4 could be implemented efficiently using leaky integrator neurons. On digital hardware, however, it is necessary to discretize and approximate the differential equation of motion. Let $\boldsymbol{s}[n]$ denote the state of the network after n iterations of the approximation, N denote the number of iterations and ϵ the size of each iteration. $\boldsymbol{s}[0]$ is the initial state of the network; this may be random or, as in this and the original paper, zero. The state of the network after time $\tau = \epsilon N$ is given by

$$\int_0^\tau \frac{d\boldsymbol{s}}{dt} dt \approx \boldsymbol{s}[N] \tag{7}$$

where

$$s_i[n] = s_i[n-1] - \epsilon \frac{\partial F}{\partial s_i}(\boldsymbol{u}, \boldsymbol{y}, \boldsymbol{y}_{target}, \beta),\ n = 1, \ldots, N. \tag{8}$$

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, so typically $N_{free} >> N_{weakly-clamped}$.

2.1.3 Other relevant methods for biological plausibility

An additional reason backpropagation is not biologically plausible is that it implies a weight matrix with feedback connections identical in weight and destination to its feedforward connections [3]; this is a problem not addressed in the original paper. However, recent work [13] has shown that networks can still train if feedback weights are initialized randomly instead of symmetrically with feedforward connections. These

findings could potentially be applied to an energy-based network trained using equilibrium propagation.

Biological neurons are known to communicate through spikes with binary values, suggesting that a spike timing based activation function for neurons is more biologically-plausible than popular functions. It has been shown [15] that an energy-based network with a spike timing-based activation function can be trained using equilibrium propagation.

The method presented in this paper allows networks with multiple layers to train effectively through a biologically-plausible tweak to their topology. Layer-skipping connections have been documented in biological brains [4].

2.2 Vanishing gradient problem

It has been observed that networks with a layered topology suffer a vanishing gradient problem. This problem is familiar in the context of conventional networks, where it is caused by an expected value of an activation function's derivative that is less than 1, causing an exponential attenuation to the gradient of the cost function with respect to distance from the output layer (or, similarly, an exploding gradient problem if the derivative is greater than 1). The problem in that context has been successfully addressed by initializing a network's weights as described in [6] and by using activation functions with uniform derivatives, such as rectified linear units $\rho(x) = \text{Max}\{0, x\}$ or hardened sigmoids $\rho(x) = \text{Max}\{0, \text{Min}\{x, 1\}\}$.

Neither of these approaches are effective at solving the problem in continuous Hopfield networks trained through equilibrium propagation. While the symptom of the problem is the same as in conventional networks, it is unclear that its cause is the same, as there is not a straightforward causal relationship between the activations of a pair of neurons, and the fact that conventional solutions are ineffective suggests that the cause is different.

One factor that likely contributes to a vanishing gradient in these networks is that it takes multiple iterations of the differential equation for clamping to perturb neurons in deep layers; since the cost function affects only the output layer, the shallowest hidden layer is not effected until the second iteration, the next layer until the third iteration, and so on. The correction to a given neuron is proportional to the extent to which it is perturbed in the weakly-clamped phase. Therefore, if a typical neuron in the output layer is corrected by δ, neglecting other factors a typical neuron l layers from the output would be corrected by

$$\delta \frac{N_{weakly-clamped} - l}{N_{weakly-clamped}}. \tag{9}$$

Since the observed attenuation to corrections is exponential with respect to distance from the output, it seems unlikely that this is a major contributor to the vanishing gradient problem.

The vanishing gradient problem causes deep networks to train very slowly, which is obviously undesirable. It also causes the magnitudes of corrections to weights between pairs of layers to differ by many orders of magnitude; this would be problematic to an analog implementation of the framework where neurons' activations have limited numerical precision.

In the original paper the vanishing gradient problem was solved 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 was 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 solves the problem without these unappealing characteristics.

2.3 Small-world networks

Our topology was inspired by small-world graph topology as described in [20]. Qualitatively, small-world topology is appealing because it is characterized by a small mean shortest path between a pair of nodes (neurons) in a graph (network) with a large amount of clustering, which seems likely to reduce attenuation to the gradient of parameters of deep neurons by providing a low-attenuation path to the output layer, while largely preserving the regimented structure that has contributed to the success of deep neural networks. The topology is also appealing because it has been observed to occur in biological brains [4].

