Backpropagation-Free Deep Learning with Recursive Local Representation Alignment

Alexander G. Ororbia[1 *], **Ankur Mali**[2 *], **Daniel Kifer**[3], **C. Lee Giles** [3]

[1] Rochester Institute of Technology, Rochester, NY 14623, USA
[2] University of South Florida, Tampa, FL 33620, USA
[3] The Pennsylvania State University, State College, PA 16801, USA
ago@cs.rit.edu, ankurarjunmali@usf.edu, duk17@psu.edu, clg20@psu.edu

Abstract

Training deep neural networks on large-scale datasets requires significant hardware resources whose costs (even on cloud platforms) put them out of reach of smaller organizations, groups, and individuals. Backpropagation (backprop), the workhorse for training these networks, is an inherently sequential process that is difficult to parallelize. Furthermore, researchers must continually develop various specialized techniques, such as particular weight initializations and enhanced activation functions, to ensure stable parameter optimization. Our goal is to seek an effective, neuro-biologically plausible alternative to backprop that can be used to train deep networks. In this paper, we propose a backprop-free procedure, *recursive local representation alignment*, for training large-scale architectures. Experiments with residual networks on CIFAR-10 and the large benchmark, ImageNet, show that our algorithm generalizes as well as backprop while converging sooner due to weight updates that are parallelizable and computationally less demanding. This is empirical evidence that a backprop-free algorithm can scale up to larger datasets.

Introduction

At the heart of training artificial neural networks (ANNs) is the calculation of adjustments that need to be made to parameters given some data. This calculation is used in tandem with an optimization procedure, such as a stochastic hill-climbing procedure, to then alter the ANN's actual parameters in order to ensure it makes better future predictions. This adjustment process entails using an algorithm that conducts credit assignment, i.e., the task of determining the contribution that individual neuronal units (within an ANN) make to the system's overall error. To conduct credit assignment and compute weight updates in state-of-the-art networks today, backprop (Rumelhart, Hinton, and Williams 1986) is the popular algorithm of choice but has been long criticized as neuro-biologically implausible (Crick 1989). While backprop provides a theoretical basis for training networks, i.e. gradient descent, it also presents practical challenges, e.g., exploding/vanishing gradients (Glorot and Bengio 2010).

In order to deal with the problems posed by backprop, researchers must resort to techniques and heuristics to stabilize learning, e.g., careful initialization of weights, often following from a network-specific analysis of backprop's learning dynamics (Glorot and Bengio 2010; He et al. 2015; Sussillo 2014; Mishkin and Matas 2015) or modifying network structure, for example by using ReLU or mish instead of sigmoid activations. Challenges such as these hampers the understanding of new or naive users that want to exploit the benefits of deep learning in novel applications. Furthermore, backprop is sequential; layers are updated in a predefined order, reducing opportunities for model parallelization. This limits models from exploiting the true processing power offered by multi-CPU/GPU setups.

This paper seeks to address the above-mentioned shortcomings and demonstrate that a biologically-motivated algorithm can scale up to large-scale architectures and large datasets. Specifically, we present a procedure better suited to parallelization, adjusting synaptic weights with local rules (in particular, layers can be updated out of order). The contributions of this work are as follows: (1) the algorithm, *recursive local representation alignment* (rec-LRA), is proposed for training large-scale ANNs. Results show that it handles non-differentiable activations, converges faster than backprop, and offers faster training for large-scale benchmarks (ImageNet), and (2), strong generalization across several datasets, including ImageNet, is demonstrated for models trained using rec-LRA. Furthermore, we extend rec-LRA to work with perturbative neural networks (PINNs), offering a fast alternative to convolution with fewer parameters.

Related Work

Connectionist researchers have long dreamed of designing biologically plausible learning algorithms that offer stable performance with improved generalization (Hinton 2002; Chalasani and Principe 2013; Scellier and Bengio 2017; Alias Parth Goyal et al. 2017; Sacramento et al. 2018; Nøkland and Eidnes 2019; Krotov and Hopfield 2019). One key motivation behind the development of alternative algorithms is the removal of the required weight symmetry between forward and backward synapses, which is an essential mechanism for backprop. This has also been referred to as the weight-transport problem (Grossberg 1987; Liao, Leibo, and Poggio 2016), a strong neuro-biological criticism of backprop and one source of its practical issues. Algorithms such as random feedback alignment (FA) (Lillicrap et al. 2016) and direct feedback

*These authors contributed equally.

alignment (DFA) (Nøkland 2016) have shown that learning without this requirement is possible. Surprisingly, even if the feedback pathway is partially decoupled and random, fixed weights are used to transmit derivative signals backward. FA replaces the transpose of the feedforward weights in backprop with a similarly-shaped random matrix while DFA directly wires the output layer's pre-activation derivative to each layer's post-activation – both algorithms use these random matrices to generate proxies for the partial derivatives normally given by backprop. Under a proposed framework known as discrepancy reduction (Ororbia II et al. 2017), it was shown that these feedback loops are better suited for generating target representations, entirely removing backprop's global feedback pathway – a key idea that our algorithm builds on. Algorithms such as target propagation (Lee et al. 2015; Bartunov et al. 2018; Ahmad, van Gerven, and Ambrogioni 2020), which are also subsumed by the discrepancy reduction framework, generate targets through auto-encoding inversion.

The idea of local learning, which has origins in the classical frameworks of Hebbian (Hebb 1949), anti-Hebbian (Földiak 1990), and competitive learning (Rumelhart and Zipser 1985), has slowly begun to gain increased attention in the training of ANNs. Recent proposals have included decoupled neural interfaces (Jaderberg et al. 2016), greedy relaxations of backprop (Belilovsky, Eickenberg, and Oyallon 2019), and others (Balduzzi, Vanchinathan, and Buhmann 2015; Taylor et al. 2016). Furthermore, (Xie and Seung 2003) demonstrated that neural models using local updates could efficiently conduct supervised learning. Earlier approaches that employed local learning included the layer-wise training procedures that were once used to pre-train networks (Vincent et al. 2008; Bengio et al. 2007; Lee et al. 2014; Ororbia II et al. 2015). The problem with these older approaches is that they were greedy–a model was built from the bottom-up, freezing lower-level parameters as higher-level feature detectors were learned. However, modern generalizations have been proposed (Belilovsky, Eickenberg, and Oyallon 2018) to solve these issues partially.

