Deep Learning Models of Learning in the Brain

Roman Pogodin

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Gatsby Computational Neuroscience Unit

University College London

January 25, 2023

I, Roman Pogodin, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

This thesis considers deep learning theories of brain function, and in particular biologically plausible deep learning. The idea is to treat a standard deep network as a high-level model of a neural circuit (e.g., the visual stream), adding biological constraints to some clearly artificial features. Two big questions are possible. First, how to train deep networks in a biologically realistic manner? The standard approach, supervised training via backpropagation, needs overly complicated machinery for backpropagation and precise labels (that are somewhat scarce in the real world). The first result in this thesis approaches the first problem, backpropagation, by avoiding it completely. A layer-wise objective is proposed, which results in local, Hebbian weight updates that use a global error signal. The second result approaches the need for precise labels. It is focused on a principled approach to self-supervised learning, framing the problem as dependence maximisation using kernel methods. Although this is a deep learning study, it is relevant to neuroscience: self-supervised learning appears to be a suitable learning paradigm for the brain as it only requires binary (same source or not) teaching signals for pairs of inputs. Second, how realistic is the architecture itself? For instance, most well-performing networks have some form of weight sharing – having the same weights for different neurons at all times. Convolutional networks share filter weights among neurons, and transformers do so for matrix-matrix products. While the operation is biologically implausible, the third result of this thesis shows that it can be successfully approximated with a separate phase of weight-sharing-inducing Hebbian learning.

Acknowledgements

I would like to thank my supervisor Peter for the guidance and support, but most importantly for being a great example of how to stay positive yet critical about science. I also want to thank my partner Valeriya for the much needed emotional support and life advice throughout my PhD. I am also grateful to my family and friends from back home who felt as close to me as ever despite the long distance and different life. Concluding the list of people, I want to thank everyone at Gatsby and SWC who has been around – you have made our building a very friendly and welcoming community. Finally, I want to thank the Gatsby Charitable Foundation and the Wellcome Trust for supporting my work and for the wonderful research environment at the Gatsby Unit.

Impact

The results presented in this thesis strengthen the connection between deep learning and theoretical neuroscience by developing deep learning-inspired learning theories for the brain. Understanding what learning rules guide the brain is one of the fundamental goals in neuroscience. In the short term, advances in this area can facilitate development of brain-computer interfaces and machine learning methods and theory. In the long term, theoretical and experimental evidence for learning mechanisms in the brain can greatly impact treatment for any condition associated with brain's development and general function, such as learning disabilities. The presented results also expand our understanding of purely deep learning methods from the perspective of statistical dependence, which can lead to more interpretable deep learning methods.

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

Please use this form to declare if parts of your thesis are already available in another format, e.g. if data, text, or figures:

- have been uploaded to a preprint server;
- are in submission to a peer-reviewed publication;
- have been published in a peer-reviewed publication, e.g. journal, textbook.

This form should be completed as many times as necessary. For instance, if you have seven thesis chapters, two of which containing material that has already been published, you would complete this form twice.

1. For a research manuscript that has already been published (if not yet published, please skip to section 2)**:**		
a) **Where was the work published?** (e.g. journal name)	Advances in Neural Information Processing Systems 33 (NeurIPS 2020)	
b) Who published the work? (e.g. Elsevier/Oxford University Press):	Advances in Neural Information Processing Systems 33 (NeurIPS 2020)	
c) **When was the work published?**	2020	
d) Was the work subject to academic peer review?	Yes	
e) Have you retained the copyright for the work?	Yes	
[If no, please seek permission from the relevant publisher and check the box next to the below statement]: ☐ *I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.*		
2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3)**:**		
a) Has the manuscript been uploaded to a preprint server? (e.g. medRxiv):		**If yes, which server?**
b) **Where is the work intended to be published?** (e.g. names of journals that you are planning to submit to)		
c) List the manuscript's authors in the intended authorship order:		
d) **Stage of publication**		

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4**):**			
R. P. obtained the results. R.P. and P.E.L. wrote the paper.			
4. In which chapter(s) of your thesis can this material be found?			
Chapter 2			
5. e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work):			
Candidate:	Roman Pogodin	**Date:**	26.08.22
Supervisor/ Senior Author (where appropriate)**:**	Peter E. Latham	**Date:**	26.08.22

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

Please use this form to declare if parts of your thesis are already available in another format, e.g. if data, text, or figures:

- have been uploaded to a preprint server;
- are in submission to a peer-reviewed publication;
- have been published in a peer-reviewed publication, e.g. journal, textbook.

This form should be completed as many times as necessary. For instance, if you have seven thesis chapters, two of which containing material that has already been published, you would complete this form twice.

1. For a research manuscript that has already been published (if not yet published, please skip to section 2)**:**		
a) **Where was the work published?** (e.g. journal name)	Advances in Neural Information Processing Systems 34 (NeurIPS 2021)	
b) Who published the work? (e.g. Elsevier/Oxford University Press):	Advances in Neural Information Processing Systems 34 (NeurIPS 2021)	
c) **When was the work published?**	2021	
d) Was the work subject to academic peer review?	Yes	
e) Have you retained the copyright for the work?	Yes	
[If no, please seek permission from the relevant publisher and check the box next to the below statement]: ☐ *I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.*		
2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3)**:**		
a) Has the manuscript been uploaded to a preprint server? (e.g. medRxiv):		**If yes, which server?**
b) **Where is the work intended to be published?** (e.g. names of journals that you are planning to submit to)		
c) List the manuscript's authors in the intended authorship order:		
d) **Stage of publication**		

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4**):**
Y.L. obtained the experimental results. Y.L. and R. P. obtained the theoretical results. Y.L., R.P., D.J.S. and A.G. wrote the paper.
4. In which chapter(s) of your thesis can this material be found?
Chapter 3
5. e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work):

Candidate:	Roman Pogodin	**Date:**	26.08.22
Supervisor/ Senior Author (where appropriate)**:**	Arthur Gretton	**Date:**	26.08.22

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

Please use this form to declare if parts of your thesis are already available in another format, e.g. if data, text, or figures:

- have been uploaded to a preprint server;
- are in submission to a peer-reviewed publication;
- have been published in a peer-reviewed publication, e.g. journal, textbook.

This form should be completed as many times as necessary. For instance, if you have seven thesis chapters, two of which containing material that has already been published, you would complete this form twice.

1. For a research manuscript that has already been published (if not yet published, please skip to section 2)**:**		
a) **Where was the work published?** (e.g. journal name)	Advances in Neural Information Processing Systems 34 (NeurIPS 2021)	
b) Who published the work? (e.g. Elsevier/Oxford University Press):	Advances in Neural Information Processing Systems 34 (NeurIPS 2021)	
c) **When was the work published?**	2021	
d) Was the work subject to academic peer review?	Yes	
e) Have you retained the copyright for the work?	Yes	
[If no, please seek permission from the relevant publisher and check the box next to the below statement]: ☐ *I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.*		
2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3)**:**		
a) Has the manuscript been uploaded to a preprint server? (e.g. medRxiv):		**If yes, which server?**
b) **Where is the work intended to be published?** (e.g. names of journals that you are planning to submit to)		
c) List the manuscript's authors in the intended authorship order:		
d) **Stage of publication**		

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4**):**			
R. P. obtained the results. R.P., Y.M., T.P.L. and P.E.L. wrote the paper.			
4. In which chapter(s) of your thesis can this material be found?			
Chapter 4			
5. e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work):			
Candidate:	Roman Pogodin	**Date:**	26.08.22
Supervisor/ Senior Author (where appropriate)**:**	Peter E. Latham	**Date:**	26.08.22

UCL Research Paper Declaration Form: referencing the doctoral candidate's own published work(s)

Please use this form to declare if parts of your thesis are already available in another format, e.g. if data, text, or figures:

- have been uploaded to a preprint server;
- are in submission to a peer-reviewed publication;
- have been published in a peer-reviewed publication, e.g. journal, textbook.

This form should be completed as many times as necessary. For instance, if you have seven thesis chapters, two of which containing material that has already been published, you would complete this form twice.

1. For a research manuscript that has already been published (if not yet published, please skip to section 2)**:**		
a) **Where was the work published?** (e.g. journal name)	1st Brain-Score Workshop (BSW 2022)	
b) Who published the work? (e.g. Elsevier/Oxford University Press):	openreview.net	
c) **When was the work published?**	2022	
d) Was the work subject to academic peer review?	Yes	
e) Have you retained the copyright for the work?	Yes	
[If no, please seek permission from the relevant publisher and check the box next to the below statement]: ☐ *I acknowledge permission of the publisher named under 1b to include in this thesis portions of the publication named as included in 1a.*		
2. For a research manuscript prepared for publication but that has not yet been published (if already published, please skip to section 3)**:**		
a) Has the manuscript been uploaded to a preprint server? (e.g. medRxiv):		**If yes, which server?**
b) **Where is the work intended to be published?** (e.g. names of journals that you are planning to submit to)		
c) List the manuscript's authors in the intended authorship order:		
d) **Stage of publication**		

3. For multi-authored work, please give a statement of contribution covering all authors (if single-author, please skip to section 4**):**			
R. P. obtained the results. R.P. and P.E.L. wrote the paper.			
4. In which chapter(s) of your thesis can this material be found?			
Chapter 5			
5. e-Signatures confirming that the information above is accurate (this form should be co-signed by the supervisor/ senior author unless this is not appropriate, e.g. if the paper was a single-author work):			
Candidate:	Roman Pogodin	**Date:**	26.08.22
Supervisor/ Senior Author (where appropriate)**:**	Peter E. Latham	**Date:**	26.08.22

Contents

List of Figures

List of Tables

Chapter 1

Introductory Material

1.1 Notation

d-dimensional (always column) vectors in $\mathbb{R}^d$: bold lowercase Latin characters ($\mathbf{x}$); $n \times d$ matrices: bold uppercase characters ($\mathbf{X}$); random variables of any dimension: uppercase Latin characters (X); elements of vectors and matrices: not in bold and indexed (x_i, X_{ij}); vectors in an arbitrary Hilbert space: lowercase Greek characters (ϕ); an estimate of a variable y: $\hat{y}$.

The standard dot product in $\mathbb{R}^d$: $\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^{d} x_i y_i$[1]; a dot product in an arbitrary Hilbert space $\mathbb{H}$: $\langle \phi, \psi \rangle_{\mathbb{H}}$; the outer product between two vectors: $\phi \psi^\top$ ($\phi \in \mathbb{R}^d$, $\psi \in \mathbb{R}^n$) or $\phi \otimes \psi$ (any two Hilbert spaces).

1.2 Activity-dependent plasticity in the brain

1.2.1 Overview

Activity-dependent plasticity, meaning synaptic changes induced by neural activity, is considered the basic mechanism behind learning and memory ([1], Chapter 8). There is no single mechanism for synaptic plasticity [2]: it depends on the synapse type (excitatory or inhibitory) and direction of change (*LTP*: long term potentiation, or *LTD*: depression); it can also happen at presynaptic or postsynaptic sites.

Theoretical models often simplify a synapse to a single scalar value, called *weight*, that connects two neurons and changes its value during a task according to a

[1] A random equation uses the "dot" notation $\mathbf{x} \cdot \mathbf{y}$ to reflect five years of disagreeing with my supervisor on what notation is better (it is not the dot notation).

learning rule ([1], Chapter 8).

1.2.2 Neuron models

Most neurons communicate through electrical pulses called action potentials or *spikes*. A (firing) rate model simplifies spiking communication to changes in the neuron's firing rate, or a number of spikes in a time interval [1]. While there's an ongoing debate on the implications of spike-based vs. rate-based models [3], rate-based models are a valuable theoretical tool as they're easier to analyse and simulate. In the remainder of this thesis, we will only use rate-based models, referred simply as "neurons".

A standard neuron model receives inputs from d neurons with firing rates $\mathbf{x}$ through weights $\mathbf{w}$, and changes its own firing rate r as:

$$\tau \dot{r} = -r + f(\mathbf{w}^\top \mathbf{x}), \tag{1.1}$$

where $f(\cdot)$ is a non-linearity that reflects neuron's response to the input; it can simply ensure the firing rate stays non-negative via $f(x) = \max(0, x)$, or additionally reflect activity saturation for large inputs via a sigmoid activation function [1].

Apart from spikes, rate-based models such as in Eq. (1.1) simplify other details of neural processing that can affect computation [1], such as dendritic trees and Dale's law. While these features can be added to rate-based models, they increase model complexity and make both theoretical analysis and simulations of such models harder. As the problems considered in this thesis do not directly involve these features, they will be omitted.

When it comes to plasticity, we can assume that the input is present for long enough, and that plasticity happens on a slower time scale than neural activity ([1], Chapter 8), such that the firing rate arrives at its steady state

$$r = f(\mathbf{w}^\top \mathbf{x}). \tag{1.2}$$

This thesis will be primarily concerned with simplified activity models like Eq. (1.2), and in particular on Hebbian-like models.

1.2.3 Hebbian learning

The origin of Hebbian learning dates back to a 1949 book by D. O. Hebb [4] (p. 62) [2], which contains the following:

> Let us assume then that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows: *When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes a part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.*

Using the model in Eq. (1.2) and assuming excitatory neurons with positive firing rates , Hebb's rule translates into a simple LTP model [1]:

$$\Delta \mathbf{w} \propto r\mathbf{x}. \tag{1.3}$$

Modifications of Eq. (1.3) can introduce LTD (e.g., in the BCM rule [9]), synaptic normalisation (e.g., Oja's rule [10]), and other nonlinear dependencies [11].

The main principle behind Hebb's rule and other instances of Hebbian learning stays constant, even if somewhat distant from Hebb's formulation: learning depends on pre- and post-synaptic activity of two neurons, and on the weight between them. This means that all information required to change the weight between these two neurons is readily available, or local. Hebb's rule is strictly unsupervised, and will be used in Chapter 4. It can be adapted to supervised tasks via neuromodulation, as discussed in Chapter 2.

[2] As R. E. Brown found out [5, 6], D. O. Hebb formulated a very similar idea in his MA thesis [7] in 1932 as a neural mechanism for Pavlovian conditioning. See an essay by P. Milner [8] for a longer version of this story.

1.3 Deep networks

1.3.1 Overview

Deep networks, or more precisely, deep artificial networks, have been around since the mid-20th century [12]. The first models, like the McCulloch-Pitts neuron from 1943 [13], were inspired by the brain, although the connection has been relatively loose throughout the history of deep learning. Here we introduce basic concepts used in deep learning, and discuss the connections with real neurons.

The simplest network with L hidden layers maps an input $\mathbf{x} \in \mathbb{R}^d$ to an output $\hat{y} \in \mathbb{R}^p$ layer by layer with weights $\mathbf{W}^k$ and point-wise nonlinearities f:

$$\mathbf{z}^1 = f\left(\mathbf{W}^1\mathbf{x}\right), \ldots, \mathbf{z}^L = f\left(\mathbf{W}^L\mathbf{z}^{L-1}\right);\ \hat{\mathbf{y}} = g\left(\mathbf{W}^{L+1}\mathbf{z}^L\right). \tag{1.4}$$

Examples of $f(\cdot)$ include

$$\tanh(x) \tag{1.5}$$

$$\text{Linear units} \quad \text{ReLU}(x) = \max(0,x) \tag{1.6}$$

$$\text{LeakyReLU for } a \geq 0 \quad \text{LReLU}(x) = \begin{cases} x & x \geq 0, \\ ax & x < 0, \end{cases} \tag{1.7}$$

$$\text{Scaled exponential LU} \quad \text{SELU}(x) = \beta(\max(0,x) + \min(0, \alpha(e^x - 1))). \tag{1.8}$$

For SELU [14], $\alpha \approx 1.67, \beta \approx 1.05$ in PyTorch `https://pytorch.org/docs/stable/generated/torch.nn.SELU.html`; $g(\cdot)$ depends on the desired form of network's predictions.

Such network can be interpreted as the steady state of a rate model, in which each layer evolves according to

$$\tau \dot{\mathbf{z}}^k = -\mathbf{z}^k + f(\mathbf{W}^k\mathbf{z}^{k-1}). \tag{1.9}$$

To use a network for a certain task, we first need to train in. Training is done by optimising a performance metric for the task, called a *loss function*. For instance, if

we have a dataset with images $\mathbf{x}$ and corresponding labels y that indicate what's on the image, the loss function can be the distance between the network's output $\hat{y}$ and the desired output y.

1.3.2 Connection between artificial and real neurons

Biologically plausible deep learning Despite having connections with real neurons throughout their history, deep networks in their current, most successful state have several features that are not realistic.

The first set of features is about how deep networks function. They communicate in analogue signals, rather than spikes (discussed above), their neurons are not purely excitatory or inhibitory (violating Dale's law [15, 16]), they lack recurrent connections present in the visual stream [17], and so on. Many of these issues are not fundamental: deep networks can be made spiking at the cost of performance [18], Dale's law [19] and recurrent connections [20] can be added with little cost.

The second set is about how deep networks are trained. They typically use gradient descent with precisely calculated updates for each synapse, large amounts of labelled data, and weight sharing among disconnected neurons. This thesis concentrates on this set of problems.

Studies that introduce realistic features into deep networks are referred to as *biologically plausible deep learning*.

Using deep networks to study the brain Deep networks show the importance of complex learning mechanisms – poorly chosen ones simply fail on hard tasks such as visual recognition, even when they're more biologically plausible [21].

On the other hand, well-performing deep networks can have neural representations similar to that of the brain, as extensively shown by studies on the primate visual stream [20, 22, 23] as well as (although less extensively) the mouse visual stream [24, 25].

Biologically plausible deep learning is concentrated on finding deep networks and algorithms for them that are both realistic and effective. It has the potential to discover principles of learning in the brain, and is the common theme of all chapters in this thesis.

Limitations of biologically plausible deep learning The search for more brain-like deep networks and algorithms is based on taking an algorithm/network that performs well on a task relevant to animals (e.g., object recognition) and trying to approximate it with a set of local computations available to neurons. While the brain does not have to rely on the same algorithm (e.g., backpropagation for credit assignment), we already know that such an algorithm works well for the task, and is obtained through trial and error by deep learning researchers. Thus, knowing if real neurons could implement/approximate it allows us to hypothesise that they do.

The limitation of biologically plausible deep learning models, including the ones presented in this thesis, is that they do not perfectly match what we know about the brain. In other words, they suggest what they brain *could* implement if it really wanted to, but don't necessarily say what it does implement. As a result, making concrete experimental predictions from this line of research is tricky.

However, knowing how to implement or approximate a certain algorithm with local computations can still be useful as a first iteration of search for truly biologically plausible learning rules, even if they have to be further refined to match the available data.

The last idea also motives the use of standard computer vision tasks in theoretical neuroscience research. Even if the task structure and amount of available data do not match what the animals have access to, computer vision tasks allow us to test how capable a certain learning scheme is and also compare it to other models in a controlled environment. It also allows us to discard algorithms that might appear reasonable but in fact are incapable of learning beyond the most simple tasks.

1.4 Dependency measures in machine learning

This section contains some machine learning concepts used in Chapter 2 and Chapter 3. While they are not directly related to neuroscience, we will make this connection in the following chapters. The text uses parts from the related papers: Pogodin and Latham (2020) and Li*, Pogodin* et al. (2021).

1.4.1 Mutual information (MI)

Definition 1. For a pair of random variables (X, Y) over $\mathbb{X} \times \mathbb{Y}$, their mutual information is

$$\mathrm{MI}(X,Y) = \int_{\mathbb{X}} \int_{\mathbb{Y}} \log \left(\frac{p_{xy}(x,y)}{p_x(x)\, p_y(y)} \right) p_{xy}(x,y) dx\, dy, \tag{1.10}$$

where p_{xy} is the joint probability density function and p_x, p_y are the marginal ones.

Mutual information measures how dependent two random variables are: if they're completely independent, $p_{xy}(x,y) = p_x(x)\, p_y(y)$ and hence MI is zero. Otherwise, it's non-negative.

In practice, mutual information is hard to use as a dependence measure: it is hard to estimate directly from a limited number of samples [28]; variational estimators are more practical, but only provide a bound on mutual information [29–31].

Finally, mutual information is poorly fit for feature learning in neural networks. To see this, consider a problem with two inputs, A and B (Fig. 1.1, green and purple),

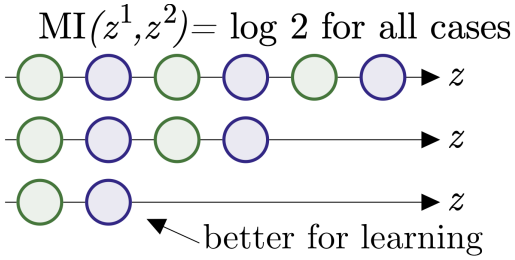

Figure 1.1: Three distributions of positive examples for two classes (green and purple) that have the same mutual information, but drastically different quality for downstream learners.

and a one-dimensional featuriser, parameterised by the integer M, which maps A to Uniform$(\{0,2,\ldots,2M\})$ and B to Uniform$(\{1,3,\ldots,2M+1\})$. When $M = 0$, the inputs are encoded into linearly separable features $A = 0$ and $B = 1$ (Fig. 1.1, bottom). Otherwise when $M > 0$, they are interspersed like *ABABABAB* – a representation which is much harder to work with for downstream learners. Nevertheless, the mutual information between the features of any two augmentations of the same input (a positive pair) is independent of M, that is $H[Z_1] - H[Z_1|Z_2] = \log 2$ for any M. Therefore, even using accurate estimators of mutual information would not

guarantee successful learning, which calls for alternative measures of dependence. (This example is taken from [27], which is the basis of Chapter 3.)

1.4.2 Hilbert-Schmidt Independence Criterion (HSIC)

Another way to measure dependence between random variables is the Hilbert-Schmidt Independence Criterion (HSIC) [32]. Similarly to mutual information, HSIC between X and Y is non-negative, and $\mathrm{HSIC}(X,Y) = 0$ if and only if X and Y are independent (at least for a certain class of kernels [33]), and large values of the measure correspond to "more dependence." The benefit of HSIC is that it has a notion of geometry, and is both statistically and computationally easy to estimate. It has been used in a variety of applications, particularly for independence testing [34], but it has also been maximised in applications such as feature selection [35], clustering [36, 37], active learning [38], and as a classification loss called HSIC Bottleneck [26, 39] (similar ideas were expressed in [40, 41]).

Before defining HSIC, we need to introduce kernel methods. This brief introduction will rely on Chapter 4 of [42]. We start by defining a kernel:

Definition 2. (4.1 in [42]) Let $\mathbb{X}$ be a non-empty set. Then a function $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is called a kernel on $\mathbb{X}$ if there exists a Hilbert space $\mathbb{H}$ and a map $\phi : \mathbb{X} \to \mathbb{H}$ such that for all $x, x' \in \mathbb{X}$ we have

$$k(x,x') = \left\langle \phi(x), \phi(x') \right\rangle_{\mathbb{H}} . \tag{1.11}$$

We call ϕ a feature map and $\mathbb{H}$ a feature space of k. The Hilbert spaces with associated kernels are called *reproducing kernel Hilbert spaces* (RKHS) [43].

An equivalent definition of RKHS is through the evaluation functional:

Definition 3. Consider a non-empty set $\mathbb{X}$, a Hilbert space $\mathbb{H}$ and an evaluation functional L_x defined for all $x \in \mathbb{X}$ as $L_x : f \to f(x)\ \forall f \in \mathbb{H}$. The Hilbert space $\mathbb{H}$ is a reproducing kernel Hilbert space if L_x is a bounded operator:

$$|L_x(f)| = |f(x)| \leq C \|f\|_{\mathbb{H}} . \tag{1.12}$$

The above definition of RKHS gives us a way to define $\phi(x)$ through L_x: by the Riesz representation theorem [44], the bounded function L_x has a unique corresponding vector $\phi(x) \in \mathbb{H}$ such that $L_x(f) = \langle f, \phi(x) \rangle_{\mathbb{H}}$.

The definition of a kernel implies that a kernel is a symmetric function $k(\mathbf{x}, \mathbf{x}')$ that maps $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and is positive-definite,

$$\forall \mathbf{x}_i \in \mathbb{R}^n, \forall c_i \in \mathbb{R}, \quad \sum_{ij} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 . \tag{1.13}$$

Consequently, the matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive-semidefinite.

We will use the following kernels,

$$\text{linear:} \qquad k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' ; \tag{1.14}$$

$$\text{cosine similarity:} \qquad k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}' / (\|\mathbf{x}\|_2 \left\|\mathbf{x}'\right\|_2) ; \tag{1.15}$$

$$\text{Gaussian:} \qquad k(\mathbf{x}, \mathbf{x}') = \exp(-\left\|\mathbf{x} - \mathbf{x}'\right\|_2^2 / (2\sigma^2)) . \tag{1.16}$$

For the linear kernel, $\mathbb{H} = \mathbb{R}^d$, $\phi(\mathbf{x}) = \mathbf{x}$. For the cosine similarity kernel, $\mathbb{H} = \mathbb{R}^d$, $\phi(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|_2$. For the Gaussian kernel, the feature map is infinite-dimensional.

HSIC measures the dependence between two random variables by first taking a nonlinear feature transformation of each, say $\phi : \mathbb{X} \to \mathbb{F}$ and $\psi : \mathbb{Y} \to \mathbb{G}$ (with $\mathbb{F}$ and $\mathbb{G}$ reproducing kernel Hilbert spaces, RKHSes), and then evaluating the norm of the cross-covariance between those features:

$$\text{HSIC}(X, Y) = \|\mathbb{E}[\phi(X) \otimes \psi(Y)] - \mathbb{E}[\phi(X)] \otimes \mathbb{E}[\psi(Y)]\|_{HS}^2 . \tag{1.17}$$

Here $\|\cdot\|_{HS}$ is the Hilbert-Schmidt norm, which for an operator $A : \mathbb{G} \to \mathbb{F}$ is defined as

$$\|A\|_{\text{HS}}^2 = \sum_{j \in J} \|A g_j\|_{\mathbb{G}}^2 , \tag{1.18}$$

where $\{g_j\}_{j \in J}$ is an orthonormal basis of $\mathbb{G}$. In finite dimensions, A is an $m \times n$

matrix, and Hilbert-Schmidt norm is the usual Frobenius norm:

$$\|A\|_{\mathrm{F}}^2 = \sum_{ij} a_{ij}^2 . \tag{1.19}$$

HSIC measures the scale of the correlation in these nonlinear features, which allows it to identify nonlinear dependencies between X and Y with appropriate features ϕ and ψ.

Dot products in an RKHS are by definition *kernel functions*: $k(x,x') = \langle \phi(x), \phi(x') \rangle_{\mathbb{F}}$ and $l(y,y') = \langle \psi(y), \psi(y') \rangle_{\mathbb{G}}$. Let (X',Y'), (X'',Y'') be independent copies of (X,Y); this gives

$$\begin{aligned} \mathrm{HSIC}(X,Y) &= \mathbb{E}\left[k(X,X')l(Y,Y')\right] - 2\,\mathbb{E}\left[k(X,X')l(Y,Y'')\right] && (1.20)\\ &\quad + \mathbb{E}\left[k(X,X')\right]\mathbb{E}\left[l(Y,Y')\right] . && (1.21) \end{aligned}$$

When $X \equiv Y$, meaning that $p_{xy}(\mathbf{x},\mathbf{y}) = p_x(\mathbf{x})\,\delta(\mathbf{y}-\mathbf{x})$, HSIC becomes

$$\mathrm{HSIC}(X,X) = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{x}'}\left(k(\mathbf{x},\mathbf{x}')\right)^2 - 2\,\mathbb{E}_{\mathbf{x}}\left(\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')\right)^2 + \left(\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')\right)^2 . \tag{1.22}$$

If both kernels are linear, it is easy to show that HSIC becomes the squared Frobenius norm of the cross-covariance,

$$\mathrm{HSIC}(X,Y) = \left\|\mathbf{C}_{\mathbf{xy}}\right\|_{\mathrm{F}}^2, \quad \mathbf{C}_{\mathbf{xy}} = \mathbb{E}_{\mathbf{xy}}\,\mathbf{x}\mathbf{y}^\top - \mathbb{E}_{\mathbf{x}}\,\mathbf{x}\,\mathbb{E}_{\mathbf{y}}\,\mathbf{y}^\top . \tag{1.23}$$

In general, HSIC follows the same intuition – it is the squared Hilbert-Schmidt norm (generalisation of the Frobenius norm) of the cross-covariance operator.

HSIC is also straightforward to estimate: given i.i.d. samples $\{(x_1,y_1),\ldots,(x_N,y_N)\}$ drawn i.i.d. from the joint distribution of (X,Y), [32] propose an estimator

$$\widehat{\mathrm{HSIC}}(X,Y) = \frac{1}{(N-1)^2}\mathrm{Tr}\left(KHLH\right), \tag{1.24}$$

where $K_{ij} = k(x_i,x_j)$ and $L_{ij} = l(y_i,y_j)$ are the kernel matrices, and $H = I - \frac{1}{N}\mathbf{1}\mathbf{1}^\top$ is called the centring matrix. This estimator has an $O(1/N)$ bias and $O(1/\sqrt{N})$

variance for N points; an unbiased estimator with the same computational cost is available [35].

Chapter 2

Three-factor Hebbian learning rules for deep networks

This chapter is based on Pogodin and Latham (2020); the results are my own.

Summary The state-of-the art machine learning approach to training deep neural networks, backpropagation, is implausible for real neural networks: neurons need to know their outgoing weights; training alternates between a bottom-up forward pass (computation) and a top-down backward pass (learning); and the algorithm often needs precise labels of many data points. Biologically plausible approximations to backpropagation, such as feedback alignment, solve the weight transport problem, but not the other two. Thus, fully biologically plausible learning rules have so far remained elusive. We present a family of learning rules that does not suffer from any of these problems. It is motivated by the information bottleneck principle (extended with kernel methods), in which networks learn to compress the input as much as possible without sacrificing prediction of the output. The resulting rules have a 3-factor Hebbian structure: they require pre- and post-synaptic firing rates and an error signal – the third factor – consisting of a global teaching signal and a layer-specific term, both available without a top-down pass. They rely on the similarity between pairs of desired outputs instead of precise labels. Moreover, to obtain good performance on hard problems and retain biological plausibility, our rules need divisive normalisation – a known feature of biological networks. Finally, simulations show that our rules perform nearly as well as backpropagation on image classification tasks.

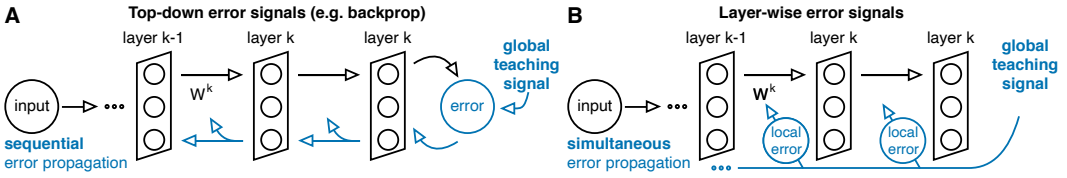

Figure 2.1: **A.** The global error signal is propagated to each layer from the layer above, and used to update the weights. **B.** The global error signal is sent directly to each layer.

2.1 Introduction

Supervised learning in deep networks is typically done using the backpropagation algorithm (or backprop), but in its present form it cannot explain learning in the brain [45]. There are three reasons for this: weight updates require neurons to know their *outgoing* weights, which they do not (the weight transport problem); the forward pass for computation and the backward pass for weight updates need separate pathways and have to happen sequentially (preventing updates of earlier layers before the error is propagated back from the top ones, see Fig. 2.1A); and a large amount of precisely labelled data is needed.

While approximations to backprop such as feedback alignment [46, 47] can solve the weight transport problem, they do not eliminate the requirement for a backward pass or the need for labels. There have been suggestions that a backward pass could be implemented with apical dendrites [48], but it's not clear how well the approach scales to large networks, and the backward pass still has to follow the forward pass in time.

Backprop is not, however, the only way to train deep feedforward networks. An alternative is to use so-called layer-wise update rules, which require only activity in adjacent (and thus connected) layers, along with a global error signal (Fig. 2.1B). Layer-wise training removes the need for both weight transport and a backward pass, and there is growing evidence that such an approach can work as well as backprop [41, 49, 50]. However, while such learning rules are local in the sense that they mainly require activity only in adjacent layers, that does not automatically imply biological plausibility.

Our work focuses on finding a layer-wise learning rule that is biologically

plausible. For that we take inspiration from the information bottleneck principle [51, 52], in which every layer minimises the mutual information between its own activity and the input to the network, while maximising the mutual information between the activity and the correct output (e.g., a label). Estimating the mutual information is hard [28], so [39] proposed the HSIC bottleneck: instead of mutual information they used "kernelized cross-covariance" called Hilbert-Schmidt independence criterion (HSIC). HSIC was originally proposed as a way to measure independence between distributions [32]. Unlike mutual information, HSIC is easy to estimate from data [32], and the information bottleneck objective keeps its intuitive interpretation. Moreover, as we will see, for classification with roughly balanced classes it needs only pairwise similarities between labels, which results in a binary teaching signal.

Here we use HSIC, but to achieve biologically plausible learning rules we modify it in two ways: we replace the HSIC between the input and activity with the kernelized covariance, and we approximate HSIC with "plausible HSIC", or pHSIC, the latter so that neurons don't need to remember their activity over many data points. (However, the objective function becomes an upper bound to the HSIC objective.) The resulting learning rules have a 3-factor Hebbian structure: the updates are proportional to the pre- and post-synaptic activity, and are modulated by a third factor (which could be a neuromodulator [53]) specific to each layer. In addition, to work on hard problems and remain biologically plausible, our update rules need divisive normalisation, a computation done by the primary visual cortex and beyond [54, 55].

In experiments we show that plausible rules generated by pHSIC work nearly as well as backprop on MNIST [56], fashion-MNIST [57], Kuzushiji-MNIST [58] and CIFAR10 [59] datasets. This significantly improves the results from the original HSIC bottleneck paper [39].

2.2 Related work

Biologically plausible approximations to backprop solve the weight transport problem in multiple ways. Feedback alignment (FA) [47] and direct feedback alignment

(DFA) [60] use random fixed weights for the backward pass but scale poorly to hard tasks such as CIFAR10 and ImageNet [21, 61], However, training the feedback pathway to match the forward weights can achieve backprop-level performance [46]. The sign symmetry method [62] uses the signs of the feedforward weights for feedback and therefore doesn't completely eliminate the weight transport problem, but it scales much better than FA and DFA [46, 63]. Other methods include target prop [64, 65] (scales worse than FA [21]) and equilibrium prop [66] (only works on simple tasks, but can work on CIFAR10 at the expense of a more complicated learning scheme [67]). However, these approaches still need to alternate between forward and backward passes.

To avoid alternating forward and backward passes, layer-wise objectives can be used. A common approach is layer-wise classification: [68] used fixed readout weights in each layer (leading to slightly worse performance than backprop); [41] achieved backprop-like performance with trainable readout weights on CIFAR10 and CIFAR100; and [49] achieved backprop-like performance on ImageNet with multiple readout layers. However, layer-wise classification needs precise labels and local backprop (or its approximations) for training. Methods such as contrastive learning [50] and information [52] or HSIC [39] bottleneck and gated linear networks [69, 70] provide alternatives to layer-wise classification, but don't focus on biological plausibility. Biologically plausible alternatives with weaker supervision include similarity matching [41, 71], with [41] reporting a backprop-comparable performance using cosine similarity, and fully unsupervised rules such as [72, 73]. Our method is related to similarity matching; see below for additional discussion.

