
Error Broadcast and Decorrelation as a Potential Artificial and Natural Learning Mechanism

Mete Erdogan^{1,2} Cengiz Pehlevan^{3,4,5} Alper T. Erdogan^{1,2}

¹KUIS AI Center, Koc University, Turkey ²EEE Department, Koc University, Turkey

³John A. Paulson School of Engineering & Applied Sciences, Harvard University, USA

⁴Kempner Institute for the Study of Natural and Artificial Intelligence, Harvard University, USA

⁵Center for Brain Science, Harvard University, USA

{merdogan18, alperdogan}@ku.edu.tr, cpehlevan@seas.harvard.edu

Abstract

We introduce *Error Broadcast and Decorrelation* (EBD), a novel learning framework for neural networks that addresses credit assignment by directly broadcasting output errors to individual layers, circumventing weight transport of backpropagation. EBD is rigorously grounded in the stochastic orthogonality property of Minimum Mean Square Error estimators. This fundamental principle states that the error of an optimal estimator is orthogonal to functions of the input. Guided by this insight, EBD defines layerwise loss functions that directly penalize correlations between layer activations and output errors, thereby establishing a principled foundation for error broadcasting. This theoretically sound mechanism naturally leads to the experimentally observed three-factor learning rule and integrates with biologically plausible frameworks to enhance performance and plausibility. Numerical experiments demonstrate EBD’s competitive or better performance against other error-broadcast methods on benchmark datasets. Our findings establish EBD as an efficient, biologically plausible, and principled alternative for neural network training. The implementation is available at: <https://github.com/meterdogan07/error-broadcast-decorrelation>.

1 Introduction

Neural networks are dominant mathematical models for biological and artificial intelligence. A major challenge in these networks is determining how to adjust individual synaptic weights to optimize a global learning objective, known as the *credit assignment problem*. In Artificial Neural Networks (ANNs), the most common solution is the *backpropagation* (BP) algorithm [1].

In contrast to ANNs, the mechanisms for credit assignment in biological neural networks remain poorly understood. While backpropagation is highly effective for training ANNs, it is not directly applicable to biological systems because it relies on biologically implausible assumptions. In its standard form, backpropagation propagates output errors backward through a separate pathway, reusing the same synaptic weights as in the forward pass (Figure 1a). This requirement for weight symmetry is not supported by biological evidence [2]. Although many experimentally motivated models of local synaptic plasticity have been proposed [3], a biologically feasible theory of credit assignment that integrates these mechanisms remains unresolved.

To address the credit assignment problem in biological networks, researchers have proposed methods known as *error broadcasting* [4–9]. These methods involve broadcasting the global output error directly to all layers, often through random projections or fixed pathways, without relying on precise backward paths or symmetric weights (as summarized in Section 1.1). This eliminates the weight

symmetry issue inherent in backpropagation. Error broadcasting offers practical benefits for hardware implementation; recent work [10] demonstrates potential for efficient neural network execution. However, despite encouraging progress in both theory and application [11, 12], error broadcasting still needs stronger theoretical foundations to fully validate and enhance its training effectiveness.

In this context, we introduce a novel learning framework termed the *Error Broadcast and Decorrelation* (EBD), which builds on basic error broadcasting by introducing layer-specific objectives grounded in estimation theory. The fundamental principle of EBD is to adjust network weights to minimize the correlation between broadcast output errors and the activations of each layer. This approach is rigorously grounded in Minimum Mean Square Error (MMSE) estimation, where an optimal estimator’s error is orthogonal to any measurable function of its input. We leverage this orthogonality principle for EBD, defining layer-specific training losses to drive layer activations (functions of the network input) towards orthogonality with the broadcast error. This enables a more distributed mechanism for credit assignment, alternative to approaches relying solely on an output-defined loss and end-to-end error propagation.

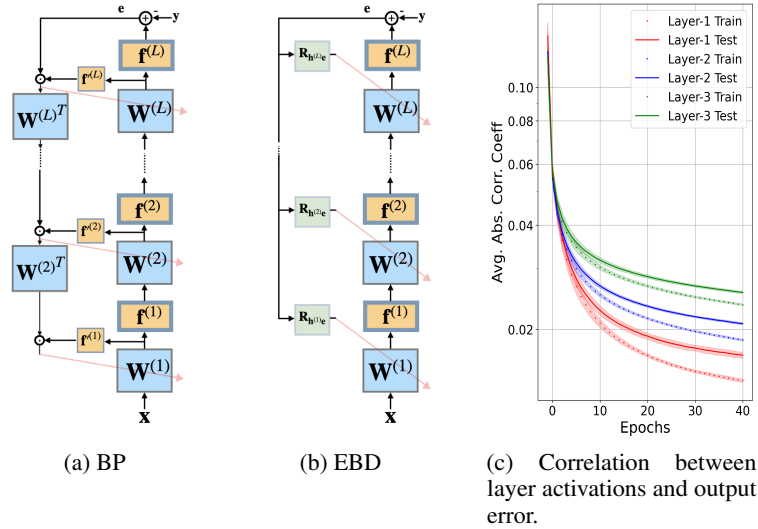


Figure 1: Comparison of error feedback mechanisms and correlation dynamics in multilayer perceptrons. (a) Backpropagation (BP) transmits errors sequentially through symmetric backward paths. (b) Error Broadcast and Decorrelation (EBD) broadcasts output errors to all layers using error-activation cross-correlations. (c) Average absolute correlation between layer activations and the output error during BP training on CIFAR-10 with MSE loss, illustrating its decline over epochs (see Appendix J).

EBD directly broadcasts output errors to layers, simplifying credit assignment and enabling parallel synaptic updates. It offers two key advantages for biologically realistic networks. First, optimizing EBD’s loss naturally leads to experimentally observed three-factor learning rules [13, 14], which extend Hebbian plasticity by incorporating a neuromodulatory signal (the third factor) modulating synaptic updates based on pre- and postsynaptic activity. Second, by broadcasting errors directly to layers as shown in Figure 1b, it overcomes the weight transport problem inherent in backpropagation and some more biologically plausible credit assignment approaches [15, 16].

We demonstrate EBD’s utility by applying it to both artificial and biologically realistic neural networks. Benchmark results show EBD matching/exceeding state-of-the-art error-broadcast techniques. Its successful application to a 10-layer biologically plausible network (CorInfoMax-EBD) provides initial evidence of depth scalability for more complex tasks.

1.1 Related work and contributions

Several frameworks have been proposed as alternatives to the backpropagation algorithm for modeling credit assignment in biological networks [8]. These include predictive coding [15, 17, 18], similarity matching [16, 19], time-contrastive approaches [20–22], forward-only methods [23–25], target

propagation [26–28], random feedback alignment [29], and learned feedback weights [30, 31]. Alternative strategies also seek to establish local learning rules by optimizing statistical objectives, such as the Hilbert-Schmidt Independence Criterion (HSIC) bottleneck [32].

Another significant alternative is error-broadcast methods, where output errors are directly transmitted to network layers without relying on precise backward pathways or symmetric weights. Two important examples of this approach are weight and node perturbation algorithms [4, 33–35], in which global error signals are broadcast to all network units. These signals reflect the change in overall error caused by individual perturbations in the network’s weights or units. A more recent and prominent example of error broadcast is Direct Feedback Alignment (DFA) [6]. In DFA, the output errors are projected onto the hidden layers through fixed random weights, effectively replacing the symmetric backward weights required in traditional backpropagation. The core challenge of weight transport has been tackled by several other methods, many of which also rely on fixed random signals or avoid feedback entirely [36, 37]. Encouragingly, a number of these biologically-plausible frameworks have demonstrated the ability to scale effectively to large datasets, underscoring their potential as viable training mechanisms [38]. This approach first emerged as a modification to the feedback alignment approach (which replaced the symmetric weights of the backpropagation algorithm with random ones). DFA has been extended and analyzed in several studies [11, 12, 39–41], demonstrating its potential in training neural networks with less biologically implausible mechanisms. Clark et al. [9] introduced another broadcast approach for a network with vector units and nonnegative weights for which three factor learning based update rule is applied.

Our framework for error broadcasting differentiates itself through

- a **principled method** based on the orthogonality property of nonlinear MMSE estimators,
- error projection weights determined by the **cross-correlation** between the output errors and the layer activations as opposed to random weights of DFA,
- dynamic Hebbian updating of projection weights as opposed to fixed weights of DFA,
- updates involving **arbitrary nonlinear functions** of layer activities, encompassing a family of three-factor learning rules,
- the option to project layer activities forward to the output layer.

In summary, our approach provides a theoretical grounding for the error broadcasting mechanism and suggests ways to enhance its effectiveness in training networks.

2 Error Broadcast and Decorrelation method

2.1 Problem statement

To illustrate our approach, we first assume a multi-layer perceptron (MLP) network with L layers. We label the input $\mathbf{x} = \mathbf{h}^{(0)} \in \mathbb{R}^{N^{(0)}}$ and layer activations $\mathbf{h}^{(k)} \in \mathbb{R}^{N^{(k)}}$ for $k = 1, \dots, L$, where $N^{(k)}$ is layer size. The layer activations are:

$$\mathbf{h}^{(k)} = f^{(k)}(\mathbf{u}^{(k)}), \quad \mathbf{u}^{(k)} = \mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}, \quad (1)$$

where $k \in \{1, \dots, L\}$ is the layer index, $f^{(k)}$ are activation functions, $\mathbf{W}^{(k)}$ weights, $\mathbf{u}^{(k)}$ preactivations and $\mathbf{b}^{(k)}$ biases. We consider input-output pairs $(\mathbf{x}, \mathbf{y})$ sampled from a joint distribution $P(\mathbf{x}, \mathbf{y})$. The performance criterion is the mean square of the output error $\epsilon = \mathbf{h}^{(L)} - \mathbf{y}$, i.e., $\mathbb{E}_{P(\mathbf{x}, \mathbf{y})}[\|\epsilon\|_2^2]$.

2.2 Error Broadcast and Decorrelation loss functions

To guide the training of our neural network (which aims to minimize this MSE), we draw inspiration from the fundamental principles of Minimum Mean Square Error (MMSE) estimation theory [42]. This theory defines an ideal estimator, denoted $\hat{\mathbf{y}}_*(\mathbf{x})$, which achieves the absolute minimum possible MSE for a given joint data distribution $P(\mathbf{x}, \mathbf{y})$. A crucial characteristic of this optimal estimator is its stochastic orthogonality property, which forms the theoretical cornerstone of our EBD approach.

Formally, considering input-output pairs $(\mathbf{x}, \mathbf{y})$ drawn from $P(\mathbf{x}, \mathbf{y})$, this optimal nonlinear MMSE estimator is given by $\hat{\mathbf{y}}_*(\mathbf{x}) = \mathbb{E}[\mathbf{y}|\mathbf{x}]$ (its derivation and properties as the optimal MSE-minimizing

function are detailed in Appendix A, Lemma A.1). Its estimation error $\epsilon_* = \mathbf{y} - \hat{\mathbf{y}}_*(\mathbf{x})$ satisfies:

$$\mathbb{E}[\mathbf{g}(\mathbf{x})\epsilon_*^T] = \mathbf{0}, \quad (2)$$

for any properly measurable function $\mathbf{g}(\mathbf{x})$ of the input $\mathbf{x}$ (see Appendix A, Lemma A.2). This means ϵ_* is orthogonal to $\mathbf{g}(\mathbf{x})$ (i.e., their expected outer product is zero). While this orthogonality property, stated in Eq. (2), is foundational, its application in constructing estimators has predominantly been in linear MMSE estimation. In that context, the estimator $\hat{\mathbf{y}}(\mathbf{x})$ is constrained to be a linear function of $\mathbf{x}$, and under the linearity constraint on the estimator, Eq. (2) is restricted to a form where $\mathbf{g}(\mathbf{x}) = \mathbf{x}$. This restricted orthogonality condition has long been used to derive parameters for linear estimators, such as Wiener-Kolmogorov and Kalman filters [43].

A key aspect of our work is to leverage the *full generality* of the orthogonality condition Eq. (2) for obtaining *nonlinear* estimators. For such estimators, this condition holds for *any* measurable function $\mathbf{g}(\mathbf{x})$ and, crucially, is not only necessary but also *sufficient* for MMSE optimality (as established in Appendix B, Theorem B.1). EBD distinctively employs this sufficiency as a constructive principle to train nonlinear estimators, specifically the neural network parameters.

We model the neural network (Eq. (1)) as a parameterized nonlinear estimator $f_\Theta(\mathbf{x}) = \mathbf{h}^{(L)}(\mathbf{x}; \Theta)$ and aim to satisfy Eq. (2) for its output error $\epsilon = f_\Theta(\mathbf{x}) - \mathbf{y}$. We choose $\mathbf{g}(\mathbf{x})$ as the network’s hidden layer activations $\mathbf{h}^{(k)}(\mathbf{x}; \Theta^{(k)})$, where $\Theta^{(k)} = (\mathbf{W}^{(k)}, \mathbf{b}^{(k)})$. This choice is motivated because:

- (i). Since each hidden-layer activation is a nonlinear function of input $\mathbf{x}$, the output error of an optimal estimator should be stochastically orthogonal to those activations. Figure 1c illustrates this phenomenon by showing the evolution of the average absolute correlation between layer activations and the error signal during backpropagation training of an MLP with three hidden layers on the CIFAR-10 dataset, based on the MSE criterion. Similar correlation declining trends are also observed across different datasets and architectures (see Appendix J). The declining correlation during MSE training reflects the MMSE estimator’s stochastic orthogonality of layer activations and output errors.,
- (ii). $\mathbf{h}^{(k)}$ depends on layer parameters $\Theta^{(k)}$, enabling their direct updates via differentiation,
- (iii). if hidden-layer activations form a “rich enough” set of functions of $\mathbf{x}$ (as elaborated below), then enforcing error orthogonality to them implies orthogonality to “every” function of $\mathbf{x}$.

Indeed, in Theorem B.2 of Appendix B, we show that when the hidden-layer activations—say, those in the first layer—form a sufficiently rich basis (for example, becoming dense in $L_2(P_{\mathbf{x}})$ as network width tends to infinity), enforcing that the output error ϵ be orthogonal to these activations naturally drives the estimator toward the true MMSE solution. Accordingly, we aim to enforce zero correlation between the output error ϵ and hidden layer activations, or more generally their nonlinear functions:

$$\mathbf{R}_{\mathbf{g}\epsilon}^{(k)} = \mathbb{E}[\mathbf{g}^{(k)}(\mathbf{h}^{(k)})\epsilon^T] = \mathbf{0}, \quad k = 0, \dots, L, \text{ with the typical choice } \mathbf{g}^{(k)}(\mathbf{h}^{(k)}) = \mathbf{h}^{(k)}. \quad (3)$$

Building on the orthogonality property and its established sufficiency for optimality (Appendix B, Theorem B.1), we define layer-specific surrogate loss functions that enforce orthogonality conditions with respect to the hidden layer activations. As demonstrated in Section 2.3, these losses yield an alternative to backpropagation by broadcasting output errors directly to network nodes (Figure 1b).

Specifically, based on the stochastic orthogonality condition in Eq. (3), we propose minimizing the Frobenius norm of the cross-correlation matrices $\mathbf{R}_{\mathbf{g}\epsilon}^{(k)}$ as a replacement for the standard MSE loss. To this end, we define the estimated cross-correlation matrix between a function $g^{(k)}$ of layer activations and the output error for batch m and layer k as

$$\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] = \lambda \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m-1] + \frac{1-\lambda}{B} \mathbf{G}^{(k)}[m] \mathbf{E}[m]^T,$$

where m is the batch index, $\lambda \in [0, 1]$ is the forgetting factor used in the autoregressive estimation, B is the batch size, $\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[0]$ is the initial value hyperparameter for the correlation matrix, and

$$\mathbf{G}^{(k)}[m] = \begin{bmatrix} \mathbf{g}^{(k)}(\mathbf{h}^{(k)}[mB+1]) & \dots & \mathbf{g}^{(k)}(\mathbf{h}^{(k)}[mB+B]) \end{bmatrix}, \quad (4)$$

is the matrix of nonlinearly transformed activations of layer k for batch m . In the above equation, $mB + l$ refers to absolute (sequence) index for the l^{th} member of batch- m . Furthermore,

$$\mathbf{E}[m] = \begin{bmatrix} \epsilon[mB+1] & \dots & \epsilon[mB+B] \end{bmatrix}, \quad (5)$$

is the output error matrix for batch m . We then define the layer-specific loss function based on the stochastic orthogonality condition for layer k as

$$J^{(k)}(\mathbf{h}^{(k)}, \epsilon)[m] = \frac{1}{2} \left\| \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \right\|_F^2, \quad (6)$$

where $\|\cdot\|_F$ denotes the Frobenius norm. This loss function captures the sum of the squared magnitudes of all cross-correlations between the components of the output error and the (potentially transformed) activations of layer k . Thus, we refer to the minimization of this loss as *decorrelation*.

2.3 Error Broadcast and Decorrelation algorithm

The functions in Eq. (6) defines individual loss functions for each hidden layer, which are used to adjust the layer parameters. These loss functions can be minimized using a gradient based algorithm.

To minimize the loss for layer k , we compute the gradient of the loss function $J^{(k)}(\mathbf{h}^{(k)}, \epsilon)$ with respect to the weight $W_{ij}^{(k)}$. The derivative can be decomposed into two terms:

$$\frac{\partial J^{(k)}(\mathbf{h}^{(k)}, \epsilon)}{\partial W_{ij}^{(k)}}[m] = \underbrace{\zeta \text{Tr} \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}[m] \frac{\partial \mathbf{G}^{(k)}[m]^T}{\partial W_{ij}^{(k)}} \right)}_{[\Delta \mathbf{W}_1^{(k)}[m]]_{ij}} + \underbrace{\zeta \text{Tr} \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \frac{\partial \mathbf{E}[m]}{\partial W_{ij}^{(k)}} \mathbf{G}^{(k)}[m]^T \right)}_{[\Delta \mathbf{W}_2^{(k)}[m]]_{ij}},$$

where $\zeta := (1 - \lambda)/B$. Similarly, the derivative with respect to the bias $b_i^{(k)}$ is given by:

$$\frac{\partial J^{(k)}(\mathbf{h}^{(k)}, \epsilon)}{\partial b_i^{(k)}}[m] = \underbrace{\zeta \text{Tr} \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}[m] \frac{\partial \mathbf{G}^{(k)}[m]^T}{\partial b_i^{(k)}} \right)}_{[\Delta \mathbf{b}_1^{(k)}[m]]_i} + \underbrace{\zeta \text{Tr} \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \frac{\partial \mathbf{E}[m]}{\partial b_i^{(k)}} \mathbf{G}^{(k)}[m]^T \right)}_{[\Delta \mathbf{b}_2^{(k)}[m]]_i}.$$

Here $\Delta \mathbf{W}_1^{(k)}, \Delta \mathbf{b}_1^{(k)}[m]$ ($\Delta \mathbf{W}_2^{(k)}, \Delta \mathbf{b}_2^{(k)}[m]$) represent the components of the gradients containing derivatives of activations (output errors) with respect to the layer parameters. As derived in Appendix C.1, we obtain the closed-form expressions for $\Delta \mathbf{W}_1^{(k)}[m]$ and $\Delta \mathbf{b}_1^{(k)}[m]$:

$$[\Delta \mathbf{W}_1^{(k)}[m]]_{ij} = \zeta \sum_{n=mB+1}^{(m+1)B} \vartheta^{(k)}[n] h_j^{(k-1)}[n], \quad [\Delta \mathbf{b}_1^{(k)}[m]]_i = \zeta \sum_{n=mB+1}^{(m+1)B} \vartheta^{(k)}[n], \quad (7)$$

where $\vartheta^{(k)}[n] = g_i^{(k)}(h_i^{(k)}[n]) f_i^{(k)}(u_i^{(k)}[n]) q_i^{(k)}[n]$, $g_i^{(k)}$ and $f_i^{(k)}$ denote the derivatives of the nonlinearity $g^{(k)}$ and the activation function $f^{(k)}$, respectively. The term $\mathbf{q}^{(k)}[m]$ is defined as:

$$\mathbf{q}^{(k)}[m] = \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \epsilon[m],$$

representing the projection of the output error onto the layer activations, with the cross-correlation matrix $\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m]$ as the transformation matrix. These projections are shown in Figure 1b.

For the special case of batchsize, $B = 1$, the weight update in (7) simplifies to

$$[\Delta \mathbf{W}_1^{(k)}[m]]_{ij} = \zeta g_i^{(k)}(h_i^{(k)}[m]) f_i^{(k)}(u_i^{(k)}[m]) q_i^{(k)}[m] h_j^{(k-1)}[m]. \quad (8)$$

The update terms $\Delta \mathbf{W}_1^{(k)}[m]$ and $\Delta \mathbf{b}_1^{(k)}[m]$ aim to adjust the activations to gradually become orthogonal to ϵ , as they are based on the derivatives of activations with respect to layer parameters. In contrast, $\Delta \mathbf{W}_2^{(k)}[m]$ and $\Delta \mathbf{b}_2^{(k)}[m]$, derived from the derivatives of the output error, work to push the output errors into a configuration more orthogonal to the activations. While both update types strive for decorrelation, a critical distinction exists: $\Delta \mathbf{W}_1^{(k)}[m]$ and $\Delta \mathbf{b}_1^{(k)}[m]$ depend only on layer activations and broadcast error signals, whereas $\Delta \mathbf{W}_2^{(k)}[m]$ and $\Delta \mathbf{b}_2^{(k)}[m]$ rely on signals propagated backward from the output layer, resembling backpropagation (see Appendix C.2).

By focusing solely on $\Delta \mathbf{W}_1^{(k)}[m]$ and $\Delta \mathbf{b}_1^{(k)}[m]$, we eliminate the need for propagation terms, resulting in a completely localized update mechanism for training the neural network. This simplification to

localized updates is supported by their positive alignment with backpropagation and full (untruncated) EBD gradient directions, as demonstrated in Appendix E.1 and E.2, respectively. Therefore, we prescribe the Error Broadcast and Decorrelation (EBD) update expressions as:

$$\mathbf{W}^{(k)}[m+1] = \mathbf{W}^{(k)}[m] - \mu^{(k)}[m] \Delta \mathbf{W}_1^{(k)}[m], \quad \mathbf{b}^{(k)}[m+1] = \mathbf{b}^{(k)}[m] - \mu^{(k)}[m] \Delta \mathbf{b}_1^{(k)}[m],$$

for $k = 1, \dots, L-1$, where $\mu^{(k)}[m]$ is the learning rate for layer k at batch m . Although these updates resemble backpropagation, a key difference lies in the error signals: the backpropagated error is replaced by the broadcasted error. Furthermore, the algorithm introduces flexibility by allowing the choice of nonlinearity functions $g^{(k)}$, which influence the gradient terms $\Delta \mathbf{W}_1^{(k)}[m]$ and $\Delta \mathbf{b}_1^{(k)}[m]$.

For the final layer ($k = L$), we utilize the standard MMSE gradient update:

$$\begin{aligned} \mathbf{W}^{(L)}[m+1] &= \mathbf{W}^{(L)}[m] - \frac{\mu^{(L)}[m]}{B} \sum_{n=mB+1}^{(m+1)B} \left(f'^{(k)}(\mathbf{u}^{(L)}[n]) \odot \boldsymbol{\epsilon}[n] \right) \mathbf{h}^{(L-1)}[n]^T, \\ \mathbf{b}^{(L)}[m+1] &= \mathbf{b}^{(L)}[m] - \frac{\mu^{(L)}[m]}{B} \sum_{n=mB+1}^{(m+1)B} f'^{(k)}(\mathbf{u}^{(L)}[n]) \odot \boldsymbol{\epsilon}[n], \end{aligned}$$

where $f'^{(L)}$ is the derivative of the activation function of the output layer.

2.4 Further EBD algorithm extensions

We propose further extensions to the EBD framework to address potential activation collapse, which can arise when minimizing correlations is the sole objective. To prevent unit-level collapse, we introduce power regularization, while entropy regularization is employed to prevent dimensional collapse. Both regularizations can be implemented in ANNs as well as biologically plausible networks. The biological plausibility of employing these regularizers in MLP-based EBD is discussed in Appendix H.1. Although CorInfoMax-EBD inherently includes entropy regularization, it can also benefit from the addition of power regularization for enhanced stability. Additionally, we introduce forward layer activation projections to improve the algorithm's versatility. We also extend the EBD formulations to more complex architectures, including Convolutional Neural Networks (CNNs) and Locally Connected (LC) networks. For further details on these extensions, please refer to Appendix D.

2.4.1 Avoiding collapse

A critical challenge with EBD is potential activation collapse, where decorrelation losses (Eq. (6)) are minimized by driving activations $\mathbf{h}^{(k)} \rightarrow 0$, even with non-zero output errors, undermining learning. To counteract this, we introduce two complementary regularizers:

Power normalization: To prevent total activation collapse, power normalization (Eq. (9)) regulates layer activation power around a target level $P^{(k)}$.

$$J_P^{(k)}(\mathbf{h}^{(k)})[m] = \sum_{l=1}^{N^{(k)}} (B^{-1} \sum_{n=mB+1}^{(m+1)B} h_l^{(k)}[n]^2 - P^{(k)})^2, \quad (9)$$

which simplifies to

$$J_P^{(k)}(\mathbf{h}^{(k)})[m] = \sum_{l=1}^{N^{(k)}} (h_l^{(k)}[n]^2 - P^{(k)})^2, \quad (10)$$

for $B = 1$.

Layer entropy: To mitigate collapse into low-dimensional subspaces, which restricts expressiveness, we incorporate layer entropy (Eq. (11)), building on prior work [44, 45].

$$J_E^{(k)}(\mathbf{h}^{(k)})[m] = \frac{1}{2} \log \det(\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I}). \quad (11)$$

Here, $\mathbf{R}_h^{(k)}[m]$ is the layer autocorrelation matrix, updated autoregressively with forgetting factor λ_E :

$$\mathbf{R}_h^{(k)}[m] = \lambda_E \mathbf{R}_h^{(k)}[m-1] + (1 - \lambda_E) \frac{1}{B} \mathbf{H}^{(k)}[m] \mathbf{H}^{(k)}[m]^T, \quad (12)$$

$$\text{where } \mathbf{H}^{(k)}[m] = [\mathbf{h}^{(k)}[mB+1] \quad \dots \quad \mathbf{h}^{(k)}[(m+1)B]], \quad (13)$$

is the activation matrix. Gradient derivations for these objectives are in Appendices C.3 and C.4.

2.4.2 Forward broadcast

In the EBD algorithm (Section 2.3), output errors are broadcast to layers to adjust weights and reduce correlations with activations. To complement this, we introduce forward broadcasting, projecting hidden layer activations onto the output layer to optimize the decorrelation loss by adjusting the final layer's parameters. Details are provided in Appendix C.5.

2.4.3 Extensions to other network architectures

The EBD approach is independent of network topology. We extend EBD to convolutional neural networks (CNNs) in Appendix D.1 and to locally connected (LC) networks in Appendix D.2.

3 EBD for biologically realistic networks

In the previous section, we introduced the EBD algorithm within the context of MLP networks. While MLPs can resemble biologically plausible networks depending on the credit assignment mechanism, in this section, we extend the application of the EBD approach to neural networks that exhibit more biologically realistic dynamics and architectures, motivated by its inherent solutions to key neuroscientific challenges: 1) EBD's direct error broadcast naturally resolves the problematic weight symmetry requirement of BP, and 2) its update rules intrinsically manifest as modulated, extended Hebbian mechanisms (three-factor learning), aligning with current understanding of synaptic plasticity. In the following subsections, we explore how EBD relates to the biologically plausible three-factor learning rule and demonstrate its integration with the biologically more realistic CorInfoMax networks [45].

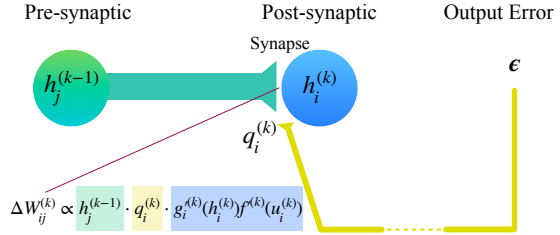


Figure 2: Error-broadcast learning as a three-factor synaptic update. The presynaptic firing rate $h_j^{(k-1)}$ (green, left) projects onto the postsynaptic neuron $h_i^{(k)}$ (blue, centre) through the synapse $W_{ij}^{(k)}$. A layer-specific broadcast of the output error $e \rightarrow q_i^{(k)}$ (yellow, right) provides the modulatory third factor that gates plasticity. Together, presynaptic activity, postsynaptic non-linear derivatives $g_i^{(k)} f^{(k)}$ (blue rectangle), and the modulatory signal form the product that drives the EBD weight change $\Delta W_{ij}^{(k)}$ displayed underneath the circuit.