Recursive Local Representation Alignment

The Problem & Notation

While our algorithm could be applied to any neural architecture (including recurrent ones), in this paper, we focus on ones that learn a nonlinear mapping f_Θ from inputs $\mathbf{x}$ to outputs $\mathbf{y}$. As usual, each input example can be modeled as a matrix $\mathbf{x} \in \mathcal{R}^{I \times C}$ (e.g., for images with I pixels and C channels) or vector $\mathbf{x} \in \mathcal{R}^I$ (e.g., for grey-scale images with I pixels or text document vectors with I distinct tokens),[1] or even as tensors. On the other hand, the target $\mathbf{y} \in \mathcal{R}^Y$ can be modeled as a one-hot encoding, where Y is the number of distinct categories in a dataset.

The nonlinear mapping $f_\Theta(\mathbf{x})$ contains a set of learnable parameters housed in the construct Θ, which are what algorithms such as backprop are trying to modify to improve predictive performance. In feedforward networks, a stack of nonlinear transformations, or $\{f_\ell(\mathbf{z}_{\ell-1}; \theta_\ell)\}_{\ell=1}^L$, is applied to the input $\mathbf{x}$. As an example, if the network is a multilayer perceptron (MLP), each transformation $\mathbf{z}_\ell = f_\ell(\mathbf{z}_{\ell-1})$ produces an output $\mathbf{z}_\ell$ from the value $\mathbf{z}_{\ell-1}$ of the previous layer with the help of a weight matrix $\theta_\ell = \{W_{(\ell-1)\to\ell}\}$. f_ℓ is decomposed into two operations (biases omitted for clarity):

$$\mathbf{z}_\ell = \phi_\ell(\mathbf{h}_\ell), \quad \mathbf{h}_\ell = W_{(\ell-1)\to\ell} \cdot \mathbf{z}_{\ell-1} \tag{1}$$

where ϕ_ℓ is an activation function, $\mathbf{z}_\ell \in \mathcal{R}^H$ is the post-activation of layer ℓ, and $\mathbf{h}_\ell \in \mathcal{R}^H$ is the pre-activation vector of layer ℓ. Note that matrix multiplication is denoted by $(\circ \cdot \circ)$, a Hadamard multiplication is denoted by $(\circ \otimes \circ)$, and $(\circ)^T$ denotes the transpose operator ($\circ$ is an argument). For convenience, we set $\mathbf{z}_0 = \mathbf{x}$ (referring to the input vector) and $\mathbf{z}_L$ is the final output/prediction made by the stacked model $f_\Theta(\mathbf{x})$. We have also introduced special notation for our synaptic matrices, where $W_{i\to j}$ indicates that this parameter matrix connects neurons in layer i to j.

For classification, the output activation is the softmax, i.e., $\mathbf{y} = \phi_L(\mathbf{v}) = \exp(\mathbf{v})/(\sum_j \exp(\mathbf{v}[j]))$, where j indexes scalar elements of a vector. Any element in the output vector, i.e., $\mathbf{y}[j] \equiv \phi_L(\mathbf{v})[j] = p(j|\mathbf{v})$, is the scalar probability of class j. Generally, the goal of training is to adjust Θ to minimize the output loss known as the negative Categorical log-likelihood, or $L(\mathbf{y}, \mathbf{v}) = -\sum_i \left(\mathbf{y} \otimes \log p(\mathbf{y}|\mathbf{v})\right)[i]$.

The Learning Algorithm

The central idea behind our algorithm, recursive local representation alignment (rec-LRA), is that every layer, not just the output layer, has a target and each layer's parameters are adjusted so that its output moves closer to its target. While this idea is also an aspect of prior work such as target-prop (Carreira-Perpiñán and Wang 2012; Bengio 2014; Lee et al. 2015), one key difference between rec-LRA and these prior efforts is that rec-LRA chooses targets that are in the "possible representation" of the associated layers. Hence, a layer's parameters are updated more effectively, i.e., a layer is *not* forced to match a target that is impossible to achieve.[2] Furthermore, rec-LRA stands in contrast to recently-explored feedback alignment algorithms (Lillicrap et al. 2016; Akrout et al. 2019), i.e., instead of trying to approximate backprop's way of creating teaching signals, rec-LRA introduces a special processing unit that creates perturbation signals locally. Thus, rec-LRA can be viewed as an alternative to such approaches and potentially as a complementary technique for most neural design choices, such as residual blocks and other layers that might be helpful for problem-specific representations that a deep network would need to acquire. Our algorithm, which builds on ideas in (Ororbia and Mali 2019) (yet is far more general, e.g., rec-LRA resolves the update-locking problem (Jaderberg et al. 2016) and is architecture-agnostic, even providing an update scheme for convolution; see Appendix[3] for a discussion of key differences), aims to break the credit assignment problem into smaller, easier sub-problems that are solvable in parallel with each other. It follows that this

[1] Vectors and matrices are assumed to be in column orientation.

[2] For example, target-prop uses noise injection (in an encoding-decoding cycle), which could generate targets that "jump" far from a layer's current activity, yielding a jolting, high-magnitude perturbation resulting in unstable optimization (Ororbia and Mali 2019).

