A BIOLOGICALLY-PLAUSIBLE ALTERNATIVE TO BACKPROPAGATION USING RECIPROCAL FEEDBACK CONNECTIONS

Mia Cameron
University of California, San Diego
La Jolla, CA 92093, USA
mcameron@ucsd.edu

Yusi Chen
Computational Neuroscience Center
University of Washington
Seattle, WA 98195, USA

Terrence Sejnowski
Computational Neurobiology Laboratory
Salk Institute for Biological Studies
La Jolla, CA 92037, USA

ABSTRACT

Artificial neural networks are frequently used to model neural systems, and have been shown to recapitulate features of biological networks, including in the hippocampus (Chen et al., 2024) and visual cortex (Yamins et al., 2014) (Khaligh-Razavi & Kriegeskorte, 2014). However, despite it's success in neural modelling, the predominant learning algorithm, backpropagation, has long been considered biologically implausible (Crick, 1989). One reason is the weight-transport problem - or the problem of how a global error signal can be accurately transmitted across the network to minimize the error resulting from each layer's parameters. Previous solutions to the weight-transport problem have proposed using a separate, error-feedback network to transport an error signal across layers (Lillicrap et al., 2020). However, more recent experimental work indicates that cortical feedback connections are more likely to be involved in reconstructing lower-level activity based on activity from higher layers, rather than being exclusively used to transmit top-level error (Mumford, 1992) (Favila et al., 2022) (Linde-Domingo et al., 2019). Under the predictive coding framework, the bidirectional, cortical processing hierarchy is more appropriately modeled as a multi-layered autoencoder (Marino, 2020). Here, we attempt to unify these two views by showing how autoencoder-like, inverse feedback connections may be used to minimize top-level error in neural networks. Our proposed mechanism, *Reciprocal Feedback*, consists of two contributions: first we show how a modification of the Recirculation autoencoder algorithm (Hinton E. & McClelland, 1988) is equivalent to learning the Moore-Penrose pseudoinverse. Then, we will show how, using a Newton-like method (Hildebrandt & Graves, 1927), locally-learned pseudoinverse feedback connections may be used to facilitate an alternative, more biologically-realistic optimization method to gradient descent, by relying on the reciprocal of the forward network rather than its gradient. Overall, we provide a mathematical framework for understanding how the hierarchical, autoencoder-like feedback connections observed in the layers of the cortex may additionally be used as a mechanism for minimizing a global error signal, using only local activity.

1 INTRODUCTION

The backpropagation algorithm is commonly regarded as the standard optimization algorithm for neural networks in machine learning (Rumelhart et al., 1986). Furthermore, artificial neural networks trained using the backpropagation algorithm have been shown to recapitulate features of biological networks, such as in the hippocampus (Chen et al., 2024) and visual cortex (Yamins et al., 2014). However, there are several aspects of the backpropagation algorithm which make it an unlikely candidate to be physically implemented in the brain (Crick, 1989) - one of these being the weight-transport problem (Grossberg, 1987).

During the computation of the gradient in the backpropagation algorithm, the error at the final layer is passed "backwards" through the transpose of the each layer weight matrix, in order to compute the error with respect to each layer parameter. While computing the transpose of each weight matrix is fast and simple in-silico, there is no known biological mechanism by which error signals may be precisely propagated backwards in biological neurons at a timescale relevant to learning.

Many algorithms which attempt to circumvent the weight transport problem and approximate backpropagation in a biologically-plausible way have been proposed. The most successful in machine learning tasks have employed a separate error network of feedback connections to transmit final-layer error backwards, in a similar procedure to backpropagation (Lillicrap et al., 2014). Some of these feedback methods, including the weight-mirror (Akrout et al., 2019) and sign-symmetry (Xiao et al., 2018) method, have been shown to have the capacity to scale to large, complex datasets.

However, empirical evidence does not suggest that the physical bidirectional feedback connections which exist in the cortex are exclusively used to transmit error from the highest layer. Instead, it is likely that cortical feedback connections may be used to reconstruct activity patterns at lower layers using information from higher layers, and vice versa (Mumford, 1992). Recent experimental work also supports the hypothesis that areas lower in the cortical processing hierarchy reconstruct activity patterns based on information from higher areas, particularly during memory tasks (Linde-Domingo et al. (2019), Favila et al. (2022), Naya et al. (2001)). This framework of predictive coding suggests that the cortical processing hierarchy may be more appropriately modelled by a series of stacked, autoencoder-like structures (Marino, 2020).

In this paper, we attempt to unify the experimentally-supported theory of the cortex as a series of autoencoders, with the error feedback network models which have shown to be successful in emulating backpropagation and solving machine learning problems. **More specifically, we show how autoencoder-like, pseudoinverse feedback connections can be used to globally minimize error in a fully-connected, feedforward network.**

First, we will show how a simple modification of the Recirculation algorithm is capable of training a pair of matrices to be the unique pseudoinverses of each other. Then, we will show that using only the pseudoinverse at each layer, it is possible to propagate the error backwards through the network, and adjust each layer weight in a direction that reduces the global error. For that derivation, we will apply the Newton-like method described by Hildebrandt & Graves (1927) and Ben-Israel (1966) as an alternative to gradient descent. Physically, these two phases may be synchronized in a "wake-sleep" cycle (Figure 2). During the wake phase, predictions are generated from inputs, compared with targets, and forward weights are modified to minimize the error between them. During the "sleep" phase, random noise is generated at each layer, so that the forward and feedback matrices align to be the pseudoinverses of each other (Figure 1). We also show that this method outperforms the Random Feedback Alignment algorithm on the MNIST and CIFAR-10 image classification tasks.

2 RELATED WORK

Our work is closely related to the Target Propagation algorithm (Bengio, 2014) (Lee et al., 2015), in the sense that each layer can be thought of as an autoencoder. However, they adjust each layer's parameters through an auxiliary "target" activation at each layer, which is generated by propagating the final-layer target activation backwards through each autoencoder layer. Our method, on the other hand, uses the final-layer error directly to adjust parameter weights.

Another biologically-plausible algorithm which uses random noise during a "sleep" period to learn appropriate feedback weights is the weight-mirror algorithm Akrout et al. (2019). In that case, they

trained the feedback weights to approximate the transpose of the forward weights as opposed to the pseudoinverse. This allows for a better direct approximation to backpropagation, but does not align with the autoencoder-like structure of feedback connections that would be expected.

