THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Local Learning Rules

For Deep Neural Networks with Two-State Neurons

Rasmus Kjær Høier

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

Local Learning Rules
For Deep Neural Networks with Two-State Neurons

Rasmus Kjær Høier
ISBN 978-91-8103-176-8

Acknowledgements, dedications, and similar personal statements in this thesis, reflect the author's own views.

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5634
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Printed by Chalmers Digital Printing
Gothenburg, Sweden, March 2025

Local Learning Rules

For Deep Neural Networks with Two-State Neurons

Rasmus Kjær Høier
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The way artificial neural networks are trained with backpropagation requires a degree of synchronization of operations, and non-local knowledge of the computational graph of the network, which is infeasible in noisy asynchronous circuitry (be it biological, analog electronic or optical). Learning algorithms based on temporal or spatial neural activity differences allow estimating gradients, and hence learning, without these problematic requirements.

In this thesis, we explore a number of such alternative learning algorithms. Paper A presents a variation of contrastive Hebbian learning, which achieves Lipschitz-1 hidden layers by construction. Paper B focuses on efficient training on traditional digital hardware by presenting a variant of backpropagation compatible with quantized weights. Paper C returns to the topic of contrastive Hebbian learning by presenting a new local learning algorithm for training feedforward networks based on neurons possessing two internal states. These dyadic neurons perform credit assignment by encoding errors as differences and predictions as averages of the internal states. Paper D provides a new variation of dual propagation and provides derivations of both the original and the new variant. Paper E presents a general framework for *dyadic learning*, which encompasses dual propagation in feedforward models and equilibrium propagation (a well-known variant of contrastive Hebbian learning) on Hopfield models as special cases while also being applicable to arbitrarily connected networks. The case of a skew-symmetric Hopfield network is found to be particularly intriguing as it, like the model from paper A, provides Lipschitz-1 layers by construction.

Keywords: Contrastive Hebbian learning, lifted neural networks, local learning, biologically inspired learning, Hopfield networks, quantized training.

List of Publications

This thesis is based on the following publications:

[A] **R. Høier** and C. Zach, "Lifted Regression/Reconstruction Networks". BMVC 2020.

[B] H. Le, **R. Høier**, C.T. Lin and C. Zach, "AdaSTE: An Adaptive Straight-Through Estimator to Train Binary Neural Networks". CVPR 2022.

[C] **R. Høier**, D. Staudt and C. Zach, "Dual Propagation: Accelerating Contrastive Hebbian Learning with Dyadic Neurons". ICML 2023.

[D] **R. Høier** and C. Zach, "Two Tales of Single-Phase Contrastive Hebbian Learning". ICML 2024.

[E] **R. Høier**, K. Kalinin, M. Ernoult and C. Zach, "Dyadic Learning In Recurrent and Feedforward Models". MLNCP workshop @NeurIPS 2024.

Other publications by the author, not included in this thesis, are:

[F] **R. Høier** and C. Zach, "A Lagrangian Perspective on Dual Propagation". *First Workshop on Machine Learning with New Compute Paradigms at Neurips*, New Orleans, December 2023.

[G] M. Ernoult, **R. Høier**, J. Kendall, "A cookbook for hardware-friendly implicit learning on static data". *Second Workshop on Machine Learning with New Compute Paradigms at Neurips*, Vancouver, December 2024.

Contents

Acknowledgments

First of all, thank you, Christopher, for the stimulating discussions and your generous sharing of ideas and mathematical insights. To the other members (past and present) of the Zach sub-group, Huu, Xixi, Sophia, Yara, Alex, and James, thank you for the helpful feedback and discussions and for your company these past 5 years. Thank you, to all the other wonderful past and present members of the computer vision group! Lukas, Georg, Kunal, José, Sofie, Jennifer, Josef, Ida, Fredrik, Erik, Roman, both Davids, both Carls, Richard and Victor.

Working in a fairly niche field, it was eye-opening for me to meet researchers from other institutions working on similar problems at workshops and conferences post-pandemic. Thank you, Maxence, for the many hours you dedicated to discussing learning algorithms with me. Your insights were crucial in shaping my understanding of the field. Thank you, Jannes, for giving me the opportunity to learn about hardware-algorithm co-design. Thank you, Kirill, for the whiteboard sessions and for insisting on the relevance of asymmetrically connected networks. Thank you, Ben and Jack, for discussions and new perspectives.

I am also grateful for my undergraduate thesis supervisors at Nuclear Physics in Lund, Hanno and Kevin, and my manager at MAXIV, Steve. Working with you gave me the confidence I needed to pursue research in a new field.

To my family, far-Kresten, mor-Bit, Marie, and Nikolaj. Thank you for your patience and support. When things were difficult, I tried to imagine how you would handle them, and it usually helped.

Finally, I want to thank my dear wife, Cibele. I don't think I could have done this without your support.

This research project was supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Part I

Overview

CHAPTER 1

Introduction

The programs encoded by artificial neural networks are fundamentally different from software written by humans. Rather than writing explicit instructions, we provide an over-parametrized function with a large amount of data and let gradient descent take care of *programming* the correct parameter values for the network. This view has led to artificial neural networks being referred to as software 2.0 [1], highlighting the fundamentally different way these programs are created. The qualitative differences between traditional software and this new kind of software also translate to qualitatively different hardware requirements. While manual programming requires reproducible hardware with discrete states and a user-friendly programming interface, neural network training does not impose these requirements. In principle, all you need is hardware with many tunable parameters capable of providing an estimate of the gradient of a loss function with respect to these parameters and, of course, large amounts of data.

The early 20th century saw a diverse range of analog computing platforms developed. Aided by their superior programmability and the invention of the transistor, digital computers became dominant in the 1950s. A parallel evolution from physically wiring in programs to externally stored programs

(such as punched cards) and digitally stored programs greatly aided the success of digital computing. Early artificial neural networks were largely simulated on digital hardware, for example, Rosenblatts perceptron [2], Fukushimas Neocognitron [3] and the connectionist models of the parallel distributed processing group [4]. Following the success of [5]–[7], GPUs have been preferred over CPUs for artificial neural network training and deployment.

The energy consumed by these chips has motivated a number of efforts toward developing more energy-efficient digital chips tailored to specific use cases, as well as more radical efforts aimed at developing analog and optical hardware accelerators for AI. The more radical approaches could potentially require orders of magnitude less energy than digital computing but impose new constraints. In many cases, one will only have access to the final steady state reached during inference, while the trajectory taken and intermediate derivatives are inaccessible. This necessitates developing alternatives to the backpropagation of errors algorithm, which rely solely on information regarding the final states of the network.

The work in this thesis is primarily focused on developing learning algorithms that require little to no inter-neuron synchronization and only require knowledge of the neural states at equilibrium. Although this is motivated by the ongoing development of new hardware accelerators, the focus here is on algorithms, and hardware is only addressed in passing.

While exotic hardware could potentially lead to massive gains in the speed and energy efficiency of AI models, it is hard to predict when or even if a moonshot technology will become mature enough for widespread adoption. Training quantized neural networks is a more pragmatic approach to the rising costs of AI, which can speed up training and inference of neural networks on already existing digital hardware. This topic is explored in one of the included papers using many of the same mathematical tools as in the other papers.

1.1 Thesis outline

The first part of the thesis part I motivates the research topic and provides background information regarding the learning algorithms explored in the included articles. Chapter 2 mainly serves as an extended motivation and discusses the notion of locality with regard to synaptic updates and relates it to biology and hardware. Chapter 3 explores some important learning algorithms

from the literature (backpropagation, variants of contrastive Hebbian learning, and lifted network training) through the lens of constrained optimization. Chapter 4 presents a general framework for local learning, generalizing learning algorithms explored in the included research papers as well as one of the reference algorithms. Chapter 5 summarizes the included research papers, and chapter 6 contains the conclusion and future works. Part II consists of the included research papers.

1.2 Notation

This section explains the notation used in part I of the thesis. We consider a supervised learning setting where a dataset is made up of a set of inputs X and desired outputs Y (labels). For simplicity we omit indices and denote a single datapoint from the dataset by lowercase x and y. We use s to denote a vector of neural states and W to denote the network's adjacency matrix, usually referred to as the weight matrix. Biases are omitted to reduce clutter. s_k denotes the k-th element of s and W_{ij} denotes the strength of the synaptic connection from neuron j to neuron i.

When discussing layered networks in particular we overload the notation slightly and let s be a a list of layerwise state-vectors: $s = \{s_0, s_1, ..., s_L\}$, where $s_0 = x$. This will usually be clear from context, but to avoid unnecessary confusion, we will use subscript l rather than subscript k in the layered setting. We denote the number of states in a layer l as d_l, such that $s_l \in \mathbb{R}^{d_l}$. The set of weights is defined as $W = \{W_l^{d_{l+1} \times d_l}\}_{l=0}^{L-1}$. It will occasionally be convenient to distinguish between neural states before and after the activation function is applied. We use a, defined through $s := f(a)$, to denote the neural pre-activations.