[3] https://www.cs.rit.edu/~ago/reclra_aaai2023_appendix.pdf

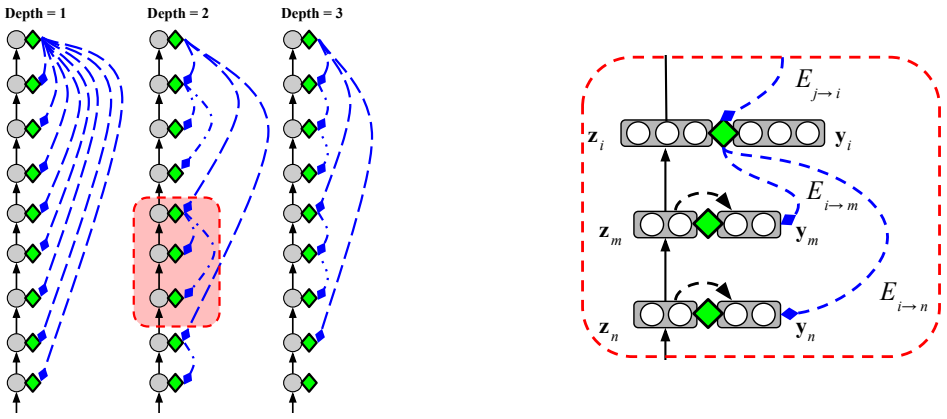

Figure 1: Rec-LRA error transmission circuitry. (a) Possible error transmission pathways (ordered left to right by recursive depth). (b) Zoom-in of a target creation circuit for the area in the red box of the model to the left.

divide-and-conquer behavior facilitates distributed training if high-performance computing resources are available.

To specify rec-LRA, we start by defining the function it is meant to optimize, i.e., *total discrepancy* (Ororbia II et al. 2017), which is a "pseudo-energy function" that measures the amount of overall system disorder, and can be seen as approximately optimizing variational free energy (Friston 2009) (allowing for connections to made between our algorithm's objective and those of normative Bayesian brain models). Specifically, this function computes the degree of mismatch between the current activity of a neural model's layers and the activity of a set target activities (that rec-LRA determines). These targets represent values that the network's neuronal processing elements should have taken to better predict aspects of its environment. Under the framework of discrepancy reduction, a neural system is minimizing the weighted sum of local representational mismatch functions:

$$\mathcal{D}(\Theta) = \sum_{\ell=1}^{L} \kappa_\ell \left(||\mathbf{z}_\ell - \mathbf{y}_\ell||_p \right)^q \tag{2}$$

where $\{\mathbf{y}_1, \cdots, \mathbf{y}_\ell, \cdots, \mathbf{y}_L\}$ are the layer-wise targets and $\mathbf{y}_L$ is the output (i.e. it is $\mathbf{y}$). The value p sets the type of distance function or norm used to compute the mismatch between a state's prediction and the actual target, i.e., $p = 2$ is the L2 (Euclidean) norm, and $p = 1$ is the L1 (Manhattan) norm (typically $q = p$). For this study, we set $p = q = 2$, choosing the Euclidean distance function as our representational mismatch function (though we remark that $q = 1$ could be useful to encourage neural activities to weakly embody/-model homeostatic constraints in the brain, as its activity patterns are typically quite sparse). The scalar κ_ℓ is a local coefficient that, while typically set to one for all layers, i.e., $\kappa_1 = \cdots = \kappa_\ell = \cdots = \kappa_L = 1$, if set to values less than one, one could simulate different time-scales of parameter evolution within various levels of the model.

By taking derivatives of Equation 2 with respect to each layer of neurons, one can derive vectors of special neurons called "error neurons", or $\mathbf{e}_\ell$ (see Appendix for derivation). These neurons measure the difference between the post-activity values of one set of neurons $\mathbf{z}_\ell$ with a corresponding set of target activity values $\mathbf{y}_\ell$, which are motivated by neuro-mechanistic predictive coding theory (Rao and Ballard 1999; Clark 2013) (such as those that were identified as signaling the detection of errors in the human medial frontal cortex (Fu et al. 2019)). These error units form the backbone of a two-phase learning process, using only forward operations: 1) a target generation phase aided by the use of synaptic weights that transmit mismatch signals across the system, and 2) a local weight update that does not require knowledge of the point-wise derivatives of the layer-wise activities.

One particularly powerful yet unexplored aspect of the discrepancy framework is that the target generation process is not constrained to be symmetrical to the feed-forward phase that computes $f_\Theta(\mathbf{x})$. This means that, when conducting credit assignment, error information is not constrained to trace backwards the same pathway taken by the signals that moved forward through the network during inference. This contrasts with backprop, which requires derivative information to move back along a global feedback pathway that starts from the output layer and back along the same weights used to carry information forward, i.e., a specific error circuitry that follows from applying the chain rule of calculus to the output cost function. This feedback pathway is not only neuro-biologically implausible, but it is the central cause of the well-known vanishing/-exploding gradient problem (Glorot and Bengio 2010) since a single error signal traversing back along the central information propagation pathway of $f_\Theta(\mathbf{x})$ is constantly multiplied by the local derivatives of each layer that it passes through. In our learning framework, error signals are instead transmitted to the regions of the subgraphs that require them through the use of what we call *skip-error connections*, or special, learnable synapses that carry error messages to neural activities that require "correction" (greatly reducing the credit assignment transmission length). Skip-error synapses facilitate the transmission of mismatch signals computed by neurons at layer i directly to any layer j, serving as a short-circuit pathway to generate corrective perturbations to the activities directly (this, as a result, resolves backprop's update-locking problem (Jaderberg et al. 2016)). One could also interpret these circuit

pathways as "error highways", similar to the forward skip synapses used to improve stability in backprop-based ANNs (Srivastava, Greff, and Schmidhuber 2015).

Under rec-LRA (some error pathways are depicted in Figure 1), targets can be likened to latent representations that would be more desirable when predicting $\mathbf{y}$ from $\mathbf{x}$. For a layer i of neurons, a target is computed by taking the mismatch computed by error neurons $\mathbf{e}_j$ at layer j in the network and transmitted across a set of error synapses $E_{j\rightarrow i}$, yielding a (vector) displacement signal (perturbation) that communicates to layer i's error units $\mathbf{e}_i$ just how much the layer's activity needs to be adjusted to better please the mapping from $\mathbf{x}$ to $\mathbf{y}$. Formally, layer i's target ($\mathbf{y}_i$) is computed as:

$$\mathbf{y}_i = \phi_i(\mathbf{h}_i - \beta \mathbf{d}_i) \qquad \text{// Note: } \mathbf{d}_i = E_{j\rightarrow i} \cdot \mathbf{e}_j \qquad (3)$$