2.3 A kernel methods-based layer-wise objective

Consider an L-layer feedforward network with input $\mathbf{x}$, layer activity $\mathbf{z}^k$ (for now without divisive normalisation) and output $\hat{\mathbf{y}}$,

$$\mathbf{z}^1 = f\left(\mathbf{W}^1\mathbf{x}\right), \ldots, \mathbf{z}^L = f\left(\mathbf{W}^L\mathbf{z}^{L-1}\right);\ \hat{\mathbf{y}} = f\left(\mathbf{W}^{L+1}\mathbf{z}^L\right). \tag{2.1}$$

The standard training approach is to minimise a loss, $l(\mathbf{y},\hat{\mathbf{y}})$, with respect to the weights, where $\mathbf{y}$ is the desired output and $\hat{\mathbf{y}}$ is the prediction of the network. Here, though, we take an alternative approach: we use layer-wise objective functions, $l_k(\mathbf{x},\mathbf{z}^k,\mathbf{y})$ in layer k, and minimise each $l_k(\mathbf{x},\mathbf{z}^k,\mathbf{y})$ with respect to the weight in that layer, $\mathbf{W}^k$ (simultaneously for every k). The performance of the network is still measured with respect to $l(\mathbf{y},\hat{\mathbf{y}})$, but that quantity is explicitly minimised only with respect to the output weights, $\mathbf{W}^{L+1}$.

To choose the layer-wise objective function, we turn to the information bottleneck [51], which minimises the mutual information between the input and the activity in layer k, while maximising the mutual information between the activity in layer k and the desired output [52]. Mutual information, however, is notoriously hard to compute [28] and poorly suits representation learning as it doesn't encode geometry of neural representations (see the discussion above in Section 1.4.1) . [39] proposed an alternative based on the Hilbert-Schmidt Independence Criterion (HSIC) – the HSIC bottleneck. HSIC is a kernel-based method for measuring independence between probability distribution [32]. Similarly to the information bottleneck, this method tries to balance compression of the input with prediction of the correct output, with a (positive) balance parameter γ,

$$\min_{\mathbf{W}^k}\Big(\mathrm{HSIC}(X,Z^k)-\gamma\mathrm{HSIC}(Y,Z^k)\Big)\,,\ k=1,\ldots,L\,, \tag{2.2}$$

where X, Z^k and Y are random variables, with a distribution induced by the input (the $\mathbf{x}$) and output (the $\mathbf{y}$). HSIC is a measure of dependence: it is zero if its arguments are independent, and increases as dependence increases,

$$\mathrm{HSIC}(A,B) = \int \Delta P_{ab}(\mathbf{a}_1,\mathbf{b}_1)\,k(\mathbf{a}_1,\mathbf{a}_2)\,k(\mathbf{b}_1,\mathbf{b}_2)\,\Delta P_{ab}(\mathbf{a}_2,\mathbf{b}_2)\,, \tag{2.3}$$

where

$$\Delta P_{ab}(\mathbf{a},\mathbf{b}) \equiv (p_{ab}(\mathbf{a},\mathbf{b})-p_a(\mathbf{a})p_b(\mathbf{b}))\,d\mathbf{a}\,d\mathbf{b}\,. \tag{2.4}$$

The kernels, $k(\cdot,\cdot)$ (which might be different for $\mathbf{a}$ and $\mathbf{b}$), are symmetric and positive

definite functions, the latter to insure that HSIC is non-negative. More details on kernels and HSIC are given in Section 1.4.2 and Appendix A.1 (the notation here is slightly different to make the chapter-specific explanations more clear).

HSIC gives us a layer-wise cost function, which eliminates the need for back-prop. However, there is a downside: estimating it from data requires memory. This becomes clear when we consider the empirical estimator of Eq. (2.3) given m observations [32],

$$\begin{aligned}\widehat{\mathrm{HSIC}}(A,B) = &\frac{1}{m^2}\sum_{ij} k(\mathbf{a}_i,\mathbf{a}_j)k(\mathbf{b}_i,\mathbf{b}_j) + \frac{1}{m^2}\sum_{ij} k(\mathbf{a}_i,\mathbf{a}_j)\frac{1}{m^2}\sum_{kl} k(\mathbf{b}_k,\mathbf{b}_l) \\ &- \frac{2}{m^3}\sum_{ijk} k(\mathbf{a}_i,\mathbf{a}_k)k(\mathbf{b}_j,\mathbf{b}_k)\,.\end{aligned} \tag{2.5}$$

In a realistic network, data points are seen one at a time, so to compute the right hand side from m samples, $m-1$ data point would have to be remembered. We solve this in the usual way, by stochastic gradient descent. For the first term we use two data points that are adjacent in time; for the second, we accumulate and store the average over the kernels (see Eq. (2.16) below). The third term, however, is problematic; to compute it, we would have to use three data points. Because this is implausible, we make the approximation

$$\frac{1}{m}\sum_{k} k(\mathbf{a}_i,\mathbf{a}_k)k(\mathbf{b}_j,\mathbf{b}_k) \approx \frac{1}{m^2}\sum_{kl} k(\mathbf{a}_i,\mathbf{a}_k)k(\mathbf{b}_j,\mathbf{b}_l)\,. \tag{2.6}$$

Essentially, we replace the third term in Eq. (2.5) with the second. This leads to "plausible" HSIC, which we call pHSIC,

$$\mathrm{pHSIC}(A,B) = \left(\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2\mathbf{b}_2} - \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2}\right)\left(k(\mathbf{a}_1,\mathbf{a}_2)\,k(\mathbf{b}_1,\mathbf{b}_2)\right)\,. \tag{2.7}$$

We can also write it down in terms of feature maps for A and B:

$$\mathrm{pHSIC}(A,B) = \|\mathbb{E}[\phi(A)\otimes\psi(B)]\|^2_{HS} - \|\mathbb{E}[\phi(A)]\otimes\mathbb{E}[\psi(B)]\|^2_{HS}\,. \tag{2.8}$$

A theoretical downside of pHSIC is that it doesn't guarantee independence: if

A and B are independent, pHSIC is trivially zero. The converse is not true: consider a linear kernel over both A and B, and $A \sim \mathcal{N}(1,1)$, $B \sim 3-2A$. Then, we have

$$\text{HSIC}(A,B) = (\mathbb{E}AB - \mathbb{E}A\,\mathbb{E}B)^2 = (\mathbb{E}(3A-2A^2)-1)^2 = 4\,, \tag{2.9}$$

$$\text{pHSIC}(A,B) = (\mathbb{E}AB)^2 - (\mathbb{E}A\,\mathbb{E}B)^2 = 1-1 = 0\,. \tag{2.10}$$

However, it will not be an issue in our case: the final objective will have a $\text{pHSIC}(A,A)$ penalty. This quantity upper-bounds $\text{HSIC}(A,A)$ (see Appendix A.1.1, Eq. (A.3) for a proof) and reaches zero for a constant A (i.e., when A is independent of A), therefore it can correctly enforce independence.

While pHSIC is easier to compute than HSIC, there is still a potential problem: computing $\text{HSIC}(X, Z^k)$ requires $k(\mathbf{x}_i, \mathbf{x}_j)$, as can be seen in the above equation. But if we knew how to build a kernel that gives a reasonable distance between inputs, we wouldn't have to train the network for classification. So we make one more change: rather than minimising the dependence between X and Z^k (by minimising the first term in Eq. (2.2)), we minimise the (kernelized) covariance of Z^k. To do this, we replace X with Z^k in Eq. (2.2), and define $\text{pHSIC}(A,A)$ via Eq. (2.7) but with $p_{ab}(\mathbf{a},\mathbf{b})$ set to $p_a(\mathbf{a})\delta(\mathbf{a}-\mathbf{b})$,

$$\text{pHSIC}(A,A) = \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2}\left(k(\mathbf{a}_1,\mathbf{a}_2)\right)^2 - \left(\mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2}k(\mathbf{a}_1,\mathbf{a}_2)\right)^2 . \tag{2.11}$$

This gives us the new objective,

$$\min_{\mathbf{W}^k} \mathrm{p}\left(\text{HSIC}(Z^k,Z^k) - \gamma\,\text{pHSIC}(Y,Z^k)\right),\ k=1,\ldots,L\,. \tag{2.12}$$

The new objective preserves the intuition behind the information bottleneck, which is to throw away as much information as possible about the input. It's also an upper bound on the "true" HSIC objective as long as the kernel over the desired outputs is centred ($\mathbb{E}_{\mathbf{y}_1}k(\mathbf{y}_1,\mathbf{y}_2)=0$; see Appendix A.1.1), and doesn't significantly change the performance compared to the "true" HSIC objective (see Figs. A.1 to A.4 in Appendix A.3.8).

Centring the output kernel is straightforward in classification with balanced classes: take $\mathbf{y}$ to be centred one-hot encoded labels (for n classes, $y_i = 1 - 1/n$ for label i and $-1/n$ otherwise), and use the cosine similarity kernel:

$$k(\mathbf{y}_1, \mathbf{y}_2) = \frac{\mathbf{y}_1^\top \mathbf{y}_2}{\|\mathbf{y}_1\| \|\mathbf{y}_2\|} . \tag{2.13}$$

In addition, the teaching signal is binary in this case:

$$k(\mathbf{y}_i, \mathbf{y}_j) = \begin{cases} 1 & \text{if } \mathbf{y}_i = \mathbf{y}_j , \\ -1/(n-1) & \text{otherwise} . \end{cases} \tag{2.14}$$

We will use exactly this signal in our experiments, as the datasets we used are balanced. For slightly unbalanced classes, the signal can still be binary (see Eq. (A.7)).

When $\gamma = 2$, our objective is related to similarity matching. For the cosine similarity kernel over activity, it is close to [41], and for the linear kernel, it is close to [71, 72]. However, the update rule for the cosine similarity kernel is implausible, and for the linear kernel the rule performs poorly (see below and in Appendices A.2.4 and A.2.5). We thus turn to the Gaussian kernel.

2.4 Circuit-level details of the gradient: a hidden 3-factor Hebbian structure

2.4.1 General update rule

To derive the update rule for gradient descent, we need to estimate the gradient of Eq. (2.12). This is relatively straightforward, so we leave the derivation to Appendix A.2, and just report the update rule,

$$\Delta \mathbf{W}^k \propto \sum_{ij} \left(\gamma \mathring{k}(\mathbf{y}_i, \mathbf{y}_j) - 2\mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k) \right) \frac{d}{d\mathbf{W}^k} k(\mathbf{z}_i^k, \mathbf{z}_j^k) , \tag{2.15}$$

where the circle above k means empirical centring,

$$\mathring{k}(\mathbf{a}_i, \mathbf{a}_j) \equiv k(\mathbf{a}_i, \mathbf{a}_j) - \frac{1}{m^2} \sum_{i'j'} k(\mathbf{a}_{i'}, \mathbf{a}_{j'}) \,. \tag{2.16}$$

This rule has a 3-factor form with a global and a local part (Fig. 2.2A): for every pair of points i and j, every synapse needs the same multiplier, and this multiplier needs the similarities between labels (the global signal) and layer activities (the local signal) on two data points. As mentioned in the previous section, on our problems the teaching signal $k(\mathbf{y}_i, \mathbf{y}_j)$ is binary, but we will keep the more general notation here.

However, the derivative in Eq. (2.15) is not obviously Hebbian. In fact, it gives rise to a simple Hebbian update only for some kernels. Below we consider the Gaussian kernel, and in Appendix A.2.4 we show that the cosine similarity kernel (used in [41]) produces an unrealistic rule.

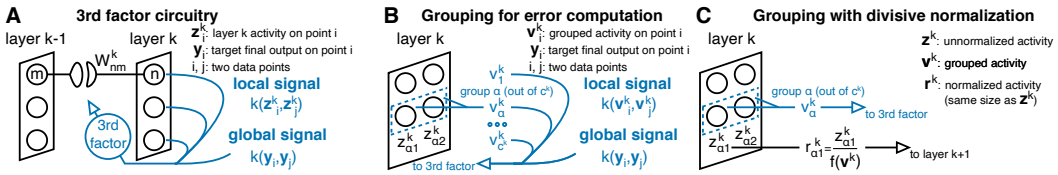

Figure 2.2: **A.** The weight update uses a 3rd factor consisting of a global teaching signal and a local (to the layer) signal, both capturing similarities between two data points. **B.** Grouping: neurons in layer k form c^k groups, each represented by a single number; those numbers are used to compute the local signal. **C.** Grouping with divisive normalisation: the activity of every neuron is normalised using the grouped signal before passing to the next layer (the error is computed as in B).

2.4.2 Gaussian kernel: two-point update

The Gaussian kernel is given by $k(\mathbf{z}^k_i, \mathbf{z}^k_j) = \exp(-\|\mathbf{z}^k_i - \mathbf{z}^k_j\|^2/2\sigma^2)$. To compute updates from a stream of data (rather than batches), we approximate the sum in Eq. (2.15) with only two points, for which we'll again use i and j. If we take a linear fully connected layer (see Eq. (A.17) for the general case), the update over

two points is

$$\Delta \mathbf{W}^k \propto M_{ij}^k (\mathbf{z}_i^k - \mathbf{z}_j^k)(\mathbf{z}_i^{k-1} - \mathbf{z}_j^{k-1})^\top \tag{2.17a}$$

$$M_{ij}^k = -\frac{1}{\sigma^2}\left(\gamma \mathring{k}(\mathbf{y}_i, \mathbf{y}_j) - 2\mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k)\right) k(\mathbf{z}_i^k, \mathbf{z}_j^k)\,. \tag{2.17b}$$

Here M_{ij}^k is the layer-specific third factor.

The role of the third factor is to ensure that if labels, $\mathbf{y}_i$ and $\mathbf{y}_j$, are similar, then the activity, $\mathbf{z}_i^k$ and $\mathbf{z}_j^k$, is also similar, and vice-versa. To see why it has that effect, assume $\mathbf{y}_i$ and $\mathbf{y}_j$ are similar and $\mathbf{z}_i^k$ and $\mathbf{z}_j^k$ are not. That makes M_{ij}^k negative, so the update rule is anti-Hebbian, which tends to move activity closer together. Similarly, if $\mathbf{y}_i$ and $\mathbf{y}_j$ are not similar and $\mathbf{z}_i^k$ and $\mathbf{z}_j^k$ are, M_{ij}^k is positive, and the update rule is Hebbian, which tends to move activity farther apart.

In the current implementation, M is a continuous signal, which might be problematic for a realistic implementation. However, it might be possible to binarize or, at least, approximate M based on the intuition behind the third factor (i.e., it's either positive or negative, based on the similarity between activities and labels). Binarizing a continuous modulatory signal has been successfully used in three-factor Hebbian learning before [74]; a similar implementation could be used here.

The biological implementation of the third factor M is usually attributed to neuromodulators (e.g. through dopaminergic neurons [53]). However, in our case the third factor also contains a term that requires averaging of activity within the layer. This could potentially be done by glia as it can influence plasticity [75, 76] and in some cases do so across synapses [77, 78].

2.4.3 Gaussian kernel with grouping and divisive normalisation

The Gaussian kernel described above works well on small problems (as we'll see below), but in wide networks it needs to compare very high-dimensional vectors, for which there is no easy metric. To circumvent this problem, [41] computed the variance of the response over each channel in each convolutional layer, and then used it for cosine similarity.

We will use the same approach, which starts by arranging neurons into c^k

groups, labelled by α, so that $z^k_{\alpha n}$ is the response of neuron n in group α of layer k (Fig. 2.2B). We'll characterise each group by its "smoothed" variance (meaning we add a positive offset δ), denoted u^k_α,

$$u^k_\alpha \equiv \frac{\delta}{c^k_\alpha} + \frac{1}{c^k_\alpha}\sum_{n'}\left(\mathring{z}^k_{\alpha n'}\right)^2; \quad \mathring{z}^k_{\alpha n} \equiv z^k_{\alpha n} - \frac{1}{c^k_\alpha}\sum_{n'} z^k_{\alpha n'}, \tag{2.18}$$

where c^k_α is the number of neurons in the group α of layer k. One possible kernel would compare the standard deviation (so the square root of u^k_α) across different data points. However, we can get better performance by exponentiation and centring across channels. We thus define a new variable v^k_α,

$$v^k_\alpha = (u^k_\alpha)^{1-p} - \frac{1}{c^k}\sum_{\alpha'}(u^k_{\alpha'})^{1-p}, \tag{2.19}$$

and use this grouped activity in the Gaussian kernel,

$$k(\mathbf{z}^k_i, \mathbf{z}^k_j) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{v}^k_i - \mathbf{v}^k_j\right\|^2\right), \tag{2.20}$$

where, recall, i and j refer to different data points, and $(\mathbf{v}^k)_\alpha = v^k_\alpha$.

To illustrate the approach, we consider a linear network; see Eq. (A.24) in Appendix A.2 for the general case. Taking the derivative of $k(\mathbf{z}^k_i, \mathbf{z}^k_j)$ with respect to the weight in layer k, we arrive at

$$\frac{d\,k(\mathbf{z}^k_i, \mathbf{z}^k_j)}{d\,W^k_{\alpha nm}} \propto -\frac{1}{\sigma^2}k(\mathbf{z}^k_i, \mathbf{z}^k_j)\,(v^k_{\alpha,i} - v^k_{\alpha,j})\left(\frac{\mathring{z}^k_{\alpha n,j}}{(u^k_{\alpha,i})^p}z^{k-1}_{m,i} - \frac{\mathring{z}^k_{\alpha n,j}}{(u^k_{\alpha,j})^p}z^{k-1}_{m,j}\right). \tag{2.21}$$

Because of the term u^k_α in this expression, the learning rule is no longer strictly Hebbian. We can, though, make it Hebbian by assuming that the activity of the presynaptic neurons is $\mathring{z}^k_{\alpha n,j}/(u^k_{\alpha,j})^p$,

$$z^k_{\alpha n} = f\left(\sum_m W^k_{\alpha nm} r^{k-1}_m\right); \quad r^k_{\alpha n} = \frac{\mathring{z}^k_{\alpha n}}{(u^k_\alpha)^p}. \tag{2.22}$$

This automatically introduces divisive normalisation into the network (Fig. 2.2C),

a common "canonical computation" in the brain [54], and in our case makes the update 3-factor Hebbian. It also changes the network from Eq. (2.1) to Eq. (2.22), but that turns out to improve performance. The resulting update rule (again for a linear network; see Eq. (A.24) for the general case) is

$$\begin{aligned} \Delta W^k_{\alpha nm} &\propto M^k_{\alpha,ij}\left(r^k_{\alpha n,i} r^{k-1}_{m,i} - r^k_{\alpha n,j} r^{k-1}_{m,j}\right) \\ M^k_{\alpha,ij} &= -\frac{1}{\sigma^2}\left(\gamma \mathring{k}(\mathbf{y}_i,\mathbf{y}_j) - 2\mathring{k}(\mathbf{z}^k_i,\mathbf{z}^k_j)\right) k(\mathbf{z}^k_i,\mathbf{z}^k_j)\,(v^k_{\alpha,i} - v^k_{\alpha,j})\,. \end{aligned} \tag{2.23}$$

The circuitry to implement divisive normalisation would be recurrent, but is out of scope of this work (see, however, [79] for network models of divisive normalisation). For a convolutional layer, α would denote channels (or groups of channels), and the weights would be summed within each channel for weight sharing. When p, the exponent in Eq. (2.19), is equal to 0.5, our normalisation scheme is equivalent to divisive normalisation in [80] and to group normalisation [81].

Note that the rule is slightly different from the one for the basic Gaussian kernel (Eq. (2.17)): the weight change is no longer proportional to differences of pre- and post-synaptic activities; instead, it is proportional to the difference in their product times the global factor for the channel. This form bears a superficial resemblance to Contrastive Hebbian learning [82, 83]; however, that method doesn't have a third factor, and it generates points i and j using backprop-like feedback connections.

2.4.4 Online update rules for the Gaussian kernel are standard Hebbian updates

Our objective and update rules so far have used batches of data. However, we introduced pHSIC in Section 3.3 because a realistic network has to process data point by point. To show how this can work with our update rules, we first switch indices of points $i,\ j$ to time points $t,\ t-\Delta t$.

Gaussian kernel

The update in Eq. (2.17) becomes

$$\Delta \mathbf{W}^k(t) \propto M^k_{t,t-\Delta t}(\mathbf{z}^k_t - \mathbf{z}^k_{t-\Delta t})(\mathbf{z}^{k-1}_t - \mathbf{z}^{k-1}_{t-\Delta t})^\top\,. \tag{2.24}$$

As an aside, if the point $t-\Delta t$ was presented for some period of time, its activity can be replaced by the mean activity: $\mathbf{z}^k_{t-\Delta t} \approx \boldsymbol{\mu}^k_t$ and $z^{k-1}_{t-\Delta t} \approx \boldsymbol{\mu}^{k-1}_t$. The update becomes a standard Hebbian one,

$$\Delta \mathbf{W}^k(t) \propto M^k_{t,t-\Delta t}(\mathbf{z}^k_t - \boldsymbol{\mu}^k_t)(\mathbf{z}^{k-1}_t - \boldsymbol{\mu}^{k-1}_t)^\top . \tag{2.25}$$

Gaussian kernel with divisive normalisation

The update in Eq. (2.23) allows two interpretations. The first one works just like before: we first introduce time, and then assume that the previous point $r^k_{\alpha n,t-\Delta t}$ is close to the short-term average of activity $\mu^k_{\alpha n,t}$. This results in

$$\Delta W^k_{\alpha nm}(t) \propto M^k_{\alpha,t\,,t-\Delta t}\left(r^k_{\alpha n,t} r^{k-1}_{m,t} - \mu^k_{\alpha n,t}\,\mu^{k-1}_{m,t}\right) . \tag{2.26}$$

The second one uses the fact that for points at times $t-\Delta t,\ t,\ t+\Delta t$ the Hebbian term of point t appears twice: first as $M^k_{\alpha,t\,,t-\Delta t}\, r^k_{\alpha n,t} r^{k-1}_{m,t}$, and then as $-M^k_{\alpha,t+\Delta t\,,t}\, r^k_{\alpha n,t} r^{k-1}_{m,t}$. Therefore we can separate the Hebbian term at time t and write the update as

$$\Delta W^k_{\alpha nm}(t) \propto \left(M^k_{\alpha,t\,,t-\Delta t} - M^k_{\alpha,t+\Delta t\,,t}\right) r^k_{\alpha n,t} r^{k-1}_{m,t} . \tag{2.27}$$

While the Hebbian part of Eq. (2.27) is easier than in Eq. (2.26), it requires the third factor to span a longer time period. In both cases (and also for the plain Gaussian kernel), computing the third factor with local circuitry is relatively straightforward; for details see Eq. (2.41) in Section 2.5.

2.5 Circuitry to implement the update rules

In this section we outline the circuitry needed to compute the Hebbian terms and the third factor for the Gaussian kernel (plain and with grouping and divisive normalisation).

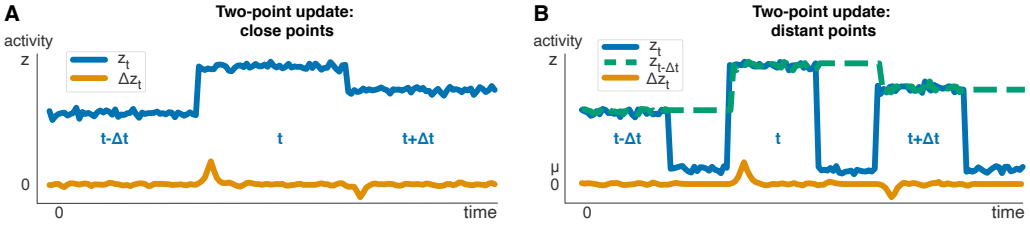

Figure 2.3: **A.** First scenario of the Hebbian updates for two points: plasticity (proportional to Δz_t, orange line) happens when the activity (blue line) switches from one data point to another. **B.** Second scenario: plasticity happens when the second point comes in some time after the first one, which uses memorised activity from the first point (dashed green line).

2.5.1 Hebbian terms

In Section 2.4.4 we proposed online versions for our update rules. For the plain Gaussian kernel (the discussion below also applies to the version with grouping and divisive normalisation), the update is

$$\Delta \mathbf{W}^k(t) \propto M^k_{t,t-\Delta t}(\mathbf{z}^k_t - \mathbf{z}^k_{t-\Delta t})(\mathbf{z}^{k-1}_t - \mathbf{z}^{k-1}_{t-\Delta t})^\top, \tag{2.28}$$

where $t-\Delta t$ represents the data point before the one at time t. In the main text we suggested that $z^k_{t-\Delta t}$ can be approximated by short-term average of activity in layer k.

When Δt is small, the change $z^k_{n,t} - z^k_{n,t-\Delta t}$ for a neuron n can be computed by a smoothed temporal derivative, implemented by convolution with a kernel κ (see Fig. 2.3A),

$$z^k_{n,t} - z^k_{n,t-\Delta t} \approx \Delta z^k_{n,t} \equiv (\kappa * z^k_n)(t); \quad \kappa(t) \propto -(t-c_1)\, e^{-c_2|t-c_1|}\, \Theta(t), \tag{2.29}$$

with c_1 and c_2 are positive.

If Δt is large and potentially variable, the short-term average won't accurately represent the previous point. However, if between trials z^k_n returns to some background activity μ^k_n (see Fig. 2.3B), we can still apply these difference-based updates. In this case the neuron needs to memorise the last significant deviation from the background (i.e., it needs to remember $z^k_{n,t-\Delta t} - \mu^k_{n,t-\Delta t}$). This can be done with a

one-dimensional nonlinear differential equation with "memory", such as

$$\dot{\omega}_{n,t} = \left(z^k_{n,t} - \mu^k_{n,t}\right)^3 - \tanh\left(\left|z^k_{n,t} - \mu^k_{n,t}\right|^3\right) \omega_{n,t} - c\,\omega_{n,t} \tag{2.30}$$

with c small. The intuition is as follows: if the neuron is in the background state, the right hand side of Eq. (2.30) is nearly zero (expect for the leak term $c\,\omega_{n,t}$). Otherwise, the deviation from the mean is large and the right hand side approaches $(z^k_{n,t} - \mu^k_{n,t})^3 - (1+c)\,\omega_{n,t}$, as long as $|z^k - \mu^k_{n,t}| \gg 1$ (due to tanh saturation). This quickly erases the previous $z^k_{n,t-\Delta t}$ and memorises the new one (see Fig. 2.3B with no leak term). We can therefore compute the difference between the last and the current large deviations by convolving the (real) cube root $(\omega^k_{n,t})^{1/3}$ with the same kernel as above, leading to

$$\mathbf{z}^k_{n,t} - \mathbf{z}^k_{n,t-\Delta t} \approx \Delta z^k_{n,t} \equiv \left(\kappa * (\omega^k_n)^{1/3}\right)(t)\,. \tag{2.31}$$

2.5.2 3rd factor for the Gaussian kernel

The update equation for the Gaussian kernel (repeating Eq. (2.17) but in time rather that indices ij, and assuming a linear network as it doesn't affect the third factor) is

$$\Delta W^k_{nm,t} \propto M^k_t\,(z^k_{n,t} - z^k_{n,t-\Delta t})(z^{k-1}_{m,t} - z^{k-1}_{m,t-\Delta t})\,, \tag{2.32}$$

$$M^k_t = -\frac{1}{\sigma^2}\left(\gamma \mathring{k}(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,\mathring{k}(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t})\right) k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t})\,. \tag{2.33}$$

We'll assume that information about the labels, $k(\mathbf{y}_t, \mathbf{y}_{t-\Delta t})$, comes from outside the circuit, so to compute the third factor we just need to compute $k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t})$ (and then centre everything). For the Gaussian kernel, we thus need $\left\|\Delta \mathbf{z}^k_t\right\|^2$, which is given by

$$b^k_{1,t} = \sum_n (\Delta z^k_{n,t})^2\,, \tag{2.34}$$

so that $k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t}) = \exp(-b^k_{1,t}/(2\sigma^2))$. That gives us the uncentred component of

the third factor, denoted $b^k_{2,t}$,

$$b^k_{2,t} = \gamma k(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t}) = \gamma k(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\exp\left(-\frac{1}{2\sigma^2} b^k_{1,t}\right). \quad (2.35)$$

To compute the mean, so that we may centre the third factor, we take an exponentially decaying running average,

$$b^k_{3,t} = \beta\, b^k_{2,t} + (1-\beta)\, b^k_{3,t} \quad (2.36)$$

with $\beta \in (0,1)$ (so that past points are erased as the weights change). Thus, the third factor becomes

$$M^k_t = -\frac{1}{\sigma^2}\left(b^k_{2,t} - b^k_{3,t}\right)\exp\left(-\frac{1}{2\sigma^2} b^k_{1,t}\right). \quad (2.37)$$

If we think of $b^k_{1,t}$, $b^k_{2,t}$ and $b^k_{3,t}$ as neurons, the first two need nonlinear dendrites to compute this signal. In addition, $b^k_{1,t}$ should compute $\Delta z^k_{n,t}$ from $z^k_{n,t}$ at the dendritic level.

2.5.3 3rd factor for the Gaussian kernel with grouping and divisive normalisation

The update for the Gaussian kernel with grouping (repeating Eq. (2.23) but in time, and assuming a linear network as it doesn't affect the third factor) is

$$\Delta W^k_{\alpha nm,t} \propto M^k_{\alpha,t}\left(r^k_{\alpha n,t} r^{k-1}_{m,t} - r^k_{\alpha n,t-\Delta t} r^{k-1}_{m,t-\Delta t}\right), \quad (2.38a)$$

$$M^k_{\alpha,t} = M^k_t\,(v^k_{\alpha,t} - v^k_{\alpha,t-\Delta t}), \quad (2.38b)$$

$$M^k_t = -\frac{1}{\sigma^2}\left(\gamma \mathring{k}(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,\mathring{k}(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t})\right) k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t}). \quad (2.38c)$$

The Hebbian term – the term in parentheses in Eq. (2.38a) – corresponds to the difference of pre- and post-synaptic products, rather than a product of differences: $r^k_{\alpha n,t} r^{k-1}_{m,t} - r^k_{\alpha n,t-\Delta t} r^{k-1}_{m,t-\Delta t} \approx \Delta(r^k_{\alpha n,t} r^{k-1}_{m,t})$. We can compute this difference as before (Eq. (2.29) or Eq. (2.31)), although this update rule requires a different interaction

of the pre- and post-synaptic activity compared to the plain Gaussian kernel in Eq. (2.32).

The third factor is almost the same as for the plain Gaussian kernel, but there are two differences. First, we need to compute the centred normalisation $v^k_{\alpha,t}$ (as in Eq. (2.19)) and its change over time for each group α,

$$\tilde{b}^k_{\alpha,t} = \Delta v^k_{\alpha,t}\,. \tag{2.39}$$

Second, M^k_t is computed for $\Delta v^k_{\alpha,t}$ rather than $\Delta z^k_{n,t}$, such that (cf. Eq. (2.34))

$$\tilde{b}^k_{1,t} = \sum_\alpha (\tilde{b}^k_{\alpha,t})^2\,, \tag{2.40}$$

and $\tilde{b}^k_{2,t}$ (Eq. (2.35) but with $\tilde{b}^k_{1,t}$) and $\tilde{b}^k_{3,t}$ (Eq. (2.36) but with $\tilde{b}^k_{1,t}$ and $\tilde{b}^k_{2,t}$) stay the same.

The third factor becomes

$$M^k_{\alpha,t} = -\frac{1}{\sigma^2}\left(\tilde{b}^k_{2,t} - \tilde{b}^k_{3,t}\right)\exp\left(-\frac{1}{2\sigma^2}\tilde{b}^k_{1,t}\right)\tilde{b}^k_{\alpha,t}\,. \tag{2.41}$$

Essentially, it is the same third factor computation as for the Gaussian kernel, but with an additional group-specific signal $b^k_{\alpha,t}$.

2.6 Experiments

2.6.1 Experimental setup

We compared our learning rule against stochastic gradient descent (SGD), and with an adaptive optimiser and batch normalisation. While networks with adaptive optimisers (e.g. Adam [84]) and batch normalisation (or batchnorm, [85]) perform better on deep learning tasks and are often used for biologically plausible algorithms (e.g. [41, 46]), these features imply non-trivial circuitry (e.g. the need for gradient steps in batchnorm). As our method focuses on what circuitry implements learning, the results on SGD match this focus better.

We used the batch version of the update rule (Eq. (2.15)) only to make large-

scale simulations computationally feasible. We considered the Gaussian kernel, and also the cosine similarity kernel, the latter to compare with previous work [41]. (Note, however, that the cosine similarity kernel gives implausible update rules, see Appendix A.2.4.) For both kernels we tested 2 variants of the rules: plain (without grouping), and with grouping and divisive normalisation (as in Eq. (2.23)). (Grouping *without* divisive normalisation performed as well or worse, and we don't report it here as the resulting update rules are less plausible; see Table A.6 in Appendix A.3.) We also tested backprop, and learning of only the output layer in small-scale experiments to have a baseline result (also with divisive normalisation in the network). In large-scale experiments, we also compared our approach to feedback alignment [47], sign symmetry [62] and layer-wise classification [41, 68].

The nonlinearity was leaky ReLU (LReLU [86]; with a slope of 0.1 for the negative input), but for convolutional networks trained with SGD we changed it to SELU [14] as it performed better. All parameters (learning rates, learning rate schedules, grouping and kernel parameters) were tuned on a validation set (10% of the training set). Optimising HSIC instead of our approximation, pHSIC, didn't improve performance, and the original formulation of the HSIC bottleneck (Eq. (2.2)) performed much worse (not shown). The datasets were MNIST [56], fashion-MNIST [57], Kuzushiji-MNIST [58] and CIFAR10 [59]. We used standard data augmentation for CIFAR10 [59], but no augmentation for the other datasets. All simulation parameters are provided in Appendix A.3. The implementation is available on GitHub: https://github.com/romanpogodin/plausible-kernelized-bottleneck.

2.6.2 Small fully connected network

We start with a 3-layer fully connected network (1024 neurons in each layer). To determine candidates for large-scale experiments, we compare the proposed rules to each other and to backprop. We thus delay comparison with other plausible learning methods to the next section; for performance of plausible methods in shallow networks see e.g. [87]. The models were trained with SGD for 100 epochs with dropout [88] of 0.05, batch size of 256, and the bottleneck balance parameter $\gamma = 2$ (other values of γ performed worse); other parameters are provided in Appendix A.3.

Our results (summarised in Table 2.1 for mean test accuracy; see Table A.3 for deviations between max and min) show a few clear trends across all four datasets: the kernel-based approaches with grouping and divisive normalisation perform similarly to backprop on easy datasets (MNIST and its slightly harder analogues), but not on CIFAR10; grouping and divisive normalisation had little effect on the cosine similarity performance; the Gaussian kernel required grouping and divisive normalisation for decent accuracy. However, we'll see that the poor performance on CIFAR10 is not fundamental to the method; it's because the network is too small.

2.6.3 Large convolutional networks and CIFAR10

Because all learning rules perform reasonably well on MNIST and its related extensions, in what follows we consider only CIFAR10, with the architecture used in [41]: **conv128-256-maxpool-256-512-maxpool-512-maxpool-512-maxpool-fc1024** and its version with double the convolutional channels (denoted 2x; the size of the fully connected layer remained the same). All networks were trained for 500 epochs with $\gamma = 2$, dropout of 0.05 and batch size of 128 (accuracy jumps in Fig. 2.4 indicate learning rate decreases); the rest of the parameters are provided in Appendix A.3.