3.1 Three factor learning rule and EBD

The three-factor learning rule for biological neural networks extends the traditional two-factor Hebbian rule by incorporating a modulatory signal into synaptic updates based on presynaptic and postsynaptic activity [13, 46]. While backpropagation can be expressed similarly, it is not typically considered a three-factor rule in neuroscience, as its 'third factor' is a locally tailored signal specific to each neuron requiring a biologically implausible dual network with symmetric weights, unlike global neuromodulatory signals [13]. In contrast, EBD update for batchsize $B = 1$ in (8) naturally matches the three-factor structure:

$$\Delta W_{ij}^{(k)} \propto \underbrace{g_i^{(k)}(h_i^{(k)}) f^{(k)}(u_i^{(k)})}_{\text{Postsynaptic}} \underbrace{q_i^{(k)}}_{\text{Modulatory}} \underbrace{h_j^{(k-1)}}_{\text{Presynaptic}},$$

where $q_i^{(k)}$ is the projected global error. Thus, EBD supports various three-factor rules depending on nonlinearity $g^{(k)}$. For example, $g_i^{(k)}(h_i^{(k)}) = h_i^{(k)2}$ yields the error-modulated Hebbian update

[11, 47]: $\Delta W_{ij}^{(k)} \propto h_i^{(k)} f'(u_i^{(k)}) q_i^{(k)} h_j^{(k-1)}$. By enabling diverse three-factor updates via different nonlinear functions, EBD holds potential for modeling biologically consistent neural learning processes.

Figure 2 breaks the EBD weight update (8) into its three interacting factors. The presynaptic activity $h_j^{(k-1)}$ from the sending unit; the postsynaptic term is $g_i^{(k)} f'(u_i^{(k)})$ computed from the receiving unit's own activation; and the modulatory broadcast error $q_i^{(k)}$, derived from the network's output error ϵ . Multiplying these three quantities produces the weight change $\Delta W_{ij}^{(k)}$ shown beneath the diagram, revealing that EBD naturally realises the classical three-factor learning rule in neural networks.

3.2 CorInfoMax-EBD: CorInfoMax with three factor learning rule

One of the significant advantages of the EBD framework is its flexibility to broadcast output errors into network nodes, which can be leveraged to transform time-contrastive, biologically plausible approaches into non-contrastive forms. To illustrate this property, we propose a modification of the recently introduced CorInfoMax framework [45] (see Appendix F for a summary). The CorInfoMax approach uses correlative information flow between layers as its objective function:

$$\begin{aligned} \mathcal{J}_{CI}[m] &= \sum_{k=1}^{L-1} (\hat{I}_r^{(\varepsilon_k)}(\mathbf{h}^{(k-1)}, \mathbf{h}^{(k)})[m] + \hat{I}_l^{(\varepsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m]), \quad \text{where,} \\ \hat{I}_r^{(\varepsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m] &= 0.5 \log \det(\hat{\mathbf{R}}_{\mathbf{h}^{(k+1)}}[m] + \varepsilon_k \mathbf{I}) - 0.5 \log \det(\hat{\mathbf{R}}_{\mathbf{e}^{\rightarrow(k+1)}}[m] + \varepsilon_k \mathbf{I}), \\ \hat{I}_l^{(\varepsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m] &= 0.5 \log \det(\hat{\mathbf{R}}_{\mathbf{h}^{(k)}}[m] + \varepsilon_k \mathbf{I}) - 0.5 \log \det(\hat{\mathbf{R}}_{\mathbf{e}^{\leftarrow(k)}}[m] + \varepsilon_k \mathbf{I}), \end{aligned}$$

are alternative forms of correlative mutual information between nodes, defined in terms of the correlation matrices of layer activations, i.e., $\hat{\mathbf{R}}_{\mathbf{h}^{(k)}}$ and forward and backward prediction errors ($\hat{\mathbf{R}}_{\mathbf{e}^{\rightarrow(k+1)}}$ and $\hat{\mathbf{R}}_{\mathbf{e}^{\leftarrow(k)}}$). Here, forward/backward prediction errors are defined by

$$\mathbf{e}^{\rightarrow(k+1)}[n] = \mathbf{h}^{(k+1)}[n] - \mathbf{W}^{(f,k)}[m] \mathbf{h}^{(k)}[n], \quad \mathbf{e}^{\leftarrow(k)}[n] = \mathbf{h}^{(k)}[n] - \mathbf{W}^{(b,k)}[m] \mathbf{h}^{(k+1)}[n],$$

respectively. Here, $\mathbf{W}^{(f,k)}[m]$ ($\mathbf{W}^{(b,k)}[m]$) is the forward (backward) prediction matrix for layer k .

This objective leads to network dynamics corresponding to a structure with feedforward and feedback prediction weights, and lateral connections $\mathbf{B}^{(k)}$ that maximize layer entropy. In the original work [45], the two-phase EP approach [22] is proposed to train the network weights. As an alternative, we propose employing the EBD update rule to replace the two-phase EP adaptation. The proposed CorInfoMax-EBD algorithm is described by the following update equations defined in Algorithm 1.

Algorithm 1 CorInfoMax-EBD Algorithm for Updating Weights in Layer k

Input: Batch size B , layer index k , iteration step m , learning rates $\mu^{(f,k)}$, $\mu^{(b,k)}$, $\mu^{(d_f,k)}$, $\mu^{(d_b,k)}$, $\mu^{(d_l,k)}$, factors λ_d , λ_E , γ_E , activations $\mathbf{H}^{(k)}$ in Eq. (13), the nonlinear function of layer activations $\mathbf{G}^{(k)}$ in Eq. (4) and their derivatives $\mathbf{G}_d^{(k)}$ in Eq. (28), the derivative of activations $\mathbf{F}_d^{(k)}$ in Eq. (29), output error $\mathbf{E}$ in Eq. (5), prediction errors $\mathbf{E}^{\leftarrow(k)}$ and $\mathbf{E}^{\rightarrow(k)}$ in Eq. (54)-(55), lateral outputs $\mathbf{Z}^{(k)}$ in Eq. (56).

Output: Updated weights $\mathbf{W}^{(f,k)}$, $\mathbf{W}^{(b,k)}$, $\mathbf{B}^{(k)}$.

Step 1: Update error projection weights: $\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] = \lambda_d \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m-1] + \frac{1-\lambda_d}{B} \mathbf{G}^{(k)}[m] \mathbf{E}[m]^T$

Step 2: Project errors to layer k : $\mathbf{Q}^{(k)}[m] = \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}[m]$

Step 3: Find the gradient of the nonlinear function of activations for layer k :

$$\Phi^{(k)}[m] = \mathbf{F}_d^{(k)}[m] \odot \mathbf{Q}^{(k)}[m] \odot \mathbf{G}_d^{(k)}[m]$$

Step 4: Update forward, backward and lateral weights for layer k :

$$\mathbf{W}^{(f,k)}[m] = \mathbf{W}^{(f,k)}[m-1] + \left(B^{-1} \mu^{(f,k)}[m] \mathbf{E}^{\rightarrow(k)}[m] - B^{-1} \mu^{(d_f,k)}[m] \Phi^{(k)}[m] \right) \mathbf{H}^{(k-1)}[m]^T$$

$$\mathbf{W}^{(b,k)}[m] = \mathbf{W}^{(b,k)}[m-1] + \left(B^{-1} \mu^{(b,k)}[m] \mathbf{E}^{\leftarrow(k)}[m] - B^{-1} \mu^{(d_b,k)}[m] \Phi^{(k)}[m] \right) \mathbf{H}^{(k+1)}[m]^T$$

$$\mathbf{B}^{(k)}[m] = \lambda_E^{-1} \mathbf{B}^{(k-1)}[m] - B^{-1} \gamma_E \mathbf{Z}^{(k)}[m] \mathbf{Z}^{(k)}[m]^T - B^{-1} \mu^{(d_l,k)}[m] \Phi^{(k)}[m] \mathbf{H}^{(k)}[m]^T$$

Table 1: Accuracy (%) results for MLP, CNN, and LC networks on MNIST and CIFAR-10. Best and second-best results are bold and underlined. GEVB results are from Clark et al. [9].

Dataset	Model	DFA	DFA+E (ours)	NN- GEVB	MS- GEVB	BP (MSE)	EBD (ours)
MNIST	MLP	98.1±0.21	98.2±0.03	98.1	97.7	98.7±0.05	98.2±0.08
	CNN	99.1±0.05	99.1±0.07	97.7	98.2	99.5±0.04	99.1±0.07
	LC	<u>98.9±0.03</u>	<u>98.9±0.04</u>	98.2	98.2	99.1±0.04	<u>98.9±0.04</u>
CIFAR-10	MLP	52.1±0.33	52.2±0.49	52.4	51.1	56.4±0.33	55.5±0.19
	CNN	58.4±1.59	58.6±0.66	66.3	61.6	75.2±0.28	66.4±0.43
	LC	62.2±0.21	62.1±0.19	58.9	59.9	67.8±0.27	<u>64.3±0.26</u>

Table 2: Accuracy (%) results for the CNN on CIFAR-100.

Dataset	Model	DFA	BP (CE)	EBD (ours)
CIFAR-100	CNN	41.9±0.32	60.5±0.17	<u>45.9±0.69</u>

Here, we assume layer activations $\mathbf{H}^{(k)}$, forward (backward) prediction errors $\vec{\mathbf{E}}$ ($\overleftarrow{\mathbf{E}}$), output error $\mathbf{E}$, and lateral weight outputs $\mathbf{Z}$ are computed via CorInfoMax network dynamics in [45] (see Appendix F). Integrating EBD enables single-phase updates per input, eliminating the less biologically plausible two-phase mechanism required by CorInfoMax-EP. EP’s two-phase approach—separate label-free and label-connected phases—is implausible, as biological neurons unlikely alternate between distinct global phases for learning. Our method simplifies the process, aligns more closely with biological learning, and achieves comparable or superior performance to CorInfoMax-EP (see Section 4).

The CorInfoMax-EBD scheme introduced in this section is more biologically plausible than the earlier MLP-based EBD formulation, as its learning rules can be implemented through local mechanisms such as lateral and three-factor Hebbian/anti-Hebbian updates, realistic neuron models with apical and basal dendrites, and feedback via backward predictors. Additionally, both the entropy and power normalization terms in CorInfoMax are realizable using biologically plausible operations, particularly in the online setting with single-sample updates. See Appendix H.2 for further discussion.

4 Numerical experiments

In this section, we evaluate the performance of the proposed Error Broadcast and Decorrelation (EBD) approach on benchmark datasets: MNIST [48] and CIFAR-10/100 [49]. For experiments with MNIST and CIFAR-10 involving MLP, CNN and LC, we use the same architectures used in [9]; while for CIFAR-100 we adopt a CNN architecture closely following that of [41]. We also tested the proposed CorInfoMax-EBD model against the CorInfoMax-EP model of [45]. More details about architectures, implementations, hyperparameter selections, and experimental outputs are provided in the Appendix I.

EBD test accuracy results compared to BP (with MSE criterion) and three error-broadcast methods: DFA without and with entropy regularization (DFA-E) [6], global error vector broadcasting (nonnegative-(NN-GEVB) and mixed-sign-(MS-GEVB))[9] are in Table 1 for both MNIST and CIFAR-10. Under our training setup, BP yielded comparable test accuracies for these datasets with both MSE and Cross-Entropy losses, though we report only the MSE results. In addition, CIFAR-100 results of EBD compared to BP (with Cross-Entropy criterion) and DFA is given in Table 2. Lastly, the test accuracies for biological CorInfoMax networks trained with EP and EBD methods are in Table 3.

These results show that EBD-trained networks achieve equivalent performance on the MNIST dataset and significantly better performance on the CIFAR-10 and CIFAR-100 datasets compared to other error broadcasting methods. These improvements of EBD in Table 1 over DFA can be attributed to the adaptability of error projection weights in EBD. The improvement of CorInfoMax-EBD over CorInfoMax-EP in Table 3 can be attributed to CorInfoMax-EBD incorporating error decorrelation in updating lateral weights, whereas CorInfoMax-EP relies only on (anti-)Hebbian updates. Particularly noteworthy is the performance of CorInfoMax-EBD, which not only substantially improves upon the original CorInfoMax-EP on CIFAR-10 (e.g., 55.79% vs. 50.97% for 3-layers with batch size 20) but also demonstrates encouraging scalability with depth, with a 10-layer CorInfoMax-EBD achieving 96.38% on MNIST and 54.89% on CIFAR-10 using online learning (batch size 1). This highlights EBD’s potential in deeper, more complex biological networks.

Table 3: Accuracy (%) results for EP and EBD CorInfoMax (CIM) algorithms on MNIST and CIFAR-10. Best and second-best are bold and underlined. Column marked with [*] is from Bozkurt et al. [45].

Dataset	CIM-EP [*]	CIM-EBD (Ours)	CIM-EBD (Ours)	CIM-EBD (Ours)
	3-Layers (batch size : 20)	3-Layers (batch size : 20)	3-Layers (batch size : 1)	10-Layers (batch size : 1)
MNIST	97.6	<u>97.5±0.12</u>	94.9±0.16	96.4±0.11
CIFAR-10	51.0	<u>55.7±0.17</u>	53.4±0.33	<u>54.9±0.58</u>

5 Conclusions, extensions and limitations

Conclusions. We introduced the Error Broadcast and Decorrelation framework, a biologically plausible alternative to backpropagation. EBD addresses the credit assignment problem by minimizing correlations between layer activations and output errors, offering fresh insights into biologically realistic learning. This approach provides a theoretical foundation for existing error broadcast mechanisms and three-factor learning rules in biological neural networks and facilitates flexible implementations in neuromorphic and artificial neural systems. EBD’s error-broadcasting mechanism aligns with biological processes using local updates, and notably, has proven effective for training deep recurrent biologically-plausible networks (e.g., the 10-layer CorInfoMax-EBD), thereby addressing a key challenge in effectively scaling deep, biologically plausible learning with local rules. Moreover, EBD’s simplicity and parallelism suit efficient hardware, like neuromorphic systems.

Extensions. The MMSE orthogonality property underlying EBD offers significant promise for new algorithms, deeper theoretical understanding, and neural network analysis in both artificial and biological contexts. Further theoretical extensions, drawing from the groundwork laid in Appendix B.2, could focus on deriving tighter convergence guarantees for EBD in practical (finite-width) settings and on investigating the impact of more adaptive choices for the decorrelation functions $g^{(k)}$. In addition, EBD provides theoretical underpinnings for error-broadcast mechanisms with three-factor learning rules, enabling the conversion of two-phase contrastive methods into a single-phase approach. We are currently unaware of similar theoretical properties for alternative loss functions. Finally, our numerical experiments in Appendix J.2 reveal that similar decorrelation behavior occurs for networks trained with backpropagation and categorical cross entropy loss, suggesting that decorrelation may be a general feature of the learning process and an intriguing avenue for further investigation.

Impact and limitations. This paper seeks to advance the fields of Machine Learning and Computational Neuroscience by proposing a novel learning mechanism. As a foundational learning algorithm, we do not identify specific negative societal impacts arising directly from the EBD mechanisms beyond general considerations common to advancements in machine learning. While EBD offers a theoretically-grounded framework for error broadcast based and three factor learning that has yielded competitive (and in some cases, superior) performance against other error-broadcast methods on the presented benchmarks, several aspects warrant future investigation:

Scalability: The current work evaluates EBD on MLP, CNN, LC and recurrent biological networks for image classification tasks like MNIST and CIFAR-10. Results on the 10-layer CorInfoMax-EBD demonstrate the potential to scale EBD to deeper, biologically realistic recurrent architectures using online learning. However, assessing EBD’s performance on significantly larger datasets, or its applicability to diverse large-scale architectures in other domains, remains an important open direction. While related methods like DFA have been explored in such contexts [41], comprehensive empirical validation of EBD itself under those conditions is needed.

Computational Cost and Hyperparameters: The dynamic updating of error projection matrices $\hat{\mathbf{R}}_{g\epsilon}^{(k)}$ and the optional inclusion of regularization terms like layer entropy (discussed in Appendix G and Appendix I.7) contribute to computational and memory overhead compared to simpler schemes like DFA with fixed projectors, or standard backpropagation. EBD also introduces several hyperparameters (e.g., learning rates for decorrelation and regularization, forgetting factors) that require careful tuning, although this offers flexibility. Future work could explore more efficient update mechanisms or automated tuning strategies.

Acknowledgements

This work was supported by KUIS AI Center Research Award. C.P. was supported by an NSF CAREER Award (IIS-2239780) and a Sloan Research Fellowship. This work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence.

References

- [1] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0.
- [2] Francis Crick. The recent excitement about neural networks. *Nature*, 337:129–132, 1989.
- [3] Jeffrey C Magee and Christine Grienberger. Synaptic plasticity forms and functions. *Annual Review of Neuroscience*, 43(1):95–117, 2020.
- [4] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.
- [5] Justin Werfel, Xiaohui Xie, and H Seung. Learning curves for stochastic gradient descent in linear feedforward networks. *Advances in Neural Information Processing Systems*, 16, 2003.
- [6] Arild Nokland. Direct feedback alignment provides learning in deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [7] Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial Intelligence*, 260:1–35, July 2018. doi: 10.1016/j.artint.2017.06.003.
- [8] James CR Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019.
- [9] David Clark, L.F. Abbott, and SueYeon Chung. Credit assignment through broadcasting a global error vector. In *Advances in Neural Information Processing Systems*, 2021.
- [10] Ziao Wang, Kilian Müller, Matthew Filipovich, Julien Launay, Ruben Ohana, Gustave Pariente, Safa Mokaadi, Charles Brossollet, Fabien Moreau, Alessandro Cappelli, et al. Optical training of large-scale transformers and deep neural networks with direct feedback alignment. *arXiv preprint arXiv:2409.12965*, 2024.
- [11] Blake Bordelon and Cengiz Pehlevan. The influence of learning rule on representation dynamics in wide neural networks. In *International Conference on Learning Representations*, 2022.
- [12] Julien Launay, Iacopo Poli, and Florent Krzakala. Principled training of neural networks with direct feedback alignment. *arXiv preprint arXiv:1906.04554*, 2019.
- [13] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in Neural Circuits*, 12:53, 2018.
- [14] Łukasz Kuśmierz, Takuya Isomura, and Taro Toyozumi. Learning with three factors: modulating hebbian plasticity with errors. *Current Opinion in Neurobiology*, 46:170–177, 2017.
- [15] James CR Whittington and Rafal Bogacz. An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity. *Neural Computation*, 29(5):1229–1262, 2017.
- [16] Shanshan Qin, Nayantara Mudur, and Cengiz Pehlevan. Contrastive similarity matching for supervised learning. *Neural Computation*, 33(5):1300–1328, 2021.

- [17] Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.
- [18] Siavash Golkar, Tiberiu Tesileanu, Yanis Bahroun, Anirvan Sengupta, and Dmitri Chklovskii. Constrained predictive coding as a biologically plausible model of the cortical hierarchy. *Advances in Neural Information Processing Systems*, 35:14155–14169, 2022.
- [19] Yanis Bahroun, Shagesh Sridharan, Atithi Acharya, Dmitri B Chklovskii, and Anirvan M Sengupta. Unlocking the potential of similarity matching: Scalability, supervision and pre-training. *arXiv preprint arXiv:2308.02427*, 2023.
- [20] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.
- [21] Randall C O’Reilly. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, 8(5):895–938, 1996.
- [22] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017.
- [23] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.
- [24] Matilde Tristany Farinha, Thomas Ortner, Giorgia Dellaferrera, Benjamin Grewe, and Angeliki Pantazi. Efficient biologically plausible adversarial training. *arXiv preprint arXiv:2309.17348*, 2023.
- [25] Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In *International Conference on Machine Learning*, pages 4937–4955. PMLR, 2022.
- [26] Yann Le Cun. Learning process in an asymmetric threshold network. In *Disordered Systems and Biological Organization*, pages 233–240. Springer, 1986.
- [27] Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.
- [28] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, pages 498–515. Springer, 2015.
- [29] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):13276, 2016.
- [30] John F Kolen and Jordan B Pollack. Backpropagation without weight transport. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)*, volume 3, pages 1375–1380. IEEE, 1994.
- [31] Li Ji-An and Marcus K Benna. Deep learning without weight symmetry. *arXiv preprint arXiv:2405.20594*, 2024.
- [32] Wan-Duo Kurt Ma, J P Lewis, and W Bastiaan Kleijn. The hsic bottleneck: Deep learning without back-propagation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5085–5092, 2020.
- [33] Amir Dembo and Thomas Kailath. Model-free distributed learning. *IEEE Transactions on Neural Networks*, 1(1):58–70, 1990.
- [34] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. *Advances in Neural Information Processing Systems*, 5, 1992.

- [35] Ila R Fiete and H Sebastian Seung. Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical Review Letters*, 97(4):048104, 2006.
- [36] Mohamed Akrouf, Colin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [37] Corentin Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in Neuroscience*, 15:629892, 2021.
- [38] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. In *International Conference on Learning Representations (ICLR)*, 2019.
- [39] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in Neural Information Processing Systems*, 31, 2018.
- [40] Donghyeon Han and Hoi-jun Yoo. Efficient convolutional neural network training with direct feedback alignment. *arXiv preprint arXiv:1901.01986*, 2019.
- [41] Julien Launay, Iacopo Poli, François Boniface, and Florent Krzakala. Direct feedback alignment scales to modern deep learning tasks and architectures. *Advances in Neural Information Processing Systems*, 33:9346–9360, 2020.
- [42] Athanasios Papoulis and S Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Europe: New York, NY, USA, 2002.
- [43] Thomas Kailath, Ali H Sayed, and Babak Hassibi. *Linear estimation*. Prentice-Hall information and system sciences series. Prentice Hall, 2000. ISBN 9780130224644.
- [44] Serdar Ozsoy, Shadi Hamdan, Sercan O Arik, Deniz Yuret, and Alper T. Erdogan. Self-supervised learning with an information maximization criterion. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [45] Bariscan Bozkurt, Cengiz Pehlevan, and Alper T. Erdogan. Correlative information maximization: a biologically plausible approach to supervised deep neural networks without weight symmetry. *Advances in Neural Information Processing Systems*, 37, 2023.
- [46] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 9:85, 2016.
- [47] Yonatan Loewenstein and H Sebastian Seung. Operant matching is a generic outcome of synaptic plasticity based on the covariance between reward and neural activity. *Proceedings of the National Academy of Sciences*, 103(41):15224–15229, 2006.
- [48] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- [49] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report Technical Report, University of Toronto, 2009.
- [50] Anuran Makur. Inference and information lecture notes. Lecture Notes, 2017. URL <https://www.cs.purdue.edu/homes/amakur/docs/6.437%20Recitation%20Notes%20Anuran%20Makur.pdf>. Purdue University, Department of Computer Science.
- [51] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1313–1320. Curran Associates, Inc., 2008.

- [52] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561. IEEE, 2008.
- [53] Yitong Sun, Anna C. Gilbert, and Ambuj Tewari. On the approximation properties of random ReLU features. *arXiv preprint arXiv:1810.04374*, 2018.
- [54] Alex TL Leong, Russell W Chan, Patrick P Gao, Ying-Shing Chan, Kevin K Tsia, Wing-Ho Yung, and Ed X Wu. Long-range projections coordinate distributed brain-wide neural activity with a specific spatiotemporal profile. *Proceedings of the National Academy of Sciences*, 113(51):E8306–E8315, 2016.
- [55] Leena Ali Ibrahim, Shuhan Huang, Marian Fernandez-Otero, Mia Sherer, Yanjie Qiu, Spurti Vemuri, Qing Xu, Robert Machold, Gabrielle Pouchelon, Bernardo Rudy, et al. Bottom-up inputs are required for establishment of top-down connectivity onto cortical layer 1 neurogliaform cells. *Neuron*, 109(21):3473–3485, 2021.
- [56] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. <http://yann.lecun.com/exdb/mnist/>, 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [57] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.
- [58] Tony F Chan, Gene H Golub, and Randall J LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982: Part I: Proceedings in Computational Statistics*, pages 30–41. Springer, 1982.

Appendix

Table of contents

A Preliminaries on nonlinear Minimum Mean Square Error (MMSE) estimation	17
B On the stochastic orthogonality condition based network training	18
B.1 The stochastic orthogonality condition and linear MMSE estimation	18
B.2 The use of stochastic orthogonality conditions for nonlinear MMSE estimation . .	18
C The derivation of update terms	22
C.1 $\Delta \mathbf{W}_1$ and $\Delta \mathbf{b}_1$ calculation	22
C.2 $\Delta \mathbf{W}_2$ and $\Delta \mathbf{b}_2$ calculation	23
C.3 Update corresponding to the layer entropy regularization	24
C.4 Update corresponding to the power normalization regularization	25
C.5 On the EBD with forward projections	25
D Additional extensions of EBD	27
D.1 Extensions to Convolutional Neural Networks (CNNs)	27
D.2 Extensions to Locally Connected (LC) Networks	29
E Gradient alignment in EBD	31
E.1 Alignment between EBD updates and backpropagation gradients	31
E.2 On gradient truncation and biological plausibility	32
F Background on online Correlative Information Maximization (CorInfoMax) based biologically plausible neural networks	33
F.1 The derivation of the CorInfoMax network	33
F.2 CorInfoMax-EP learning dynamics	34
F.3 CorInfoMax-EP	35
G Implementation complexity of the EBD approach	36
G.1 Complexity analysis: Error Propagation vs. Error Broadcast	36
H On the biologically plausible nature of Entropy and Power-normalization updates	39
H.1 MLP implementation with Entropy and Power-normalization regularizations	39
H.2 CorInfoMax-EBD implementation	39
H.3 Summary and conclusions	40
I Supplementary on numerical experiments	41
I.1 Architectures	41
I.2 CorInfoMax-EBD	42
I.3 Multi-Layer Perceptron	46

I.4	Convolutional Neural Network	48
I.5	Locally Connected Network	50
I.6	Implementation details for Direct Feedback Alignment (DFA) and backpropagation training	51
I.7	Runtime comparisons for the update rules	52
I.8	Reproducibility	52
I.9	Computational resources	52
I.10	Accuracy and loss curves	53
J	Calculation of the correlation between layer activations and output error	56
J.1	Correlation in the mean squared error (MSE) criterion-based training	56
J.2	Correlation in the cross-entropy criterion-based training	57

A Preliminaries on nonlinear Minimum Mean Square Error (MMSE) estimation

Let $\mathbf{y} \in \mathbb{R}^p$ and $\mathbf{x} \in \mathbb{R}^n$ represent two non-degenerate random vectors with a joint probability density function $f_{\mathbf{y}\mathbf{x}}(\mathbf{y}, \mathbf{x})$ and conditional density $f_{\mathbf{y}|\mathbf{x}}(\mathbf{y}|\mathbf{x})$. The goal of nonlinear minimum mean square error (MMSE) estimation is to find an estimator function $\mathbf{b} : \mathbb{R}^n \rightarrow \mathbb{R}^p$ that minimizes the mean squared error (MSE), which is defined as:

$$MSE(\mathbf{b}) = \mathbb{E}[\|\mathbf{y} - \mathbf{b}(\mathbf{x})\|_2^2].$$

Lemma A.1. *The best nonlinear MMSE estimate of $\mathbf{y}$ given $\mathbf{x}$ is:*

$$\mathbf{b}_*(\mathbf{x}) = \mathbb{E}_{\mathbf{y}|\mathbf{x}}(\mathbf{y}|\mathbf{x}).$$

The proof of Lemma A.1 relies on the following fundamental result (see, for example, Papoulis and Pillai [42]), which is central to the development of the entire EBD framework in the current article:

Lemma A.2. *The estimation error for $\mathbf{b}_*(\mathbf{x}) = \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]$, denoted as $\mathbf{e}_* = \mathbf{y} - \mathbf{b}_*(\mathbf{x})$, is orthogonal to any vector-valued function $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^k$ of $\mathbf{x}$. Formally, we have:*

$$\mathbb{E}[\mathbf{e}_* \mathbf{g}(\mathbf{x})^T] = \mathbf{0}.$$

Proof. (Lemma A.2) The proof follows simple steps:

$$\begin{aligned} \mathbb{E}[\mathbf{e}_* \mathbf{g}(\mathbf{x})^T] &= \mathbb{E}_{\mathbf{x}} [\mathbb{E}_{\mathbf{y}|\mathbf{x}} [(\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]) \mathbf{g}(\mathbf{x})^T | \mathbf{x}]] \\ &= \mathbb{E}_{\mathbf{x}} [(\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]) \mathbf{g}(\mathbf{x})^T] = \mathbf{0}. \end{aligned}$$

□

Using Lemma A.2, we can now prove Lemma A.1:

Proof. (Lemma A.1) Let $\mathbf{b} : \mathbb{R}^n \rightarrow \mathbb{R}^p$ be any arbitrary function. The corresponding MSE can be written as:

$$MSE(\mathbf{b}) = \mathbb{E}[\|\mathbf{y} - \mathbf{b}(\mathbf{x})\|_2^2].$$

By adding and subtracting $\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]$, we can decompose the error as:

$$\begin{aligned} MSE(\mathbf{b}) &= \mathbb{E} [\|\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] + \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x})\|_2^2] \\ &= \mathbb{E}[\|\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]\|_2^2] + \mathbb{E}(\|\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x})\|_2^2) \\ &\quad + 2\mathbb{E}[(\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}])^T (\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x}))] \\ &= \mathbb{E}[\|\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]\|_2^2] + \mathbb{E}[\|\mathbb{E}_{\mathbf{y}|\mathbf{x}}(\mathbf{y}|\mathbf{x}) - \mathbf{b}(\mathbf{x})\|_2^2] \\ &\quad + 2\mathbb{E} [Tr ((\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]) (\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x}))^T)] \\ &= \mathbb{E}[\|\mathbf{y} - \mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}]\|_2^2] + \mathbb{E}[\|\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x})\|_2^2] \\ &\quad + 2Tr(\mathbb{E}[\mathbf{e}_* (\mathbb{E}_{\mathbf{y}|\mathbf{x}}[\mathbf{y}|\mathbf{x}] - \mathbf{b}(\mathbf{x}))^T]). \end{aligned}$$

The third term, representing the cross product, vanishes by Lemma A.2, leaving us with:

$$MSE(\mathbf{b}) = \mathbb{E}[\|\mathbf{y} - \mathbf{b}_*(\mathbf{x})\|_2^2] + \mathbb{E}[\|\mathbf{b}_*(\mathbf{x}) - \mathbf{b}(\mathbf{x})\|_2^2].$$

Since the second term is always non-negative, the MSE is minimized when $\mathbf{b}(\mathbf{x}) = \mathbf{b}_*(\mathbf{x})$. □

B On the stochastic orthogonality condition based network training

This appendix provides further theoretical grounding for the Error Broadcast and Decorrelation (EBD) framework, addressing the use of the nonlinear Minimum Mean Square Error (MMSE) orthogonality condition for deriving estimators and the implications of orthogonality in high-dimensional spaces. We begin by re-establishing the geometric interpretation of stochastic orthogonality in the linear MMSE setting. We then demonstrate the sufficiency of the nonlinear MMSE orthogonality condition for an estimator to be optimal. Crucially, we address, under reasonable assumptions, how the EBD algorithm, by enforcing orthogonality to hidden unit activations, can approximate orthogonality to arbitrary measurable functions of the input, particularly in the infinite width limit. Finally, we briefly touch upon the scaling of these orthogonality conditions.