$$\mathbf{e}_j = \mathbf{z}_j - \mathbf{y}_j \qquad \text{// Assuming } p = q = 2 \qquad (4)$$

noting that all that is required for computing a target at i is its original pre-activation vector and the reuse of its post-synaptic activation function $\phi_i(\circ)$. β is the modulation factor that controls the influence of the transmitted displacement message from node j to i. Again, notice that we explicitly indicate the direction of transmission from region j to i with the subscript notation $j \rightarrow i$ for error synapses $E_{j\rightarrow i}$. In an MLP, $\mathbf{h}^i$ would be the pre-activation of a layer i (Equation 1) with the post-activity of that layer $\mathbf{z}^i$ computed by applying a non-linearity, such as the linear rectifier, $\phi^i(v) = max(0, v)$, or a non-differentiable one such as signum, $\phi^i(v) = sign(v)$. However, $\mathbf{h}^i$ could be the output of a complex function, i.e., a stack of convolution and max-pooling operators, as in the case of a residual network. The error neurons $\mathbf{e}_i$ at layer i then compare the target $\mathbf{y}_i$ to the original activity $\mathbf{z}_i$.

Once a target for any layer has been computed, such as the one for layer i described above, the update for a synapse follows a Hebbian-like form (the rule can also be derived from the objective, as shown in the Appendix). For example, if layer i was connected to an earlier layer/processing stage k by a dense weight matrix $W_{k\rightarrow i}$, then the update using the target representation computed above would be:

$$\Delta W_{k\rightarrow i} = (\mathbf{z}_i - \mathbf{y}_i) \cdot (\mathbf{z}_k)^T = \mathbf{e}_i \cdot (\mathbf{z}_k)^T. \qquad (5)$$

If the layers i and k were related by something other than a dense matrix, such as a set of filters or noise maps, the update rule could be readily adjusted to deal with the operation under question (for example, the rule would follow the form in the next sub-section). The error synapses $E_{j\rightarrow i}$ that relay information from layer j to i are updated using a local rule:

$$\Delta E_{j\rightarrow i} = \gamma(-\mathbf{d}_i \cdot (\mathbf{e}_j)^T) \qquad (6)$$

where γ is a factor for controlling the strength of the error synaptic adjustment (a value less than 1 is used to change the error synapses more slowly than the forward ones).

So armed with the perspective above, rec-LRA, as a general procedure, would first run the forward pass procedure of $f_\Theta(\mathbf{x})$ and then compute the targets and mismatch signals (Equations 3 & 4) which can immediately be used for weight update calculations (Equations 5 & 6). The inherent parallelism in the target and mismatch computations stems from the fact that the error pathway need not be symmetrical to the forward transmission one. If a point j had error synapses connecting to points m and n, the error transmission to each point could happen in parallel since the displacement calculation at n does not depend on that at m. This means that transmission of mismatch signals to each of j's neighbors can be done on separate processors, if available. In Figure 1, this type of error transmission circuitry is graphically depicted (Figure 1b).

For a feedforward architecture, rec-LRA would start operating at the output layer L, then compute the targets for the inner regions that the error neurons at L connect to, and then recursively call itself on each of those target regions, subsequently computing the appropriate error neuron vectors and further computing targets for regions that connect to those regions, and so on and so forth. The base case for the recursion's termination would be when it encounters regions that do not immediately connect to anywhere else. This is formally depicted in rec-LRA's architecture-agnostic algorithmic form, the full details of which are provided in the Appendix. It is important to note that the weight matrices (both $W_{k\rightarrow i}$ & $E_{j\rightarrow i}$) that connect to a region i can be readily updated as soon as the local error neuron signals are available. To exploit the potential speed offered by rec-LRA's parallel nature, one could allocate each recursive call to a cluster/set of CPUs/GPUs dedicated to generating targets/updates for various parts of the operator graph.

While rec-LRA is architecture-agnostic, one may note that the design/wiring of its error pathways is flexible.To ease the design process, a modeler could automate this design choice by employing an outer search method, e.g., neural architecture search (Elsken, Metzen, and Hutter 2018), or could craft pathways that take into account the dimensionality of the network's layers and the number of processors available (see Appendix for details on/analysis of wiring patterns). For example, one might choose the middle wiring pattern (the depth 2 models) in Figure 1 if layers 2, 5, and 8 are bottleneck layers (they contain low numbers of neurons) and 5 GPUs are available – rec-LRA would first use 3 GPUs to parallel compute targets/mismatches for layers 8, 5, and 2 and then 5 GPUs to parallel compute the targets/mismatches for the remaining layers $(1, 3, 4, 6, 7)$. Alternatively, if one is using an architecture with repeating design "blocks", e.g., a transformer for language processing (Devlin et al. 2018) or a residual network (He et al. 2016a) for image processing (studied in the next section), one could use the model's natural grouping of processing layers to create an error transmission pathway.

Residual Neural Networks and rec-LRA

Residual neural networks (ResNets) (He et al. 2016a,b), recently reaching state-of-the-art performance on popular vision benchmarks, are architectures that are composed of many hidden layers wired together with a special forward connectivity pattern. Specifically, residual networks utilize shortcut connections that allow the propagation of information to jump over some hidden layers, specifically, those that might not prove useful in mapping $\mathbf{x}$ to $\mathbf{y}$. Formally, the layers in the network that permit a residual mapping are defined as: $\mathbf{z}_\ell = f_\ell(\mathbf{z}_{\ell-1}; \theta_\ell) + \mathbf{z}_{\ell-g}$ where g controls the length of the gap/skip, typically of size 2 or 3. The idea behind this formulation is that, in the event that fitting the transformation

Algorithm 1: Rec-LRA (depth 2) for $f_\Theta(\mathbf{x})$ w/ residual gap g.

