Biologically Motivated Algorithms for Propagating Local Target Representations

Alexander G. Ororbia[*]
Rochester Institute of Technology
102 Lomb Memorial Drive, Rochester NY, USA 14623
ago@cs.rit.edu

Ankur Mali[*]
Penn State University
Old Main, State College, PA 16801
aam35@ist.psu.edu

Abstract

Finding biologically plausible alternatives to back-propagation of errors is a fundamentally important challenge in artificial neural network research. In this paper, we propose a learning algorithm called error-driven Local Representation Alignment (LRA-E), which has strong connections to predictive coding, a theory that offers a mechanistic way of describing neurocomputational machinery. In addition, we propose an improved variant of Difference Target Propagation, another procedure that comes from the same family of algorithms as LRA-E. We compare our procedures to several other biologically-motivated algorithms, including two feedback alignment algorithms and Equilibrium Propagation. In two benchmarks, we find that both of our proposed algorithms yield stable performance and strong generalization compared to other competing back-propagation alternatives when training deeper, highly nonlinear networks, with LRA-E performing the best overall.

Behind the modern achievements in artificial neural network research is back-propagation of errors (**?**) (or "backprop"), the key training algorithm used in computing updates for the parameters that define the computational architectures applied to problems ranging from computer vision to speech recognition. However, though neural architectures are inspired by our current understanding of the human brain, the connections to the actual mechanisms of systems of natural neurons are often very loose, at best. More importantly, backprop faces some of the strongest neuro-biological criticisms, argued to be a highly implausible way in which learning occurs in the human brain.

Among the many problems with back-propagation, some of the most prominent are: 1) the "weight transport problem", where the feedback weights that carry back error signals must be the transposes of the feedforward weights, 2) forward and backward propagation utilize different computations, and 3) the error gradients are stored separately from the activations. These problems, as originally argued in (**?**; **?**), largely center around one critical component of backprop–the global feedback pathway needed for transporting error derivatives across the system. This pathway is necessary given the design of modern supervised learning systems–a loss measures error between a model's output units and a target, e.g., class label, and the global pathway relates how the internal processing elements affect this error. When considering modern theories of the brain (**?**; **?**; **?**), which posit that local computations occur at multiple levels of a somewhat hierarchical structure, this global pathway should not be necessary to learn effectively. Furthermore, this pathway results makes training very deep networks difficult–due to many multiplications that underly traversing along the global feedback pathway, error gradients explode/vanish (**?**). To fix this, gradients are kept within reasonable magnitudes by requiring layers to behave sufficiently linearly. However, this remedy creates other highly undesirable side-effects, e.g., adversarial samples (**?**), and prevents usage of neuro-biological mechanisms such as lateral competition and discrete-valued/stochastic activations (since the pathway requires precise knowledge of function derivatives (**?**)).

If we remove this global feedback pathway, we create a new problem–what are the learning signals for the hidden processing elements? This problem is one of the main concerns of the recently introduced *Discrepancy Reduction* family of learning algorithms (**?**). In this paper, we will develop two learning algorithms within this family–error-driven Local Representation Alignment and adaptive noise Difference Target Propagation. In experiments on two classification benchmarks, we will show that these two algorithms generalize better than a variety of other biologically motivated learning approaches, all without employing the global feedback pathway required by back-propagation.

Coordinated Local Learning Algorithms

Algorithms within the Discrepancy Reduction (**?**) family offer computational mechanisms for two key steps when learning from patterns. These steps include:

1. Search for latent representations that better explain the input/output, also known as target representations. This creates the need for local (higher-level) objectives that will guide current latent representations towards better ones.

2. Reduce, as much as possible, the mismatch between a model's currently "guessed" representations and target representations. The sum of the internal, local losses is also defined as the total discrepancy in a system, and can also be thought of as a sort of pseudo-energy function.

[*] Both authors contributed equally.

This general process forms the basis of what we call *coordinated local learning rules*. Computing targets with these kinds of rules should not require an actual pathway, as in back-propagation, and instead make use of top-down and bottom-up signals to generate targets. This idea is particularly motivated by the theory of predictive coding (**?**) (which started to impact modern machine learning applications (**?**)), which claims that the brain is in a continuous process of creating and updating hypotheses (using error information) to predict the sensory input. This paper will explore two ways in which this hypothesis updating (in the form of local target creation) might happen: 1) through error-correction in Local Representation Alignment (LRA-E), and 2) through repeated encoding and decoding as in Difference Target Propagation (DTP). It should also be noted that one is not restricted to only using neural building blocks–LRA-E could be used to train stacked Gradient Boosted Decision Trees (GBDTs), which would be faster than in (**?**), which employed a form of target propagation to calculate local updates.

The idea of learning locally, in general, is slowly becoming prominent in the training of artificial neural networks, with recent proposals including decoupled neural interfaces (**?**) and kickback (**?**) (which was derived specifically for regression problems). Furthermore, (**?**) demonstrated that neural models using simple local Hebbian updates (within a predictive coding framework) could efficiently conduct supervised learning. Far earlier approaches that employed local learning included the layer-wise training procedures that were once used to build models for unsupervised learning (**?**), supervised learning (**?**), and semi-supervised learning (**?**; **?**). The key problem with these older algorithms is that they were greedy–a model was built from the bottom-up, freezing lower-level parameters as higher-level feature detectors were learnt.