Note that the notation used in part I differs somewhat from the notation used in the included papers in part II (more so for the first three papers and less so for the last two papers).

CHAPTER 2

Background

While backpropagation is immensely powerful, it requires a strictly synchronized order of operations (a bottom-up forward pass and a top-down backward pass through the network), as well as knowledge of derivatives of non-linearities and sharing of weights between the forward and backward passes (referred to as the weight transport problem). These requirements are easy to fulfill on general-purpose programmable digital hardware such as CPUs, GPUs, and TPUs, though these devices are highly energy demanding [8]. Furthermore, the biological implausibility of backpropagation was pointed out early on [9] and has since motivated a rich literature on less implausible alternatives. Interestingly, this somewhat speculative research on *biologically plausible* credit assignment has strong synergies with research on new computing paradigms for energy-efficient AI, which often share some of the same constraints as biological circuits.

2.1 The von Neumann bottleneck

In modern digital computers, memory and processing cores are physically separate, which makes it necessary to transfer data and instructions between

memory and processors. This separation of memory and processing units is a key aspect of the well-known Von Neumann architecture. The bus piping data between memory and processing units is referred to as the *von Neumann bottleneck* [10], as the limited bandwidth of this bus often becomes the limiting factor in program runtime. This problem is exacerbated by the fact that processor speeds have increased faster than data transfer rates. The von Neumann bottleneck not only slows down computation but also wastes energy because cores must idly stand by while waiting for data and instructions. Modern chip design is, of course, much more complex. For instance, the von Neumann bottleneck is partially alleviated by integrating smaller high-speed memory on-chip close to the processing units. Nonetheless, the rate at which data can be transferred remains a limiting factor.

2.2 Locality of synaptic plasticity

Co-locating memory and processing is one way to avoid the Von Neumann bottleneck, but this necessitates considering what information will be locally available to a given processing unit. In the context of learning this means neurons will have to asynchronously carry out computations using only information from neurons they are directly connected to.

It is common in the literature to motivate algorithms on the merits of locality, though a proper definition is often absent (We are guilty of this as well in the included papers). In the following, we consider a synaptic plasticity rule to be local in space if updating a synapse W_{ij}, connecting neurons i and j, only involves the states of neurons i and j. This is satisfied by all learning rules considered here, though architectures involving weight sharing (such as CNNs) make algorithms non-local in space. We consider a plasticity rule to be local in time if it does not require inferring distinct sets of states sequentially. For a more rigorous but also quite dense treatment of the topic of locality of synaptic plasticity, we recommend [11].

Table 2.1 classifies a few algorithms according to the above definitions of locality, whether they map to a single circuit, and whether they avoid requiring knowledge of the exact derivatives of activation functions.

Backpropagation is classified as spatially local here because the weight updates can be expressed in terms of activity and error states computed directly at the synapse, assuming the same circuit is used in the forward and

backward pass. As the forward and backward passes require quite different circuitry (This is discussed in chapter 3.1 and illustrated in Fig 3.1), a spatially non-local physical implementation may be more practical. This has to do with the forward pass involving the weights and elementwise nonlinearities, whereas the backward pass involves the transpose of the weights and elementwise multiplication by the derivative of the nonlinearities. Backpropagation is not temporally local, as activities and error states are computed in two distinct phases.

Contrastive Hebbian learning and equilibrium propagation use activities inferred in two distinct temporal phases to update weights. The two phases differ only with respect to boundary conditions and can consequently be run sequentially on the same physical circuitry, making them temporally non-local. The weight updates of contrastive Hebbian learning and equilibrium propagation only require knowledge of the states local to a given synapse, making them local in space. It is possible to make temporally local variants of these algorithms through the use of twin-circuits [12], [13]. However, the twin-circuit approach makes the algorithm non-local in space. Furthermore, constructing accurate physical twin circuits for large-scale networks is likely going to be challenging.

Dual propagation, the method of auxiliary coordinates, and predictive coding have distinct neural compartments, which makes it possible to compute predictions and errors simultaneously in a spatially and temporally local manner. Predictive coding and the method of auxiliary coordinates also require knowledge of the derivatives of the activation functions, which is problematic for hardware, where exact knowledge of device imperfections is not practically available.

All of these algorithms are explored in greater detail in chapter 3 and chapter 4.

2.3 Weight transport

The weight transport problem typically refers to the way backpropagation during the backward phase of learning requires transporting errors using the transpose of the weights used in the forward phase. This entails that if neuron i transmits activity through a scalar weight W_{ji} in the forward phase, then an error is transmitted through that same connection in the backward phase.

Algorithm	Spatial Locality	Temporal Locality	avoids f'
BP	✓	×	×
CHL & EP	✓	×	✓
DP	✓	✓	✓
MAC & PC	✓	✓	×

Table 2.1: Properties of different learning algorithms.

This strict symmetry does not agree with the connectivity of biological neural networks, as pointed out early on [9]. The weight transport problem also appears in the feedforward models trained by predictive coding and dual propagation. Although the dynamics are feedforward during inference, this is not the case during training, where activity is flowing forward through the weights while errors simultaneously flow backward through the transpose of the weights. The Hopfield models trained with contrastive Hebbian learning and equilibrium propagation also give rise to a variant of the weight transport problem by requiring symmetric connectivity $W_{ij} = W_{ji}$ [1]. Thus, the weight transport problem applies to all of the methods listed in table 2.1.

There is extensive research on the effect of applying ad hoc modifications to neural connectivity in order to circumvent the weight transport problem. [14] replaced the backward transposed weight matrix used in the backward phase of BP with a randomly initialized static matrix. [15] route error signals directly from the output layer to each hidden layer via static random weights. [16] modify the CHL dynamics to employ static random weights for all top-down connections. These ad hoc modifications do not outright prevent learning but do prevent the scaling to deep convolutional networks [17].

2.4 Hardware constraints of some non-von Neumann compute platforms

The kinds of hardware we are able to build today are very different from the brain, and hence, the built-in physical constraints differ. Blindly trying to

[1] Strictly speaking, you can apply EP and CHL to models with asymmetric weight matrices, but only the symmetric part of the weight matrix is actually used in the neural dynamics. This is discussed briefly in section 4.2.

emulate the constraints of one substrate (e.g. the brain) on a substrate that has completely different constraints is likely to be detrimental to performance [18]. For instance, biological neural circuits exhibit asymmetrical neural connectivity, but for some types of hardware, symmetric connectivity is not only possible but necessary. While the weight transport problem is critical in terms of biological plausibility, it is less of a concern when designing novel AI hardware accelerators. However, the locality constraints of section 2.2 are of critical importance. In the following, we will briefly consider two alternative types of computing hardware for which on-device learning requires alternative learning methods.

Hybrid analog-electronic/optical computing

Matrix multiplication is the core operation both for neural network inference and training. Recently, a hybrid analog optical circuit AIM[2] (analog iterative machine) has been proposed as a fast and energy efficient matrix-vector multiplication engine [19]. In this circuit, states are represented by analog currents and converted to light in order to carry out matrix-vector multiplication in the optical domain. Given an N element state vector, a beam splitter is applied to create N copies of the state vector. The resulting $N \times N$ array is then elementwise attenuated by a spatial light modulator array (SLM)[3] W before the now attenuated $N \times N$ set of states are converted to current and summed column-wise.

At 0.6 nanoseconds per iteration, matrix-vector multiplication is extremely rapid in this hybrid circuit. This makes it attractive for recurrently connected models such as Hopfield networks, which need multiple iterations of matrix-vector multiplication in order to converge to a fixed point. However, as all computations are carried out in the analog domain, it is not possible to learn the weights using backpropagation. Of the models listed in table 2.1, CHL, EP, and DP can be applied to train models on this type of hardware as they only require knowledge of the final fixed points. BP, MAC and PC are not amenable to this type of hardware as they require exact knowledge of the derivatives of the nonlinearities.

[2] More recent iterations employ the name AOC (Analog optical computer).

[3] Note that an SLM array can only attenuate, so signed weights require duplicating the states and having one SLM represent positive weights and another SLM represent negative weights.

The weight transport problem appears in a weaker form here as all the proposed methods require encoding a degree of weight sharing (between the weights and their transpose) when mapping weights to the SLM. In practice, device imperfections will make this weight sharing approximate at best.

Resistive networks

In resistive networks, the learnable parameters are the resistors connecting node voltages. Physical relaxation of such a circuit has been shown to correspond to minimizing a particular energy function called the pseudo-power [20], [21]. Interestingly, resistive networks can be reparametrized as Hopfield networks with symmetric connectivity [22], though the converse is not the case. I.e., resistive networks are a subset of Hopfield networks. By modeling weights as conductances in a resistive network, the weight transport problem is entirely bypassed, as edges are naturally bidirectional, satisfying $W_{ij} = W_{ji}$ by construction.