3 LOCAL LEARNING OF THE PSEUDOINVERSE AT EACH LAYER

We will first show that a linear modification of the Recirculation algorithm is capable of learning a pair of pseudoinverse matrices when its inputs are random, mean-zero noise. For generality, we will denote the forward (input-to-hidden) weight matrix as $\boldsymbol{V}$, and backward (hidden-to-input) weight matrix as $\boldsymbol{U}$. Furthermore, we will denote the random input layer vectors as $\boldsymbol{y}$, and hidden layer vectors as $\boldsymbol{h}$. We will also assume the random input layer vector $\boldsymbol{y}$ has a mean of 0, and variance of σ^2.

A theorem described in Ben-Israel & Cohen (1966) defines an iterative matrix computation for the pseudoinverse for an arbitrary matrix $\boldsymbol{A}$. Starting from a matrix $\boldsymbol{X}_0$ which satisfies $(\boldsymbol{X}_0\boldsymbol{A})^T = \boldsymbol{X}_0\boldsymbol{A}$, a sequence is generated by:

$$\boldsymbol{X}_{t+1} - \boldsymbol{X}_t = \boldsymbol{X}_t - \boldsymbol{X}_t\boldsymbol{A}\boldsymbol{X}_t$$

where $\boldsymbol{X}_t$ converges to $\boldsymbol{A}^+$.

The pseudoinverse iteration above can be rewritten using the forward and feedback matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, and the fact that $\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \sigma^2\boldsymbol{I}$ (derivation in section A.2 of the Appendix).

$$\begin{aligned}\Delta\boldsymbol{U} &= \boldsymbol{U} - \boldsymbol{U}\boldsymbol{V}\boldsymbol{U} \\ &= \frac{1}{\sigma^2}(\boldsymbol{U} - \boldsymbol{U}\boldsymbol{V}\boldsymbol{U})\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] \\ &= \frac{1}{\sigma^2}\mathbb{E}[(\boldsymbol{U}\boldsymbol{y} - \boldsymbol{U}\boldsymbol{V}\boldsymbol{U}\boldsymbol{y})\boldsymbol{y}^T]\end{aligned}$$

Now, using similar dynamics to the Recirculation algorithm (though omitting the sigmoid nonlinearity operator), we define $\boldsymbol{h}$, $\hat{\boldsymbol{y}}$, $\hat{\boldsymbol{h}}$ by:

$$\begin{aligned}\boldsymbol{h} &= \boldsymbol{U}\boldsymbol{y} \\ \hat{\boldsymbol{y}} &= \boldsymbol{V}\boldsymbol{h} \\ \hat{\boldsymbol{h}} &= \boldsymbol{U}\hat{\boldsymbol{y}}\end{aligned}$$

Altogether, the $\Delta\boldsymbol{U}$ above can be written in Recirculation dynamics as:

$$\Delta\boldsymbol{U} = \frac{1}{\sigma^2}\mathbb{E}[(\boldsymbol{h} - \hat{\boldsymbol{h}})\boldsymbol{y}^T]$$

So, if the input patterns $\boldsymbol{y}$ have a mean of 0, $\boldsymbol{U}$ will converge to the pseudoinverse of $\boldsymbol{V}$ when averaged over many inputs. Likewise, the same procedure on $\boldsymbol{V}$ will converge to the pseudoinverse of $\boldsymbol{U}$ (Figure 1).

Running this "sleep" phase algorithm on both U and V results in convergence of weights towards the pseudoinverse of each other, even with random initialization conditions. (implementation details in section A.3)

4 TRAINING THE WHOLE MULTI-LAYER NETWORK

The network architecture we will consider in this paper is a fully-connected, feedforward network consisting of a series of hidden layers, connected by forward weight matrices ($\boldsymbol{W}_l$), and feedback weight matrices ($\boldsymbol{W}_l{}^+$).

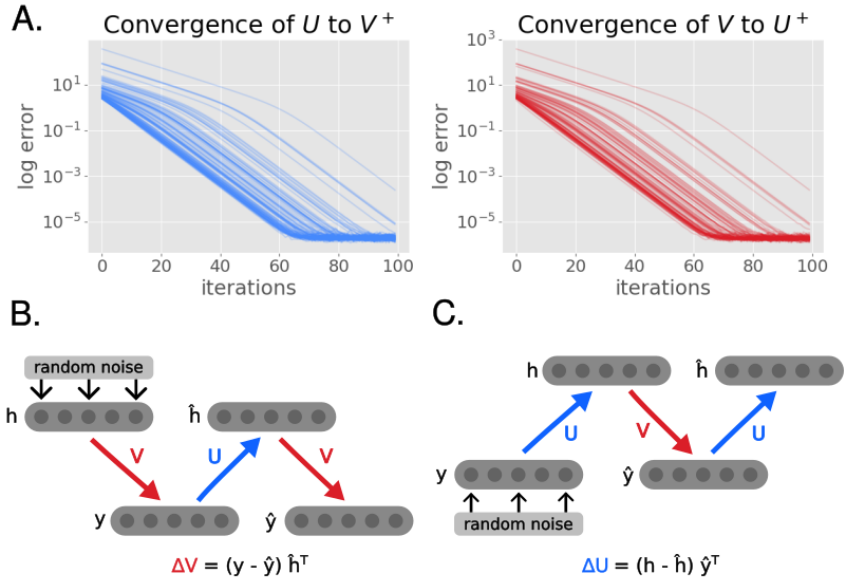

Figure 1: Using a local learning rule, the matrices U and V align to become the pseudoinverse of each other. A. The Frobenius norm of the difference between the pseudoinverse of U (derived through its SVD) and V, and vice versa, over 100 random initializations. B. and C. are figures adapted from Hinton E. & McClelland (1988), showing the Recirculation learning procedure with random noise inputs.

As in a standard feedforward network, each layer consists of an application of matrix-multiplication to the previous hidden layer activation vector, followed by the application of a bijective, nonlinear activation function ($\sigma(\cdot)$) on each element. Mathematically, for a network consisting of L layers, the pre-activation vector ($\boldsymbol{a}_l$) and activation vector ($\boldsymbol{h}_l$) are defined by:

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{x} \\ \boldsymbol{a}_l &= \boldsymbol{W}_l \boldsymbol{h}_{l-1} \\ \boldsymbol{h}_l &= \sigma(\boldsymbol{a}_l) \end{aligned}$$

for each layer $l \leq L$.