Another important idea underlying algorithms such as LRA and DTP is that learning is possible with asymmetry–which directly resolves the weight-transport problem (**?**; **?**), another strong neuro-biological criticism of backprop. This is even possible, surprisingly, if the feedback weights are random and fixed, which is at the core of two algorithms we will also compare to–Random Feedback Alignment (RFA) (**?**) and Direct Feedback Alignment (DFA) (**?**). RFA replaces the transpose of the feedforward weights in backprop with a similarly-shaped random matrix while DFA directly connects the output layer's pre-activation derivative to each layer's post-activation. It was shown in (**?**; **?**) that these feedback loops would be better suited in generating target representations.

Local Representation Alignment

To concretely describe how LRA is practically implemented, we will specify how LRA is applied to a 3-layer feedforward network, or multilayer perceptron (MLP). Note that LRA generalizes to models with more layers ($L \geq 3$).

The pre-activities of the MLP at layer ℓ are denoted as $\mathbf{h}^\ell$ while the post-activities, or the values output by the nonlinearity $\phi_\ell(\cdot)$, are denoted as $\mathbf{z}^\ell$. The target variable used to correct the output units ($\mathbf{z}^L$) is denoted as $\mathbf{y}^L_z$ ($\mathbf{y}^L_z = \mathbf{y}$, or $\mathbf{y}^L_z = \mathbf{x}$ if we are learning auto-associative functions). Connecting one layer of neurons $\mathbf{z}^{\ell-1}$, with pre-activities $\mathbf{h}^{\ell-1}$, to another layer $\mathbf{z}^\ell$, with pre-activities $\mathbf{h}^\ell$, are synaptic weights W_ℓ. The propagation equations for computing pre-activtion and post-activation values for layer ℓ are:

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

Before computing targets or updates, we first must define the set of local losses, one per layer of neurons except for the input neurons, that constitute the measure of total discrepancy inside the MLP, $\{\mathcal{L}_1(\mathbf{y}^1_z, \mathbf{z}^1), \mathcal{L}_2(\mathbf{y}^2_z, \mathbf{z}^2), \mathcal{L}_3(\mathbf{y}^3_z, \mathbf{z}^3)\}$. With losses defined, we can then explicitly formulate the error units $\mathbf{e}_\ell$ for each layer as well, since any given layer's error units correspond to the first derivative of that layer's loss with respect to that layer's post-activation values. For the MLP's output layer, we could assume a categorical distribution, which is appropriate for 1-of-k classification tasks, and use the following negative log likelihood loss:

$$\mathcal{L}_\ell(\mathbf{y}^\ell_z, \mathbf{z}^\ell) = -\frac{1}{2}\sum_{i=1}^{|\mathbf{z}|} \mathbf{y}^\ell_z[i] \log \mathbf{z}^\ell[i],$$
$$\mathbf{e}_\ell = \mathbf{e}_\ell(\mathbf{y}^\ell_z, \mathbf{z}^\ell) = \frac{-\mathbf{y}^\ell_z}{\mathbf{z}^\ell}, \tag{2}$$

where the loss is computed over all dimensions $|\mathbf{z}|$ of the vector $\mathbf{z}$ (where a dimension is indexed/accessed by integer i). Note that for this loss function, we assume that $\mathbf{z}$ is a vector of probabilities computed by using the softmax function as the output nonlinearity, $\mathbf{z}^3 = \frac{exp(\mathbf{h}^3)}{\sum_i exp(\mathbf{h}^3_i)}$. For the hidden layers, we can choose between a wider variety of loss functions, and in this paper, we experimented with assuming either a Gaussian or Cauchy distribution over the hidden units. For the Gaussian distribution (or L2 norm), we have the following:

$$\mathcal{L}_\ell(\mathbf{z}, \mathbf{y}) = \frac{1}{(2\sigma^2)} \sum_{i=1}^{|\mathbf{z}|} (\mathbf{y}_i - \mathbf{z}_i)^2$$
$$\mathbf{e}_\ell = \mathbf{e}_\ell(\mathbf{y}^\ell_z, \mathbf{z}^\ell) = \frac{-(\mathbf{y}^\ell_z - \mathbf{z}^\ell)}{\sigma^2} \tag{3}$$

where σ^2 represents fixed scalar variance (we set $\sigma^2 = 1/2$). For the Cauchy distribution (or log-penalty), we obtain:

$$\mathcal{L}_\ell(\mathbf{z}, \mathbf{y}) = \sum_{i=1}^{|\mathbf{z}|} \log(1 + (\mathbf{y}_i - \mathbf{z}_i)^2)$$
$$\mathbf{e}_\ell = \mathbf{e}_\ell(\mathbf{y}^\ell_z, \mathbf{z}^\ell) = \frac{-2(\mathbf{y}^\ell_z - \mathbf{z}^\ell)}{(1 + (\mathbf{y}^\ell_z - \mathbf{z}^\ell)^2)}. \tag{4}$$

For the activation function used in calculating the hidden post-activities, we use the hyperbolic tangent, or $\phi_\ell(v) = \frac{exp(2v)-1}{exp(2v)+1}$. Using the Cauchy distribution proved particularly useful in our experiments because it encourages sparse representations and aligns nicely with the biological considerations of sparse coding (**?**) and predictive sparse decomposition (**?**) as well as the lateral competition (**?**) that naturally occurs in groups of neural processing elements. These are relatively simple local losses for measuring the agreement between representations and targets and future work should entail developing even better metrics.