The small-worldness of a network is typically characterized by a characteristic path length L and a

clustering coefficient C. The characteristic path length is the minimal number of edges in a path joining a pair of nodes, averaged over all pairs of nodes in the network. Specifically, for a graph with N nodes where l_{ij} denotes the smallest number of edges needed to connect the two nodes,

$$L := \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} l_{ij}. \tag{10}$$

The neighborhood of a node is the set of nodes with which it shares an edge. The clustering coefficient is the proportion of nodes in the neighborhood of a given node that share an edge, averaged over all nodes in the network. Specifically, for a graph with N nodes, if $\text{Nbhd}(n_i)$ denotes the neighborhood of node n_i, N_i denotes the number of nodes in $\text{Nbhd}(n_i)$, and $c_{jk} = \begin{cases} 1 & n_k \in \text{Nbhd}(n_j) \\ 0 & \text{else} \end{cases}$,

$$C := \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_i} \sum_{\substack{n_j \in \\ \text{Nbhd}(n_i)}} \sum_{\substack{n_k \in \\ \text{Nbhd}(n_i), \\ k \neq j}} c_{jk}. \tag{11}$$

An algorithm was introduced in [20] to convert a regimented graph into a small-world graph [2]. For each edge in the network, with probability p: randomly pick a pair of nodes that do not share an edge. Add an edge between these nodes, and remove the existing edge. p is typically tuned so that the graph will have a large clustering coefficient.

We have found that the performance of a network does not track closely with its clustering coefficient or its characteristic path length. We believe this to be because the characteristic path length depends only on the path to a neuron that attenuates the gradient the least, but in actuality attenuation to the gradient also heavily depends on the number of paths it can take in parallel to a destination. We have had success using networks with around $p = 8\%$ of their edges replaced, which is far more than is necessary to make the network small-world; nonetheless, we believe that the success of this topology is for reasons similar to those listed above.

2.4 Nonlinearities learning residuals

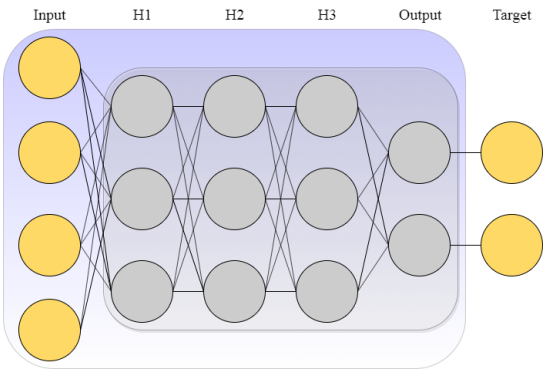
In previous work layer-skipping connections have been used to alter the task of nonlinear layers from matching the desired output to matching the residual between the desired output and the output of a previous layer [7], [9]. This has proven extremely successful for training very-deep networks using backpropagation. If the underlying function describing a training dataset is $\boldsymbol{y} = f(\boldsymbol{x})$, layer-skipping connections can be run in parallel with a collection of nonlinear layers so that they need only match the function $f(\boldsymbol{x}) - \boldsymbol{x}$. The rationale behind this technique is that not all nonlinearities are useful for approximating f, and if a nonlinearity is not useful it must be trained to approximate an identity mapping, which may be difficult. By adding a parallel identity mapping or linear mapping it becomes easy to minimize the contribution by a set of nonlinearities - their weights can simply be driven towards zero.

This is a possible explanation for the effectiveness of our topology. While we do not explicitly structure layer-skipping connections as identity mappings, they could potentially provide a means for the output of a deep layer to at-least partially pass an unhelpful nonlinear transformation.

