Automatic discovery of local learning rules

Andrea Perin[*] Aalto University
Fabio Anselmi[†] Università degli Studi di Trieste
Stéphane Deny[†] Aalto University

Abstract

We present a general method to replace backpropagation with meta-learned local learning rules, applicable in principle to any deep neural architecture and task. The method automatically discovers local loss functions specific to each layer by (1) constructing a dictionary of monomials of local tensor variables, (2) regressing over coefficients weighting these monomials to build layerwise polynomial losses whose gradient updates together conspire to satisfy a target global objective function. We demonstrate the practicality of our approach on minimal problems of reconstruction, classification, and self-supervised learning.

1 Introduction

The plausibility of backpropagation as a model of biological learning is disputed [e.g., 6, 10, 11, 20], mainly because it demands sharing gradient information across all layers of deep networks at each update step. A more plausible candidate would have to be local, requiring limited information sharing across the network [3]. From a machine learning perspective, replacing backpropagation with local learning rules—when supported by suitable hardware [15]—offers clear advantages in memory efficiency and speed: (1) it reduces the need for cross-layer information transfer, and (2) it enables all layers to be updated in parallel.

Different approaches have been proposed to replace backpropagation with local learning rules. For simple problems and shallow networks, closed-form learning rules have been derived analytically [e.g., 16, 17, 8, 18]. For more complex problems and deeper networks, hand-engineered local learning rules have shown empirical success in specific settings [e.g., 12, 28, 23, 30]. Methods have also been proposed to *automatically discover* effective local learning rules [e.g., 14, 7, 26, 13, 21, 5]. Typically, these methods train auxiliary multi-layer perceptrons (MLPs) to implement local weight updates. These methods, however, do not produce interpretable equations for the learning rules, as MLPs perform highly non-linear and complex transformations of their inputs.

Here, we start from the observation that, for simple problems, analytical solutions take the form of polynomials in the problem variables (weight matrices, layer pre- and post-activations). We postulate that, for more complex problems (deeper networks and complex tasks), approximate solutions exist that are expressible in polynomials of accessible local variables. We design a method that turns the construction of local learning rules into a systematic and learnable procedure: for each layer, (1) we automatically generate a dictionary of monomials of local tensor variables and (2) we learn the coefficients weighting these elements to build polynomial local losses. The gradients of these losses induce local updates that collectively satisfy a target global objective. We demonstrate the feasibility of our approach on minimal problems of reconstruction, classification, and self-supervised learning.

2 Method

We start by choosing a *global loss function* $\mathcal{L}$ that we wish to optimize, and a network architecture f_θ to train. For each layer in the network, we define its *neighbourhood of degree* d as the set of

[*]Corresponding author: andrea.perin@aalto.fi. [†]These authors jointly and equally supervised the work.

Conference on Physics and AI at Stanford University (PAI 2026).

variables including its inputs, weights, and outputs, as well as those of all the layers that are at most d steps away in the computational graph (Fig. 1). We then construct scalar functions of elements in its neighbourhood involving at most order K, possibly repeated, terms.

Specifically, we consider *trace monomials*, defined as those scalars that are computed by taking up to K, possibly repeated, neighbourhood elements, and contracting all their indexes two by two according to shape compatibility. For example, for a linear layer with weight matrix $W \in \mathbb{R}^{d\times m}$, input $x \in \mathbb{R}^m$ and output $y \in \mathbb{R}^d$, a few possible trace monomials are

$$\widehat{\mathcal{L}}_1 = y^T W x = \sum_{ij} W_{ij} x_j y_i, \quad \widehat{\mathcal{L}}_2 = ||x||^2 = \sum_i x_i x_i, \quad \widehat{\mathcal{L}}_3 = ||y^T W||^2 = \sum_{ijk} W_{ij} W_{kj} y_k y_i,$$

where each contracted index appears *twice and only twice* in the expression. We develop a method, inspired by Penrose tensor diagram notation [19], to generate all trace monomials up to order K. Using these monomials as dictionary elements, for each layer ℓ in the network we construct a *local loss function*, defined as a weighted sum of its trace monomials, $\widehat{\mathcal{L}}^\ell = \sum_n \alpha_n^\ell \widehat{\mathcal{L}}_n^\ell$. We refer to the weights α_n^ℓ as *rule coefficients* for layer ℓ, and initialize them to 0.