We will define the scalar error function $\mathcal{E}$ as the mean squared value of the difference between the final-layer output ($\boldsymbol{h}_L$), and target output ($\boldsymbol{h}_L^*$).

$$\mathcal{E} = \frac{1}{2} ||\boldsymbol{h}_L - \boldsymbol{h}_L^*||_2^2$$

We will also define the error vector, denoted $\boldsymbol{e}$, as the difference between the final layer activation ($\boldsymbol{h}_L$) and the final layer target ($\boldsymbol{h}_L^*$):

$$\boldsymbol{e} = \boldsymbol{h}_L - \boldsymbol{h}_L{}^*$$

During gradient descent, the gradient of $\mathcal{E}$ with respect to each parameter $\boldsymbol{W}_l$ approaches 0: $(\boldsymbol{J}_{\boldsymbol{W}_l}^{\mathcal{E}})^T \boldsymbol{e}(\boldsymbol{x}, \boldsymbol{W}_l) \to 0$. By optimizing $\boldsymbol{e}$ directly, such that $\boldsymbol{e}(x, W_l) \to 0$, the gradient with respect to $\mathcal{E}$ will also approach 0. In essence, minimizing the error vector $\boldsymbol{e}$ minimizes the scalar error $\mathcal{E}$.

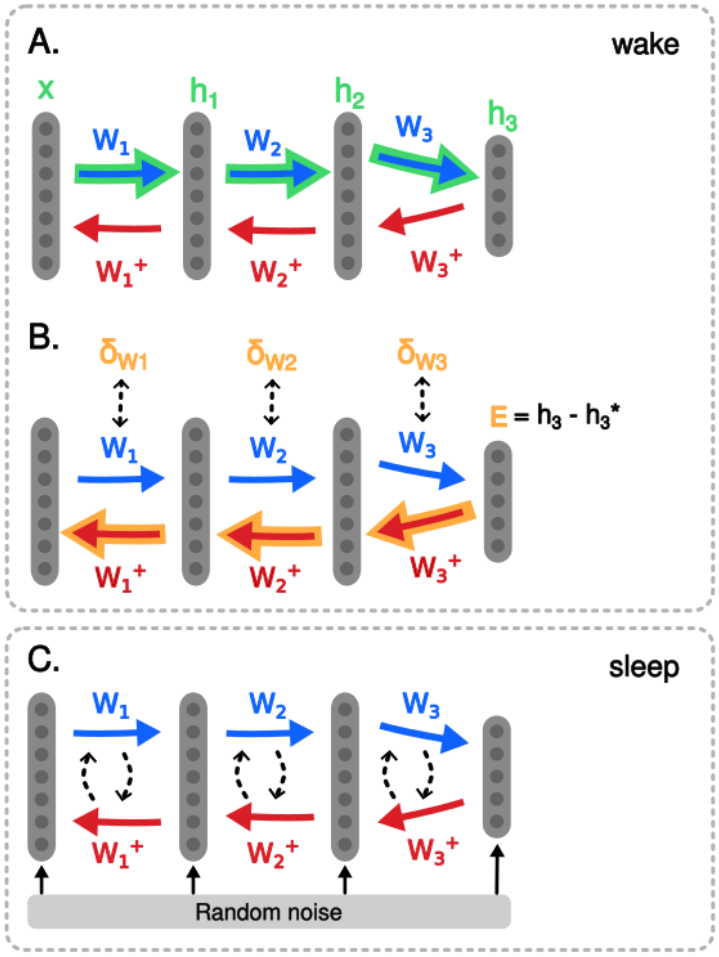

Figure 2: Illustration of the three phases of network operation in a simple 4-layer network. A. Forward propagation of input x, represented by green arrows. B. Backwards propagation of the error vector (represented by the orange arrows) through the feedback matrices. C. "Sleep" phase, where forward and feedback matrices are trained to be the pseudoinverse of each other. Part C. of Figure 1 illustrates this section in greater detail.

Notation and Preliminaries

Following the notation defined in Magnus & Neudecker (2019), the Jacobian of a vector $\boldsymbol{y}$ with respect to a matrix $\boldsymbol{X}$ is:

$$\boldsymbol{J}_X^y = \frac{\partial \boldsymbol{y}}{\partial (\text{vec } \boldsymbol{X})^T}$$

It is also relevant to note that the chain rule for Jacobians is different to the chain rule for gradients. For functions $Y(X)$ and $X(Z)$, the chain rule for Jacobians is:

$$\boldsymbol{J}_Z^Y = \boldsymbol{J}_X^Y \boldsymbol{J}_Z^X$$

(which is in the reverse order from the chain rule for gradients).

We will also assume that the vector function $\boldsymbol{e}$ is in the class $C'(X_0)$, meaning that has a differential at every point $x \in X_0$ and that differential is continuous. So, for each parameter $\boldsymbol{W}_l$, there exists a differential with Jacobian matrix $\boldsymbol{J}_{W_l}^E$ such that

$$E(\boldsymbol{x}, \boldsymbol{W}_l^1) - E(\boldsymbol{x}, \boldsymbol{W}_l^0) - \boldsymbol{J}_{W_l}^E (\text{vec } \boldsymbol{W}_l^1 - \text{vec } \boldsymbol{W}_l^0) = r(\text{vec } \boldsymbol{W}_0) ||\text{vec } \boldsymbol{W}_l^1 - \text{vec } \boldsymbol{W}_l^0||$$

where r is a function in which

$$\lim_{\text{vec } \boldsymbol{W}_l^1 \to \text{vec } \boldsymbol{W}_l^0} ||r(\text{vec } \boldsymbol{W}_0)|| = 0$$

A Newton-like method described in Hildebrandt & Graves (1927) and Ben-Israel (1966) uses the left multiplicative reciprocal of the Jacobian of a function in order to minimize it. Unlike gradient descent, it does not require computing the transpose of the Jacobian - making it potentially a more biologically-plausible method for global error minimization throughout the network. Briefly, it can be stated as follows:

Theorem 1 (Hildebrandt-Graves) *If X_0 is a subset of $\mathbb{R}^n$, $\boldsymbol{y}_0 \in \mathbb{R}^k$ is a vector, $B_r(\boldsymbol{y}_0)$ is a ball of radius r centered at $\boldsymbol{y}_0$, and $F : (X_0, B_r(\boldsymbol{y}_0)) \to \mathbb{R}^m$ is a vector function such that:*

1. *There exists a matrix $\boldsymbol{A} : B_r(y_0) \to \mathbb{R}^k$, with left reciprocal $\boldsymbol{T}$ and positive number M such that:*

$$\begin{gathered} \boldsymbol{T}\boldsymbol{A}\boldsymbol{y} = \boldsymbol{y} \\ M||\boldsymbol{T}|| < 1 \\ ||\boldsymbol{A}(\boldsymbol{y}_1 - \boldsymbol{y}_2) - F(\boldsymbol{x}, \boldsymbol{y}_1) + F(\boldsymbol{x}, \boldsymbol{y}_2)|| \leq M||\boldsymbol{y}_1 - \boldsymbol{y}_2|| \end{gathered}$$

   for all $\boldsymbol{x} \in X_0$ and every $\boldsymbol{y} \in B_r(y_0)$ where $\boldsymbol{y} \in R(\boldsymbol{T})$.