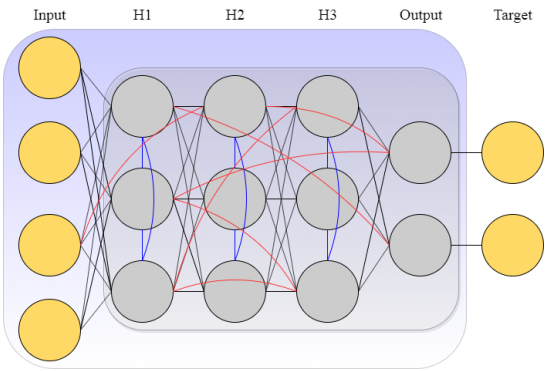
3 Related work

Much research has been done in pursuit of a biologically-plausible deep learning algorithm. [12], [22] and [16] are other algorithms that address the weight transport problem. [13] addresses the need for symmetric feedback weights, a problem not addressed by equilibrium propagation. [15] implements equilibrium propagation

[2] Our algorithm is actually slightly different: instead of replacing each edge with a probability p, we replace a specific number n of the edges currently in the network. Therefore, for a network with N edges our algorithm is equivalent to the one from [20] with $p = 1 - (\frac{N-1}{N})^n$. For ease of comparison with other work and to contextualize how many edges have been replaced, we will henceforth describe networks using our topology in terms of p and not n.



(a) Topology of the basic multilayer network. There is a connection between every pair of neurons in adjacent layers. There are no connections between neurons in the same layer. There are no layer-skipping connections. To combat the vanishing gradient problem, the learning rate is higher for weights deeper in the network.



(b) Changes in our topology with respect to the basic topology. Green connections were added to connect every pair of neurons within a layer. Red connections (8% of existing connections) were chosen at random and removed. Blue connections were chosen at random and added to replace removed connections. These changes allow a uniform learning rate regardless of a weight's depth in the network.

Figure 1

using spiking neurons like are present in a biological brain. [3] and [**?**] discuss the criteria such an algorithm would need to satisfy. [2] surveys promising biologically-motivated algorithms and evaluates their effectiveness on hard algorithms.

[18], [5] and [14] discuss neuromorphic architecture that could potentially implement equilibrium propagation as an analog computer.

Layer-skipping connections have been explored in other contexts. [7] and [19] use layer-skipping connections as a linear transformation in parallel with nonlinear layers to great effect in very-deep convolutional networks. [21] and [10] use a small-world topology in conventional multilayer feedforward networks.

The vanishing gradient problem was a big obstacle to the training of conventional deep networks through backpropagation, and [9], [6] and [**?**] provide effective means for solving it in that context.

4 Implementation

We implemented the equilibrium propagation learning framework [17] using the Pytorch library [**?**], and verified that we could recreate the experiments run in [17]. All of our networks use a hardened sigmoid activation function

$$\rho(x) = \text{Max}\{0, \text{Min}\{x, 1\}\} \tag{12}$$

with

$$\rho'(x) := \begin{cases} 1 & 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases} \tag{13}$$

(the derivative at $x = 0$ and $x = 1$ is chosen to be 1 to avoid saturation). All of our networks use a squared-error cost function with no regularization term. The bulk of our testing was done on the MNIST dataset [11] with the following 3 networks.

4.1 Basic topology with unique learning rates

We recreated the 5-layer network used in [17]. Its topology is illustrated in figure 1a.

The network consists of an input layer with 28^2 neurons, 5 hidden layers with 500 neurons, 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 [**?**]: a weight connecting layer j with n_j neurons to layer $j+1$ with n_{j+1} neurons is drawn from

$$U[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}]. \tag{14}$$

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. 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.

4.2 Basic topology with single learning rate

To provide a reference point for our topology we ran tests on a network that is identical to that in section 4.1 except that it has a single learning rate of .02 across the entire network.

4.3 Our topology

To generate a network with our topology we use as a starting point the topology described in section 4.1. We then connect every pair of neurons in a given layer. Finally, for a given number of connections, we remove an existing connection at random and replace it with a random connection between two unconnected neurons in the network. [3] 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 when replacing around 8% of a network's connections in this manner.

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 (14). Weights of intralayer connections and layer-skipping connections are drawn from $U[-.05, .05]$,where the value .05 was chosen empirically. While we do not theoretically justify this initialization, it allows for good performance.