We now describe a *training episode*. We start by randomly initializing the network parameters. Each parameter is then recursively updated by using the gradients obtained from their respective local loss function, as in standard gradient descent, for a certain number T of time steps. For each layer ℓ in the network (dropping the ℓ superscript to simplify the notation):

$$\theta^{(t+1)}(\alpha) \leftarrow \theta^{(t)}(\alpha) + \sum_{n=1}^{N} \alpha_n \nabla_\theta \widehat{\mathcal{L}}_n(\alpha).$$

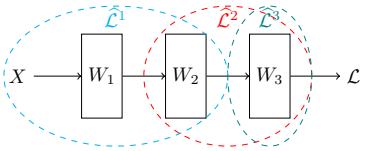


Figure 1: Schematic illustrating key notations used in the method. Here, the degree of the neighbourhood is 2, as indicated by the ellipses.

We compute the global loss $\mathcal{L}$ on these resulting weights, and its gradients with respect to each layer's rule coefficients. We update each layer's rule coefficients, and conclude the training episode. In practice, a number N_{seed} of these episodes takes place at once using the same rules, thus batching the process over different copies of the network parameters.

We repeat this process, each time running a new training episode with the freshly updated rule coefficients and a new set of initializations of the network parameters. This outer loop is executed until the value of $\mathcal{L}$ is below a chosen threshold.

2.1 Implementation details

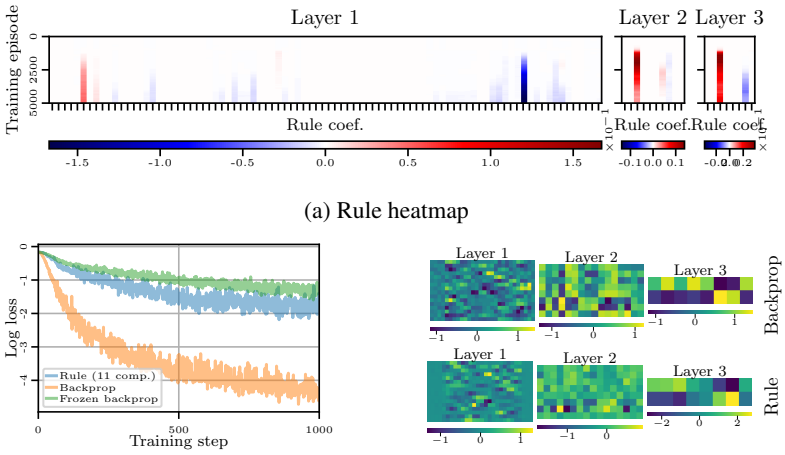
Increasing time horizon: During training, we progressively increase the number of time steps T in a training episode. Specifically, we start with $T = 25$, and increase it by 1 every 10 training episodes. Numerically, we rely on checkpointed backprop [27, 25], as implemented in the *Equinox* library [9].

Early stopping: For a given initialization of the network parameters, a specific rule may quickly lead to divergent weights or, on the contrary, to vanishing weights. We stop a training episode as soon as the max-norm of *any* of the greedily-updated weights becomes either larger or smaller than a chosen maximum/minimum safety value. This saves compute time, but most importantly ensures that the resulting gradients for the rule coefficients are not themselves exploding or vanishing. We use a maximum of 10^2 and a minimum of 10^{-2}.

Minimax training: At every initialization of the network parameters, we sample N_{seed} different copies in parallel. After following the current rule for the necessary number of time steps, we keep a subset $N_{\text{keep}} < N_{\text{seed}}$ of the *worst-performing* seeds. The gradients for the rule coefficients are the average of the N_{keep} worst performing seeds. We set $N_{\text{seed}} = 256$ and $N_{\text{keep}} = 16$.