Table 2.1: Mean test accuracy over 5 runs for a 3-layer (1024 neurons each) fully connected net. Last layer: training of the last layer; cossim: cosine similarity; grp: grouping; div: divisive normalisation.

	backprop		last layer		pHSIC: cossim		pHSIC: Gaussian	
		div		div		grp+div		grp+div
MNIST	98.6	98.4	92.0	95.4	94.9	96.3	94.6	98.1
fashion-MNIST	90.2	90.8	83.3	85.7	86.3	88.1	86.5	88.8
Kuzushiji-MNIST	93.4	93.5	71.2	78.2	80.4	87.2	80.2	91.1
CIFAR10	60.0	60.3	39.2	38.0	51.1	47.6	41.4	46.4

For SGD with divisive normalisation (and also grouping for pHSIC-based methods; first two rows in Table 2.2), layer-wise classification, sign symmetry and the cosine similarity kernel performed as well as backprop (consistent with previous findings [41, 63]); feedback alignment (and layer-wise classification with FA) performed significantly worse. In those cases, increasing the width of the network had a marginal effect on performance. The Gaussian kernel performed worse

Table 2.2: Mean test accuracy on CIFAR10 over 5 runs for the 7-layer conv nets (1x and 2x wide). FA: feedback alignment; sign sym.: sign symmetry; layer class.: layer-wise classification; cossim: cosine similarity; divnorm: divisive normalisation; bn: batchnorm.

	backprop	FA	sign sym.	layer class.		pHSIC + grouping	
					+FA	cossim	Gaussian
1x + SGD + divnorm	91.0	80.4	89.5	90.5	81.0	89.8	86.2
2x + SGD + divnorm	90.9	80.6	91.3	91.3	81.2	91.3	90.4
1x + AdamW + bn	94.1	82.4	93.6	92.1	90.3	91.3	89.9
2x + AdamW + bn	94.3	81.6	93.9	92.1	91.1	91.9	91.0

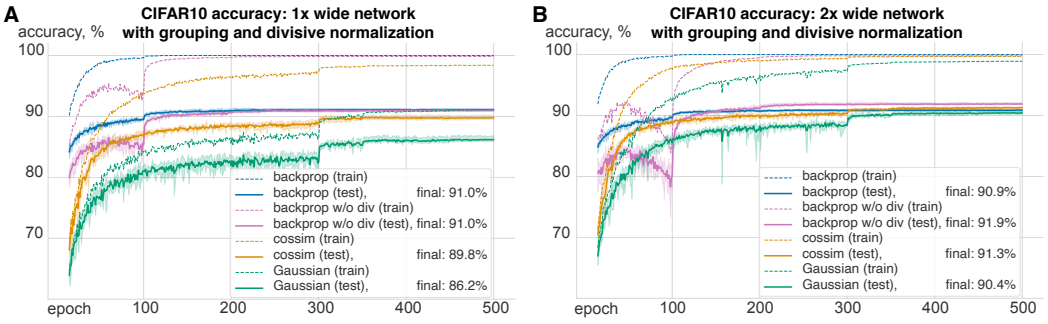

Figure 2.4: Performance of backprop, cosine similarity kernel (cossim) and Gaussian kernel on CIFAR10 with SGD, grouping and divisive normalisation (and without for backprop; in pink). Solid lines: mean test accuracy over 5 random seeds; shaded areas: min/max test accuracy over 5 seeds; dashed lines: mean training accuracy over 5 seeds. **A.** 1x wide network: cosine similarity nearly matches backprop but doesn't achieve perfect training accuracy; Gaussian kernel lags behind cosine similarity. **B.** 2x wide network: backprop performance slightly improves; both kernels nearly match backprop performance, but Gaussian kernel still doesn't achieve perfect training accuracy.

than backprop on the 1x network, but increasing the width closed the performance gap. A closer look at the learning dynamics of pHSIC-based methods and backprop (Fig. 2.4) reveals low training accuracy with the Gaussian kernel on the 1x wide net, vs. almost 100% for 2x, explaining low test accuracy.

For AdamW [89] with batchnorm [85] (last two rows in Table 2.2), performance improved for all objectives, but it improved more for backprop (and sign symmetry) than for the pHSIC-based objectives. However, batch normalisation (and in part adaptive learning rates of AdamW) introduces yet another implausible feature to the training method due to the batch-wide activity renormalisation.

Only backprop, layer-wise classification (without feedback alignment) and the

cosine similarity kernels performed well without any normalisation (see Tables A.6 and A.7); for the other methods some kind of normalisation was crucial for convergence. Backprop without divisive normalisation (solid pink line in Fig. 2.4) had non-monotonic performance, which can be fixed with a smaller learning rate at the cost of slightly worse performance.

The small difference between backpropagation and layer-wise methods, and in particular the pHSIC objective with the Gaussian kernel and divisive normalisation, is likely due to early overfitting of layer-wise methods. Essentially, each layer learns features specific to its own objective [90], which can impair downstream performance as some information could be lost early on. While the difference is small in our case, it can potentially increase on harder tasks; this can be fixed by grouping several layers into a block for a single block-wise objective (see [90]) at the expense of the need of backpropagation within the block.

2.7 Discussion

We proposed a layer-wise objective for training deep feedforward networks based on the kernelized information bottleneck, and showed that it can lead to biologically plausible 3-factor Hebbian learning. Our rules work nearly as well as backpropagation on a variety of image classification tasks. Unlike in classic Hebbian learning, where the pre- and post-synaptic activity triggers weight changes, our rules suggest large *fluctuations* in this activity should trigger plasticity (e.g. when a new object appears). Our main update rule (Eq. (2.38a)) can be implemented as a difference between standard Hebbian terms, effectively offloading the computations of fluctuations to the third factor. However, plasticity under this rule would still happen when either the input or the teaching signal change significantly. Such difference-based rules are reminiscent of Differential Hebbian learning [91] that is used to account for STDP data. However, it is not clear how Differential Hebbian learning coincides with neuromodulation, and if it can work for temporally distant data points (like in Fig. 2.3B).

Our learning rules do not need precise labels; instead they need only a binary

signal: whether or not the previous and the current point have the same label. This allows networks to build representations with weaker supervision. We did train the last layer with precise labels, but that was only to compute accuracy; the network would learn just as well without it. To completely avoid supervision in hidden layers, it is possible to adapt our learning scheme to the contrastive (or self-supervised) setting as in Contrastive Predictive Coding [29, 50] and SimCLR [92] (which stands for " simple framework for contrastive learning of visual representations").

Our rules do, though, need a global signal, which makes up part of the third factor. Where could it come from? In the case of the visual system, we can think of it as a "teaching" signal coming from other sensory areas. For instance, the more genetically pre-defined olfactory system might tell the brain that two successively presented objects smell differently, and therefore should belong to different classes. The third factor also contains a term that is local to the layer, but requires averaging of activity within the layer. This could be done by cells that influence plasticity, such as dopaminergic neurons [53] or glia [75–78]. The rules we discussed predict a specific form of the third factor and a corresponding circuitry that computes it locally (Section 2.5).

Although our approach is a step towards fully biologically plausible learning rules, it still suffers from some unrealistic features. The recurrence in our networks is limited to that necessary for divisive normalisation, and has no excitatory within-layer recurrence or top-down signals. Those might be necessary to go from image classification to more realistic tasks (e.g., video). Our networks also allow negative firing rates due to the use of leaky ReLU and SELU nonlinearities. The latter (which we used to compensate for the lack of batchnorm) saturates for large negative inputs, and therefore the activity of each neuron can be viewed as a value relative to the background firing. Our main experiments also use convolutional networks, which are implausible due to weight sharing among neurons. Achieving good performance without weight sharing is an open question, although there are some results for backprop [21]. Finally, a sequential implementation of our objective relies on a random order of input examples. If a network sees similar inputs with the same

label for a long time, its representations can collapse to a constant value (a similar phenomenon occurs in self-supervised learning [93]). The frequency of "label switches" required for successful learning can be an interesting future direction.

We showed that our rules can compete with backprop and its plausible approximations on CIFAR10, even though they rely on less supervision and simpler error signals. It should be possible to scale our learning rules to larger datasets, such as CIFAR100 [59] and ImageNet [94], as suggested by results from other layer-wise rules [41, 49, 50]. The layer-wise objectives can also make the theoretical analysis of deep learning easier. In fact, recent work analysed a similar type of kernel-based objectives, showing its optimality with one hidden layer and backprop-comparable performance in deeper networks [95].

The human brain contains about 10^{11} neurons, of which only about 10^6 – less than one per 100,000 – are directly connected to the outside world; the rest make up hidden layers. Understanding how such a system updates its synaptic strengths is one of the most challenging problems in neuroscience. We proposed a family of biologically plausible learning rules for feedforward networks that have the potential to solve this problem. For a complete understanding of learning they need, of course, to be adapted to unsupervised and recurrent settings, and verified experimentally. In addition, our learning rules are much more suitable for neuromorphic chips than standard backprop, due to the distributed nature of weight updates, and so could massively improve their scalability.

Chapter 3

A kernel methods approach to self-supervised learning

This chapter is based on Li*, Pogodin* et al. (2021); the results are obtained in close collaboration with other authors, and in particular Yazhe Li (we are the two joint first authors). The theoretical results presented in this chapter are mostly my results, while the experimental ones are due to Yazhe Li; the latter are mostly omitted. Yazhe Li's code is available at `https://github.com/deepmind/ssl_hsic`; my own code was not publicly shared.

Summary We approach self-supervised learning of image representations from a statistical dependence perspective, proposing Self-Supervised Learning with the Hilbert-Schmidt Independence Criterion (SSL-HSIC). SSL-HSIC maximises dependence between representations of transformations of an image and the image identity, while minimising the kernelized variance of those representations. This framework yields a new understanding of InfoNCE (variation of the Noise-Contrastive Estimation loss), a variational lower bound on the mutual information (MI) between different transformations. While the MI itself is known to have pathologies which can result in learning meaningless representations, its bound is much better behaved: we show that it implicitly approximates SSL-HSIC (with a slightly different regularizer). Our approach also gives us insight into Bootstrap Your Own Label (BYOL), a negative-free SSL method, since SSL-HSIC similarly learns local neighbourhoods of samples. SSL-HSIC allows us to directly optimise statistical dependence in

time linear in the batch size, without restrictive data assumptions or indirect mutual information estimators. Trained with or without a target network, SSL-HSIC matches the current state-of-the-art for standard linear evaluation on ImageNet [96], semi-supervised learning and transfer to other classification and vision tasks such as semantic segmentation, depth estimation and object recognition.

3.1 Introduction

Learning general-purpose visual representations without human supervision is a long-standing goal of machine learning. Specifically, we wish to find a feature extractor that captures the image semantics of a large unlabelled collection of images, so that e.g. various image understanding tasks can be achieved with simple linear models. One approach takes the latent representation of a likelihood-based generative model [97–103]; such models, though, solve a harder problem than necessary since semantic features need not capture low-level details of the input. Another option is to train a *self-supervised* model for a "pretext task," such as predicting the position of image patches, identifying rotations, or image inpainting [104–109]. Designing good pretext tasks, however, is a subtle art, with little theoretical guidance available. Recently, a class of models based on contrastive learning [29, 92, 110–115] has seen substantial success: dataset images are cropped, rotated, colour shifted, etc. into several *views*, and features are then trained to pull together representations of the "positive" pairs of views of the same source image, and push apart those of "negative" pairs (from different images). These methods are either understood from an information theoretic perspective as estimating the mutual information between the "positives" [29], or explained as aligning features subject to a uniformity constraint [116]. Another line of research [93, 117] attempts to learn representation without the "negative" pairs, but requires either a target network or stop-gradient operation to avoid collapsing.

We examine the contrastive framework from a statistical dependence point of view: feature representations for a given transformed image should be highly dependent on the image identity (Fig. 3.1). To measure dependence, we turn to the

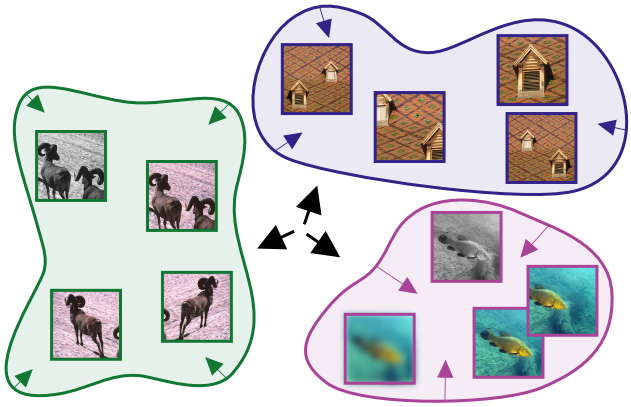

Figure 3.1: Statistical dependence view of contrastive learning: representations of transformed images should highly depend on image identity. Measuring dependence with HSIC, this pushes different images' representation distributions apart (black arrows) and pulls representations of the same image together (coloured shapes).

Hilbert-Schmidt Independence Criterion (HSIC) [32], and propose a new loss for self-supervised learning which we call SSL-HSIC. Our loss is inspired by HSIC Bottleneck [26, 39], an alternative to Information Bottleneck [118], where we use the image identity as the label, but change the regularisation term.

Through the dependence maximisation perspective, we present a unified view of various self-supervised losses. Previous work [119] has shown that the success of InfoNCE cannot be solely attributed to properties of mutual information, in particular because mutual information (unlike kernel measures of dependence) has no notion of geometry in feature space: for instance, *all* invertible encoders achieve maximal mutual information, but they can output dramatically different representations with very different downstream performance [119]. Variational bounds on mutual information do impart notions of locality that allow them to succeed in practice, departing from the mutual information quantity that they try to estimate. We prove that InfoNCE, a popular such bound, in fact approximates SSL-HSIC with a variance-based regularisation. Thus, InfoNCE can be thought of as working because it implicitly estimates a kernel-based notion of dependence. We additionally show SSL-HSIC is related to metric learning, where the features learn to align to the structure induced by the self-supervised labels. This perspective is closely related to the objective of Bootstrap Your Own Label (BYOL) [93], and can explain properties such as

alignment and uniformity [116] observed in contrastive learning.

Our perspective brings additional advantages, in computation and in simplicity of the algorithm, compared with existing approaches. Unlike the indirect variational bounds on mutual information [29–31], SSL-HSIC can be directly estimated from mini-batches of data. Unlike "negative-free" methods, the SSL-HSIC loss itself penalises trivial solutions, so techniques such as target networks are not needed for reasonable outcomes. Using a target network does improve the performance of our method, however, suggesting target networks have other advantages that are not yet well understood. Finally, we employ random Fourier features [120] in our implementation, resulting in cost linear in batch size.

Our main contributions are as follows:

- We introduce SSL-HSIC, a principled self-supervised loss using kernel dependence maximisation.
- We present a unified view of contrastive learning through dependence maximisation, by establishing relationships between SSL-HSIC, InfoNCE, and metric learning.
- On ImageNet, our method achieves top-1 accuracy of 74.8% and top-5 accuracy of 92.2% with linear evaluations (see Fig. 3.3 for a comparison with other methods), top-1 accuracy of 80.2% and Top-5 accuracy of 94.7% with fine-tuning, and competitive performance on a diverse set of downstream tasks.

3.2 Background

3.2.1 Self-supervised learning

Recent developments in self-supervised learning, such as contrastive learning, try to ensure that features of two random views of an image are more associated with each other than with random views of other images. Typically, this is done through some variant of a classification loss, with one "positive" pair and many "negatives." Other methods can learn solely from "positive" pairs, however. There have been many variations of this general framework in the past few years.

[29] first formulated the InfoNCE loss (as a variation of the Noise-Contrastive Estimation loss), which estimates a lower bound of the mutual information between the feature and the context. SimCLR [92, 121] (which stands for " simple framework for contrastive learning of visual representations") carefully investigates the contribution of different data augmentations, and scales up the training batch size to include more negative examples. Momentum Contrast (MoCo) [112] increases the number of negative examples by using a memory bank. Bootstrap Your Own Label (BYOL) [93] learns solely on positive image pairs, training so that representations of one view match that of the other under a moving average of the featuriser. Instead of the moving average, SimSiam [117] suggests a stop-gradient on one of the encoders is enough to prevent BYOL from finding trivial solutions. SwAV [114] (which stands for "Swapping Assignments between multiple Views of the same image") clusters the representation online, and uses distance from the cluster centres rather than computing pairwise distances of the data. Barlow Twins [115] uses an objective related to the cross-correlation matrix of the two views, motivated by redundancy reduction. It is perhaps the most related to our work in the literature (and their covariance matrix can be connected to HSIC [122]), but our method measures dependency more directly. While Barlow Twins decorrelates components of final representations, we maximise the dependence between the image's abstract identity and its transformations.

On the theory side, InfoNCE is proposed as a variational bound on Mutual Information between the representation of two views of the same image [29, 31]. [119] observes that InfoNCE performance cannot be explained solely by the properties of the mutual information, however, but is influenced more by other factors, such as the formulation of the estimator and the architecture of the feature extractor. Essentially, representations with the same MI can have drastically different representational qualities, as explained earlier in Section 1.4.1.

Later theories suggest that contrastive losses balance alignment of individual features and uniformity of the feature distribution [116], or in general alignment and some loss-defined distribution [123]. We propose to interpret the contrastive loss

through the lens of statistical dependence, and relate it to metric learning, which naturally leads to alignment and uniformity.

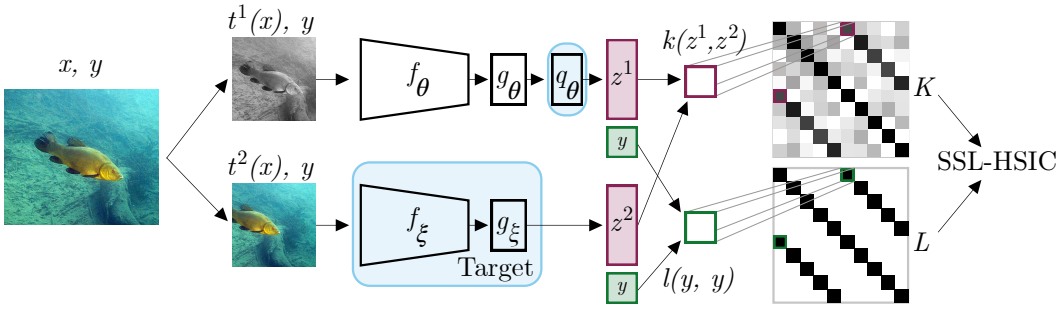

Figure 3.2: Architecture and SSL-HSIC objective. A self-supervised label y – an indicator of the image identity – is associated with an image x. Image transformation functions t are sampled and applied to the original image, resulting in views $t^1(x)$ and $t^2(x)$. Features z^1 and z^2 are obtained after passing the augmented views through encoder (f), projector (g), and possibly predictor (q) networks, while label y is retained. Kernel matrices, K for the latents and L for the labels, are computed on the mini-batch of data; SSL-HSIC is estimated with K and L as in (3.9). The blue boxes reflect two potential options: when using a target network, ξ is a moving average of θ, and a predictor network q is added; without the target network, q is removed and ξ is simply equal to θ.

3.3 Self-supervised learning with Kernel Dependence Maximisation

Our method builds on the self-supervised learning framework used by most of the recent self-supervised learning approaches [92, 93, 110, 112, 114, 115, 117]. For a dataset with N points x_i, each point goes through a random transformation $t^p(x_i)$ (e.g. random crop), and then forms a feature representation $z_i^p = f_\theta(t^p(x_i))$ with an encoder network f_θ. We associate each image x_i with its identity y_i, which works as a one-hot encoded label: $y_i \in \mathbb{R}^N$ and $(y_i)_d = 1$ iff $d = i$ (and zero otherwise). To match the transformations and image identities, we maximise the dependence between z_i and y_i such that z_i is predictive of its original image. To build representations suitable for downstream tasks, we also need to penalise high-variance representations. These ideas come together in our HSIC-based objective (an overview of HSIC is provided

in Section 1.4.2) for self-supervised learning, which we term SSL-HSIC:

$$L_{\text{SSL-HSIC}}(\theta) = -\text{HSIC}(Z,Y) + \gamma\sqrt{\text{HSIC}(Z,Z)}\,. \tag{3.1}$$

Unlike contrastive losses, which make the z_i^p from the same x_i closer and those from different x_j more distant, we propose an alternative way to match different transformations of the same image with its *abstract identity* (e.g. position in the dataset).

Our objective also resembles the HSIC bottleneck for supervised learning [39] (in particular, the version of [26]), but ours uses a square root for $\text{HSIC}(Z,Z)$. The square root makes the two terms on the same scale: $\text{HSIC}(Z,Y)$ is effectively a dot product, and $\sqrt{\text{HSIC}(Z,Z)}$ a norm, so that e.g. scaling the kernel by a constant does not change the relative amount of regularisation; this also gives better performance in practice. Other prior work on maximising HSIC [37, 124] used $\text{HSIC}(Z,Y)/\sqrt{\text{HSIC}(Z,Z)\,\text{HSIC}(Y,Y)}$, or equivalently [125] the distance correlation [126]; the kernel-target alignment [127, 128] is also closely related. Here, the overall scale of either kernel does not change the objective. Our $\text{HSIC}(Y,Y)$ is constant (hence absorbed in γ), and we found an additive penalty to be more stable in optimisation than dividing the estimators.

Due to the one-hot encoded labels, we can rewrite $\text{HSIC}(Z,Y)$ as (see Theorem 1 in Appendix B.1)

$$\text{HSIC}(Z,Y) \propto \mathbb{E}_{z_1,z_2\sim pos}\left[k(z_1,z_2)\right] - \mathbb{E}_{z_1}\mathbb{E}_{z_2}\left[k(z_1,z_2)\right]\,, \tag{3.2}$$

where the first expectation is over the distribution of "positive" pairs (those from the same source image), and the second one is a sum over all image pairs, including their transformations. The first term in (3.2) pushes representations belonging to the same image identity together, while the second term keeps mean representations for each identity apart (as in Fig. 3.1). The scaling of $\text{HSIC}(Z,Y)$ depends on the choice of the kernel over Y, and is irrelevant to the optimisation.

This form also reveals two key theoretical results. Section 3.3.1 shows that

InfoNCE is better understood as an HSIC-based loss than a mutual information between views. Then, $\text{HSIC}(Z,Y)$ is proportional to the average kernel-based distance between the distribution of views for each source image (the maximum mean discrepancy, MMD; see Appendix B.2.2).

3.3.1 Connection to InfoNCE

In this section we show the connection between InfoNCE and our loss; see Appendix B.2.1 for the full derivation. The standard definition of InfoNCE takes $2N$ points, such that each point i is paired with another positive example i',

$$\hat{L}_{\text{InfoNCE}}(\theta) = -\frac{1}{2N}\sum_{i=1}^{2N}\log\frac{\exp(k(z_i,z_{i'}))}{\sum_{j\neq i}k(z_i,z_k)}\,. \tag{3.3}$$

We first write InfoNCE in its infinite sample size limit (see [116] for a derivation) as

$$L_{\text{InfoNCE}}(\theta) = -\mathbb{E}_{z_1,z_2\sim\text{pos}}\left[k(z_1,z_2)\right] + \mathbb{E}_{z_1}\log\mathbb{E}_{z_2}\left[\exp\left(k(z_1,z_2)\right)\right], \tag{3.4}$$

where the last two expectations are taken over all points, and the first is over the distribution of positive pairs. The kernel $k(z_1,z_2)$ was originally formulated as a scoring function in the form of a dot product [29], and then a scaled cosine similarity [92]. Both functions are valid kernels.

Now assume that $k(z_1,z_2)$ doesn't deviate much from $\mathbb{E}_{z_2}\left[k(z_1,z_2)\right]$, Taylor-expand the exponent in (3.4) around $\mathbb{E}_{z_2}\left[k(z_1,z_2)\right]$, then expand $\log(1+\mathbb{E}_{z_2}(\dots)) \approx \mathbb{E}_{z_2}(\dots)$. We obtain an $\text{HSIC}(Z,Y)$-based objective:

$$L_{\text{InfoNCE}}(\theta) \approx \underbrace{-\mathbb{E}_{z_1,z_2\sim\text{pos}}\left[k(z_1,z_2)\right] + \mathbb{E}_{z_1}\mathbb{E}_{z_2}\left[k(z_1,z_2)\right]}_{\propto -\text{HSIC}(Z,Y)} + \frac{1}{2}\underbrace{\mathbb{E}_{z_1}\left[\mathbb{V}\text{ar}_{z_2}\left[k(z_1,z_2)\right]\right]}_{\text{variance penalty}}. \tag{3.5}$$

Since the scaling of $\text{HSIC}(Z,Y)$ is irrelevant to the optimization, we assume scaling to replace $\propto$ with $=$. In the small variance regime, we can show that for the

right γ,

$$-\text{HSIC}(Z,Y)+\gamma\text{HSIC}(Z,Z)\leq L_{\text{InfoNCE}}(\theta)+o(\text{variance})\,. \tag{3.6}$$

For $\text{HSIC}(Z,Z)\leq 1$, we also have that

$$-\text{HSIC}(Z,Y)+\gamma\text{HSIC}(Z,Z)\leq L_{\text{SSL}-\text{HSIC}}(\theta) \tag{3.7}$$

due to the square root. InfoNCE and SSL-HSIC in general don't quite bound each other due to discrepancy in the variance terms. Although we did not test the discrepancy in our experiments, both loss functions performed similarly (Table 3.8a).

Why should we prefer the HSIC interpretation of InfoNCE? Initially, InfoNCE was suggested as a variational approximation to the mutual information between two views [29]. It has been observed, however, that using tighter estimators of mutual information leads to worse performance [119]. It is also simple to construct examples where InfoNCE finds different representations while the underlying MI remains constant [119]. Alternative theories suggest that InfoNCE balances alignment of "positive" examples and uniformity of the overall feature representation [116], or that (under strong assumptions) it can identify the latent structure in a hypothesized data-generating process, akin to nonlinear ICA [129]. Our view is consistent with these theories, but doesn't put restrictive assumptions on the input data or learned representations. In ablation studies we show that our interpretation gives rise to a better objective in practice.

3.3.2 Estimator of SSL-HSIC

To use SSL-HSIC, we need to correctly and efficiently estimate (3.1). Both points are non-trivial: the self-supervised framework implies non-i.i.d. batches (due to positive examples), while the estimator in (1.24) assumes i.i.d. data; moreover, the time to compute (1.24) is quadratic in the batch size.

First, for $\text{HSIC}(Z,Z)$ we use the biased estimator in (1.24). Although the i.i.d. estimator (1.24) results in an $O(1/B)$ bias for B original images in the batch size (see Corollary 1), the batch size B is large in our case and therefore the bias is negligible.

For $\mathrm{HSIC}(Z,Y)$ the situation is more delicate: the i.i.d. estimator needs re-scaling, and its bias depends on the number of positive examples M, which is typically very small (usually 2). We propose the following estimator:

$$\widehat{\mathrm{HSIC}}(Z,Y) = \frac{\Delta l}{N}\left(\frac{1}{BM(M-1)}\sum_{ipl} k(z_i^p, z_i^l) - \frac{1}{B^2M^2}\sum_{ijpl} k(z_i^p, z_j^l) - \frac{1}{M-1}\right), \tag{3.8}$$

where i and j index original images, and p and l their random transformations; k is the kernel used for latent Z, l is the kernel used for the labels, and $\Delta l = l(i,i) - l(i,j)$ (l for same labels minus l for different labels). Note that due to the one-hot structure of self-supervised labels Y, the standard (i.i.d.-based) estimator would miss the $1/N$ scaling and the $M-1$ correction (the latter is important in practice, as we usually have $M=2$). See Theorem 2 for the derivations.

For convenience, we assume $\Delta l = N$ (any scaling of l can be subsumed by γ), and optimise

$$\widehat{L}_{\mathrm{SSL-HSIC}}(\theta) = -\widehat{\mathrm{HSIC}}(Z,Y) + \gamma\sqrt{\widehat{\mathrm{HSIC}}(Z,Z)}. \tag{3.9}$$

The computational complexity of the proposed estimators is $O(B^2M^2)$ for each mini-batch of size B with M augmentations. We can reduce the complexity to $O(BM)$ by using random Fourier features (RFF) [120], which approximate the kernel $k(z_1,z_2)$ with a carefully chosen random D-dimensional approximation $R(z_1)^\top R(z_2)$ for $R(z): \mathbb{R}^{D_z} \to \mathbb{R}^D$, such that $k(z_1,z_2) = \mathbb{E}\left[R(z_1)^\top R(z_2)\right]$. Fourier frequencies are sampled independently for the two kernels on Z in $\mathrm{HSIC}(Z,Z)$ at each training step. We leave the details on how to construct $R(z)$ for the kernels we use to Appendix B.3.

Random Fourier features are not the only way to approximate kernel matrices. Data subsampling can reduce the effective batch size, but for αB subsampled points computational complexity would still scale quadratically with batch size B. The Nyström method [130] is another approach that can reduce computational complexity, but it performed poorly in our preliminary experiments (not shown).

3.3.3 Connection with biology

The loss discussed here can be approximated by the pHSIC loss proposed in Chapter 2. Therefore, in a single layer, SSL-HSIC could be computed in the same manner. It also has the same intuition for the nature of the supervision signal: the positive examples discussed in this chapter function identically to same class examples in Chapter 2.

The difference, however, comes from the layer-wise nature of the approach in Chapter 2 and the top-down approach here, as the latter requires backpropagation. A number of works successfully applied self-supervised losses in a layer-wise setting [131, 132], therefore SSL-HSIC can also be applied layer-wise.

3.4 Experiments

Note: the experimental results are due to Yazhe Li. They are included here for completeness. I have conducted similar experiments on a smaller scale during this project, but I didn't have access to the computational resources needed for the final experiments.

In this section, we present our experimental setup, where we assess the performance of the representation learned with SSL-HSIC both with and without a target network. First, we train a model with a standard ResNet-50 backbone using SSL-HSIC as objective on the training set of ImageNet ILSVRC-2012 [96]. The main result is summarised in Fig. 3.3. For evaluation, we retain the backbone as a feature extractor for downstream tasks. We evaluate the representation on various downstream tasks including classification, object segmentation, object detection and depth estimation.

3.4.1 Implementation

Architecture Fig. 3.2 illustrates the architecture we used for SSL-HSIC in this section. To facilitate comparison between different methods, our encoder f_θ uses the standard ResNet-50 backbone without the final classification layer. The output of the encoder is a 2048-dimension embedding vector, which is the representation used for downstream tasks. As in BYOL [93], our projector g and predictor q networks are

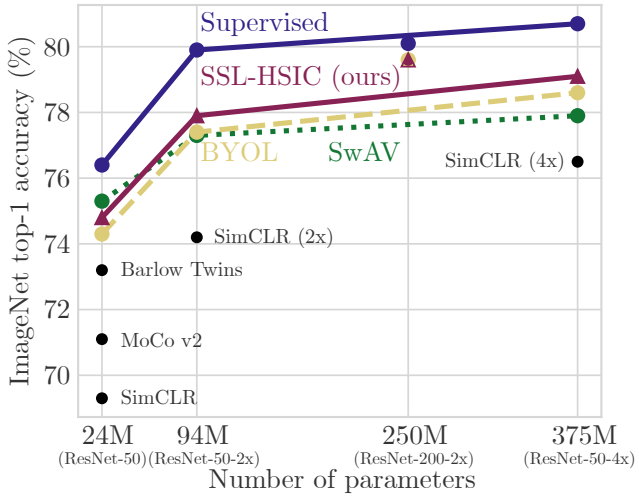

Figure 3.3: Top-1 accuracies with linear evaluation for different ResNet architecture and methods: supervised (as in [93]), SSL-HSIC (with a target network; ours), BYOL [93], SwAV [114], SimCLR [92], MoCo v2 [113] and Barlow Twins [115]. SimCLR is a variational approximation of mutual information; other methods are conceptually similar, but do not directly approximate mutual information.

2-layer MLPs with 4096 hidden dimensions and 256 output dimensions. The outputs of the networks are batch-normalized and rescaled to unit norm before computing the loss. We use an inverse multiquadric kernel (IMQ) for the latent representation (approximated with 512 random Fourier features that are resampled at each step; see Sec. B.3.2.1 for details) and a linear kernel for labels. γ in (3.1) is set to 3. When training without a target network, unlike SimSiam [117], we do not stop gradients for either branch. If the target network is used, its weights are an exponential moving average of the online network weights. We employ the same schedule as BYOL [93], $\tau = 1 - 0.01 \cdot (\cos(\pi t/T) + 1)/2$ with t the current step and T the total training steps.

Image augmentation Our method uses the same data augmentation scheme as BYOL (see Appendix B.4.1). Briefly, we first draw a random patch from the original image and resize it to 224×224. Then, we apply a random horizontal flip, followed by color jittering, consisting of a random sequence of brightness, contrast, saturation, hue adjustments, and an optional grayscale conversion. Finally Gaussian blur and solarization are applied, and the view is normalized with ImageNet statistics.

Optimization We train the model with a batch size of 4096 on 128 Cloud

TPU v4 cores. Again, following [92, 93], we use the LARS optimizer [133] with a cosine decay learning rate schedule over 1000 epochs. The base learning rate to all of our experiments is 0.4 and it is scaled linearly [134] with the batch size $lr = 0.4 \times batch\_size/256$. All experiments use weight decay of 10^{-6}.

Learning kernel parameters We use a linear kernel for labels, since the type of kernel only scales (3.9). Our inverse multiquadric kernel for the latent Z has an additional kernel scale parameter. We optimize this along with all other parameters, but regularize it to maximize the entropy of the distribution $k_\sigma(s)$, where $s_{ij} = \|z_i - z_j\|^2$; this amounts to maximizing $\log\|k'_\sigma(s)\|^2$ (Sec. B.4.1.2).

3.4.2 Evaluation Results

Linear evaluation on ImageNet Learned features are evaluated with the standard linear evaluation protocol commonly used in evaluating self-supervised learning methods [29, 92, 93, 110–115]. Table 3.1 reports the top-1 and top-5 accuracies obtained with SSL-HSIC on ImageNet validation set, and compares to previous self-supervised learning methods. Without a target network, our method reaches 72.2% top-1 and 90.7% top-5 accuracies. Adding the target network, our method outperforms most previous methods, achieving top-1 accuracy of 74.8% and top-5 accuracy of 92.2%. The fact that we see performance gains from adopting a target network suggests that its effect is not yet well understood, although note discussion in [93] which points to its stabilizing effect.

Table 3.1: Linear evaluation on the ImageNet validation set.

	Top-1(%)	Top-5(%)
Supervised [135]	75.9	92.8
SimCLR [92]	69.3	89.0
MoCo v2 [113]	71.1	90.1
BYOL [93]	74.3	91.6
SwAV [114]	**75.3**	-
Barlow Twins [115]	73.2	91.0
SSL-HSIC (w/o target)	72.2	90.7
SSL-HSIC (w/ target)	74.8	**92.2**

Table 3.2: Fine-tuning on 1%, 10% and 100% of the ImageNet training set and evaluating on the validation set.

	Top-1(%)			Top-5(%)		
	1%	10%	100%	1%	10%	100%
Supervised [135]	25.4	56.4	75.9	48.4	80.4	92.8
SimCLR [92]	48.3	65.6	76.0	75.5	87.8	93.1
BYOL [93]	53.2	68.8	**77.7**	78.4	89.0	**93.9**
SwAV [114]	53.9	**70.2**	-	78.5	**89.9**	-
Barlow Twins [115]	**55.0**	69.7	-	**79.2**	89.3	-
SSL-HSIC (w/o target)	45.3	65.5	76.4	72.7	87.5	93.2
SSL-HSIC (w/ target)	52.1	67.9	77.2	77.7	88.6	93.6

Semi-supervised learning on ImageNet We fine-tune the network pretrained with SSL-HSIC on 1%, 10% and 100% of ImageNet, using the same ImageNet splits

as SimCLR [92]. Table 3.2 summarizes the semi-supervised learning performance. Our method, with or without a target network, has competitive performance in both data regimes. The target network has the most impact on the small-data regime, with 1% labels.

Table 3.3: Comparison of transfer learning performance on 12 image datasets. Supervised-IN is trained on ImageNet with supervised pretrainining. Random init trains on individual dataset with randomly initialized weights. MPCA refers to mean per-class accuracy; AP50 is average precision at IoU=0.5.