Resistive networks have been trained with equilibrium propagation [21]. More recently, resistive networks have been trained in hardware using a modified version of coupled learning (a variation of CHL) [12], where the distinct learning phases were carried out in parallel on *twin* circuits rather than sequentially on the same circuit.

CHAPTER 3

Deep learning as constrained optimization

The use of backpropagation in deep learning is today ubiquitous. However, the degree of synchronization between layers during the forward and backward pass and the need to compute derivatives of nonlinearities makes it impractical for some alternative compute paradigms. In this chapter, we will look at supervised end-to-end credit assignment through the lens of constrained optimization. First, in section 3.1, we look at two derivations of backpropagation in artificial neural networks. In section 3.2, we look at *local* alternatives to backpropagation based on the concept of energy minimization.

3.1 Two views on backpropagation

We consider the problem of learning the parameters of a deep feedforward neural network from a dataset (X, Y). For brevity, we use x and y to denote an arbitrary data point from this dataset. We consider a multi-layer perceptron with layerwise state-vectors $s_0 = x$, $a_l = W_{l-1}s_{l-1}$ and $s_l = f_l(a_{l-1})$ for $1 \leq l \leq L$, where f_l is an elementwise activation function. We denote the number of states in a layer l as d_l, such that $s_l \in \mathbb{R}^{d_l}$. The goal is to learn parameters $W = \{W_l^{d_{l+1} \times d_l}\}_{l=0}^{L-1}$ that minimize a loss function $\ell(s_L, y)$.

Note that although we here consider backpropagation in the context of feedforward networks, it can also be applied to recurrent neural networks by unrolling the temporal dynamics [23].

Using the chain rule

We find it convenient to use denominator layout, which means $\partial \ell / \partial a_l$ and $\partial \ell / \partial W_l$ will have the same shape as a_l and W_l respectively (in numerator layout, the dimensions would be transposed). The use of denominator layout also means that we must write the chain rule in opposite order from what is done in single variable calculus (e.g. $\frac{\partial p(q(x))}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial p}{\partial q}$ rather than $\frac{\partial p(q(x))}{\partial x} = \frac{\partial p}{\partial q}\frac{\partial q}{\partial x}$.

The goal here is to derive the gradient of the loss by repeatedly applying the chain rule.

$$\min_W \ell(s_L(W, x), y), \quad \text{where for } k = 1, ..., L \quad s_l = f_l(W_{l-1} s_{l-1}) \tag{$\mathcal{P}_1$}$$

For simplicity, we consider a single data point and do not limit ourselves to a specific loss function.

It is convenient first to find the expression for the partial derivatives of the preactivations $\partial \ell \partial a_l$ of each layer in the network. In the following $\odot$ is used to denote partial derivatives.

$$\begin{aligned} \frac{\partial \ell}{\partial a_L} &= \frac{\partial s_L}{\partial a_L}\frac{\partial \ell}{\partial s_L} = f_L'(a_L) \odot \ell' \\ &:= \delta_L \end{aligned} \tag{3.1}$$

When computing the partial derivative for the preceding layer, $L-1$, we reuse that we already computed the partial derivative with respect to a_L.

$$\begin{aligned} \frac{\partial \ell}{\partial a_{L-1}} &= \frac{\partial s_{L-1}}{\partial a_{L-1}}\frac{\partial a_L}{\partial s_{L-1}} \underbrace{\frac{\partial s_L}{\partial a_L}\frac{\partial \ell}{\partial s_L}}_{\delta_L} = f_{L-1}'(a_{L-1}) \odot (W_{L-1})^\top \delta_L \\ &:= \delta_{L-1} \end{aligned} \tag{3.2}$$

We note that each time we go one step further backward, we reuse the partial derivatives computed at the previous layer, yielding the recursive relation.

$$\delta_l := \frac{\partial \ell}{\partial a_l} = f_l'(a_l) \odot W_l^\top \delta_{l+1} \tag{3.3}$$

Finally, applying the chain rule one step further through the linear transformation $a_{l+1} = W_l s_l$ gives

$$\frac{\partial \ell}{\partial W_l} = \frac{\partial a_{l+1}}{\partial W_l}\frac{\partial \ell}{\partial a_{l+1}} = \delta_{l+1} s_l^\top. \tag{3.4}$$

The computational graph of backprop is illustrated in the cartoon in Fig 3.1. The states are computed in a sequential manner, with different operations used in the forward (matrix multiplication using the weights and elementwise nonlinearities) and backward pass (matrix multiplication using the transposed weights and elementwise multiplication by the derivative of the activation function). Note that the preactivations $a_l = W_{l-1}s_{l-1}$ are reused during the backward pass. While the weights W_l transport activity in the forward pass, in the backward pass the transpose of the weights $W_l^\top$ transport error vectors δ_{l+1}. This is the classical example of the weight transport problem mentioned in section 2.3. This synchronized order of operations, the different types of operations carried out in the two phases, and the sharing of weights and preactivations across the forward and the backward pass make implementing backpropagation directly in physical hardware challenging.

Backpropagation in quantized neural networks

A common strategy for reducing the computational cost of running neural networks on digital hardware is to quantize weights and/or activities. For both activities and weights, this can be achieved by employing a quantized activation function such as the sign function (note that weights usually do not have activation functions). Incorporating discrete activation functions can speed up inference significantly, as multiplication operations can be replaced by addition and subtraction operations. The downside is that vanilla backpropagation won't work as the gradient of the sign function is either zero or infinity (at the origin). Paper B deals specifically with the case where weights are binarized to {-1, 1} and presents a principled way that permits error signals to flow through the quantizing activation function.

The Lagrangian method

An alternative approach to deriving the state and weight updates of backpropagation-based training is to employ the method of Lagrange multipliers [24]. This method frames problem $\mathcal{P}_1$ as a constrained optimization problem where the behavior of the network (in this case, nonlinear feedforward connectivity) is enforced via a set of equality constraints.

$$\min_W \ell(s_L(W, x), y), \quad \text{s.t. } s_l = f_l(W_{l-1}s_{l-1}) \quad \forall l \in \{1, 2, \ldots, L\} \qquad (\mathcal{P}_2)$$