4.4 Deep network on a linear dataset

We also ran a variety of simpler experiments to assess the scalability of our findings. To do this we generated a dataset consisting of outputs that are a fixed linear combination of a collection of random inputs. To generate a linear dataset with N datapoints, m input dimensions and n output dimensions, we first generate N inputs $\boldsymbol{x_i} \in \mathbb{R}^m$, $i = 1, \ldots n$, with elements independently drawn from $U[0, 1]$. We then generate a matrix $\boldsymbol{T} \in \mathbb{R}^{n \times m}$ with elements independently drawn from $U[0, \frac{1}{7}]$. Finally we create N pairs $(\boldsymbol{x_i}, \boldsymbol{T x_i})$. We have observed that it is nontrivial for a network to represent such a relationship using a hardened sigmoid activation function (12), so this simple dataset is still indicative of the effectiveness of a network topology.

We ran experiments on this dataset with networks consisting of 12 layers of 50 neurons, with variable numbers of replaced connections, and otherwise generated identically to the network described in section 4.3.

4.5 Tracking the training rates of individual pairs of layers

To observe the nature and extent of the vanishing gradient on different networks we periodically measured the root-mean-square correction to the weights between individual pairs of layers. Specifically, 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

$$\Delta w^l(b) := \sqrt{\frac{\sum_{i=1}^{N^l} \sum_{j=1}^{N^{l+1}} (\Delta w_{ij}^l(b))^2}{N^l N^{l+1}}} \tag{15}$$

[3] We have also tried adding new connections without replacing existing ones, and observed roughly the same performance. In this paper we replace connections to limit the number of free parameters added to the network and avoid the possibility of conflating improvement due to the changed topology with that due to additional parameters.

Network	Learning rate(s)	ϵ	β	N_{free}	$N_{weakly-clamped}$
Basic, unique learning rates	.128, .032, .008, .002	.5	1.0	500	8
Basic, one learning rate	.02	.5	1.0	500	8
Our topology	.02	.5	1.0	500	8

Table 1: Hyperparameters of networks tested on MNIST dataset

as our metric of the extent to which the weights between layers l and $l+1$ trained in response to batch b. Note that this measurement ignores the training of the weights of intralayer and layer-skipping connections.

Since this measurement tends to be volatile, for clarity we plot the average of $\Delta w^l(b')$ for b' in a neighborhood of batch b. Specifically, for some n we define

$$\Delta \hat{w}^l(b) := \frac{1}{2n+1} \sum_{b'=b-n}^{b+n} \Delta w^l(b'). \tag{16}$$

Then by plotting the traces $\Delta \hat{w}^l(b)$ for each layer l with respect to b we can compare the extent to which each layer trained over a given span of batches.

4.6 Tracking the spread of training rates as a scalar

We observe that in networks with our topology, weights connecting to the output layer tend to train with a rate around an order of magnitude higher than deeper weights, independently of the number of replaced connections. We also note that with few replaced connections it is common for training rates to vary by several orders of magnitude. These observations informed our definition of a scalar metric of the spread of the training rates of a network's layers. For a network trained for N batches with $\Delta w^l(b)$ measured every n batches and where $\Delta w^l(b)$ is defined by (15), we first define

$$\Delta w^l := \sum_{b=1}^{\lfloor N/n \rfloor} \Delta w^l(nb) \tag{17}$$

as a metric of how much the weights between layers l and $l+1$ were corrected over the course of training. For a network with $2L+2$ layers where $l_{min}(i)$ denotes the pair of layers with the i^{th} smallest Δw^l, we finally define

$$\text{Spread} := \frac{\sum_{i=L+1}^{2L} \Delta w^{l_{min}(i)}}{\sum_{i=1}^{L} \Delta w^{l_{min}(i)}}. \tag{18}$$

Note that this metric ignores the difference between deep pairs and the pair connecting to the output layer, as it is assumed to be independent of the number of replaced connections.

5 Results

5.1 MNIST dataset

We compared the three network topologies described in sections 4.1, 4.2, and 4.3 on the MNIST dataset [11]. The hyperparameters for the networks are shown in table 1. All networks were trained for 250 epochs with 50,000 training examples, 10,000 test examples and a batch size of 20.