Dataset Metric	Birdsnap Top-1	Caltech101 MPCA	Cifar10 Top-1	Cifar100 Top-1	DTD Top-1	Aircraft MPCA	Food Top-1	Flowers MPCA	Pets MPCA	Cars Top-1	SUN397 Top-1	VOC2007 AP50
Linear:												
Supervised-IN [92]	53.7	**94.5**	**93.6**	78.3	74.9	**61.0**	72.3	94.7	**91.5**	**67.8**	**61.9**	82.8
SimCLR [92]	37.4	90.3	90.6	71.6	74.5	50.3	68.4	90.3	83.6	50.3	58.8	80.5
BYOL [93]	57.2	94.2	91.3	**78.4**	75.5	60.6	75.3	**96.1**	90.4	66.7	62.2	82.5
SSL-HSIC (w/o target)	50.6	92.3	91.5	75.9	75.3	57.9	73.6	95.0	88.2	59.3	61.0	81.4
SSL-HSIC (w/ target)	**57.8**	93.5	92.3	77.0	**76.2**	58.5	**75.6**	95.4	91.2	62.6	61.8	**83.3**
Fine-tuned:												
Supervised-IN [92]	75.8	93.3	97.5	**86.4**	74.6	86.0	88.3	**97.6**	**92.1**	**92.1**	**94.3**	85.0
Random init [92]	**76.1**	72.6	95.9	80.2	64.8	85.9	86.9	92.0	81.5	91.4	53.6	67.3
SimCLR [92]	75.9	92.1	97.7	85.9	73.2	88.1	88.2	97.0	89.2	91.3	63.5	84.1
BYOL [93]	76.3	**93.8**	**97.8**	86.1	**76.2**	88.1	**88.5**	97.0	91.7	91.6	63.7	**85.4**
SSL-HSIC (w/o target)	73.1	91.5	97.4	85.3	75.3	87.1	87.5	96.4	90.6	91.6	62.2	84.1
SSL-HSIC (w/ target)	74.9	**93.8**	**97.8**	84.7	75.4	**88.9**	87.7	97.3	91.7	91.8	61.7	84.1

Transfer to other classification tasks To investigate the generality of the representation learned with SSL-HSIC, we evaluate the transfer performance for classification on 12 natural image datasets [136–145] using the same procedure as [92, 93, 146]. Table 3.3 shows the top-1 accuracy of the linear evaluation and fine-tuning performance on the test set. SSL-HSIC gets state-of-the-art performance on 3 of the classification tasks and reaches strong performance on others for this benchmark, indicating the learned representations are robust for transfer learning.

Transfer to other vision tasks To test the ability of transferring to tasks other than classification, we fine-tune the network on semantic segmentation, depth estimation and object detection tasks. We use Pascal VOC2012 dataset [145] for semantic segmentation, NYU v2 dataset [147] for depth estimation and COCO [148] for object detection. Object detection outputs either bounding box or object segmentation (instance segmentation). Details of the evaluations setup is in Appendix B.4.2. Table 3.4 and Table 3.5 shows that SSL-HSIC achieves competitive performance on all three vision tasks.

Table 3.4: Fine-tuning performance on semantic segmentation and depth estimation. Mean Intersection over Union (mIoU) is reported for semantic segmentation. Relative error (rel), root mean squared error (rms), and the percent of pixels (pct) where the error is below 1.25^n thresholds are reported for depth estimation.

	VOC2012	NYU v2				
Method	mIoU	pct.<1.25	pct.$<1.25^2$	pct.$<1.25^3$	rms	rel
Supervised-IN	74.4	81.1	95.3	98.8	0.573	**0.127**
SimCLR	75.2	83.3	96.5	99.1	0.557	0.134
BYOL	**76.3**	**84.6**	96.7	99.1	0.541	0.129
SSL-HSIC(w/o target)	74.9	84.1	96.7	**99.2**	**0.539**	0.130
SSL-HSIC(w/ target)	76.0	83.8	**96.8**	99.1	0.548	0.130

Table 3.5: Fine-tuning performance on COCO object detection tasks. Precision, averaged over 10 IoU (Intersection over Union) thresholds, is reported for both bounding box and object segmentation.

Method	AP^{bb}	AP^{mk}
Supervised	39.6	35.6
SimCLR	39.7	35.8
MoCo v2	40.1	36.3
BYOL	**41.6**	37.2
SwAV	**41.6**	**37.8**
SSL-HSIC(w/o target)	40.5	36.3
SSL-HSIC(w/ target)	41.3	36.8

3.5 Ablation Studies

We present ablation studies to gain more intuition on SSL-HSIC. Here, we use a ResNet-50 backbone trained for 100 epochs on ImageNet, and evaluate with the linear protocol unless specified.

ResNet architectures In this ablation, we investigate the performance of SSL-HSIC with wider and deeper ResNet architecture. Figure 3.3 and Table 3.6 show our main results. The performance of SSL-HSIC gets better with larger networks. We used the supervised baseline from [93] which our training framework is based on ([92] reports lower performance). The performance gap between SSL-HSIC and the supervised baseline diminishes with larger architectures. In addition, Table 3.7 presents the semi-supervise learning results with subsets 1%, 10% and 100% of the ImageNet data.

Regularization term We compare performance of InfoNCE with SSL-HSIC in Table 3.8a since they can be seen as approximating the same $\mathrm{HSIC}(Z,Y)$ objective but with different forms of regularization. We reproduce the InfoNCE result in our codebase, using the same architecture and data augmentiation as for SSL-HSIC. Trained for 100 epochs (without a target network), InfoNCE achieves 66.0% top-1 and 86.9% top-5 accuracies, which is better than the result reported in [92]. For

Table 3.6: Top-1 and top-5 accuracies for different ResNet architectures using linear evaluation protocol.

	SSL-HSIC		BYOL[93]		Sup.[93]	
ResNet	Top1	Top5	Top1	Top5	Top1	Top5
50 (1x)	74.8	92.2	74.3	91.6	76.4	92.9
50 (2x)	77.9	94.0	77.4	93.6	79.9	95.0
50 (4x)	79.1	94.5	78.6	94.2	80.7	95.3
200 (2x)	79.6	94.8	79.6	94.9	80.1	95.2

Table 3.7: Top-1 and top-5 accuracies for different ResNet architectures using semi-supervised fine-tuning.

	Top1			Top5		
ResNet	1%	10%	100%	1%	10%	100%
50 (1x)	52.1	67.9	77.2	77.7	88.6	93.6
50 (2x)	61.2	72.6	79.3	83.8	91.2	94.7
50 (4x)	67.0	75.4	79.7	87.4	92.5	94.8
200(2x)	69.0	76.3	80.5	88.3	92.9	95.2

comparison, SSL-HSIC reaches 66.7% top-1 and 87.6% top-5 accuracies. This suggests that the regularization employed by SSL-HSIC is more effective.

Kernel type We investigate the effect of using different a kernel on latents Z. Training without a target network or random Fourier feature approximation, the top-1 accuracies for linear, Gaussian, and inverse multiquadric (IMQ) kernels are 65.27%, 66.67% and 66.72% respectively. Non-linear kernels indeed improve the performance; Gaussian and IMQ kernels reach very similar performance for 100 epochs. We choose IMQ kernel for longer runs, because its heavy-tail property can capture more signal when points are far apart.

Table 3.8: Linear evaluation results when varying different hyperparameters.

(a) Regularization

	Top-1	Top-5
SSL-HSIC	66.7	87.6
InfoNCE	66.0	86.9

(b) # Fourier features

# RFFs	Top-1(%)
64	66.0
128	66.2
256	66.2
512	66.4
1024	66.5
2048	66.5
No Approx.	66.7

(c) Batch size

	Top-1(%)	
Batch Size	SSL-HSIC	SimCLR
256	63.7	57.5
512	65.6	60.7
1024	66.7	62.8
2048	67.1	64.0
4096	66.7	64.6

(d) Projector/predictor size

Output Dim	Top-1(%)
64	65.4
128	66.0
256	66.4
512	66.6
1024	66.6

Number of RFF Features Table 3.8b shows the performance of SSL-HSIC with different numbers of Fourier features. The RFF approximation has a minor impact on the overall performance, as long as we resample them; fixed sets of features performed poorly. Our main result picked 512 features, for substantial computational savings with minor loss in accuracy.

Batch size Similar to most of the self-supervised learning methods [92, 93], SSL-HSIC benefits from using a larger batch size during training. However, the

drop of performance from using smaller batch size is not as pronounced as it is in SimCLR[92] as shown in Table 3.8c.

Projector and predictor output size Table 3.8d shows the performance when using different output dimension for the projector/predictor networks. The performance saturates at 512 dimensions.

3.6 Discussion

We introduced SSL-HSIC, a loss function for self-supervised representation learning based on kernel dependence maximisation. We provided a unified view on various self-supervised learning losses: we proved that InfoNCE, a lower bound of mutual information, actually approximates SSL-HSIC with a variance-based regularisation, and we can also interpret SSL-HSIC as metric learning where the cluster structure is imposed by the self-supervised label, of which the BYOL objective is a special case. We showed that training with SSL-HSIC achieves performance on par with the state-of-the-art on the standard self-supervised benchmarks.

Although using the image identity as a self-supervised label provides a good inductive bias, it might not be wholly satisfactory; we expect that some image pairs are in fact more similar than others, based e.g. on their ImageNet class label. It will be interesting to explore methods that combine label structure discovery with representation learning (as in SwAV [114]). In this paper, we only explored learning image representations, but in future work SSL-HSIC can be extended to learning structure for Y as well, building on existing work [37, 124].

Chapter 4

Biological implementation of weight sharing

This chapter is based on Pogodin et al. (2021); the theoretical and experimental results are my own, but obtained in collaboration with other authors.

Summary Convolutional networks are ubiquitous in deep learning. They are particularly useful for images, as they reduce the number of parameters, reduce training time, and increase accuracy. However, as a model of the brain they are seriously problematic, since they require weight sharing – something real neurons simply cannot do. Consequently, while neurons in the brain can be locally connected (one of the features of convolutional networks), they cannot be convolutional. Locally connected but non-convolutional networks, however, significantly underperform convolutional ones. This is troublesome for studies that use convolutional networks to explain activity in the visual system. Here we study alternatives to weight sharing that aim at the same regularisation principle, which is to make each neuron within a pool react similarly to identical inputs. The most natural way to do that is by showing the network multiple translations of the same image, akin to saccades or object manipulation in animal vision. However, this approach requires many translations, and doesn't remove the performance gap. We propose instead to add lateral connectivity to a locally connected network, and allow learning via Hebbian plasticity. This requires the network to pause occasionally for a sleep-like phase of "weight sharing". We also show that weight sharing based on lateral connection

and anti-Hebbian plasticity can be done simultaneously with training. This method enables locally connected networks to achieve nearly convolutional performance on ImageNet and improves their fit to the ventral stream data.

4.1 Introduction

Convolutional networks are a cornerstone of modern deep learning: they're widely used in the visual domain [150–152], speech recognition [153], text classification [154], and time series classification [155]. They have also played an important role in enhancing our understanding of the visual stream [156]. Indeed, simple and complex cells in the visual cortex [157] inspired convolutional and pooling layers in deep networks [158] (with simple cells implemented with convolution and complex ones with pooling). Moreover, the representations found in convolutional networks are similar to those in the visual stream [22, 159–161] (see [156] for an in-depth review).

Despite the success of convolutional networks at reproducing activity in the visual system, as a model of the visual system they are somewhat problematic. That's because convolutional networks share weights, something biological networks, for which weight updates must be local, can't do [162]. Locally connected networks avoid this problem by using the same receptive fields as convolutional networks (thus locally connected), but without weight sharing [21]. However, they pay a price for biological plausibility: locally connected networks are known to perform worse than their convolutional counterparts on hard image classification tasks [21, 163].

Here, we consider two mechanisms to bridge the gap between biologically plausible locally connected networks and implausible convolutional ones. One is to use extensive data augmentation (primarily image translations); the other is to introduce an auxiliary objective that allows some form of weight sharing, which is implemented by lateral connections; we call this approach dynamic weight sharing.

The first approach, data augmentation, is simple, but we show that it suffers from two problems: it requires far more training data than is normally used, and even then it fails to close the performance gap between convolutional and locally connected networks. The second approach, dynamic weight sharing, implements a

sleep-like phase in which neural dynamics facilitate weight sharing. This is done through lateral connections in each layer, which allows subgroups of neurons to share their activity. Through this lateral connectivity, each subgroup can first equalise its weights via anti-Hebbian learning, and then generate an input pattern for the next layer that helps it to do the same thing. Dynamic weight sharing doesn't achieve perfectly convolutional connectivity, because in each channel only subgroups of neurons share weights. However, it implements a similar inductive bias, and, as we show in experiments, it performs almost as well as convolutional networks, and also achieves better fit to the ventral stream data, as measured by the Brain-Score [22, 164].

Our study suggests that weight sharing can be implemented with Hebbian learning. While our solution does not fully resolve biological implausibility of convolutional networks as a model of the visual stream, it shows that neurons can exchange weight information with simple connectivity and plasticity rules.

4.2 Related work

Studying systems neuroscience through the lens of deep learning is an active area of research, especially when it comes to the visual system [45]. As mentioned above, convolutional networks in particular have been extensively studied as a model of the visual stream (and also inspired by it) [156], and also as mentioned above, because they require weight sharing they lack biological plausibility. They have also been widely used to evaluate the performance of different biologically plausible learning rules [21, 26, 41, 46, 60, 61, 67, 68].

Several studies have tried to relax weight sharing in convolutions by introducing locally connected networks [21, 163, 165] ([165] also shows that local connectivity itself can be learned from a fully connected network with proper weight regularisation). Locally connected networks perform as well as convolutional ones in shallow architectures [21, 165]. However, they perform worse for large networks and hard tasks, unless they're initialised from an already well-performing convolutional solution [163] or have some degree of weight sharing [166]. In this study, we seek

regularizations of locally connected networks through either data or local plasticity to improve their performance.

Convolutional networks are not the only deep learning architecture for vision: visual transformers (e.g., [167–170]), and more recently, the transformer-like architectures without self-attention [171–173], have shown competitive results. However, they still need weight sharing: at each block the input image is reshaped into patches, and then the same weight is used for all patches. Our Hebbian-based approach to weight sharing fits this computation as well (see Appendix C.1.4).

4.3 Regularisation in locally connected networks

4.3.1 Convolutional versus locally connected networks

Convolutional networks are implemented by letting the weights depend on the difference in indices. Consider, for simplicity, one dimensional convolutions and a linear network. Letting the input and output of a one layer in a network be x_j and z_i, respectively, the activity in a convolutional network is

$$z_i = \sum_{j=1}^{N} w_{i-j} x_j \,, \tag{4.1}$$

where N is the number of neurons; for definiteness, we'll assume N is the same in each layer (right panel in Fig. 4.1). Although the index j ranges over all N neurons, many, if not most, of the weights are zero: w_{i-j} is nonzero only when $|i-j| \leq k/2 < N$ for kernel size k.

For networks that aren't convolutional, the weight matrix w_{i-j} is replaced by w_{ij},

$$z_i = \sum_{j=1}^{N} w_{ij} x_j \,. \tag{4.2}$$

Again, the index j ranges over all N neurons. If all the weights are nonzero, the network is fully connected (left panel in Fig. 4.1). But, as in convolutional networks, we can restrict the connectivity range by letting w_{ij} be nonzero only when $|i-j| \leq k/2 < N$, resulting in a *locally connected*, but non-convolutional, network (centre panel in Fig. 4.1).

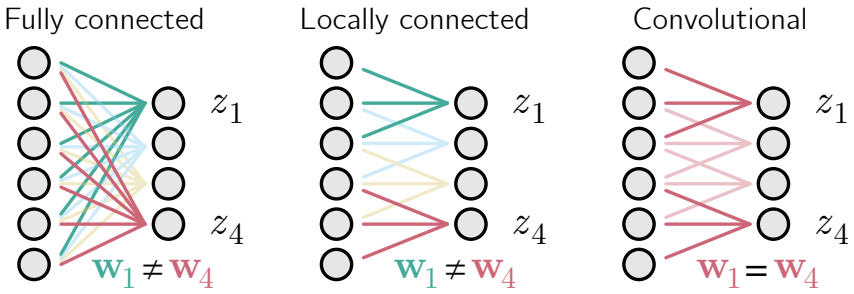

Figure 4.1: Comparison between layer architectures: fully connected (left), locally connected (middle) and convolutional (right). Locally connected layers have different weights for each neuron z_1 to z_4 (indicated by different colours), but have the same connectivity as convolutional layers.

4.3.2 Developing convolutional weights: data augmentation versus dynamic weight sharing

Here we explore the question: is it possible for a locally connected network to develop approximately convolutional weights? That is, after training, is it possible to have $w_{ij} \approx w_{i-j}$? There is one straightforward way to do this: augment the data to provide multiple translations of the same image, so that each neuron within a channel learns to react similarly (Fig. 4.2A). A potential problem is that a large number of translations will be needed. This makes training costly (see Section 4.5), and is unlikely to be consistent with animal learning, as animals see only a handful of translations of any one image.

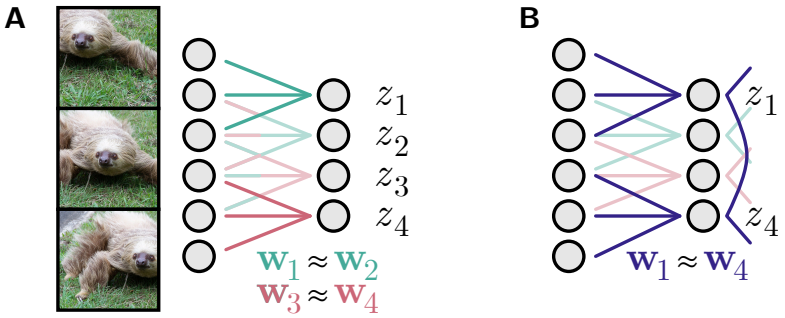

Figure 4.2: Two regularisation strategies for locally connected networks. **A.** Data augmentation, where multiple translations of the same image are presented simultaneously. **B.** Dynamic weight sharing, where a subset of neurons equalises their weights through lateral connections and learning.

A less obvious solution is to modify the network so that during learning the

weights become approximately convolutional. As we show in the next section, this can be done by adding lateral connections, and introducing a *sleep phase* during training (Fig. 4.2B). This solution doesn't need more data, but it does need an additional training step.

4.4 A Hebbian solution to dynamic weight sharing

If we were to train a locally connected network without any weight sharing or data augmentation, the weights of different neurons would diverge (region marked "training" in Fig. 4.3A). Our strategy to make them convolutional is to introduce an occasional sleep phase, during which the weights relax to their mean over output neurons (region marked "sleep" in Fig. 4.3A). This will compensate for weight divergence during learning by convergence during the sleep phase. If the latter is sufficiently strong, the weights will remain approximately convolutional.

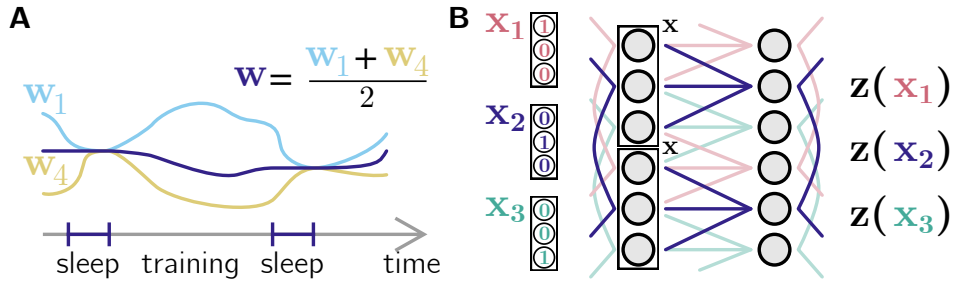

Figure 4.3: **A.** Dynamical weight sharing interrupts the main training loop, and equalises the weights through internal dynamics. After that, the weights diverge again until the next weight sharing phase. **B.** A locally connected network, where both the input and the output neurons have lateral connections. The input layer uses lateral connections to generate repeated patterns for weight sharing in the output layer. For instance, the output neurons connected by the dark blue lateral connections (middle) can receive three different patterns: $\mathbf{x}_1$ (generated by the red input grid), $\mathbf{x}_2$ (dark blue) and $\mathbf{x}_3$ (green).

To implement this, we introduce fixed (non-plastic) lateral connectivity, chosen to equalise activity in both the input and output layer. That's shown in Fig. 4.3B, where every third neuron in both the input ($\mathbf{x}$) and output ($\mathbf{z}$) layers are connected (and also shown in Fig. 4.4A). Once the connected neurons in the input layer have equal activity, all output neurons receive identical input. Since the lateral output connections also equalise activity, all connections that correspond to a translation by

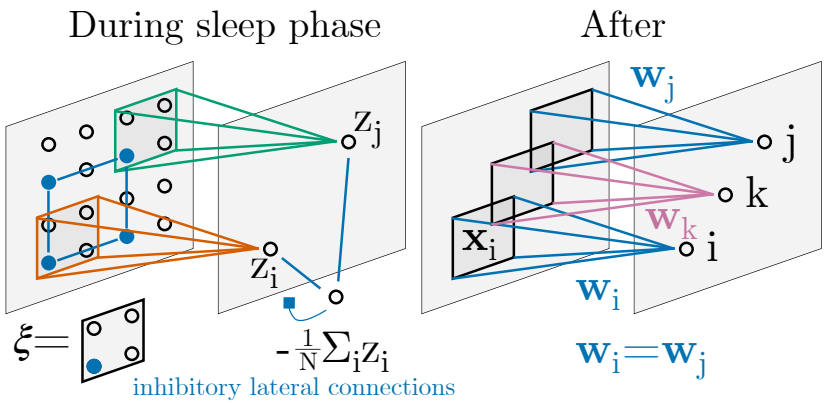

Figure 4.4: Alternative visual explanation of weight sharing with a sleep phase.

three neurons see exactly the same pre and postsynaptic activity. A naive Hebbian learning rule (with weight decay) would, therefore, make the network convolutional (Fig. 4.4B). However, we have to take care that the initial weights are not overwritten during Hebbian learning. We now describe how that is done.

To ease notation, we'll let $\mathbf{w}_i$ be a vector containing the incoming weights to the neuron i: $(\mathbf{w}_i)_j \equiv w_{ij}$. Moreover, we'll let j run from 1 to k, independent of i. With this convention, the response of neuron i, z_i, to a k-dimensional input, $\mathbf{x}$, is given by

$$z_i = \mathbf{w}_i^\top \mathbf{x} = \sum_{j=1}^{k} w_{ij} x_j \,. \tag{4.3}$$

Assume that every neuron sees the same $\mathbf{x}$, and consider the following update rule for the weights,

$$\Delta \mathbf{w}_i \propto -\left(z_i - \frac{1}{N} \sum_{j=1}^{N} z_j \right) \mathbf{x} - \gamma \left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}} \right) , \tag{4.4}$$

where $\mathbf{w}_i^{\text{init}}$ are the weights at the beginning of the sleep phase (not the overall training).

This Hebbian update effectively implements SGD over the sum of $(z_i - z_j)^2$, plus a regularizer (the second term) to keep the weights near $\mathbf{w}_i^{\text{init}}$. If we present the network with M different input vectors, $\mathbf{x}_m$, and denote the covariance matrix

$\mathbf{C} \equiv \frac{1}{M}\sum_m \mathbf{x}_m \mathbf{x}_m^\top$, then, asymptotically, the weight dynamics in Eq. (4.4) converges to

$$\mathbf{w}_i^* = (\mathbf{C}+\gamma\mathbf{I})^{-1}\left(\mathbf{C}\frac{1}{N}\sum_{j=1}^{N}\mathbf{w}_j^{\text{init}} + \gamma\mathbf{w}_i^{\text{init}}\right) \tag{4.5}$$

where $\mathbf{I}$ is the identity matrix (see Appendix C.1 for a derivation).

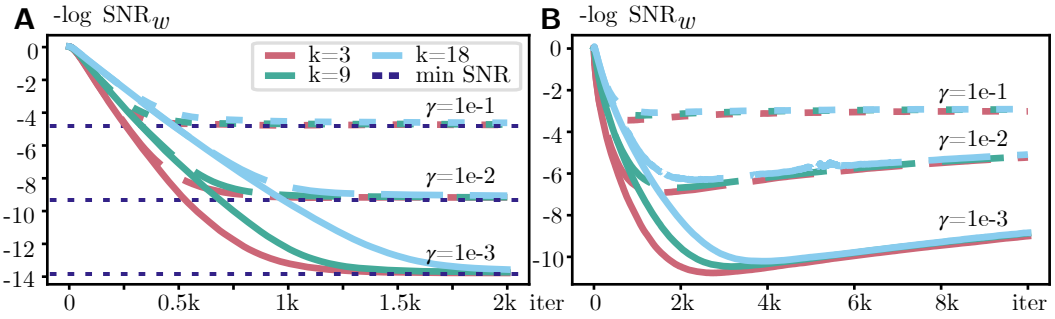

Figure 4.5: Negative logarithm of signal-to-noise ratio (mean weight squared over weight variance, see Eq. (4.6)) for weight sharing objectives in a layer with 100 neurons. Different curves have different kernel size, k (meaning k^2 inputs), and regularisation parameter, γ. **A.** Weight updates given by Eq. (4.4). Black dashed lines show the theoretical minimum. **B.** Weight updates given by Eq. (4.8), with $\alpha = 10$. In each iteration, the input is presented for 150 ms.

As long as $\mathbf{C}$ is full rank and γ is small, we arrive at shared weights: $\mathbf{w}_i^* \approx \frac{1}{N}\sum_{i=1}^{N}\mathbf{w}_i^{\text{init}}$. It might seem advantageous to set $\gamma = 0$, as non-zero γ only biases the equilibrium value of the weight. However, non-zero γ ensures that for noisy input, $\mathbf{x}_i = \mathbf{x} + \xi_i$ (such that the inputs to different neurons are the same only on average, which is much more realistic), the weights still converge (at least approximately) to the mean of the initial weights (see Theorem 6 in Appendix C.1).

In practice, the dynamics in Eq. (4.4) converges quickly. We illustrate it in Fig. 4.5A by plotting $-\log \text{SNR}_w$ over time, where SNR_w, the signal to noise ratio of the weights, is defined as

$$\text{SNR}_w = \frac{1}{k^2}\sum_j \frac{\left(\frac{1}{N}\sum_i (\mathbf{w}_i)_j\right)^2}{\frac{1}{N}\sum_i \left((\mathbf{w}_i)_j - \frac{1}{N}\sum_{i'}(\mathbf{w}_{i'})_j\right)^2}. \tag{4.6}$$

For all kernel sizes (we used 2d inputs, meaning k^2 inputs per neuron), the weights converge to a nearly convolutional solution within a few hundred iterations (note

the logarithmic scale of the y axis in Fig. 4.5A). See Appendix C.1.5 for simulation details. Thus, to run our experiments with deep networks in a realistic time frame, we perform weight sharing instantly (i.e., directly setting them to the mean value) during the sleep phase.

4.4.1 Dynamic weight sharing in multiple locally connected layers

As shown in Fig. 4.3B, the k-dimensional input, $\mathbf{x}$, repeats every k neurons. Consequently, during the sleep phase, the weights are not set to the mean of their initial value averaged across all neurons; instead, they're set to the mean averaged across a set of neurons spaced by k. Thus, in one dimension, the sleep phase equilibrates the weights in k different modules. In two dimensions (the realistic case), the sleep phase equilibrates the weights in k^2 different modules.

We need this spacing to span the whole k-dimensional (or k^2 for 2d) space of inputs. For instance, activating the red grid on the left in Fig. 4.3B generates $\mathbf{x}_1$, covering one input direction for all output neurons (and within each module, every neuron receives the same input). Next, activating the blue grid generates $\mathbf{x}_2$ (a new direction), and so on.

In multiple layers, the sleep phase is implemented layer by layer. In layer l, lateral connectivity creates repeated input patterns and feeds them to layer $l+1$. After weight sharing in layer $l+1$, the new pattern from $l+1$ is fed to $l+2$, and so on. Notably, there's no layer by layer plasticity schedule (i.e., deeper layers don't have to wait for the earlier ones to finish), as the weight decay term in Eq. (4.4) ensures the final solution is the same regardless of intermediate weight updates. As long as a deeper layer starts receiving repeated patterns, it will eventually arrive at the correct solution.

4.4.2 A realistic model that implements the update rule

Our update rule, Eq. (4.4), implies that there is a linear neuron, denoted r_i, whose activity depends on the upstream input, $z_i = \mathbf{w}_i^\top \mathbf{x}$, via a direct excitatory connection

combined with lateral inhibition,

$$r_i = z_i - \frac{1}{N}\sum_{j=1}^{N} z_j \equiv \mathbf{w}_i^\top \mathbf{x} - \frac{1}{N}\sum_{j=1}^{N} \mathbf{w}_j^\top \mathbf{x}. \tag{4.7}$$

The resulting update rule is anti-Hebbian, $-r_i\mathbf{x}$ (see Eq. (4.4)). In a realistic circuit, this can be implemented with excitatory neurons r_i and an inhibitory neuron r_{inh}, which obey the dynamics

$$\tau \dot{r}_i = -r_i + \mathbf{w}_i^\top \mathbf{x} - \alpha\, r_{\text{inh}} + b\,, \tag{4.8a}$$

$$\tau \dot{r}_{\text{inh}} = -r_{\text{inh}} + \frac{1}{N}\sum_j r_j - b\,, \tag{4.8b}$$

where b is the shared bias term that ensures non-negativity of firing rates (assuming $\sum_i \mathbf{w}_i^\top \mathbf{x}$ is positive, which would be the case for excitatory input neurons). The only fixed point of these equations is[1]

$$r_i^* = b + \mathbf{w}_i \cdot \mathbf{x} - \frac{1}{N}\sum_j \mathbf{w}_j \cdot \mathbf{x} + \frac{1}{1+\alpha}\frac{1}{N}\sum_j \mathbf{w}_j \cdot \mathbf{x} \underset{\alpha \gg 1}{\approx} b + \mathbf{w}_i \cdot \mathbf{x} - \frac{1}{N}\sum_j \mathbf{w}_j \cdot \mathbf{x}, \tag{4.9}$$

which is stable. As a result, for strong inhibition ($\alpha \gg 1$), Eq. (4.4) can be implemented with an anti-Hebbian term $-(r_i - b)\mathbf{x}$. Note that if $\mathbf{w}_i^\top \mathbf{x}$ is zero on average, then b is the mean firing rate over time. To show that Eq. (4.9) provides enough signal, we simulated training in a network of 100 neurons that receives a new $\mathbf{x}$ each 150 ms. For a range of k and γ, it converged to a nearly convolutions solution within minutes (Fig. 4.5B; each iteration is 150 ms). Having finite inhibition did lead to a worse final signal-to-noise ratio ($\alpha = 10$ in Fig. 4.5B), but the variance of the weights was still very small. Moreover, the nature of the α-induced bias suggests that stopping training before convergence leads to better results (around 2k iterations in Fig. 4.5B). See Fig. C.1 in Appendix C.1 for a discussion.

[1] As promised in the introduction, one equation uses $\cdot$ to denote $^\top$. I still disagree with the dot notation.

4.5 Experiments

We split our experiments into two parts: small-scale ones with CIFAR10, CIFAR100 [59] and TinyImageNet [174], and large-scale ones with ImageNet [94]. The former illustrates the effects of data augmentation and dynamic weight sharing on the performance of locally connected networks; the latter concentrates on dynamic weight sharing, as extensive data augmentations are too computationally expensive for large networks and datasets. We used the AdamW [89] optimizer in all runs. As our dynamic weight sharing procedure always converges to a nearly convolutional solution (see Section 4.4), we set the weights to the mean directly (within each grid) to speed up experiments. Our code is available at https://github.com/romanpogodin/towards-bio-plausible-conv (PyTorch [175] implementation).

Datasets. CIFAR10 consists of 50k training and 10k test images of size 32×32, divided into 10 classes. CIFAR100 has the same structure, but with 100 classes. For both, we tune hyperparameters with a 45k/5k train/validation split, and train final networks on the full 50k training set. TinyImageNet consists of 100k training and 10k validation images of size 64×64, divided into 200 classes. As the test labels are not publicly available, we divided the training set into 90k/10k train/validation split, and used the 10k official validation set as test data. ImageNet consists of 1.281 million training images and 50k test images of different sizes, reshaped to 256 pixels in the smallest dimension. As in the case for TinyImageNet, we used the train set for a 1.271 million/10k train/validation split, and 50k official validation set as test data.

Networks. For CIFAR10/100 and TinyImageNet, we used CIFAR10-adapted ResNet20 from the original ResNet paper [152]. The network has three blocks, each consisting of 6 layers, with 16/32/64 channels within the block. We chose this network due to good performance on CIFAR10, and the ability to fit the corresponding locally connected network into the 8G VRAM of the GPU for large batch sizes on all three datasets. For ImageNet, we took the half-width ResNet18 (meaning 32/64/128/256 block widths) to be able to fit a common architecture (albeit halved in width) in the locally connected regime into 16G of GPU VRAM. For both networks, all layers had a 3×3 receptive field (apart from a few 1×1 residual downsampling

layers), meaning that weight sharing worked over 9 individual grids in each layer.

Training details. We ran the experiments on our local laboratory cluster, which consists mostly of NVIDIA GTX1080 and RTX5000 GPUs. The small-scale experiments took from 1-2 hours per run up to 40 hours (for TinyImageNet with 16 repetitions). The large-scale experiments took from 3 to 6 days on RTX5000 (the longest run was the locally connected network with weight sharing happening after every minibatch update).

4.5.1 Data augmentations.

For CIFAR10/100, we padded the images (padding size depended on the experiment) with mean values over the training set (such that after normalisation the padded values were zero) and cropped to size 32×32. We did not use other augmentations to separate the influence of padding/random crops. For TinyImageNet, we first centre-cropped the original images to size $(48 + 2\,\text{pad}) \times (48 + 2\,\text{pad})$ for the chosen padding size pad. The final images were then randomly cropped to 48×48. This was done to simulate the effect of padding on the number of available translations, and to compare performance across different padding values on the images of the same size (and therefore locally connected networks of the same size). After cropping, the images were normalised using ImageNet normalisation values. For all three datasets, test data was sampled without padding. For ImageNet, we used the standard augmentations. Training data was resized to 256 (smallest dimension), randomly cropped to 224×224, flipped horizontally with 0.5 probability, and then normalised. Test data was resized to 256, centre cropped to 224 and then normalised. In all cases, data repetitions included multiple samples of the same image within a batch, keeping the total number of images in a batch fixed (e.g. for batch size 256 and 16 repetitions, that would mean 16 original images)

4.5.2 CIFAR10/100 and TinyImageNet

To study the effect of both data augmentation and weight sharing on performance, we ran experiments with non-augmented images (padding 0) and with different amounts of augmentations. This included padding of 4 and 8, and repetitions of 4, 8, and 16.

Table 4.1: Performance of convolutional (conv) and locally connected (LC) networks for padding of 4 in the input images (mean accuracy over 5 runs). For LC, two regularisation strategies were applied: repeating the same image n times with different translations (n reps) or using dynamic weight sharing every n batches (ws(n)). LC nets additionally show performance difference w.r.t. conv nets.