Algorithm 1 LRA-E: Target and update computations.

```
// Procedure for computing error units & targets
Input: sample $(\mathbf{y}, \mathbf{x})$ and $\Theta = \{W_1, W_2, W_3, E_2, E_3\}$
function COMPUTETARGETS($(\mathbf{y}, \mathbf{x}), \Theta$)
    // Run feedforward weights to get activities
    $\mathbf{h}^1 = W_1 \mathbf{z}^0$, $\mathbf{z}^1 = \phi_1(\mathbf{h}^1)$
    $\mathbf{h}^2 = W_2 \mathbf{z}^1$, $\mathbf{z}^2 = \phi_2(\mathbf{h}^2)$
    $\mathbf{h}^3 = W_3 \mathbf{z}^2$, $\mathbf{z}^3 = \phi_3(\mathbf{h}^3)$
    $\mathbf{y}^3_z \Leftarrow \mathbf{y}$
    $\mathbf{e}_3 = \frac{-\mathbf{y}^3_z}{\mathbf{z}^3}$, $\mathbf{y}^2_z \leftarrow \phi_2\Big(\mathbf{h}^2 - \beta(E_3 \mathbf{e}_3)\Big)$
    $\mathbf{e}_2 = -2(\mathbf{y}^2_z - \mathbf{z}^2)$
    $\mathbf{y}^1_z \leftarrow \phi_1\Big(\mathbf{h}^1 - \beta(E_2 \mathbf{e}_2)\Big)$
    $\mathbf{e}_1 = -2(\mathbf{y}^1_z - \mathbf{z}^1)$
    $\Lambda = (\mathbf{z}^3, \mathbf{z}^2, \mathbf{z}^1, \mathbf{h}^3, \mathbf{h}^2, \mathbf{h}^1, \mathbf{e}^3, \mathbf{e}^2, \mathbf{e}^1)$
    Return $\Lambda$
// Procedure(s) for computing weight updates
Input: sample $(\mathbf{y}, \mathbf{x})$ and calculations $\Lambda$
function CALCUPDATES-V1($(\mathbf{y}, \mathbf{x}), \Theta, \Lambda$)
    $\Delta W_3 = (\mathbf{e}_3 \otimes \phi'_3(\mathbf{h}_3))(\mathbf{z}_2)^T$
    $\Delta W_2 = (\mathbf{e}_2 \otimes \phi'_2(\mathbf{h}_2))(\mathbf{z}_1)^T$
    $\Delta W_1 = (\mathbf{e}_1 \otimes \phi'_1(\mathbf{h}_1))(\mathbf{x})^T$
    $\Delta E_3 = -\gamma(\Delta W_3)^T$
    $\Delta E_2 = -\gamma(\Delta W_2)^T$
    Return $(\Delta W_3, \Delta W_2, \Delta W_1, \Delta E_3, \Delta E_2)$
function CALCUPDATES-V2($(\mathbf{y}, \mathbf{x}), \Theta, \Lambda$)
    $\Delta W_3 = \mathbf{e}_3(\mathbf{z}_2)^T$
    $\Delta W_2 = \mathbf{e}_2(\mathbf{z}_1)^T$
    $\Delta W_1 = \mathbf{e}_1(\mathbf{x})^T$
    $\Delta E_3 = -\gamma(\Delta W_3)^T$
    $\Delta E_2 = -\gamma(\Delta W_2)^T$
    Return $(\Delta W_3, \Delta W_2, \Delta W_1, \Delta E_3, \Delta E_2)$
```

With local losses specified and error units implemented, all that remains is to define how targets are computed and what the parameter updates will be. At any given layer $\mathbf{z}^\ell$, starting at the output units (in our example, $\mathbf{z}^3$), we calculate the target for the layer below $\mathbf{z}^{\ell-1}$ by multiplying the error unit values at ℓ by a set of synaptic error weights E_ℓ. This projected displacement, weighted by the modulation factor β,[1] is then subtracted from the initially found pre-activation of the layer below $\mathbf{h}^{\ell-1}$. This updated pre-activity is then run through the appropriate nonlinearity to calculate the final target $\mathbf{y}^{\ell-1}_z$. This computation amounts to:

$$\mathbf{e}_\ell = -2(\mathbf{y}^\ell_z - \mathbf{z}^\ell), \quad \Delta \mathbf{h}^{\ell-1} = E_\ell \mathbf{e}_\ell \tag{5}$$

$$\mathbf{y}^{\ell-1}_z \leftarrow \phi_{\ell-1}\Big(\mathbf{h}^{\ell-1} - \beta(\Delta \mathbf{h}^{\ell-1})\Big). \tag{6}$$