B.1 The stochastic orthogonality condition and linear MMSE estimation

Within the context of this article, *stochastic orthogonality* refers to statistical uncorrelatedness. The term *orthogonality* is defined within a Hilbert space of random variables, where the inner product between two random variables a and b is given by:

$$\langle a, b \rangle \triangleq \mathbb{E}[ab]. \quad (14)$$

Here, the inner product is defined as the expected value of their product, which corresponds to their correlation. Two random variables a and b are said to be orthogonal if their inner product is zero:

$$\langle a, b \rangle = \mathbb{E}[ab] = 0. \quad (15)$$

Thus, two random variables are orthogonal if and only if their correlation is zero (see, for example, Kailath et al. [43], Chapter 3).

In the context of Minimum Mean Square Error (MMSE) estimation, where the goal is to estimate a vector $\mathbf{y} \in \mathbb{R}^p$ from observations $\mathbf{x} \in \mathbb{R}^n$, it is known (e.g., Lemma A.2 in Appendix A) that the error $\mathbf{e}_*$ of the optimal MMSE estimator $\hat{\mathbf{y}}_*(\mathbf{x})$ satisfies:

$$\mathbb{E}[\mathbf{e}_* \mathbf{g}(\mathbf{x})^\top] = \mathbf{0}_{p \times k}, \quad (16)$$

where $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^k$ is an arbitrary vector-valued measurable function of $\mathbf{x}$. This equation states that the cross-correlation matrix between the nonlinear function of the input $\mathbf{g}(\mathbf{x})$ and the output error $\mathbf{e}_*$ is a $p \times k$ zero matrix.

The matrix equation in Eq. (16) can be expressed more explicitly as $p \cdot k$ zero-correlation conditions:

$$\mathbb{E}[e_{*,i} \cdot g_j(\mathbf{x})] = 0, \quad i = 1, \dots, p, \quad j = 1, \dots, k. \quad (17)$$

Using the inner product definition in Eq. (14), we can rewrite Eq. (17) as $p \cdot k$ stochastic orthogonality conditions:

$$\langle e_{*,i}, g_j(\mathbf{x}) \rangle = 0, \quad i = 1, \dots, p, \quad j = 1, \dots, k. \quad (18)$$

This geometric interpretation, where correlation is viewed as an inner product, is foundational. In the linear MMSE estimation setting, the orthogonality condition in (18) is restricted by choosing $\mathbf{g}(\mathbf{x})$ to be linear functions of $\mathbf{x}$, most commonly the identity mapping, i.e., $\mathbf{g}(\mathbf{x}) = \mathbf{x}$. This leads to:

$$\langle e_{*,i}, x_j \rangle = 0, \quad i = 1, \dots, p, \quad j = 1, \dots, n. \quad (19)$$

These linear orthogonality conditions are fundamental and are used in reverse to derive the parameters of linear MMSE estimators, such as the Wiener and Kalman filters [43]. The estimator's structure is assumed (linear), and its parameters are found by enforcing these orthogonality conditions.

B.2 The use of stochastic orthogonality conditions for nonlinear MMSE estimation

The principle of using orthogonality conditions to define estimators extends to the nonlinear domain. A key question is whether the nonlinear MMSE orthogonality condition can be similarly used "in reverse" to identify optimal nonlinear estimators.

B.2.1 The sufficiency of the nonlinear MMSE orthogonality condition

The nonlinear orthogonality principle states that for an optimal nonlinear MMSE estimator, the estimation error is orthogonal to any measurable function of the input. The following theorem establishes that this is not only a necessary condition but also a sufficient one, providing a strong theoretical basis for using these conditions to define and seek optimal estimators [50].

Theorem B.1 (Nonlinear MMSE Estimation and Orthogonality Condition). *Let $y \in \mathbb{R}^p$ and $x \in \mathbb{R}^n$ be random vectors with a joint probability distribution. An estimator $\hat{y}(x)$ is the optimal nonlinear MMSE estimator, $\hat{y}_{MMSE}(x)$, if and only if the error $e = y - \hat{y}(x)$ satisfies:*

$$\mathbb{E}[e \cdot g(x)^\top] = \mathbf{0}_{p \times k}$$

for all measurable functions $g : \mathbb{R}^n \rightarrow \mathbb{R}^k$ for any $k \geq 1$.

Proof. The necessity part ($\Rightarrow$), i.e., if $\hat{y}(x) = \hat{y}_{MMSE}(x)$, then the orthogonality condition holds, is a standard result (established, for instance, in Lemma A.2 Appendix A).

Here we prove the sufficiency part ($\Leftarrow$). Let $f_\Theta(x)$ be an estimator such that its error $e_\Theta = y - f_\Theta(x)$ satisfies the orthogonality condition:

$$\mathbb{E}[(y - f_\Theta(x)) \cdot g(x)^\top] = \mathbf{0}_{p \times k_g} \quad (20)$$

for all measurable functions $g : \mathbb{R}^n \rightarrow \mathbb{R}^{k_g}$ (where k_g is the dimension of $g(x)$). Consider the difference between $f_\Theta(x)$ and the true MMSE estimator $\hat{y}_{MMSE}(x)$. We can write:

$$y - f_\Theta(x) = (y - \hat{y}_{MMSE}(x)) + (\hat{y}_{MMSE}(x) - f_\Theta(x)).$$

Substituting this into the assumed orthogonality condition for $f_\Theta(x)$:

$$\mathbb{E}[(y - \hat{y}_{MMSE}(x)) + (\hat{y}_{MMSE}(x) - f_\Theta(x)) \cdot g(x)^\top] = \mathbf{0}_{p \times k_g}.$$

By linearity of expectation:

$$\mathbb{E}[(y - \hat{y}_{MMSE}(x)) \cdot g(x)^\top] + \mathbb{E}[(\hat{y}_{MMSE}(x) - f_\Theta(x)) \cdot g(x)^\top] = \mathbf{0}.$$

The first term is zero due to the (necessary) orthogonality property of the MMSE estimator $\hat{y}_{MMSE}(x)$. Therefore, we are left with:

$$\mathbb{E}[(\hat{y}_{MMSE}(x) - f_\Theta(x)) \cdot g(x)^\top] = \mathbf{0}, \quad (21)$$

for any measurable function $g(x)$. Choosing $g(x) = \hat{y}_{MMSE}(x) - f_\Theta(x)$, and taking the trace of Eq. (21), and applying the cyclic property of the trace operator, we obtain

$$\mathbb{E}[\|\hat{y}_{MMSE}(x) - f_\Theta(x)\|_2^2] = 0.$$

Since $\|\hat{y}_{MMSE}(x) - f_\Theta(x)\|_2^2$ is a non-negative random variable, its expectation being zero implies that $\|\hat{y}_{MMSE}(x) - f_\Theta(x)\|_2^2 = 0$ almost surely. This means $\hat{y}_{MMSE}(x) - f_\Theta(x) = \mathbf{0}$ almost surely, or $f_\Theta(x) = \hat{y}_{MMSE}(x)$ almost surely. Thus, $f_\Theta(x)$ is the optimal MMSE estimator. $\square$

This theorem provides a strong justification: if we can find an estimator $f_\Theta(x)$ whose error $y - f_\Theta(x)$ is orthogonal to *all* measurable functions of x , then $f_\Theta(x)$ is indeed the optimal MMSE estimator. This underpins the EBD framework's objective.

B.2.2 From orthogonality to hidden units to arbitrary functions: an infinite width perspective

A critical point is that Theorem B.1 requires the estimation error to be orthogonal to *any* measurable function $g(x)$ of the input. However, the EBD algorithm, as practically implemented, enforces orthogonality of the output error $e_\Theta(x) = y - f_\Theta(x)$ to the activations of the network's hidden units, $h_j^{(k)}(x)$. The question is whether satisfying these more limited orthogonality conditions is sufficient to approach the true MMSE estimator. We argue that in the limit of infinite network width, this can indeed be the case.

The argument relies on the universal approximation capabilities of wide neural networks:

- Rahimi and Recht showed that a hidden layer with random i.i.d. Gaussian weights and appropriately chosen biases (e.g., uniform for Fourier features) can linearly approximate functions within a corresponding Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$ associated with a shift-invariant kernel k [51, 52]. Specifically, for a function in $\mathcal{H}_k$, N such hidden units (random features) can achieve an expected $L_2(P_X)$ approximation error of order $1/\sqrt{N}$, assuming the input distribution P_X has compact support.
- Building on this, Sun et.al. analyzed random ReLU networks, where hidden units are of the form $\text{ReLU}(w^\top x)$ with $w \sim \mathcal{N}(0, I)$ [53]. They demonstrated that the RKHS induced by the corresponding random feature kernel is dense in $L_2(P_X)$ under mild conditions on P_X . This establishes the universality of random ReLU features for approximating any square-integrable function with the linear combination of that hidden layer units.

Our initial setting for the EBD framework—specifically, a first hidden layer with ReLU activations initialized with random i.i.d. Gaussian weights—is essentially the same as the random ReLU feature setting analyzed in [53]. Although weights change during training, one might hypothesize that in the infinite width limit and with sufficiently controlled learning rates (to limit the deviation of weights from their initial distribution), the set of first hidden layer activations $\{h_i^{(1)}(x)\}_{i=1}^{N^{(1)}}$ could retain its universal approximation capability. This is an area for future rigorous analysis.

Under this crucial assumption that the (potentially trained) first hidden layer activations $\{h_i^{(1)}(x)\}_{i=1}^{N^{(1)}}$ can form a basis that is dense in $L_2(P_X)$ as $N^{(1)} \rightarrow \infty$, we can state the following result concerning an estimator that achieves perfect orthogonality with these activations.

Theorem B.2 (Convergence to MMSE for Estimators Orthogonal to a Dense Basis of First-Layer Activations). *Let $f_\Theta(x)$ be an estimator for y given x . Assume its error $e_\Theta(x) = y - f_\Theta(x)$ satisfies the orthogonality condition with respect to a set of $N^{(1)}$ first-layer hidden unit activations $\{h_i^{(1)}(x)\}_{i=1}^{N^{(1)}}$:*

$$\langle y - f_\Theta(x), h_i^{(1)}(x) \rangle = 0, \quad \text{for } i = 1, \dots, N^{(1)}. \quad (22)$$

Further assume that the linear span of these activations $\{h_i^{(1)}(x)\}_{i=1}^{N^{(1)}}$ is dense in $L_2(P_X)$ as $N^{(1)} \rightarrow \infty$. That is, for any $g(x) \in L_2(P_X)$ and any $\varepsilon > 0$, there exists a sufficiently large $N^{(1)}$ and a linear combination $\hat{g}^{(1)}(x) = \sum_{i=1}^{N^{(1)}} c_i h_i^{(1)}(x)$ such that:

$$\left\| g(x) - \hat{g}^{(1)}(x) \right\|_{L_2(P_X)} \leq \varepsilon. \quad (23)$$

Then, as $\varepsilon \rightarrow 0$ (corresponding to the infinite width limit where $N^{(1)} \rightarrow \infty$), $f_\Theta(x)$ converges to the optimal MMSE estimator $f_{MMSE}(x)$ in the $L_2(P_X)$ sense:

$$\|f_{MMSE}(x) - f_\Theta(x)\|_{L_2(P_X)} \rightarrow 0.$$

Proof. Consider the inner product between the error of the estimator $f_\Theta(x)$, $e_\Theta(x) = y - f_\Theta(x)$, and an arbitrary function $g(x) \in L_2(P_X)$. We can write:

$$\begin{aligned} \langle y - f_\Theta(x), g(x) \rangle &= \langle y - f_\Theta(x), g(x) - \hat{g}^{(1)}(x) + \hat{g}^{(1)}(x) \rangle \\ &= \langle y - f_\Theta(x), g(x) - \hat{g}^{(1)}(x) \rangle + \langle y - f_\Theta(x), \hat{g}^{(1)}(x) \rangle. \end{aligned}$$

The second term, $\langle y - f_\Theta(x), \hat{g}^{(1)}(x) \rangle = \left\langle y - f_\Theta(x), \sum_{j=1}^{N^{(1)}} c_j h_j^{(1)}(x) \right\rangle = \sum_{j=1}^{N^{(1)}} c_j \langle y - f_\Theta(x), h_j^{(1)}(x) \rangle$, is zero due to the assumed orthogonality condition (22). Thus, we are left with the first term. Applying the Cauchy-Schwarz inequality and using the denseness assumption Eq. (23):

$$\begin{aligned} |\langle y - f_\Theta(x), g(x) \rangle| &= |\langle y - f_\Theta(x), g(x) - \hat{g}^{(1)}(x) \rangle| \\ &\leq \|y - f_\Theta(x)\|_{L_2(P_X)} \|g(x) - \hat{g}^{(1)}(x)\|_{L_2(P_X)} \\ &\leq \|e_\Theta(x)\|_{L_2(P_X)} \cdot \varepsilon. \end{aligned} \quad (24)$$

This shows that as $\varepsilon \rightarrow 0$, the error $e_\Theta(x)$ becomes orthogonal to any $g(x) \in L_2(P_X)$, i.e., $\langle y - f_\Theta(x), g(x) \rangle \rightarrow 0$.

Now, let $f_{MMSE}(x)$ be the true MMSE estimator. By definition, its error $e_{MMSE}(x) = y - f_{MMSE}(x)$ is orthogonal to any $g(x) \in L_2(P_X)$, so $\langle y - f_{MMSE}(x), g(x) \rangle = 0$. We can rewrite the left side of Eq. (24) as:

$$\begin{aligned}\langle y - f_{\Theta}(x), g(x) \rangle &= \langle (y - f_{MMSE}(x)) + (f_{MMSE}(x) - f_{\Theta}(x)), g(x) \rangle \\ &= \langle y - f_{MMSE}(x), g(x) \rangle + \langle f_{MMSE}(x) - f_{\Theta}(x), g(x) \rangle \\ &= 0 + \langle f_{MMSE}(x) - f_{\Theta}(x), g(x) \rangle \\ &= \langle f_{MMSE}(x) - f_{\Theta}(x), g(x) \rangle.\end{aligned}$$

Substituting this back into the inequality Eq. (24):

$$|\langle f_{MMSE}(x) - f_{\Theta}(x), g(x) \rangle| \leq \|e_{\Theta}(x)\|_{L_2(P_X)} \cdot \varepsilon. \quad (25)$$

This inequality holds for any $g(x) \in L_2(P_X)$. We can choose $g(x) = f_{MMSE}(x) - f_{\Theta}(x)$ (assuming this difference is in $L_2(P_X)$). This yields:

$$\|f_{MMSE}(x) - f_{\Theta}(x)\|_{L_2(P_X)}^2 \leq \|e_{\Theta}(x)\|_{L_2(P_X)} \cdot \varepsilon.$$

As $\varepsilon \rightarrow 0$ (corresponding to $N^{(1)} \rightarrow \infty$ under the denseness assumption), and assuming the error of the estimator $f_{\Theta}(x)$, $\|e_{\Theta}(x)\|_{L_2(P_X)}$, remains bounded, the right-hand side approaches zero. Therefore:

$$\|f_{MMSE}(x) - f_{\Theta}(x)\|_{L_2(P_X)}^2 \rightarrow 0,$$

which implies $f_{\Theta}(x) \rightarrow f_{MMSE}(x)$ in the $L_2(P_X)$ sense. $\square$

Discussion of Theorem B.2: This theorem is significant because it elucidates the properties of an estimator that achieves the specific orthogonality targeted by the EBD algorithm with respect to its first-layer hidden units. Theorem B.1 established that orthogonality to *all* measurable functions is sufficient for MMSE optimality. Theorem B.2 demonstrates that, *if* an estimator's error is perfectly orthogonal to its first-layer hidden unit activations, and *if* these activations form a dense basis (a condition motivated by infinite-width random feature networks), then such an estimator indeed converges to the true MMSE solution. This provides a theoretical rationale for the EBD algorithm's objective: by striving to decorrelate output errors with hidden unit activations, EBD aims to satisfy the premise of this theorem. The result offers a theoretical justification for how achieving orthogonality with respect to a practical, finite set of internal network features can, under idealized conditions of network width and feature richness, lead to overall MMSE optimality. The "meaningfulness" of these orthogonality constraints, even in high-dimensional spaces, is thus linked to the expressive power of the network's learned features. The theorem hinges on the denseness of the feature space generated by the first hidden layer and the perfect satisfaction of the orthogonality conditions. Formalizing the conditions under which EBD effectively approximates these conditions and the extent to which universality of features is preserved during training remain important directions for future research.

B.2.3 EBD framework objective

The EBD framework designs loss functions that aim to satisfy the orthogonality conditions discussed. Specifically, for a network $f_{\Theta}(x)$, the target is to achieve:

$$\langle y - f_{\Theta}(x), h_j^{(k)}(x) \rangle = 0, \quad \text{for all output components, layers } k, \text{ and units } j. \quad (26)$$

By minimizing decorrelation losses based on Eq. (26), particularly for a universal first layer as analyzed in Theorem B.2, the network is guided towards the MMSE optimal solution. The practical EBD algorithm uses empirical estimates of these correlations and updates network weights to minimize their magnitudes.

B.2.4 Scaling of orthogonality conditions for nonlinear MMSE

The generality of the stochastic orthogonality condition in Eq. (18) for nonlinear estimators allows, in principle, for an even greater expansion of the number of constraints. If the error $e_*(x)$ of the optimal MMSE estimator is orthogonal to any $g(x)$, it is also orthogonal to any function of its own hidden unit activations, $g_m(h_j^{(k)}(x))$, since $h_j^{(k)}(x)$ is itself a function of x . Thus, one could enforce:

$$\langle e_{EBD,i}(x), g_m(h_j^{(k)}(x)) \rangle = \mathbb{E} [e_{EBD,i}(x) \cdot g_m(h_j^{(k)}(x))] = 0, \quad (27)$$

for $i = 1, \dots, p$, $j = 1, \dots, N^{(k)}$, and for a set of M different nonlinear functions $g_m(\cdot)$. This extension could potentially introduce a greater diversity of updates and more strongly enforce the conditions for MMSE optimality. While this approach theoretically increases the number of constraints and might offer benefits, it also increases computational complexity and has not been pursued in our current numerical experiments. The practical benefit versus the added cost of enforcing orthogonality against more complex functions of hidden units remains an open question. The primary EBD algorithm focuses on $g_m(h_j^{(k)}) = h_j^{(k)}$.

C The derivation of update terms

In this section, we present the detailed derivations for the EBD algorithm and its variations, as introduced in Section 2.3.

C.1 $\Delta \mathbf{W}_1$ and $\Delta \mathbf{b}_1$ calculation

In Section 2.3, we defined the weight update element $[\Delta \mathbf{W}_1]_{ij}$ as follows:

$$\zeta \text{Tr} \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}[m] \frac{\partial \mathbf{G}^{(k)}[m]^T}{\partial W_{ij}^{(k)}} \right).$$

The derivative term in this expression can be expanded as

$$\frac{\partial \mathbf{G}^{(k)}[m]}{\partial W_{ij}^{(k)}} = \mathbf{e}_i \begin{bmatrix} g'_i{}^{(k)}(h_i^{(k)}[mB+1])f'^{(k)}(u_i^{(k)}[mB+1])h_j^{(k-1)}[mB+1] \\ g'_i{}^{(k)}(h_i^{(k)}[mB+2])f'^{(k)}(u_i^{(k)}[mB+2])h_j^{(k-1)}[mB+2] \\ \vdots \\ g'_i{}^{(k)}(h_i^{(k)}[(m+1)B])f'^{(k)}(u_i^{(k)}[(m+1)B])h_j^{(k-1)}[(m+1)B] \end{bmatrix}^T,$$

where $\mathbf{e}_i$ represents the standard basis vector with all elements set to zero, except for the element at index i , which is equal to 1.

By defining the matrix

$$\mathbf{Q}^{(k)}[m] = \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}[m] = \begin{bmatrix} \mathbf{q}^{(k)}[mB+1] & \dots & \mathbf{q}^{(k)}[(m+1)B] \end{bmatrix},$$

which represents the projection of the output error onto layer k , we can express the weight update as:

$$[\Delta \mathbf{W}_1^{(k)}[m]]_{ij} = \zeta \sum_{n=mB+1}^{(m+1)B} g'_i{}^{(k)}(h_i^{(k)}[n])f'^{(k)}(u_i^{(k)}[n])q_i^{(k)}[n]h_j^{(k-1)}[n].$$

To further simplify this expression, we introduce the matrices:

$$\mathbf{G}_d^{(k)}[m] = \begin{bmatrix} \mathbf{g}^{(k)'}(\mathbf{h}^{(k)}[mB+1]) & \mathbf{g}^{(k)'}(\mathbf{h}^{(k)}[mB+2]) & \dots & \mathbf{g}^{(k)'}(\mathbf{h}^{(k)}[(m+1)B]) \end{bmatrix}, \quad (28)$$

$$\mathbf{F}_d^{(k)}[m] = \begin{bmatrix} \mathbf{f}^{(k)'}(\mathbf{u}^{(k)}[mB+1]) & \mathbf{f}^{(k)'}(\mathbf{u}^{(k)}[mB+2]) & \dots & \mathbf{f}^{(k)'}(\mathbf{u}^{(k)}[(m+1)B]) \end{bmatrix}, \quad (29)$$

and $\mathbf{Z}^{(k)}[m] = \mathbf{G}_d^{(k)}[m] \odot \mathbf{F}_d^{(k)}[m] \odot \mathbf{Q}^{(k)}[m]$, which allows us to express the weight update in a more compact form:

$$\Delta \mathbf{W}_1^{(k)}[m] = \zeta \mathbf{Z}^{(k)}[m] \mathbf{H}^{(k-1)}[m]^T.$$

Following a similar procedure, the bias update is given by:

$$\Delta \mathbf{b}_1^{(k)}[m] = \zeta \mathbf{Z}^{(k)}[m] \mathbf{1}_{B \times 1}.$$

C.2 $\Delta \mathbf{W}_2$ and $\Delta \mathbf{b}_2$ calculation

In Section 2.3, we defined the weight update element $[\Delta \mathbf{W}_2]_{ij}$ involving the derivative of the output error as

$$\zeta Tr \left(\hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \frac{\partial \mathbf{E}[m]}{\partial W_{ij}^{(k)}} \mathbf{G}^{(k)}[m]^T \right).$$

To begin, we consider the derivative term:

$$\frac{\partial \epsilon}{\partial W_{ij}^{(k)}},$$

which can be expanded as

$$\begin{aligned} \frac{\partial \epsilon}{\partial W_{ij}^{(k)}} &= \underbrace{\frac{\partial \epsilon}{\partial \mathbf{h}^{(L)}}}_{\mathbf{I}} \underbrace{\frac{\partial \mathbf{h}^{(L)}}{\partial \mathbf{u}^{(L)}}}_{\text{diag}(f'^{(k)}(\mathbf{u}^{(L)}))} \underbrace{\frac{\partial \mathbf{u}^{(L)}}{\partial \mathbf{h}^{(L-1)}}}_{\mathbf{W}^{(L)}} \underbrace{\frac{\partial \mathbf{h}^{(L-1)}}{\partial \mathbf{u}^{(L-1)}}}_{\text{diag}(f'^{(k)}(\mathbf{u}^{(L-1)}))} \cdots \\ &\quad \cdots \underbrace{\frac{\partial \mathbf{h}^{(k+1)}}{\partial \mathbf{u}^{(k+1)}}}_{\text{diag}(f'^{(k)}(\mathbf{u}^{(k+1)}))} \underbrace{\frac{\partial \mathbf{u}^{(k+1)}}{\partial \mathbf{h}^{(k)}}}_{\mathbf{W}^{(k+1)}} \underbrace{\frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{u}^{(k)}}}_{\text{diag}(f'^{(k)}(\mathbf{u}^{(k)}))} \underbrace{\frac{\partial \mathbf{u}^{(k)}}{\partial W_{ij}^{(k)}}}_{\mathbf{e}_i h_j^{(k-1)}} \end{aligned}$$

This expression reflects propagation terms, from the output back to the layer k . Defining $\Phi^{(L)}[n] = \text{diag}(f'^{(L)}(\mathbf{u}^{(L)}[n]))$, and

$$\Phi^{(k)}[n] = \Phi^{(k+1)}[n] \mathbf{W}^{(k+1)}[m] \text{diag}(f'^{(k)}(\mathbf{u}^{(k)}[n])),$$

we obtain

$$\frac{\partial \epsilon[n]}{\partial W_{ij}^{(k)}} = \Phi^{(k)}[n] h_j^{(k-1)}[n] \mathbf{e}_i.$$

Thus, the derivative of the error at time step n with respect to $W_{ij}^{(k)}$ can be written as:

$$\begin{aligned} \zeta Tr \left(\mathbf{R}_{\mathbf{g}\epsilon}^{(k)}[m] \frac{\partial \mathbf{E}[m]}{\partial W_{ij}^{(k)}} \mathbf{G}^{(k)}[m]^T \right) &= \\ \zeta Tr \left(\mathbf{R}_{\mathbf{g}\epsilon}^{(k)}[m] \sum_{n=mB+1}^{(m+1)B} \frac{\partial \epsilon[n]}{\partial W_{ij}^{(k)}} \mathbf{g}^{(k)}(h^{(k)}[n])^T \right). \end{aligned}$$