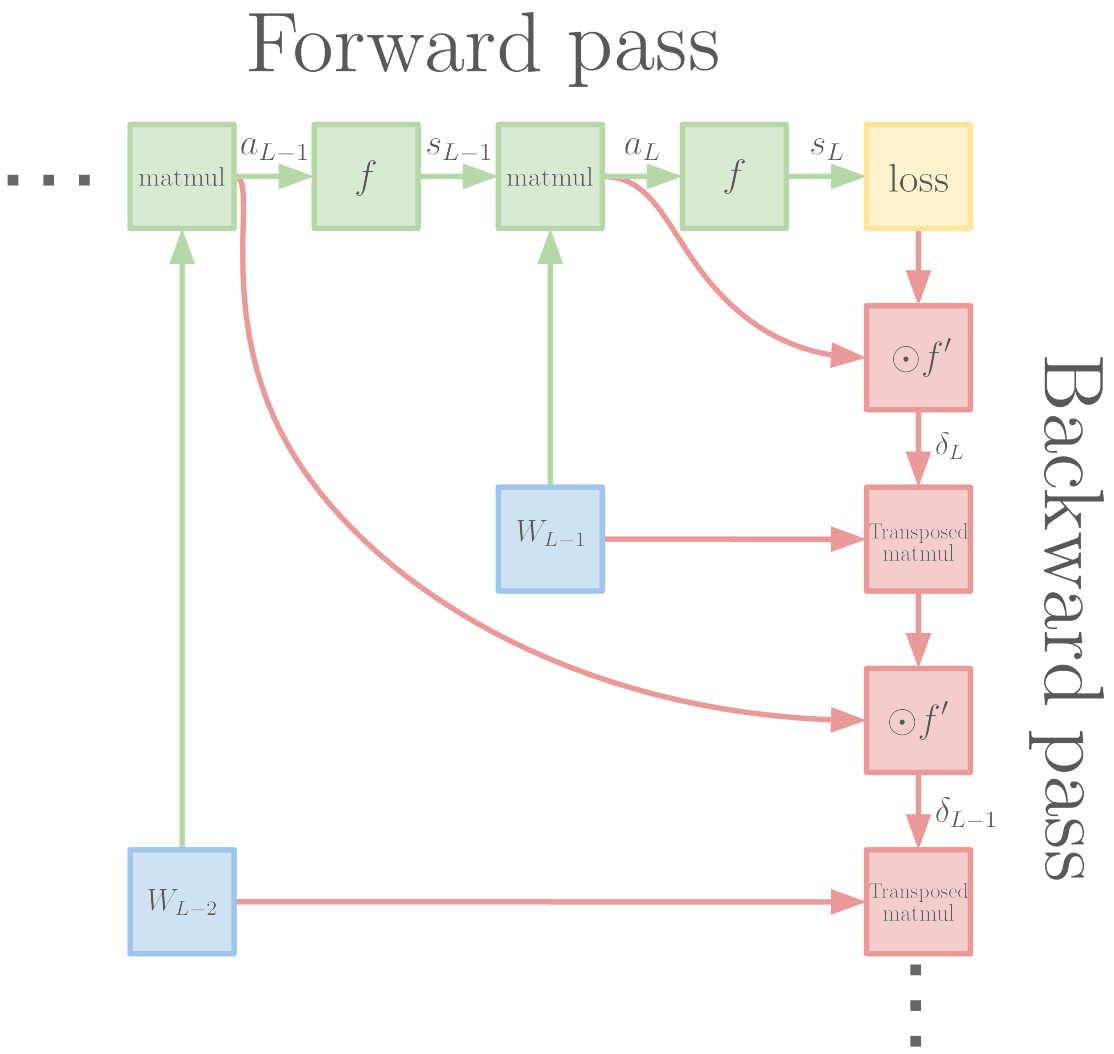


Figure 3.1: A cartoon of the forward and backward pass through the last two layers of a deep neural network. In the forward pass (green), activity is propagated through matrix multiplication and elementwise nonlinearities. In the backward pass, errors are propagated backward through the layers via the transpose of the weights (red). The preactivations $a_l = W_{l-1}s_{l-1}$ are reused during the backward pass in order to evaluate the derivative of the activation function, which is multiplied onto the backpropagated signal elementwise.

The method of Lagrange multipliers turns this constrained optimization problem into an unconstrained problem by constructing a new objective $\mathcal{L}$ in which the constraints are *baked in*.

$$\mathcal{L}(W, s, \delta) = \ell(s_L) + \sum_{l=1}^{L} \lambda_l^\top (s_l - f_l(W_{l-1} s_{l-1}))$$

Forward and backward dynamics follow from stationarity of $\mathcal{L}$.

$$\frac{\partial \mathcal{L}}{\partial \lambda_l} = 0 \Rightarrow s_l = f_l(W_{l-1} s_{l-1}) \tag{3.5}$$

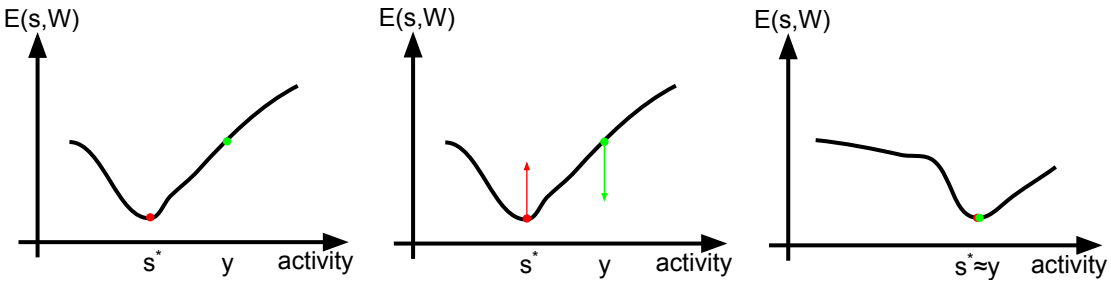
$$\frac{\partial \mathcal{L}}{\partial s_l} = 0 \Rightarrow \lambda_l = W_l^\top \lambda_{l+1} \odot f_l'(W_{l-1} s_{l-1}) \tag{3.6}$$

$$\frac{\partial \mathcal{L}}{\partial W_l} = (\lambda_{l+1} \odot f_{l+1}'(W_l s_l)) s_l^\top \tag{3.7}$$

This gives the same weight updates as the chain rule (with $\lambda_{l+1} \odot f_{l+1}'(W_l s_l) = \delta_l$).

Regarding the synchronicity of backprop

It is worth noting that the Lagrangian perspective does not impose a particular order of operations. Although the most efficient way (at least on digital hardware) is to compute the states and the weight updates via a forward pass through the layers followed by a backward pass, it is not the only feasible *schedule*. It is straightforward to verify that even choosing the update sequence at which $\{s_l, \delta_l\}$ is updated at random will result in the desired final states, assuming sufficiently many updates are made, though such an algorithm strictly speaking would not be what is typically understood by backpropagation. This can be understood from the fact that the computational graph shown in Fig 3.1 has no loops. Consequently, information only flows in one direction in this unfolded graph. This is exploited in the included paper D, which presents a derivation of the dual propagation algorithm, which can be viewed as a reparametrization of this Lagrangian formulation of backpropagation. A benefit of this particular reparametrization is that it not only avoids the need for strict synchronization but also avoids the need to compute the derivative of the activation function, which is required in backpropagation.



(a) Before training **(b)** During training **(c)** After training

Figure 3.2: A simple scalar example of EBM training. During training the parameters W are adjusted such that the minimum corresponds to the target y.

3.2 Energy based models

An alternative perspective on neural network inference and training is based on the concept of energy-based models (EBMs). In an EBM a network is characterized by a scalar potential $E(s, W)$, which is a function of a state vector s and an adjacency matrix of learnable weights W. Unlike the previous section, we do not explicitly assume a layered architecture here (but layers can be introduced by enforcing a particular block sparsity in W). The minima of the energy with respect to the states are interpreted as *memories* or *predictions*. Figure 3.2 illustrates how an EBM is trained by reshaping the energy landscape of $E(s, W)$ by adjusting W in such a way that the minima correspond to desired memories/predictions. This type of model was pioneered by, among others [25]and [26] and builds on prior work on the Ising model. These models are typically referred to as associative memory or Hopfield models. The energy function optimized in [27] and [28] is

$$E^{\text{Hopfield}}(s, W) = G(s) - \tfrac{1}{2} s^\top W s, \qquad \text{where } \nabla_s G(s) = f^{-1}(s). \tag{3.8}$$

It is possible to train such models to perform predictions by injecting a teaching signal into a subset of units. Next, we will consider two different approaches to supervised training, namely variations of *contrastive Hebbian learning* (CHL) and *lifted neural networks*.

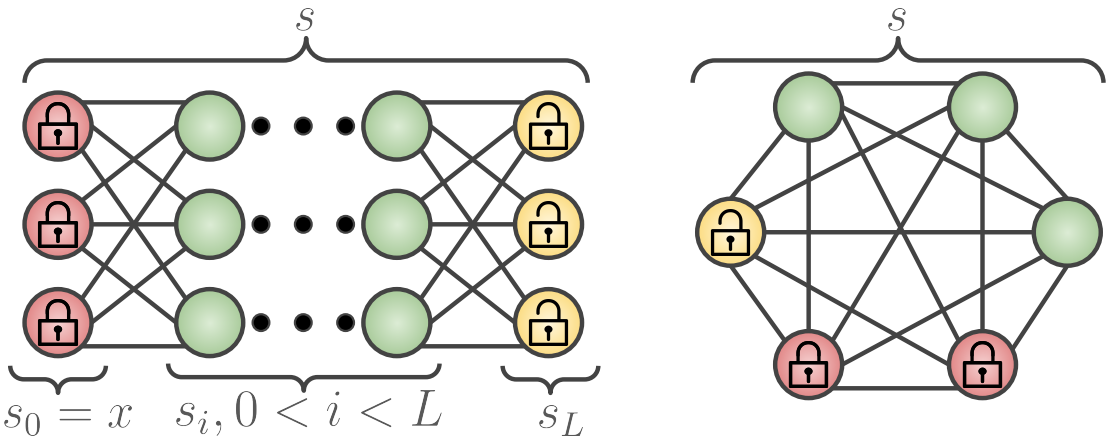


Figure 3.3: In contrastive Hebbian learning, states are inferred twice, once with output neurons (yellow) clamped to a target vector (the positive/clamped phase) and once without clamping (the negative/free phase). The input neurons (red) are clamped during both phases, and the remaining neurons (green) are minimizers of the network potential E. **Left:** A layered network. **Right:** A densely connected network.

Contrastive Hebbian learning

Contrastive Hebbian learning is a method of training neural networks by *contrasting* neural states subject to different perturbations [28]. Traditionally this is done by designating a subset of *input neurons* and a subset of *output neurons*. Figure 3.3 displays this setup for a layered and a densely connected network, with input units in red, output units in yellow, and hidden units in green. During the first phase of training, the positive or clamped phase, the input units are clamped to some input x (for example, an image), while the output units are clamped to a corresponding label y. The remaining neural activations are inferred by minimizing the energy E. During the subsequent negative/free phase, the output units are no longer clamped and all but the input units are inferred by minimizing the energy.

Following [28] contrastive Hebbian learning has been revisited in a number of works, mainly differing in terms of the boundary conditions imposed on the output neurons (the yellow neurons in figure 3.3), and the structure of the weight matrix W.