Once the targets for each layer have been found, we can then use the local loss $\mathcal{L}^\ell(\mathbf{y}^\ell_z, \mathbf{z}^\ell)$ to compute updates to the weights W_ℓ and its corresponding error weights E_ℓ.[2] The update calculation for parameters at layer ℓ would be:

$$\Delta W_\ell = (\mathbf{e}_\ell \otimes \phi'_\ell(\mathbf{h}_\ell))(\mathbf{z}_{\ell-1})^T, \Delta E_\ell = -\gamma(\Delta W_\ell)^T, \tag{7}$$

$$\text{or,} \quad \Delta W_\ell = \mathbf{e}_\ell(\mathbf{z}_{\ell-1})^T, \Delta E_\ell = -\gamma(\Delta W_\ell)^T \tag{8}$$

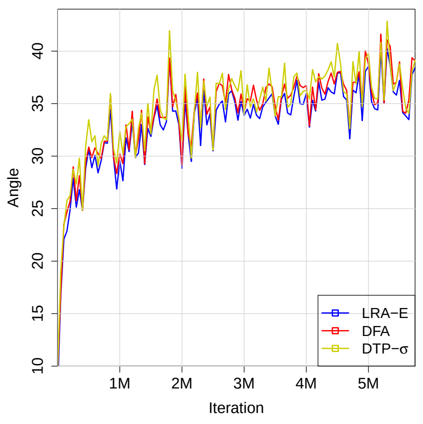
where $\otimes$ indicates the Hadamard product and γ is a decay factor (a value that we found should be set to less than 1.0) meant to ensure that the error weights change more slowly than the forward weights. An attractive property of LRA is that the derivatives of the pointwise activation functions can be dropped, yielding the second variation of the update rule, as long as the activation function is monotonically non-decreasing in its input (for stochastic activation functions, the output distribution for a larger input should stochastically dominate the output distribution for a smaller input). This is also satisfying from a biological perspective since it is unlikely that neurons would utilize point-wise activation derivatives in computing updates (**?**). The update for error weights is simply proportional to the negative transpose of the update computed for the matching forward weights, which is a computationally fast and cheap rule we propose inspired by (**?**).

In Algorithm 1, the equations in this section are combined to create the full procedure for training a 3-layer MLP (using either CALCUPDATES-V1($\cdot$) or CALCUPDATES-V2($\cdot$) to compute weight updates), assuming Gaussian local losses and their respective error units. The model is defined by $\Theta = \{W_1, W_2, W_3, E_2, E_3\}$ (biases $\mathbf{c}_\ell$ omitted for clarity). We will refer to Algorithm 1 as *LRA-E* (which easily extends to $L > 3$).

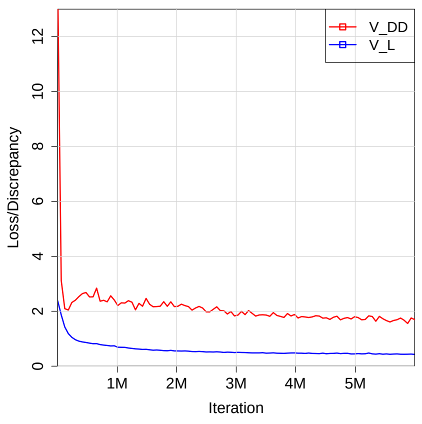
In Figure 1(a), we compare the updates calculated by *LRA-E* (as well as DFA and our proposed DTP-σ, described later) with those given by back-propagation, after each mini-batch, by plotting the angles over the first 20 epochs of learning for a 3-layer MLP (256 units per layer) trained with stochastic gradient descent (SGD) with mini-batches of 50 image samples using a categorical output loss and Gaussian local losses. As long as the angle of the updates computed from LRA are within 90 degrees of the updates obtained by back-propagation, LRA will move parameters towards the same general direction as back-propagation (which greedily points in the direction of steepest descent) and will still find good local optima. In Figure 1(a), this does indeed appear to be the case for the MLP example. The angle, fortunately, while certainly non-zero, never deviates too far from the direction pointed by back-propagation and remains relatively stable throughout the learning process. (Observe that DFA and DTP -σ have, interestingly enough, update angles that are quite similar to LRA-E.) Alongside Figure 1(a), in Figure 1(b), we plot our neural model's total internal discrepancy, $D(\mathbf{y}, \mathbf{x})$ (or *V_DD*), which is a simple linear combination of all of the internal local losses for a given data point. Observe that while the (validation) output loss (*V_L*) continually decreases, *V_DD* does not always appear to do so. We conjecture that this "bump", which appears at the start of learning, is the result of the evolution of LRA-E's error weights, which are used to directly control the target generation process. So even though backprop and LRA-E might start down the same path in error space (or on the loss surface), as indicated by the initially low angle between updates, this trajectory

[1]In this paper, $\beta = 0.1$, found with only minor prelim. tuning.

[2]Except for W_1, which has no corresponding error weights E_1.



(a) Gradient angle (degrees) compared to BP.



(b) Total discrepancy & output NLL.

Figure 1: In Figure 1(a), we compare the updates calculated by LRA-E, DFA, and DTP-σ against backprop (BP). In Figure 1(b), we show how total discrepancy for an LRA-trained MLP evolves during training on Fashion MNIST, alongside the output loss.

is not ideal for LRA's units/targets. This means that error weights will change more rapidly at training's start, resulting in targets that vary quite a bit (raising internal loss values). However, once the error weights start to converge to an approximate transpose of the feedforward weights, the process of correction becomes easier and *V_DD* desirably declines.