Substituting the definition $\tilde{\mathbf{g}}^{(k)}[n] = \mathbf{R}_{\mathbf{g}\epsilon}^{(k)}[m]^T \mathbf{g}^{(k)}(h^{(k)}[n])$, we obtain:

$$\begin{aligned} \zeta Tr \left(\mathbf{R}_{\mathbf{g}(\mathbf{h}^{(k)})\epsilon}^{(k)}[m] \frac{\partial \mathbf{E}[m]}{\partial W_{ij}^{(k)}} \mathbf{G}^{(k)}[m]^T \right) &= \\ &= \zeta Tr \left(\sum_{n=mB+1}^{(m+1)B} h_j^{(k-1)}[n] \Phi^{(k)}[n] \mathbf{e}_i \tilde{\mathbf{g}}^{(k)}[n]^T \right), \\ &= \zeta \sum_{n=mB+1}^{(m+1)B} \mathbf{e}_j^T \mathbf{h}^{(k-1)}[n] \tilde{\mathbf{g}}^{(k)}[n]^T \Phi^{(k)}[n] \mathbf{e}_i, \\ &= \mathbf{e}_i^T \left(\zeta \sum_{n=mB+1}^{(m+1)B} \Phi^{(k)}[n]^T \tilde{\mathbf{g}}^{(k)}[n] \mathbf{h}^{(k-1)T}[n] \right) \mathbf{e}_j. \end{aligned}$$

Now, defining:

$$\psi^{(k)}[n] = \Phi^{(k)}[n]^T \tilde{\mathbf{g}}^{(k)}[n],$$

and assembling these into the matrix:

$$\Psi^{(k)}[m] = \begin{bmatrix} \psi^{(k)}[mB+1] & \psi^{(k)}[mB+2] & \dots & \psi^{(k)}[(m+1)B] \end{bmatrix},$$

we can compactly express the weight and bias updates as:

$$\begin{aligned} \Delta \mathbf{W}_2^{(k)}[m] &= \zeta \Psi^{(k)}[m] \mathbf{H}^{(k-1)}[m]^T, \\ \Delta \mathbf{b}_2^{(k)}[m] &= \zeta \Psi^{(k)}[m] \mathbf{1}_{B \times 1}. \end{aligned}$$

C.3 Update corresponding to the layer entropy regularization

In Section 2.4.1, we introduced the layer entropy objective as

$$J_E^{(k)}(\mathbf{h}^{(k)})[m] = \frac{1}{2} \log \det(\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I}) \quad (30)$$

where,

$$\begin{aligned} \mathbf{R}_{\mathbf{h}}^{(k)}[m] &= \lambda_E \mathbf{R}_{\mathbf{h}}^{(k)}[m-1] + (1 - \lambda_E) \frac{1}{B} \mathbf{H}^{(k)}[m] \mathbf{H}^{(k)}[m]^T, \quad \text{and,} \\ \mathbf{H}^{(k)}[m] &= \begin{bmatrix} \mathbf{h}^{(k)}[mB+1] & \dots & \mathbf{h}^{(k)}[(m+1)B] \end{bmatrix}. \end{aligned} \quad (31)$$

The derivative of the entropy objective in Eq. (30) with respect to W_{ij} is given by

$$\begin{aligned} \frac{\partial J_E^{(k)}[m]}{\partial W_{ij}^{(k)}} &= Tr \left(\nabla_{\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I}} J_E[m] \cdot \frac{\partial (\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})}{\partial W_{ij}^{(k)}} \right) \\ &= \frac{2(1 - \lambda_E)}{B} Tr \left((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m] \frac{\partial \mathbf{H}^{(k)}[m]}{\partial W_{ij}^{(k)}} \right) \end{aligned}$$

In this expression, the derivative term can be explicitly written as

$$\begin{aligned} \frac{\partial \mathbf{H}^{(k)}[m]}{\partial W_{ij}^{(k)}} &= \mathbf{F}_d^{(k)}[m] \odot \frac{\partial (\mathbf{W}^{(k)} \mathbf{H}^{(k-1)}[m] + \mathbf{b}^{(k)})}{\partial W_{ij}^{(k)}} \\ &= \mathbf{F}_d^{(k)}[m] \odot (\mathbf{e}_i \mathbf{e}_j^T \mathbf{H}^{(k-1)}[m]) \\ &= \mathbf{e}_i (\mathbf{F}_{d_{i,:}}^{(k)}[m] \odot \mathbf{H}_{j,:}^{(k-1)}[m]) \end{aligned}$$

$$\begin{aligned} \frac{\partial J_E^{(k)}[m]}{\partial W_{ij}^{(k)}} &= \frac{2(1 - \lambda_E)}{B} Tr \left((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m] (\mathbf{F}_{d_{i,:}}^{(k)}[m] \odot \mathbf{H}_{j,:}^{(k-1)}[m])^T \mathbf{e}_i^T \right) \\ &= \frac{2(1 - \lambda_E)}{B} Tr \left((((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m]) \mathbf{H}^{(k-1)}[m]^T \mathbf{e}_j \mathbf{e}_i^T \right) \\ &= \frac{2(1 - \lambda_E)}{B} \mathbf{e}_i^T (((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m]) \mathbf{H}^{(k-1)}[m]^T \mathbf{e}_j \\ &= \frac{2(1 - \lambda_E)}{B} \left[(((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m]) \mathbf{H}^{(k-1)}[m]^T \right]_{ij} \end{aligned}$$

Consequently, we obtain

$$\nabla_{\mathbf{W}^{(k)}} J_E^{(k)}[m] = \frac{2(1 - \lambda_E)}{B} (((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m]) \mathbf{H}^{(k-1)}[m]^T. \quad (32)$$

Through a similar derivation, we also obtain

$$\nabla_{\mathbf{b}^{(k)}} J_E^{(k)}[m] = \frac{2(1 - \lambda_E)}{B} (((\mathbf{R}_{\mathbf{h}}^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m]) \mathbf{1}_{B \times 1}. \quad (33)$$

C.4 Update corresponding to the power normalization regularization

In Section 2.4.1, we introduced power normalization objective as,

$$J_P^{(k)}(\mathbf{h}^{(k)})[m] = \sum_{l=1}^{N^{(k)}} \left(\frac{1}{B} \sum_{n=mB+1}^{(m+1)B} h_l^{(k)}[n]^2 - P^{(k)} \right)^2.$$

The derivative of this objective with respect to $W_{ij}^{(k)}$ can be written as

$$\begin{aligned} \frac{\partial J_P^{(k)}[m]}{W_{ij}^{(k)}} &= 4 \underbrace{\left(\frac{1}{B} \sum_{n=mB+1}^{(m+1)B} h_i^{(k)}[n]^2 - P^{(k)} \right)}_{d_i[m]} \left(\frac{1}{B} \sum_{n=mB+1}^{(m+1)B} h_i^{(k)}[n] f^{(k)'}(u^{(k)}[n]) \frac{\partial h_i^{(k)}[n]}{\partial W_{ij}^{(k)}} \right) \\ &= 4d_i[m] \left(\frac{1}{B} \sum_{n=mB+1}^{(m+1)B} h_i^{(k)}[n] f^{(k)'}(u^{(k)}[n]) h_j^{(k-1)}[n] \right). \end{aligned}$$

Therefore, the gradient of the power-normalization objective with respect to $\mathbf{W}^{(k)}$ can be written as

$$\nabla_{\mathbf{W}^{(k)}} J_P^{(k)}[m] = \frac{4}{B} \mathbf{D}[m] (\mathbf{H}^{(k)}[m] \odot \mathbf{F}_d^{(k)}) \mathbf{H}^{(k-1)}[m]^T, \quad (34)$$

where $\mathbf{D}[m] = \text{diag}(d_1[m], d_2[m], \dots, d_{N^{(k)}}[m])$. The gradient with respect to $\mathbf{b}^{(k)}$ can be obtained in a similar way as

$$\nabla_{\mathbf{b}^{(k)}} J_P^{(k)}[m] = \frac{4}{B} \mathbf{D}[m] (\mathbf{H}^{(k)}[m] \odot \mathbf{F}_d^{(k)}) \mathbf{1}_{B \times 1}. \quad (35)$$

C.5 On the EBD with forward projections

In the EBD algorithm introduced in Section 2.3, output errors are broadcast to individual layers to modify their weights, thereby reducing the correlation between hidden layer activations and output errors. To enhance this mechanism, we introduce forward broadcasting, where hidden layer activations are projected onto the output layer. This projection facilitates the optimization of the decorrelation loss by adjusting the parameters of the final layer more effectively.

The purpose of forward broadcasting is to enhance the network's ability to minimize the decorrelation loss by directly influencing the final layer's weights using the activations from the hidden layers. By projecting the hidden layer activations forward onto the output layer, we establish a direct pathway for these activations to impact the adjustments of the final layer's weights. This mechanism allows the final layer to update its parameters in a way that reduces the correlation between the output errors and the hidden layer activations. Consequently, the errors at the output layer are steered toward being orthogonal to the hidden layer activations.

This mechanism could potentially be effective because the final layer is responsible for mapping the network's internal representations to the output space. By incorporating information from earlier layers, we enable the final layer to align its parameters more closely with the features that are most relevant for reducing the overall error.

While the proposed forward broadcasting mechanism is primarily motivated by performance optimization, it can conceptually be related to the long-range [54] and bottom-up [55] synaptic connections in the brain, which allow certain neurons to influence distant targets. These long-range bottom-up connections are actively being researched, and incorporating similar mechanisms into computational models could enhance their alignment with biological neural processes. By integrating mechanisms that mirror these neural pathways, forward broadcasting may be useful for modeling how information is transmitted across different neural circuits.

C.5.1 Gradient derivation for the EBD with forward projections

We derive the gradients of the layer decorrelation losses with respect to the parameters of the final layer. The partial derivative of the objective function $J^{(k)}(\mathbf{h}^{(k)}, \epsilon)$ with respect to the final layer

weights can be written as:

$$\begin{aligned}
\frac{\partial J^{(k)}(\mathbf{h}^{(k)}, \boldsymbol{\epsilon})}{\partial W_{ij}^{(L)}}[m] &= \zeta Tr \left(\hat{\mathbf{R}}_{\mathbf{g}(\mathbf{h}^{(k)})\boldsymbol{\epsilon}}[m] \frac{\partial(\mathbf{E}[m]\mathbf{G}^{(k)}[m]^T)}{\partial W_{ij}^{(L)}} \right) \\
&= \underbrace{\zeta Tr \left(\hat{\mathbf{R}}_{\mathbf{g}(\mathbf{h}^{(k)})\boldsymbol{\epsilon}}[m] \frac{\partial \mathbf{E}[m]}{\partial W_{ij}^{(L)}} \mathbf{G}^{(k)}[m]^T \right)}_{[\Delta \mathbf{W}^{(L,k),f}[m]]_{ij}}, \\
&= \zeta Tr \left(\mathbf{R}_{\mathbf{g}(\mathbf{h}^{(k)})\boldsymbol{\epsilon}}[m] \sum_{n=mB+1}^{(m+1)B} \frac{\partial \boldsymbol{\epsilon}[n]}{\partial W_{ij}^{(L)}} \mathbf{g}(h^{(k)}[n])^T \right).
\end{aligned}$$

Substituting the definition $\tilde{\mathbf{g}}^{(k)}[n] = \mathbf{R}_{\mathbf{g}(\mathbf{h}^{(k)})\boldsymbol{\epsilon}}[m]^T \mathbf{g}(h^{(k)}[n])$, we can further express the partial derivative as:

$$\begin{aligned}
\frac{\partial J^{(k)}(\mathbf{h}^{(k)}, \boldsymbol{\epsilon})}{\partial W_{ij}^{(L)}}[m] &= \zeta Tr \left(\sum_{n=mB+1}^{(m+1)B} h_j^{(L-1)}[n] \Phi^{(L)}[n] \mathbf{e}_i \tilde{\mathbf{g}}^{(k)}[n]^T \right), \\
&= \zeta \sum_{n=mB+1}^{(m+1)B} \mathbf{e}_j^T \mathbf{h}^{(L-1)}[n] \tilde{\mathbf{g}}^{(k)}[n]^T \Phi^{(L)}[n] \mathbf{e}_i, \\
&= \mathbf{e}_i^T \left(\zeta \sum_{n=mB+1}^{(m+1)B} \Phi^{(L)}[n]^T \tilde{\mathbf{g}}^{(k)}[n] \mathbf{h}^{(L-1)T} \right) \mathbf{e}_j, \\
&= \mathbf{e}_i^T \left(\zeta \sum_{n=mB+1}^{(m+1)B} (f'(\mathbf{u}^{(L)}[n]) \odot \tilde{\mathbf{g}}^{(k)}[n]) \mathbf{h}^{(L-1)T} \right) \mathbf{e}_j.
\end{aligned}$$

Next, defining the following terms:

$$\psi^{(k,L)}[n] = f'(\mathbf{u}^{(L)}[n]) \odot \tilde{\mathbf{g}}^{(k)}[n],$$

and assembling them into the matrix:

$$\boldsymbol{\Psi}^{(k,L)}[m] = \begin{bmatrix} \psi^{(k,L)}[mB+1] & \psi^{(k,L)}[mB+2] & \dots & \psi^{(k,L)}[(m+1)B] \end{bmatrix},$$

we can write the weight update as:

$$\Delta \mathbf{W}^{(L,k),f}[m] = \zeta \boldsymbol{\Psi}^{(k,L)}[m] \mathbf{H}^{(k-1)}[m]^T.$$

Following a similar procedure, the bias update can be written as:

$$\Delta \mathbf{b}^{(L,k),f}[m] = \zeta \boldsymbol{\Psi}^{(k,L)}[m] \mathbf{1}_{B \times 1}.$$

Based on these expressions, we can write

$$\begin{aligned}
[\Delta \mathbf{W}^{(L,k),f}[m]]_{ij} &= \zeta \sum_{n=mB+1}^{(m+1)B} f^{(L)'}(u_i^{(L)}[n]) \tilde{g}_i^{(k)}[n] \mathbf{h}_j^{(L-1)} \\
[\Delta \mathbf{b}^{(L,k),f}[m]]_i &= \zeta \sum_{n=mB+1}^{(m+1)B} f^{(L)'}(u_i^{(L)}[n]) \tilde{g}_i^{(k)}[n].
\end{aligned}$$

D Additional extensions of EBD

D.1 Extensions to Convolutional Neural Networks (CNNs)

Let $\mathbf{H}^{(k)} \in \mathbb{R}^{P^{(k)} \times M^{(k)} \times N^{(k)}}$ represent the output of the k^{th} layer of a Convolutional Neural Network (CNN), where $P^{(k)}$ is the number of channels and the layer's output is $M^{(k)} \times N^{(k)}$ dimensional. Furthermore, we use $\mathbf{W}^{(k,p)} \in \mathbb{R}^{P^{(k-1)} \times \Omega^{(k)} \times \Omega^{(k)}}$ and $\mathbf{b}^{(k,p)} \in \mathbb{R}$ to represent the filter tensor weights and bias coefficient respectively for the channel- p of the k^{th} layer, and $\Omega^{(k)}$ is the symmetric convolution kernel size. Then a convolutional layer can be defined as

$$\mathbf{H}^{(k,p)} = f(\mathcal{U}^{(k,p)}), \quad (36)$$

$$\mathcal{U}^{(k,p)} = (\mathbf{H}^{(k-1)} * \mathbf{W}^{(k,p)}) + \mathbf{b}^{(k,p)}, \quad (37)$$

where the symbol " $*$ " represents the convolution¹ operation that acts upon both the spatial and channel dimensions to generate the p^{th} channel of k^{th} layer output $\mathbf{H}^{(k,p)}$, and f is the nonlinearity acted on the convolution output.

D.1.1 Error Broadcast and Decorrelation formulation

Similar to Eq. (3), we have the cross-correlation between output errors ϵ and the arbitrary function of the k^{th} layer activation of the p^{th} channel denoted as $\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})$, for each layer and spatial indexes $r \in \mathbb{Z} : 1 \leq r \leq M^{(k)}$ and $s \in \mathbb{Z} : 1 \leq s \leq N^{(k)}$ as

$$\mathbf{R}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}[q, r, s] = \mathbb{E}[\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[r, s]\epsilon_q] = \mathbf{0}. \quad (38)$$

Then this cross-correlation must ideally be zero due to the stochastic orthogonality condition. We can then write the loss for layer- k at batch- m as:

$$J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m] = \frac{1}{2} \sum_{q=1}^{n_c} \left\| \hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}[m, q, :, :] \right\|_F^2, \quad (39)$$

where $\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}$ is the recurrently estimated cross-correlation using the training batches. Then we can optimize the network by taking the derivative of the loss function with respect to the weight $\mathbf{W}_{hij}^{(k,p)}$ corresponding to input channel h and weight spatial indexes $i, j \in \mathbb{Z} : 1 \leq i, j \leq \Omega^{(k)}$ as

$$\begin{aligned} & \frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]}{\partial \mathbf{W}_{hij}^{(k,p)}} \\ &= \zeta \sum_{q=1}^{n_c} \sum_{n=mB+1}^{(m+1)B} \epsilon_q[n] \cdot Tr \left(\left(\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}[m, q, :, :] \right)^T \frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p)}} \right) \\ &= \zeta \sum_{q=1}^{n_c} \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \epsilon_q[n] \left[\left(\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}[m, q, :, :] \odot \frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p)}} \right) \right]_{[r,s]}, \end{aligned} \quad (40)$$

in which n_c is the error dimension, $N^{(k)}$ and $M^{(k)}$ are the width and height of the k^{th} layer, and the derivative with respect to the ϵ term is neglected. The inner partial derivative term can be written as

$$\frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p)}} = \mathbf{g}'^{(k)}(\mathbf{H}^{(k,p)})[n, :, :] \odot \frac{\partial \mathbf{H}^{(k,p)}[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p)}}, \quad (41)$$

and using the Eq. (36) and Eq. (37),

$$\frac{\partial \mathbf{H}^{(k,p)}[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p)}} = f'(\mathcal{U}^{(k,p)})[n, :, :] \odot (\mathcal{E}_{hij}^{(k)} * \mathbf{H}^{(k-1)}[n, :, :]). \quad (42)$$

¹Although we call it as convolution, in CNNs, the actual operation used is the correlation operation where the kernel is unflipped.

where $\mathcal{E}_{hij}^{(k)} \in \mathbb{R}^{P^{(k-1)} \times \Omega^{(k)} \times \Omega^{(k)}}$ is a Kronecker delta tensor that occurs as the gradient of $\mathbf{W}^{(k,p)}$ with respect to $\mathbf{W}_{hij}^{(k,p)}$. Combining the expressions, we have

$$\phi[n, p, :, :] = \sum_{q=1}^{n_c} \epsilon_q[n] \cdot \left(\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon} [n, q, :, :] \odot \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)}[n, :, :]) \odot f'(\mathcal{U}^{(k,p)}[n, :, :]) \right). \quad (43)$$

Then, combining the Equations (40), (41), (42), and then writing the convolution explicitly, we have

$$\begin{aligned} \frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]}{\partial \mathbf{W}_{hij}^{(k,p)}} &= \zeta \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \left[\phi[n, p, :, :] \odot (\mathcal{E}_{hij}^{(k)} * \mathbf{H}^{(k-1)}[n, :, :]) \right]_{[r,s]} \\ &= \zeta \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \phi[n, p, r, s] \cdot \left(\sum_{h', i', j'} \mathcal{E}_{hij}^{(k)}[h', i', j'] \cdot \mathbf{H}^{(k-1, h')}[n, r + i', s + j'] \right). \end{aligned}$$

By the definition of the delta function $\mathcal{E}_{hij}^{(k)}$ and writing the resulting expression as a 2D convolution between $\mathbf{H}^{(k-1)}$ and ϕ respectively, we have

$$\begin{aligned} &= \zeta \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \phi[n, p, r, s] \cdot \mathbf{H}^{(k-1, h)}[n, r + i, s + j] \\ &= \zeta \sum_{n=mB+1}^{(m+1)B} \left[\phi[n, p, :, :] * \mathbf{H}^{(k-1, h)}[n, :, :] \right]_{[i,j]}. \end{aligned}$$

The resulting expression for the weight update is:

$$\frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]}{\partial \mathbf{W}_h^{(k,p)}} = \zeta \sum_{n=mB+1}^{(m+1)B} (\phi[n, p, :, :] * \mathbf{H}^{(k-1, h)}[n, :, :]). \quad (44)$$

Similarly, it can be shown that the bias update:

$$\frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]}{\partial \mathbf{b}^{(k,p)}} = \zeta \sum_{n=mB+1}^{(m+1)B} \sum_{r=1}^{N^{(k)}} \sum_{s=1}^{M^{(k)}} \phi[n, p, r, s].$$

The convolutional layer parameters can be trained using these gradient formulas for each layer separately, and can be calculated by utilizing the batched convolution operation.

D.1.2 Weight entropy objective

The layer entropy objective is computationally cumbersome for a convolutional layer that has multiple dimensions. Therefore, we propose the weight-entropy objective to avoid dimensional collapse

$$J_E^{(k)}(\mathbf{W}^{(k)}) = \frac{1}{2} \log \det(\mathbf{R}_{\overline{\mathbf{W}}^{(k)}} + \varepsilon^{(k)} \mathbf{I}),$$

where we define $\overline{\mathbf{W}}^{(k)} \in \mathbb{R}^{P^{(k)} \times P^{(k-1)} \cdot \Omega^{(k)} \cdot \Omega^{(k)}}$ as the unraveled version of the full size weight tensor $\mathbf{W}^{(k)}$, and the covariance matrix $\mathbf{R}_{\overline{\mathbf{W}}^{(k)}}$ is conditionally defined as:

$$\mathbf{R}_{\overline{\mathbf{W}}^{(k)}} = \begin{cases} \overline{\mathbf{W}}^{(k)T} \overline{\mathbf{W}}^{(k)}, & \text{if } P^{(k)} \geq P^{(k-1)} \cdot \Omega^{(k)} \cdot \Omega^{(k)}, \\ \overline{\mathbf{W}}^{(k)} \overline{\mathbf{W}}^{(k)T}, & \text{otherwise,} \end{cases}$$

to decrease its dimensions and reduce the computational costs for further steps. Then, the derivative of this objective can be written as:

$$\Delta J_E^{(k)}(\mathbf{W}^{(k)}) = \begin{cases} \overline{\mathbf{W}}^{(k)} \mathbf{R}_{\overline{\mathbf{W}}^{(k)}}^{-1}, & \text{if } P^{(k)} \geq P^{(k-1)} \cdot \Omega^{(k)} \cdot \Omega^{(k)}, \\ \mathbf{R}_{\overline{\mathbf{W}}^{(k)}}^{-1} \overline{\mathbf{W}}^{(k)}, & \text{otherwise.} \end{cases}$$

Therefore, $\frac{\partial J_E^{(k)}(\mathbf{W}^{(k)})}{\partial \mathbf{W}_{hij}^{(k,p)}}$ can be obtained by reshaping $\Delta J_E^{(k)}(\mathbf{W}^{(k)})$ as the weight tensor $\mathbf{W}^{(k)}$.

D.1.3 Activation sparsity regularization

To further regularize the model, we enforce the layer activation sparsity loss that is given as

$$J_{\ell_1}^{(k)}(\mathbf{H}^{(k,p)}) = \frac{\|\mathbf{H}^{(k,p)}\|_1}{\|\mathbf{H}^{(k,p)}\|_2}. \quad (45)$$

The gradient of the sparsity loss with respect to the hidden layer can be written as:

$$\Delta J_{\ell_1}^{(k)}(\mathbf{H}^{(k,p)}) = \frac{1}{\|\mathbf{H}^{(k,p)}\|_2} \text{sign}(\mathbf{H}^{(k,p)}) - \frac{\|\mathbf{H}^{(k,p)}\|}{\|\mathbf{H}^{(k,p)}\|_2^3} \mathbf{H}^{(k,p)}. \quad (46)$$

Then, the gradient of the loss with respect to the model weights can be calculated in a similar manner as the Eq. (44):

$$\frac{\partial J_{\ell_1}^{(k)}(\mathbf{H}^{(k,p)})[m]}{\partial \mathbf{W}_h^{(k,p)}} = \frac{1}{B} \sum_{n=mB+1}^{(m+1)B} \left(\Delta J_{\ell_1}^{(k)}(\mathbf{H}^{(k,p)})[n, p, :, :] * \mathbf{H}^{(k-1,h)}[n, :, :] \right).$$

D.2 Extensions to Locally Connected (LC) Networks

Let $\mathbf{H}^{(k)} \in \mathbb{R}^{P^{(k)} \times M^{(k)} \times N^{(k)}}$ represent the output of the k^{th} layer of a Locally Connected Network (LC), where $P^{(k)}$ is the number of channels and the layer's output is $M^{(k)} \times N^{(k)}$ dimensional. We use $\mathbf{W}^{(k,p,r,s)} \in \mathbb{R}^{P^{(k-1)} \times \Omega^{(k)} \times \Omega^{(k)}}$ and $\mathbf{b}^{(k,p,r,s)} \in \mathbb{R}$ to represent the filter tensor weights and bias coefficient at spatial locations $r \in \mathbb{Z} : 1 \leq r \leq M^{(k)}$ and $s \in \mathbb{Z} : 1 \leq s \leq N^{(k)}$, for channel- p of the k^{th} layer, where $\Omega^{(k)}$ is the local receptive field size. Then a locally connected layer can be defined as

$$\mathbf{H}^{(k,p)} = f(\mathcal{U}^{(k,p)}), \quad (47)$$

$$\mathcal{U}^{(k,p)} = (\mathbf{H}^{(k-1)} \circledast \mathbf{W}^{(k,p)}) + \mathbf{b}^{(k,p)}, \quad (48)$$

where the symbol " $\circledast$ " represents the locally connected operation which acts upon both the spatial and channel dimensions, but without weight sharing across spatial locations, generating the p^{th} channel of the k^{th} layer output $\mathbf{H}^{(k,p)}$, and f is the nonlinearity applied to the result.