CHL with layerwise discounting

In [29], it was shown that CHL approximates backpropagation in feedforward models when applied to a recurrent model with weak feedback. This is achieved by adding layerwise discounting factors to the network potential. The use of weak feedback makes the teaching signal prone to vanishing in deeper networks due to the finite numerical precision of digital computers. In noisy (e.g., analog electronic) circuits, the weak feedback requirement is particularly problematic, as the teaching signal may drown out in noise. The model explored in paper A is an instance of contrastive Hebbian learning with layerwise discounting, but employs a different energy function than the model of [29].

Equilibrium propagation

Rather than clamping output neurons during the positive phase equilibrium propagation (EP) [30] nudges the output neurons by adding a term $\beta\ell(s, y)$ to the network potential. EP is typically applied to symmetric Hopfield models but has been applied to feedforward models [31], nonlinear resistive networks [22] and spiking neural networks [32]. In the limit of $\beta \to 0$, equilibrium propagation's weight updates correspond to gradient descent on $\ell(s(\theta), y)$. Followup works on equilibrium propagation has explored continual weight updates [33], reducing gradient estimator bias [34] and improving robustness to strong nudging through the use of complex-valued neurons [35].

Coupled learning

Coupled learning (CpL) [36] also employs weak feedback, but goes about it in a different manner. Whereas CHL [28] fixes output units to the target vector y and EP nudges them towards lower loss, CpL first infers the free states and then clamps the output units to a convex combination of the free solution and the target vector. CpL has been successfully implemented in small-scale physical circuits [12], but the resulting gradient estimate lacks the strong theoretical guarantees of EP [37].

Deriving CHL and EP using the optimal value reformulation

In the following, we will show a simple method for deriving equilibrium propagation and contrastive Hebbian learning based on a particular refor-

mulation of an inequality constraint, called *the optimal value reformulation* (OVR) [38]. The core idea is that $s = \arg\min'_s E(s')$ is equivalent to the constraint $E(s) \leq min_{s'} E(s')$ since the inequality constraint is only satisfied when s actually is the minimizer of E. Using the optimal value reformulation, contrastive Hebbian learning and equilibrium propagation can be derived in a very concise manner [39], without specifying the underlying energy $E(s, W)$, loss function $\ell(s, y)$ or connectivity W.

$$\min_W \ell(s, y) \quad \text{s.t. } s = \arg\min_{s'} E(s', W) \tag{$\mathcal{P}_3$}$$

Renaming the variable s as s^+ and using the optimal value reformulation, this can be rewritten in a more malleable form. The optimal value reformulation replaces the constraint $s = \arg\min_{s^0} E(s^0, W)$ by the equivalent constraint $E(s, W) \leq \min_{s^0} E(s^0, W)$.

$$\Leftrightarrow \min_W \ell(s^+, y) \quad \text{s.t. } E(s^+, W) \leq \min_{s^0} E(s^0, W). \tag{3.9}$$

We then proceed to enforce the constraint through a penalty term, weighted by a scaling factor $1/\beta$. The penalty term will always be non-negative since $E(s^+, W) \geq \min_{s^0} E(s^0, W)$ by construction.

$$\min_W \left(\min_{s^+} \left(\ell(s^+, y) + \tfrac{1}{\beta} E(s^+, W) \right) - \min_{s^0} \tfrac{1}{\beta} E(s^0, W) \right) \tag{3.10}$$

Multiplying by β and rewriting $-\min_{s^0} E(s^0, W)$ as $\max_{s^0} -E(s^0, W)$ yields

$$\min_W \min_{s^+} \max_{s^0} \left(\beta \ell(s^+, y) + E(s^+, W) - E(s^0.W) \right) \tag{3.11}$$

This formulation recovers Contrastive Hebbian learning in the limit $\beta \to \infty$ (the output neurons effectively become clamped) and recovers equilibrium propagation in the limit $\beta \to 0$ (the output units receive an infinitesimal nudge). Coupled learning, being more of a heuristic method, does not fit neatly into this formulation due to the way the loss is computed in terms of both s^0 and s^+ during the nudged phase: $\ell(s^+, \alpha y + (1 - \alpha) s^0)$, where $\alpha \in [0, 1]$.

Lifted neural networks

Lifted neural networks also seek to optimize an objective subject to constraints on the neural states. Lifted networks typically assume a layered architecture

and employ quadratic energies rather than the Hopfield energies typically employed in contrastive Hebbian learning. In the following, we return to the overloaded notation where subscript l denotes the layer index.

$$\min_W \ell(s_L(W,x),y), \quad \text{s.t. } s_l = \arg\min_{s_l} E_l(s_l, s_{l-1}, \theta_{l-1})) \quad \forall l \ \in \{1,2,\ldots,L\} \tag{$\mathcal{P}_4$}$$

Rather than dealing with this constrained optimization problem, an unconstrained optimization problem is constructed by enforcing the constraints via penalty terms.

$$\min_W \ \min_s \ell(s_L) + \tfrac{1}{\beta}\sum_{l=1}^{L} E_l(s_l, s_{l-1}, W_{l-1}) \tag{3.12}$$

This formulation of neural network training was first proposed by Carreira-Perpinan & Wang in [40], where it was referred to as the method of auxiliary coordinates (MAC). The inner problem can be solved using gradient descent, exact coordinate descent, or even off-the-shelf solvers. The potential employed in [40] is

$$E_l^{MAC}(s_l, s_{l-1}, W_{l-1}) = \frac{1}{2}||s_l - f_l(W_{l-1}s_{l-1})||_2^2 \tag{3.13}$$

Since then, a number of works, sometimes referred to as *lifted neural networks*, employing similar objectives have been proposed. *Alternating minimization* (AM) [41], differs in that the inner optimization is not carried out with respect to activations but rather with respect to preactivations a_l (defined through $s_l = f_l(a_l)$), by employing

$$E_l^{AM}(s_l, a_{l-1}, W_{l-1}) = \frac{1}{2}||a_l - W_{l-1}f_{l-1}(a_{l-1})||_2^2 \tag{3.14}$$

In a parallel line of research, motivated by biological plausibility, training predictive coding networks in a layered model also corresponds to solving problem 3.12 while employing the potential given by Eq 3.14. This connection has previously been highlighted in [39].

The appearance of the activation function in equation 3.13 and equation 3.14 has the disadvantage that inference dynamics will involve the derivative of the activation function, which, depending on the inference method can lead to slow convergence, while also being problematic from the perspective of analog

amenability and biological plausibility. An alternative approach is to express the energy purely in terms of activity s (as opposed to in terms of both s and a) and add non-negativity constraints. This formulation is based on an interpretation of the ReLU nonlinearity as a proximal operator [42].

$$E_l^{prox}(s_l, s_{l-1}, W_{l-1}) = \frac{1}{2}||s_l - W_{l-1}s_{l-1}||_2^2 + \mathbb{I}(s_l)_{\geq 0} \tag{3.15}$$

Here $\mathbb{I}(s_l)_{\geq 0}$ is an indicator function, which has value ∞ if any element of s_l is negative and zero otherwise. This imposes a hard projection to the non-negative real numbers $\mathbb{R}_0^{d_l}$ when updating the states, resulting in ReLU-type neurons. A restriction of this formulation is that it limits the networks to ReLU units (or with a slight modification to hard sigmoid units). Equation 3.15 is the energy function used in paper A.

Lifted learning as CHL The lifted formulation can be seen as a special case of contrastive Hebbian learning in a layered feedforward network with quadratic layerwise energies E_l [31]. In this setting, the energy of the free phase is always exactly zero, making equation 3.11 equivalent to equation 3.12.

CHAPTER 4

Dyadic learning

Contrastive Hebbian learning (including the variant explored in paper A) relies on two sequential inference phases, layerwise discounting and requires multiple neural state updates in order for inference to converge. Motivated by these limitations, the paper C and D proposes a local learning algorithm, dual propagation, for training feedforward models based on dyadic neurons (neurons with two internal states). Dual propagation propagates errors and activities simultaneously (hence the name) and inference converges rapidly. However, the underlying models are qualitatively different (CHL is for symmetric Hopfield models and DP is for feedforward models).

Paper E provides a general framework that bridges the gap between these algorithms. Starting from neuron-level energy-based constraints we derive a contrastive learning objective, which does assume a particular connectivity (beyond a lack of self-connections). When we specialize to Hopfield models and restrict the connectivity, this dyadic learning framework recovers as special cases equilibrium propagation in symmetric Hopfield models, dual propagation in feedforward models, and a class of inherently robust Lipschitz-1 models when the underlying connectivity is chosen to be a skew-symmetric.