Inputs: $\mathbf{x}$, $\mathbf{y}$, $\Theta = \{W_1, \cdots, W_L\}$, β, γ, g
  $\Theta_E = \{E_{L\rightarrow(L-1)}, \cdots, E_{i\rightarrow j}, \cdots, E_{L\rightarrow 1}\}$
// Inference procedure for network $f_\Theta(\mathbf{x})$
function RunModel($\mathbf{x}$, Θ)
    for $\ell = 1$ to L **do**
        if $\ell \mod g \equiv 0$ **then**
            $\mathbf{h}_\ell = W_\ell \cdot \mathbf{h}_{\ell-1} + \mathbf{z}_{\ell-g}$
        else
            $\mathbf{h}_\ell = W_\ell \cdot \mathbf{h}_{\ell-1}$
        $\mathbf{z}_\ell = \phi_\ell(\mathbf{h}_\ell)$
    return $\mathcal{Z} = \{\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_L\}$
// Compute error neurons given activities
function CalcErrUnits($\mathbf{y}$, Θ, Θ_E, $\mathcal{Z}$)
    $\mathbf{y}_L = \mathbf{y}, \quad \mathbf{e}_L = \mathbf{z}_L - \mathbf{y}_L$
    for $\ell = (L-1)$ to 1 **do**
        if $\ell \mod g \equiv 0$ **then**
            $\mathbf{d}_\ell = E_{L\rightarrow\ell} \cdot \mathbf{e}_L$
        else
            $\mathbf{d}_\ell = E_{\ell+1\rightarrow\ell} \cdot \mathbf{e}_{\ell+1}$
        $\mathbf{y}_\ell = \phi_\ell(\mathbf{h}_\ell - \beta\mathbf{d}_\ell), \quad \mathbf{e}_\ell = \mathbf{z}_\ell - \mathbf{y}_\ell$
    return $\mathcal{E} = \{\mathbf{e}_1, \cdots, \mathbf{e}_L\}$
          $\Upsilon = \{\mathbf{d}_1, \cdots, \mathbf{d}_L\}$
// Compute updates given error neurons
function ComputeUpdates($\mathcal{E}$, $\mathcal{Z}$, Υ)
    for $\ell = 1$ to L **do**
        $\Delta W_\ell = \mathbf{e}_\ell \cdot (\mathbf{z}_{\ell-1})^T$
        if $\ell > 1$ **then**
            if $\ell \mod g \equiv 0$ **then**
                $\Delta E_{L\rightarrow\ell} = -\gamma(\mathbf{d}_{\ell-1} \cdot (\mathbf{e}_L)^T)$
            else
                $\Delta E_{\ell+1\rightarrow\ell} = -\gamma(\mathbf{d}_{\ell-1} \cdot (\mathbf{e}_\ell)^T)$
    $\Delta = \{\Delta W_1, \Delta W_2, \Delta E_2, \cdots, \Delta W_L, \Delta E_L\}$
    return Δ

function $f_\ell(\mathbf{z}_{\ell-1}; \theta_\ell)$ is too challenging, the residual mapping (as indicated by the equation's second term) will be easier to optimize. This gives the network the choice of retaining the input if it finds that a particular layer(s) is unnecessary. The transformation $f_\ell(\mathbf{z}_{\ell-1}; \theta_\ell)$ could range from a linear transformation to a stack of fully-connected layers (as in Equation 1). In computer vision, it is often formulated as a residual "block", i.e., a stack of operations such as convolutions, relu activations ($\mathbf{v} = max(0, \mathbf{v})$), pooling, normalization layers, etc. (see Appendix for the block we used).

Training a residual network with rec-LRA exploits the block-based structure of the network to craft the error message transmission pathways. Since a residual block is a stack of nonlinear transformations, we can choose to embed a vector of error neurons at the output of each residual block and wire them to the output error neurons at layer L. In the case of the two residual blocks (see Appendix, for a visual depiction) for the first level of recursion, we would wire the output layer L directly (via $E_{L\rightarrow i}$) to any residual block output vector $\mathbf{z}_i$. Wiring skip-error connections in this way means that rec-LRA treats each residual block as a computational subgraph (mapping representation $\mathbf{z}_{i-g}$ to $\mathbf{z}_i$). Once skip-error connections wired to each block generate their desired target, rec-LRA will recursively enter the block (with its own unique error pathway) to compute its internal error neurons and weight updates, independently of the blocks above and below, decoupling its update calculation from the rest of the blocks. In treating the residual blocks as decoupled computation sub-graphs, one could view the output of each as a "meta-representation" (in Figure 1a, for the middle model, these would be layers 2, 5, and 9), or a post-activation layer that serves as the first focus of LRA's target generation process. The other layers within it serve as computational "support" layers (layers $1, 3, 4, 6, 7$ in Figure 1a). Algorithm 1 depicts how rec-LRA operates on a (fully-connected) residual network with skip g.

While rec-LRA in the form we have proposed works strictly with strictly neuro-cognitively plausible learning rules (Equations 5, 6 as used in Algorithm 1), one could opt to mix other learning algorithms within the rec-LRA framework. For instance, one could use rec-LRA to generate meta-representation targets for a residual block output $\mathbf{z}_i$ and employ a procedure like backprop (treating the block's output error neuron vector as a proxy for $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_i}$) or Hebbian rules (Hebb 1949) to compute local updates (reducing the number of error synaptic matrices needed, further saving on memory).

Replacing Convolution with Fixed Perturbation: To further save on computation, we investigated replacing convolution with a fixed noise "pseudo-convolution", a mechanism proposed in (Juefei-Xu, Naresh Boddeti, and Savvides 2018) (referred to as a "perturbative layer"). As was shown in (Juefei-Xu, Naresh Boddeti, and Savvides 2018), the pseudo-convolution is drastically faster than the actual convolutional, while the generalization performance of the underlying model using it is comparable to one with convolution. A pseudo-convolution, in our framework, is computed as:

$$\mathbf{z}^c_\ell = \sum_{m=1}^{M} w^m_\ell \phi_r(\mathbf{z}^m_{\ell-1} + \mathbf{n}^m_{\ell-1}), \text{where, } \phi_r(v) = \max(0, v)$$

where w^ℓ_m is a scalar weight that is applied to its corresponding noise map. In the above formula, we see that M noise maps $\mathbf{n}^m_\ell$ must be cycled through in order to compute the final desired output channel (map) $\mathbf{z}^\ell_c$. The idea is that, for the price of the memory required to store the pre-generated M noise maps (the elements of each are each sampled from a centered Gaussian distribution, $\sim \mathcal{N}(0, \sigma^2)$), we side-step the need for learn-able kernel parameters for the convolution operation (cutting out another convolution per filter update). The only parameters in a pseudo-convolution that require updating are the linear combination weights (one update per scalar weight applied to each noise map). Under rec-LRA, which would embed error units next to $\mathbf{z}^c_\ell$, the local update for the mth noise map weight w^ℓ_m would be:

$$\Delta w^m_\ell = \sum_i \sum_j \left(\mathbf{e}_\ell * \left(\phi_r(\mathbf{z}^m_{\ell-1} + \mathbf{n}^m_{\ell-1})\right)^T \right)[i, j]$$

where the update is collapsed by summing over all dimensions to get a final scalar update for the weight w^m_ℓ.