5.1.1 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 in around 25% more epochs. These results are shown

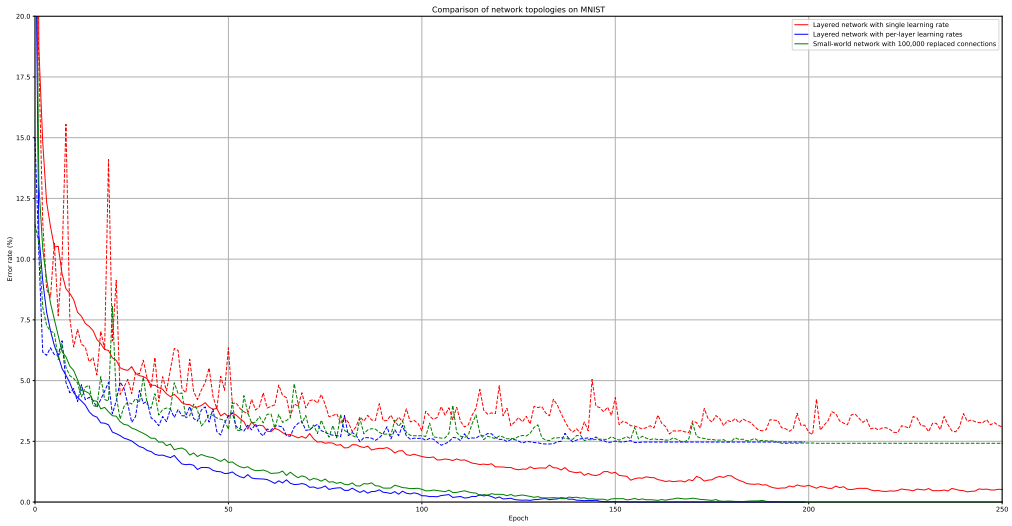


Figure 2: Comparison of network topologies on MNIST dataset. Dotted lines show test error and solid lines show training error. In red is a network with a standard multilayer feedforward topology and a single learning rate. In blue is a network with the same multilayer feedforward topology and unique learning rates for each layer, tuned to promote uniformity in the extent to which each pair of layers trains, as described in [17]. In green is a network with a multilayer feedforward topology with fully connected layers and with around 8% of its connections replaced by random layer-skipping connections.

in figure 2.

Notice that the basic 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 basic 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 in the context of the MNIST dataset, our network with a single learning rate is practically interchangeable with the basic network with unique learning rates.

5.1.2 Training rates of individual pairs of layers

We tracked the training rates of individual pairs of layers as described in (15) and (16) and found evidence that the extent of the vanishing gradient problem is the primary cause of the performance disparity seen in section 5.1.1. These results are shown in figure 3.

Notice in the basic network with one learning rate that 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 basic 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.

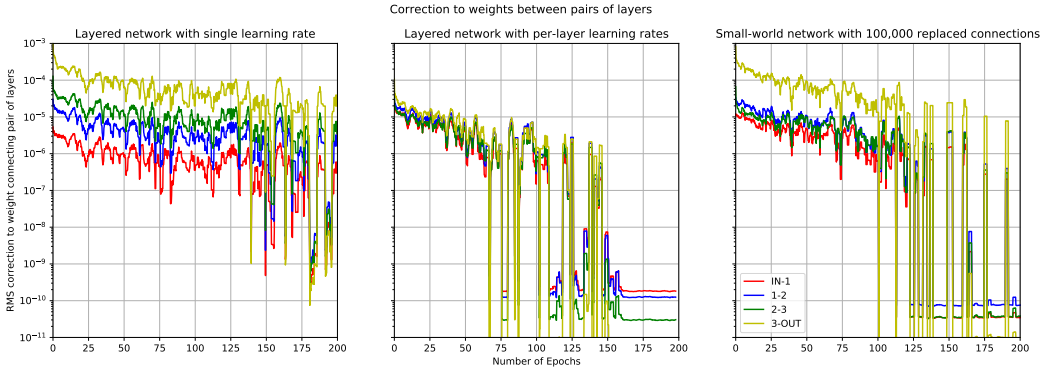


Figure 3: Observation of extent of training of individual layers for different network topologies. Measurements were taken while running trials shown in figure 2, and have been averaged as described in (**??**). To the left is a network with a standard multilayer feedforward topology and a single learning rate. In the center is a network with the same multilayer feedforward topology and unique learning rates for each layer, as described in [17]. To the right is a network with a multilayer feedforward topology with fully-connected layers and with around 8% of its connections replaced by random layer-skipping connections.