4.1 Derivation via the optimal value reformulation

The following is a condensed version of the derivation of the dyadic learning objective found in the appendix of paper E. Our starting point is the following constrained optimization problem.

$$\min_W \ell(s, y) \quad \text{s.t. } \forall \text{ k } s_k = \arg\min_{s'_k} E_k(s'_k, s_{\setminus k}, W_{k,\setminus k}) \tag{$\mathcal{P}_5$}$$

This problem states that we wish to optimize some training loss $\ell(s, y)$ subject to the constraint that each individual neuron s_k is the minimizer of an energy E_k, which we assume has a unique minimizer. E_k takes as arguments s_k, all the other neurons $s_{\setminus k}$ (s with the k-th element removed) and the synaptic connections between s_k and $s_{\setminus k}$ denoted by $W_{k,\setminus k}$ (The k-th row of W with the k-th element removed). For brevity, the weight argument is omitted in the following. Next we introduce new variables s^+ and s^- and define $\bar{s} := \frac{1}{2}(s^+ + s^-)$ and rewrite the problem as

$$\begin{aligned} &\min_W \min_{s^+} \max_{s^-} \tfrac{1}{2}\ell(s^+, y) + \tfrac{1}{2}\ell(s^-, y) \\ &\text{s.t. } \forall \text{ k} \begin{cases} s_k^+ &= \arg\min_{s'_k} E_k(s'_k, \bar{s}_{\setminus k}) \\ s_k^- &= \arg\min_{s'_k} E_k(s'_k, \bar{s}_{\setminus k}) \end{cases} \end{aligned} \tag{4.1}$$

This reparametrization might look a little strange, but since E_k is has a unique minimizer and we are enforcing that s_k^+ and s_k^- are minimizers of E_k, then problem 4.1 remains equivalent to problem $\mathcal{P}_5$. The benefit of this reparametrization is that we can now apply the optimal value reformulation (twice), which makes it possible to construct penalizers.

$$\begin{aligned} &\min_W \min_{s^+} \max_{s^-} \tfrac{1}{2}\ell(s^+, y) + \tfrac{1}{2}\ell(s^-, y) \\ &\text{s.t. } \forall \text{ k} \begin{cases} E_k(s_k^+, \bar{s}_{/k}) \leq \min_{s'_k} E_k(s'_k, \bar{s}_{/k}) \\ E_k(s_k^-, \bar{s}_{/k}) \leq \min_{s'_k} E_k(s'_k, \bar{s}_{/k}) \end{cases} \end{aligned} \tag{4.2}$$

In much the same way as when going from Eq $\mathcal{P}_3$ to Eq 3.11 in the previous chapter, we now turn the constrained optimization problem into an unconstrained one by approximately enforcing the constraints via penalty terms weighted by $1/\beta > 0$. As usual, the constraints are strictly satisfied in the limit $\beta \to 0$. Since we are minimizing with respect to s^+ and maximizing with

regard to s^-, the penalty terms have opposite signs, which means the terms involving s'_k cancel out. This leads us to the dyadic learning objective.

$$\min_W \min_{s^+} \max_{s^-} \tfrac{1}{2}\ell(s^+, y) + \tfrac{1}{2}\ell(s^-, y) + \tfrac{1}{\beta}\sum_k \left(E_k(s_k^+, \bar{s}_{/k}) - E_k(s_k^-, \bar{s}_{/k})\right) \quad (4.3)$$

This objective is a generalization of the dual propagation objective from paper C and D. However, unlike the dual propagation objective, this objective does not assume a particular energy function or connectivity.

4.2 Specializing to a Hopfield-like energy

We consider a particular kind of Hopfield-like energy given by

$$E_k(s_k, s_{\setminus k}) = G(s_k) - s_k W_{k\setminus k} s_{\setminus k} - s_k \theta_{0,k} x \quad (4.4)$$

The first term G as usual determines the activation function through $\nabla_s G(s) = f^{-1}(s)$. The second term denotes interactions between s_k and the other neurons, and the last term denotes interactions between s_k and a set of static inputs x. This objective resembles a classical continuous Hopfield model but has a notable difference. At equilibrium, this energy satisfies

$$s = f(Ws + \theta_0 x), \quad (4.5)$$

whereas the traditional Hopfield energy ($E^{\text{Hopfield}}(x, s, W, \theta_0) = G(s) - \frac{1}{2}s^\top W s - s^\top \theta_0 x$), satisfies

$$s = f(\tfrac{1}{2}(W + W^\top)s + \theta_0 x). \quad (4.6)$$

The appearance of $\frac{1}{2}(W+W^\top)$ means that CHL and EP only use the symmetric part of the weight matrix[1], whereas this dyadic objective allows us to train models with symmetric, feedforward, skew-symmetric, and general asymmetric connectivity (as long as $W_{ii} = 0$). This is a benefit as symmetric connectivity introduces the weight transport problem and can be difficult to realize in physical hardware[2]. Of course, if $W = W^\top$, then we get the same behavior as

[1] Since we know that: $W = \underbrace{\tfrac{1}{2}(W + W^\top)}_{\text{symmetric}} + \underbrace{\tfrac{1}{2}(W - W^\top)}_{\text{skew}-\text{symmetric}}$

[2] The exception being the resistive networks mentioned in Chapter 2.4.

CHL and EP. With this choice of energy equation 4.3 becomes

$$\begin{aligned}\min_{\{W,\theta_0\}} \min_{s^+} \max_{s^-} &\tfrac{1}{2}\ell(s^+, y) + \tfrac{1}{2}\ell(s^-, y) \\ &+ \tfrac{1}{\beta}\left(G(s^+) - G(s^-) - (s^+ - s^-)^\top (W\bar{s} + \theta_0 x)\right).\end{aligned} \tag{4.7}$$

Introducing a concatenated total state-vector $\begin{pmatrix} s^+ \\ s^- \end{pmatrix}$ Eq. 4.3 can be rewritten as.

$$\begin{aligned}&\min_{W,\theta_0} \min_{s^+} \max_{s^-} \tfrac{1}{2}\ell(s^+, y) + \tfrac{1}{2}\ell(s^-, y) \\ &+ \tfrac{1}{\beta}\Bigg(G(s^+) - G(s^-) - (s^+ - s^-)^\top \theta_0 x \\ &- \tfrac{1}{4}\begin{pmatrix} s^+ \\ s^- \end{pmatrix}^\top \underbrace{\begin{pmatrix} W + W^\top & W - W^\top \\ -W + W^\top & -W - W^\top \end{pmatrix}}_{\mathcal{W}(W)} \begin{pmatrix} s^+ \\ s^- \end{pmatrix} \Bigg).\end{aligned} \tag{4.8}$$

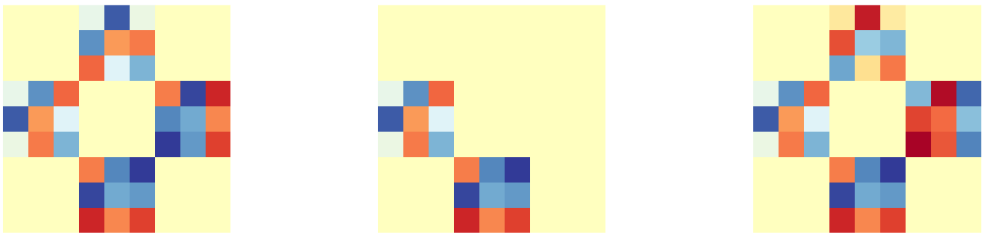
It is interesting that regardless of the structure of W, the parametrized contrastive weight matrix $\mathcal{W}(W)$ is always symmetric. However, the underlying structure of W does have a profound impact on both the training and inference dynamics of the resulting model.

Structured weights

In the absence of a teaching signal (or equivalently in the limit $\beta \to 0$) we get $s^+ = s^-$, which means terms involving $W^\top$ cancel out, and the resulting dynamics satisfy Eq 4.5 at equilibrium. The exact structure of W determines whether the model has layers and what kind of feedback it employs (if any). In the following we will briefly discuss the most interesting cases, which are skew-symmetric, feedforward and symmetric connectivity. Each of these cases are illustrated in Fig 4.1. Layers are imposed via block-sparsity.

Symmetric weights

In the case of symmetric W (illustrated in Fig. 4.1a), then $\frac{1}{2}(W + W^\top) = W$ and $\frac{1}{2}(W - W^\top) = 0$. This means the blocks governing interactions between s^+ and s^- vanish from the objective, making it possible to carry out the minimization with respect to s^+ and the maximization with respect to s^-



(a) Symmetric **(b)** Feedforward **(c)** Skew-symmetric

Figure 4.1: Examples of differently structured adjacency matrices of a simple three-layer model with three neurons per layer. Entries below the main diagonal correspond to bottom-up/feedforward connections (layer 1 to layer 2 and layer 2 to layer 3. Entries above the main diagonal correspond to top-down/feedback connections (layer 2 to layer 1 and layer 3 to layer 2).

separately. This objective and the resulting dynamics correspond to equilibrium propagation in a continuous Hopfield model.