Improving Difference Target Propagation

As mentioned earlier, Difference Target Propagation (DTP) (and also, less directly, recirculation (**?**; **?**)), like LRA-E, also falls under the same family of algorithms concerned with minimizing internal discrepancy, as shown in (**?**; **?**). However, DTP takes a very different approach to computing alignment targets–instead of transmitting messages through error units and error feedback weights as in LRA-E, DTP employs feedback weights to learn the inverse of the mapping created by the feedforward weights. However, (**?**) showed that DTP struggles to assign good local targets as the network becomes deeper, i.e., more highly nonlinear, facing an initially promising albeit brief phase in learning where generalization error decreases (within the first few epochs) before ultimately collapsing (unless very specific initializations are used). One potential cause of this failure could be the lack of a strong enough mechanism to globally coordinate the local learning problems created by the encoder-decoder pairs that underlie the system. In particular, we hypothesize that this problem might be coming from the noise injection scheme, which is local and fixed, offering no adaptation to each specific layer and making some of the layerwise optimization problems more difficult than necessary. Here, we will aim to remove this potential cause through an adaptive layerwise corruption scheme.

Assuming we have a target calculated from above $\mathbf{y}^\ell_z$, we consider the forward weights W_ℓ connecting the layer $\mathbf{z}^{\ell-1}$ to layer $\mathbf{z}^\ell$ and the decoding weights E_ℓ that define the inverse mapping between the two. The first forward propagation step is the same as in Equation 1. In contrast to LRA-E's error-driven way of computing targets, we consider each pair of neuronal layers, $(\mathbf{z}^\ell, \mathbf{z}^{\ell-1})$, as forming a particular type of encoding/decoding cycle that will be used in computing layerwise targets. To calculate the target $\mathbf{y}^{\ell-1}_z$, we update the original post-activation $\mathbf{z}^{\ell-1}$ using the linear combination of two applications of the decoding weights as follows:

$$\mathbf{y}^{\ell-1}_z = \mathbf{z}^{\ell-1} - \big(\phi_{\ell-1}(E_\ell \mathbf{z}^\ell) + \phi_{\ell-1}(E_\ell \mathbf{y}^\ell_z)\big) \quad (9)$$

where we see that we decode two times, one from the original post-activation calculated from the feedforward pass of the MLP and another from the target value generated from the encoding/decoding process from the layer pair above, e.g. $(\mathbf{z}^{\ell+1}, \mathbf{z}^\ell)$. This will serve as the target when training the forward weights for the layer below $W_{\ell-1}$. To train the inverse-mapping weights E_ℓ, as required by the original version of DTP, zero-mean Gaussian noise, $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with fixed standard deviation σ, is injected into $\mathbf{z}^{\ell-1}$ followed by re-running the encoder and the decoder on this newly corrupted activation vector. Formally, this is defined as:

$$\widehat{\mathbf{y}}^{\ell-1}_z = \mathbf{z}^{\ell-1} + \epsilon, \quad \widehat{\mathbf{z}}^{\ell-1} = \phi_{\ell-1}(E_\ell \phi_\ell(W_\ell \widehat{\mathbf{y}}^{\ell-1}_z)) \quad (10)$$

This process we will refer to as *DTP*. In our proposed, improved variation of DTP, or *DTP-σ*, we will take an "adaptive" approach to the noise injection process ϵ. To develop our adaptive noise scheme, we have taken some insights from studies of biological neuron systems, which show there are varying levels of signal corruption in different neuronal layers/groups (**?**; **?**; **?**; **?**). It has been argued that this noise variability enhances neurons' overall ability to detect and transmit signals across a system (**?**; **?**; **?**) and, furthermore, that the presence of this noise yields more robust representations (**?**; **?**; **?**). There also is biological evidence demonstrating that an increase in the noise level across successive groups of neurons is thought to help in local neural computation (**?**; **?**; **?**).

In light of this, the standard deviation σ of the noise process should be a function of the noise across layers, and an interesting way in which we implemented this was to make σ_ℓ (the standard deviation of the noise injection at layer ℓ)

a function of the local loss measurements. At the top layer, we can set $\sigma_L = \alpha$ (a small, fixed value such as $\alpha = 0.01$ worked well in our experiments). The standard deviation for the layers below would be a function of where the noise process is within the network, indexed by ℓ. This means that:

$$\sigma_\ell = \sigma_{\ell+1} - \mathcal{L}_\ell(\mathbf{y}_z^{\ell-1}, \mathbf{z}^{\ell-1}) \tag{11}$$

noting that the local loss chosen for DTP is a Gaussian loss (but with the input arguments flipped–the target value is now the corrupted initial encoding and the prediction is the clean, original encoding, or $\mathcal{L}_\ell(\mathbf{z} = \mathbf{y}_z^{\ell-1}, \mathbf{y} = \mathbf{z}^{\ell-1})$).