D.2.1 Error Broadcast and Decorrelation formulation

For the LC network, the stochastic orthogonality condition and the corresponding loss $J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]$ for layer- k at batch- m can be written equivalently as Eq. (38) and Eq. (39) respectively. Then the optimization can be performed by taking the derivative of the loss function with respect to $\mathbf{W}_{hij}^{(k,p,r,s)}$ corresponding to input channel h , weight spatial indexes $i, j \in \mathbb{Z} : 1 \leq i, j \leq \Omega^{(k)}$ as

$$\begin{aligned} & \frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \epsilon)[m]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} \\ &= \zeta \sum_{q=1}^{n_c} \sum_{n=mB+1}^{(m+1)B} \epsilon_q[n] \cdot Tr \left((\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon} [m, q, :, :])^T \frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} \right) \\ &= \zeta \sum_{q=1}^{n_c} \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \epsilon_q[n] \left[(\hat{\mathbf{R}}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon} [m, q, :, :]) \odot \frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} \right]_{[r,s]}. \end{aligned} \quad (49)$$

The inner partial derivative term can be written as

$$\frac{\partial \mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} = \mathbf{g}'^{(k)}(\mathbf{H}^{(k,p)})[n, :, :] \odot \frac{\partial \mathbf{H}^{(k,p)}[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}}, \quad (50)$$

and using Eq. (47) and Eq. (48), we obtain:

$$\frac{\partial \mathbf{H}^{(k,p)}[n, :, :]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} = f'(\mathcal{U}^{(k,p)})[n, :, :] \odot (\mathcal{E}_{hij}^{(k)} \circledast \mathbf{H}^{(k-1)}[n, :, :]). \quad (51)$$

Here, $\mathcal{E}_{hij}^{(k,r,s)} \in \mathbb{R}^{P^{(k-1)} \times \Omega^{(k)} \times \Omega^{(k)}} \times M^{(k)} \times N^{(k)}$ is a Kronecker delta tensor that occurs as the gradient of $\mathbf{W}^{(k,p)}$ with respect to $\mathbf{W}_{hij}^{(k,p,r,s)}$. Combining the expressions in Eq. (49), Eq. (50), Eq. (51), and the expression for ϕ as in Eq. (43) which is equivalent for both CNNs and LCs, and then writing the locally connected operation explicitly, we have

$$\begin{aligned} \frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \boldsymbol{\epsilon})[m]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} &= \zeta \sum_{n=mB+1}^{(m+1)B} \sum_{r,s} \left[\phi[n, p, :, :] \odot (\mathcal{E}_{hij}^{(k,r,s)} \circledast \mathbf{H}^{(k-1)}[n, :, :]) \right]_{[r,s]} \\ &= \zeta \sum_{n=mB+1}^{(m+1)B} \phi[n, p, r, s] \cdot \left(\sum_{\substack{h', i', j' \\ r', s'}} \mathcal{E}_{hij}^{(k,r,s)}[h', i', j', r', s'] \cdot \mathbf{H}^{(k-1, h')}[n, r' + i', s' + j'] \right). \end{aligned}$$

Then, by the definition of the Kronecker delta, the resulting expression for the weight update is:

$$\frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \boldsymbol{\epsilon})[m]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} = \zeta \sum_{n=mB+1}^{(m+1)B} \left(\phi[n, p, r, s] \cdot \mathbf{H}^{(k-1, h)}[n, r + i, s + j] \right). \quad (52)$$

Similarly, it can be shown that the bias update is:

$$\frac{\partial J^{(k)}(\mathbf{H}^{(k,p)}, \boldsymbol{\epsilon})[m]}{\partial \mathbf{b}^{(k,p,r,s)}} = \zeta \sum_{n=mB+1}^{(m+1)B} \phi[n, p, r, s].$$

D.2.2 Weight entropy objective

Similar to CNNs, we propose the weight-entropy objective to avoid dimensional collapse in LCs

$$J_E^{(k)}(\mathbf{W}^{(k)}) = \frac{1}{2} \log \det(\mathbf{R}_{\mathbf{W}^{(k)}} + \varepsilon^{(k)} \mathbf{I}),$$

where we define $\overline{\mathbf{W}}^{(k)} \in \mathbb{R}^{P^{(k)} \times P^{(k-1)} \cdot M^{(k)} \cdot N^{(k)} \cdot \Omega^{(k)} \cdot \Omega^{(k)}}$ as the unraveled version of the full size weight tensor $\mathbf{W}^{(k)}$, then the covariance matrix $\mathbf{R}_{\mathbf{W}^{(k)}}$ is defined as:

$$\mathbf{R}_{\mathbf{W}^{(k)}} = \overline{\mathbf{W}}^{(k)T} \overline{\mathbf{W}}^{(k)}.$$

Then, the derivative of this objective can be written as:

$$\Delta J_E^{(k)}(\mathbf{W}^{(k)}) = \overline{\mathbf{W}}^{(k)} \mathbf{R}_{\mathbf{W}^{(k)}}^{-1}$$

$\frac{\partial J_E^{(k)}(\mathbf{W}^{(k)})}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}}$ can be obtained by reshaping $\Delta J_E^{(k)}(\mathbf{W}^{(k)})$ as the weight tensor $\mathbf{W}^{(k)}$.

D.2.3 Activation sparsity regularization

The layer activation sparsity loss for the LC is the same as the one given for the CNN in Eq. (45), with its gradient with respect to the activations as in Eq. (46). Then, the gradient of the loss with respect to the model weights can be calculated in a similar manner as the expression in Eq. (52):

$$\frac{\partial J_{\ell_1}^{(k)}(\mathbf{H}^{(k)})[m]}{\partial \mathbf{W}_{hij}^{(k,p,r,s)}} = \frac{1}{B} \sum_{n=mB+1}^{(m+1)B} \left(\Delta J_{\ell_1}^{(k)}(\mathbf{H}^{(k)})[n, p, r, s] \circledast \mathbf{H}^{(k-1, h)}[n, r + i, s + j] \right).$$

E Gradient alignment in EBD

E.1 Alignment between EBD updates and backpropagation gradients

To investigate the relationship between the EBD update directions and the gradients produced by backpropagation (BP), we analyze the cosine similarity between the EBD update vectors and the corresponding BP gradients throughout training. This analysis quantifies how well the EBD learning dynamics align with traditional gradient-based optimization methods.

We conduct experiments on two architectures: a 3-layer multilayer perceptron (MLP) and a locally connected (LC) network, both trained on CIFAR-10. Figure 3 and Figure 4 illustrate the cosine similarity between EBD updates and BP gradients at each training epoch.

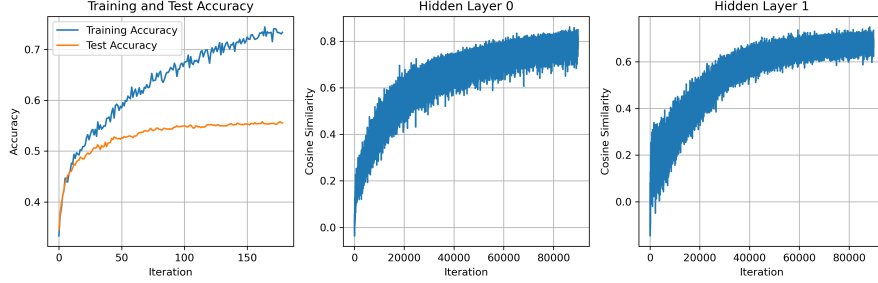


Figure 3: Cosine similarity between EBD updates and backpropagation gradients in a 3-layer MLP trained on CIFAR-10. Alignment is consistently positive and improves during training.

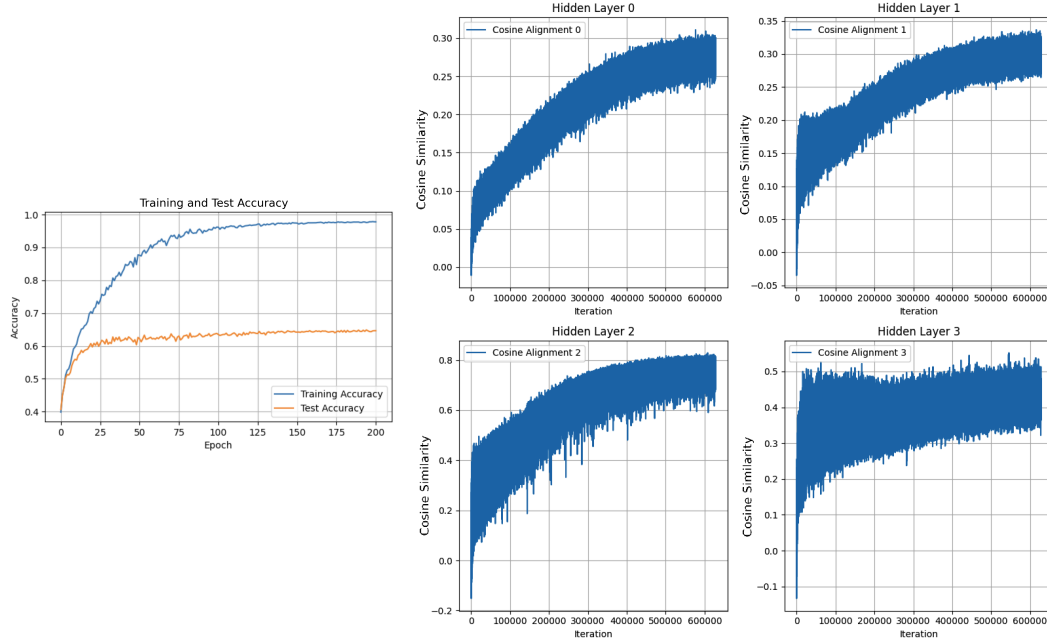


Figure 4: Cosine similarity between EBD updates and backpropagation gradients in a locally connected network on CIFAR-10. Positive alignment indicates directional consistency between EBD and BP.

These results demonstrate that EBD update directions are not arbitrary but align with the descent direction of the loss function as indicated by BP, supporting its effectiveness as a gradient-free but principled optimization strategy.

E.2 On gradient truncation and biological plausibility

The decorrelation objective used in EBD naturally decomposes into two sets of parameter updates per layer k :

$$(\Delta W_1^{(k)}, \Delta b_1^{(k)}) \quad \text{and} \quad (\Delta W_2^{(k)}, \Delta b_2^{(k)}).$$

Here, $(\Delta W_1^{(k)}, \Delta b_1^{(k)})$ corresponds to updates that modify the hidden representation to reduce correlation with the output error, while $(\Delta W_2^{(k)}, \Delta b_2^{(k)})$ corresponds to updates that aims to reshape the error signal itself.

For reasons of local learning and biological plausibility, EBD retains only the $(\Delta W_1^{(k)}, \Delta b_1^{(k)})$ component and drops the error-shaping terms $(\Delta W_2^{(k)}, \Delta b_2^{(k)})$, thereby avoiding the backward propagation of gradients through the network.

To assess the impact of this truncation, we measured the cosine similarity between the full gradient (which includes both components) and the truncated update used in EBD. As shown in Figure 5, the truncated update direction remains consistently aligned with the full gradient throughout training on CIFAR-10 using a 3-layer MLP. This positive alignment suggests that the retained component is sufficient for effective learning, validating our simplification.

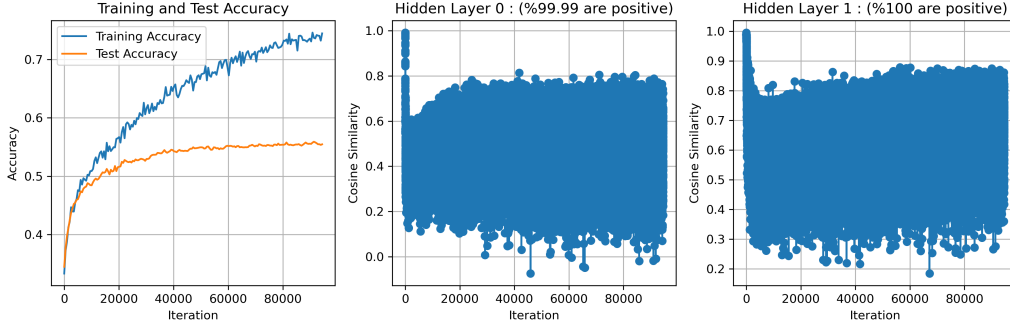


Figure 5: Cosine similarity between the full decorrelation gradient (including $(\Delta W_2^{(k)}, \Delta b_2^{(k)})$) and the truncated EBD update (only $(\Delta W_1^{(k)}, \Delta b_1^{(k)})$). Positive similarity confirms that the truncated update remains a valid descent direction.

F Background on online Correlative Information Maximization (CorInfoMax) based biologically plausible neural networks

Bozkurt et.al. recently proposed a framework, which we refer as CorInfoMax-EP, to address weight symmetry problem corresponding to backpropagation algorithm Bozkurt et al. [45]. In this section, we provide a brief summary of this framework.

The CorInfoMax-EP framework utilizes an online optimization setting to maximize correlative information between two consecutive layers:

$$\sum_{k=0}^{L-1} \hat{I}^{(\epsilon)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m] - \frac{\beta}{2} \|\mathbf{y}[m] - \mathbf{h}^{(L)}[m]\|_2^2,$$

where $\hat{I}^{(\epsilon)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m]$ is the correlative mutual information between layers k and $k+1$, and the term on the right corresponds to the mean square error between the network output $\mathbf{h}^{(L)}[m]$ and the training label $\mathbf{y}[m]$. This framework utilizes two alternative but equivalent forms for the correlative mutual information

$$\begin{aligned} \hat{I}_r^{(\epsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m] &= \frac{1}{2} \log \det(\hat{\mathbf{R}}_{\mathbf{h}^{(k+1)}}[m] + \varepsilon_k \mathbf{I}) - \frac{1}{2} \log \det(\hat{\mathbf{R}}_{\mathbf{e}_*^{\rightarrow(k+1)}}[m] + \varepsilon_k \mathbf{I}), \\ \hat{I}_l^{(\epsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m] &= \frac{1}{2} \log \det(\hat{\mathbf{R}}_{\mathbf{h}^{(k)}}[m] + \varepsilon_k \mathbf{I}) - \frac{1}{2} \log \det(\hat{\mathbf{R}}_{\mathbf{e}_*^{\leftarrow(k)}}[m] + \varepsilon_k \mathbf{I}), \end{aligned}$$

defined in terms of the correlation matrices of layer activations, i.e., $\hat{\mathbf{R}}_{\mathbf{h}^{(k)}}$ and the correlation matrices of forward and backward prediction errors ($\hat{\mathbf{R}}_{\mathbf{e}_*^{\rightarrow(k+1)}}$ and $\hat{\mathbf{R}}_{\mathbf{e}_*^{\leftarrow(k)}}$) between two consecutive layers. Here, forward/backward prediction errors are defined by

$$\mathbf{e}_*^{\rightarrow(k+1)}[n] = \mathbf{h}^{(k+1)}[n] - \mathbf{W}^{(f,k)}[m] \mathbf{h}^{(k)}[n], \quad \mathbf{e}_*^{\leftarrow(k)}[n] = \mathbf{h}^{(k)}[n] - \mathbf{W}^{(b,k)}[m] \mathbf{h}^{(k+1)}[n],$$

respectively. Here, $\mathbf{W}^{(f,k)}[m]$ ($\mathbf{W}^{(b,k)}[m]$) is the forward (backward) prediction matrix for layer k .

In order to enable online implementation, the exponentially weighted correlation matrices for hidden layer activations and prediction errors are defined as follows:

$$\begin{aligned} \hat{\mathbf{R}}_{\mathbf{h}^{(k)}}[m] &= \frac{1-\lambda}{1-\lambda^m} \sum_{i=1}^m \lambda^{m-i} \mathbf{h}^{(k)}[m] \mathbf{h}^{(k)}[m]^T, \\ \hat{\mathbf{R}}_{\mathbf{e}_*^{\rightarrow(k)}}[m] &= \frac{1-\lambda}{1-\lambda^m} \sum_{i=1}^m \lambda^{m-i} \mathbf{e}_*^{\rightarrow(k)}[m] \mathbf{e}_*^{\rightarrow(k)}[m]^T, \\ \hat{\mathbf{R}}_{\mathbf{e}_*^{\leftarrow(k)}}[m] &= \frac{1-\lambda}{1-\lambda^m} \sum_{i=1}^m \lambda^{m-i} \mathbf{e}_*^{\leftarrow(k)}[m] \mathbf{e}_*^{\leftarrow(k)}[m]^T. \end{aligned}$$

Through the trace approximation of $\log \det(\cdot)$ function, we obtain:

$$\begin{aligned} &\log \det \left(\hat{\mathbf{R}}_{\mathbf{e}_*^{\rightarrow(k+1)}}[m] + \varepsilon \mathbf{I} \right) \\ &\approx \frac{1}{\varepsilon_k} \sum_{i=1}^t \lambda^{t-i} \|\mathbf{h}^{(k+1)}[i] - \mathbf{W}_{ff,*}^{(k)}[m] \mathbf{h}^{(k)}[i]\|_2^2 + \varepsilon_k \|\mathbf{W}_{ff,*}^{(k)}[m]\|_F^2 + N_{k+1} \log(\varepsilon_k) \\ &\log \det \left(\hat{\mathbf{R}}_{\mathbf{e}_*^{\leftarrow(k)}}[m] + \varepsilon_k \mathbf{I} \right) \\ &\approx \frac{1}{\varepsilon_k} \sum_{i=1}^t \lambda^{t-i} \|\mathbf{h}^{(k)}[i] - \mathbf{W}_{fb,*}^{(k)}[m] \mathbf{h}^{(k+1)}[i]\|_2^2 + \varepsilon_k \|\mathbf{W}_{fb,*}^{(k)}[m]\|_F^2 + N_k \log(\varepsilon_k), \end{aligned}$$

F.1 The derivation of the CorInfoMax network

Based on the definitions above, the following layerwise objectives can be defined:

$$\hat{J}_k(\mathbf{h}^{(k)})[m] = \hat{I}_r^{(\epsilon_{k-1})}(\mathbf{h}^{(k-1)}, \mathbf{h}^{(k)})[m] + \hat{I}_l^{(\epsilon_k)}(\mathbf{h}^{(k)}, \mathbf{h}^{(k+1)})[m], \text{ for } k = 1, \dots, L-1,$$

i.e., correlative information maximization objectives for the hidden layers, and the mixture of correlation maximization and MSE objectives for the final layer

$$\hat{J}_L(\mathbf{h}^{(L)})[m] = \hat{I}_r^{(\epsilon_{L-1})}(\mathbf{h}^{(L-1)}, \mathbf{h}^{(L)})[m] - \frac{\beta}{2} \|\mathbf{h}^{(L)}[m] - \mathbf{y}[m]\|_2^2.$$

The gradient of the hidden layer objective functions with respect to the corresponding layer activations can be written as:

$$\nabla_{\mathbf{h}^{(k)}} \hat{J}_k(\mathbf{h}^{(k)})[m] = 2\gamma \mathbf{B}_{\mathbf{h}^{(k)}}[m] \mathbf{h}^{(k)}[m] - \frac{1}{\epsilon_{k-1}} \vec{\mathbf{e}}^{(k)}[m] - \frac{1}{\epsilon_k} \leftarrow{\mathbf{e}}^{(k)}[m], \quad (53)$$

where $\gamma = \frac{1-\lambda}{\lambda}$, and $\mathbf{B}_{\mathbf{h}^{(k)}}[m] = (\hat{\mathbf{R}}_{\mathbf{h}^{(k)}}[m] + \epsilon_{k-1} \mathbf{I})^{-1}$, i.e., the inverse of the layer correlation matrix.

For the output layer, we can write the gradient as

$$\nabla_{\mathbf{h}^{(L)}} \hat{J}_L(\mathbf{h}^{(L)})[m] = \gamma \mathbf{B}_{\mathbf{h}^{(L)}}[m] \mathbf{h}^{(L)}[m] - \frac{1}{\epsilon_{L-1}} \vec{\mathbf{e}}^{(L)}[m] - \beta(\mathbf{h}^{(L)}[m] - \mathbf{y}[m]).$$

The gradient ascent updates corresponding to these expressions can be organized to obtain CorInfoMax network dynamics:

$$\begin{aligned} \tau_{\mathbf{u}} \frac{d\mathbf{u}^{(k)}[m; s]}{ds} &= -g_{lk} \mathbf{u}^{(k)}[m; s] + \frac{1}{\epsilon_k} \mathbf{M}^{(k)}[m] \mathbf{h}^{(k)}[m; s] - \frac{1}{\epsilon_{k-1}} \vec{\mathbf{e}}_u^{(k)}[m; s] - \frac{1}{\epsilon_k} \leftarrow{\mathbf{e}}_u^{(k)}[m; s], \\ \vec{\mathbf{e}}_u^{(k)}[m; s] &= \mathbf{u}^{(k)}[m; s] - \mathbf{W}_{ff}^{(k-1)}[t] \mathbf{h}^{(k-1)}[m; s], \\ \leftarrow{\mathbf{e}}_u^{(k)}[m; s] &= \mathbf{u}^{(k)}[m; s] - \mathbf{W}_{fb}^{(k)}[m] \mathbf{h}^{(k+1)}[m; s], \\ \mathbf{h}^{(k)}[m; s] &= \sigma_+(\mathbf{u}^{(k)}[m; s]), \end{aligned}$$

where m is the sample index, s is the time index for the network dynamics, $\tau_{\mathbf{u}}$ is the update time constant, $\mathbf{M}^{(k)}[t] = \epsilon_k(2\gamma \mathbf{B}_{\mathbf{h}^{(k)}}[t] + g_{lk} \mathbf{I})$, and $\sigma_+ = \min(1, \max(u, 0))$ represents the elementwise clipped-ReLU function, which is the projection operation corresponding to the combination of the nonnegativity constraint $\mathbf{h}^{(k)} \geq 0$ and the boundedness constraint $\|\mathbf{h}^{(k)}\|_{\infty} \leq 1$ on the activations of the network.

Note that Bozkurt et al. [45] takes one more step to organize the network dynamics into a form that fits into the form of a network with three compartment (soma, basal dendrite and apical dendrite compartments) neuron model.

F.2 CorInfoMax-EP learning dynamics

The CorInfoMax-EP framework in Bozkurt et al. [45] employs equilibrium propagation (EP) to update feedforward and feedback weights of the CorInfoMax network.

F.2.1 Feedforward and feedback weights

In the CorInfoMax objective, feedforward and feedback weights correspond to forward and backward predictors corresponding to the regularized least squares objectives

$$C_{ff}(\mathbf{W}_{ff}^{(k)}[m]) = \epsilon_k \|\mathbf{W}_{ff}^{(k)}[m]\|_F^2 + \|\vec{\mathbf{e}}^{(k+1)}[m]\|_2^2,$$

and

$$C_{fb}(\mathbf{W}_{fb}^{(k)}[m]) = \epsilon_k \|\mathbf{W}_{fb}^{(k)}[m]\|_F^2 + \|\leftarrow{\mathbf{e}}^{(k)}[m]\|_2^2,$$

respectively. The derivatives of these functions with respect to forward and backward synaptic weights can be written as

$$\frac{\partial C_{ff}(\mathbf{W}_{ff}^{(k)}[m])}{\partial \mathbf{W}_{ff}^{(k)}[m]} = 2\epsilon_k \mathbf{W}_{ff}^{(k)}[m] - 2\vec{\mathbf{e}}^{(k+1)}[m] \mathbf{h}^{(k)}[m]^T,$$

and

$$\frac{\partial C_{fb}(\mathbf{W}_{fb}^{(k)}[m])}{\partial \mathbf{W}_{fb}^{(k)}[m]} = 2\varepsilon_k \mathbf{W}_{fb}^{(k)}[m] - 2\mathbf{e}^{\leftarrow(k)}[m] \mathbf{h}^{(k+1)}[m]^T.$$

The EP based updates of the feedforward and feedback weights are obtained by evaluating these gradients in two different phases: the nudge phase ($\beta = \beta' > 0$), and the free phase ($\beta = 0$):

$$\begin{aligned} \delta \mathbf{W}_{ff}^{(k)}[m] &\propto \frac{1}{\beta'} \left((\mathbf{e}^{\rightarrow(k+1)}[m] \mathbf{h}^{(k)}[m]^T) \Big|_{\beta=\beta'} - (\mathbf{e}^{\rightarrow(k+1)}[m] \mathbf{h}^{(k)}[m]^T) \Big|_{\beta=0} \right), \\ \delta \mathbf{W}_{fb}^{(k)}[m] &\propto \frac{1}{\beta'} \left((\mathbf{e}^{\leftarrow(k)}[m] \mathbf{h}^{(k+1)}[m]^T) \Big|_{\beta=\beta'} - (\mathbf{e}^{\leftarrow(k)}[m] \mathbf{h}^{(k+1)}[m]^T) \Big|_{\beta=0} \right). \end{aligned}$$

F.2.2 Lateral weights

The lateral weight updates derived from the weight correlation matrices of the layer activations, using the matrix inversion lemma [43]:

$$\mathbf{B}^{(k)}[m+1] = \lambda_r^{-1}(\mathbf{B}^{(k)}[m] - \gamma \mathbf{z}^{(k)}[m] \mathbf{z}^{(k)}[m]^T), \text{ where } \mathbf{z}^{(k)}[m] = \mathbf{B}^{(k)}[m] \mathbf{h}^{(k)}[m] \Big|_{\beta=\beta'}.$$

F.3 CorInfoMax-EP

Although the CorInfoMax-EP algorithm derivation above is based on single input sample based updates, it can be extendable to batch updates. Assuming a batch size of B , and we define the following matrices:

$$\mathbf{H}^{(k)}[m] = \begin{bmatrix} \mathbf{h}^{(k)}[mB+1] & \mathbf{h}^{(k)}[mB+2] & \dots & \mathbf{h}^{(k)}[(m+1)B] \end{bmatrix},$$

as the activation matrix for the layer- k ,

$$\mathbf{E}^{\leftarrow(k)}[m] = \begin{bmatrix} \mathbf{e}^{\leftarrow(k)}[mB+1] & \mathbf{e}^{\leftarrow(k)}[mB+2] & \dots & \mathbf{e}^{\leftarrow(k)}[(m+1)B] \end{bmatrix}, \quad (54)$$

as the backward prediction matrix for the layer- k ,

$$\mathbf{E}^{\rightarrow(k)}[m] = \begin{bmatrix} \mathbf{e}^{\rightarrow(k)}[mB+1] & \mathbf{e}^{\rightarrow(k)}[mB+2] & \dots & \mathbf{e}^{\rightarrow(k)}[(m+1)B] \end{bmatrix}, \quad (55)$$

as the forward prediction matrix for the layer- k ,

$$\mathbf{Z}^{(k)}[m] = \begin{bmatrix} \mathbf{z}^{(k)}[mB+1] & \mathbf{z}^{(k)}[mB+2] & \dots & \mathbf{z}^{(k)}[(m+1)B] \end{bmatrix}, \quad (56)$$

as the lateral weights' output matrix for the layer- k , and

$$\mathbf{E} = \begin{bmatrix} \boldsymbol{\epsilon}[mB+1] & \boldsymbol{\epsilon}[mB+2] & \dots & \boldsymbol{\epsilon}[(m+1)B] \end{bmatrix},$$

as the output error matrix.

In terms of these definitions, Algorithm 2 lays out the details of the CorInfoMax-EP algorithm:

Algorithm 2 CorInfoMax Equilibrium Propagation (CorInfoMax-EP) Update for Layer k

Require: Learning rate parameters $\lambda_E, \mu^{(f,k)}[m], \mu^{(b,k)}[m]$

Require: Previous synaptic weights $\mathbf{W}^{(f,k)}[m-1]$ (forward), $\mathbf{W}^{(b,k)}[m-1]$ (backward), $\mathbf{B}^{(k)}$ (lateral)

Require: Batch size B

Require: Layer activations $\mathbf{H}^{(k)}[m]$, preactivations $\mathbf{U}^{(k)}[m]$, output errors $\mathbf{E}^{(k)}[m]$, lateral weight outputs $\mathbf{Z}^{(k)}[m]$, forward prediction errors $\vec{\mathbf{E}}^{(k)}[m]$ and backward prediction errors $\leftarrow{\mathbf{E}}^{(k)}[m]$ computed by CorInfoMax network dynamics described in Bozkurt et al. [45]

Ensure: Updated weights $\mathbf{W}^{(f,k)}[m], \mathbf{W}^{(b,k)}[m], \mathbf{B}^{(k)}[m]$

1: $\gamma_E \leftarrow \frac{1-\lambda_E}{\lambda_E}$

Update forward weights for layer k :

2: $\Delta \mathbf{W}_{\text{EP}}^{(f,k)}[m] \leftarrow -\frac{\mu^{(d_f,k)}[m]}{B\beta'} \left(\left(\vec{\mathbf{E}}^{(k+1)}[m] \mathbf{H}^{(k)}[m]^T \right) \Big|_{\beta=\beta'} - \left(\vec{\mathbf{E}}^{(k+1)}[m] \mathbf{H}^{(k)}[m]^T \right) \Big|_{\beta=0} \right)$

3: $\mathbf{W}^{(f,k)}[m] \leftarrow \mathbf{W}^{(f,k)}[m-1] + \Delta \mathbf{W}_{\text{EP}}^{(f,k)}[m]$

Update backward weights for layer k :

4: $\Delta \mathbf{W}_{\text{EP}}^{(b,k)}[m] \leftarrow -\frac{\mu^{(d_b,k)}[m]}{B\beta'} \left(\left(\leftarrow{\mathbf{E}}^{(k)}[m] \mathbf{H}^{(k)}[m]^T \right) \Big|_{\beta=\beta'} - \left(\leftarrow{\mathbf{E}}^{(k)}[m] \mathbf{H}^{(k)}[m]^T \right) \Big|_{\beta=0} \right)$

5: $\mathbf{W}^{(b,k)}[m] \leftarrow \mathbf{W}^{(b,k)}[m-1] + \Delta \mathbf{W}_{\text{EP}}^{(b,k)}[m]$

Update Lateral weights for layer k :

6: $\Delta \mathbf{B}_E^{(k)}[m] \leftarrow -\frac{\gamma_E}{B} \mathbf{Z}^{(k)}[m] \mathbf{Z}^{(k)}[m]^T$

7: $\mathbf{B}^{(k)}[m] \leftarrow \frac{1}{\lambda_E} \mathbf{B}^{(k)}[m] + \Delta \mathbf{B}_E^{(k)}[m]$

G Implementation complexity of the EBD approach

In this section, we analyze the computational and memory complexity trade-offs of the proposed Error Broadcast and Decorrelation (EBD) approach.