Learning rate scaling: For the inner loop, we use SGD with unit learning rate. For the rule coefficients, we use Nesterov SGD clipped at 0.01 with momentum 0.9, learning rate 0.01 and staircase exponential decay schedule. Additionally, for an episode of length T, we scale the corresponding update by $1/\sqrt{T}$. This is the variance scaling for a random walk taking T steps.

Sparse prior on rule coefficients: Numerically, we form rule coefficients α as a Hadamard product between two vectors α_1 and α_2 on which gradient updates are applied. This procedure results in an implicit sparse prior [1] on rule coefficients, which we found to improve optimization results. α_1 is randomly initialized and α_2 is set to 0 at initialization.



(a) Rule heatmap

(b) Loss comparison

(c) Learned weights

Figure 2: Classification task. (a) Evolution of α coefficients throughout training, for a representative run. (b) Loss evolution throughout application of the learned rules (blue), compared to backpropagation (orange) and backpropagation with first layer frozen (green). (c) Weights of the network after training with backpropagation (top) vs. our learned rules (bottom).

3 Results

To keep computational load minimal, we use a "sliced-MNIST" dataset, in which we take digits of class 0 and 1 only, and extract the 15th row of pixels in the image, resulting in an input of dimensionality 28. We mean-center the dataset, and, in order to make the classification problem more challenging, we randomly flip the sign of each sample vector. Architecture-wise, we use a three-layer MLP with ReLU activations σ, and we consider neighbourhoods of degree 2 (Fig. 1). Layer 1 has access to variables $\{x, W_1, \sigma(W_1 x), W_2, \sigma(W_2 \sigma(W_1 x))\}$. Layers 2 and 3 have access to the whole downstream computational graph. We use a batch size of 64, and K = 4,2,2 for the respective layers. The self-supervised learning task is an exception, having neighbourhood of degree 1 and K=4,4,4.

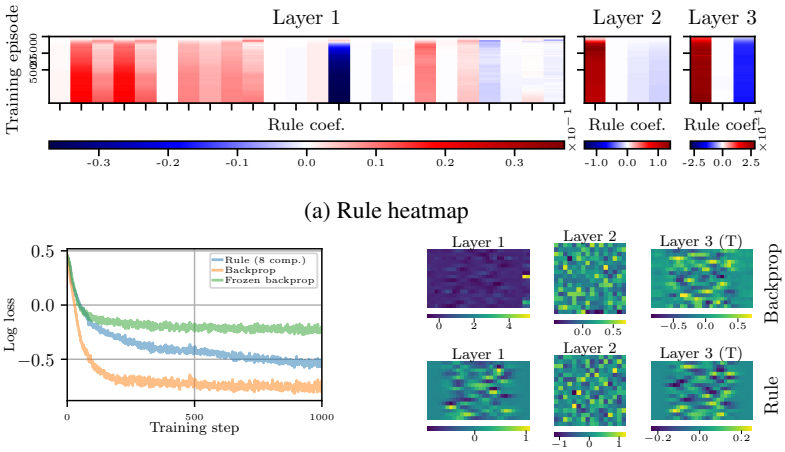
Classification We test our method on a binary classification task consisting in predicting digit class. We use a 3-layer MLP with size $W_1 : (16, 28), W_2 : (8, 16), W_3 : (2, 8)$, and define the global loss as the classical cross-entropy loss. Local learning rules are learned on a training set and subsequently applied on a separate testing set. During training, α coefficients are sparsely selected by the method (Fig. 2), leading to interpretable local loss equations. As local rules are subsequently applied the loss decreases slower than it does with backpropagation. However, it decreases faster than it does with a degree-2 local version of backpropagation where the first layer is frozen to initial weights. This demonstrates that our local rules cooperate in a non-trivial way. Final classification accuracy obtained with our method (0.997) is on par with backpropagation (1.00). Moreover, the learned weights are structurally similar to those obtained with backpropagation.