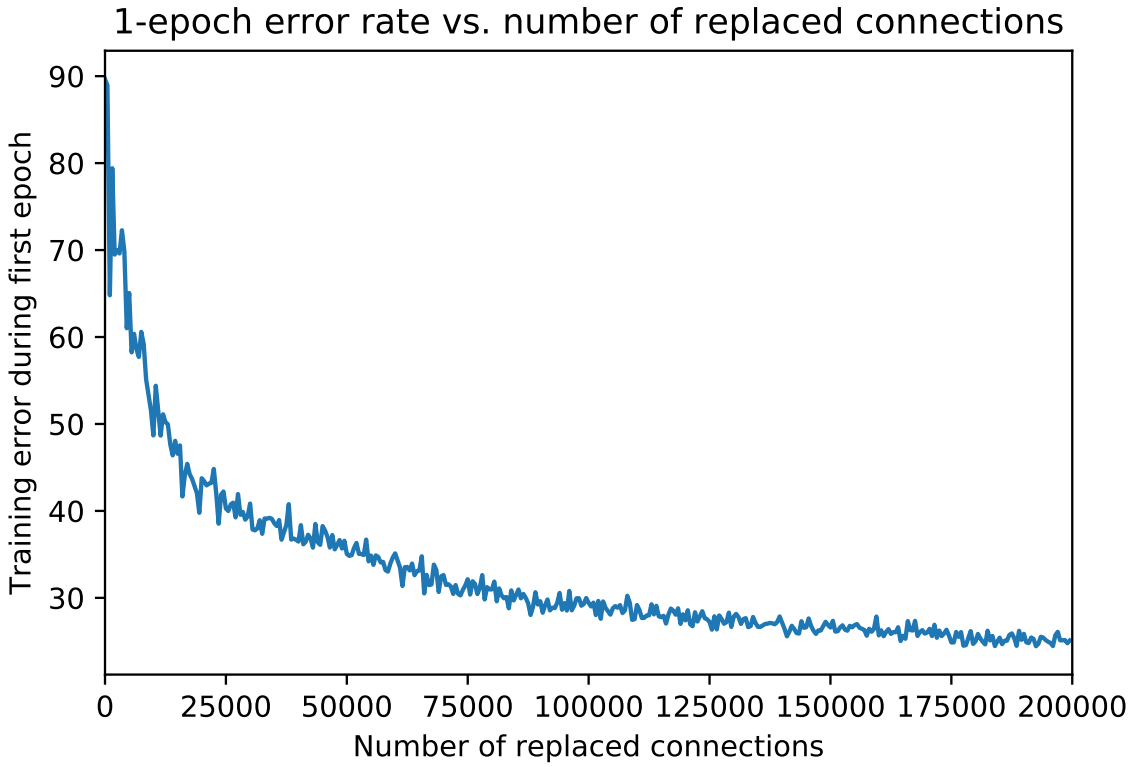


Figure 4: The training error after one epoch of a network with our topology, as connections are replaced.