Regulariser	Connectivity	CIFAR10		CIFAR100				TinyImageNet			
		Top-1 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
-	conv	88.3	-	59.2	-	84.9	-	38.6	-	65.1	-
	LC	80.9	-7.4	49.8	-9.4	75.5	-9.4	29.6	-9.0	52.7	-12.4
Data Translation	LC - 4 reps	82.9	-5.4	52.1	-7.1	76.4	-8.5	31.9	-6.7	54.9	-10.2
	LC - 8 reps	83.8	-4.5	54.3	-5.0	77.9	-7.0	33.0	-5.6	55.6	-9.5
	LC - 16 reps	85.0	-3.3	55.9	-3.3	78.8	-6.1	34.0	-4.6	56.2	-8.8
Weight Sharing	LC - ws(1)	87.4	-0.8	58.7	-0.5	83.4	-1.6	41.6	3.0	66.1	1.1
	LC - ws(10)	85.1	-3.2	55.7	-3.6	80.9	-4.0	37.4	-1.2	61.8	-3.2
	LC - ws(100)	82.0	-6.3	52.8	-6.4	80.1	-4.8	37.1	-1.5	62.8	-2.3

Without augmentations, locally connected networks performed much worse than convolutional, although weight sharing improved the result a little bit (see Table C.1). For padding of 4 (mean accuracy over 5 runs Table 4.1, see Table C.5 for max-min accuracy), increasing the number of repetitions increased the performance of locally connected networks. However, even for 16 repetitions, the improvements were small compared to weight sharing (especially for top-5 accuracy on TinyImageNet). For CIFAR10, our results are consistent with an earlier study of data augmentations in locally connected networks [176]. For dynamic weight sharing, doing it moderately often – every 10 iterations, meaning every 5120 images – did as well as 16 repetitions on CIFAR10/100. For TinyImageNet, sharing weights every 100 iterations (about every 50k images) performed much better than data augmentation.

Sharing weights after every batch performed almost as well as convolutions (and even a bit better on TinyImageNet, although the difference is small if we look at top-5 accuracy, which is a less volatile metric for 200 classes), but it is too frequent to be a plausible sleep phase. We include it to show that best possible performance of partial weight sharing is comparable to actual convolutions.

For a padding of 8, the performance did improve for all methods (including convolutions), but the relative differences had a similar trend as for a padding of 4 (see Table C.3). We also trained locally connected networks with one repetition, but for longer and with a much smaller learning rate to simulate the effect of data repetitions. Even for 4x-8x longer runs, the networks barely matched the performance

Table 4.2: Performance of convolutional (conv) and locally connected (LC) networks on ImageNet for 0.5x width ResNet18 (1 run). For LC, we also used dynamic weight sharing every *n* batches. LC nets additionally show performance difference w.r.t. the conv net.

Connectivity	Weight sharing frequency	ImageNet			
		Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
conv	-	63.5	-	84.7	-
LC	-	46.7	-16.8	70.0	-14.7
LC	1	61.7	-1.8	83.1	-1.6
LC	10	59.3	-4.2	81.1	-3.6
LC	100	54.5	-9.0	77.7	-7.0

of a 1-repetition network on standard speed (not shown).

4.5.3 ImageNet

On ImageNet, we did not test image repetitions due to the computational requirements (e.g., running 16 repetitions with our resources would take almost 3 months). We used the standard data augmentation, meaning that all networks see different crops of the same image throughout training.

Our results are shown in Table 4.4. Weight sharing every 1 and 10 iterations (256/2560 images, respectively, for the batch size of 256) achieves nearly convolutional performance, although less frequent weight sharing results in a more significant performance drop. In contrast, the purely locally connected network has a large performance gap with respect to the convolutional one. It is worth noting that the trade-off between weight sharing frequency and performance depends on the learning rate, as weights diverge less for smaller learning rates. It should be possible to decrease the learning rate and increase the number of training epochs, and achieve comparable results with less frequent weight sharing.

4.5.4 Brain-Score of ImageNet-trained networks

In addition to ImageNet performance, we evaluated how well representations built by our networks correspond to ventral stream data in primates. For that we used the Brain-Score [22, 164], a set of metrics that evaluate deep networks' correspondence to neural recordings of cortical areas V1 [177, 178], V2 [177], V4 [179], and inferior temporal (IT) cortex [179] in primates, as well as behavioural data [180]. The

Table 4.3: Brain-Score of ImageNet-trained convolutional (conv) and locally connected (LC) networks on ImageNet for 0.5x width ResNet18 (higher is better). For LC, we also used dynamic weight sharing every n batches. *The models were evaluated on Brain-Score benchmarks available during submission. If new benchmarks are added and the models are re-evaluated on them, the final scores might change; the provided links contain the latest results.

Connectivity	Weight sharing frequency	Brain-Score						
		average score	V1	V2	V4	IT	behaviour	link*
conv	-	.357	.493	.313	.459	.370	.148	brain-score.org/model/876
LC	-	.349	**.542**	.291	.448	.354	.108	brain-score.org/model/877
LC	1	**.396**	.512	**.339**	.468	**.406**	**.255**	brain-score.org/model/880
LC	10	.385	.508	.322	**.478**	.399	.216	brain-score.org/model/878
LC	100	.351	.523	.293	.467	.370	.101	brain-score.org/model/879

advantage of Brain-Score is that it provides a standardised benchmark that looks at the whole ventral stream, and not only its isolated properties like translation invariance (which many early models focused on [181–184]). We do not directly check for translation invariance in our models (only through V1/V2 data). However, as our approach achieves convolutional solutions (see above), we trivially have translation equivariance after training: translating the input will translate the layer's response (in our case, each k-th translation will produce the same translated response for a kernel of size k). (In fact, it's pooling layers, not convolutions, that achieve some degree of invariance in convolutional networks – this is an architectural choice and is somewhat tangential to our problem.)

Our results are shown in Table C.11 (higher is better; see http://www.brain-score.org/ for the scores of other models). The well-performing locally connected networks (with weight sharing/sleep phase every 1 or 10 iterations) show overall better fit compared to their fully convolutional counterpart. Interestingly, the worst-performing purely locally connected network had the second best V1 fit, despite overall poor performance.

In general, Brain-Score correlates with ImageNet performance (Fig. 2 in [22]). This means that the worse Brain-Score performance of the standard locally connected network and the one with weight sharing every 100 iterations can be related to their poor ImageNet performance (Table 4.4). (It also means that our method can potentially increase Brain-Score performance of larger models.)

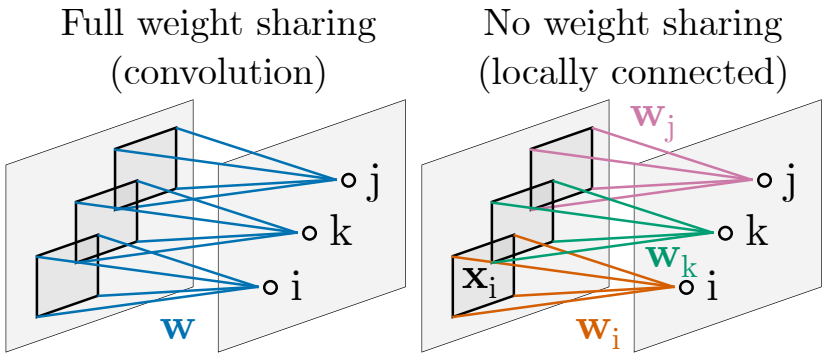

Figure 4.6: Examples of a convolutional layer (**A**) and a locally connected layer (**B**).

4.6 Sharing weights with noise-cancelling anti-Hebbian plasticity

4.6.1 Introduction

In a convolutional layer, all neurons within one channel have the same weight $\mathbf{w}$, but different location-specific inputs $\mathbf{x}_i$ (Fig. 4.6A), so the output is $z_i = \mathbf{w}^\top \mathbf{x}_i$, and during training the weights remain convolutional (Fig. 4.7A). In a locally connected layer (Fig. 4.6B), the weights are also neuron-specific: $z_i = \mathbf{w}_i^\top \mathbf{x}_i$. Even if the weights start at the same value, they diverge during training, which can be compensated with a sleep phase (Fig. 4.7B).

Previously, we discussed **solution B** (Fig. 4.7B), or the sleep phase from Pogodin et al. (2021): optimise the given loss $L(\mathbf{x}, y, \mathbf{w}_1, \ldots, \mathbf{w}_j)$ for several iterations, then stop and share weights with anti-Hebbian plasticity ("sleep phase").

This approach has several limitations: it can only share weights for neurons with non-overlapping inputs, it needs precise grid-like connectivity, it depends on the frequency of sleep phases, and local connections have to have the same size for all neurons in a layer.

Here we develop an algorithm that keeps the weights close to convolutional throughout training (Fig. 4.7C). This removes the dependence of network's performance on the frequency of the sleep phase, and may improve generalisation (w.r.t. the sleep phase solution), as individual weights never overfit when not "sleeping".

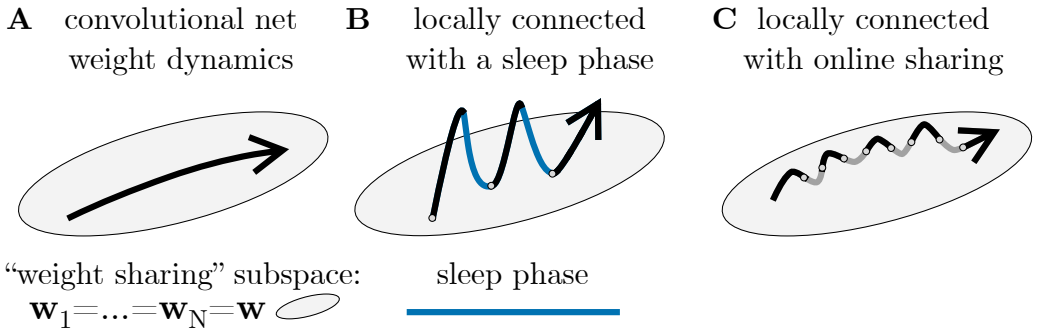

Figure 4.7: Weight dynamics for different layer types. The grey zone indicates a subspace with shared weights. **A.** Convolutional weights are always shared. **B.** Locally connected weights diverge from the shared solution, but converge back during the sleep phase. **C.** Locally connected weights stay near the shared solution with online weight sharing.

4.6.2 Proposed update rule

To derive the new update rule, we start by adding a weight sharing penalty to an arbitrary loss function L:

$$\tilde{L} \equiv L(\mathbf{x}, y, \mathbf{w}_1, \ldots, \mathbf{w}_N) + \frac{\gamma}{2N} \sum_{ij} \|\mathbf{w}_i - \mathbf{w}_j\|^2 . \tag{4.10}$$

The gradient of this new loss function is straightforward to compute:

$$\frac{\partial \tilde{L}(\ldots)}{\partial \mathbf{w}_i} = \frac{\partial L(\ldots)}{\partial \mathbf{w}_i} + \frac{\gamma}{N} \sum_j (\mathbf{w}_i - \mathbf{w}_j) . \tag{4.11}$$

This is not yet implementable in real neurons, as each weight $\mathbf{w}_i$ needs information about other weights. However, we can approximate it with a noise injection $\boldsymbol{\xi}$ into the input neurons, such that the output ones receive a random projection of the input weight, $\boldsymbol{\xi}^\top \mathbf{w}_i$.

For a zero-mean vector $\boldsymbol{\xi}$ with $\mathbb{E}\boldsymbol{\xi}\boldsymbol{\xi}^\top = I$, we can approximate the gradient as

$$\widehat{\frac{\partial \tilde{L}(\ldots)}{\partial \mathbf{w}_i}} \equiv \frac{\partial L(\ldots)}{\partial \mathbf{w}_i} + \gamma \boldsymbol{\xi}\boldsymbol{\xi}^\top \left(\mathbf{w}_i - \frac{1}{N} \sum_j \mathbf{w}_j \right) . \tag{4.12}$$

This gradient equals $\frac{\partial \tilde{L}(\ldots)}{\partial \mathbf{w}_i}$ in expectation.

To finally implement this update rule in a network, we again use lateral connec-

tivity to share noise $\boldsymbol{\xi}$ among neurons, but not input $\mathbf{x}_i$. We can do it by assuming that data-dependent activity $\mathbf{x}_i$ is transient and therefore filtered out by the lateral connectivity, while the slowly changing noise is not. Therefore, the firing rate at neuron z_i becomes

$$z_i = \mathbf{w}_i^\top (\mathbf{x}_i + \boldsymbol{\xi}) - \frac{1}{N} \sum_j \mathbf{w}_j^\top \boldsymbol{\xi} \,. \tag{4.13}$$

The new estimate of the gradient now contains a Hebbian term,

$$\widehat{\frac{\partial \tilde{L}(\dots)}{\partial \mathbf{w}_i}} \equiv \frac{\partial L(\dots)}{\partial \mathbf{w}_i} + \gamma z_i \boldsymbol{\xi} \,, \tag{4.14}$$

which is still equal to the correct gradient in expectation. It is worth noting that the presynaptic part of the Hebbian component is temporally averaged to contain only the slow noise $\boldsymbol{\xi}$, while the post-synaptic component contains the standard firing rate.

The overall rule can be interpreted as noise cancellation, as weight sharing dynamics learns to remove the $\boldsymbol{\xi}$ component from z_i ($z_i = \mathbf{w}_i^\top \mathbf{x}_i$ for shared weights). However, this rule has the same connectivity restrictions as the sleep phase method.

4.6.3 Proposed update rule with mean weight constraints

We can further simplify lateral connectivity used in this model by relaxing the weight sharing constraint to the mean sharing constraint:

$$\tilde{L}_{\text{mean}} \equiv L(\mathbf{x}, y, \mathbf{w}_1, \dots, \mathbf{w}_N) + \frac{\gamma}{2N} \sum_{ij} (\mathbf{1}^\top \mathbf{w}_i - \mathbf{1}^\top \mathbf{w}_j)^2 \,. \tag{4.15}$$

The gradient of this objective is the following:

$$\frac{\partial \tilde{L}_{\text{mean}}(\dots)}{\partial \mathbf{w}_i} = \frac{\partial L(\dots)}{\partial \mathbf{w}_i} + \frac{\gamma}{N} \sum_j \mathbf{1} (\mathbf{1}^\top \mathbf{w}_i - \mathbf{1}^\top \mathbf{w}_j) \,. \tag{4.16}$$

Now it can be approximated with a single zero-mean **number** ξ, rather than a

vector, as

$$z_i = \mathbf{w}_i^\top (\mathbf{x}_i + \xi\,\mathbf{1}) - \xi \frac{1}{N} \sum_j \mathbf{w}_j^\top \mathbf{1}\,, \tag{4.17}$$

$$\frac{\partial \widehat{\tilde{L}_{\text{mean}}(\dots)}}{\partial\,\mathbf{w}_i} = \frac{\partial\, L(\dots)}{\partial\,\mathbf{w}_i} + \gamma \xi\,\mathbf{z}_i\,\mathbf{1}\,. \tag{4.18}$$

We still need a slow noise process to distinguish the weight sharing signal from regular inputs, but the lateral connections do not need to form a greed. Instead, they need to connect all neurons in a channel. This model is much simpler, although provides less regularisation.

4.6.4 Choice of network architectures.

In [149], partial weight sharing was done in every layer, requiring precise grids of lateral connectivity to work. The grids were needed to provide non-overlapping inputs to neurons for correct weight sharing. In 1x1 convolutions (i.e., with connections to only one neuron in each input channel; Fig. 4.6C), each grid contains the whole channel, massively simplifying lateral connectivity. Many large popular architectures, such as ResNet34, ResNet50, MobileNetV3 [185], contain blocks of [1x1, NxN, 1x1] convolutions. The same pattern is used in a relatively shallow, but recurrent CORnet-S [20] (which is proposed as a more anatomically realistic neural network). Those models usually show a better fit to brain data [23]. Here, we use MobileNetV3-small due to its small size, as removing weight sharing greatly increases GPU memory requirements. We show that maintaining weight sharing only in 1x1 layers is enough, as long as the initialisation is convolutional.

The benefit of weight sharing in 1x1 layers only is the simplicity of implementation: the mean rule Eq. (4.18) is equivalent to Eq. (4.14).

4.6.5 ImageNet performance.

Performance is summarised in Table 4.4. Locally connected networks always perform better with convolutional initialisation than with random initialisation. Making 1x1 layers convolutional greatly improves performance, and our weight sharing method (Eq. (4.18)) approximates that behaviour; both approaches also lead to a better fit

to the ventral stream data (measured by the Brain-Score [23]). Layers that use Eq. (4.18) stay nearly convolutional (weight standard deviation $\ll |\text{mean}|$); for all other locally connected layers, even with a convolutional start, the weights quickly diverge to a non-convolutional state.

accuracy, %	conv	LC	LC + conv init	LC + conv 1x1	LC + conv init + conv 1x1	LC + conv init + dyn 1x1 (Eq. (4.18))
Top-1	65.1	42.6	44.8	51.0	62.0	56.0
Top-5	85.8	65.6	67.8	74.6	83.6	78.7
Brain-Score [23]	.353	.327	.335	.335	.389	.355

Table 4.4: LC: locally connected; conv init: convolutional initialization; conv 1x1: replacing LC layers with 1x1 filters by convolutions; dyn 1x1: using Eq. (4.18) in 1x1 LC layers.

4.7 Discussion

4.7.1 Weight sharing with a sleep phase

We presented two ways to circumvent the biological implausibility of weight sharing, a crucial component of convolutional networks. The first was through data augmentation via multiple image translations. The second was dynamic weight sharing via lateral connections, which allows neurons to share weight information during a sleep-like phase; weight updates are then done using Hebbian plasticity. Data augmentation requires a large number of repetitions in the data, and, consequently, longer training times, and yields only small improvements in performance. However, only a small number of repetitions can be naturally covered by saccades, and active object manipulation happens for a limited number of visual stimuli. Dynamic weight sharing needs a separate sleep phase, rather than more data, and yields large performance gains. In fact, it achieves near convolutional performance even on hard tasks, such as ImageNet classification, making it a much more likely candidate than data augmentation for the brain. In addition, well-performing locally connected networks trained with dynamic weight sharing achieve a better fit to the ventral stream data (measured by the Brain-Score [22, 164]). The sleep phase can occur during actual sleep, when the network (i.e., the visual system) stops receiving visual inputs, but maintains some internal activity. This is supported by plasticity studies during sleep

(e.g. [186, 187]).

4.7.2 Weight sharing with noise-cancelling plasticity

We also proposed an online learning model for weight sharing. It uses lateral connections and anti-Hebbian plasticity, runs during training, and achieves good performance on the ImageNet visual recognition task. Our model applies mainly to 1x1 convolutions, but we showed that good performance can be achieved if other channels are locally connected and initialised to be convolutional. These results compliment the sleep phase-based model, but a hybrid scheme is possible where both happen at once. Combining the two can, potentially, increase performance, but we leave that to future work. Our model can be interpreted as a form of homeostatic plasticity. However, unlike the standard role of homeostatic plasticity, which is to compensate for weight divergence during Hebbian learning, in our model its role is to improve the performance of gradient-based learning (or its approximations).

4.7.3 Limitations of the approach

Our approach has several limitations. First, the pattern generation scheme needs layers to have filters of the same size. Second, we assume that the very first layer (e.g., V1) receives inputs from another area (e.g., LGN) that can generate repeated patterns, but doesn't need weight sharing.

Then, the structure of convolutional layers as used in deep learning significantly differs from the brain. A standard convolutional layer typically has from dozens to hundreds of individual channels. Early visual stream (including V1) also contains parallel processing streams/channels that correspond to cell-type specific modules [188], although their number is limited (A 2001 study [189] lists several types with distinct connectivity). It is not clear if convolutional networks can perform well with such a limited number of channels.

Precise and non-plastic lateral connectivity also poses a problem. This can be genetically encoded, or learned early on using correlations in the input data (if layer l can generate repeated patterns, layer $l+1$ can modify its lateral connectivity based on input correlations). In addition, we turned the lateral connections off during the

awake phase in the experiments with the sleep phase. This is because they could potentially interfere with network's computation, although leaving them on would work like the centring part of the Layer Norm [190]. We used Batch Norm in our networks; replacing it with Layer Norm and leaving the lateral connections always on could be a solution to this problem.

Lateral connections do in fact exist in the visual stream, with neurons that have similar tuning curves showing strong lateral connections [191]. Moreover, at least in V1 lateral connections have been found to span 7mm of space with a 0.75mm periodicity [192]; the same study concluded that neurons with similar orientation specificity tend to be laterally connected. This is not yet ideal for our model: we assumed identical lateral connections among individual neurons, rather than random connections between neural columns. Our model could be adapted to to match fixed random projections of weights to account for this feature, which could be an interesting direction for future work.

Next, the sleep phase works iteratively over layers. This can be implemented with neuromodulation that enables plasticity one layer at a time. Alternatively, weight sharing could work simultaneously in the whole network due to weight regularisation (as it ensures that the final solution preserves the initial average weight), although this would require longer training due to additional noise in deep layers. Third, in our scheme the lateral connections are used only for dynamic weight sharing, and not for training or inference. As our realistic model in Section 4.4.2 implements this connectivity via an inhibitory neuron, we can think of that neuron as being silent outside of the sleep phase.

Finally, we trained networks using backpropagation, which is not biologically plausible [45]. However, our weight sharing scheme is independent of the wake-phase training algorithm, and therefore can be applied along with any biologically plausible update rule.

4.7.4 Conclusions

Our approach to dynamic weight sharing is not relevant only to convolutions. First, it is applicable to non-convolutional networks, and in particular visual transformers

[167–170] (and more recent MLP-based architectures [171–173]). In such architectures, input images (and intermediate two-dimensional representations) are split into non-overlapping patches; each patch is then transformed with the *same* fully connected layer – a computation that would require weight sharing in the brain. This can be done by connecting neurons across patches that have the same relative position, and applying our weight dynamics (see Appendix C.1.4). Second, [46] faced a problem similar to weight sharing – weight transport (i.e., neurons not knowing their output weights) – when developing a plausible implementation of backprop. Their weight mirror algorithms used an idea similar to ours: the value of one weight was sent to another through correlations in activity.

It is possible that other approaches could achieve similar performance in locally connected network without explicit "convolutional" regularisation like in our approach. One particular candidate is unsupervised/self-supervised learning, as it avoids overfitting to the train set (something we observed in purely locally connected networks) due to the chosen objective/data structure; this is a potential future direction. Extensive data augmentation (or simply larger datasets) and large networks could potentially work better too, but our experiments with locally connected networks indicate that overfitting to the training set is hard to overcome is such settings.

Our study shows that both performance and the computation of convolutional networks can be reproduced in more realistic architectures. While the exact weight sharing scheme proposed here is unlikely to exist in the visual stream, our model suggests that neurons can exchange weight information and, perhaps more importantly for learning, gradient information, which facilitates generalisation in neural networks.

Chapter 5

Locally connected networks as ventral stream models

This chapter is based on Pogodin and Latham (2022); the results are my own.

Summary Most deep learning models of the ventral stream, and convolutional networks in particular, share weights among neurons. Weight sharing during learning is crucial for good performance on image recognition tasks, but it is not biologically plausible. In this work, we compare performance and Brain-Score results of ImageNet-trained networks in multiple configurations: convolutional, locally connected (i.e., convolutional without weight sharing), and locally connected with anti-Hebbian plasticity mechanisms that promote weight sharing. We also study the role of initialisation on performance of those networks. We find that the more weight sharing networks have, the better they perform on both ImageNet and Brain-Score, which can sometimes be further improved with a convolutional initialisation. However, locally connected networks outperform their convolutional counterparts on purely neural data (areas V1, V2, V4, IT), but not on behavioural responses. Moreover, ImageNet performance negatively correlates with correspondence to V1 data, suggesting that better models of early visual processing don't necessarily provide a good input for models of deeper visual areas.

5.1 Introduction

Convolutional networks not only have great performance on image recognition tasks, but also develop representations similar to the primate visual stream. For instance, they provide a good fit to multiple areas of the ventral stream [22], and even explain the separation between the ventral and the dorsal streams [25]. However, the biological plausibility of convolutional networks is questionable, since they need to share weights among neurons during training (Fig. 5.1A). If they don't (even if they are locally connected, Fig. 5.1B), they perform much worse on image recognition tasks, and show a worse fit to the visual stream [149]. This issue is not limited to convolutional networks – any network involving matrix-matrix multiplication (with one matrix representing neurons, and the other one representing weights) needs weight sharing (e.g., a recently popular architecture, transformers). Recently it was shown that locally connected networks (Fig. 5.1B) can share weights through anti-Hebbian plasticity, but the network must stop training for a "sleep phase" that uses lateral connectivity in each layer [149] (Fig. 5.1C,D).

In this work, we study how well locally connected and convolutional networks correspond to the ventral stream data, as measure by the Brain-Score [22, 164], which combines recordings of behavioural responses (which differ from classification accuracy, see [22]) and areas V1, V2, V4, and IT to naturalistic stimuli in primates.

Brain-Score provides a unified way to compare visual representations in deep networks to the ones in the primate visual cortex, which also allows us to compare different deep networks among each other. As the Brain-Score combines data from different species and different imaging techniques, it allows us to compare models "on average". The downside of this approach is that it is (in its current form) limited to static images; it also doesn't involve any learning. Moreover, as the Brain-Score is based on representation similarity analysis, it doesn't directly compare individual neural representation in deep networks to the brain data. While it only allows us to claim that some networks distinguish image categories in a similar way to the brain, it also makes the approach more model-agnostic (i.e., it doesn't require the model to be anatomically identical to the visual stream, or to have certain realistic features

such as spikes).

We evaluate the role of initial conditions, frequency of weight sharing (as in [149]) and architectural differences. We find that better performing models typically result in a better Brain-Score, but locally connected networks provide a better match to non-behavioural data. In addition, poorly performing models match better to V1 data.

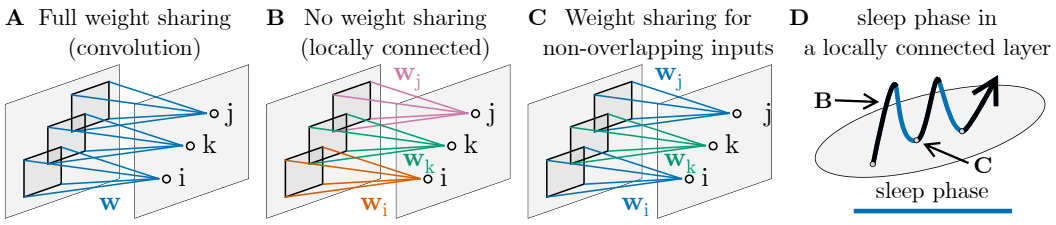

Figure 5.1: **A.** Convolutional layer: all neurons see a patch of the input and have the same weight $\mathbf{w}_i = \mathbf{w}_k = \mathbf{w}_j = \mathbf{w}$. **B.** Locally connected layer: same, but with different weights. **C.** Only neurons with non-overlapping inputs share weights (equivalent to a stack of strided convolutions). **D.** Training in a locally connected layer leads to different weights (like in **B**); a sleep phase ([149]) shares weights among neurons with non-overlapping inputs (like in **C**).

Locally connected (LC) networks with a weight sharing sleep phase can achieve results similar to their convolutional counterpart [149]. The sleep phase uses lateral connectivity to create shared inputs for different neurons and equalise their activity with anti-Hebbian learning, which leads to convolution-like weight sharing (see the corresponding paper for mathematical details). However, [149] only considered a relatively small model (half-width ResNet-18) and didn't study the role of initial conditions and architectural changes.

Here, we conduct experiments on a standard ResNet-18 network in multiple configurations: convolutional, locally connected with random initialisation (as in [149]), locally connected with a convolutional initialisation (as in [163]), and locally connected with a convolutional first layer. As the first layer in a ResNet-18 has the highest resolution (224 by 224 with stride 2 vs. 56 by 56 with stride 1 for the next one) and the largest filter size (7 by 7 vs. 3 by 3 for all other layers), it's potentially prone to more overfitting than other layers. Making it convolutional is inspired by pre-V1 processing done in the visual system.

5.1.1 Training details

We trained all networks on ImageNet [94] with standard augmentations. We used AdamW [89] with a batch size of 128. We used Nvidia A100 GPUs (40GB VRAM; for GPUs with less memory, the model would not fit on a single one as just the LC weights take up about 11GB). We used the implementation from [149], which is available at https://github.com/romanpogodin/towards-bio-plausible-conv. Results and links to Brain-Score models are provided in Table C.11.

As training locally connected network required a significant amount of computational resources, we were only able to test single instances of networks and thus unable to provide error bars for the results.

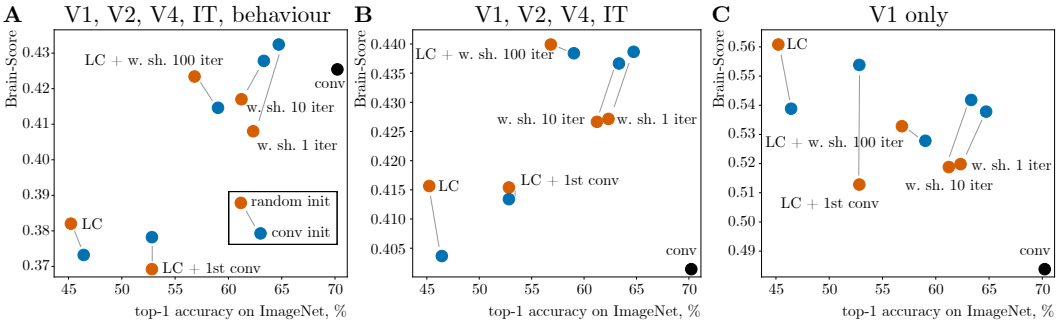

Figure 5.2: ImageNet top-1 accuracy vs. Brain-Score for several ResNet-18 networks: convolutional, locally connected (LC), LC with a convolutional first layer, LC with a weight sharing sleep phase every 1/10/100 iterations. Orange: randomly initialised LC net; blue: LC net with a convolutional initialisation. **A.** Average Brain-Score (V1, V2, V4, IT, behavioural data). **B.** Same, but without behavioural data. **C.** Only V1 data.

5.1.2 Results (all brain areas)

First, we compared ImageNet top-1 test accuracy and the Brain-Score of all our models (Fig. 5.2A). Overall, networks with better ImageNet accuracy also have a higher Brain-Score (Pearson correlation coefficient ≈ 0.84, $p \approx 0.001$). However, if we exclude behavioural data from the analysis and only use neural data (Fig. 5.2B), a fully convolutional network shows the worst overall fit to neural data despite the best performance. One possible explanation is that LC networks produce a more diverse set of activations, as there's less regularity in the weights.

5.1.3 Results (V1)

For V1 data specifically, purely LC networks show the best V1 score (Fig. 5.2C) and ImageNet accuracy negatively correlates with Brain-Score (Pearson correlation coefficient $\approx -0.65,\ p \approx 0.03$). These results are consistent with those obtained for smaller networks in [149]. Here, we attempted to investigate what drives the discrepancy between V1 data and the rest of the results.

First, we checked if representations in earlier layers (that typically match V1) are useful for ImageNet, and if readout accuracy for these layers correlates with the V1 score. We trained a linear readout on top of the second ResNet block (out of four), and observed very poor performance (2-6%) in purely LC networks, as opposed to 8-20% in networks with more regularisation. Therefore, good V1 representations alone are not very helpful for ImageNet.

We also trained a shorter ResNet with only two blocks (10 layers total) to see if the depth of the model affects the phenomenon. It doesn't: the V1 score of a short ResNet LC network is 0.544, vs. 0.511 for its convolutional counterpart that performs better on ImageNet. However, the difference in score is twice as small for the shorter architecture compared to a full ResNet-18.

Finally, we tested untrained networks to see if different initial conditions can explain V1 scores. For a randomly initialised locally connected network, the V1 score was predictably low: 0.347 (much smaller than everything in Fig. 5.2C). A convolutional initialisation, however, resulted in a score of 0.516 – *higher than for a trained convolutional network*. This surprising result suggests that the good V1 fit of convolutional networks comes primarily from the architecture and translation equivariance of convolutional layers, while training on ImageNet leads to a more ImageNet-specific representation that don't necessarily correspond to V1 representations.

5.2 Discussion

We explored how deep networks that substitute implausible weight sharing of convolutions with realistic mechanisms, namely local connections and sleep-phase induced

partial weight sharing, perform on ImageNet and Brain-Score. We found that those two metrics generally correlate (Pearson correlation coefficient $\approx 0.84,\ p \approx 0.001$), and that a convolutional initialisation of locally connected networks can improve both of them. Moreover, we found that V1 performance *negatively* correlates with ImageNet accuracy, both for full networks (Pearson correlation coefficient $\approx -0.65,\ p \approx 0.03$) and for representations extracted from V1-like early layers.

The discrepancy between V1 and all other brain areas shows that training (on ImageNet) is not always beneficial for all architectures: while locally connected networks with any initialisation end up with a higher match to the V1 data after training, a fully convolutional one sees a decrease in V1 match. This is in contrast to deeper areas, where training always increases the match to data. We also found that improving V1 fit of a model might not imply improvements for deeper areas, which is somewhat consistent with the idea that the visual cortex is not strictly hierarchical [194]. However, our findings are specific to the architectural changes we've tested (i.e., amount of weight sharing). Therefore, they don't contradict previous studies that don't show the V1 fit/ImageNet accuracy discrepancy in different convolutional networks [178].

Overall, our results show that deep learning models of the visual stream can be improved by adding realistic training constraints, such as the lack of weight sharing at all times. The improvement, however, is only visible when we compare the models to real data (in our case, via Brain-Score [22, 164]), and not to machine learning benchmarks.

Chapter 6

Conclusions and Future Work

Conclusions In Chapter 2, we discuss a biologically plausible mechanism for training deep networks with three-factor Hebbian plasticity. The method increases dependence between neural activity in each layer of a feedforward network and the desired outputs. This is done by optimising a kernel methods-based measure of statistical dependence called the Hilbert-Schmidt Independence Criterion (HSIC; [32]). Our results provide an effective method to train deep networks. Moreover, the HSIC-derived three-factor Hebbian updates make predictions about the specific role of neuromodulators and divisive normalisation in learning.

In Chapter 3, we approach self-supervised learning (SSL) with HSIC-based objectives. Previously, [119] noted the disconnect between the mutual information (MI) optimisation view of self-supervised learning and the actual behaviour of mutual information. In particular, learning with variational estimators of MI, such as InfoNCE [29], can happen even without MI changes. We show that InfoNCE is also an HSIC-based objective, which explains learning in self-supervised models more directly and allows to improve computational performance of SSL using kernel methods.

In Chapter 4 and Chapter 5, we discussed the role and potential implementations of weight sharing in the models of the visual stream. Weight sharing has been an important part of convolutional networks, which are particularly good at image recognition [151, 152] and also match well to the activity in the private visual stream [22, 23]. We showed that weight sharing can indeed be implemented through anti-

Hebbian learning and specifically wired lateral connections. Providing a plausible model of weight sharing allows us to treat convolutional networks not only as functional models of the visual stream (i.e., on the level of neural representations), but also as a more literal circuit model. We also showed that relaxing convolutional weights to fully match the proposed weight sharing model makes networks more visual stream-like at the functional level.

Future work Studies at the intersection of deep learning and theoretical neuroscience have three major possible directions: making models perform better and on a wider range of tasks, making models more biological, and deriving concrete experimental predictions based on the models.

The last task is perhaps the hardest, as an experiment should have a carefully proposed null hypothesis to rule out a computation we expect to see, as many algorithms can be implemented in different ways with the same final goal. One example is translation invariance observed in the primate visual stream (note that it doesn't imply exact weight sharing). It's been observed that human responses are not exactly translation invariant on newly learned stimuli [195], but the testing phase of the experiment immediately followed the training phase. Therefore, while this result rules out strict weight sharing (which we did not expect to occur), it doesn't rule out a potential "sleep phase" with some degree of weight equalisation that happens.

The extension of the Chapter 2 layer-wise learning approach to the self-supervised learning setup in Chapter 3 is straightforward: changing the label similarity used in supervised learning to positive/negative (i.e., same/different source) pair indicator used in self-supervised learning is the only required change. Moreover, the approximation of the HSIC objective in Chapter 2 can also be applied to Chapter 3. The approximation removed computation over triples of points needed for HSIC, meaning that we can interpret self-supervised learning with this approximation as comparing pairs of (any) examples, rather than as a pair of positive examples and a large number of negative examples. Such approximation would make self-supervised learning easier to implement in a realistic network, and potentially simplify the theoretical analysis of self-supervised learning.

Finally, as discussed in Section 4.7.3, the weight sharing model presented in Chapter 4 is not yet realistic and could be further developed. In particular, it might be possible to extend the idea to randomly connected columns of neurons rather than individual neurons.