The updates to the weights are calculated by differentiating each local loss with respect to the appropriate encoder weights, or $\Delta W_{\ell-1} = \frac{\partial \mathcal{L}(\mathbf{z}^{\ell-1}, \mathbf{y}_z^{\ell-1})}{\partial W_{\ell-1}}$, or with respect to the decoder synaptic weights $\Delta E_\ell = \frac{\partial \mathcal{L}(\widehat{\mathbf{z}}^\ell, \widehat{\mathbf{y}}_z^\ell)}{\partial E_\ell}$. Note that the order of the input arguments to each loss function for these two partial derivatives is important for obtaining the correct sign to multiply the gradients by, and furthermore staying aligned with the original formulation of DTP (**?**), .

As we will see in our experimental results, *DTP-σ* is a much more stable learning algorithm (especially with respect to the original DTP), especially when training deeper/wider networks. *DTP-σ* benefits from a stronger form of overall coordination among its internal encoding/decoding subproblems through the pair-wise comparison of local loss values that drive the hidden layer corruption.

A Comment on the Efficiency of LRA-E and DTP

Note that *LRA-E*, while a bit slower than backprop per update (given its use of the error weights to generate hidden layer targets), is much faster than *DTP* and *DTP-σ*. Specifically, if we focus on matrix multiplications used to find targets, which make up the bulk of the computation underlying both processes, *LRA-E* only requires $2(L-1)$ matrix multiplications while *DTP* and *DTP-σ* require $4(L-3)+L$ multiplications. In particular, DTP has a very expensive target generation phase, requiring 2 applications of the encoder parameters (1 of these is from the network's initial feedfoward pass) and 3 applications of the decoder parameters to create targets to train the forward weights and inverse-mapping weights.

Experimental Results

In this section, we present experimental results of training MLPs using a variety of learning algorithms.

MNIST: This dataset [3] contains 28×28 images with gray-scale pixel feature values in the range of $[0, 255]$. The only preprocessing applied to this data is to normalize the pixel values to the range of $[0, 1]$ by dividing them by 255.

Fashion MNIST: This database (**?**) contains $28x28$ grey-scale images of clothing items, meant to serve as a much more difficult drop-in replacement for MNIST itself. Training contains 60000 samples and testing contains 10000, each image is associated with one of 10 classes. We create a validation set of 2000 samples from the training split. Preprocessing was the same as on MNIST.

[3]Available at the URL: http://yann.lecun.com/exdb/mnist/.

For both datasets and all models, over 100 epochs, we calculate updates over mini-batches of 50 samples. Furthermore, we do not regularize parameters any further, e.g., drop-out (**?**) or weight penalties. All feedfoward architectures for all experiments were of either 3, 5, or 8 hidden layers of 256 processing elements. The post-activation function used was the hyperbolic tangent and the top layer was chosen to be a maximum-entropy classifier (i.e., a softmax function). The output layer objective for all algorithms was to minimize the categorical negative log likelihood.

Parameters were initialized using a scheme that gave best performance on the validation split of each dataset on a per-algorithm basis. Though we wanted to use very simple initialization schemes for all algorithms, in preliminary experiments, we found that the feedback alignment algorithms as well as *DTP* (and *DTP-σ*) worked best when using a uniform fan-in-fan-out scheme (**?**). (**?**) confirms this result, originally showing how these algorithms often are unstable or fail to perform well using initializations based on simple uniform or Gaussian distributions. For LRA-E, however, we initialized the parameters using a zero-mean Gaussian distribution (variance of 0.05).

The choice of parameter update rule was also somewhat dependent on the learning algorithm employed. Again, as shown in (**?**), it is difficult to get good, stable performance from algorithms, such as the original DTP, when using simple SGD. As done in (**?**), we used the RMSprop (**?**) adaptive learning rate with a global step size of $\lambda = 0.001$. For Backprop, RFA, DFA, and LRA-E, we were able to use SGD ($\lambda = 0.01$).

Classification Performance

In this experiment, we compare all of the algorithms discussed earlier. These include back-propagation (Backprop), Random Feedback Alignment (RFA) (**?**), Direct Feedback Alignment (DFA) (**?**), Equilibrium Propagation (**?**) (Equil-Prop) and the original Difference Target Propagation (**?**) (DTP). Our algorithms include our proposed, improved version of DTP (*DTP-σ*) and the proposed error-driven Local Representation Alignment (LRA-E).

The results of our experiments are presented in Tables 1 and 2. Test and training scores are reported for the set of model parameters that had lowest validation error. Observe that LRA-E is the most stable and consistently well-performing algorithm compared to the other backprop alternatives, closely followed by our *DTP-σ*. More importantly, algorithms like Equil-Prop and DTP appear to break down when training deeper networks, i.e., the 8-layer MLP. Note that while DTP was used to successfully train a 7-layer network of 240 units (using RMSprop) (**?**), we followed the same settings reported for networks deeper than 7 and in our experiments uncovered that the algorithm begins to struggle as the layers are made wider, starting with the width of 256. However, this problem is rectified using DTP-σ, leading to much more stable performance and even to cases where the algorithm completely overfits the training set (as in the case of 3 and 5 layers for MNIST). Nonetheless, LRA-E still performs best with respect to generalization across both datasets, despite using a naïve initialization scheme. Table 3 shows the