Reconstruction We train a non-linear autoencoder on a reconstruction task on the same dataset. We use a 3-layer MLP with size $W_1 : (16, 28), W_2 : (16, 16), W_3 : (28, 16)$. The target is the L2 loss between samples and their reconstructions. Results are reported in Fig. 3.

Self-supervised learning We train a 3-layer MLP with size $W_1 : (16, 28), W_2 : (8, 16), W_3 : (8, 8)$, on a self-supervised learning task, where augmentations are random shifts and flips, using the Barlow Twins loss function [29]. For this specific case, we use neighborhoods of size 1 for all layers, thus enforcing *purely local update rules*. Results are reported in Fig. 4.

4 Discussion

We presented a general method to learn equations for local learning rules. Other methods, such as SINDy [4], share a similar aim of inferring equations from data. They are usually employed to

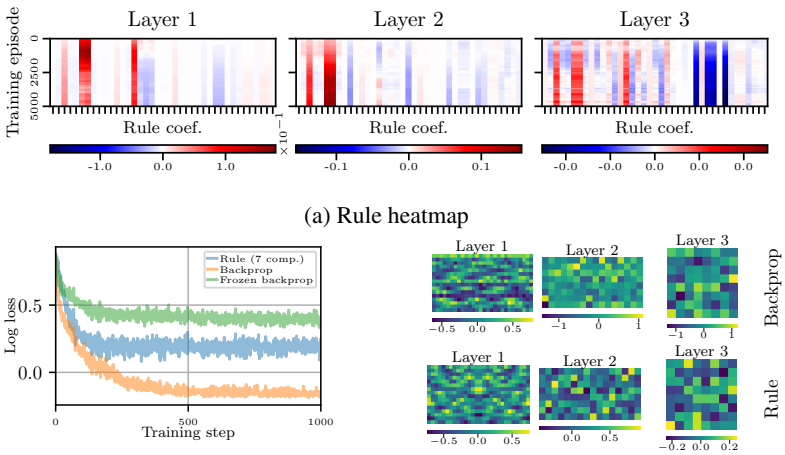


(a) Rule heatmap

(b) Loss comparison

(c) Learned weights

Figure 3: Reconstruction task. Same caption as Fig. 2. (T) stands for transpose.



(a) Rule heatmap

(b) Loss comparison

(c) Learned weights

Figure 4: Self-supervised learning task. Same caption as Fig. 2.

reverse-engineer dynamical systems from observations of their trajectories. Therein lies a crucial distinction: our method does not use trajectory data as a groundtruth signal, and instead *relies solely on the signal given by the global loss*. A trajectory-based method, on the contrary, would pose both practical as well as theoretical problems: practical because of the data requirements (a whole trajectory, in our case, would consist of many copies of the weights of a model), and theoretical because what we want to approximate is not the backprop-induced dynamics (which could be indeed impossible to replicate with local losses), but instead the backprop-induced *performance*. In analogy with physics, we wish to apply constraints on the *macroscopic observable of the complex system* (the network's performance as measured by the global loss) rather than on its microscopic configurations.

Future work will focus on scaling up the method to more realistic architectures and datasets. Improving the stability of the training process should be prioritized, as it currently is quite sensitive to hyperparameter choices, in part because the optimization landscape is fractal [24] (but see [22]). Another promising direction is to analyse the algebraic structure and interpretability of the rules learned by the method and, crucially, their *transferability* across datasets and architectures. If such transferability holds, the method could be used to discover local rules on simple datasets and then apply them to more complex ones. Lastly, we mention a possible alternative training method, known as the *adjoint* method [as used in, e.g., 2], which relies on the implicit function theorem, and unlike the greedy procedure we presented, does not require to keep track of the whole trajectory.