2. *the radius r satisfies:*

$$||\boldsymbol{T}||||F(\boldsymbol{x}, \boldsymbol{y}_0)|| < r(1 - M||\boldsymbol{T}||)$$

   for all $\boldsymbol{x} \in X_0$

Then there exists a solution $\boldsymbol{y}^*$ which for every $\boldsymbol{x} \in X_0$ satisfies

$$\boldsymbol{T}F(\boldsymbol{x}, \boldsymbol{y}^*) = 0$$

which can be obtained using the iteration:

$$\boldsymbol{y}_{t+1} = \boldsymbol{y}_t - \boldsymbol{T}F(\boldsymbol{x}, \boldsymbol{y}_t)$$

A proof of this theorem, based on that of Ben-Israel (1966), can be found in section A.1.

In the context of an artificial neural network, we can consider X_0 to be the set of n-dimensional input vectors, $\boldsymbol{y}$ to be the parameters of the network (in vector form), and F to be the function mapping each input and parameter pair to a vector (in our case, the error vector resulting from each input-target pair). $\boldsymbol{y}_0$ would be initial value of the parameter, and the solution $\boldsymbol{y}^*$ is the final value of the parameter at the end of that iteration. Since $\boldsymbol{y}$ is a vector, and $\boldsymbol{W}_l$ are each matrices, we will use a vectorized version of each weight matrix for our derivation: vec $\boldsymbol{W}_l$.

4.1 Deriving the Jacobian at each layer

First, we will derive the Jacobian matrix, $\boldsymbol{J}_{W_l}^E$, of the error vector with respect to each parameter $\boldsymbol{W}_l$ in terms of the forward weight matrices and activation vectors.

Using the chain rule described above, the Jacobian matrix of each layer parameter is given by:

$$\begin{aligned} [J_{W_l}^E]_{ij} &= \frac{\partial E}{\partial W_{ij}^l} \\ &= \sum_k \frac{\partial h_k^l}{\partial W_{ij}} \frac{\partial E}{\partial h_k^l} \\ &= a_j^{l-1} \frac{\partial E}{\partial h_i^l} \end{aligned}$$

More compactly, in matrix notation:

$$\begin{aligned}\boldsymbol{J}_{W_l}^E &= \boldsymbol{J}_{W_l}^{h_l} \boldsymbol{J}_{h_l}^E \\ &= (\boldsymbol{a}_{l-1} \otimes \boldsymbol{J}_{h_l}^E)\end{aligned}$$

Next, we need to find a recursive expression to compute $\boldsymbol{J}_{h_l}^E$ at each layer. (similarly to how the backpropagation algorithm uses a recursive expression to compute the gradient $\nabla_{W_l} \boldsymbol{e}$). Using the chain rule again:

$$\begin{aligned}\boldsymbol{J}_{h_l}^E &= \boldsymbol{J}_{h_{l+1}}^E \boldsymbol{J}_{a_{l+1}}^{h_{l+1}} J_{h_l}^{a_{l+1}} \\ &= \boldsymbol{J}_{h_{l+1}}^E \mathcal{D}_\sigma \boldsymbol{W}_{l+1}\end{aligned}$$

where $\mathcal{D}_\sigma$ is a diagonal matrix representing the derivative of the elementwise, nonlinear operator $\sigma(\cdot)$.

All of the derivations so far are the same as those used in backpropagation, but transposed (since we are computing the Jacobian rather than the gradient matrix). Now, we have a recursive expression for the Jacobian of each forward matrix parameter, $\boldsymbol{W}_l$.

4.2 Finding reciprocals of the Jacobian

In order to apply the Hildebrandt-Graves Theorem to optimize the parameters $\boldsymbol{W}_l$, we need to find left reciprocal matrices $\boldsymbol{\mathcal{B}}_l$, such that for each parameter, $\boldsymbol{\mathcal{B}}_l \boldsymbol{J}_{W_l}^E \boldsymbol{y} = \boldsymbol{y}$.

To aid in this derivation, we will also define the left reciprocal of the activation Jacobian ($\boldsymbol{J}_{h_l}^E$) as $\boldsymbol{B}_l$.

To begin finding a suitable reciprocal, we will assume that we have access to the Moore-Penrose pseudoinverse of each forward weight (as we've shown is obtainable with a modification of the Recirculation algorithm). Following standard notation, we will denote the unique Moore-Penrose pseudoinverse of each layer weight matrix as $\boldsymbol{W}_L^+$. Physically, would be as a feedback connection between adjacent layers.

First, we will find a left reciprocal of the activation Jacobian, $\boldsymbol{J}_{h_l}^E$. Like the Jacobian itself, the reciprocal $\boldsymbol{B}_l$ can be computed recursively, using the pseudoinverses of each layer weight matrix:

$$\begin{aligned}\boldsymbol{B}_l &= (\mathcal{D}_\sigma \boldsymbol{W}_{l+1})^+ \boldsymbol{B}_{l+1} \\ &= \boldsymbol{W}_{l+1}^+ \mathcal{D}_\sigma^+ \boldsymbol{B}_{l+1}\end{aligned}$$

By checking each of the four Moore-Penrose conditions (Penrose, 1955), it can be shown that this recursion on $\boldsymbol{B}_l$ preserves the pseudoinverse properties $(1, 2, 3)$. Furthermore, when every $\boldsymbol{W}_l$ is full row-rank, $\boldsymbol{B}_l$ acts as a multiplicative left reciprocal (Tian, 2020). And, if we initialize each $\boldsymbol{W}_l$ with random weights, it is almost certainly going to be a full-rank matrix.