Model	3 Layers		5 Layers		8 Layers	
	Train	Test	Train	Test	Train	Test
Backprop	1.78	3.02	2.4	2.98	2.91	3.02
Equil-Prop	3.82	4.99	7.59	9.21	89.96	90.91
RFA	3.01	3.13	2.99	3.4	3.59	3.76
DFA	4.07	4.17	3.71	3.88	3.81	3.85
DTP	0.74	2.8	4.408	4.94	10.89	10.1
DTP-σ (ours)	0.00	2.38	0.00	2.57	0.00	2.56
LRA-E (ours)	0.86	2.20	0.16	1.97	0.08	2.55

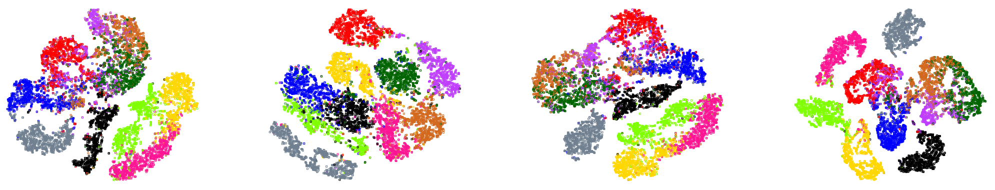
Table 1: MNIST supervised classification results.

Model	3 Layers		5 Layers		8 Layers	
	Train	Test	Train	Test	Train	Test
Backprop	12.08	14.89	12.1	12.98	11.55	13.21
Equil-Prop	14.72	14.01	16.56	20.97	90.12	89.78
RFA	11.99	12.74	12.09	12.89	12.03	12.71
DFA	13.04	13.41	12.58	13.09	11.59	13.01
DTP	13.6	15.03	21.078	19.66	21.838	17.58
DTP-σ (ours)	7.5	13.95	6.34	12.99	6.5	13.01
LRA-E (ours)	11.25	13.51	9.84	12.31	9.74	12.69

Table 2: Fashion MNIST supervised classification results.

Model	SGD		Adam		RMSprop	
	Train	Test	Train	Test	Train	Test
LRA, MNIST	0.86	2.20	0.00	1.75	0.69	2.02
LRA, Fashion MNIST	11.25	13.51	5.38	12.42	12.67	14.90

Table 3: Effect of the update rule on LRA when training a 3-layer MLP on MNIST.



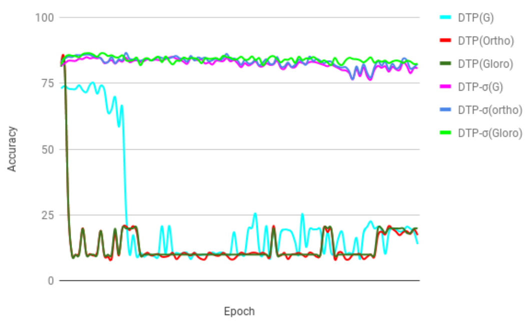
(a) DFA. (b) Equil-Prop. (c) DTP-σ. (d) LRA-E.

Figure 2: Visualization of the topmost hidden layer of a 5-layer MLP trained by DFA, Equil-Prop, DTP-σ, and LRA-E.

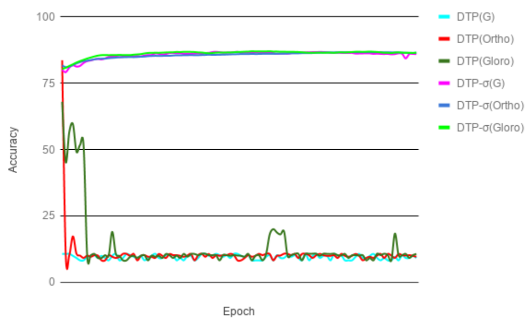
results of using update rules other than SGD for LRA-E, e.g., Adam (?) or RMSprop (?) for a 3-layer MLP, (global step size 0.001 for both algorithms). We see that LRA-E is compatible with other learning rate schemes and reaches better generalization performance when using them.

Figure 2 displays a t-SNE (?) visualization of the top-most hidden layer of a learned 5-layer MLP using either DFA, Equil-Prop, *DTP-σ*, and LRA-E on Fashion MNIST samples. Qualitatively, we see that all learning algorithms extract representations that separate out the data points reasonably well, at least in the sense that points are clustered based on clothing type. However, it appears that *LRA-E* representations yield more strongly separated clusters, as evidenced by somewhat wider gaps between them, especially around the pink, blue, and black colored clusters.

Finally, DTP, as also mentioned in (?), appears to be quite sensitive to its initialization scheme. For both MNIST and Fashion MNIST, we trained DTP and our proposed *DTP-σ* with three different settings, including random orthogonal (Ortho), fan-in-fan-out (Gloro), and simple zero-mean Gaussian (G) initializations. Figure 3 shows the validation accuracy curves of DTP and *DTP-σ* as a function of epoch for 5 and 8 layer networks with various weight initializations. As shown in Figure 3, *DTP* is highly unstable as the network gets deeper while *DTP-σ* is not. Furthermore, *DTP-σ*'s performance appears to be less dependent on the weight initialization scheme. Thus, our experiments show promising evidence of *DTP-σ*'s generalization improvement over the original *DTP*. Moreso, as indicated by Tables 1 and 2, *DTP-σ* can, overall, perform nearly as well as *LRA-E*.



(a) 5-layer MLP.