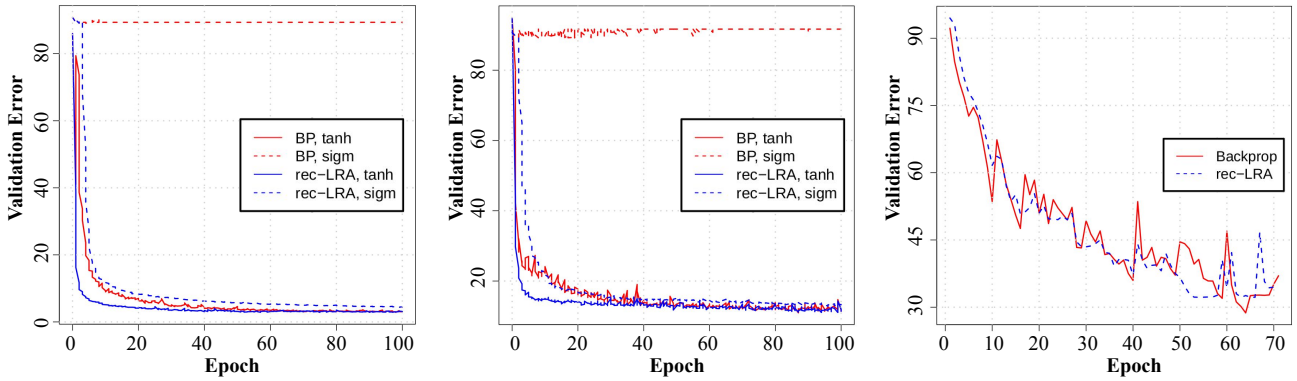

Figure 2: Error for networks on MNIST (left), FMNIST (middle), and ImageNet (using residual networks) (right).

Experiments

We experiment with the proposed rec-LRA and compare it to results reported for other backprop-alternatives. Specifically, we adapted rec-LRA to fully-connected MLPs, convolutional networks (CNNs), and residual networks (ResNet). (See Appendix for experimental, computing infrastructure, hyper-parameter, and code details as well as for metric standard deviation measurements.)

MNIST & Fashion MNIST: This dataset contains 28×28 images with gray-scale pixel values, i.e., range is $[0, 255]$. The only preprocessing applied was to normalize pixel values to the range of $[0, 1]$ by dividing them by 255. On the other hand, Fashion MNIST (FMNIST) (Xiao, Rasul, and Vollgraf 2017) serves as a challenging drop-in replacement for MNIST. Fashion MNIST (pre-processed the same as MNIST) contains images depicting one of 10 clothing items. For both, training had 60000 samples, testing had 10000, and 2000 validation samples were drawn from the training set.

In Table 1, we report our classification error (note: all scores for all results in this section are means over 10 trials, see Appendix for standard deviations) on both training and test sets for rec-LRA. Prior results have been reported for backprop (BP) as well as relevant biologically-motivated algorithms such as feedback alignment (FA), direct feedback alignment (DFA), error-driven local representation alignment (LRA-E), equilibrium propagation (E-Prop), and target propagation (TP) ((Bartunov et al. 2018) and (Ororbia and Mali 2019)). For the rec-LRA results, we report 4 variations (all had 5 layers, 256 units each), each using a different activation function. To be comparable to prior work, the first variant of rec-LRA uses hyperbolic tangent units (rLRA, tanh). The next two variants used were linear rectifier units (rLRA, relu) and exponential linear units (rLRA, elu) to demonstrate compatibility with popular activations. Finally, signum units (rLRA, sign) were tested in order to investigate rec-LRA's ability to train non-differentiable networks. We observe that rec-LRA outperforms all algorithms on FMNIST, including backprop. On MNIST, rec-LRA outperforms all other gradient-free alternatives but does not beat out backprop. While signum networks do not reach the performance of the best networks, they are not the worst performing, offering evidence that non-differentiable networks can make viable classifiers.

	MNIST		FMNIST	
Algorithm	Train	Test	Train	Test
BP	**0.00**	**1.48**	**12.10**	**12.98**
TP	0.00	1.86	21.078	19.66
E-Prop	7.59	9.21	16.56	20.97
LRA-E	0.16	1.97	**9.84**	**12.31**
FA	**0.00**	**1.85**	12.09	12.89
DFA	0.85	2.75	12.58	13.09
WM	0.08	1.99	**7.00**	**12.49**
rLRA, tanh	**0.00**	**1.82**	**6.57**	**11.87**
rLRA, relu	0.22	2.26	8.95	14.13
rLRA, elu	0.09	1.93	9.39	13.17
rLRA, sign (non-diff)	0.85	2.33	12.42	14.88

Table 1: MNIST & FMNIST error (lower is better). Note: Non-diff refers to non-differentiable activation (rRLA only).

We further analyzed the training dynamics of more complex, nonlinear networks, i.e., 8 layers of 256 logistic sigmoid or tanh neurons, trained via backprop and rec-LRA over 100 epochs. Deep sigmoidal models are known to be difficult to train due to the vanishing gradient problem (Glorot and Bengio 2010), especially if Gaussian initialization is used. In Figure 2, for MNIST and FMNIST, we observe that rec-LRA successfully trains networks of both kinds of units with the same initialization and converges sooner. The fact that this result holds for the tanh units, which are friendlier to backprop-centric optimization, offers some evidence of rec-LRA's potential robustness and stability.