Consider the following extension of the model from Chapter 4. At each location indexed by i, the column of output neurons is $\mathbf{z}_i$ receives a vector of inputs $\mathbf{H}_i\mathbf{x}$ for a repeated input $\mathbf{x}$ and a fixed random matrix of local input connections $\mathbf{H}_i$. In the output layer, $\mathbf{z}_i$ contributes to the inhibitory neuron through a fixed random matrix $\mathbf{R}_i$ and receives an input from the inhibitory pool through a fixed random matrix $\mathbf{B}_i$ (all matrices are square for simplicity). As a result, for the plastic weights $\mathbf{W}_i$,

$$\mathbf{z}_i = \mathbf{W}_i\mathbf{H}_i\mathbf{x} - \frac{1}{N}\mathbf{B}_i\sum_j \mathbf{R}_j\mathbf{W}_j\mathbf{H}_j\mathbf{x}, \tag{6.1}$$

$$\Delta\mathbf{W}_i = \mathbf{z}_i(\mathbf{H}_i\mathbf{x})^\top. \tag{6.2}$$

In the original method, we had $\sum_i \Delta\mathbf{W}_i = 0$ and so the weights converged to the average weight. Here, the invariant is different. Assume that $\mathbf{B}_i\mathbf{R}_i$ averages to an identity, such that $\frac{1}{N}\sum_i \mathbf{B}_i\mathbf{R}_i \approx \mathbb{E}\,\mathbf{B}\mathbf{R} = \mathbf{I}$, and that all $\mathbf{H}_i$ are invertible. Then,

$$\mathbf{R}_i\Delta\mathbf{W}_i(\mathbf{H}_i^\top)^{-1} = \mathbf{R}_i\left(\mathbf{W}_i\mathbf{H}_i\mathbf{x} - \frac{1}{N}\mathbf{B}_i\sum_j \mathbf{R}_j\mathbf{W}_j\mathbf{H}_j\mathbf{x}\right)\mathbf{x}^\top\mathbf{H}_i^\top(\mathbf{H}_i^\top)^{-1}, \tag{6.3}$$

and therefore

$$\frac{1}{N}\sum_i \mathbf{R}_i\Delta\mathbf{W}_i(\mathbf{H}_i^\top)^{-1} = \frac{1}{N}\sum_i \mathbf{R}_i\left(\mathbf{W}_i\mathbf{H}_i - \frac{1}{N}\mathbf{B}_i\sum_j \mathbf{R}_j\mathbf{W}_j\mathbf{H}_j\right)\mathbf{x}\mathbf{x}^\top \tag{6.4}$$

$$= \frac{1}{N}\sum_i \mathbf{R}_i\mathbf{W}_i\mathbf{H}_i\mathbf{x}\mathbf{x}^\top - \frac{1}{N^2}\sum_i \mathbf{R}_i\mathbf{B}_i\sum_j \mathbf{R}_j\mathbf{W}_j\mathbf{H}_j\mathbf{x}\mathbf{x}^\top \approx 0. \tag{6.5}$$

The Hebbian updates will thus preserve the weighted average of the weights. The resulting "shared" solution will not exhibit translation equivariance of the standard convolutions, but it should still regularise the network. Exploring how this weight sharing (or, rather, weight exchange) mechanism affects performance and whether it better fits the primate visual system is a potential future direction.

Appendix A

Chapter 2 Appendix

A.1 Kernel methods, HSIC and pHSIC

A.1.1 pHSIC

We define the "plausible" HSIC by substituting $\mathbb{E}_{\mathbf{xy}}\mathbb{E}_{\mathbf{x}'}\mathbb{E}_{\mathbf{y}'}$ in the definition of HSIC by $\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}}\mathbb{E}_{\mathbf{x}'}\mathbb{E}_{\mathbf{y}'}$,

$$\mathrm{pHSIC}(X,Y) = \left(\mathbb{E}_{\mathbf{xy}}\mathbb{E}_{\mathbf{x}'\mathbf{y}'} - \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}}\mathbb{E}_{\mathbf{x}'}\mathbb{E}_{\mathbf{y}'}\right)\left(k(\mathbf{x},\mathbf{x}')\,k(\mathbf{y},\mathbf{y}')\right). \tag{A.1}$$

Therefore, $\mathrm{HSIC}(X,Y) = \mathrm{pHSIC}(X,Y)$ when $\mathbf{a}$ and $\mathbf{b}$ are independent ($\mathbb{E}_{\mathbf{xy}} = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}}$, which is not useful in our case), and when $\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}') = 0$ or $\mathbb{E}_{\mathbf{y}'}k(\mathbf{y},\mathbf{y}') = 0$.

In addition, $\mathrm{pHSIC}(X,X)$ becomes the variance of $k(\mathbf{x},\mathbf{x}')$ with respect to $p_{\mathbf{x}}(\mathbf{x})p_{\mathbf{x}}(\mathbf{x}')$,

$$\mathrm{pHSIC}(X,X) = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{x}'}\left(k(\mathbf{x},\mathbf{x}')\right)^2 - \left(\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')\right)^2 = \mathbb{V}\mathrm{ar}\left(k(\mathbf{x},\mathbf{x}')\right). \tag{A.2}$$

By combining Eq. (1.22) and Eq. (A.2), we can show that $\mathrm{HSIC}(X,X) \leq \mathrm{pHSIC}(X,X)$: denoting $\mu_{\mathbf{x}}(\mathbf{x}) = \mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')$, we have

$$\begin{aligned}\mathrm{pHSIC}(X,X) - \mathrm{HSIC}(X,X) &= 2\,\mathbb{E}_{\mathbf{x}}\left(\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')\right)^2 - 2\left(\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{x}'}k(\mathbf{x},\mathbf{x}')\right)^2 \\ &= 2\left(\mathbb{E}_{\mathbf{x}}\mu_{\mathbf{x}}(\mathbf{x})^2 - \left(\mathbb{E}_{\mathbf{x}}\mu_{\mathbf{x}}(\mathbf{x})\right)^2\right) \\ &= 2\,\mathbb{V}\mathrm{ar}\left(\mu_{\mathbf{x}}(\mathbf{x})\right) \geq 0.\end{aligned} \tag{A.3}$$

As a result, our objective,

$$\mathrm{pHSIC}(Z^k, Z^k) - \gamma \mathrm{pHSIC}(Y, Z^k)\,, \tag{A.4}$$

is an upper bound on the "true" objective whenever $\mathrm{pHSIC}(Y, Z^k) = \mathrm{HSIC}(Y, Z^k)$, which is the case when $\mathbb{E}_{\mathbf{y}_2} k(\mathbf{y}_1, \mathbf{y}_2) = 0$.

Finally, the empirical estimate of pHSIC (which can be derived just like for HSIC in [32]) is

$$\widehat{\mathrm{pHSIC}}(X,Y) = \frac{1}{m^2}\sum_{ij} k(\mathbf{x}_i, \mathbf{x}_j)k(\mathbf{y}_i, \mathbf{y}_j) - \frac{1}{m^2}\sum_{ij} k(\mathbf{x}_i, \mathbf{x}_j)\frac{1}{m^2}\sum_{ql} k(\mathbf{y}_q, \mathbf{y}_l)\,. \tag{A.5}$$

A.1.2 How much information about the label do we need?

In our rules, the information about the label comes only through $k(\mathbf{y}_i, \mathbf{y}_j)$, where $\mathbf{y}$ is a one-hot vector – for n classes, an n-dimensional vector of mainly zeros with only a single one (which corresponds to its label). For $k(\mathbf{y}_i, \mathbf{y}_j)$ we use the cosine similarity kernel, with $\mathbf{y}$ centred. If the dataset is balanced (i.e., all classes have the same probability, $1/n$), $\mathbb{E}_{\mathbf{y}_j} k(\mathbf{y}_i, \mathbf{y}_j) = 0$ and the resulting kernel is

$$k(\mathbf{y}_i, \mathbf{y}_j) = \frac{(\mathbf{y}_i - \frac{1}{n}\mathbf{1}_n)^\top (\mathbf{y}_j - \frac{1}{n}\mathbf{1}_n)}{\left\|\mathbf{y}_i - \frac{1}{n}\mathbf{1}_n\right\| \left\|\mathbf{y}_j - \frac{1}{n}\mathbf{1}_n\right\|} = \frac{\mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] - \frac{1}{n}}{1 - \frac{1}{n}} = \begin{cases} 1, & \mathbf{y}_i = \mathbf{y}_j\,, \\ -\frac{1}{n-1}, & \text{otherwise}\,. \end{cases} \tag{A.6}$$

If there are many classes, this signal approaches $\mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right]$, which is the same as the uncentred linear kernel.

Equation (A.6) is especially convenient, because the kernel takes on only two values, 1 and $-1/(n-1)$. Consequently, precise labels are not needed. This is not the case for unbalanced classes, as $\mathbb{E}\mathbf{y}_i \neq \mathbf{1}_n/n$ and the centring of $\mathbf{y}$ doesn't make the normalised vector centred. However, taking the linear kernel gives an almost

binary signal,

$$\begin{aligned} k(\mathbf{y}_i, \mathbf{y}_j) &= (\mathbf{y}_i - \mathbf{p})^\top (\mathbf{y}_j - \mathbf{p}) = \mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] + \sum_k p_k^2 - p_i - p_j \\ &= \mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] + O\left(\frac{1}{n}\right), \end{aligned} \tag{A.7}$$

as long as the probability of each class k, p_k, is $O(1/n)$ (i.e., they are roughly balanced). As a result, the teaching signal is nearly binary, and we can compute it without knowing the probability of each class.

A.2 Derivations of the update rules for plausible kernelized information bottleneck

A.2.1 General update rule

Here we derive the gradient of pHSIC in our network, along with its empirical estimate. Our starting point is the observation that because the network is feedforward, the activity in layer k is a deterministic function of the previous layer and the weights: $Z^k = f\left(\mathbf{W}^k, Z^{k-1}\right)$ (this includes both feedforward layers and layers with divisive normalisation). Therefore, we can write the expectations of any function $g(Y, Z^k)$ (which need not actually depend on Y) in terms of Z^{k-1},

$$\mathbb{E}_{\mathbf{y}\mathbf{z}^k} g(\mathbf{y}, \mathbf{z}^k) = \mathbb{E}_{\mathbf{y}\mathbf{z}^{k-1}} g\left(\mathbf{y}, f\left(\mathbf{W}^k, \mathbf{z}^{k-1}\right)\right). \tag{A.8}$$

As a result, we can write the gradients of pHSIC (assuming we can exchange the order of differentiation and expectation, and the function is differentiable at $\mathbf{W}^k$) as

$$\frac{d\,\mathrm{pHSIC}(Y)(Z^k)}{d\,\mathbf{W}^k} = \left(\mathbb{E}_{\mathbf{y}_1\mathbf{z}_1^{k-1}}\mathbb{E}_{\mathbf{y}_2\mathbf{z}_2^{k-1}} - \mathbb{E}_{\mathbf{y}_1}\mathbb{E}_{\mathbf{z}_1^{k-1}}\mathbb{E}_{\mathbf{y}_2}\mathbb{E}_{\mathbf{z}_2^{k-1}}\right) \tag{A.9}$$

$$k(\mathbf{y}_1, \mathbf{y}_2)\,\frac{d\,k\left(f\left(\mathbf{W}^k, \mathbf{z}_1^{k-1}\right), f\left(\mathbf{W}^k, \mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k}, \tag{A.10}$$

$$\frac{d\,\mathrm{pHSIC}(Z^k)(Z^k)}{d\,\mathbf{W}^k} = \tag{A.11}$$

$$2\,\mathbb{E}_{\mathbf{z}_1^{k-1}}\mathbb{E}_{\mathbf{z}_2^{k-1}} k(\mathbf{z}_1^k,\mathbf{z}_2^k)\frac{d\,k\left(f\left(\mathbf{W}^k,\mathbf{z}_1^{k-1}\right),f\left(\mathbf{W}^k,\mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k} \tag{A.12}$$

$$-2\left(\mathbb{E}_{\mathbf{z}_1^{k-1}}\mathbb{E}_{\mathbf{z}_2^{k-1}} k(\mathbf{z}_1^k,\mathbf{z}_2^{k-1})\right) \tag{A.13}$$

$$\times\left(\mathbb{E}_{\mathbf{z}_1^{k-1}}\mathbb{E}_{\mathbf{z}_2^{k-1}}\frac{d\,k\left(f\left(\mathbf{W}^k,\mathbf{z}_1^{k-1}\right),f\left(\mathbf{W}^k,\mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k}\right). \tag{A.14}$$

To compute these derivatives from data, we take empirical averages (see Eq. (A.5)),

$$\begin{aligned}\frac{d\left(\widehat{\mathrm{pHSIC}}(Z^k,Z^k)-\gamma\widehat{\mathrm{pHSIC}}(Y,Z^k)\right)}{d\,\mathbf{W}^k} =& \\ 2\frac{1}{m^2}\sum_{ij}k(\mathbf{z}_i^k,\mathbf{z}_j^k)\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,\mathbf{W}^k} - 2\frac{1}{m^2}\sum_{ql}k(\mathbf{z}_q^k,\mathbf{z}_l^k)\frac{1}{m^2}\sum_{ij}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,\mathbf{W}^k} & \\ -\gamma\frac{1}{m^2}\sum_{ij}k(\mathbf{y}_i,\mathbf{y}_j)\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,\mathbf{W}^k}+\gamma\frac{1}{m^2}\sum_{ql}k(\mathbf{y}_q,\mathbf{y}_l)\frac{1}{m^2}\sum_{ij}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,\mathbf{W}^k}.&\end{aligned} \tag{A.15}$$

Making the definition $\mathring{k}(\mathbf{a}_i,\mathbf{a}_j)=k(\mathbf{a}_i,\mathbf{a}_j)-\sum_{ql}k(\mathbf{a}_q,\mathbf{a}_l)/m^2$, this simplifies to

$$\begin{aligned}&\frac{d\left(\widehat{\mathrm{pHSIC}}(Z^k,Z^k)-\gamma\widehat{\mathrm{pHSIC}}(Y,Z^k)\right)}{d\,\mathbf{W}^k}\\ &\qquad=\frac{1}{m^2}\sum_{ij}\left(2\mathring{k}(\mathbf{z}_i^k,\mathbf{z}_j^k)-\gamma\mathring{k}(\mathbf{y}_i,\mathbf{y}_j)\right)\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,\mathbf{W}^k}.\end{aligned} \tag{A.16}$$

The key quantity in the above expressions is the derivative of the kernel with respect to the weights. Below we compute those for the Gaussian and cosine similarity kernels, and explain why the cosine similarity update is implausible.

A.2.2 Gaussian kernel

For the Gaussian kernel, the derivative with respect to a single weight is

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{nm}^k} &= \frac{d}{d\,W_{nm}^k}\exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{z}_i^k-\mathbf{z}_j^k\right\|^2\right)\\ &= -\frac{k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{\sigma^2}\left(z_{n,i}^k-z_{n,j}^k\right)\frac{d\left(z_{n,i}^k-z_{n,j}^k\right)}{d\,W_{nm}^k}\\ &= -\frac{k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{\sigma^2}\left(z_{n,i}^k-z_{n,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}-f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\right).\end{aligned} \tag{A.17}$$

For the linear network $f'(x)=1$, and so the gradient w.r.t. $\mathbf{W}^k$ becomes an outer product (Eq. (2.17)).

A.2.3 Gaussian kernel with grouping and divisive normalisation

For the circuit with grouping and divisive normalisation, the kernel is a function of $\mathbf{v}$, not $\mathbf{z}$ (see Eqs. (2.18), (2.19) and (2.20)). This makes the derivative with respect to $\mathbf{z}$ more complicated than the above expression would suggest. Specifically, using Eq. (2.20) for the kernel with grouping, we have

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{\alpha nm}^k} &= \frac{d}{d\,W_{\alpha nm}^k}\exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{v}_i^k-\mathbf{v}_j^k\right\|^2\right)\\ &= -\frac{k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{\sigma^2}\sum_{\alpha'}(v_{\alpha',i}^k-v_{\alpha',j}^k)\frac{d\,(v_{\alpha',i}^k-v_{\alpha',j}^k)}{d\,W_{\alpha nm}^k},\end{aligned} \tag{A.18}$$

where the sum over α' appears due to centring (so all α are coupled). Using Eq. (2.19) to express $\mathbf{v}$ in terms of $\mathbf{u}$, the above derivative is

$$\frac{d\,v_{\alpha',i}^k}{d\,W_{\alpha nm}^k} = \frac{d\left((u_{\alpha',i}^k)^{1-p}-\frac{1}{c_\alpha^k}\sum_{\alpha''}(u_{\alpha'',i}^k)^{1-p}\right)}{d\,W_{\alpha nm}^k} = \left(\delta_{\alpha\alpha'}-\frac{1}{c_\alpha^k}\right)\frac{d\,(u_{\alpha,i}^k)^{1-p}}{d\,W_{\alpha nm}^k}. \tag{A.19}$$

Then using Eq. (2.18) to express $\mathbf{u}$ in terms of $\mathbf{z}$, we have

$$\begin{aligned}
\frac{d\,(u^k_{\alpha,i})^{1-p}}{dW^k_{\alpha nm}} &= \frac{1-p}{(u^k_{\alpha,i})^p}\frac{d\left(\frac{\delta}{c^k_\alpha}+\frac{1}{c^k_\alpha}\sum_{n'}\left(\mathring{z}^k_{\alpha n',i}\right)^2\right)}{dW^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',i}\frac{d\,\mathring{z}^k_{\alpha n',i}}{dW^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',i}\frac{d\left(z^k_{\alpha n',i}-\frac{1}{c^k_\alpha}\sum_{n''}z^k_{\alpha n'',i}\right)}{dW^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',i}\left(\delta_{n'n}-\frac{1}{c^k_\alpha}\right)\frac{d\,z^k_{\alpha n,i}}{dW^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,i})^p}\frac{2}{c^k_\alpha}\mathring{z}^k_{\alpha n,i}\frac{d\,z^k_{\alpha n,i}}{dW^k_{\alpha nm}}.
\end{aligned} \tag{A.20}$$

Inserting this expression into Eq. (A.19), inserting that into Eq. (A.18), and performing a small amount of algebra, we arrive at

$$\frac{d\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{dW^k_{\alpha nm}} = -\frac{k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{\sigma^2}\sum_{\alpha'}(v^k_{\alpha',i}-v^k_{\alpha',j})\left(\delta_{\alpha\alpha'}-\frac{1}{c^k_\alpha}\right)\frac{2\,(1-p)}{c^k_\alpha} \tag{A.21}$$

$$\times\left(\frac{\mathring{z}^k_{\alpha n,i}}{(u^k_{\alpha,i})^p}\frac{d\,z^k_{\alpha n,i}}{dW^k_{\alpha nm}}-\frac{\mathring{z}^k_{\alpha n,j}}{(u^k_{\alpha,j})^p}\frac{d\,z^k_{\alpha n,j}}{dW^k_{\alpha nm}}\right). \tag{A.22}$$

As $v^k_{\alpha,i}$ is centred with respect to α and the last term doesn't depend on α', the final expression becomes

$$\frac{d\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{dW^k_{\alpha nm}} = -\frac{2\,(1-p)\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{\sigma^2 c^k_\alpha}(v^k_{\alpha,i}-v^k_{\alpha,j}) \tag{A.23}$$

$$\times\left(\frac{\mathring{z}^k_{\alpha n,i}}{(u^k_{\alpha,i})^p}f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,i}-\frac{\mathring{z}^k_{\alpha n,j}}{(u^k_{\alpha,j})^p}f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,j}\right). \tag{A.24}$$

A.2.4 Cosine similarity kernel

Assuming that $\mathbf{z}^k$ is bounded away from zero (because $\mathbf{z}^k/\left\|\mathbf{z}^k\right\|$ is not continuous at 0; adding a smoothing term to the norm would help but won't change the derivation),

the derivative of the cosine similarity kernel is

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{nm}^k} &= \frac{d}{d\,W_{nm}^k}\frac{(\mathbf{z}_i^k)^\top \mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|} \\ &= \frac{1}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\frac{d\left(z_{n,i}^k z_{n,j}^k\right)}{d\,W_{nm}^k} - \frac{(\mathbf{z}_i^k)^\top \mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|^2\left\|\mathbf{z}_j^k\right\|^2}\frac{d\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}{d\,W_{nm}^k}.\end{aligned} \tag{A.25}$$

The first derivative is simple,

$$\frac{d\left(z_{n,i}^k z_{n,j}^k\right)}{d\,W_{nm}^k} = z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1} + z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}. \tag{A.26}$$

However, in this form it is hard to interpret, as both terms have the pre-synaptic activity at one point and the post-synaptic activity at the other. We can, though, can re-arrange it into differences in activity,

$$\begin{aligned}\frac{d\left(z_{n,i}^k z_{n,j}^k\right)}{d\,W_{nm}^k} &= \left(z_{n,i}^k - z_{n,j}^k\right) f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1} + z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1} \\ &\quad + \left(z_{n,j}^k - z_{n,i}^k\right) f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1} + z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1} \\ &= -\left(z_{n,i}^k - z_{n,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1} - f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\right) \\ &\quad + z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1} + z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}.\end{aligned} \tag{A.27}$$

The second one is slightly harder,

$$\begin{aligned}\frac{d\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}{dW_{nm}^k} &= \left\|\mathbf{z}_i^k\right\|\frac{d\sqrt{\sum_{n'}(z_{n',j}^k)^2}}{dW_{nm}^k}+\left\|\mathbf{z}_j^k\right\|\frac{d\sqrt{\sum_{n'}(z_{n',i}^k)^2}}{dW_{nm}^k}\\ &= \frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,j}^k\frac{dz_{n,j}^k}{dW_{nm}^k}+\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,i}^k\frac{dz_{n,i}^k}{dW_{nm}^k}\\ &= \frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}+\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}.\end{aligned} \tag{A.28}$$

Grouping these results together,

$$\begin{aligned}\frac{dk(\mathbf{z}_i^k,\mathbf{z}_j^k)}{dW_{nm}^k} = &-\frac{\left(z_{n,i}^k-z_{n,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}-f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\right)}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\\ &+\frac{z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}+\frac{z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\\ &-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|^2\left\|\mathbf{z}_j^k\right\|^2}\frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\\ &-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|^2\left\|\mathbf{z}_j^k\right\|^2}\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}.\end{aligned} \tag{A.29}$$

As all terms share the same divisor, we can write the last expression more concisely as

$$\begin{aligned}\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|\frac{dk(\mathbf{z}_i^k,\mathbf{z}_j^k)}{dW_{nm}^k} = &\\ &-\left(z_{n,i}^k-z_{n,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1}-f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\right)\\ &+\sum_{s=i,j}\left(1-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_s^k\right\|^2}\right)z_{n,s}^k f'\left(\mathbf{W}^k\mathbf{z}_s^{k-1}\right)_n z_{m,s}^{k-1}.\end{aligned} \tag{A.30}$$

Considering a linear network for simplicity, the weight update over two points i, j (negative of a single term in the sum in Eq. (A.16)) for the cosine similarity kernel becomes

$$\begin{aligned}\Delta W^k_{nm} &= M^k_{ij}\left(z^k_{n,i}-z^k_{n,j}\right)\left(z^{k-1}_{m,i}-z^{k-1}_{m,j}\right)\\ &\quad - M^k_{ij}\sum_{s=i,j}\left(1-\frac{(\mathbf{z}^k_i)^\top\mathbf{z}^k_j}{\left\|\mathbf{z}^k_s\right\|^2}\right)z^k_{n,s}z^{k-1}_{m,s},\\ M^k_{ij} &= \frac{1}{\left\|\mathbf{z}^k_i\right\|\left\|\mathbf{z}^k_j\right\|}\left(2\mathring{k}(\mathbf{z}^k_i,\mathbf{z}^k_j)-\gamma\mathring{k}(\mathbf{y}_i,\mathbf{y}_j)\right).\end{aligned} \tag{A.31}$$

This rule is biologically implausible (or very hard to implement) for two reasons. First, to compute M^k_{ij} the layer needs to track three signal simultaneously: $(\mathbf{z}^k_i)^\top\mathbf{z}^k_j$, $\left\|\mathbf{z}^k_i\right\|$ and $\left\|\mathbf{z}^k_j\right\|$ (versus only one for the Gaussian kernel, $\left\|\mathbf{z}^k_i-\mathbf{z}^k_j\right\|$). Second, assuming point i comes after j, the pre-factor of the Hebbian term (the sum in Eq. (A.31)) for $s=j$ can't be computed until the network receives point i. As a result, the update requires three independent plasticity pathways (one for $M^k_{ij}\left(z^k_{n,i}-z^k_{n,j}\right)\left(z^{k-1}_{m,i}-z^{k-1}_{m,j}\right)$ and two for the sum in Eq. (A.31)), because each term in Eq. (A.31) combines the pre- and post-synaptic activity on different timescales and with different third factors.

Adding grouping and divisive normalisation to this rule is straightforward: we need to use the grouped response v^k_α (Eq. (2.19)) in the kernel as $k(\mathbf{z}^k_i,\mathbf{z}^k_j)=(\mathbf{v}^k_i)^\top\mathbf{v}^k_j/(\left\|\mathbf{v}^k_i\right\|\left\|\mathbf{v}^k_j\right\|)$, and repeat the calculation above (divisive normalisation will appear here too as it results from differentiating v^k_α). As it would only make the circuitry more complicated, we omit the derivation. Note that if we use grouping with $p=0.5$ but don't introduce divisive normalisation, our objective resembles the "sim-bpf" loss in [41] (but it doesn't match it exactly, as we also introduce centring of the kernel and group convolutional layers over multiple channels rather than one).

A.2.5 Linear kernel

Derivation of the update for the linear kernel only needs the derivative from Eq. (A.26), therefore by doing the same calculations we obtain

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{nm}^k} &= \frac{d\,(\mathbf{z}_i^k)^\top \mathbf{z}_j^k}{d\,W_{nm}^k} = \frac{d\left(z_{n,i}^k z_{n,j}^k\right)}{d\,W_{nm}^k} \\ &= -\left(z_{n,i}^k - z_{n,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,i}^{k-1} - f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,j}^{k-1}\right) \\ &\quad + \sum_{s=i,\,j} z_{n,s}^k f'\left(\mathbf{W}^k\mathbf{z}_s^{k-1}\right)_n z_{m,s}^{k-1}.\end{aligned} \tag{A.32}$$

This is much easier to compute than the cosine similarity kernel: it only uses $(\mathbf{z}_i^k)^\top \mathbf{z}_j^k$ in the third factor, and needs two plasticity channels rather than three (as now both terms in the sum use the same third factor). However, we couldn't achieve good performance with this kernel.

A.3 Experimental details

A.3.1 Network architecture

Each hidden layer of the network had its own optimiser, such that weight updates happen during the forward pass.

Each layer had the following structure: linear/convolutional operation → batch-norm (if any) → nonlinearity → pooling (if any) → local loss computation (doesn't modify activity) → divisive normalisation (if any) → dropout.

None of the hidden layers had the bias term, but the output layer did. The last layer (or the whole network for backprop) was trained with the cross-entropy loss. Non-pHSIC methods with divisive normalisation used the same group arrangement, but did not have grouping in the objectives.

A.3.2 Choice of kernels for pHSIC

The gradient over each batch was computed as in Eq. (A.16).

We used cosine similarity with centred labels (Eq. (A.6); as all datasets are balanced, we don't need to know the probability of a class to centre). The kernels for $\mathbf{z}$ were plain Gaussian (Eq. (1.16)), Gaussian with grouping and divisive normalisation

("grp+div"; Eq. (2.20) such that the next layer sees $r^k_{\alpha n} \equiv \mathring{z}^k_{\alpha n}/(u^k_\alpha)^p$) and grouping without divisive normalisation ("grp"; also Eq. (2.20) but the next layer sees $z^k_{\alpha n}$), plain cosine similarity (Eq. (1.15)), and cosine similarity with grouping and with or without divisive normalisation ($k(\mathbf{z}^k_i, \mathbf{z}^k_j) = (\mathbf{v}^k_i)^\top \mathbf{v}^k_j / (\left\|\mathbf{v}^k_i\right\| \left\|\mathbf{v}^k_j\right\|)$ with $\mathbf{v}$ as in Eq. (2.19)).

A.3.3 Objective choice for layer-wise classification

As proposed in [41], each convolutional layer is first passed through an average pooling layer such that the final number of outputs is equal to 2048 (e.g. a layer with 128 channels and 32 by 32 images is pooled with an 8 by 8 kernel with stride $= 8$); the resulting 2048-dimensional vector is transformed into a 10-dimensional vector (for class prediction) by a linear readout layer. Fully connected layers are transformed directly with the corresponding linear readout layer. In the layer-wise classification with feedback alignment, feedback alignment is applied to the readout layer.

A.3.4 Pre-processing of datasets

MNIST, fashion-MNIST and Kuzushiji-MNIST images were centred by 0.5 and normalised by 0.5.

For CIFAR10, each training image was padded by zeros from all sides with width 4 (resulting in a 40 by 40 image for each channel) and randomly cropped to the standard size (32 by 32), then flipped horizontally with probability 0.5, and then centred by $(0.4914, 0.4822, 0.4465)$ (each number corresponds to a channel) and normalised by $(0.247, 0.243, 0.261)$. For validation and test, the images were only centred and normalised.

A.3.5 Shared hyperparameters for all experiments

We used the default parameters for AdamW, batchnorm, LReLU and SELU; for grouping without divisive normalisation we used $p = 0.5$ to be comparable with the objective in [41]. The rest of the parameters (including the ones below) were tuned on a validation set (10% of the training set for all datasets).

Weight decay for the local losses was 1e-7, and for the final/backprop it was 1e-6; the learning rates were multiplied by 0.25, with individual schedules described

below. For SGD, the momentum was 0.95; for AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon =$1e-8. Batchnorm had momentum of 0.1 and $\varepsilon = 1e-5$, with initial scale $\gamma = 1$ and shift $\beta = 0$. Leaky ReLU had the slope 0.01; $\text{SELU}(x) = \text{scale}(\max(0,x) + \min(0, \alpha(\exp(x) - 1)))$ had $\alpha \approx 1.6733$ and scale ≈ 1.0507 (precise values were found numerically in [14]; note that dropout for SELU was changed to alpha dropout, as proposed in [14]). All convolutions used 3 by 3 kernel with padding$= 1$ (on each side), stride$= 1$ and dilation$= 1$ and no groups; max pooling layers used 2 by 2 kernels with stride$= 2$, dilation$= 1$ and no padding. Grouping with divisive normalisation used $\delta = 1$ and $p = 0.2$ (backprop, pHSIC) or $p = 0.5$ (FA, sign symmetry, layer-wise classification), and $p = 0.5$ without divisive normalisation. Gaussian kernels used $\sigma = 5$. The balance parameter was set to $\gamma = 2$.

A.3.6 Small network

The dropout for all experiments was 0.01, with LReLU for nonlinearity. The networks were trained for 100 epochs, and the learning rates were multiplied by 0.25 at epochs 50, 75 and 90. The individual parameters, η_f for final/backprop initial learning rate, η_l for the local initial learning rate, c^k for the number of groups in the objective, are given in Table A.1; the final results are given in Table A.2 (same as Table 2.1 but with "grp") and Table A.3 (max - min accuracy).

A.3.7 Large network

The dropout for all experiments was 0.05, with LReLU for AdamW+batchnorm and SELU for SGD. The networks were trained for 500 epochs, and the learning rates were multiplied by 0.25 at epochs 300, 350, 450 and 475 (and at 100, 200, 250, 275 for backprop with SGD). The individual parameters, η_f for final/backprop initial learning rate, η_l for the local initial learning rate, c^k for the number of groups in the objective, are given in Table A.4 and Table A.5. The results are given in Table A.6 and Table A.7 (mean test accuracy) and Table A.8 and Table A.9 (max - min accuracy). The batch manhattan method mentioned in Table A.5 was proposed in [62]; it is used to stabilise feedback alignment and sign symmetry algorithms by substituting the gradient w.r.t. the loss (i.e. before adding momentum and weight

decay) with its sign for each weight update. However, in our experiments it didn't improve performance in most of the cases. Without any normalisation, we didn't find a successful set of parameters for the Gaussian kernel with grouping and for the methods with feedback alignment and sign symmetry. In those cases, the training either diverged completely or was stuck at low training and even lower test errors (e.g. around 40% training error for the Gaussian kernel with grouping, and around 80% training error for layer-wise classification with feedback alignment).

A.3.8 Difference between pHSIC and HSIC in the large network

While we reported all results for pHSIC, training with HSIC instead did not lead to a significant change in the results (not shown). Moreover, the difference between the two objectives stays small during training, as we illustrate below.

As explained in Appendix A.1.1, our objectives differ from HSIC only in the first term, $\text{pHSIC}(Z^k, Z^k)$, due to centring of labels. We trained the 1x wide networks from the previous section (grouping + divisive normalisation with SGD, grouping + batchnorm with AdamW) and plotted

$$\frac{\text{pHSIC}(Z^k, Z^k) - \text{HSIC}(Z^k, Z^k)}{\text{pHSIC}(Z^k, Z^k)} \tag{A.33}$$

as a function of training epoch. We compute this quantity on the training data, but the test data gives the same behaviour (not shown).

The results show that for both the cosine similarity and the Gaussian kernel, the relative distance between pHSIC and HSIC (Eq. (A.33)) stays small in all layers expect the first one, but even there it remains relatively constant when trained on the pHSIC objective with SGD + divisive normalisation (Fig. A.1) or AdamW + batchnorm (Fig. A.2); the same holds when the objective is HSIC (Fig. A.3 for SGD + divisive normalisation and Fig. A.4 for AdamW + batchnorm), although earlier layers have larger values when compared to pHSIC training.