References

[1] Nicola Aladrah, Emanuele Ballarin, Matteo Biagetti, Alessio Ansuini, Alberto d'Onofrio, and Fabio Anselmi. Understanding and inverse design of implicit bias in stochastic learning: a geometric perspective. *arXiv preprint arXiv:2601.06597*, 2026.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[3] Colin Bredenberg, Ezekiel Williams, Cristina Savin, Blake Richards, and Guillaume Lajoie. Formalizing locality for normative synaptic plasticity models. *Advances in neural information processing systems*, 36:5653–5684, 2023.

[4] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[5] Basile Confavreux, Poornima Ramesh, Pedro J Goncalves, Jakob H Macke, and Tim Vogels. Meta-learning families of plasticity rules in recurrent spiking networks using simulation-based inference. *Advances in Neural Information Processing Systems*, 36:13545–13558, 2023.

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

[7] Keren Gu, Sam Greydanus, Luke Metz, Niru Maheswaranathan, and Jascha Sohl-Dickstein. Meta-learning biologically plausible semi-supervised update rules. *bioRxiv*, pages 2019–12, 2019.

[8] Aapo Hyvärinen and Erkki Oja. Independent component analysis by general nonlinear hebbian-like learning rules. *signal processing*, 64(3):301–313, 1998.

[9] Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations. *Differentiable Programming workshop at Neural Information Processing Systems 2021*, 2021.

[10] 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.

[11] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[12] Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. *Advances in neural information processing systems*, 32, 2019.

[13] Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

[14] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. *International Conference on Learning Representations*, 2019.

[15] Ali Momeni, Babak Rahmani, Benjamin Scellier, Logan G Wright, Peter L McMahon, Clara C Wanjura, Yuhang Li, Anas Skalli, Natalia G Berloff, Tatsuhiro Onodera, et al. Training of physical neural networks. *Nature*, 645(8079):53–61, 2025.

[16] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[17] Erkki Oja. Neural networks, principal components, and subspaces. *International journal of neural systems*, 1(01):61–68, 1989.

[18] Cengiz Pehlevan, Tao Hu, and Dmitri B Chklovskii. A hebbian/anti-hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data. *Neural computation*, 27(7):1461–1495, 2015.

[19] Roger Penrose et al. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1(221-244):3, 1971.

[20] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11):1761–1770, 2019.

[21] Navid Shervani-Tabar and Robert Rosenbaum. Meta-learning biologically plausible plasticity rules with random feedback pathways. *Nature Communications*, 14(1):1805, 2023.

[22] Zakhar Shumaylov, Peter Zaika, Philipp Scholl, Gitta Kutyniok, Lior Horesh, and Carola-Bibiane Schönlieb. When is a system discoverable from data? discovery requires chaos. *arXiv preprint arXiv:2511.08860*, 2025.

[23] Shoaib Ahmed Siddiqui, David Krueger, Yann LeCun, and Stéphane Deny. Blockwise self-supervised learning at scale. *Transactions in Machine Learning Research*, 2024.

[24] Jascha Sohl-Dickstein. The boundary of neural network trainability is fractal. *arXiv preprint arXiv:2402.06184*, 2024.

[25] Philipp Stumm and Andrea Walther. New algorithms for optimal online checkpointing. *SIAM Journal on Scientific Computing*, 32(2):836–854, 2010.

[26] Danil Tyulmankov, Guangyu Robert Yang, and LF Abbott. Meta-learning synaptic plasticity and memory addressing for continual familiarity detection. *Neuron*, 110(3):544–557, 2022.

[27] Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009.

[28] Yuwen Xiong, Mengye Ren, and Raquel Urtasun. LoCo: Local contrastive representation learning. *Advances in neural information processing systems*, 33:11142–11153, 2020.

[29] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International conference on machine learning*, pages 12310–12320. PMLR, 2021.

[30] Wu S Zihan, Ariane Delrocq, Wulfram Gerstner, and Guillaume Bellec. Can local learning match self-supervised backpropagation? *arXiv preprint arXiv:2601.21683*, 2026.