CIFAR-10: The CIFAR-10 dataset has $50,000$ training and $10,000$ test images, across 10 categories. Images are of size 32×32 pixels. $5,000$ training samples were set aside to measure validation metrics. Global contrast normalization and ZCA whitening were used to pre-process the images. While CIFAR-10 is more challenging than MNIST/FMNIST, we observe in Table 2 (a) that rec-LRA outperforms networks trained with other gradient-free methods, i.e, target prop, weight mirroring (WM) (Akrout et al. 2019), and feedback alignment. Furthermore, rec-LRA comes close to the performance of the backprop-trained model, evidencing its ability

a) **CIFAR-10**	Train	Test
TP	28.69	39.47
FA	27.46	37.44
DFA	28.74	44.41
CNN-BP	7.89	30.17
CNN-WM	11.79	36.57
CNN-rLRA	10.20	33.50
ResNet-BP	**5.00**	**5.94**
ResNet-WM	**5.97**	**7.04**
ResNet-rLRA	**5.50**	**6.42**

b) **CIFAR-10**	Valid Err	# Ep
cResNet-BP	6.22	125
pResNet-BP	5.94	105
cResNet-WM	7.04	138
pResNet-WM	6.99	111
cResNet-rLRA	7.00	120
pResNet-rLRA	6.42	104

Table 2: CIFAR-10 (a) error & (b) conv (c) vs. pconv (p) performance comparison (Ep = epoch).

ImageNet	Top-1	Top-5
CNN, TP	98.34	94.56
CNN, FA	93.08	82.54
ResNet, FA+BP	73.01	51.24
ResNet, SS (Xiao et al. 2018)	37.91	16.18
ResNet, SS+BP (Xiao et al. 2018)	37.01	15.44
ResNet, FA+WM (Akrout et al. 2019)	**30.20**	N/A
ResNet, KP (Akrout et al. 2019)	**29.2**	N/A
CNN, BP	62.58	39.89
CNN, rLRA	73.69	49.78
ResNet, BP	**28.15**	**9.81**
ResNet, rLRA	**30.78**	**12.04**

Table 3: ImageNet generalization (Top-1 & Top-5) error.

to work on natural images. In the Appendix, we dissect the networks' predictions and visualize latent representations.

To test how performance would change when using either convolution (conv) or pseudo-convolution (pconv), further experiments were conducted on CIFAR-10. We use ResNet-18 with 64 perturbation masks for using pconv (pResNet) and 64 filters for vanilla Resnet (cResNet) (models used skip $g = 2$). The initial learning rate was set to 10^{-2} and a polynomial decay scheme was used to decay the learning rate up to $1e-5$ (with a warm-up schedule set to 40). We trained models with varying batch sizes and report the validation error in Table 2 (b), recording the number of epochs required for each to achieve optimal results ("# Ep"). As seen in Table 2 (b), we found that the performance difference in using pconv over conv was negligible across batch sizes (but gained a speed-up of roughly 0.73 seconds per mini-batch – note that tuned implementations of $1x1$ conv would change the speed-up).

ImageNet: The large-scale benchmark ImageNet (Russakovsky et al. 2015), specifically the ILSVRC-2010 subset, contains over 1.2 million images, of size 224×224, where each classifies as one out of 1000 different categories. Given that the number of classes is large, it is a convention to report two types of error rates: top-1 and top-5. The top-5 error rate is the fraction of test images for which the correct label is not among the 5 classes considered most probable by the evaluated model. In Table 3, we observe that rec-LRA-trained models outperformed ones trained via other gradient-free methods and come quite close to the performance of the backprop-trained architecture (for both top-1/top-5 test errors). We also report the performance of the (best-performing) sign symmetry (SS) method of (Xiao et al. 2018), which we outperform although the margin of improvement is narrower (note that SS also uses partial backprop), weight mirrors (WM), (Akrout et al. 2019), and the classical Kolen-Pollack (KP) algorithm. In Figure 2 (right), we see that rec-LRA does reach a lower validation error sooner than backprop (though this result is not as obvious as it was for MNIST/FMNIST). Furthermore, rec-LRA converges more smoothly than a backprop-trained ResNet. Finally, we measured wall-clock training time for both networks to determine if rec-LRA training offered a speed-up even though we implemented it in simulation without distributed computing hardware (rec-LRA would run dramatically faster with parallelization). Notably, in terms of total training run-time over 90 epochs using a small set of 8 V100 GPUs, the backprop ResNet took 3 hours and 45 minutes (min) to train (speed was about 2.5-2.7 min/epoch) while rec-LRA took 2.127 min/epoch, training over the course of 3 hours and 12 min. Exploiting the inherent parallelism of rec-LRA in implementation would significantly speed up training (see Appendix for limitations).

We note that the state-of-the-art performance of deep networks on ImageNet is still better (Xie et al. 2019) than that obtained by backprop/gradient-free algorithms such as our own and in (Bartunov et al. 2018). However, our aim was to show that a backprop-free algorithm can generalize on difficult, large-scale datasets, although integrating the modern-day heuristics that have been used to carefully tune ImageNet models would boost our networks' performance further.

Conclusions

This paper proposed the neurobiologically-inspired learning algorithm, recursive local representation alignment (rec-LRA), for training deep neural architectures. rec-LRA generalizes as well as backpropagation (backprop) and outperforms other current gradient/backprop-free procedures across several datasets, notably on the massive-scale ImageNet. Furthermore, it offers improved convergence due to faster, parallelizable weight updates, as shown in our experiments. As a result, this work offers empirical evidence that a backprop-free learning procedure for artificial neural networks can indeed scale up to larger datasets.

References

Ahmad, N.; van Gerven, M. A.; and Ambrogioni, L. 2020. GAIT-prop: A biologically plausible learning rule derived from backpropagation of error. *arXiv preprint arXiv:2006.06438*.

Akrout, M.; Wilson, C.; Humphreys, P.; Lillicrap, T.; and Tweed, D. B. 2019. Deep Learning without Weight Transport.