G.1 Complexity analysis: Error Propagation vs. Error Broadcast

Considering the standard MLP implementation outlined in Section 2, we compare the memory and computational requirements of the error backpropagation and error broadcast approaches as follows:

G.1.1 Delivering output error information to layers

- **Memory Requirements:** For the standard backpropagation algorithm, the error is propagated through the transposed forward filters $\mathbf{W}^{(k)}$. Consequently, no additional memory is required for the weights used in error propagation. In contrast, the error broadcast approach uses error projection matrices $\hat{\mathbf{R}}_{\mathbf{g}_\epsilon}^{(k)}$, which require memory storage of $\mathcal{O}(N^{(k)}N^{(L)})$ for each layer. Thus, the total additional storage requirement for broadcast weights is given by $\mathcal{O}((\sum_{l=1}^L N^{(l)})N^{(L)})$.
- **Computational Requirements:** In the standard backpropagation algorithm, propagating the error to layer k (from layer $k+1$) requires $\mathcal{O}(BN^{(k)}N^{(k+1)})$ multiplications per batch, where B is the batch size. On the other hand, the broadcast algorithm projects the output error to layer k , requiring $\mathcal{O}(BN^{(k)}N^{(L)})$ multiplications. Additionally, the projection matrix $\hat{\mathbf{R}}_{\mathbf{g}_\epsilon}^{(k)}$ is updated at the end of each batch using the Hebbian rule. Therefore, the overall computational complexity for the EBD approach is:

$$\mathcal{O}((B+1)N^{(k)}N^{(L)}).$$

When the number of output elements $N^{(L)}$ is significantly smaller than the hidden dimensions $N^{(k)}$, the computational cost of the broadcast algorithm is lower. Furthermore, the error projection

operations can be implemented in parallel for all layers, whereas backpropagation must be executed sequentially.

G.1.2 Additional cost of Entropy Regularization

- **Memory requirements:** As described in Section 2.4.1, layer entropies are based on the covariance matrix $\mathbf{R}_h^{(k)}$, which requires $\mathcal{O}((N^{(k)})^2)$ additional memory storage. In a computationally optimized implementation, storing the inverse covariance matrix $\mathbf{B}_h^{(k)} = \mathbf{R}_h^{(k)-1}$ may be preferred, though it still requires the same memory allocation.
- **Computational requirements:** The main computational load is due to the computation of the gradient of the layer entropy function. The expression for the gradient is obtained in Appendix C.3 as

$$\nabla_{\mathbf{W}^{(k)}} J^{(k)}[m] = 2 \frac{1-\lambda_E}{B} \left[(\mathbf{R}_h^{(k)}[m] + \epsilon \mathbf{I})^{-1} \mathbf{H}^{(k)}[m] \odot f'(\mathbf{W}^{(k)} \mathbf{H}^{(k-1)}[m] + \mathbf{b}^{(k)} \mathbf{1}^T) \right] (\mathbf{H}^{(k-1)}[m])^T,$$

where for each batch, we update the layer correlation matrix matrix $\mathbf{R}_h^{(k)}[m]$. We can divide the computational requirement into following pieces:

– Correlation Matrix Recursion

Recall we have

$$\mathbf{R}_h^{(k)}[m] = \lambda_E \mathbf{R}_h^{(k)}[m-1] + \frac{1-\lambda_E}{B} \mathbf{H}^{(k)}[m] (\mathbf{H}^{(k)}[m])^T.$$

In order to form $\mathbf{R}_h^{(k)}[m]$ explicitly at iteration m , we must compute $\mathbf{H}^{(k)}[m] (\mathbf{H}^{(k)}[m])^T$ and then add the result to $\lambda_E \mathbf{R}_h^{(k)}[m-1]$.

- * The matrix–matrix product $\mathbf{H}^{(k)}[m] \in \mathbb{R}^{N^{(k)} \times B}$ times its transpose in $\mathbb{R}^{B \times N^{(k)}}$ yields an $N^{(k)} \times N^{(k)}$ matrix, costing $\mathcal{O}(N^{(k)2} B)$.
- * Adding $\lambda_E \mathbf{R}_h^{(k)}[m-1]$ to that product is another $\mathcal{O}((N^{(k)})^2)$ operation, though usually smaller in comparison to the product above if B is moderate.

Hence the complexity of *forming the new correlation matrix* $\mathbf{R}_h^{(k)}[m]$ at each iteration is

$$\mathcal{O}(N^{(k)2} B).$$

– Naive Matrix Inversion and Gradient Computation

Once $\mathbf{R}_h^{(k)}[m]$ is formed, we need to invert $\mathbf{R}_h^{(k)}[m] + \epsilon \mathbf{I}$ to evaluate $J^{(k)}[m]$ and its gradient. Naive inversion of an $N^{(k)} \times N^{(k)}$ matrix is $\mathcal{O}((N^{(k)})^3)$. After this inversion, we multiply $(\mathbf{R}_h^{(k)}[m] + \epsilon \mathbf{I})^{-1}$ by $\mathbf{H}^{(k)}[m] \in \mathbb{R}^{N^{(k)} \times B}$, which costs $\mathcal{O}((N^{(k)})^2 B)$. Next, we do the elementwise multiplication ($\odot$) with $f'(\mathbf{W}^{(k)} \mathbf{H}^{(k-1)}[m] + \mathbf{b}^{(k)} \mathbf{1}^T)$, costing $\mathcal{O}(N^{(k)} B)$. Finally, we multiply by $(\mathbf{H}^{(k-1)}[m])^T \in \mathbb{R}^{B \times N^{(k-1)}}$, which costs $\mathcal{O}(N^{(k)} B N^{(k-1)})$.

Summing all these terms, the dominant operations in *naive* update and inversion are:

1. Forming $\mathbf{R}_h^{(k)}[m]$: $\mathcal{O}(N^{(k)2} B)$.
2. Inverting $(\mathbf{R}_h^{(k)}[m] + \epsilon \mathbf{I})$: $\mathcal{O}((N^{(k)})^3)$.
3. Multiplying inverse by $\mathbf{H}^{(k)}[m]$: $\mathcal{O}((N^{(k)})^2 B)$.
4. Final multiplication by $(\mathbf{H}^{(k-1)}[m])^T$: $\mathcal{O}(N^{(k)} B N^{(k-1)})$.

Thus, overall cost per batch is

$$\mathcal{O}(N^{(k)2} B + (N^{(k)})^3 + (N^{(k)})^2 B + N^{(k)} B N^{(k-1)}),$$

which could be simplified to

$$\mathcal{O}((N^{(k)})^3 + (N^{(k)})^2 B + N^{(k)} B N^{(k-1)}).$$

If $N^{(k)}$ is large, the cubic term $(N^{(k)})^3$ associated with the matrix inversion typically dominates.

An alternative is to update the inverse of the correlation matrix *incrementally* using the fact that

$$\mathbf{R}_h^{(k)}[m] = \lambda_E \mathbf{R}_h^{(k)}[m-1] + \frac{1-\lambda_E}{B} \mathbf{H}^{(k)}[m](\mathbf{H}^{(k)}[m])^T,$$

so the new correlation matrix differs from the previous one by a low-rank term of rank at most $\min(N^{(k)}, B)$. Neglecting the ϵ term, the recursion for the inverse can be obtained using matrix inversion lemma [43] as

$$\mathbf{B}_h^{(k)}[m] = \lambda_E^{-1} \left(\mathbf{B}_h^{(k)}[m-1] - \mathbf{V}[m] \left(\frac{\lambda_E B}{1-\lambda_E} \mathbf{I} + \mathbf{H}^{(k)}[m]^T \mathbf{V}[m] \right)^{-1} \mathbf{V}[m]^T \right), \quad (57)$$

where $\mathbf{V}[m] = \mathbf{B}_h^{(k)}[m-1] \mathbf{H}^{(k)}[m]$.

The corresponding computational cost is:

- Two matrix-matrix multiplications of shape $(N^{(k)} \times N^{(k)})$ with $(N^{(k)} \times B)$, costing $\mathcal{O}((N^{(k)})^2 B)$.
- Inverting the $B \times B$ matrix, costing $\mathcal{O}(B^3)$ if B is not too large.

Thus, the update of the inverse alone costs

$$\mathcal{O}((N^{(k)})^2 B + B^3),$$

instead of $\mathcal{O}((N^{(k)})^3)$. If $B \ll N^{(k)}$, this can be a large savings compared to the naive cubic cost. Once this updated inverse is in hand, the subsequent multiplications to form the gradient (e.g. $(\mathbf{R}_h^{(k)}[m] + \epsilon \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]$, etc.) still take $\mathcal{O}((N^{(k)})^2 B + N^{(k)} B N^{(k-1)})$. Overall, for each new time step m , the dominant costs become

$$\mathcal{O}((N^{(k)})^2 B + B^3 + (N^{(k)})^2 B + N^{(k)} B N^{(k-1)}),$$

which is usually simplified to

$$\mathcal{O}((N^{(k)})^2 B + B^3 + N^{(k)} B N^{(k-1)}).$$

Hence, using the Woodbury identity is beneficial whenever B is much smaller than $N^{(k)}$, because $N^{(k)2} B + B^3 \ll (N^{(k)})^3$.

G.1.3 Additional cost of Power-normalization

- **Memory requirements:** The power normalization described in Section 2.4.1, involves a power estimate parameter per hidden unit, so it will require additional $N^{(k)}$ storage elements for the layer- k .
- **Computational requirements:** The gradient for the power normalization regularization function derived in Appendix C.4 takes the form:

$$\nabla_{\mathbf{W}^{(k)}} J_P^{(k)}[m] = \frac{4}{B} \mathbf{D}[m] (\mathbf{H}^{(k)}[m] \odot \mathbf{F}_d^{(k)}) \mathbf{H}^{(k-1)}[m]^T,$$

Based on this expression, the required number of operations per batch for layer- k is $\mathcal{O}(B N^{(k)} N^{(k-1)})$.

Therefore, we can consider the impact of the power-normalization on memory and computational requirements as negligible.

In Section I.7, we provide empirical runtime results for the EBD algorithm, relative to the backpropagation algorithm. These experimental results show a 7 to 8 time increase in the runtimes of the MLP model with the EBD algorithm (employing entropy regularization) relative to the BP algorithm. The runtime increase is less for CNN and LC models.

Finally, we note that the implementation complexity analysis provided above is for the MLP based EBD approach. For the biologically more realistic CorInfoMax-EBD networks, entropy maximization is implemented through lateral weights (see Appendices F, F.2.2 and H), whose update requires $\mathcal{O}((N^{(k)})^2)$ multiplications per sample.

H On the biologically plausible nature of Entropy and Power-normalization updates

As discussed in Section 2.4.1, the layer-entropy and power-normalization objectives are introduced to avert potential collapse of network coefficients in the EBD algorithm. A natural question arises regarding the biological plausibility of the EBD framework when these regularizations are incorporated. We address this question by examining two specific cases:

1. **MLP Implementation with Entropy and Power-normalization regularizations (Section 2)**
2. **CorInfoMax-EBD implementation (Section 3.2)**

H.1 MLP implementation with Entropy and Power-normalization regularizations

In Section 2, we presented an MLP-based EBD framework that uses batch-SGD to optimize the feedforward weights with EBD, along with entropy and power-normalization losses. As outlined in Sections 2.3 and 3.1, the gradient-based updates of the EBD loss naturally reduce to a three-factor update rule, which is considered biologically plausible. We now examine whether adding the layer-entropy objective in Eq. (11) and the power-normalization objective in Eq. (9) preserves this biological realism.

H.1.1 Power normalization-based SGD updates

Appendix C.4 derives the gradient expression for the power-normalization loss:

$$\nabla_{\mathbf{W}^{(k)}} J_P^{(k)}[m] = \frac{4}{B} \mathbf{D}[m] (\mathbf{H}^{(k)}[m] \odot \mathbf{F}_d^{(k)}[m]) \mathbf{H}^{(k-1)}[m]^T.$$

Focusing on an individual element of this matrix gives

$$[\nabla_{\mathbf{W}^{(k)}} J_P^{(k)}[m]]_{ij} = \frac{4}{B} d_i[m] \sum_{n=mB+1}^{mB+B} h_i^{(k)}[n] f_i^{(k)'}(u_i^{(k)}[n]) h_j^{(k-1)}[n]. \quad (58)$$

This update depends only on the activations of the neurons connected by the synapse W_{ij} , thus satisfying a local learning rule. However, the summation over the batch index in Eq. (58) might be considered biologically implausible unless the batch size $B = 1$. In practice, one can interpret the summation for $B > 1$ as an integral of local updates over the time window corresponding to the batch, which may still be reasonably viewed as local integration in a biological setting.

H.1.2 Layer entropy regularization-based SGD updates

Appendix C.3 derives the gradient of the layer-entropy objective:

$$\nabla_{\mathbf{W}^{(k)}} J_E^{(k)}[m] = \frac{2(1 - \lambda_E)}{B} \left[((\mathbf{R}_h^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{H}^{(k)}[m]) \odot \mathbf{F}_d^{(k)}[m] \right] \mathbf{H}^{(k-1)}[m]^T. \quad (59)$$

Examining an individual element of this gradient shows

$$[\nabla_{\mathbf{W}^{(k)}} J_E^{(k)}[m]]_{ij} = \frac{2(1 - \lambda_E)}{B} \sum_{n=mB+1}^{mB+B} v_i^{(k)}[n] f_i^{(k)'}(u_i^{(k)}[n]) h_j^{(k-1)}[n],$$

where $\mathbf{v}^{(k)}[n] = (\mathbf{R}_h^{(k)}[m] + \varepsilon^{(k)} \mathbf{I})^{-1} \mathbf{h}^{(k)}[n]$. Because $v_i^{(k)}[n]$ depends on all neurons' activations in the layer, the layer-entropy update for feedforward weights is not strictly local and hence violates the criteria for strict biological plausibility.

Nevertheless, this limitation is circumvented by the CorInfoMax-EBD approach, wherein the lateral (recurrent) weights, rather than the feedforward weights, implement the layer-entropy maximization objective. We discuss this in the next section.

H.2 CorInfoMax-EBD implementation

Section 3.2 introduces a more biologically realistic network by combining the CorInfoMax framework—known to yield recurrent networks closely reflecting biological dynamics—with the proposed EBD approach to enable a three-factor update rule in supervised learning.

H.2.1 Power-normalization-based SGD updates

In CorInfoMax-EBD, we adopt the same power-normalization gradient in Eq. (58) for updating feedforward weights. Therefore, by setting the batch size to $B = 1$, these updates remain local and thus biologically plausible.

H.2.2 Layer entropy maximization

As summarized in Appendix F, CorInfoMax networks inherently include layer-entropy maximization via the correlative-information objective. Crucially, this entropy maximization is implemented through *lateral weights* of the RNN structure rather than by modifying feedforward weights. Specifically, from the gradient of the correlative-information objective (see Eq. (53)):

$$\nabla_{\mathbf{h}^{(k)}} \hat{J}_k(\mathbf{h}^{(k)})[m] = 2\gamma \mathbf{B}_{\mathbf{h}^{(k)}}[m] \mathbf{h}^{(k)}[m] - \frac{1}{\epsilon_{k-1}} \vec{\mathbf{e}}^{(k)}[m] - \frac{1}{\epsilon_k} \overleftarrow{\mathbf{e}}^{(k)}[m], \quad (60)$$

the first term, $2\gamma \mathbf{B}_{\mathbf{h}^{(k)}}[m] \mathbf{h}^{(k)}[m]$, corresponds to the layer-entropy maximization. Here, the lateral weight matrix $\mathbf{B}_{\mathbf{h}^{(k)}}[m]$ approximates the inverse of the layer correlation matrix. As described in Appendix F.2.2 (and in [45]), the lateral weights can be updated by an anti-Hebbian rule:

$$\mathbf{B}^{(k)}[m+1] = \lambda_{\mathbf{r}}^{-1} \left(\mathbf{B}^{(k)}[m] - \gamma \mathbf{z}^{(k)}[m] \mathbf{z}^{(k)}[m]^T \right),$$

where

$$\mathbf{z}^{(k)}[m] = \mathbf{B}^{(k)}[m] \mathbf{h}^{(k)}[m] \Big|_{\beta=\beta'}.$$

Once again, this update is strictly local if $B = 1$, while for $B > 1$, the rank- B extension may break strict locality. We demonstrate in Section 4 that CorInfoMax-EBD with $B = 1$ yields comparable or superior performance to CorInfoMax-EP with larger batch sizes.

H.3 Summary and conclusions

In summary, the CorInfoMax-EBD implementation described in Section 3.2 offers a more biologically plausible approach to supervised learning compared to the MLP-based EBD approach in Section 2 due to several factors:

- Using **lateral weights** to impose layer-entropy maximization in a biologically realistic manner;
- Employing **feedforward/feedback weights** for forward and backward predictive coding;
- Adopting **neuron models** with distinct compartments (soma, basal dendrites, and apical dendrites);
- Incorporating **EBD updates**, which naturally embody a three-factor learning rule; and
- Leveraging **power-normalization updates**, which satisfy local-learning constraints when $B = 1$.

These features stem from the CorInfoMax-EP framework [45], enhanced by our proposed EBD-based regularizations. This architecture reconciles the benefits of layer-entropy and power-normalization objectives with the demands of biological plausibility.

I Supplementary on numerical experiments

The models were trained on an NVIDIA Tesla V100 GPU, using the hyperparameters detailed in the sections below. Each experiment was conducted five times under identical settings, and the reported results reflect the average performance. We used the standard train/test splits for the datasets, with MNIST comprising 60,000 training examples and CIFAR-10 comprising 50,000, while both datasets included 10,000 test examples. The MNIST dataset [56] is made available under the Creative Commons Attribution-Share Alike 3.0 license. The CIFAR-10 dataset [45], originating from the University of Toronto, is publicly available for academic research purposes. Both datasets were accessed via standard deep learning library functionalities.

Rather than utilizing automatic differentiation tools, we manually implemented the gradient calculations for the EBD algorithm, utilizing batched operations to ensure computational efficiency. As a side note, the $(1 - \lambda)$ factors present in the derived update expressions are absorbed into the learning rate constants and thus eliminated. In our experiments, we trained the MLP models for **120** epochs and the CNN and LC models for **100** epochs on MNIST and **200** epochs on CIFAR-10. In addition, we trained the CNN model for the CIFAR-100 dataset for **300** epochs, and the CorInfoMax-EBD (3-layer, batch size = 20) model for **60** epochs.

I.1 Architectures

The architectural details of MLP, CNN and LC networks for the MNIST and CIFAR-10 datasets are shown in Tables 4 and 5, respectively, while the CNN model used in the CIFAR-100 experiments is detailed in Table 7. The structure of the MNIST and CIFAR-10 models are the same as in the reference [9], while the CIFAR-100 model closely matches [6], differing only in the MaxPool shape. In all architectures, we used ReLU as the nonlinear functions except the last layer. Furthermore, the architectural details of the biologically more realistic CorInfoMax network for MNIST and CIFAR-10 datasets are shown in Table 6. These techniques are the same as examples in Appendix J.5 of [45].

Table 4: MNIST architectures. **FC**: fully connected; **Conv**: convolutional; **LC**: locally connected. FC layers are reported by hidden size. Conv/LC layers are reported as (channels, kernel size, stride, padding). Pooling layers use stride 1; we report the kernel size.

MLP		Convolutional		Locally connected	
FC1	1024	Conv1	64, 3×3 , 1, 1	LC1	32, 3×3 , 1, 1
FC2	512	AvgPool	2×2	AvgPool	2×2
		Conv2	32, 3×3 , 1, 1	LC2	32, 3×3 , 1, 1
		AvgPool	2×2	AvgPool	2×2
		FC1	1024	FC1	1024

Table 5: CIFAR-10 architectures. Conventions are the same as in Table 4.

MLP		Convolutional		Locally connected	
FC1	1024	Conv1	128, 5×5 , 1, 2	LC1	64, 5×5 , 1, 2
FC2	512	AvgPool	2×2	AvgPool	2×2
FC3	512	Conv2	64, 5×5 , 1, 2	LC2	32, 5×5 , 1, 2
		AvgPool	2×2	AvgPool	2×2
		Conv3	64, 2×2 , 2, 0	LC3	32, 2×2 , 2, 0
		FC1	1024	FC1	512

Table 6: CorInfoMax architectures. Conventions are the same as in Table 4.

MNIST		CIFAR-10	
FC1	500	FC1	1000
FC2	500	FC2	500

Table 7: CIFAR-100 architectures. Conventions are the same as in Table 4.

Convolutional	
Conv1	96, 5×5 , 1, 2
MaxPool	2×2
Conv2	128, 5×5 , 1, 2
MaxPool	2×2
Conv3	256, 5×5 , 1, 2
MaxPool	2×2
Dropout	p
FC1	2048
Dropout	p
FC2	2048
Softmax	100

I.2 CorInfoMax-EBD

In this section, we offer additional details regarding the numerical experiments conducted with the CorInfoMax Error Broadcast and Decorrelation (CorInfoMax-EBD) algorithm. Appendix I.2.1 elaborates on the general implementation details. Appendix I.2.2 presents the fundamental learning steps of the algorithm, which are based on the EBD method. Appendices I.2.3 and I.2.4 discuss the initialization of the algorithm’s variables and describe the hyperparameters. Finally, Appendix I.2.5 (3-Layer and batch size=20, 3-Layer batch size=1) and Appendix I.2.6 (10-Layer batch size=1) detail the specific hyperparameter configurations used in our numerical experiments for the MNIST and CIFAR-10 datasets. In Appendix I.10 we present the accuracy and loss learning curves for the CorInfoMax-EBD, shown in Figures 6.(g)-(h) and Figures 7.(g)-(h), respectively.

I.2.1 Implementation details

We implemented the CorInfoMax-EBD algorithm based on the repository available at GitHub². This repository from Bozkurt et al. [45], used as a basis for our CorInfoMax-EBD implementation, did not specify an explicit license in its public repository at the time of access. Our use and modification are for academic research purposes, building upon the published scientific work presented in [45]. The following modifications were made to the original code:

- **Reduction to a single phase:** We simplified the algorithm by reducing it to a single phase. Specifically, we removed the nudge phase, during which the label is coupled to the network dynamics. In this modified version, the network operates solely in the free phase, where the label is decoupled from the network. This change aligns with the removal of time-contrastive updates from the CorInfoMax-EP algorithm.
- **Algorithmic updates:** We incorporated the updates outlined in Algorithm 3.
- **Hyperparameters:** We maintained the same hyperparameters for the neural dynamics as in the original code. Additionally, new hyperparameters specific to the learning dynamics were introduced, which are detailed in Appendix I.2.4.

In the CorInfoMax-EBD implementation the following loss and regularization functions are used

- EBD loss: $J^{(k)}$,
- Power normalization loss: $J_P^{(k)}$,
- ℓ_2 weight regularization (weight decay): $J_{\ell_2}^{(k)}$,
- Activation sparsity regularization: $J_{\ell_1}^{(k)} = \|\mathbf{H}^{(k)}\|_1$.

²<https://github.com/BariscanBozkurt/Supervised-CorInfoMax>

I.2.2 Algorithm

The CorInfoMax-EBD algorithm follows the same neural dynamics framework detailed in [45] for computing neuron activations. Consequently, we only outline the steps specific to the learning process, which distinguishes it from the original CorInfoMax-EP algorithm described in [45]. The full iterative process for updating weights in the CorInfoMax-EBD algorithm is provided in Algorithm 3.

Algorithm 3 CorInfoMax Error Broadcast and Decorrelation (CorInfoMax-EBD) Update for Layer k

- Require:** Learning rate parameters $\lambda_d, \lambda_E, \mu^{(d,k)}[m], \mu^{(f,k)}[m], \mu^{(b,k)}[m]$
Require: Previous synaptic weights $\mathbf{W}^{(f,k)}[m-1]$ (forward), $\mathbf{W}^{(b,k)}[m-1]$ (backward), $\mathbf{B}^{(k)}[m-1]$ (lateral)
Require: Previous error projection weights $\mathbf{R}_{g(\mathbf{h}^{(k)})\epsilon}[m-1]$
Require: Batch size B
Require: Layer activations $\mathbf{H}^{(k)}[m]$ in Eq. (13), the derivative of activations $\mathbf{F}_d^{(k)}$ in Eq. (29), in Eq. (5), prediction errors $\overset{\leftarrow}{\mathbf{E}}$ and $\overset{\rightarrow}{\mathbf{E}}^{(k)}$ in Eq. (54)-Eq. (55), lateral weight outputs $\mathbf{Z}^{(k)}$ in Eq. (56) computed by CorInfoMax network dynamics described in Bozkurt et al. [45] (and Appendix F)
Require: The nonlinear function of layer activations $\mathbf{G}^{(k)}$ in Eq. (4) and the derivative of the nonlinear function of layer activations $\mathbf{G}_d^{(k)}$ in Eq. (28)
Ensure: Updated weights $\mathbf{W}^{(f,k)}[m], \mathbf{W}^{(b,k)}[m], \mathbf{B}^{(k)}[m]$

Error projection weight update for layer k :

$$1: \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \leftarrow \lambda_d \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m-1] + \frac{1-\lambda_d}{B} \mathbf{G}^{(k)}[m] \mathbf{E}^{(k)}[m]^T$$

Project errors to layer k :

$$2: \mathbf{Q}^{(k)}[m] \leftarrow \hat{\mathbf{R}}_{\mathbf{g}\epsilon}^{(k)}[m] \mathbf{E}^{(k)}[m]$$

Find the gradient of the nonlinear function of activations for layer k :

$$3: \Phi^{(k)}[m] = \mathbf{F}_d^{(k)}[m] \odot \mathbf{Q}^{(k)}[m] \odot \mathbf{G}_d^{(k)}[m]$$

Update forward weights for layer k :

$$4: \Delta \mathbf{W}_{\text{EBD}}^{(f,k)}[m] \leftarrow -\frac{\mu^{(d_f,k)}[m]}{B} \Phi^{(k)}[m] \mathbf{H}^{(k-1)}[m]^\top$$

$$5: \Delta \mathbf{W}_{\text{Pred}}^{(f,k)}[m] \leftarrow \frac{\mu^{(f,k)}[m]}{B} \overset{\rightarrow}{\mathbf{E}}^{(k)}[m] (\mathbf{H}^{(k-1)}[m])^\top$$

$$6: \mathbf{W}^{(f,k)}[m] \leftarrow \mathbf{W}^{(f,k)}[m-1] + \Delta \mathbf{W}_{\text{EBD}}^{(f,k)}[m] + \Delta \mathbf{W}_{\text{Pred}}^{(f,k)}[m]$$

Update backward weights for layer k :

$$7: \Delta \mathbf{W}_{\text{EBD}}^{(b,k)}[m] \leftarrow -\frac{\mu^{(d_b,k)}[m]}{B} \Phi^{(k)}[m] \mathbf{H}^{(k+1)}[m]^\top$$

$$8: \Delta \mathbf{W}_{\text{Pred}}^{(b,k)}[m] \leftarrow \frac{\mu^{(b,k)}[m]}{B} \overset{\leftarrow}{\mathbf{E}}^{(k)}[m] \mathbf{H}^{(k+1)}[m]^\top$$

$$9: \mathbf{W}^{(b,k)}[m] \leftarrow \mathbf{W}^{(b,k)}[m-1] + \Delta \mathbf{W}_{\text{EBD}}^{(b,k)}[m] + \Delta \mathbf{W}_{\text{Pred}}^{(b,k)}[m]$$

Update Lateral weights for layer k :

$$10: \Delta \mathbf{B}_{\text{EBD}}^{(k)}[m] \leftarrow -\frac{\mu^{(d_l,k)}[m]}{B} \Phi^{(k)}[m] \mathbf{H}^{(k)}[m]^\top$$

$$11: \Delta \mathbf{B}_E^{(k)}[m] \leftarrow -\frac{\gamma_E}{B} \mathbf{Z}^{(k)}[m] \mathbf{Z}^{(k)}[m]^\top$$

$$12: \mathbf{B}^{(k)}[m] \leftarrow \frac{1}{\lambda_E} \mathbf{B}^{(k)}[m] + \Delta \mathbf{B}_E^{(k)}[m] + \Delta \mathbf{B}_{\text{EBD}}^{(k)}[m]$$

I.2.3 Initialization of algorithm variables

We initialize the variables $\mathbf{W}^{(f,k)}$, $\mathbf{W}^{(b,k)}$, and $\mathbf{R}_{\mathbf{h}^{(k)}} \epsilon$ using PyTorch’s Xavier uniform initialization with its default parameters for the MNIST dataset. For the CIFAR-10 dataset is initialized with gain 0.25. For the lateral weights $\mathbf{B}^{(k)}$, we first generate a random matrix $\mathbf{J}^{(k)}$ of the same dimensions, also using the Xavier uniform distribution, with gain= 1 for the MNIST dataset and with gain= 0.5 for the CIFAR-10 dataset. We then compute $\mathbf{B}^{(k)}[0] = \mathbf{J}^{(k)} \mathbf{J}^{(k)T}$, ensuring that $\mathbf{B}^{(k)}[0]$ is a positive definite symmetric matrix.