Table A.1: Parameters for the 3-layer fully connected net (1024 neurons per layer). Last layer: training of the last layer; cossim: cosine similarity; grp: grouping; div: divisive normalisation.

	backprop		last layer		pHSIC: cossim			pHSIC: Gaussian		
		div		div		grp	grp+div		grp	grp+div
MNIST										
η_f	5e-2	5e-3	5e-2	5e-2	5e-3	5e-3	5e-3	5e-4	5e-4	1e-3
η_l					0.5	0.6	0.4	0.6	1.0	1.0
c^k		16		16		16	16		32	32
f-MNIST										
η_f	5e-3	5e-3	5e-2	5e-2	5e-3	1e-3	5e-4	5e-4	5e-4	5e-4
η_l					1.0	0.6	1.0	0.5	1.0	1.0
c^k		32		32		32	32		32	32
K-MNIST										
η_f	5e-2	5e-2	5e-2	5e-2	5e-3	5e-3	5e-4	1e-3	1e-3	1e-3
η_l					0.6	0.4	0.4	0.6	1.0	1.0
c^k		32		16		16	16		32	32
CIFAR10										
η_f	5e-3	5e-3	5e-2	1e-2	1e-3	5e-3	5e-3	5e-3	5e-4	1e-3
η_l		32		32	1.0	0.4	0.1	0.1	0.6	1.0
c^k						32	32		32	32

Table A.2: Mean test accuracy over 5 random seeds for a 3-layer fully connected net (1024 neurons per layer). Last layer: training of the last layer; cossim: cosine similarity; grp: grouping; div: divisive normalisation.

	backprop		last layer		pHSIC: cossim			pHSIC: Gaussian		
		grp+div		grp+div		grp	grp+div		grp	grp+div
MNIST	98.6	98.4	92.0	95.4	94.9	95.8	96.3	94.6	98.4	98.1
f-MNIST	90.2	90.8	83.3	85.7	86.3	88.7	88.1	86.5	88.6	88.8
K-MNIST	93.4	93.5	71.2	78.2	80.4	86.2	87.2	80.2	92.7	91.1
CIFAR10	60.0	60.3	39.2	38.0	51.1	52.5	47.6	41.4	48.4	46.4

Table A.3: Same as Table A.2, but max minus min test accuracy over 5 random seeds.

	backprop		last layer		pHSIC: cossim			pHSIC: Gaussian		
		grp+div		grp+div		grp	grp+div		grp	grp+div
MNIST	0.2	0.1	0.3	0.3	1.4	0.5	0.6	0.2	0.3	0.2
f-MNIST	0.2	0.4	0.3	0.2	0.6	1.1	0.3	0.2	0.6	0.2
K-MNIST	0.3	0.3	1.1	0.8	1.0	1.0	0.9	1.0	0.4	1.2
CIFAR10	0.6	0.9	1.2	1.4	1.4	2.0	1.4	0.5	1.0	0.6

Table A.4: Parameters for the 7-layer conv nets (CIFAR10; 1x and 2x wide). Cossim: cosine similarity; divnorm: divisive normalisation; bn: batchnorm. Empty entries: experiments for which we didn't find a satisfying set of parameters due to instabilities in the methods.

	backprop		pHSIC: cossim		pHSIC: Gaussian	
		div	grp	grp+div	grp	grp+div
1x wide net + SGD						
η_f	5e-3	6e-3	5e-5	5e-4		1e-4
η_l			3e-2	0.5		0.4
c^k		64	32	64		64
2x wide net + SGD						
η_f	6e-3	6e-3	5e-5	5e-4		1e-4
η_l			3e-2	0.5		0.4
c^k		64	32	64		64
1x wide net + AdamW + batchnorm						
η_f	5e-3	5e-3	5e-4	5e-4	5e-4	5e-4
η_l			5e-4	5e-4	5e-3	1e-2
c^k		64	32	64	64	64
2x wide net + AdamW + batchnorm						
η_f	5e-3	5e-3	5e-4	5e-4	5e-4	5e-4
η_l			5e-4	5e-4	5e-3	5e-3
c^k		64	128	64	128	128

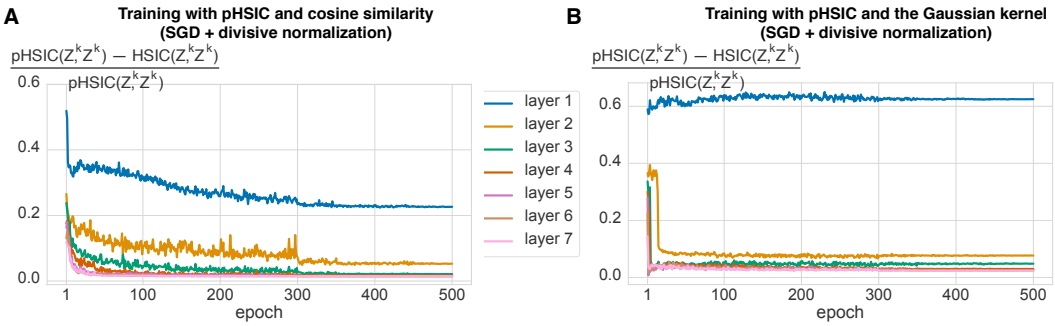

Figure A.1: Training of 1x networks with pHSIC, SGD and divisive normalisation. Y-axis represents $\left(\text{pHSIC}(Z^k,Z^k) - \text{HSIC}(Z^k,Z^k)\right)/\text{pHSIC}(Z^k,Z^k)$. **A.** Cosine similarity kernel **B.** Gaussian kernel.

Table A.5: Parameters for the 7-layer conv nets (CIFAR10; 1x and 2x wide). FA: feedback alignment; sign sym.: sign symmetry; layer class.: layer-wise classification; divnorm: divisive normalisation; bn: batchnorm. Empty entries: experiments for which we didn't find a satisfying set of parameters due to instabilities in the methods.

	FA	sign sym.	layer class.	
				+FA
1x wide net + SGD				
η_f			5e-3	
η_l			1e-3	
2x wide net + SGD				
η_f			5e-3	
η_l			1e-3	
1x wide net + SGD + divnorm				
η_f	1e-3	5e-4	5e-3	5e-3
η_l			5e-3	5e-3
c^k	64	64	64	64
batch manhattan		+		
2x wide net + SGD + divnorm				
η_f	5e-4	5e-4	5e-3	5e-3
η_l			5e-3	5e-3
c^k	128	128	64	128
batch manhattan		+		
1x wide net + AdamW + bn				
η_f	5e-4	5e-4	5e-3	5e-3
η_l			5e-4	1e-3
2x wide net + AdamW + bn				
η_f	5e-4	5e-4	5e-3	5e-3
η_l			5e-4	5e-4

Table A.6: Mean test accuracy on CIFAR10 over 5 runs for 7-layer conv nets (1x and 2x wide). Cossim: cosine similarity; divnorm: divisive normalisation; bn: batchnorm. Empty entries: experiments for which we didn't find a satisfying set of parameters due to instabilities in the methods.

	backprop		pHSIC: cossim		pHSIC: Gaussian	
		div	grp	grp+div	grp	grp+div
1x wide net + SGD	91.0	91.0	88.8	89.8		86.2
2x wide net + SGD	91.9	90.9	89.4	91.3		90.4
1x wide net + AdamW + batchnorm	94.1	94.3	91.3	90.1	89.9	89.4
2x wide net + AdamW + batchnorm	94.3	94.5	91.9	91.0	91.0	91.2

Table A.7: Mean test accuracy on CIFAR10 over 5 runs for 7-layer conv nets (1x and 2x wide). FA: feedback alignment; sign sym.: sign symmetry; layer class.: layer-wise classification; divnorm: divisive normalisation; bn: batchnorm. Empty entries: experiments for which we didn't find a satisfying set of parameters due to instabilities in the methods.

	FA	sign sym.	layer class.	
				+FA
1x wide net + SGD			90.0	
2x wide net + SGD			90.3	
1x wide net + SGD + divnorm	80.4	89.5	90.5	81.0
2x wide net + SGD + divnorm	80.6	91.3	91.3	81.2
1x wide net + AdamW + bn	82.4	93.6	92.1	90.3
2x wide net + AdamW + bn	81.6	93.9	92.1	91.1

Table A.8: Same as Table A.6, but max minus min test accuracy over 5 random seeds.

	backprop		pHSIC: cossim		pHSIC: Gaussian	
		div	grp	grp+div	grp	grp+div
1x wide net + SGD	0.4	0.4	0.7	0.7		0.9
2x wide net + SGD	0.3	0.3	2.4	0.2		0.5
1x wide net + AdamW + batchnorm	0.3	0.4	0.2	0.5	0.3	0.5
2x wide net + AdamW + batchnorm	0.5	0.3	0.3	0.3	0.4	0.5

Table A.9: Same as Table A.7, but max minus min test accuracy over 5 random seeds. *The large deviation is due to one experiment with about 85% accuracy.

	FA	sign sym.	layer class.	
				+FA
1x wide net + SGD			0.4	
2x wide net + SGD			0.1	
1x wide net + SGD + divnorm	1.3	5.5*	0.4	1.1
2x wide net + SGD + divnorm	0.9	0.4	0.1	0.9
1x wide net + AdamW + bn	0.4	0.3	0.6	0.5
2x wide net + AdamW + bn	0.9	0.4	0.1	0.4

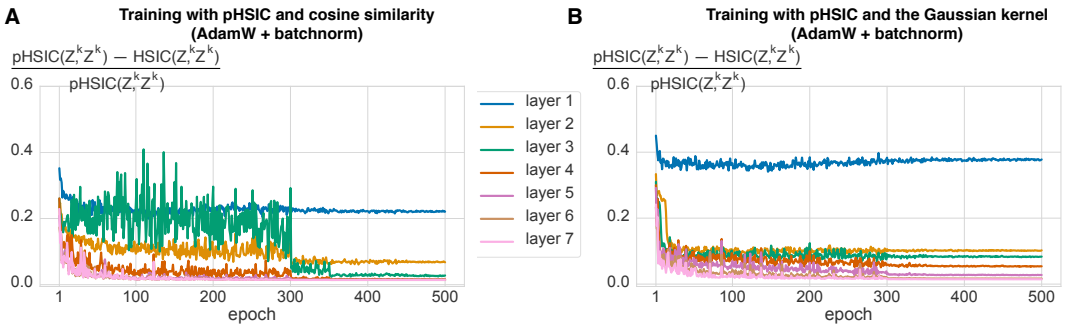

Figure A.2: Training of 1x networks with pHSIC, AdamW and batchnorm. Y-axis represents $\left(\text{pHSIC}(Z^k,Z^k) - \text{HSIC}(Z^k,Z^k)\right)/\text{pHSIC}(Z^k,Z^k)$. **A.** Cosine similarity kernel **B.** Gaussian kernel.

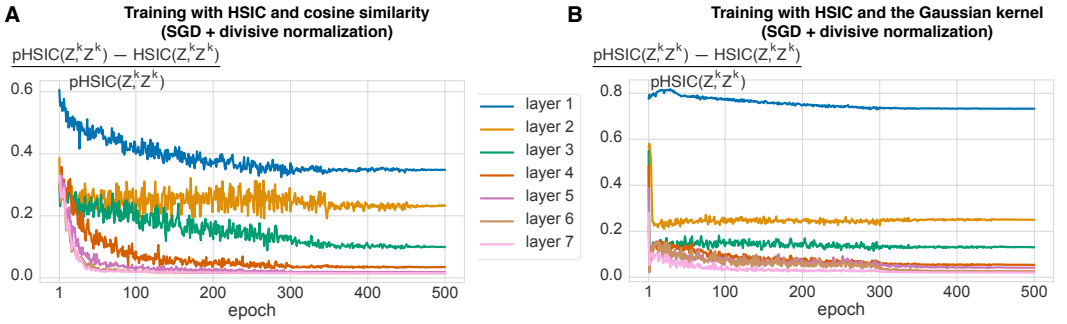

Figure A.3: Training of 1x networks with HSIC, SGD and divisive normalisation. Y-axis represents $\left(\text{pHSIC}(Z^k,Z^k) - \text{HSIC}(Z^k,Z^k)\right)/\text{pHSIC}(Z^k,Z^k)$. **A.** Cosine similarity kernel **B.** Gaussian kernel.

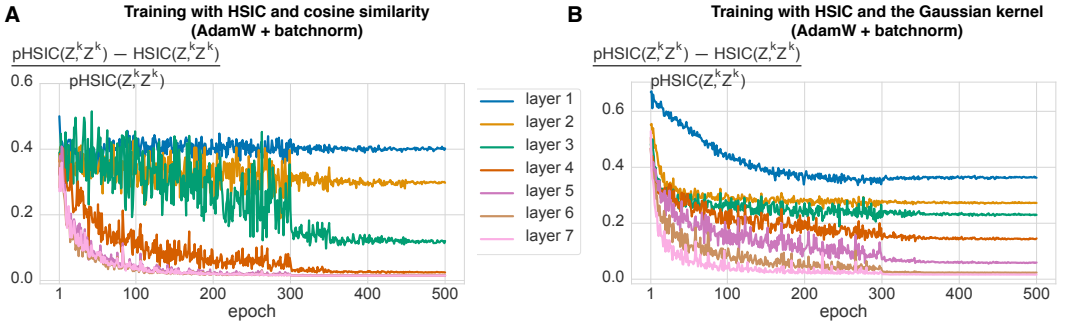

Figure A.4: Training of 1x networks with HSIC, AdamW and batchnorm. Y-axis represents $\left(\text{pHSIC}(Z^k,Z^k) - \text{HSIC}(Z^k,Z^k)\right)/\text{pHSIC}(Z^k,Z^k)$. **A.** Cosine similarity kernel **B.** Gaussian kernel.

Appendix B

Chapter 3 Appendix

B.1 HSIC estimation in the self-supervised setting

Estimators of HSIC typically assume i.i.d. data, which is not the case for self-supervised learning – the positive examples are not independent. Here we show how to adapt our estimators to the self-supervision setting.

B.1.1 Exact form of HSIC(Z, Y)

Starting with $\mathrm{HSIC}(Z,Y)$, we assume that the "label" y is a one-hot encoding of the data point, and all N data points are sampled with the same probability $1/N$. With a one-hot encoding, any kernel that is a function of $\mathbf{y}_i^\top \mathbf{y}_j$ or $\|\mathbf{y}_i - \mathbf{y}_j\|$ (e.g. linear, Gaussian or IMQ) have the form

$$l(\mathbf{y}_i, \mathbf{y}_j) = \begin{cases} l_1 & \mathbf{y}_i = \mathbf{y}_j, \\ l_0 & \text{otherwise} \end{cases} \equiv \Delta l \, \mathbb{I}(\mathbf{y}_i = \mathbf{y}_j) + l_0 \tag{B.1}$$

for some $\Delta l = l_1 - l_0$.

Theorem 1. *For a dataset with N original images sampled with probability* $1/N$*, and a kernel over image identities defined as in* (B.1)*,* $\mathrm{HSIC}(Z,Y)$ *takes the form*

$$\mathrm{HSIC}(Z,Y) = \frac{\Delta l}{N} \mathbb{E}_{Z,Z' \sim \mathrm{pos}} \left[k(Z,Z') \right] - \frac{\Delta l}{N} \mathbb{E} \left[k(Z,Z') \right] , \tag{B.2}$$

where $Z, Z' \sim \mathrm{pos}$ *means* $p_{\mathrm{pos}}(Z,Z') = \sum_i p(i) p(Z|i) p(Z'|i)$ *for image probability* $p(i) = 1/N$.

Proof. We compute HSIC (defined in (1.20)) term by term. Starting from the first, and denoting independent copies of Z, Y with Z', Y',

$$\begin{aligned}
\mathbb{E}\left[k(Z,Z')l(Y,Y')\right] &= \Delta l\,\mathbb{E}\left[k(Z,Z')\mathbb{I}[Y=Y']\right] + l_0\,\mathbb{E}\left[k(Z,Z')\right] \\
&= \Delta l \sum_{i=1}^{N}\sum_{j=1}^{N} \mathbb{E}_{Z|y_i,Z'|y_j}\left[\frac{1}{N^2}k(Z,Z')\mathbb{I}[y_i=y_j]\right] + l_0\,\mathbb{E}\left[k(Z,Z')\right] \\
&= \frac{\Delta l}{N}\sum_{i=1}^{N} \mathbb{E}_{Z|y_i,Z'|y_i}\left[\frac{1}{N}k(Z,Z')\right] + l_0\,\mathbb{E}\left[k(Z,Z')\right] \\
&= \frac{\Delta l}{N}\,\mathbb{E}_{Z,Z'\sim\text{pos}}\left[k(Z,Z')\right] + l_0\,\mathbb{E}\left[k(Z,Z')\right],
\end{aligned}$$

where $\mathbb{E}_{Z,Z'\sim\text{pos}}$ is the expectation over positive examples (with Z and Z' are sampled independently conditioned on the "label").

The second term, due to the independence between Z' and Y'', becomes

$$\mathbb{E}\left[k(Z,Z')l(Y,Y'')\right] = \mathbb{E}_{ZY}\mathbb{E}_{Z'}\left[k(Z,Z')\left(\frac{\Delta l}{N}+l_0\right)\right] = \left(\frac{\Delta l}{N}+l_0\right)\mathbb{E}\left[k(Z,Z')\right].$$

And the last term becomes identical to the second one,

$$\mathbb{E}\left[k(Z,Z')\right]\mathbb{E}\left[l(Y,Y')\right] = \left(\frac{\Delta l}{N}+l_0\right)\mathbb{E}\left[k(Z,Z')\right].$$

Therefore, we can write $\text{HSIC}(Z,Y)$ as

$$\begin{aligned}
\text{HSIC}(Z,Y) &= \mathbb{E}\left[k(Z,Z')l(Y,Y')\right] - 2\,\mathbb{E}\left[k(Z,Z')l(Y,Y'')\right] + \mathbb{E}\left[k(Z,Z')\right]\mathbb{E}\left[l(Y,Y')\right] \\
&= \frac{\Delta l}{N}\,\mathbb{E}_{Z,Z'\sim\text{pos}}\left[k(Z,Z')\right] - \frac{\Delta l}{N}\mathbb{E}\left[k(Z,Z')\right],
\end{aligned}$$

as the terms proportional to l_0 cancel each other out. □

The final form of $\text{HSIC}(Z,Y)$ shows that the Y kernel and dataset size come in only as pre-factors. To make the term independent of the dataset size (as long as it is finite), we can assume $\Delta l = N$, such that

$$\text{HSIC}(Z,Y) = \mathbb{E}_{Z,Z'\sim\text{pos}}\left[k(Z,Z')\right] - \mathbb{E}\left[k(Z,Z')\right].$$

B.1.2 Estimator of HSIC(Z, Y)

Theorem 2. *In the assumptions of Theorem 1, additionally scale the Y kernel to have $\Delta l = N$, and the Z kernel to be $k(\mathbf{z},\mathbf{z}) = 1$. Assume that the batch is sampled as follows: $B < N$ original images are sampled without replacement, and for each image M positive examples are sampled independently (i.e., the standard sampling scheme in self-supervised learning). Then denoting each data point $\mathbf{z}_i^p$ for "label" i and positive example p,*

$$\widehat{\mathrm{HSIC}}(Z,Y) = \left(\frac{M}{M-1} + \frac{N-1}{N(B-1)} - \frac{M}{N(M-1)}\right)\frac{1}{BM^2}\sum_{ipl} k(\mathbf{z}_i^p, \mathbf{z}_i^l) \tag{B.3}$$

$$- \frac{B(N-1)}{(B-1)N}\frac{1}{B^2M^2}\sum_{ijpl} k(\mathbf{z}_i^p, \mathbf{z}_j^l) - \frac{N-1}{N(M-1)}\,. \tag{B.4}$$

is an unbiased estimator of (B.2).

While we assumed that $k(\mathbf{z},\mathbf{z}) = 1$ for simplicity, any change in the scaling would only affect the constant term (which is irrelevant for gradient-based learning). Recalling that $|k(\mathbf{z},\mathbf{z}')| \le \max(k(\mathbf{z},\mathbf{z}), k(\mathbf{z}',\mathbf{z}'))$, we can then obtain a slightly biased estimator from Theorem 2 by simply discarding small terms:

Corollary 1. *If $|k(\mathbf{z},\mathbf{z}')| \le 1$ for any $\mathbf{z},\mathbf{z}'$, then*

$$\widehat{\mathrm{HSIC}}(Z,Y) = \frac{1}{BM(M-1)}\sum_{ipl} k(\mathbf{z}_i^p, \mathbf{z}_i^l) - \frac{1}{B^2M^2}\sum_{ijpl} k(\mathbf{z}_i^p, \mathbf{z}_j^l) - \frac{1}{M-1} \tag{B.5}$$

has a $O(1/B)$ bias.

Proof of Theorem 2. To derive an unbiased estimator, we first compute expectations of two sums: one over all positive examples (same i) and one over all data points.

Starting with the first,

$$\mathbb{E}\left[\frac{1}{BM^2}\sum_{ipl} k(\mathbf{z}_i^p, \mathbf{z}_i^l)\right] = \mathbb{E}\left[\frac{1}{BM^2}\sum_{ip,l\neq p} k(\mathbf{z}_i^p, \mathbf{z}_i^l)\right] + \mathbb{E}\left[\frac{1}{BM^2}\sum_{ip} k(\mathbf{z}_i^p, \mathbf{z}_i^p)\right] \tag{B.6}$$

$$= \frac{M-1}{M}\mathbb{E}_{Z,Z'\sim\mathrm{pos}}\left[k(Z,Z')\right] + \frac{1}{M}\,. \tag{B.7}$$

As for the second sum,

$$\mathbb{E}\left[\frac{1}{B^2M^2}\sum_{ijpl}k(\mathbf{z}_i^p,\mathbf{z}_j^l)\right]=\mathbb{E}\left[\frac{1}{B^2M^2}\sum_{i,j\neq i,pl}k(\mathbf{z}_i^p,\mathbf{z}_j^l)\right]+\mathbb{E}\left[\frac{1}{B^2M^2}\sum_{ipl}k(\mathbf{z}_i^p,\mathbf{z}_i^l)\right].$$

The first term is tricky: $\mathbb{E}\left[k(\mathbf{z}_i^p,\mathbf{z}_j^l)\right]\neq\mathbb{E}\left[k(Z,Z')\right]$ because we sample without replacement. But we know that $p(y,y')=p(y)p(y'|y)=1/(N(N-1))$, therefore for $i\neq j$

$$\begin{aligned}\mathbb{E}\,k(\mathbf{z}_i^p,\mathbf{z}_j^l)&=\sum_{y,y'\neq y}\frac{1}{N(N-1)}\mathbb{E}_{Z|y,Z'|y'}\,k(Z,Z') &\text{(B.8)}\\&=\sum_{yy'}\frac{1}{N(N-1)}\mathbb{E}_{Z|y,Z'|y'}\,k(Z,Z')-\sum_{y}\frac{1}{N(N-1)}\mathbb{E}_{Z|y,Z'|y}k(Z,Z') &\text{(B.9)}\\&=\frac{N}{N-1}\mathbb{E}\,k(Z,Z')-\frac{1}{N-1}\mathbb{E}_{Z,Z'\sim\text{pos}}k(Z,Z')\,. &\text{(B.10)}\end{aligned}$$

Using the expectations for ipl and $ijpl$,

$$\begin{aligned}\mathbb{E}\,\frac{1}{B^2M^2}\sum_{ijpl}k(\mathbf{z}_i^p,\mathbf{z}_j^l)&=\mathbb{E}\,\frac{1}{B^2M^2}\sum_{i,j\neq i,pl}k(\mathbf{z}_i^p,\mathbf{z}_j^l)+\mathbb{E}\,\frac{1}{B^2M^2}\sum_{ipl}k(\mathbf{z}_i^p,\mathbf{z}_j^l) &\text{(B.11)}\\&=\frac{B-1}{B(N-1)}\left(N\,\mathbb{E}\,k(Z,Z')-\mathbb{E}_{Z,Z'\sim\text{pos}}\,k(Z,Z')\right)+\frac{M-1}{BM}\mathbb{E}_{Z,Z'\sim\text{pos}}\,k(Z,Z')+\frac{1}{BM} &\text{(B.12)}\\&=\frac{(B-1)N}{B(N-1)}\mathbb{E}\,k(Z,Z')-\frac{B-1}{B(N-1)}\mathbb{E}_{Z,Z'\sim\text{pos}}\,k(Z,Z') &\text{(B.13)}\\&\quad+\frac{M-1}{BM}\mathbb{E}_{Z,Z'\sim\text{pos}}k(Z,Z')+\frac{1}{BM} &\text{(B.14)}\\&=\frac{(B-1)N}{B(N-1)}\mathbb{E}\,k(Z,Z')+\frac{1}{B}\left(\frac{M-1}{M}-\frac{B-1}{N-1}\right)\mathbb{E}_{Z,Z'\sim\text{pos}}\,k(Z,Z')+\frac{1}{BM}\,. &\text{(B.15)}\end{aligned}$$

Combining (B.6) and (B.11) shows that (B.3) is indeed an unbiased estimator.

□

It's worth noting that the i.i.d. estimator (1.24) is flawed for $\text{HSIC}(Z,Y)$ for two reasons: first, it misses the $1/N$ scaling of $\text{HSIC}(Z,Y)$ (however, it's easy to fix by rescaling); second, it misses the $1/(M(M-1))$ correction for the ipl sum. As we typically have $M=2$, the latter would result in a large bias for the (scaled) i.i.d.

estimator.

B.1.3 Estimator of HSIC(Z, Z)

Before discussing estimators of $\mathrm{HSIC}(Z,Z)$, note that it takes the following form:

$$\mathrm{HSIC}(Z,Z) = \mathbb{E}\left[k(Z,Z')^2\right] - 2\mathbb{E}_Z\left[\mathbb{E}_{Z'}\left[k(Z,Z')\right]^2\right] + \left(\mathbb{E}\left[k(Z,Z')\right]\right)^2.$$

This is because X and Y in $\mathrm{HSIC}(X,Y)$ become the *same* random variable, so $p(X,Y) = p_Z(X)\delta(X-Y)$ (see [26], Appendix A).

Theorem 3. *Assuming $k(\mathbf{z},\mathbf{z}') \leq 1$ for any $\mathbf{z},\mathbf{z}'$, the i.i.d. HSIC estimator by [32],*

$$\widehat{\mathrm{HSIC}}(Z,Z) = \frac{1}{(BM-1)^2}\mathrm{Tr}\left(KHKH\right),$$

where $H = I - \frac{1}{BM}\mathbf{1}\mathbf{1}^\top$, has a $O(1/B)$ bias for the self-supervised sampling scheme.

Proof. First, observe that

$$\mathrm{Tr}\left(KHKH\right) = \mathrm{Tr}\left(KK\right) - \frac{2}{BM}\mathbf{1}^\top KK\mathbf{1} + \frac{1}{B^2M^2}\left(\mathbf{1}^\top K\mathbf{1}\right)^2.$$

Starting with the first term, and using again the result of (B.8) for sampling without replacement,

$$\begin{aligned}
\mathbb{E}\left[\mathrm{Tr}\left(KK\right)\right] &= \mathbb{E}\left[\sum_{ijpl} k(\mathbf{z}_i^p,\mathbf{z}_j^l)^2\right] = \mathbb{E}\left[\sum_{i,j\neq i,pl} k(\mathbf{z}_i^p,\mathbf{z}_j^l)^2\right] + \mathbb{E}\left[\sum_{ipl} k(\mathbf{z}_i^p,\mathbf{z}_j^l)^2\right] \\
&= \frac{B(B-1)M^2}{N-1}\left(N\,\mathbb{E}\,k(Z,Z')^2 - \mathbb{E}_{Z,Z'\sim\mathrm{pos}}k(Z,Z')^2\right) + \mathbb{E}\sum_{ipl} k(\mathbf{z}_i^p,\mathbf{z}_j^l)^2 \\
&= B^2M^2\,\mathbb{E}\,k(Z,Z')^2 + O(BM^2).
\end{aligned}$$

Similarly, the expectation of the second term is

$$\begin{aligned}
\mathbb{E}\,\mathbf{1}^\top KK\mathbf{1} &= \mathbb{E}\sum_{ijqpld} k(\mathbf{z}_i^p,\mathbf{z}_q^d)k(\mathbf{z}_j^l,\mathbf{z}_q^d) \\
&= \mathbb{E}\sum_{i,j\neq i,q\neq\{i,j\},pld} k(\mathbf{z}_i^p,\mathbf{z}_q^d)k(\mathbf{z}_j^l,\mathbf{z}_q^d) + O(B^2M^3).
\end{aligned}$$

Here we again need to take sampling without replacement into account, and again it will produce a very small correction term. For $i \neq j \neq q$, repeating the calculation in (B.8),

$$\begin{aligned}\mathbb{E}\, k(\mathbf{z}_i^p, \mathbf{z}_j^l) k(\mathbf{z}_i^p, \mathbf{z}_q^d) &= \sum_{y,y' \neq y,y'' \neq \{y,y'\}} \frac{1}{N(N-1)(N-2)} \mathbb{E}_{Z|y,Z'|y',Z''|y''}\, k(Z,Z')k(Z,Z'') \\ &= \mathbb{E}\, k(Z,Z')k(Z,Z'') + O(1/N)\,.\end{aligned}$$

As $B < N$, we obtain that

$$\mathbb{E}\, \mathbf{1}^\top K K \mathbf{1} = B(B-1)(B-2)M^3\, \mathbb{E}\, k(Z,Z')k(Z,Z'') + O(B^2M^3)\,.$$

Finally, repeating the same argument for sampling without replacement,

$$\begin{aligned}\mathbb{E}\left(\mathbf{1}^\top K \mathbf{1}\right)^2 &= \mathbb{E} \sum_{ijqrpldf} k(\mathbf{z}_i^p, \mathbf{z}_j^l) k(\mathbf{z}_q^d, \mathbf{z}_r^f) \\ &= \mathbb{E} \sum_{i,j \neq i,q \neq \{i,j\}, r \neq \{i,j,q\}, pldf} k(\mathbf{z}_i^p, \mathbf{z}_j^l) k(\mathbf{z}_q^d, \mathbf{z}_r^f) + O(B^3M^4) \\ &= B(B-1)(B-2)(B-3)M^4\, \mathbb{E}\, k(Z,Z')k(Z'',Z''') + O(B^3M^4)\,.\end{aligned}$$

Combining all terms together, and expressing $B(B-1)$ (and similar) terms in big-O notation,

$$\begin{aligned}\mathbb{E}\frac{\mathrm{Tr}\,(KHKH)}{(BM-1)^2} &= \mathbb{E}\left(k(Z,Z')^2 - 2k(Z,Z')k(Z,Z'') + k(Z,Z')k(Z'',Z'''')\right) + O\left(\frac{1}{B}\right) \\ &= \mathbb{E}\, k(Z,Z')^2 - 2\mathbb{E}_Z\left(\mathbb{E}_{Z'} k(Z,Z')\right)^2 + \left(\mathbb{E}\, k(Z,Z')\right)^2 + O\left(\frac{1}{B}\right) \\ &= \mathrm{HSIC}(Z,Z) + O\left(\frac{1}{B}\right)\,.\end{aligned}$$

□

Essentially, having M positive examples for the batch size of BM changes the bias from $O(1/(BM))$ (i.i.d. case) to $O(1/B)$. Finally, note that even if $\widehat{\mathrm{HSIC}}(Z,Z)$ is unbiased, its square root is not.

B.2 Theoretical properties of SSL-HSIC

B.2.1 InfoNCE connection

To establish the connection with InfoNCE, define it in terms of expectations:

$$L_{\text{InfoNCE}}(\theta) = \mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[\exp\left(k(Z,Z')\right)\right]\right] - \mathbb{E}_{Z,Z'\sim\text{pos}}\left[k(Z,Z')\right] . \tag{B.16}$$

To clarify the reasoning in the main text, we can Taylor expand the exponent in (B.16) around $\mu_1 \equiv \mathbb{E}_{Z'}\left[k(Z,Z')\right]$. For $k(Z,Z') \approx \mu_1$,

$$\begin{aligned}
&\mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[\exp\left(k(Z,Z')\right)\right]\right] \\
&\approx \mathbb{E}_Z\left[\mu_1\right] + \mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[1 + k(Z,Z') - \mu_1 + \frac{(k(Z,Z')-\mu_1)^2}{2}\right]\right] \\
&= \mathbb{E}_Z\left[\mu_1\right] + \mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[1 + \frac{(k(Z,Z')-\mu_1)^2}{2}\right]\right] .
\end{aligned}$$

Now expanding $\log(1+x)$ around zero,

$$\begin{aligned}
\mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[\exp\left(k(Z,Z')\right)\right]\right] &\approx \mathbb{E}_Z\left[\mu_1\right] + \mathbb{E}_Z\mathbb{E}_{Z'}\left[\frac{(k(Z,Z')-\mu_1)^2}{2}\right] \\
&= \mathbb{E}_Z\mathbb{E}_{Z'}\left[k(Z,Z')\right] + \frac{1}{2}\mathbb{E}_Z\left[\mathbb{V}\text{ar}_{Z'}\left[k(Z,Z')\right]\right] .
\end{aligned}$$

The approximate equality relates to expectations over higher order moments, which are dropped. The expression gives the required intuition behind the loss, however: when the variance in $k(Z,Z')$ w.r.t. Z' is small, InfoNCE combines $-\text{HSIC}(Z,Y)$ and a variance-based penalty. In general, we can always write (assuming $\Delta l = N$ in $\text{HSIC}(Z,Y)$ as before)

$$L_{\text{InfoNCE}}(\theta) = -\text{HSIC}(Z,Y) + \mathbb{E}_Z\left[\log \mathbb{E}_{Z'}\left[\exp\left(k(Z,Z') - \mu_1\right)\right]\right] . \tag{B.17}$$

In the small variance regime, InfoNCE also bounds an HSIC-based loss. To show this, we will need a bound on $\exp(x)$:

Lemma 1. *For* $0 < \alpha \leq 1/4$ *and* $x \geq -\left(1+\sqrt{1-4\alpha}\right)/(2\alpha)$,

$$\exp(x) \geq 1 + x + \alpha x^2 . \tag{B.18}$$

Proof. The quadratic equation $1 + x + \alpha x^2$ has two roots ($x_1 \leq x_2$):

$$x_{1,2} = \frac{\pm\sqrt{1-4\alpha} - 1}{2\alpha} .$$

Both roots are real, as $\alpha \leq 1/4$. Between x_1 and x_2, (B.18) holds trivially as the rhs is negative.

For $x \geq -2$ and $\alpha \leq 1/4$,

$$\exp(x) \geq 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} \geq 1 + x + \alpha x^2 .$$

The first bound always holds; the second follows from

$$\frac{x}{3!} + \frac{x^2}{4!} + \frac{x^3}{5!} \geq \alpha - \frac{1}{2} ,$$

as the lhs is monotonically increasing and equals $-7/30$ at $x = -2$. The rhs is always smaller than $-1/4 < -7/30$. As $x_2 \geq -2$ (due to $\alpha \leq 1/4$), (B.18) holds for all $x \geq x_1$. □

We can now lower-bound InfoNCE:

Theorem 4. *Assuming that the kernel over* Z *is bounded as* $|k(\mathbf{z}, \mathbf{z}')| \leq k^{\max}$ *for any* $\mathbf{z}, \mathbf{z}'$*, and the kernel over* Y *satisfies* $\Delta l = N$ *(defined in* (B.1)*). Then for* γ *satisfying* $\min\{-2, -2k^{\max}\} = -\left(1+\sqrt{1-4\gamma}\right)/(2\gamma)$,

$$-\operatorname{HSIC}(Z,Y) + \gamma \operatorname{HSIC}(Z,Z) \leq L_{\text{InfoNCE}}(\theta) - \mathbb{E}_Z \frac{\left(\gamma \mathbb{V}\text{ar}_{Z'}\left[k(Z,Z')\right]\right)^2}{1 + \gamma \mathbb{V}\text{ar}_{Z'}\left[k(Z,Z')\right]} .$$

Proof. As we assumed the kernel is bounded, for $k(Z,Z') - \mu_1 \geq -2k^{\max}$ (almost surely; the factor of 2 comes from centring by μ_1). Now if we choose γ that satisfies $\min\{-2, -2k^{\max}\} = -\left(1+\sqrt{1-4\gamma}\right)/(2\gamma)$ (the minimum is to apply our bound

even for $k^{\max} < 1$), then (almost surely) by Lemma 1,

$$\exp\left(k(Z,Z') - \mu_1\right) \geq 1 + k(Z,Z') - \mu_1 + \gamma\left(k(Z,Z') - \mu_1\right)^2 .$$

Therefore, we can take the expectation w.r.t. Z', and obtain

$$\mathbb{E}_Z \log \mathbb{E}_{Z'} \exp\left(k(Z,Z') - \mu_1\right) \geq \mathbb{E}_Z \log\left(1 + \gamma \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right]\right) .$$

Now we can use that $\log(1+x) \geq x/(1+x) = x - x^2/(1+x)$ for $x > -1$, resulting in

$$\mathbb{E}_Z \log \mathbb{E}_{Z'} \exp\left(k(Z,Z') - \mu_1\right) \geq \gamma \mathbb{E}_Z \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right] + \mathbb{E}_Z \frac{\left(\gamma \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right]\right)^2}{1 + \gamma \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right]} .$$

Using (B.17), we obtain that

$$\begin{aligned} L_{\text{InfoNCE}}(\theta) &= -\mathrm{HSIC}(Z,Y) + \mathbb{E}_Z \log \mathbb{E}_{Z'} \exp\left(k(Z,Z') - \mu_1\right) \\ &\geq -\mathrm{HSIC}(Z,Y) + \gamma \mathbb{E}_Z \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right] + \mathbb{E}_Z \frac{\left(\gamma \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right]\right)^2}{1 + \gamma \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right]} . \end{aligned}$$

Finally, noting that by Cauchy-Schwarz

$$\begin{aligned} \mathrm{HSIC}(Z,Z) &= \mathbb{E}_{Z,Z'} k(Z,Z')^2 - 2\,\mathbb{E}_Z\left(\mathbb{E}_{Z'} k(Z,Z')\right)^2 + \left(\mathbb{E}_{Z,Z'} k(Z,Z')\right)^2 \\ &\leq \mathbb{E}_{Z,Z'} k(Z,Z')^2 - \mathbb{E}_Z\left(\mathbb{E}_{Z'} k(Z,Z')\right)^2 = \mathbb{E}_Z \mathbb{V}\mathrm{ar}_{Z'}\left[k(Z,Z')\right] , \end{aligned}$$

we get the desired bound. □

Theorem 4 works for any bounded kernel, because $-\left(1+\sqrt{1-4\gamma}\right)/(2\gamma)$ takes values in $(\infty, -2]$ for $\gamma \in (0, 1/4]$. For inverse temperature-scaled cosine similarity kernel $k(\mathbf{z},\mathbf{z}') = \mathbf{z}^\top \mathbf{z}'/(\tau\|\mathbf{z}\|\|\mathbf{z}'\|)$, we have $k^{\max} = 1/\tau$. For $\tau = 0.1$ (used in SimCLR [92]), we get $\gamma = 0.0475$. For the Gaussian and the IMQ kernels, $k(\mathbf{z},\mathbf{z}') \geq -\mu_1 \geq -1$, so we can replace γ with $\frac{1}{3}$ due to the following inequality: for

$x \geq a$,

$$\exp(x) \geq 1 + x + \frac{x^2}{2} + \frac{x^3}{6} \geq 1 + x + \frac{a+3}{6}x^2,$$

where the first inequality is always true.