In Wallach, H.; Larochelle, H.; Beygelzimer, A.; dÁlché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 976–984. Curran Associates, Inc.

Alias Parth Goyal, A. G.; Ke, N.; Ganguli, S.; and Bengio, Y. 2017. Variational Walkback: Learning a Transition Operator as a Stochastic Recurrent Net. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*, 4392–4402. Curran Associates, Inc.

Balduzzi, D.; Vanchinathan, H.; and Buhmann, J. M. 2015. Kickback Cuts Backprop's Red-Tape: Biologically Plausible Credit Assignment in Neural Networks. In *AAAI*, 485–491.

Bartunov, S.; Santoro, A.; Richards, B.; Marris, L.; Hinton, G. E.; and Lillicrap, T. 2018. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, 9390–9400.

Belilovsky, E.; Eickenberg, M.; and Oyallon, E. 2018. Greedy layerwise learning can scale to imagenet. *arXiv preprint arXiv:1812.11446*.

Belilovsky, E.; Eickenberg, M.; and Oyallon, E. 2019. Decoupled greedy learning of cnns. *arXiv preprint arXiv:1901.08164*.

Bengio, Y. 2014. How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *CoRR*, abs/1407.7906.

Bengio, Y.; Lamblin, P.; Popovici, D.; Larochelle, H.; et al. 2007. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19: 153.

Carreira-Perpiñán, M. Á.; and Wang, W. 2012. Distributed optimization of deeply nested systems. *CoRR*, abs/1212.5921.

Chalasani, R.; and Principe, J. C. 2013. Deep predictive coding networks. *arXiv preprint arXiv:1301.3541*.

Clark, A. 2013. Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3): 181–204.

Crick, F. 1989. The recent excitement about neural networks. *Nature*, 337(6203): 129–132.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*.

Földiak, P. 1990. Forming sparse representations by local anti-Hebbian learning. *Biological cybernetics*, 64(2): 165–170.

Friston, K. 2009. The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7): 293–301.

Fu, Z.; Wu, D.-A. J.; Ross, I.; Chung, J. M.; Mamelak, A. N.; Adolphs, R.; and Rutishauser, U. 2019. Single-neuron correlates of error monitoring and post-error adjustments in human medial frontal cortex. *Neuron*, 101(1): 165–177.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Grossberg, S. 1987. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1): 23 – 63.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In *European conference on computer vision*, 630–645. Springer.

Hebb, D. O. 1949. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology.*, 62–78.

Hinton, G. E. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8): 1771–1800.

Jaderberg, M.; Czarnecki, W. M.; Osindero, S.; Vinyals, O.; Graves, A.; and Kavukcuoglu, K. 2016. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*.

Juefei-Xu, F.; Naresh Boddeti, V.; and Savvides, M. 2018. Perturbative neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3310–3318.

Krotov, D.; and Hopfield, J. J. 2019. Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, 116(16): 7723–7731.

Lee, C.-Y.; Xie, S.; Gallagher, P.; Zhang, Z.; and Tu, Z. 2014. Deeply-Supervised Nets. *arXiv:1409.5185 [cs, stat]*.

Lee, D.-H.; Zhang, S.; Fischer, A.; and Bengio, Y. 2015. Difference Target Propagation. In *Proceedings of the 2015th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ECMLPKDD'15, 498–515. Switzerland: Springer. ISBN 978-3-319-23527-1.

Liao, Q.; Leibo, J. Z.; and Poggio, T. A. 2016. How important is weight symmetry in backpropagation? In *AAAI*, 1837–1844.

Lillicrap, T. P.; Cownden, D.; Tweed, D. B.; and Akerman, C. J. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7: 13276.

Mishkin, D.; and Matas, J. 2015. All you need is a good init. *CoRR*, abs/1511.06422.

Nøkland, A. 2016. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, 1037–1045.

Nøkland, A.; and Eidnes, L. H. 2019. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*.

Ororbia, A. G.; and Mali, A. 2019. Biologically motivated algorithms for propagating local target representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4651–4658.

Ororbia II, A. G.; Haffner, P.; Reitter, D.; and Giles, C. L. 2017. Learning to Adapt by Minimizing Discrepancy. *arXiv preprint arXiv:1711.11542*.

Ororbia II, A. G.; Reitter, D.; Wu, J.; and Giles, C. L. 2015. Online learning of deep hybrid architectures for semi-supervised categorization. In *Machine Learning and Knowledge Discovery in Databases (Proceedings, ECML PKDD 2015)*, volume 9284 of *Lecture Notes in Computer Science*, 516–532. Porto, Portugal: Springer.

Rao, R. P.; and Ballard, D. H. 1999. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1).

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *nature*, 323(6088): 533–536.

Rumelhart, D. E.; and Zipser, D. 1985. Feature discovery by competitive learning. *Cognitive science*, 9(1): 75–112.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252.

Sacramento, J.; Costa, R. P.; Bengio, Y.; and Senn, W. 2018. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, 8721–8732.

Scellier, B.; and Bengio, Y. 2017. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11: 24.

Srivastava, R. K.; Greff, K.; and Schmidhuber, J. 2015. Highway networks. *arXiv preprint arXiv:1505.00387*.

Sussillo, D. 2014. Random Walks: Training Very Deep Nonlinear Feed-Forward Networks with Smart Initialization. *CoRR*, abs/1412.6558.

Taylor, G.; Burmeister, R.; Xu, Z.; Singh, B.; Patel, A.; and Goldstein, T. 2016. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, 2722–2731.

Vincent, P.; Larochelle, H.; Bengio, Y.; and Manzagol, P.-A. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, 1096–1103. ACM.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Xiao, W.; Chen, H.; Liao, Q.; and Poggio, T. 2018. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*.

Xie, Q.; Hovy, E.; Luong, M.-T.; and Le, Q. V. 2019. Self-training with Noisy Student improves ImageNet classification. *arXiv preprint arXiv:1911.04252*.

Xie, X.; and Seung, H. S. 2003. Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural computation*, 15(2): 441–454.