I.2.4 Description of hyperparameters

Table 8 presents a description of the hyperparameters used in the CorInfoMax-EBD implementation.

Table 8: Detailed explanation of hyperparameter notations for the CorInfoMax-EBD algorithm

Hyperparameter	Description
$\alpha[m]$	Learning rate dynamic scaling factor
$\alpha_2[m]$	Learning rate dynamic scaling factor 2
$\mu^{(d_f,k)}$	Learning rate for decorrelation loss (forward weights)
$\mu^{(d_b,k)}$	Learning rate for decorrelation loss (backward weights)
$\mu^{(d_l,k)}$	Learning rate for decorrelation loss (lateral weights)
$\mu^{(f,k)}$	Learning rate for forward prediction
$\mu^{(b,k)}$	Learning rate for backward prediction
$\mu^{(p,k)}$	Learning rate for power normalization loss
$p^{(k)}$	Target power level
$\mu_{f,\ell_1}^{(k)}$	Learning rate for activation sparsity (forward weights)
$\mu_{b,\ell_1}^{(k)}$	Learning rate for activation sparsity (backward weights)
$\mu_{f,w-\ell_2}^{(k)}$	Forward weight ℓ_2 -regularization coefficient
$\mu_{b,w-\ell_2}^{(k)}$	Backward weight ℓ_2 -regularization coefficient
λ_E	Layer correlation matrix update forgetting factor
λ_d	Error-layer activation cross-correlation forgetting factor
$m^{(d)}$	Momentum factor for decorrelation forward weight gradient
B	Batch size

I.2.5 Hyperparameters for 3-Layer MNIST and CIFAR-10 Models

Table 9 and 10 summarizes the hyperparameters used in the 3-layer CorInfoMax-EBD experiments for the MNIST and CIFAR-10 datasets with a batch size of 20 and 1 respectively. The iteration index is denoted by m in all expressions.

Table 9: 3-Layer CorInfoMax-EBD hyperparameters for MNIST and CIFAR-10 datasets ($B = 20$).

Hyperparameter	MNIST	CIFAR-10
$\alpha[m]$	$\frac{1}{3 \times 10^{-3} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{1}{3 \times 10^{-3} \times \lfloor \frac{m}{10} \rfloor + 1}$
$\alpha_2[m]$	$\frac{1}{3 \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{1}{3 \times \lfloor \frac{m}{10} \rfloor + 1}$
$\mu^{(d_f,k)}[m]$	$[96, 60, 1e5]\alpha[m]$	$[80, 50, 1e5]\alpha[m]$ for epoch= 0 $[320, 400, 1e5]\alpha[m]$ for epoch> 0
$\mu^{(d_b,k)}[m]$	$[96, 60, 1e5]\alpha[m]$	$[0, 0, 0]\alpha[m]$
$\mu^{(d_l,k)}[m]$	$[0.25, 0.25, 0.25]\alpha[m]$ for epoch= 0 $[0.5, 0.5, 0.5]\alpha[m]$ for epoch> 0	$[0.5, 0.5, 0.5]\alpha[m]$ for epoch= 0 $[2.0, 2.0, 2.0]\alpha[m]$ for epoch> 0
$\mu^{(f,k)}[m]$	$[0.11 \times 10^{-18}, 0.06 \times 10^{-18}, 0.035 \times 10^{-18}]\alpha[m]$	$[0.11 \times 10^{-18}, 0.06 \times 10^{-18}, 0.035 \times 10^{-18}]\alpha[m]$
$\mu^{(b,k)}[m]$	$[1.125 \times 10^{-18}, 0.375 \times 10^{-18}]\alpha[m]$	$[1.125 \times 10^{-18}, 0.375 \times 10^{-18}]\alpha[m]$
$\mu^{(p,k)}[m]$	$[4.4 \times 10^{-3}, 6 \times 10^{-3}, 3.5 \times 10^{-12}]\alpha_2[m]$	$[4.4 \times 10^{-3}, 6 \times 10^{-3}, 3.5 \times 10^{-12}]\alpha_2[m]$
$p^{(k)}$	$[2.5, 2.5, 0.1]$	$[2.5, 2.5, 0.1]$
$\mu_{f,\ell_1}^{(k)}[m]$	$[0.008, 0.135, 0]\alpha_2[m]$	$[0.008, 0.135, 0]\alpha_2[m]$
$\mu_{b,\ell_1}^{(k)}[m]$	$[0, 0.35, 0.05]\alpha_2[m]$	$[0, 0.35, 0.05]\alpha_2[m]$
$\mu_{f,w-\ell_2}^{(k)}[m]$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$
$\mu_{b,w-\ell_2}^{(k)}[m]$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$
λ_E	0.999999	0.999999
λ_d	0.99999	0.99999
$m^{(d)}[m]$	$0.99 \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} + 0.999 \left(1 - \frac{1}{\lfloor \frac{m}{10} \rfloor + 1}\right)$	$0.99 \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} + 0.999 \left(1 - \frac{1}{\lfloor \frac{m}{10} \rfloor + 1}\right)$
B	20	20

Table 10: 3-Layer CorInfoMax-EBD hyperparameters for MNIST and CIFAR-10 datasets ($B = 1$).

Hyperparameter	MNIST	CIFAR-10
$\alpha[m]$	$\frac{1}{3 \times 10^{-3} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{1}{3 \times 10^{-3} \times \lfloor \frac{m}{10} \rfloor + 1}$
$\alpha_2[m]$	$\frac{1}{3 \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{1}{3 \times \lfloor \frac{m}{10} \rfloor + 1}$
$\mu^{(d_f, k)}[m]$	$[4.8, 3.0, 5 \times 10^3] \alpha[m]$	$[4, 2.5, 5 \times 10^3] \alpha[m]$ for epoch = 0 $[16, 20, 5 \times 10^3] \alpha[m]$ for epoch > 0
$\mu^{(d_b, k)}[m]$	$[4.8, 3.0, 5 \times 10^3] \alpha[m]$	$[0, 0, 0] \alpha[m]$
$\mu^{(d_t, k)}[m]$	$[0.0125, 0.0125, 0.0125] \alpha[m]$ for epoch = 0 $[0.025, 0.025, 0.025] \alpha[m]$ for epoch > 0	$[0.025, 0.025, 0.025] \alpha[m]$ for epoch = 0 $[0.1, 0.1, 0.1] \alpha[m]$ for epoch > 0
$\mu^{(f, k)}[m]$	$[0.11 \times 10^{-18}, 0.06 \times 10^{-18}, 0.035 \times 10^{-18}] \alpha[m]$	$[0.11 \times 10^{-18}, 0.06 \times 10^{-18}, 0.035 \times 10^{-18}] \alpha[m]$
$\mu^{(b, k)}[m]$	$[1.125 \times 10^{-18}, 0.375 \times 10^{-18}] \alpha[m]$	$[1.125 \times 10^{-18}, 0.375 \times 10^{-18}] \alpha[m]$
$\mu^{(p, k)}[m]$	$[2.2 \times 10^{-4}, 3 \times 10^{-4}, 3.5 \times 10^{-12}] \alpha_2[m]$	$[2.2 \times 10^{-4}, 3 \times 10^{-4}, 3.5 \times 10^{-12}] \alpha_2[m]$
$p^{(k)}$	$[2.5, 2.5, 0.1]$	$[0.125, 0.125, 0.005]$
$\mu_{f, \ell_1}^{(k)}[m]$	$[0.0004, 0.00675, 0] \alpha_2[m]$	$[0.0004, 0.00675, 0] \alpha_2[m]$
$\mu_{b, \ell_1}^{(k)}[m]$	$[0, 0.0175, 0.0025] \alpha_2[m]$	$[0, 0.0175, 0.0025] \alpha_2[m]$
$\mu_{f, w - \ell_2}^{(k)}[m]$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$
$\mu_{b, w - \ell_2}^{(k)}[m]$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{8 \times 10^{-2}}{10^{-2} \times \lfloor \frac{m}{10} \rfloor + 1}$
λ_E	0.99999995	0.99999995
λ_d	0.99999	0.99999
$m^{(d)}[m]$	$0.99 \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} + 0.999 \left(1 - \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} \right)$	$0.99 \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} + 0.999 \left(1 - \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} \right)$
B	1	1

I.2.6 Hyperparameters for 10-layer CorInfoMax-EBD on MNIST and CIFAR-10 datasets for batch size 1

Table 11 list the hyperparameters used in the 10-Layer CorInfoMax-EBD numerical experiments for the MNIST and CIFAR-10 datasets with a batch size of 1. In these experiments, a weight thresholding scheme is applied to the network weights for every 5000 samples, where the weights with 0.00003 scale (relative to the peak) are set to zero.

 Table 11: 10-Layer CorInfoMax-EBD hyperparameters: MNIST and CIFAR-10 datasets ($B = 1$).

Hyperparameter	Value
$\alpha[m]$	$\frac{1}{3 \times 10^{-3} \times \lfloor \frac{m}{10} \rfloor + 1}$
$\alpha_2[m]$	$\frac{1}{3 \times \lfloor \frac{m}{10} \rfloor + 1}$
$\mu^{(d_f, k)}[m]$	$[3.5 \quad 3.5 \quad \dots \quad 3.5 \quad 6e4] \alpha[m]$
$\mu^{(d_t, k)}[m]$	$0.03 \alpha[m] \cdot \mathbf{1}_{1 \times 10}$
$\mu^{(f, k)}[m]$	$[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.11, 0.06, 0.035] \cdot 1e(-18) \cdot \alpha[m]$
$\mu^{(b, k)}[m]$	$[1.1, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4] \cdot 1e(-18) \cdot \alpha[m]$
$\mu^{(p, k)}[m]$	$[2, 2, 2, 2, 2, 2, 2, 5, 1e-7] \cdot 1e(-3) \cdot \alpha_2[m]$
$p^{(k)}$	$[1, 1, 1, 1, 1, 1, 1, 2, 0.1]$
$\mu_{f, \ell_1}^{(k)}[m]$	$[0.16, 0.16, 0.16, 0.16, 0.16, 0.16, 0.16, 0.16, 0.0] \alpha_2[m]$
$\mu_{f, w - \ell_2}^{(k)}[m]$	$\mathbf{0}_{1 \times 10}$
$\mu_{b, w - \ell_2}^{(k)}[m]$	$5e-4 \cdot \alpha_2[m] \mathbf{1}_{1 \times 10}$
λ_E	0.99999995
λ_d	$0.999999\gamma + (1 - \gamma)0.99999999$ with $\gamma = \frac{1}{\frac{m}{5} + 1}$
$m^{(d)}[m]$	$0.99 \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} + 0.999 \left(1 - \frac{1}{\lfloor \frac{m}{10} \rfloor + 1} \right)$
B	1

I.3 Multi-Layer Perceptron

In this section, we provide additional details about the numerical experiments conducted to train Multi-layer Perceptrons (MLPs) using the EBD algorithm (MLP-EBD). Appendix I.3.1 outlines the implementation details of these experiments, while Appendix I.3.2 discusses the initialization of algorithm variables. Information about hyperparameters and their values for the MNIST and CIFAR-10 datasets can be found in Appendices I.3.3-I.3.4. In Appendix I.10 we present the accuracy and loss learning curves for the MLP architecture, shown in Figures 6.(a)-(b) and Figures 7.(a)-(b), respectively.

I.3.1 Implementation details

For the MLP experiments using the proposed EBD approach, we adopted the same network architecture as described in [9], detailed in Tables 4 and 5.

In the MLP-EBD implementation, the following loss and regularization functions were employed:

- EBD loss: $J^{(k)}$,
- Power normalization loss: $J_P^{(k)}$,
- Entropy objective: $J_E^{(k)}$,
- ℓ_2 weight regularization (weight decay): $J_{\ell_2}^{(k)}$,
- Activation sparsity regularization: $J_{\ell_1}^{(k)} = \|\mathbf{H}^{(k)}\|_1$.

Additionally, we imposed a weight-sparsity constraint by setting WS percent of the weights to zero during the initialization phase and maintaining these weights at zero throughout training.

I.3.2 Initialization of algorithm variables

We use the Pytorch framework’s Xavier uniform initialization with gain value 10^{-2} on the $\mathbf{R}_{\mathbf{h}^{(k)}} \epsilon$ variables, and Kaiming uniform distribution with gain 0.75 for synaptic weights $\mathbf{W}^{(k)}$.

I.3.3 Description of hyperparameters

Table 12 provides the description of the hyperparameters for the MLP-EBD implementation.

Table 12: Description of the hyperparameter notations for MLP-EBD.

Hyperparameter	Description
$\alpha[m]$	Learning rate dynamic scaling factor
$\alpha_2[m]$	Learning rate dynamic scaling factor 2
$\mu^{(d,b,k)}$	Learning rate for (backward projection) decorrelation loss
$\mu^{(d,f,k)}$	Learning rate for (forward projection) decorrelation loss
$\mu^{(E,k)}$	Learning rate for entropy objective
$\mu^{(p,k)}$	Learning rate for power normalization loss
$p^{(k)}$	Target power level
$\mu_{\ell_1}^{(k)}$	Learning rate for activation sparsity
$\mu_{w-\ell_2}^{(k)}$	Weight ℓ_2 -regularization coefficient
λ_E	Layer autocorrelation matrix update forgetting factor
λ_d	Error-layer activation cross-correlation forgetting factor
$m^{(d)}$	Momentum factor for decorrelation gradient
B	Batch size
WS	Weight Sparsity

I.3.4 Hyperparameters for MLP-EBD on MNIST and CIFAR-10 Datasets

Table 13 summarizes the hyperparameters used in the MLP-EBD experiments for the MNIST and CIFAR-10 datasets. The iteration index is denoted by m in all expressions.

Table 13: MLP-EBD hyperparameters for MNIST and CIFAR-10 datasets.

Hyperparameter	MNIST	CIFAR-10
$\alpha[m]$	$\frac{1}{1.5 \times \lfloor \frac{m}{10} \rfloor + 1}$	$\frac{1}{1.5 \times \lfloor \frac{m}{10} \rfloor + 1}$
$\alpha_2[m]$	$\frac{\lfloor \frac{m}{10} \rfloor}{3 \times 10^4} + 1$	$\frac{\lfloor \frac{m}{10} \rfloor}{3 \times 10^4} + 1$
$\mu^{(d,b,k)}[m]$	$18000 \alpha[m] \alpha_2[m]$ for $k = 0, 1$ $20000 \alpha[m] \alpha_2[m]$ for $k = 2$	$[4000, 2000, 2000, 3500] \alpha[m] \alpha_2[m]$
$\mu^{(d,f,k)}[m]$	$0.005 \alpha[m] \alpha_2[m]$ for $k = 0, 1$	$0.005 \alpha[m] \alpha_2[m]$ for $k = 0, 1$
$\mu^{(E,k)}[m]$	$[2.5 \times 10^{-4}, 1.5 \times 10^{-3}, 0] \alpha[m]$	$[2.5 \times 10^{-4}, 1.5 \times 10^{-3}, 1.5 \times 10^{-3}, 0] \alpha[m]$
$\mu^{(p,k)}[m]$	$[4 \times 10^{-3}, 6 \times 10^{-3}, 1 \times 10^{-10}] \alpha[m]$	$[4 \times 10^{-3}, 6 \times 10^{-3}, 6 \times 10^{-3}, 1 \times 10^{-10}] \alpha[m]$
$p^{(k)}[m]$	$[0.25, 0.25, 0.1] \alpha[m]$	$[0.25, 0.25, 0.25, 0.1] \alpha[m]$
$\mu_{\ell_1}^{(k)}$	$[0.8, 0.3, 0] \alpha[m]$	$[0.8, 0.3, 0.3, 0] \alpha[m]$
$\mu_{w-\ell_2}^{(k)}$	$1.6 \times 10^{-4} \alpha[m]$ for all layers	$1.6 \times 10^{-4} \alpha[m]$ for all layers
λ_E	0.99999	0.99999
λ_d	0.999999	0.999999
$m^{(d)}$	0.9999 for all layers	0.9999 for all layers
B	20	20
WS	55	40

I.4 Convolutional Neural Network

In this section, we offer additional details regarding the numerical experiments for training Convolutional Neural Networks (CNNs) using EBD algorithm (CNN-EBD). Section I.4.1 provides information about implementation details. Appendices I.4.2 and I.4.3 discuss the initialization of the algorithm’s variables and describe the hyperparameters. Finally, Appendix I.4.4 detail the specific hyperparameter configurations used in our numerical experiments for the MNIST, CIFAR-10 and CIFAR-100 datasets. In Appendix I.10 we present the accuracy and loss learning curves for the CNN, shown in Figures 6.(c)-(d) and Figures 7.(c)-(d), respectively.

I.4.1 Implementation details

The architectures we utilized for the CNN networks can be found in tables 4 and 5 respectively for the MNIST and CIFAR10 datasets. In the training, we used the Adam optimizer with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ [57]. Also, the model biases are not utilized. In the CNN-EBD implementation the following loss and regularization functions as detailed in section D.1 are used:

- EBD loss: $J^{(k)}$,
- Entropy objective: $J_E^{(k)}$,
- Activation sparsity regularization: $J_{\ell_1}^{(k)}$.

Specifically for the CIFAR-100 experiments, we applied both training and test time augmentation of data to improve model generalization and handle increased difficulty in the task. At training time, each image was randomly translated into 2-pixels with reflection padding and a deterministic alternating horizontal flip that ensures that every image is flipped every other epoch, reducing redundancy compared to standard random flipping. During evaluation, we used test-time augmentation combining horizontal flipping and multi-crop averaging over six translated views (the original, two one-pixel translations, and their mirrored counterparts).

I.4.2 Initialization of algorithm variables

We use the Kaiming normal initialization for the weights, with a common standard deviation scaling parameter $\sigma_{\mathbf{W}}$, on both the linear and convolutional layers. Furthermore, the estimated cross-correlation variable $\mathbf{R}_{\mathbf{h}^{(k)}\epsilon}$ (linear layers) and $\mathbf{R}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}$ (convolutional layers) are initialized with zero mean normal distributions with standard deviations $\sigma_{\mathbf{R}_{lin}}$ and $\sigma_{\mathbf{R}_{conv}}$ respectively.

I.4.3 Description of hyperparameters

Table 14 describes the notation for the hyperparameters used to train CNNs using the Error Broadcast and Decorrelation (EBD) approach.

Table 14: Description of the hyperparameter notations for CNN-EBD.

Hyperparameter	Description
α_{exp}	Exponential learning rate decay parameter.
$\alpha[i]$	Learning rate dynamic scaling factor where i is the epoch index
$\mu^{(d,b,k)}$	Learning rate for (backward projection) decorrelation loss
$\mu^{(E,k)}$	Learning rate for entropy objective
$\mu_{\ell_1}^{(k)}$	Learning rate for activation sparsity
$\sigma_{\mathbf{W}}$	Standard deviation of the weight initialization.
$\sigma_{\mathbf{R}_{lin}}$	Std. dev. of $\mathbf{R}_{\mathbf{h}^{(k)}\epsilon}$ initialization in linear layers
$\sigma_{\mathbf{R}_{conv}}$	Std. dev. of $\mathbf{R}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}$ initialization in convolutional layers
$\sigma_{\mathbf{R}_{local}}$	Gain parameter for $\mathbf{R}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}$ initialization in locally connected layers
λ_E	Layer autocorrelation matrix update forgetting factor
λ_d	Error-layer activation cross-correlation forgetting factor
λ_R	Convergence parameter for λ as in Equations (61), (62)
ϵ_L	Entropy objective epsilon parameter for linear layers
ϵ	Entropy objective epsilon parameter for conv. or locally con. layers
β	Adam Optimizer weight decay parameter
p	Dropout probability
B	Batch size

We also introduce a convergence parameter λ_R which increases the estimation parameter for the decorrelation loss λ_d , together with the estimation parameter for the layer entropy objective λ_E , to converge to 1 as the training proceeds with the following Equations (61), (62) where i is the epoch index:

$$\lambda_d^{(i+1)} = \lambda_d^{(i)} + \lambda_R \cdot (1 - \lambda_d^{(i)}), i \geq 0. \quad (61)$$

$$\lambda_E^{(i+1)} = \lambda_E^{(i)} + \lambda_R \cdot (1 - \lambda_E^{(i)}), i \geq 0. \quad (62)$$

I.4.4 Hyperparameters for MNIST, CIFAR-10 and CIFAR-100 datasets

Table 15, lists the hyperparameters as defined in Table 14, used in the CNN-EBD training experiments.

Table 15: Hyperparameters for CNN-EBD for the MNIST, CIFAR-10 and CIFAR-100 datasets, where i denotes the epoch index.

Hyperparameter	MNIST	CIFAR-10	CIFAR-100
α_{exp}	0.97	0.97	0.97
$\alpha[i]$	$10^{-4} \cdot \alpha_{\text{exp}}^{-i}$	$10^{-4} \cdot \alpha_{\text{exp}}^{-i}$	$10^{-4} \cdot \alpha_{\text{exp}}^{-i}$
$\mu^{(d,b,k)}[i]$	$0.1\alpha[i]$ for $k = 0, 1, 2, 3$ $10\alpha[i]$ for $k = 4$	$0.1\alpha[i]$ for $k = 0, 1, 2, 3$ $10\alpha[i]$ for $k = 4$	$0.1\alpha[i]$ for $k = 0, 1, 2, 3, 4$ $10\alpha[i]$ for $k = 5$
$\mu^{(E,k)}[i]$	$[1 \ 1 \ 1 \ 10 \ 0] 10^{-7}\alpha[i]$	$[1 \ 1 \ 1 \ 1 \ 1] 10^{-6}\alpha[i]$	$[0 \ 0 \ 0 \ 5 \ 5 \ 5] 10^{-7}\alpha[i]$
$\mu_{\ell_1}^{(k)}[i]$	$[1 \ 1 \ 1 \ 10 \ 0] 10^{-11}\alpha[i]$	$[1 \ 1 \ 1 \ 10^2 \ 0] 10^{-10}\alpha[i]$	$[0 \ 0 \ 0 \ 1 \ 1 \ 0] 10^{-7}\alpha[i]$
$\sigma_{\mathbf{W}}$	$\sqrt{\frac{1}{6}}$	$\sqrt{\frac{1}{6}}$	$\sqrt{\frac{1}{6}}$
$\sigma_{\mathbf{R}_{lin}}$	$1e-2$	$1e-2$	$1e-2$
$\sigma_{\mathbf{R}_{conv}}$	$1e-2$	$1e-2$	$1e-2$
λ_d	0.99999	0.99999	0.99999
λ_E	0.99999	0.99999	0.99999
λ_R	$2e-2$	$2e-2$	$2e-2$
β	$1e-8$	$1e-5$	$1e-5$
ϵ_L	$1e-8$	$1e-8$	$1e-8$
ϵ	$1e-5$	$1e-5$	$1e-5$
B	16	16	16
p	N/A	N/A	0.075

I.5 Locally Connected Network

In this section, we offer additional details regarding the numerical experiments for the training of Locally Connected Networks (LCs) using EBD algorithm (LC-EBD). Appendix I.5.1 provides information about implementation details. Appendices I.5.2 and I.5.3 discuss the initialization of the algorithm’s variables and describe the hyperparameters. Finally, Appendix I.5.4 detail the specific hyperparameter configurations used in our numerical experiments for the MNIST and CIFAR-10 datasets. In Appendix I.10 we present the accuracy and loss learning curves for the LCs, shown in Figures 6.(e)-(f) and Figures 7.(e)-(f), respectively.

I.5.1 Implementation details

The training procedure mirrors the CNN approach described in Section I.4.1 for CNNs. In the LC-EBD implementation, the loss and regularization functions detailed in section D.2 are used:

- EBD loss: $J^{(k)}$,
- Entropy objective: $J_E^{(k)}$,
- Activation sparsity regularization: $J_{\ell_1}^{(k)}$.

I.5.2 Initialization of algorithm variables

We use the Kaiming uniform initialization for the weights, with a common standard deviation scaling parameter $\sigma_{\mathbf{W}}$, on both the linear and locally connected layers. The estimated cross-correlation variable $\mathbf{R}_{\mathbf{h}^{(k)}\epsilon}$ (linear layers) is initialized with a normal distribution with zero mean and standard deviation $\sigma_{\mathbf{R}_{lin}}$. Also, the parameter $\mathbf{R}_{\mathbf{g}^{(k)}(\mathbf{H}^{(k,p)})\epsilon}$ (locally connected layers) is initialized with Pytorch framework’s Xavier uniform initialization with the gain parameter equal to $\sigma_{\mathbf{R}_{local}}$.

I.5.3 Description of hyperparameters

Table 14 in the CNN section, again describes the notation for the hyperparameters used to train LCs using the Error Broadcast and Decorrelation (EBD) approach. The convergence parameter λ_R introduced in equations Eq. (61) and Eq. (61) is used as well.

I.5.4 Hyperparameters for MNIST and CIFAR-10 datasets

Table 16, lists the hyperparameters as defined in Table 14, used in the LC-EBD training experiments.

Table 16: Hyperparameters for LC-EBD for both the MNIST and CIFAR-10 datasets, where i denotes the epoch index.

Hyperparameter	MNIST	CIFAR-10
α_{exp}	0.96	0.98
$\alpha[i]$	$10^{-4} \cdot \alpha_{\text{exp}}^{-i}$	$10^{-4} \cdot \alpha_{\text{exp}}^{-i}$
$\mu^{(d,b,k)}[i]$	$0.1\alpha[i]$ for $k = 0, 1, 2, 3$ $10\alpha[i]$ for $k = 4$	$0.5\alpha[i]$ for $k = 0, 1, 2, 3$ $5\alpha[i]$ for $k = 4$
$\mu^{(E,k)}[i]$	$\begin{bmatrix} 1 & 1 & 1 & 10^2 & 0 \end{bmatrix} 10^{-9}\alpha[i]$	$\begin{bmatrix} 1 & 1 & 1 & 10 & 10^3 \end{bmatrix} 10^{-11}\alpha[i]$
$\mu_{\ell_1}^{(k)}[i]$	$\begin{bmatrix} 1 & 1 & 1 & 10 & 0 \end{bmatrix} 10^{-11}\alpha[i]$	$\begin{bmatrix} 1 & 1 & 1 & 10 & 0 \end{bmatrix} 10^{-13}\alpha[i]$
$\sigma_{\mathbf{W}}$	$\sqrt{\frac{1}{6}}$	$\sqrt{\frac{1}{6}}$
$\sigma_{\mathbf{R}_{lin}}$	1	$1e - 3$
$\sigma_{\mathbf{R}_{local}}$	1	$1e - 1$
λ_d	0.99999	0.99999
λ_E	0.99999	0.99999
λ_R	$3e - 2$	$3e - 2$
β	$1e - 8$	$1e - 6$
ϵ_L	$1e - 8$	$1e - 8$
ϵ	$1e - 5$	$1e - 5$
B	16	16
p	N/A	N/A

I.6 Implementation details for Direct Feedback Alignment (DFA) and backpropagation training

This section presents further details on the numerical experiments comparing Direct Feedback Alignment (DFA) and Backpropagation (BP) methods, conducted under the same training conditions and number of epochs as those used for our proposed EBD algorithm. The results of these experiments are provided in Table 1. We also include the DFA+E method, which extends DFA by incorporating correlative entropy regularization similar to the EBD. Note that, when the update on the $\mathbf{R}_{\mathbf{h}}^{(k)}_{\epsilon}$ is fixed to its initialization, the EBD algorithm reduces to standard DFA.

For BP-based models trained on MNIST, CIFAR-10 and CIFAR-100, we used the Adam optimizer with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ [57]. For DFA and DFA+E models, we again used the Adam optimizer for CNN and LC models, while MLP models were trained using SGD with momentum.

In Tables 17 and 18, we detail the hyperparameters for models trained with BP, DFA, and DFA+E update rules on MNIST and CIFAR-10 respectively. In Table 19, we give the hyperparameters for the CNN model (detailed in Table 7) trained with with BP and DFA. Some of the learning rate and the learning rate decay values or methodologies are linked to the tables corresponding to the hyperparameter details of its EBD counterpart, where the same method is also utilized for its DFA or DFA+E counterpart. Unlinked values denote a constant value applied to each layer, or the step decay multiplier applied per epoch. Additionally, sparsity inducing losses are not utilized for BP, DFA and DFA+E models.