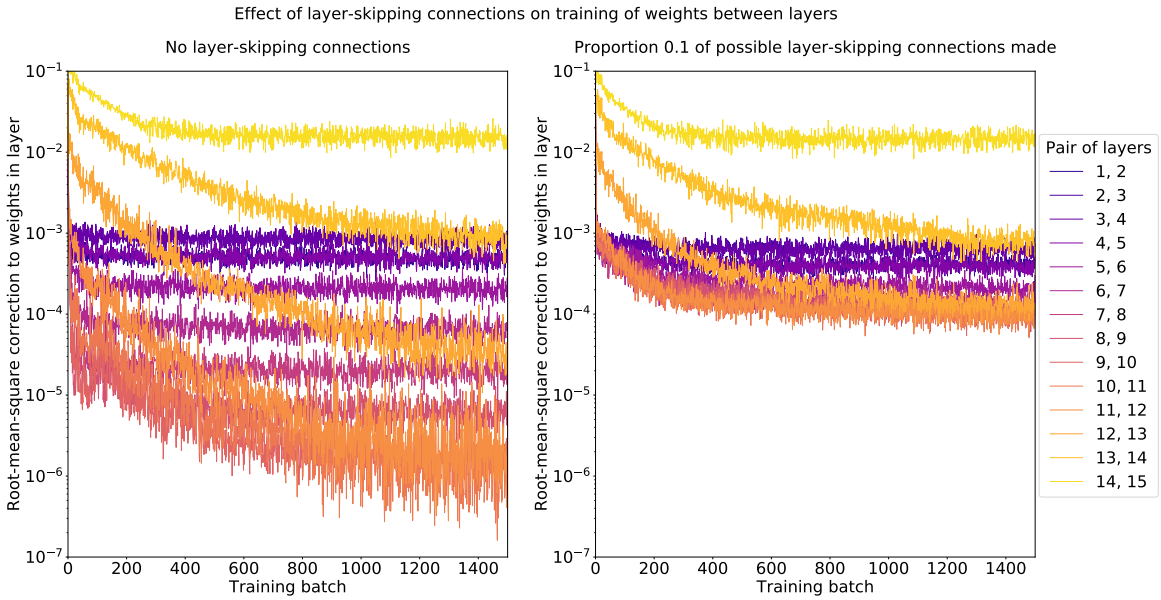


Figure 5: Training rates of pairs of layers in a 15-layer network trained on a linear dataset. To the left is a network with no replaced connections. To the right is a network with 10% of its connections replaced. Trace color changes from yellow for shallow layer pairs to purple for deep layer pairs.

5.1.3 Error rate after one epoch as connections are added

We tracked the training error after one epoch of a network with our topology, with varying numbers of layer-skipping connections and found that the error rate decays approximately exponentially. These results are shown in figure 4.

The error rate drops quickly as connections are replaced early on. We believe this is because when a forward connection is added, in addition to providing a path via. one connection between the two connected neurons, it also provides paths via. at most 3 jumps to all pairs of neurons in the two layers, as a result of the intralayer connectivity. This benefit is exhausted when all pairs of layers are connected, leading to the more-gradual improvement later on. It appears to take around 25,000 replaced connections to reach this regime, likely because the extent of attenuation of a gradient depends less on the minimum path length between two neurons and more on the number of low-attenuation paths it has available.

We found that our topology performs significantly worse than the basic topology with one learning rate when few connections are replaced, possibly due to inappropriate hyperparameter choices for intraconnected layers. We have seen some evidence that replacement of connections makes a network more-forgiving of poor weight matrix initialization, but have not thoroughly probed the issue.

5.2 Linear dataset

We ran trials using a 15-layer network with 50 neurons per layer and with a varying number of replaced connections, on linear datasets as described in section 4.4. We used a learning rate of .01, $\epsilon = .5$, $\beta = 2.0$, $N_{free} = 20$, $N_{weakly-clamped} = 4$, 10,000 training examples and no test examples, and a batch size of 20.

5.2.1 Training rates of individual pairs of layers

We observed the training rates of individual pairs of layers as described in (15) and (16) and found that the qualities observed in section 5.1.2 are still present in this much-deeper network. These results are shown in figure 5.