Using the activation reciprocal, $\boldsymbol{B}_l$, the reciprocal of the layer parameter, $\boldsymbol{\mathcal{B}}_l$, can be defined with:

$$\boldsymbol{\mathcal{B}}_l = \left(\frac{1}{||\boldsymbol{a}||_2^2} \boldsymbol{a}_{l-1}^T \otimes \boldsymbol{B}_l \right)$$

So, $\boldsymbol{\mathcal{B}}_l$ has the property that:

$$\begin{aligned}\boldsymbol{\mathcal{B}}_l \boldsymbol{J}_{W_l}^E &= \left(\frac{1}{||\boldsymbol{a}||_2^2} \boldsymbol{a}_{l-1}^T \otimes \boldsymbol{B}_l \right) \left(\boldsymbol{a}_{l-1} \otimes \boldsymbol{J}_{h_l}^E \right) \\ &= \boldsymbol{B}_l \boldsymbol{J}_{h_l}^E\end{aligned}$$

This would imply that for all $\boldsymbol{y}$ such that $\boldsymbol{B}_l \boldsymbol{J}_{h_l}^E \boldsymbol{y} = \boldsymbol{y}$, then $\boldsymbol{\mathcal{B}}_l \boldsymbol{J}_{W_l}^E \boldsymbol{y} = \boldsymbol{y}$.

Therefore, using the Jacobian $\boldsymbol{J}_{W_l}^E$ as the matrix $\boldsymbol{A}$, and the reciprocal $\boldsymbol{\mathcal{B}}_l$ as its reciprocal $\boldsymbol{T}$, Theorem 1 allows an update direction to be computed for $\boldsymbol{W}_l$ at each layer which minimizes the error vector.

Specifically, the weight update direction $\delta_{W_l}^t$ is given by:

$$\begin{aligned}\delta_{W_l}^t &= \boldsymbol{\mathcal{B}}_l \boldsymbol{e}(x, W_l) \\ &= \left(\frac{1}{||\boldsymbol{a}||_2^2}\boldsymbol{a}_{l-1}^T \otimes \boldsymbol{B}_l\right)\boldsymbol{e}(x, W_l) \\ &= \frac{1}{||\boldsymbol{a}||_2^2}\boldsymbol{B}_l \boldsymbol{e}(x, W_l)\boldsymbol{a}_{l-1}^T\end{aligned}$$

And during each step of learning,

$$\boldsymbol{W}_l^{t+1} = \boldsymbol{W}_l^t - \lambda \delta_{W_l}^t$$

Where λ is the manually determined learning rate.

In conclusion, Theorem 1 allows us to minimize the error vector $\boldsymbol{e}$ with respect to each parameter $\boldsymbol{W}_l$. Unlike backpropagation, we only require the pseudoinverse of each forward weight matrix, rather than its exact transpose, making it potentially more biologically-plausible by circumventing the weight transpose problem.

However, to restate some important assumptions we have made, in order for Theorem 1 to minimize the error of the network:

1. The condition number of the Jacobian and reciprocal must be kept low, so that condition 1 can be satisfied.
2. Each weight matrix W_l must be full-row rank for the matrix $\mathcal{B}_l$, as defined above, to be its left reciprocal. However, if each weight matrix is initialized from a random distribution, it is almost certainly going to be full-rank.
3. The nonlinear operator σ must be bijective and continous. For instance, this could be a leaky softplus activation.

Algorithm 1

for $((\boldsymbol{x}, \boldsymbol{h}_L^*) \in$ batch **do**
  $\boldsymbol{W}_1^+, \ldots, \boldsymbol{W}_L^+ \leftarrow$ Update Pseudoinverse ▷ (sleep)
  $\boldsymbol{h}_0 \leftarrow \boldsymbol{x}$
  for $l \in L$ **do** ▷ Forward pass (wake)
    $\boldsymbol{a}_l \leftarrow \boldsymbol{W}_l \boldsymbol{h}_{l-1}$
    $\boldsymbol{h}_l \leftarrow \sigma(\boldsymbol{a}_l)$
  end for
  $\boldsymbol{e} \leftarrow \boldsymbol{h}_L - h_L^*$
  for $l \in L$ **do** ▷ Backward pass (wake)
    $\boldsymbol{B}_l \leftarrow \boldsymbol{W}_l^+ \boldsymbol{\mathcal{D}}_{\boldsymbol{\sigma}}^+ \boldsymbol{B}_{l+1}$
    $\delta_{W_l} \leftarrow (\boldsymbol{B}_l \boldsymbol{e})\boldsymbol{a}_{l-1}^T$
  end for
  $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_L \leftarrow \boldsymbol{W}_1 - \lambda\delta_{W_1}, \ldots, \boldsymbol{W}_L - \lambda\delta_{W_L}$
end for

5 COMPUTATIONAL EXPERIMENTS

This algorithm was tested on two classification problems - MNIST and CIFAR-10. For both datasets, we used the cross-entropy loss function, and SGD with a constant learning rate of 0.001. However, we used a noncontinuous, leaky ReLU nonlinearity in order to reduce computational costs. Training

was done on a RTX 3060 GPU, using the PyTorch library (Paszke et al., 2019). The "random" algorithm is the feedback alignment algorithm as described in Lillicrap et al. (2014), which uses feedback connections fixed with random weights to transmit error backwards through the network. Code is available in the supplementary material.