Table 17: Hyperparameter details for models trained on the MNIST dataset, including learning rate, L2 regularization coefficient, learning rate decay, and number of epochs for MLP, CNN, and LC models using BP, DFA, and DFA+E methods.

Model	Method	Learning Rate ($\mu^{(d,b,k)}$)	L2 Reg. Coef.	LR Decay (α_{exp})	Epochs
MLP	BP	$5e-5$	$1e-5$	0.96	120
	DFA	Table-13	Table-13	Table-13	120
	DFA+E	Table-13	Table-13	Table-13	120
CNN	BP	$5e-5$	$1e-8$	0.97	100
	DFA	Table-15	$1e-8$	0.97	100
	DFA+E	Table-15	$1e-8$	0.97	100
LC	BP	$5e-5$	$1e-8$	0.96	100
	DFA	Table-16	$1e-8$	0.96	100
	DFA+E	Table-16	$1e-8$	0.96	100

Table 18: Hyperparameter details for models trained on the CIFAR-10 dataset, including learning rate, L2 regularization coefficient, learning rate decay, and number of epochs for MLP, CNN, and LC models using BP, DFA, and DFA+E methods.

Model	Method	Learning Rate ($\mu^{(d,b,k)}$)	L2 Reg. Coef.	LR Decay (α_{exp})	Epochs
MLP	BP	$5e-5$	$1e-5$	0.85	120
	DFA	Table-13	0	Table-13	120
	DFA+E	Table-13	0	Table-13	120
CNN	BP	$5e-5$	$1e-5$	0.92	200
	DFA	Table-15	$1e-5$	0.97	200
	DFA+E	Table-15	$1e-5$	0.97	200
LC	BP	$1e-4$	$1e-6$	0.90	200
	DFA	Table-16	$1e-6$	0.96	200
	DFA+E	Table-16	$1e-6$	0.96	200

Table 19: Hyperparameter details for the CNN model trained on the CIFAR-100 dataset, including learning rate, L2 regularization coefficient, learning rate decay, dropout probability and number of epochs using BP and DFA methods.

Model	Method	Learning Rate ($\mu^{(d,b,k)}$)	L2 Reg. Coef.	LR Decay (α_{exp})	Dropout (p)	Epochs
CNN	BP	$5e-5$	$1e-5$	0.97	0.5	200
	DFA	$[0.1 \ 0.1 \ 0.1 \ 1 \ 1 \ 50] \cdot 0.25\alpha[i]$	0	0.95	0.075	300

I.7 Runtime comparisons for the update rules

In this section, we present the relative average runtimes from the simulations, normalized to BP for the MNIST and CIFAR-10 models in Tables 20 and 21 respectively, for the models that we implemented and demonstrated their performance in Table 1.

The results show that entropy regularization in both EBD and DFA+E more than doubles the average runtime. However, these runtimes could be significantly improved by optimizing the implementation of the entropy gradient terms, specifically by avoiding repeated matrix inverse calculations. A more efficient approach would involve directly updating the inverses of the correlation matrices instead of recalculating both the matrices and their inverses at each step. This strategy aligns with the CorInfoMax-(EP/EBD) network structure. Nonetheless, we chose not to pursue this optimization, as CorInfoMax networks already employ it effectively.

The efficiency of the DFA, DFA+E, and EBD methods can be further enhanced through low-level optimizations and improved implementations.

Table 20: Average Runtimes in MNIST (relative to BP)

Model	DFA	DFA+E	BP	EBD
MLP	3.40	7.68	1.0	8.06
CNN	1.68	2.95	1.0	3.85
LC	1.61	3.57	1.0	3.54

Table 21: Average Runtimes in CIFAR-10 (relative to BP)

Model	DFA	DFA+E	BP	EBD
MLP	2.85	6.94	1.0	7.61
CNN	2.10	3.24	1.0	4.11
LC	1.35	2.01	1.0	2.41

I.8 Reproducibility

To facilitate the reproducibility of our results, we have included the following:

- i. Detailed information on the derivation of the weight and bias updates of the Error Broadcast and Decorrelation (EBD) Algorithm for various networks in Appendix C for MLPs, D.1 for CNNs, D.2 for LCs,
- ii. Full list of hyperparameters used in the experiments in Appendix I.2.5, I.3.4, I.4.4, I.5.4,
- iii. Algorithm descriptions for CorInfoMax Error Broadcast and Decorrelation (CorInfoMax-EBD) Algorithm in pseudo-code format in Appendix I.2.2,
- iv. Python scripts, Jupyter notebooks, and bash scripts for replicating the individual experiments and reported results are included in the supplementary zip file.

I.9 Computational resources

All experiments were conducted within a High-Performance Computing (HPC) facility. Each experimental run utilized a single NVIDIA Tesla V100 GPU equipped with 32GB of HBM2 memory. To provide context on execution times for our proposed CorInfoMax-EBD models:

- Training the 3-Layer CorInfoMax-EBD model (as described in Appendix I.2.5) for 30 epochs required approximately 22 hours.
- Training the 10-Layer CorInfoMax-EBD model (as described in Appendix I.2.6) for 100 epochs took approximately 75 hours. Execution times for other models (MLP, CNN, LC) and baseline methods were generally shorter; relative runtime comparisons are provided in Appendix I.7.

I.10 Accuracy and loss curves

Figures 6 and 7 present the training/test accuracy and MSE loss curves over epochs for the CIFAR-10 and MNIST datasets. Solid lines represent test curves; dashed lines denote training curves.

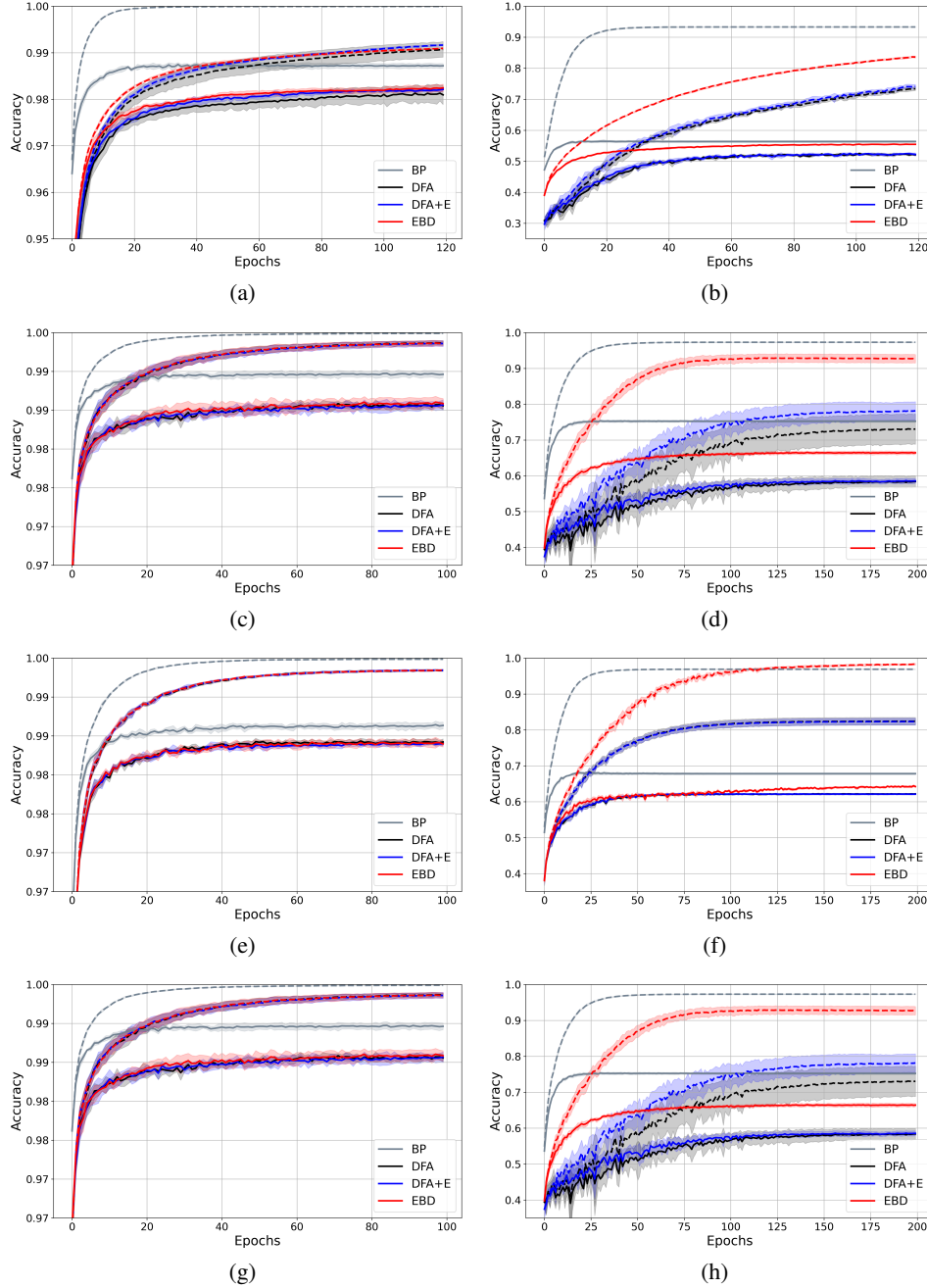


Figure 6: Train and test accuracies plotted as a function of algorithm epochs for various update rules (averaged over $n = 5$ runs associated with the corresponding $\pm$ std envelopes) for the (a) MLP on MNIST (b) MLP on CIFAR-10 (c) CNN on MNIST (d) CNN on CIFAR-10 (e) LC on MNIST (f) LC on CIFAR-10 (g) 10-Layer CorInfoMax-EBD with batchsize=1 on MNIST (h) 10-Layer CorInfoMax-EBD with batchsize=1 on CIFAR-10.

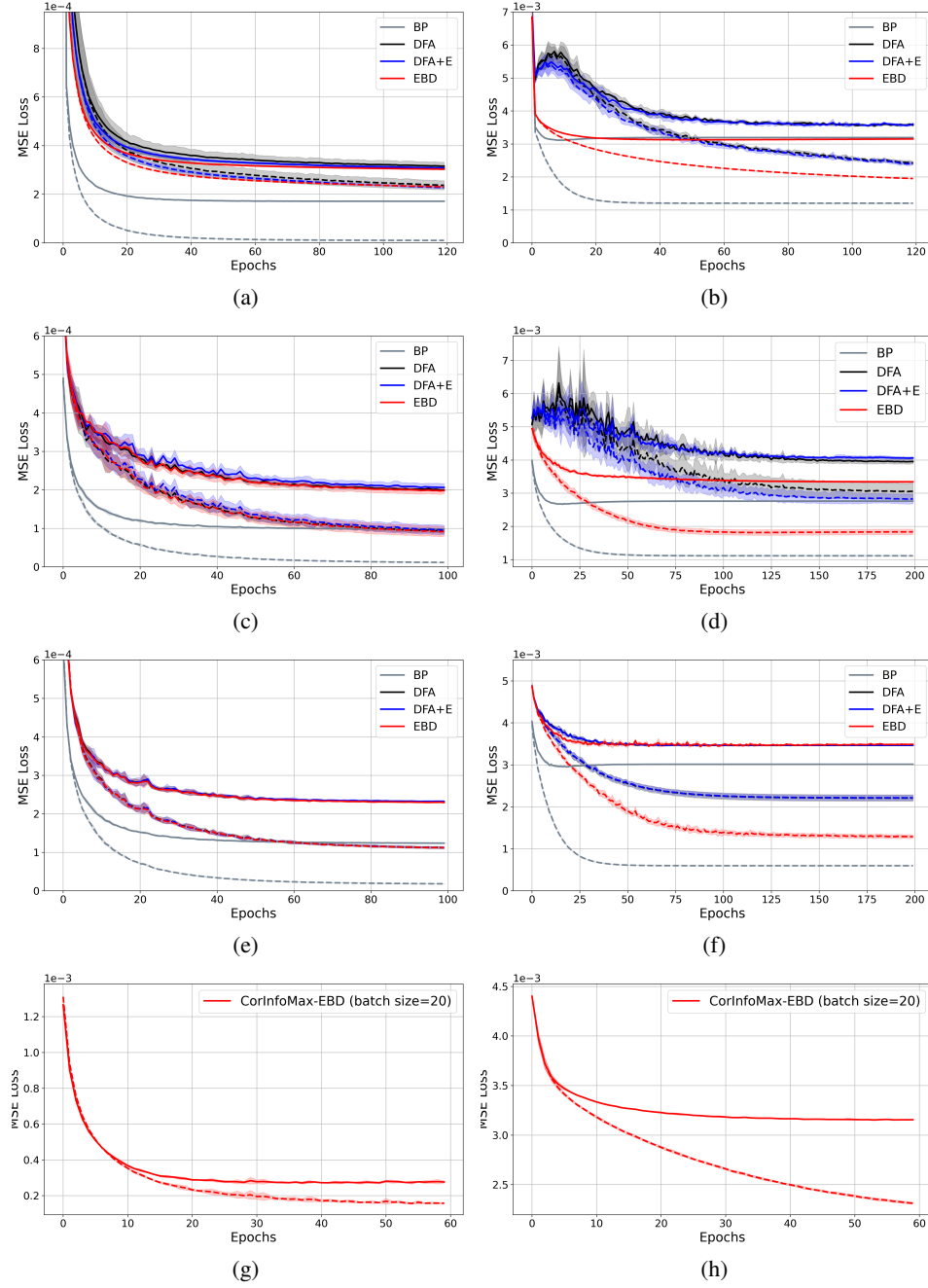


Figure 7: Train and test mean squared errors (MSE) plotted as a function of algorithm epochs for various update rules (averaged over $n = 5$ runs associated with the corresponding $\pm$ std envelopes) for the (a) MLP on MNIST (b) MLP on CIFAR-10 (c) CNN on MNIST (d) CNN on CIFAR-10 (e) LC on MNIST (f) LC on CIFAR-10 (g) 3-Layer CorInfoMax-EBD with batchsize=20 on MNIST (h) 3-Layer CorInfoMax-EBD with batchsize=20 on CIFAR-10.

Figure 8 shows the training/test accuracy curves over epochs in the 10-Layer CorInfoMax-EBD numerical experiments for the CIFAR-10 and MNIST datasets. Furthermore, Figure 9 presents the training/test accuracy curves over epochs for the CNN model trained with the CIFAR-100 dataset. Solid lines represent test curves; dashed lines denote training curves. Solid lines represent test curves; dashed lines denote training curves.

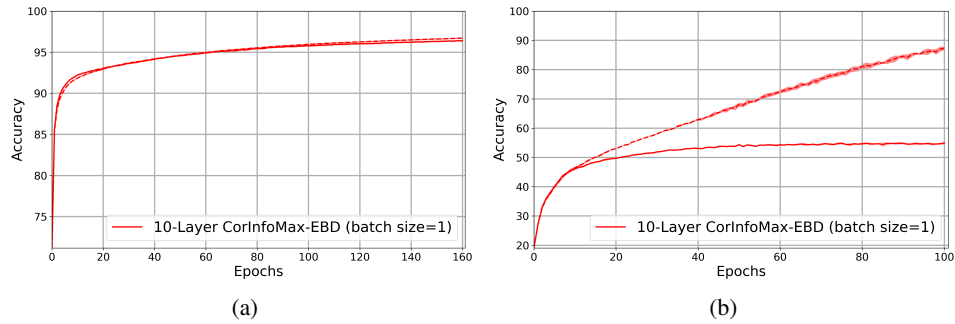


Figure 8: Train and test accuracies plotted as a function of algorithm epochs (averaged over $n = 5$ runs associated with the corresponding $\pm$ std envelopes) for training with (a) 10-Layer CorInfoMax-EBD with batchsize=1 on MNIST (b) 10-Layer CorInfoMax-EBD with batchsize=1 on CIFAR-10.

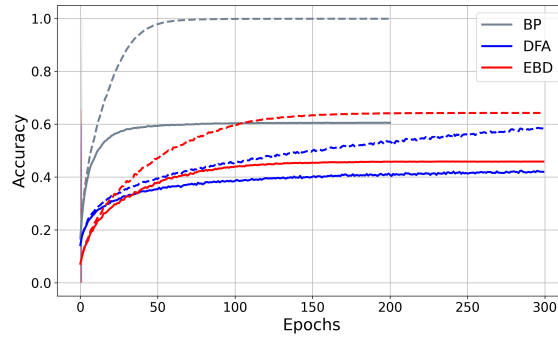


Figure 9: Train and test accuracies plotted as a function of algorithm epochs for various update rules (averaged over $n = 5$ runs associated with the corresponding $\pm$ std envelopes) for training with the CNN network on CIFAR-100.

J Calculation of the correlation between layer activations and output error

Figure 1c illustrates the decrease in the average absolute correlation between hidden activations and output error during backpropagation, using a Multi-layer Perceptron (MLP) model with the architecture outlined in Table 5, on the CIFAR-10 dataset. Details for the MSE based training and the Cross-Entropy based training are explained in Appendices J.1 and J.2 respectively.

J.1 Correlation in the mean squared error (MSE) criterion-based training

The MLP models are trained using the Stochastic Gradient Descent (SGD) optimizer with a small learning rate of 10^{-4} and the MSE criterion. In both plots, the initial value represents the correlation before training begins. The reduction in correlation observed during training provides insight into the core principle of the EBD algorithm.

To compute these correlations, we apply a batched version of Welford’s algorithm [58], which efficiently calculates the Pearson correlation coefficient between hidden activations and errors in a memory-efficient way by using streaming statistics.

Welford’s algorithm works by accumulating the necessary statistics (e.g., sums and sums of squares) across batches of data and finalizing the correlation computation only after all data has been processed, avoiding the need to store all hidden activations simultaneously.

Given the hidden activations $\mathbf{h}^{(k)} \in \mathbb{R}^{b \times N^{(k)}}$, where b is the batch size and $N^{(k)}$ is the number of hidden units, and the errors $\epsilon \in \mathbb{R}^{b \times k}$, where k is the number of output dimensions (e.g., classes); the goal is to compute the Pearson correlation coefficient between activations h_i for each hidden unit i and the corresponding error values across all samples as:

$$\rho^{(k)} = \frac{\text{Cov}(\mathbf{h}^{(k)}, \epsilon)}{\sqrt{\sigma_{\mathbf{h}^{(k)}}^2} \sqrt{\sigma_{\epsilon}^2 + \epsilon}}$$

where ϵ is a small constant for numerical stability. Finally, we compute the average of the absolute values of the correlation coefficients for each hidden layer k to generate the corresponding plots.

Figure 10 shows the correlation throughout training on the MNIST dataset, employing the MLP model detailed in Table 4, where we observe the correlation decay behavior similar to Figure 1c.

To verify that the correlation–decay phenomenon observed for fully-connected networks extends to convolutional architectures, we trained a compact five-layer CNN consisting of three Conv+ReLU+MaxPool blocks (with 32, 64, and 128 3×3 filters, respectively) followed by a fully-connected layer with 512 hidden units and an output layer with 10 logits. The network was optimized for 30 epochs on CIFAR-10 using mini-batch SGD ($\eta = 10^{-3}$, momentum 0.9) and the mean-squared error criterion on one-hot labels. After every forward pass we streamed the Pearson correlation between each hidden activation vector and the output error using the batched Welford estimator. Figure 11 plots the epoch-wise evolution of the average absolute correlation coefficient for both training and test data. As with the MLP in Figure 1c, all layers exhibit a pronounced monotonic decline, confirming that back-propagation progressively enforces the stochastic orthogonality of hidden activations and output errors in convolutional networks as well, as expected.

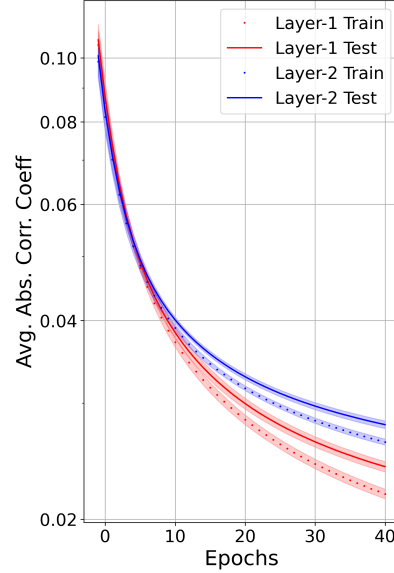


Figure 10: The evolution of the average absolute correlation between layer activations and the error signal during backpropagation training of an MLP with two hidden layers (using the MSE criterion) on the MNIST dataset, showing the correlation decrease over epochs, on both the training and test sets.

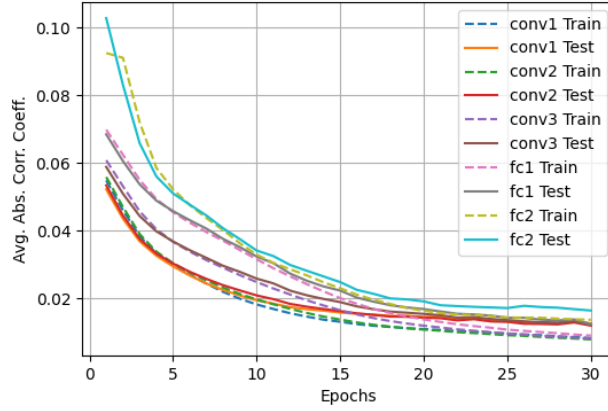


Figure 11: The evolution of the average absolute correlation between layer activations and the error signal during back-propagation training of a CNN on CIFAR-10. The CNN contains three convolutional layers and two fully-connected layers and is trained with the MSE criterion. The correlation decreases over epochs on both the training and test sets, mirroring the behavior observed for the MLP architecture and supporting the generality of the correlation decay across architectures.

J.2 Correlation in the cross-entropy criterion-based training

Although the stochastic orthogonality property is specifically associated with the MSE loss, we also explored the dynamics of cross-correlation between layer activations and output errors when cross-entropy is used as the training criterion.

With the same experimental setup as described in Appendix J.1, but replacing the MSE loss with cross-entropy, we obtained the correlation evolution curves shown in Figure 12a for CIFAR-10 and in Figure 12b for MNIST dataset. Notably, the correlation between layer activations and output errors still decreases over epochs, despite the change in the loss function.

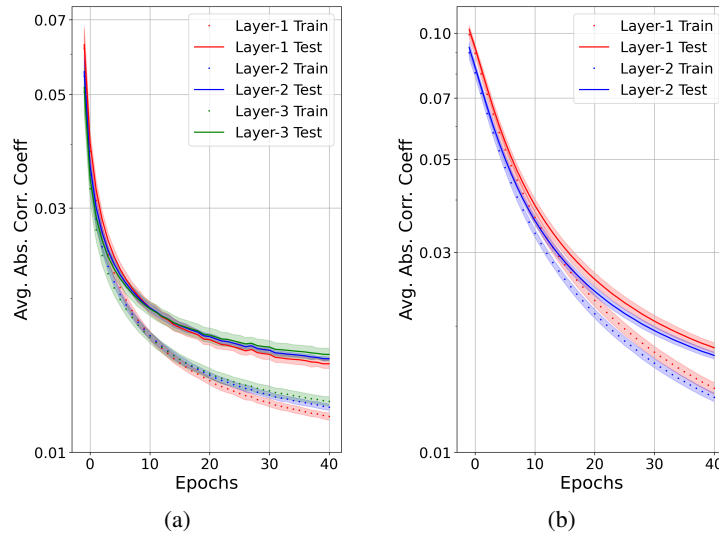


Figure 12: Evolution of the average absolute correlation between layer activations and output errors during backpropagation training of an MLP with three hidden layers, trained using cross-entropy loss. (a) CIFAR-10 dataset and (b) MNIST dataset. Despite the use of cross-entropy, the correlation decreases similarly to the MSE criterion.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [\[Yes\]](#)

Justification: The abstract and introduction claim EBD is a novel, MMSE-grounded framework for biologically plausible credit assignment, circumventing weight transport, leading to three-factor rules, integrating with CorInfoMax, and achieving competitive or better performance against other error-broadcast methods on benchmarks. These claims are supported by the theoretical development in Sections 2 and 3 (e.g., MMSE orthogonality, three-factor learning rule derivation in 3.1) and experimental results in Section 4 (Tables 1 and 2). The scope, including scalability as an area for further exploration, is also appropriately stated.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: Section 5 (Impact and Limitations”) explicitly discusses limitations concerning scalability, computational costs/hyperparameter complexity of EBD.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate ”Limitations” section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren’t acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: The theoretical grounding of EBD on the MMSE orthogonality principle is detailed in Section 2.2. Appendix A provides preliminaries on nonlinear MMSE estimation. Appendix B elaborates on the stochastic orthogonality condition, its sufficiency for MMSE optimality (Theorem B.1), and its application to network training, including arguments for using hidden unit activations under idealized conditions (Theorem B.2). These sections outline the assumptions and provide the theoretical arguments/proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: Section 4 outlines the experimental setup. Appendix I ("Supplementary on Numerical Experiments") provides extensive details regarding network architectures (I.1), hyperparameters for all presented models (EBD variants and baselines like BP, DFA) across datasets (I.3-I.7), optimizers, learning rate schedules, and other implementation specifics necessary to understand and reproduce the experiments. We also provide all codes as supplementary to the article.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

- (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
- (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
- (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the implementation of the proposed method in the supplementary material, accompanied by a README file that outlines the structure and usage instructions.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The datasets (MNIST, CIFAR-10) use standard train/test splits as detailed in Appendix I (lines 1051-1053). Appendix I extensively lists hyperparameters, their selection rationale or values for different models/datasets (e.g., I.3.4-I.3.7, I.4.3-I.4.4, I.5.3-I.5.4, I.6.3-I.6.4, I.7), optimizer types (Adam, SGD with momentum), and other training configurations (epochs, batch sizes).

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Experiments were conducted five times, and average and standard deviation values are reported in Tables. In Appendix I, Figures 5 and 6 explicitly state that they show mean results with standard deviation envelopes from these multiple runs. Table 2 reports mean and standard deviation for the 10-layer CorInfoMax-EBD results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: All experiments utilized a single NVIDIA Tesla V100 GPU with 32GB HBM2 memory, accessed via an HPC facility (details in Appendix I). Appendix I.9 provides specific execution time examples for the more computationally intensive CorInfoMax-EBD models. Appendix I.7 further provides relative runtime comparisons for other models and methods.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: The research proposes a novel learning algorithm and evaluates it using standard public datasets. The work is foundational. The "Impact and Limitations" section (Section 5, lines 315-318) outlines the positive aim of advancing the fields and notes that specific negative societal impacts directly from EBD mechanisms are not identified beyond general ML considerations.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [\[Yes\]](#)

Justification: Section 5 ("Impact and Limitations") discusses the positive societal impact by stating the aim to advance Machine Learning and Computational Neuroscience. It also addresses the negative aspect by stating that, as a foundational algorithm, no specific negative societal impacts directly attributable to EBD are identified beyond general considerations common to advancements in machine learning.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [\[NA\]](#)

Justification: This research proposes a new learning algorithm (EBD) and primarily evaluates it on standard, publicly available datasets (MNIST, CIFAR-10). The paper does not introduce or release new large-scale datasets or pre-trained models that would typically carry a high risk for misuse requiring specific safeguards (such as those for large generative models or models trained on sensitive scraped data).

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [\[Yes\]](#)

Justification: The sources for the standard datasets used (MNIST [52], CIFAR-10 [45]) are cited in Section 4 and their licenses/terms of public availability (CC BY-SA 3.0 for MNIST, public availability for research for CIFAR-10) are noted in Appendix I. The external code repository from Bozkurt et al. [45] used as a basis for the CorInfoMax-EBD implementation is cited, and its observed licensing status (no explicit license found in the public repository at the time of access) is transparently stated in Appendix I.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: The primary new assets are the proposed EBD algorithm and its variations, along with their software implementation. As stated in the filled checklist answer for question 5, the code implementation and a README file with usage instructions are provided in the supplementary material.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: This research does not involve crowdsourcing or direct research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This research does not involve direct research with human subjects that would necessitate IRB approval or a discussion of participant risks.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs were not used as a component of the core research methodology, theoretical development, or experimental analysis presented in this paper. Any use of LLMs was restricted to assistance with writing and editing the manuscript, which does not require declaration per NeurIPS policy.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>) for what should or should not be described.