Observe that the training rate of the output layer is around an order of magnitude higher than that of

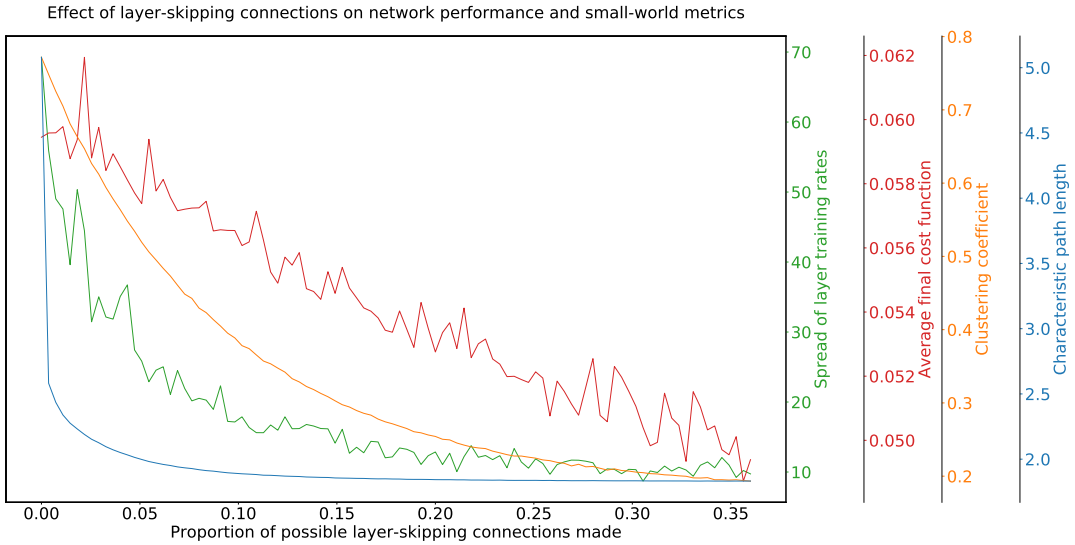


Figure 6: Spread, cost function after one epoch, and small-world metrics of a 15-layer network trained on a linear dataset with a varying proportion of its connections replaced. Green: spread of training rates of layer pairs. Red: cost function after training for 1 epoch. Blue: characteristic path length. Red: clustering coefficient.

deeper layers both in a network with no replaced connections and in a network with 10% of its connections replaced. Also observe that the variation in training rates of pairs of layers is around 4 orders of magnitude higher in the network with no connections replaced than it is in the network with 10% of its connections replaced. Both of these observations are consistent with results seen in section 5.1.2.

An interesting feature of the graphs for which we do not have an explanation is that the training rate of a layer pair does not strictly decrease with its depth; instead it is minimal approximately at the center of the network and increases with proximity to the input or output layer.

5.2.2 Spread, cost after one epoch, and small-world metrics

We observed that the spread (18) of training rates decreases exponentially with the number of replaced connections and that the cost function after one epoch decreases linearly. There is not a strong apparent correlation with the small-world metrics described in section 2.3. These results are shown in figure 6.

We believe the relationship between the spread and the number of replaced connections is due to the same reasons as the relationship between the error rate after 1 epoch and the number of replaced connections discussed in section 5.1.3. We believe the reason for the lack of correlation with the characteristic path length is that it is based on the shortest path between a given pair of neurons, but the attenuation to the gradient as it travels between the pair also depends strongly on the number of low-attenuation paths that it can take in parallel. We expect that a lower clustering coefficient will correlate with overtraining as a network behaves with increasing similarity to a single-layer network, but that is beyond the scope of this paper. We did not expect it to correlate with the spread or cost function of a network after one epoch, and it does not appear to.

6 Conclusion

We believe that our topology is a suitable substitute for unique learning rates as a solution to the vanishing gradient problem. While it is not as effective, it is simpler, more biologically-plausible, and would be easier to implement in an analog computer.

There are several directions in which future research could be taken. It would be useful to find and mathematically-justify an effective weight initialization scheme for a network with our topology. It would be interesting to explore the effect of adding (instead of replacing) layer-skipping connections, training a network, then removing the connections and training it further, to see if they provide a residual benefit even after removal; doing so could allow a network in an analog computer to be trained quickly, and the added connections removed to reduce the power consumption and heat load of the network.

References

[1] my github repository for eqp code.

[2] 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.

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

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

[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] 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. Towards a biologically plausible backprop. *CoRR*, abs/1602.05179, 2016.

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

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

[21] 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.

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