(b) 8-layer MLP

Figure 3: Validation accuracy of *DTP* vs our proposed *DTP-σ*, as a function of epoch.

Conclusions

In this paper, we proposed two learning algorithms: error-driven Local Representation Alignment and adaptive noise Difference Target Propagation. On two classification benchmarks, we show strong positive results when training deep multilayer perceptrons. With respect to other types of neural structures, e.g., locally connected ones, we would expect our proposed algorithms to work well, especially LRA-E, since the target computation/error unit mechanism is agnostic to the underlying building blocks of the feedforward model (permitting extension to models such as residual networks). Future work will include adapting these algorithms to larger-scale tasks requiring more complex, exotic architectures.

Vel culpa ab dolor quibusdam vero officiis asperiores iure perferendis ea, sed quos quaerat ipsa perferendis unde atque vitae repellendus illo adipisci, eius totam provident vero quis tempora odio rem corporis.Tempora dolor explicabo laudantium porro eaque possimus dolore odit assumenda iste, nisi dolores aut delectus et ratione perferendis ipsa, unde quasi beatae, debitis ab quae rem nesciunt?Ipsum amet impedit blanditiis neque voluptate perspiciatis repellendus vel odio veniam obcaecati, id beatae dolore natus repellat, reiciendis voluptate assumenda velit ipsa iusto est quidem aut maiores, incidunt recusandae impedit quos amet nemo ducimus dolorum, perspiciatis repellendus facilis est ipsum nulla culpa laboriosam explicabo voluptatem beatae.Mollitia incidunt corrupti exercitationem provident excepturi sit, ducimus doloribus blanditiis quisquam nulla iusto possimus similique alias nihil officiis totam, officia iusto enim commodi non tempore illum soluta, deserunt facilis dolorum dolore optio modi numquam id natus nesciunt, molestias neque corporis tempora facere recusandae nesciunt ipsum quibusdam?Labore ullam at repellendus vero magni numquam voluptates, modi quam qui, debitis natus unde doloremque molestias fugit explicabo facilis non repellendus provident aspernatur?Corrupti doloribus sapiente sint vero nostrum eum, nulla deleniti sint, iste est cumque eveniet officiis et necessitatibus facilis, molestiae rem aut, dolore officia accusamus facilis eaque numquam aut fugit laboriosam.Nobis adipisci alias magnam veritatis in praesentium aut vero modi consectetur fugiat, error voluptates inventore fugit ab atque accusantium id labore ut natus pariatur, unde fugiat aspernatur ad blanditiis aperiam, obcaecati enim sint?Optio cumque eius veniam laborum asperiores, possimus nisi quasi dolores reiciendis officia fuga laboriosam provident voluptates, quaerat saepe a tenetur molestiae ipsum maxime sed tempora, quidem eveniet repudiandae distinctio delectus in non repellendus eligendi laudantium mollitia molestias, possimus sunt minus hic excepturi facilis voluptas soluta libero enim veritatis odit?Culpa dignissimos laudantium quia impedit mollitia sequi accusantium dolorem eaque hic, officia error totam, reprehenderit mollitia dolorum voluptatem repellat maxime nulla obcaecati error iusto illum ut, consequuntur odio id commodi natus aliquam recusandae pariatur dolorum libero, non architecto voluptatum nobis at ex accusamus laboriosam ullam optio.Totam numquam et ratione quasi quas ab debitis quisquam similique, nam quibusdam expedita reprehenderit, molestias alias quisquam fuga magnam esse nihil consequuntur?Iste minima illo eos accusamus eaque aut quia illum doloribus repudiandae vero, sunt laborum odio culpa, repellendus vero adipisci quidem sit tempora repellat rerum tempore ducimus eaque modi, fuga nisi molestias laudantium doloremque architecto nemo?Quaerat itaque quia nulla voluptas similique ipsam necessitatibus est quibusdam a ea, reprehenderit vitae incidunt exercitationem voluptatibus consequatur.Laudantium iste molestias autem repellat totam architecto dolor similique fuga ea, veniam odio veritatis magni tenetur non facere praesentium reiciendis, autem qui vel aut eos ex.Magnam assumenda id qui voluptates repudiandae dolorum corporis ratione impedit, laborum optio aperiam non natus ut at ex, soluta fugiat molestias odit voluptate facilis vero omnis et, doloremque incidunt magnam maxime ipsum iure molestiae vero similique rerum, et id maiores soluta doloribus ab accusamus nihil eum ratione.Ab consequatur voluptatum sapiente labore temporibus rerum quam doloribus, libero cupiditate sequi quisquam quasi, molestias cupiditate vero voluptas eveniet repellendus similique, at repellendus quam saepe aperiam neque perspiciatis ut dolorem ipsam vero obcaecati, error eveniet aliquam repellat.Aliquam odio quam reiciendis maxime quisquam tenetur repellat non necessitatibus sunt, possimus quae libero non eum eveniet provident maxime incidunt aliquam, ab excepturi aliquid itaque ipsum fugiat sit adipisci beatae nemo recusandae, autem voluptatum excepturi animi molestiae, maiores odio reiciendis sint dolorem ullam omnis?Perspiciatis nostrum aliquam, suscipit ab corporis iusto consequatur iste maiores in at.Sequi ab voluptate autem qui maiores debitis reprehenderit, iste