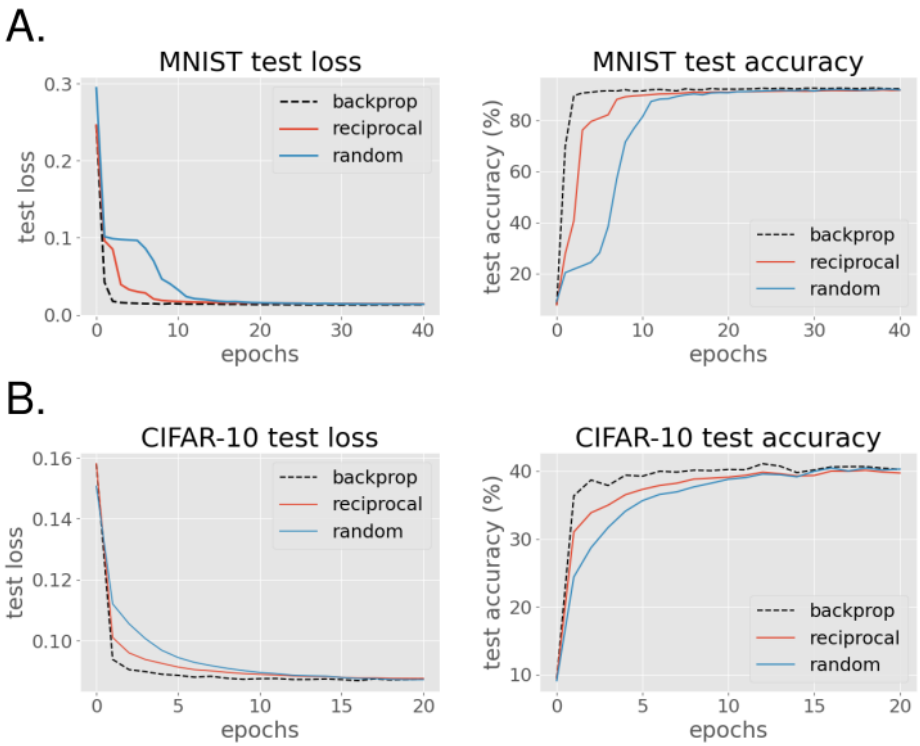

Figure 3: Reciprocal feedback learning rule compared to backpropagation and random feedback, on MNIST digit classification and CIFAR-10 image classification datasets.

5.1 MNIST Classification

The MNIST classifier had 5 hidden layers, and an architecture of $(28 \times 28) - 400 - 200 - 100 - 50 - 10$. It was run with a learning rate of 0.001, batch size of 40, for 40 epochs.

5.2 CIFAR-10 Classification

The CIFAR-10 classifier has 4 hidden layers: $((32 \times 32 \times 3) - 1000 - 500 - 100 - 10$, and was run with a learning rate of 0.001 and batch size of 40 for 20 epochs. However, CIFAR-10 is generally not solved well with a fully-connected, feedforward network, and requires a locally-connected architecture to achieve high accuracy (Lee et al., 2015).

	MNIST		CIFAR-10	
Algorithm	**Accuracy (%)**	**Epochs**	**Accuracy (%)**	**Epochs**
Backprop	92.35	2	40.23	1
Reciprocal	91.78	7	39.72	4
Random	91.92	11	40.29	6

Table 1: Final classification accuracy and number of epochs required to reach 90% of the asymptotic error for each algorithm

6 DISCUSSION

We have shown how locally-learned pseudoinverse feedback connections can be used to train a feedforward, fully-connected neural network, using an alternative optimization method to gradient descent. With our method, we alleviate the need to compute the weight transpose at each layer - suggesting a possible solution to the weight transport problem of the backpropagation algorithm. Furthermore, the use of pseudoinverse feedback connections may better align with the evidence-based, neuroscientific model of the cortex as a series of autoencoder-like structures.

To apply Theorem 1 to a neural network, any suitable left multiplicative reciprocal of the Jacobian may be used, not necessarily the $\mathcal{B}$ recursion based on the Moore-Penrose pseudoinverse. Single-layer, autoencoder-like algorithms other than Recirculation may also be able to learn a suitable generalized inverse at each layer. For instance, Tapson & van Schaik (2013) describes an online, biologically-plausible algorithm for computing the pseudoinverse of a network's weights. Furthermore, given the rank-one, outer-product structure of forward matrix updates, it may be possible to make local, online updates to the pseudoinverse feedback weights using the Sherman-Morrison formula.

This method may also be related to Gauss-Newton optimization for neural networks, which uses the direct pseudoinverse of the whole network Jacobian at each layer (Botev et al., 2017). The main issue with implementing this using only layer-wise pseudoinverses is that unique, direct pseudoinverse of the whole network cannot be composed sequentially using each layer's pseudoinverse. In other words, since it is an approximation of the second-derivative, the chain rule for Hessians must be used instead, followed by taking the pseudoinverse of the result separately (see section A.4). However, it is notable that second-order, Newton-like methods tend to converge faster and towards flatter minima than first-order methods like gradient descent (Martens, 2010). The optimization method of Hildebrandt and Graves is understudied in the context of machine learning, and its convergence properties compared to gradient descent are still unknown.

Due to the high cost of computing the pseudoinverse in a digital computer, this method is significantly more computationally expensive than backpropagation. However, when implemented in a physical, analog system, the relative costs of computing the inverse and transpose may be different. In the context of neural modelling, the weights of artificial neural networks are often considered analogous to resistance parameters of physical neuron models (Abbott & Dayan, 2005). Because of the applicability of Ohm's and Kirchoff's laws, analogously representing quantities as resistances have allowed certain matrix computations (such as matrix inversion) to be performed with greater efficiency than in digital computers (Sun et al., 2019).

REFERENCES

L.F. Abbott and P. Dayan. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Computational Neuroscience Series. MIT Press, 2005. ISBN 9780262311427. URL https://books.google.com/books?id=Wi4MEAAAQBAJ.

Mohamed Akrout, Collin Wilson, Peter Conway Humphreys, Timothy P. Lillicrap, and Douglas B. Tweed. Deep learning without weight transport. *CoRR*, abs/1904.05391, 2019. URL http://arxiv.org/abs/1904.05391.

Adi Ben-Israel. A newton-raphson method for the solution of systems of equations. *Journal of Mathematical Analysis and Applications*, 15(2):243–252, 1966. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(66)90115-6. URL https://www.sciencedirect.com/science/article/pii/0022247X66901156.

Adi Ben-Israel and Dan Cohen. On iterative computation of generalized inverses and associated projections. *SIAM Journal on Numerical Analysis*, 3(3):410–419, 1966. ISSN 00361429. URL http://www.jstor.org/stable/2949637.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation, 2014. URL https://arxiv.org/abs/1407.7906.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning, 2017. URL https://arxiv.org/abs/1706.03662.

Yusi Chen, Huanqiu Zhang, Mia Cameron, and Terrence Sejnowski. Predictive sequence learning in the hippocampal formation. *Neuron*, 112(15):2645–2658.e4, 2024. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2024.05.024. URL https://www.sciencedirect.com/science/article/pii/S0896627324003714.

Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, January 1989. ISSN 1476-4687. doi: 10.1038/337129a0. URL http://dx.doi.org/10.1038/337129a0.

Serra E. Favila, Brice A. Kuhl, and Jonathan Winawer. Perception and memory have distinct spatial tuning properties in human visual cortex. *Nature Communications*, 13(1), October 2022. ISSN 2041-1723. doi: 10.1038/s41467-022-33161-8. URL http://dx.doi.org/10.1038/s41467-022-33161-8.

Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, January 1987. ISSN 1551-6709. doi: 10.1111/j.1551-6708.1987.tb00862.x. URL http://dx.doi.org/10.1111/j.1551-6708.1987.tb00862.x.

T. H. Hildebrandt and Lawrence M. Graves. Implicit functions and their differentials in general analysis. *Transactions of the American Mathematical Society*, 29(1):127–153, 1927. URL http://www.jstor.org/stable/1989282.

Geoffrey Hinton E. and James L. McClelland. Learning representations by recirculation. *American Institute of Physics*, 1988.

Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLOS Comp. Bio.*, 10(11):e1003915, November 2014. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1003915. URL http://dx.doi.org/10.1371/journal.pcbi.1003915.

Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation, 2015. URL https://arxiv.org/abs/1412.7525.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks, 2014. URL https://arxiv.org/abs/1411.0247.

Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, April 2020. ISSN 1471-0048. doi: 10.1038/s41583-020-0277-3. URL http://dx.doi.org/10.1038/s41583-020-0277-3.

Juan Linde-Domingo, Matthias S. Treder, Casper Kerrén, and Maria Wimber. Evidence that neural information flow is reversed between object perception and object reconstruction from memory. *Nature Communications*, 10(1), January 2019. ISSN 2041-1723. doi: 10.1038/s41467-018-08080-2. URL http://dx.doi.org/10.1038/s41467-018-08080-2.

Jan Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley, 3 edition, 2019.

Joseph Marino. Predictive coding, variational autoencoders, and biological connections. *CoRR*, abs/2011.07464, 2020. URL https://arxiv.org/abs/2011.07464.

James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 735–742, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

D. Mumford. On the computational architecture of the neocortex: Ii the role of cortico-cortical loops. *Biological Cybernetics*, 66(3):241–251, January 1992. ISSN 1432-0770. doi: 10.1007/bf00198477. URL http://dx.doi.org/10.1007/BF00198477.

Yuji Naya, Masatoshi Yoshida, and Yasushi Miyashita. Backward spreading of memory-retrieval signal in the primate temporal cortex. *Science*, 291(5504):661–664, January 2001. ISSN 1095-9203. doi: 10.1126/science.291.5504.661. URL http://dx.doi.org/10.1126/science.291.5504.661.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, July 1955. ISSN 1469-8064. doi: 10.1017/s0305004100030401. URL http://dx.doi.org/10.1017/S0305004100030401.

D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, pp. 533–536, 1986.

Zhong Sun, Giacomo Pedretti, Elia Ambrosi, Alessandro Bricalli, Wei Wang, and Daniele Ielmini. Solving matrix equations in one step with cross-point resistive arrays. *Proceedings of the National Academy of Sciences*, 116(10):4123–4128, February 2019. ISSN 1091-6490. doi: 10.1073/pnas.1815682116. URL http://dx.doi.org/10.1073/pnas.1815682116.

J. Tapson and A. van Schaik. Learning the pseudoinverse solution to network weights. *Neural Networks*, 45:94–100, September 2013. ISSN 0893-6080. doi: 10.1016/j.neunet.2013.02.008. URL http://dx.doi.org/10.1016/j.neunet.2013.02.008.

Yongge Tian. A family of 512 reverse order laws for generalized inverses of a matrix product: a review. *Heliyon*, 6(9), 2020.

Will Xiao, Honglin Chen, Qianli Liao, and Tomaso A. Poggio. Biologically-plausible learning algorithms can scale to large datasets. *CoRR*, abs/1811.03567, 2018. URL http://arxiv.org/abs/1811.03567.

Daniel L. K. Yamins, Ha Hong, Charles F. Cadieu, Ethan A. Solomon, Darren Seibert, and James J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, May 2014. ISSN 1091-6490. doi: 10.1073/pnas.1403112111. URL http://dx.doi.org/10.1073/pnas.1403112111.

A Appendix

A.1 Proof of Theorem 1

This proof of Theorem 1 is based on that in the original paper Hildebrandt & Graves (1927), and its extension to pseudoinverses in Ben-Israel (1966).

For this section, we will consider the vector space E^n, equipped with the vector 2-norm, and the matrix space $E^{n\times m}$ with the spectral norm. Together, these have the property that for any $x \in E^n$ and $A \in E^{n\times m}$, $||Ax|| \leq ||A||||x||$.

Furthermore, let the function F be in the class $C'(Y)$. (i.e. both the mapping F and its Jacobian are continuous in the open set $Y \in E^n$). Also, we will notate the open ball centered at x_0 as $B_r(x_0) = \{x \in E^n : ||x - x_0|| < r\}$.

Theorem 1 states the following: let X_0 be a subset of E^n, $y_0 \in E^n$ be a point, and $F : X_0 \times B_r(y_0) \to E^m$ be a function such that:

1. there exists a linear function $A : B_r(y_0) \to E^m$, with reciprocal T and positive number M such that:

$$TAy = y$$
$$M||T|| < 1$$
$$||A(y_1 - y_2) - F(x, y_1) + F(x, y_2)|| \leq M||y_1 - y_2||$$

for all $x \in X_0$ and every $y \in B_r(y_0)$ where $y \in R(T)$.

2. the radius r satisfies:

$$||T||||F(x, y_0)|| < r(1 - M||T||)$$

for all $x \in X_0$

Then there exists a solution $y(x)$ which for every $x \in X_0$ satisfies

$$TF(x, y(x)) = 0$$

Consider the function:

$$G(x, y) = y - TF(x, y)$$

where $y \in B_r(y_0) \bigcap R(T)$. Then, using the property of the reciprocal,

$$\begin{aligned} G(x, y_1) - G(x, y_2) &= y_1 - y_2 - TF(x, y_1) + TF(x, y_2) \\ &= TA(y_1 - y_2) - TF(x, y_1) + TF(x, y_2) \\ &= T(A(y_1 - y_2) - F(x, y_1) + F(x, y_2)) \end{aligned}$$

So, by assumption 1:

$$||G(x, y_1) - G(x, y_2)|| \leq ||T||M||y_1 - y_2|| \qquad < ||y_1 - y_2||$$

So, for any $y \in B_r(y_0)$, $G(x, y) \in B_r(G(x, y_0))$.