Lower triangular weights

If W is lower triangular (illustrated in Fig. 4.1b), then the dynamics correspond exactly to dual propagation applied to a feedforward model. In this model, the mean of neurons' two internal states (s^+ and s^-) is propagated forward through W, and the difference is propagated backward through $W^\top$. You can think of the feedforward model as a special kind of Hopfield model with lower triangular connectivity.

Skew-symmetric weights

If W is skew-symmetric (illustrated in Fig. 4.1c), then $\frac{1}{2}(W + W^\top) = 0$ and $\frac{1}{2}(W - W^\top) = W$. This means that we end up with a bipartite graph where elements of s^+ only interact directly with elements of s^- and vice versa. In general, this means that one has to infer both the s^+ and s^- states simultaneously. However, in the case of a layered network (achieved by introducing a particular block sparsity into W), then the training dynamics decouple into one min-max problem over odd layers of s^+ and even layers of s^- and another min-max problem over even layers of s^+ and odd layers of s^-. This allows training to be carried out in two sequential phases using

the same physical hardware. This is similar to equilibrium propagation but differs as each phase is a min-max problem rather than a pure minimization problem. As shown in paper E this model has the remarkable property that the dynamics of the hidden layers are Lipschitz-1, giving it a degree of inherent robustness to perturbations.

A good intuition for this model's inherent stability is to think of each pair of neurons as a negative feedback system[3]. Intuitively, if neuron A is exciting neuron B, then neuron B will inhibit neuron A, effectively tempering the degree to which A excites B.

Interpolating between the three structured cases

It is possible to smoothly interpolate between all three of the structured cases by parametrizing W in terms of a lower triangular matrix θ and a scalar parameter $\lambda \in [0, 1]$. Defining $W_\lambda := \theta - \theta^\top + 2\lambda\theta^\top$. If $\lambda = 0$, then W_λ will be skew-symmetric, if $\lambda = 1$ then W_λ will be symmetric, and if $\lambda = 1/2$ then W_λ will be lower triangular, giving rise to a feedforward model. Figure 4.1 shows the three extreme cases (fully skew-symmetric, lower-triangular (feedforward), and symmetric). The use of this interpolation property is discussed in the future works section of chaper 6.

Arbitrarily connected weights

In the most general case, we place no restrictions on W (apart from not having self-connections). In this case, the dyadic neurons will propagate the differences in their internal states through the network via the transposed weight matrix $W^\top$ and the mean through W. This is somewhat similar to the case of lower triangular W discussed above. However, in that setting, errors/differences only flow down through the layers, and activities/means only flow up, whereas in the unstructured setting, errors and activities flow in both directions.

4.3 Chapter summary

In this chapter, we have shown how to derive a training objective by reparametrizing and relaxing the initial neural network training problem. By altering the network's connectivity, the dyadic learning framework encompasses equilibrium

[3] In control systems negative feedback is often used to improve the stability of a system.

propagation and dual propagation-based training as special cases in Hopfield and feedforward models respectively. Furthermore, the dyadic learning framework opens the door to training asymmetric (including skew-symmetric) models. There are a number of ways to build upon this, including exploring robust skew-symmetric models for noisy analog hardware, more biologically plausible asymmetric synaptic connectivity, and new choices of energy functions. We will discuss some of these directions in chapter 6.

CHAPTER 5

Summary of included papers

This chapter provides a summary of the included papers.

5.1 Paper A

R. Høier and C. Zach
Lifted Regression/Reconstruction Networks
Proceedings of the British Machine Vision Conference 2020.

We propose a new neural network model, in which each pair of adjacent layers behave like an auto-encoder, simultaneously performing regression and reconstruction. The model employs quadratic penalty terms as network energy similar to many lifted networks and predictive coding networks, but learning is implemented running inference twice under different boundary conditions as in contrastive Hebbian learning. When constraining the reconstruction weights to be identical to the regression weights the network has Lipschitz-1 hidden layers by construction. This is a useful property with regards to robustness to perturbations (e.g. noise or adversarial perturbations).

5.2 Paper B

H. Le, **R. Høier**, C.T. Lin and C. Zach
AdaSTE: An Adaptive Straight-Through Estimator to Train Binary Neural Networks
The IEEE/CVF Conference on Computer Vision and Pattern Recognition 2022.

Employing binary ({-1,1}) weights in neural networks, allows replacing costly Floating point multiplication with addition/subtraction. This makes the training of quantized neural networks an attractive near term avenue for addressing the high energy costs of running AI systems on existing digital hardware. However, this requires modifying the learning rules as quantizing functions (such as the sign function) have zero valued derivatives almost everywhere. Here we consider binary neural network training through the lense of constrained optimization and optimize the networks loss function subject to the constraint that the weights must be binary. By relaxing this problem appropriately we arrive at a gradient estimator that conditionally ignores the derivative of the activation function. The resulting gradient estimator is based on directional finite differences with an adaptive choice of spacing. This allows informative gradients to flow through the sign function, and helps the network to retain plasticity.

5.3 Paper C

R. Høier, D. Staudt and C. Zach
Dual Propagation: Accelerating Contrastive Hebbian Learning with Dyadic Neurons
International Conference on Machine Learning 2023.

Inspired by lifted neural networks and compartmental neuron models we propose a variant of contrastive Hebbian learning in which each neuron is a

dyad with two internal states. Having two states permits neurons to simultaneously propagate activity and error signals and facilitates approximating the gradients computed by backpropagation. The resulting single-phased algorithm essentially braids the two phases of traditional contrastive Hebbian learning into a single phase, which radically reduces the number of iterations required during simulations. A variety of different inference schedules are shown to work well on MNIST and FashionMNIST, including one where the order in which neuron states are updated is chosen at random. Using the most efficient inference schedule the algorithm is benchmarked on convolutional networks and the CIFAR10, CIFAR100 and ImageNet32x32 datasets, where it is shown to outperform state-of-the-art biologically inspired learning algorithms.

5.4 Paper D

R. Høier and C. Zach
Two Tales of Single-Phase Contrastive Hebbian Learning
International Conference on Machine Learning 2024.

Optimizing the dual propagation objective has been shown to approximate backpropagation in the previous paper, but a rigorous derivation of the objective was missing. Here we show how to derive the dual propagation objective, using relaxations of the optimal value reformulation, and how to derive a new variation of dual propagation, using a reparametrization of the Lagrangian associated with the training problem ($\mathcal{P}_2$). Whereas the original dual propagation required a certain hyper parameter governing the feedback nudging signal arriving at each layer to be $\alpha = 1/2$ the new variant avoids such a requirement. We show experimentally that asymmetric nudging ($\alpha = 0$) permits using a stronger teaching signal. This may be critical when training on noisy compute substrates where a weak teaching signal will drown out in noise.

5.5 Paper E

R. Høier, K. Kalinin, M. Ernoult and C. Zach
Dyadic Learning In Recurrent and Feedforward Models
Machine Learning with New Compute Paradigms.

We propose a framework for training arbitrarily connected Hopfield models. The paper focuses on a specific parametrization of the connectivity, which permits smoothly interpolating between symmetric Hopfield models, feedforward models and skew-symmetric Hopfield models. The framework encompasses as special cases the known training algorithms equilibrium propagation (on Hopfield models) and dual propagation (on feed-forward models). The skew-symmetric setting is particularly interesting as it is Lipschitz-1 (and hence robust to perturbations) by construction. The skew-symmetric model is further motivated by drawing parallels to negative feedback loops, which are widely used for their stabilizing effects.

CHAPTER 6

Concluding Remarks and Future Work

6.1 Conclusion

There is an interesting dependence between research on neural network learning algorithms and the hardware available for deploying and training neural networks. Which algorithmic innovations seem worthwhile to pursue is highly dependent on the assumptions one makes about future hardware and vice versa.

The main underlying motivation behind the research presented here is that there will be a space for analog and/or optical hardware accelerators in the AI compute landscape of the future. This will necessitate learning algorithms that do not rely on exact knowledge of activation functions and their derivatives, and which require little to no inter-neuron synchronization. A secondary but persistent motivation throughout this research project has been to develop less implausible (from a biological point of view) learning algorithms.

In papers A, C, D and E, we have developed learning algorithms which have these characteristics. In a sense, the research project comes full circle with the last paper (E, as the dyadic learning framework generalizes the algorithm developed in C and D to arbitrarily connected architectures, one of which has

a similar Lipschitz-1 property to that of the LRRN model of paper A, while being significantly easier to simulate.

Paper B is a bit of an outlier compared to the other four papers as it focuses on reducing the compute footprint of existing digital hardware by training neural networks with binary weights. However, in terms of methods, the general approach of reformulating and relaxing a constrained optimization problem is shared with the other papers.