Next, consider the sequence $\{y_k\}$ defined by:

$$\begin{aligned} y_1 &= G(x, y_0) \\ y_{k+1} &= G(x, y_k) \end{aligned}$$

Using assumption 2:

$$\begin{aligned} ||y_1 - y_0|| &= ||y_0 - TF(x, y_0) - y_0|| \\ &\leq ||T||||F(x, y_0)|| \\ &< r(1 - M||T||) \end{aligned}$$

Define $p = M||T|| < 1$. Then, by induction, we can show that $\{y_k\}$ is Cauchy:

$$||y_{k+1} - y_k|| < p^k r(1 - p)$$

Furthermore, it can also be shown by induction that $y_{k+1} - y_k \in R(T)$.

So, $\{y_k\}$ is convergent in $B_r(y_0) \bigcap R(T)$, and converges towards a vector y^*, which satisfies

$$TF(x, y^*) = 0$$

And, if $N(T) \subset N(A^T)$, it also satisfies:

$$A^T F(x, y^*) = 0$$

Meaning that the minima that Theorem 1 converges to would be the same as that as gradient descent.

A.2 THE EXPECTED VALUE OF THE OUTER PRODUCT OF RANDOM, MEAN-ZERO VECTORS

In this section, we will show that $\boldsymbol{E}[\boldsymbol{y}\boldsymbol{y}^T] = \sigma^2\boldsymbol{I}$. Consider a random vector $\boldsymbol{y}$, whose elements are independent with a mean of 0 and variance of σ^2. Each of these vectors can be decomposed into the form:

$$\boldsymbol{y} = \sum_{i=1}^{N} y_i \boldsymbol{e}^{(i)}$$

where $\boldsymbol{e}^{(i)}$ is a one-hot vector at index i, and y_i is a random scalar.

So, the outer product of two of these random vectors can be rewritten as:

$$\begin{aligned}\boldsymbol{y}\boldsymbol{y}^T &= (\sum_{i=1}^{N} y_i \boldsymbol{e}^{(i)})(\sum_{j=1}^{N} y_j {\boldsymbol{e}^{(j)}}^T \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} y_i \boldsymbol{e}^{(i)} {\boldsymbol{e}^{(j)}}^T y_j\end{aligned}$$

Since y is random with mean 0, when $i \neq j$ then $\mathbb{E}[y_i y_j] = \mathbb{E}[y_i]\mathbb{E}[y_j] = 0$. Otherwise, $\mathbb{E}[y_i^2] = \sigma^2$.

$$\begin{aligned}\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] &= \mathbb{E}[\sum_{i=1}^{N}\sum_{j=1}^{N} y_i \boldsymbol{e}^{(i)} {\boldsymbol{e}^{(j)}}^T y_j] \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{E}[y_i y_j] \boldsymbol{e}^{(i)} {\boldsymbol{e}^{(j)}}^T \\ &= \sum_{i=1}^{N} \mathbb{E}[y_i^2] \boldsymbol{e}^{(i)} {\boldsymbol{e}^{(i)}}^T \\ &= \sigma^2 \sum_{i=1}^{N} \boldsymbol{e}^{(i)} {\boldsymbol{e}^{(i)}}^T \\ &= \sigma^2 \boldsymbol{I}\end{aligned}$$

A.3 DETAILS OF PSEUDOINVERSE ITERATION

For the simulations in Figure 3, U and V were initialized as (10×10) square matrices. Their values were randomly initialized, with $\boldsymbol{E}_U$ and $\boldsymbol{E}_V$ randomly sampled from a uniform distribution with range $[-1, 1]$. This was done in order to keep the condition number low. Furthermore, initializations with a condition number over 20 were removed.

$$\begin{aligned}\boldsymbol{U}_0 &:= \boldsymbol{I} - \boldsymbol{E}_U \\ \boldsymbol{V}_0 &:= \boldsymbol{I} - \boldsymbol{E}_V\end{aligned}$$

Each matrix was updated with the local learning rule:

$$\begin{aligned}\boldsymbol{U}_{t+1} &:= \lambda \boldsymbol{U}_t + (1-\lambda)(\boldsymbol{U}_t - \boldsymbol{U}_t\boldsymbol{V}_t\boldsymbol{U}_t) \\ \boldsymbol{V}_{t+1} &:= \lambda \boldsymbol{V}_t + (1-\lambda)(\boldsymbol{V}_t - \boldsymbol{V}_t\boldsymbol{U}_t\boldsymbol{V}_t)\end{aligned}$$

Where λ is a decay constant used to increase stability. During simulations this was set to $\lambda = 0.9$.

A.4 COMPARISON WITH GAUSS-NEWTON OPTIMIZATION

Newton's optimization method involves preconditioning the gradient with the inverse of the Hessian, in order to account for the curvature of the loss landscape. Geometrically, this can be thought of as taking larger "steps" when the landscape is flatter, and smaller "steps" when it is sharper - generally resulting in fewer steps needed than regular gradient descent.

In the context of optimizing the weights of a neural network, with the Hessian H_{W_l}, the block-diagonal approximation of the optimal Newton direction is given by:

$$\delta_{W_l} = (H_{W_l})^{-1} \nabla_{W_l} e \tag{1}$$

The Gauss-Newton method approximates the Hessian with $H_{W_l} \approx (J^e_{W_l})^T J^e_{W_l}$. Using the Gauss-Newton approximation, and expanding the gradient term in (1) results in:

$$\delta_{W_l} = ((J^e_{W_l})^T J^e_{W_l})^{-1} (J^e_{W_l})^T e \tag{2}$$

Furthermore, if we assume $J^e_{W_l}$ has full row rank, (2) is equivalent to the pseudoinverse expression:

$$\delta_{W_l} = (J^e_{W_l})^+ e \tag{3}$$

If it was the case that, like the transpose, $(J^e_{W_{l+1}} J^e_{W_l} \dots)^+ = \dots (J^e_{W_l})^+ (J^e_{W_{l+1}})^+$, then our method would be equivalent to block-diagonal Gauss-Newton optimization (Botev et al., 2017). Unfortunately, that does not generally apply to the pseudoinverse. Instead, the expected recursion with respect to the whole network would be

$$(J^e_{W_l})^+ = (W_l^T \mathcal{D}_\sigma^T (J^e_{W_{l+1}})^T J^e_{W_{l+1}} \mathcal{D}_\sigma W_l)^{-1} W_l^T \mathcal{D}_\sigma^T (J^e_{W_{l+1}})^T \tag{4}$$

Which is significantly more complicated to implement using local connections.