6.2 Future work

General asymmetric Hopfield networks

In feedforward networks, the weight transport problem has been addressed by variations of feedback alignment [14]. However, feedforward models are unrealistic as models of biological learning as they are entirely driven by bottom-up connections during the forward pass and entirely driven by top-down connections during the backward pass. In contrast biological circuits do not exhibit such a strict separation between feedforward and feedback processing. Indeed, both feedforward and feedback processing are believed to play important roles in vision and overlap temporally [43].

Contrastive Hebbian learning and equilibrium propagation do employ feedback connections, but also impose strictly symmetric connectivity, which is also biologically implausible. In contrast, in the general formulation of dyadic learning (when the weights are not parametrized) inference is both recurrent and avoids weight transport. However, the weight transport problem does appear at training time, as means of activities are transported through the weights, and differences are transported through the transpose of the weights.

A step towards a more biologically plausible learning algorithm would be to take inspiration from feedback alignment and replace the weight transpose in the dynamics of dyadic learning with a distinct weight matrix. An appealing aspect of this approach is that unlike efforts to make CHL and EP [16], [44] avoid weight transport, this modification only affects training dynamics (leaving inference dynamics completely unchanged).

Skew-symmetric resistive networks

The skew-symmetric Hopfield model's inherent robustness to perturbations makes it an interesting algorithm for training neural networks in noisy analog hardware. Dyadic learning using a skew-symmetric Hopfield model should permit a strong gain on the teaching signal, which is necessary to prevent the teaching signal from drowning out in noise. As mentioned in section 2.4, resistive networks constitute a subset of continuous Hopfield networks, so it is natural to wonder if one could adapt the skew-symmetric model to the setting of resistive networks where circuit noise is a real challenge. An electrical component called a gyrator may be useful for modeling such skew-symmetric connectivity in analog hardware (J. Kendall, personal communication, November 2024). It is likely that exact skew-symmetry will be hard to enforce in physical hardware due to device non-idealities. Consequently, exploring the impact of non-idealities on the robustness properties is a natural auxiliary research question.

Homotopy methods for faster training of (skew-)symmetric Hopfield models

Simulating the training of symmetric and skew-symmetric Hopfield models is time-consuming as inference requires tens or hundreds of iterations to converge (and must be repeated twice). Furthermore, initialization is crucial, and there is less knowledge regarding how best to initialize these models than there is for feedforward models.

The parametrized version of dyadic learning permits gradually transforming a feedforward model ($\lambda = 1/2$) into a symmetric ($\lambda = 1$) or skew-symmetric Hopfield model ($\lambda = 0$). This would allow us to use established best practices for feedforward network initialization and training to decrease the training time of Hopfield models by gradually adjusting λ. Alternatively, one could take off-the-shelf feedforward models and gradually turn them into (skew-) symmetric Hopfield models through fine-tuning.

References

[1] A. Karpathy, "Software 2.0," *Medium*, 2017, Accessed: February 1rst 2025.

[2] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[4] D. E. Rumelhart, G. E. Hinton, J. L. McClelland, *et al.*, "A general framework for parallel distributed processing," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, no. 45-76, p. 26, 1986.

[5] K.-S. Oh and K. Jung, "Gpu implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311–1314, 2004.

[6] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*, IEEE, 2012, pp. 3642–3649.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[8] S. Luccioni, Y. Jernite, and E. Strubell, "Power hungry processing: Watts driving the cost of ai deployment?" In *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, 2024, pp. 85–99.

[9] F. Crick, "The recent excitement about neural networks," *Nature*, vol. 337, pp. 129–132, 1989.

[10] J. Backus, "Can programming be liberated from the von neumann style? a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[11] C. Bredenberg, E. Williams, C. Savin, B. Richards, and G. Lajoie, "Formalizing locality for normative synaptic plasticity models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[12] S. Dillavou, M. Stern, A. J. Liu, and D. J. Durian, "Demonstration of decentralized physics-driven learning," *Physical Review Applied*, vol. 18, no. 1, p. 014 040, 2022.

[13] S. Dillavou, B. D. Beyer, M. Stern, M. Z. Miskin, A. J. Liu, and D. J. Durian, "Machine learning without a processor: Emergent learning in a nonlinear electronic metamaterial," *arXiv preprint arXiv:2311.00537*, 2023.

[14] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, p. 13 276, 2016.

[15] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 1037–1045.

[16] G. Detorakis, T. Bartley, and E. Neftci, "Contrastive hebbian learning with random feedback weights," *Neural Networks*, vol. 114, pp. 1–14, 2019.

[17] M. Refinetti, S. D'Ascoli, R. Ohana, and S. Goldt, "Align, then memorise: The dynamics of learning with feedback alignment," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8925–8935.

[18] J. Laydevant, L. G. Wright, T. Wang, and P. L. McMahon, "The hardware is the software," *Neuron*, vol. 112, no. 2, pp. 180–183, 2024.

[19] K. P. Kalinin, G. Mourgias-Alexandris, H. Ballani, *et al.*, "Analog iterative machine (aim): Using light to solve quadratic optimization problems with mixed variables," *arXiv preprint arXiv:2304.12594*, 2023.

[20] W. JOHNSON, *Nonlinear electrical networks*, 2010.

[21] J. Kendall, R. Pantone, K. Manickavasagam, Y. Bengio, and B. Scellier, "Training end-to-end analog neural networks with equilibrium propagation," *arXiv preprint arXiv:2006.01981*, 2020.

[22] B. Scellier, "A fast algorithm to simulate nonlinear resistive networks," *arXiv preprint arXiv:2402.11674*, 2024.

[23] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[24] Y. Lecun, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, Morgan Kaufmann, 1988, pp. 21–28.

[25] S.-I. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Transactions on computers*, vol. 100, no. 11, pp. 1197–1206, 1972.

[26] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[27] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons.," *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[28] J. R. Movellan, "Contrastive hebbian learning in the continuous hopfield model," in *Connectionist Models*, Elsevier, 1991, pp. 10–17.

[29] X. Xie and H. S. Seung, "Equivalence of backpropagation and contrastive hebbian learning in a layered network," *Neural computation*, vol. 15, no. 2, pp. 441–454, 2003.

[30] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[31] B. Millidge, Y. Song, T. Salvatori, T. Lukasiewicz, and R. Bogacz, "Backpropagation at the infinitesimal inference limit of energy-based models: Unifying predictive coding, equilibrium propagation, and contrastive hebbian learning," *arXiv preprint arXiv:2206.02629*, 2022.

[32] E. Martin, M. Ernoult, J. Laydevant, *et al.*, "Eqspike: Spike-driven equilibrium propagation for neuromorphic implementations," *Iscience*, vol. 24, no. 3, p. 102 222, 2021.

[33] M. Ernoult, J. Grollier, D. Querlioz, Y. Bengio, and B. Scellier, "Equilibrium propagation with continual weight updates," *arXiv preprint arXiv:2005.04168*, 2020.

[34] A. Laborieux, M. Ernoult, B. Scellier, Y. Bengio, J. Grollier, and D. Querlioz, "Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias," *Frontiers in neuroscience*, vol. 15, p. 129, 2021.

[35] A. Laborieux and F. Zenke, "Holomorphic equilibrium propagation computes exact gradients through finite size oscillations," *Advances in Neural Information Processing Systems*, vol. 35, pp. 12 950–12 963, 2022.

[36] M. Stern, D. Hexner, J. W. Rocks, and A. J. Liu, "Supervised learning in physical networks: From machine learning to learning machines," *Physical Review X*, vol. 11, no. 2, p. 021 045, 2021.

[37] B. Scellier, M. Ernoult, J. Kendall, and S. Kumar, "Energy-based learning algorithms for analog computing: A comparative study," *arXiv preprint arXiv:2312.15103*, 2023.

[38] J. V. Outrata, "A note on the usage of nondifferentiable exact penalties in some special optimization problems," *Kybernetika*, vol. 24, no. 4, pp. 251–258, 1988.

[39] C. Zach, "Bilevel programs meet deep learning: A unifying view on inference learning methods," *arXiv preprint arXiv:2105.07231*, 2021.

[40] M. Carreira-Perpinan and W. Wang, "Distributed optimization of deeply nested systems," in *Artificial Intelligence and Statistics*, 2014, pp. 10–19.

[41] A. Choromanska, B. Cowen, S. Kumaravel, *et al.*, "Beyond backprop: Online alternating minimization with auxiliary variables," in *International Conference on Machine Learning*, PMLR, 2019, pp. 1193–1202.

[42] Z. Zhang and M. Brand, "Convergent block coordinate descent for training tikhonov regularized deep neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 1721–1730.

[43] D. Wyatte, D. J. Jilk, and R. C. O'Reilly, "Early recurrent feedback facilitates visual object recognition under challenging conditions," *Frontiers in psychology*, vol. 5, p. 674, 2014.

[44] B. Scellier, A. Goyal, J. Binas, T. Mesnard, and Y. Bengio, "Generalization of equilibrium propagation to vector field dynamics," *arXiv preprint arXiv:1808.04873*, 2018.