B.2.2 MMD interpretation of HSIC(X,Y)

The special label structure of the self-supervised setting allows us to understand $\mathrm{HSIC}(X,Y)$ in terms of the maximum mean discrepancy (MMD). Denoting labels as i and j and corresponding mean feature vectors (in the RKHS) as μ_i and μ_j,

$$\mathrm{MMD}^2(i,j) = \|\mu_i - \mu_j\|^2 = \langle \mu_i, \mu_i \rangle + \langle \mu_j, \mu_j \rangle - 2 \langle \mu_i, \mu_j \rangle .$$

Therefore, the average over all labels becomes

$$\begin{aligned}
\frac{1}{N^2}\sum_{ij}\mathrm{MMD}^2(i,j) &= \frac{2}{N}\sum_i \langle \mu_i, \mu_i \rangle - \frac{2}{N^2}\sum_{ij}\langle \mu_i, \mu_j \rangle \\
&= \frac{2}{N}\sum_i \langle \mu_i, \mu_i \rangle - \frac{2}{N^2}\left\langle \sum_i \mu_i, \sum_j \mu_j \right\rangle \\
&= 2\mathbb{E}_i \mathbb{E}_{Z|i,Z'|i} \left\langle \phi(Z), \phi(Z') \right\rangle - 2 \left\langle \mathbb{E}_i \mathbb{E}_{Z|i}\phi(Z), \mathbb{E}_j \mathbb{E}_{Z'|j}\phi(Z') \right\rangle \\
&= 2\mathbb{E}_{Z,Z' \sim \mathrm{pos}} \left[k(Z,Z') \right] - 2\mathbb{E}_Z \mathbb{E}_{Z'} \left[k(Z,Z') \right] ,
\end{aligned}$$

where the second last line uses that all labels have the same probability $1/N$, and the last line takes the expectation out of the dot product and uses $k(Z,Z') = \langle \phi(Z), \phi(Z') \rangle$.

Therefore,

$$\frac{1}{2N^2}\sum_{ij}\mathrm{MMD}^2(i,j) = \frac{N}{\Delta l}\mathrm{HSIC}(Z,Y).$$

B.3 Random Fourier Features (RFF)

B.3.1 Basics of RFF

Random Fourier features were introduced by [120] to reduce computational complexity of kernel methods. Briefly, for translation-invariant kernels $k(\mathbf{z}-\mathbf{z}')$ that satisfy $k(0)=1$, Bochner's theorem gives that

$$k(\mathbf{z}-\mathbf{z}') = \int p(\omega) e^{i\omega^\top(\mathbf{z}-\mathbf{z}')} d^n\omega = \mathbb{E}_\omega\left[e^{i\omega^\top \mathbf{z}}\left(e^{i\omega^\top \mathbf{z}'}\right)^*\right],$$

where the probability distribution $p(\omega)$ is the n-dimensional Fourier transform of $k(\mathbf{z}-\mathbf{z}')$.

As both the kernel and $p(\omega)$ are real-valued, we only need the real parts of the exponent. Therefore, for $b \sim \text{Uniform}[0, 2\pi]$,

$$k(\mathbf{z}-\mathbf{z}') = \mathbb{E}_{\omega,b}\left[2\cos(\omega^\top \mathbf{z}+b)\cos(\omega^\top \mathbf{z}'+b)\right].$$

For N data points, we can draw D ω_d from $p(\omega)$, construct RFF for each points $\mathbf{z}_i$, put them into matrix $R \in \mathbb{R}^{N\times D}$, and approximate the kernel matrix as

$$K \approx RR^\top,\ R_{id} = \sqrt{\frac{2}{D}}\cos(\omega_d^\top \mathbf{z}_i + b),$$

and $\mathbb{E}\, RR^\top = K$.

For the Gaussian kernel, $k(\mathbf{z}-\mathbf{z}') = \exp(-\|\mathbf{z}-\mathbf{z}'\|^2/2)$, we have $p(\omega) = (2\pi)^{-D/2}\exp(-\|\omega\|^2/2)$ [120]. We are not aware of literature on RFF representation for the inverse multiquadratic (IMQ) kernel; we derive it below using standard methods.

B.3.2 RFF for the IMQ kernel

Theorem 5. *For the inverse multiquadratic (IMQ) kernel,*

$$k(\mathbf{z},\mathbf{z}') \equiv k(\mathbf{z}-\mathbf{z}') = \frac{c}{\sqrt{c^2+\|\mathbf{z}-\mathbf{z}'\|^2}},$$

the distribution of random Fourier features $p(\omega)$ is proportional to the following (for $s = \|w\|$),

$$p(\omega) \equiv \hat{h}(s) \propto \frac{K_{\frac{n-2}{2}+\frac{1}{2}}(cs)}{s^{\frac{n-2}{2}+\frac{1}{2}}} = \frac{K_{\frac{n-1}{2}}(cs)}{s^{\frac{n-1}{2}}}, \tag{B.19}$$

where K_ν is the modified Bessel function (of the second kind) of order ν.

Proof. To find the random Fourier features, we need to take the Fourier transform of this kernel,

$$\hat{k}(\omega) = \int e^{-i\omega^\top \mathbf{z}} k(\mathbf{z}) d^n \mathbf{z}.$$

As the IMQ kernel is radially symmetric, meaning that $k(\mathbf{z}, \mathbf{z}') = h(r)$ for $r = \|\mathbf{z} - \mathbf{z}'\|$, its Fourier transform can be written in terms of the Hankel transform [196, Section B.5] (with $\|\omega\| = s$)

$$\hat{k}(\omega) = \hat{h}(s) = \frac{(2\pi)^{n/2}}{s^{\frac{n-2}{2}}} H_{\frac{n-2}{2}}\left[r^{\frac{n-2}{2}} h(r)\right](s).$$

The Hankel transform of order ν is defined as

$$H_\nu[g(t)](s) = \int_0^\infty J_\nu(st) g(t) t \, dt,$$

where $J_\nu(s)$ is the Bessel function (of the first kind) of order ν.

As $h(r) = c/\sqrt{c^2 + r^2}$,

$$H_{\frac{n-2}{2}}\left[r^{\frac{n-2}{2}} h(r)\right](s) = c \frac{\sqrt{2} c^{\frac{n-2}{2}+1/2}}{\sqrt{s}\Gamma(\frac{1}{2})} K_{\frac{n-2}{2}+\frac{1}{2}}(cs),$$

where K_ν is a modified Bessel function (of the second kind) of order ν.

Therefore, by using a table of Hankel transforms [197, Chapter 9, Table 9.2],

$$\hat{h}(s) \propto \frac{K_{\frac{n-2}{2}+\frac{1}{2}}(cs)}{s^{\frac{n-2}{2}+\frac{1}{2}}} = \frac{K_{\frac{n-1}{2}}(cs)}{s^{\frac{n-1}{2}}}.$$

□

B.3.2.1 How to sample

To sample random vectors from (B.19), we can first sample their directions as uniformly distributed unit vectors $\mathbf{d}/\|\mathbf{d}\|$, and then their amplitudes s from $\hat{h}(s)s^{n-1}$ (the multiplier comes from the change to spherical coordinates).

Sampling unit vectors is easy, as for $\mathbf{d} \sim \mathcal{N}(0, I)$, $\mathbf{d}/\|\mathbf{d}\|$ is a uniformly distributed unit vector.

To sample the amplitudes, we numerically evaluate

$$\tilde{p}(s) = \hat{h}(s)s^{n-1} = K_{\frac{n-1}{2}}(cs)s^{\frac{n-1}{2}} \tag{B.20}$$

on a grid, normalise it to get a valid probability distribution, and sample from this approximation. As for large orders K_ν attains very large numbers, we use mpmath [198], an arbitrary precision floating-point arithmetic library for Python. As we only need to sample $\tilde{p}(s)$ once during training, this adds a negligible computational overhead.

Finally, note that for any IMQ bias c, we can sample s from (B.20) for $c = 1$, and then use $\tilde{s} = s/c$ to rescale the amplitudes. This is because

$$P(s/c \leq x) = P(s \leq cx) = C\int_0^{cx} K_{\frac{n-1}{2}}(t)t^{\frac{n-1}{2}}dt = Cc^{\frac{n-1}{2}}\int_0^{x} K_{\frac{n-1}{2}}(c\tilde{t})\tilde{t}^{\frac{n-1}{2}}cd\tilde{t}\,.$$

In practice, we evaluate $\tilde{p}(s)$ for $c = 1$ on a uniform grid over $[10^{-12}, 100]$ with 10^4 points, and rescale for other c (for output dimensions of more that $D = 128$, the grid endpoint is 120 for $D \geq 1024$, 150 for $D \geq 2048$ and 200 for $D \geq 4096$.).

B.3.3 RFF for SSL-HSIC

To apply RFF to SSL-HSIC, we will discuss the HSIC(Z, Y) and the HSIC(Z, Z) terms separately. We will use the following notation:

$$k(\mathbf{z}_i^p, \mathbf{z}_j^l) \approx \sum_{d=1}^{D} r_d^{ip} r_d^{jl}$$

for D-dimensional RFF $\mathbf{r}^{ip}$ and $\mathbf{r}^{jl}$.

Starting with the first term, we can rewrite (B.5) as

$$\begin{aligned}\widehat{\mathrm{HSIC}}(Z,Y)_{\mathrm{RFF}} &= \frac{1}{BM(M-1)}\sum_{ipld} r_d^{ip} r_d^{il} - \frac{1}{B^2M^2}\sum_{ijpld} r_d^{ip} r_d^{jl} - \frac{1}{M-1} \\ &= \frac{1}{BM(M-1)}\sum_{id}\left(\sum_p r_d^{ip}\right)^2 - \frac{1}{B^2M^2}\sum_d\left(\sum_{ip} r_d^{ip}\right)^2 - \frac{1}{M-1}\,.\end{aligned}$$

The last term in the equation above is why we use RFF: instead of computing $\sum_{ijpl}$ in $O(B^2M^2)$ operations, we compute $\sum_{ip}$ in $O(BM)$ and then sum over d, resulting in $O(BMD)$ operations (as we use large batches, typically $BM > D$). As $\mathrm{HSIC}(Z,Y)$ is linear in k, $\mathbb{E}_{\omega,b}\widehat{\mathrm{HSIC}}(Z,Y)_{\mathrm{RFF}} = \widehat{\mathrm{HSIC}}(Z,Y)$.

To estimate $\widehat{\mathrm{HSIC}}(Z,Z)$, we need to sample RFF twice. This is because

$$\widehat{\mathrm{HSIC}}(Z,Z) = \frac{1}{(BM-1)^2}\mathrm{Tr}\left(KHKH\right),$$

therefore we need the first K to be approximated by $RR^\top$, and the second – by an independently sampled $\tilde{R}\tilde{R}^\top$. This way, we will have $\mathbb{E}_{\omega,b,\tilde{\omega},\tilde{b}}\widehat{\mathrm{HSIC}}(Z,Z)_{\mathrm{RFF}} = \widehat{\mathrm{HSIC}}(Z,Z)$.

Therefore, we have (noting that $HH = H$)

$$\begin{aligned}\widehat{\mathrm{HSIC}}(Z,Z)_{\mathrm{RFF}} &= \frac{1}{(BM-1)^2}\mathrm{Tr}\left(RR^\top H\tilde{R}\tilde{R}^\top H\right) = \frac{1}{(BM-1)^2}\|R^\top H\tilde{R}\|_F^2 \\ &= \frac{1}{(BM-1)^2}\|R^\top HH\tilde{R}\|_F^2 \\ &= \frac{1}{(BM-1)^2}\sum_{d_1,d_2}\left(\sum_{ip}\left(r_{d_1}^{ip} - \frac{1}{BM}\sum_{jl} r_{d_1}^{jl}\right)\left(r_{d_2}^{ip} - \frac{1}{BM}\sum_{jl} r_{d_2}^{jl}\right)\right)^2.\end{aligned}$$

To summarise the computational complexity of this approach, computing D random Fourier features for a Q-dimensional $\mathbf{z}$ takes $O(DQ)$ operations (sampling $D \times K$ Gaussian vector, normalising it, sampling D amplitudes, computing $\omega_d^\top z$ D times), therefore $O(BMDQ)$ for BM points. After that, computing $\mathrm{HSIC}(Z,Y)$ takes $O(BMD)$ operations, and $\mathrm{HSIC}(Z,Z)$ – $O(BMD^2)$ operations. The resulting complexity per batch is $O(BMD(Q+D))$. Note that we sample new features every batch.

In contrast, computing SSL-HSIC directly would cost $O(Q)$ operations per entry of K, resulting in $O((BM)^2Q)$ operations. Computing HSIC would then be quadratic in batch size, and the total complexity would stay $O((BM)^2Q)$.

In the majority of experiments, $B = 4096$, $M = 2$, $Q = 128$ and $D = 512$, and the RFF approximation performs faster (with little change in accuracy; see Table 3.8b).

B.4 Experiment Details

Note: the experimental results are due to Yazhe Li. They are included here for completeness. I have conducted similar experiments on a smaller scale during this project, but I didn't have access to the computational resources needed for the final experiments.

B.4.1 ImageNet Pretraining

B.4.1.1 Data augmentation

We follow the same data augmentation scheme as BYOL [93] with exactly the same parameters. For completeness, we list the augmentations applied and parameters used:

- random cropping: randomly sample an area of 8% to 100% of the original image with an aspect ratio logarithmically sampled from $3/4$ to $4/3$. The cropped image is resized to 224×224 with bicubic interpolation;
- flip: optionally flip the image with a probability of 0.5;
- color jittering: adjusting brightness, contrast, saturation and hue in a random order with probabilities 0.8, 0.4, 0.4, 0.2 and 0.1 respectively;
- color dropping: optionally converting to grayscale with a probability of 0.2;
- Gaussian blurring: Gaussian kernel of size 23×23 with a standard deviation uniformly sampled over $[0.1, 2.0]$;
- solarization: optionally apply color transformation $x \mapsto x \cdot 1_{x<0.5} + (1-x) \cdot 1_{x \geq 0.5}$ for pixels with values in $[0, 1]$. Solarization is only applied for the second view, with a probability of 0.2.

B.4.1.2 Optimizing kernel parameters

Since we use radial basis function kernels, we can express the kernel $k(s)$ in term of the distance $s = \|z_i - z_j\|^2$. The entropy of the kernel distance $k_\sigma(s_{ij})$ can be expressed as follows:

$$\begin{aligned} H[k] &= -\int p(k)\log p(k)dk \\ &= -\int q(s)\log\left(q(s)\left|\frac{ds}{dk}\right|\right)ds \\ &= H[s] + \int q(s)\log\left|\frac{dk}{ds}\right|ds \\ &= \mathbb{E}\left[\log|k'_\sigma(s)|\right] + \text{const} \\ &\propto \mathbb{E}\left[\log|k'_\sigma(s)|^2\right] + \text{const.} \end{aligned}$$

We use the kernel distance entropy to automatically tune kernel parameters: for the kernel parameter σ, we update it to maximize $\mathbb{E}\left[\log|k'_\sigma|^2\right]$ (for IMQ, we optimize the bias c) at every batch. This procedure makes sure the kernel remains sensitive to data variations as representations move closer to each other.

B.4.2 Evaluations

B.4.2.1 ImageNet linear evaluation protocol

After pretraining with SSL-HSIC, we retain the encoder weights and train a linear layer on top of the frozen representation. The original ImageNet training set is split into a training set and a local validation set with 10000 data points. We train the linear layer on the training set. Spatial augmentations are applied during training, i.e., random crops with resizing to 224×224 pixels, and random flips. For validation, images are resized to 256 pixels along the shorter side using bicubic resampling, after which a 224×224 center crop is applied. We use `SGD` with Nesterov momentum and train over 90 epochs, with a batch size of 4096 and a momentum of 0.9. We sweep over learning rate and weight decay and choose the hyperparameter with top-1 accuracy on local validation set. With the best hyperparameter setting, we report the final performance on the original ImageNet validation set.

B.4.2.2 ImageNet semi-supervised learning protocol

We use ImageNet 1% and 10% datasets as SimCLR [92]. During training, we initialize the weights to the pretrained weights, then fine-tune them on the ImageNet subsets. We use the same training procedure for augmentation and optimization as the linear evaluation protocol.

B.4.2.3 Linear evaluation protocol for other classification datasets

We use the same dataset splits and follow the same procedure as BYOL [93] to evaluate classification performance on other datasets, i.e. 12 natural image datasets and Pascal VOC 2007. The frozen features are extracted from the frozen encoder. We learn a linear layer using logistic regression in `sklearn` with l2 penalty and `LBFGS` for optimization. We use the same local validation set as BYOL [93] and tune hyperparameter on this local validation set. Then, we train on the full training set using the chosen weight of the l2 penalty and report the final result on the test set.

B.4.2.4 Fine-tuning protocol for other classification datasets

Using the same dataset splits described in Sec. B.4.2.3, we initialize the weights of the network to the pretrained weights and fine-tune on various classification tasks. The network is trained using `SGD` with Nestrov momentum for 20000 steps. The momentum parameter for the batch normalization statistics is set to $\max(1-10/s, 0.9)$ where s is the number of steps per epoch. We sweep the weight decay and learning rate, and choose hyperparameters that give the best score on the local validation set. Then we use the selected weight decay and learning rate to train on the whole training set to report the test set performance.

B.4.2.5 Transfer to semantic segmentation

In semantic segmentation, the goal is to classify each pixel. The head architecture is a fully-convolutional network (FCN)-based [199] architecture as [93, 112]. We train on the `train_aug2012` set and report results on `val2012`. Hyperparameters are selected on a 2119 images, which is the same held-out validation set as [93]. A standard per-pixel softmax cross-entropy loss is used to train the FCN. Training uses random scaling (by a ratio in $[0.5, 2.0]$), cropping (crop size 513), and horizontal

flipping for data augmentation. Testing is performed on the $[513, 513]$ central crop. We train for 30000 steps with a batch size of 16 and weight decay 10^{-4}. We sweep the base learning rate with local validation set. We use the best learning rate to train on the whole training set and report on the test set. During training, the learning rate is multiplied by 0.1 at the $70th$ and $90th$ percentile of training. The final result is reported with the average of 5 seeds.

B.4.2.6 Transfer to depth estimation

The network is trained to predict the depth map of a given scene. We use the same setup as BYOL [93] and report it here for completeness. The architecture is composed of a ResNet-50 backbone and a task head which takes the *conv*5 features into 4 upsampling blocks with respective filter sizes 512, 256, 128, and 64. Reverse Huber loss function is used for training. The frames are down-sampled from $[640, 480]$ by a factor 0.5 and center-cropped to size $[304, 228]$. Images are randomly flipped and color transformations are applied: greyscale with a probability of 0.3; brightness adjustment with a maximum difference of 0.1255; saturation with a saturation factor randomly picked in the interval $[0.5, 1.5]$; hue adjustment with a factor randomly picked in the interval $[-0.2, 0.2]$. We train for 7500 steps with batch size 256, weight decay 0.001, and learning rate 0.05.

B.4.2.7 Transfer to object detection

We follow the same setup for evaluating COCO object detection tasks as in DetCon [200]. The architecture used is a Mask-RCNN [201] with feature pyramid networks [202]. During training, the images are randomly flipped and resized to $(1024 \cdot s) \times (1024 \cdot s)$ where $s \in [0.8, 1.25]$. Then the resized image is cropped or padded to a 1024×1024. We fine-tune the model for 12 epochs ($1\times$ schedule [112]) with `SGD` with momentum with a learning rate of 0.3 and momumtem 0.9. The learning rate increases linearly for the first 500 iterations and drops twice by a factor of 10, after $2/3$ and $8/9$ of the total training time. We apply a weight decay of 4×10^{-5} and train with a batch size of 64.

Appendix C

Chapter 4 and 5 Appendix

C.1 Dynamic weight sharing

C.1.1 Noiseless case

Each neuron receives the same k-dimensional input $\mathbf{x}$, and its response z_i is given by

$$z_i = \mathbf{w}_i^\top \mathbf{x} = \sum_{j=1}^{k} w_{ij} x_j \,. \tag{C.1}$$

To equalise the weights $\mathbf{w}_i$ among all neurons, the network minimises the following objective,

$$\mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1, \ldots, \mathbf{w}_N) = \frac{1}{4MN} \sum_{m=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(z_i - z_j\right)^2 + \frac{\gamma}{2} \sum_{i=1}^{N} \|\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\|^2 \tag{C.2}$$

$$= \frac{1}{4MN} \sum_{m=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\mathbf{w}_i^\top \mathbf{x}_m - \mathbf{w}_j^\top \mathbf{x}_m\right)^2 + \frac{\gamma}{2} \sum_{i=1}^{N} \|\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\|^2 \,, \tag{C.3}$$

where $\mathbf{w}_i^{\text{init}}$ is the weight at the start of dynamic weight sharing. This is a strongly convex function, and therefore it has a unique minimum.

The SGD update for one $\mathbf{x}_m$ is

$$\Delta \mathbf{w}_i \propto -\left(z_i - \frac{1}{N} \sum_{j=1}^{N} z_j\right) \mathbf{x}_m - \gamma \left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) . \tag{C.4}$$

To find the fixed point of the dynamics, we first set the sum over the gradients to zero,

$$\sum_i \frac{d\mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1,\ldots,\mathbf{w}_N)}{d\mathbf{w}_i} = \frac{1}{M}\sum_{i,m}\left(z_i - \frac{1}{N}\sum_{j=1}^{N} z_j\right)\mathbf{x}_m + \gamma\sum_i\left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) \quad \text{(C.5)}$$

$$= \gamma\sum_i\left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) = 0\,. \quad \text{(C.6)}$$

Therefore, at the fixed point the mean weight $\boldsymbol{\mu}^* = \sum_i \mathbf{w}_i^*/N$ is equal to $\boldsymbol{\mu}^{\text{init}} = \sum_i \mathbf{w}_i^{\text{init}}/N$, and

$$\frac{1}{N}\sum_{i=1}^{N} z_i = \frac{1}{N}\sum_{i=1}^{N} \mathbf{w}_i^{*\top}\mathbf{x}_m = \left(\boldsymbol{\mu}^{\text{init}}\right)^\top \mathbf{x}_m\,. \quad \text{(C.7)}$$

We can now find the individual weights,

$$\frac{d\mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1,\ldots,\mathbf{w}_N)}{d\mathbf{w}_i} = \frac{1}{M}\sum_{m}\left(z_i - \frac{1}{N}\sum_{j=1}^{N} z_j\right)\mathbf{x}_m + \gamma\left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) \quad \text{(C.8)}$$

$$= \frac{1}{M}\sum_m \mathbf{x}_m\mathbf{x}_m^\top\left(\mathbf{w}_i - \boldsymbol{\mu}^{\text{init}}\right) + \gamma\left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) = 0\,. \quad \text{(C.9)}$$

Denoting the covariance matrix $\mathbf{C} \equiv \frac{1}{M}\sum_m \mathbf{x}_m\mathbf{x}_m^\top$, we see that

$$\mathbf{w}_i^* = (\mathbf{C}+\gamma\mathbf{I})^{-1}\left(\mathbf{C}\boldsymbol{\mu}^{\text{init}} + \gamma\mathbf{w}_i^{\text{init}}\right) = (\mathbf{C}+\gamma\mathbf{I})^{-1}\left(\mathbf{C}\frac{1}{N}\sum_{i=1}^{N}\mathbf{w}_i^{\text{init}} + \gamma\mathbf{w}_i^{\text{init}}\right), \quad \text{(C.10)}$$

where $\mathbf{I}$ is the identity matrix. From Eq. (C.10) it is clear that $\mathbf{w}_i^* \approx \boldsymbol{\mu}^{\text{init}}$ for small γ and full rank $\mathbf{C}$. For instance, for $\mathbf{C} = \mathbf{I}$,

$$\mathbf{w}_i^* = \frac{1}{1+\gamma}\boldsymbol{\mu}^{\text{init}} + \frac{\gamma}{1+\gamma}\mathbf{w}_i^{\text{init}}\,. \quad \text{(C.11)}$$

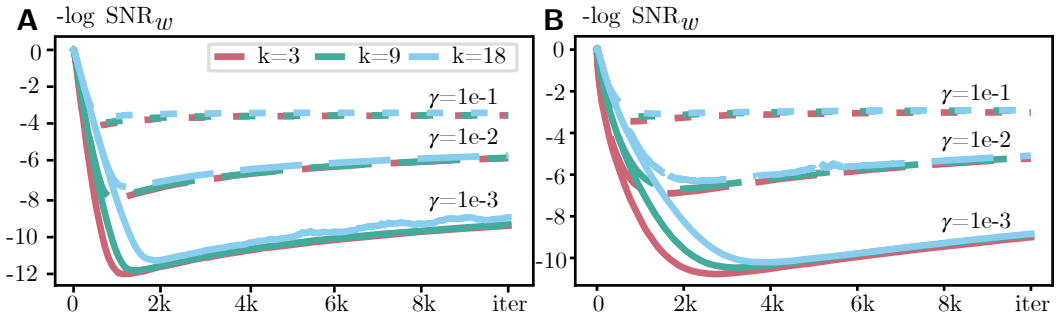

Figure C.1: Logarithm of inverse signal-to-noise ratio (mean weight squared over weight variance, see Eq. (4.6)) for weight sharing objectives in a layer with 100 neurons. **A.** Dynamics of Eq. (C.12) for different kernel sizes k (meaning k^2 inputs) and γ. **B.** Dynamics of weight update that uses Eq. (4.8b) for $\alpha = 10$, different kernel sizes k and γ. In each iteration, the input is presented for 150 ms.

C.1.2 Biased noiseless case, and its correspondence to the realistic implementation

The realistic implementation of dynamic weight sharing with an inhibitory neuron (Section 4.4.2) introduces a bias in the update rule: Eq. (C.4) becomes

$$\Delta \mathbf{w}_i \propto -\left(z_i - \frac{\alpha}{N(1+\alpha)} \sum_{j=1}^{N} z_j\right) \mathbf{x}_m - \gamma\left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) \tag{C.12}$$

for inhibition strength α.

Following the same derivation as for the unbiased case, we can show that the weight dynamics converges to

$$\sum_i \frac{d\,\mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1, \ldots, \mathbf{w}_N)}{d\,\mathbf{w}_i} = \frac{1}{M} \sum_{i,m} \left(z_i - \frac{\alpha}{1+\alpha} \frac{1}{N} \sum_{j=1}^{N} z_j\right) \mathbf{x}_m + \gamma \sum_i \left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) \tag{C.13}$$

$$= \frac{1}{1+\alpha} \mathbf{C} \sum_i \mathbf{w}_i + \gamma \sum_i \left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\right) = 0\,. \tag{C.14}$$

Therefore $\boldsymbol{\mu}^* = \gamma\left(\frac{1}{1+\alpha}\mathbf{C} + \gamma\mathbf{I}\right)^{-1} \boldsymbol{\mu}^{\text{init}}$, and

$$\mathbf{w}_i^* = (\mathbf{C} + \gamma\mathbf{I})^{-1} \left(\frac{\gamma\alpha}{1+\alpha} \mathbf{C} \left(\frac{1}{1+\alpha}\mathbf{C} + \gamma\mathbf{I}\right)^{-1} \boldsymbol{\mu}^{\text{init}} + \gamma \mathbf{w}_i^{\text{init}}\right). \tag{C.15}$$

For $\mathbf{C} = \mathbf{I}$, this becomes

$$\mathbf{w}_i^* = \frac{\gamma}{1+\gamma}\left(\frac{\alpha}{1+\gamma(1+\alpha)}\boldsymbol{\mu}^{\text{init}} + \mathbf{w}_i^{\text{init}}\right). \tag{C.16}$$

As a result, the final weights are approximately the same among neurons, but have a small norm due to the γ scaling.

The dynamics in Eq. (C.12) correctly captures the bias influence in Eq. (4.8b), producing similar SNR plots; compare Fig. C.1A (Eq. (C.12) dynamics) to Fig. C.1B (Eq. (4.8b) dynamics). The curves are slightly different due to different learning rates, but both follow the same trend of first finding a very good solution, and then slowly incorporating the bias term (leading to smaller SNR).

C.1.3 Noisy case

Realistically, all neurons can't see the same $\mathbf{x}_m$. However, due to the properties of our loss, we can work even with noisy updates. To see this, we write the objective function as

$$\mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1, \ldots, \mathbf{w}_N) = \frac{1}{M}\sum_{m=1}^{M} f(\mathbf{W}, \mathbf{X}_m) \tag{C.17}$$

where matrices $\mathbf{W}$ and $\mathbf{X}$ satisfy $(\mathbf{W})_i = \mathbf{w}_i$ and $(\mathbf{X}_m)_i = \mathbf{x}_m$, and

$$f(\mathbf{W}, \mathbf{X}_m) = \frac{1}{4N}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(\mathbf{w}_i^\top \mathbf{x}_m - \mathbf{w}_i^\top \mathbf{x}_m\right)^2 + \frac{\gamma}{2}\sum_{i=1}^{N}\|\mathbf{w}_i - \mathbf{w}_i^{\text{init}}\|^2. \tag{C.18}$$

We'll update the weights with SGD according to

$$\Delta\mathbf{W}^{k+1} = -\eta_k \frac{d}{d\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)} + \mathbf{E}^k)\Big|_{\mathbf{W}^k}, \tag{C.19}$$

where $(\mathbf{E}^k)_i = \boldsymbol{\varepsilon}_i$ is zero-mean input noise and $m(k)$ is chosen uniformly.

Let's also bound the input mean and noise as

$$\mathbb{E}_{\mathbf{E}}\left\|\mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i\right\|^2 \leq \sqrt{c_{x\varepsilon}}, \quad \mathbb{E}_{\mathbf{E}}\left\|\mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i\right\|^4 \leq c_{x\varepsilon}. \tag{C.20}$$

With this setup, we can show that SGD with noise can quickly converge to the correct solution, apart from a constant noise-induced bias. Our analysis is standard and follows [203], but had to be adapted for our objective and noise model.

Theorem 6. *For zero-mean isotropic noise* $\mathbf{E}$ *with variance* σ^2*, uniform SGD sampling* $m(k)$ *and inputs* $\boldsymbol{x}_m$ *that satisfy Eq.* (C.20)*, choosing* $\eta_k = O(1/k)$ *leads to*

$$\mathbb{E}\,\|\mathbf{W}^{k+1} - \mathbf{W}^*\|_F^2 = O\left(\frac{\left\|\mathbf{W}^{\text{init}} - \mathbf{W}^*\right\|_F}{k+1}\right) + O\left(\sigma^2 \|\mathbf{W}^*\|_F^2\right), \tag{C.21}$$

where $(\mathbf{W}^*)_i$ *is given by Eq.* (C.10)*.*

Proof. Using the SGD update,

$$\|\mathbf{W}^{k+1} - \mathbf{W}^*\|_F^2 = \left\|\mathbf{W}^k - \eta_k \frac{d}{d\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)} + \mathbf{E}^k)\Big|_{\mathbf{W}^k} - \mathbf{W}^*\right\|_F^2 \tag{C.22}$$

$$= \left\|\mathbf{W}^k - \mathbf{W}^*\right\|_F^2 - 2\eta_k \left\langle \mathbf{W}^k - \mathbf{W}^*,\, \frac{d}{d\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)} + \mathbf{E}^k)\Big|_{\mathbf{W}^k} \right\rangle \tag{C.23}$$

$$+ \eta_k^2 \left\| \frac{d}{d\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)} + \mathbf{E}^k)\Big|_{\mathbf{W}^k} \right\|_F^2. \tag{C.24}$$

We need to bound the second and the third terms in the equation above.

Second term. As f is γ-strongly convex in $\mathbf{W}$,

$$-\left\langle \mathbf{W}^k - \mathbf{W}^*,\, \frac{d}{d\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)} + \mathbf{E}^k)\Big|_{\mathbf{W}^k} \right\rangle \tag{C.25}$$

$$\leq f(\mathbf{W}^*, \mathbf{X}_{m(k)} + \mathbf{E}^k) - f(\mathbf{W}^k, \mathbf{X}_{m(k)} + \mathbf{E}^k) - \frac{\gamma}{2}\|\mathbf{W}^k - \mathbf{W}^*\|_F^2. \tag{C.26}$$

As f is convex in $\mathbf{X}$,

$$f(\mathbf{W}^*, \mathbf{X}_{m(k)} + \mathbf{E}^k) - f(\mathbf{W}^k, \mathbf{X}_{m(k)} + \mathbf{E}^k) \leq f(\mathbf{W}^*, \mathbf{X}_{m(k)}) - f(\mathbf{W}^k, \mathbf{X}_{m(k)}) \tag{C.27}$$

$$+ \left\langle \frac{d}{d\mathbf{X}} f(\mathbf{W}^*, \mathbf{X})\Big|_{\mathbf{X}_{m(k)} + \mathbf{E}^k} - \frac{d}{d\mathbf{X}} f(\mathbf{W}^k, \mathbf{X})\Big|_{\mathbf{X}_{m(k)}}, \mathbf{E}^k \right\rangle. \tag{C.28}$$

We only need to clarify one term here,

$$\left(\frac{d}{d\mathbf{X}}f(\mathbf{W}^*,\mathbf{X})\Big|_{\mathbf{X}_{m(k)}+\mathbf{E}^k}\right)_i = \left(\frac{d}{d\mathbf{X}}f(\mathbf{W}^*,\mathbf{X})\Big|_{\mathbf{X}_{m(k)}}\right)_i + \left(\mathbf{w}_i^{*\top}\boldsymbol{\varepsilon}_i - \frac{1}{N}\sum_j \mathbf{w}_j^{*\top}\varepsilon_j\right)\mathbf{w}_i^*\,. \tag{C.29}$$

Now we can take the expectation over $m(k)$ and $\mathbf{E}$. As $m(k)$ is uniform, and $\mathbf{W}^*$ minimises the global function,

$$\mathbb{E}_{m(k)}\left(f(\mathbf{W}^*,\mathbf{X}_{m(k)}) - f(\mathbf{W}^k,\mathbf{X}_{m(k)})\right) = \mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1^*,\ldots,\mathbf{w}_N^*) - \mathscr{L}_{\text{w. sh.}}(\mathbf{w}_1^k,\ldots,\mathbf{w}_N^k) \leq 0\,. \tag{C.30}$$

As $\mathbf{E}^k$ is zero-mean and isotropic with variance σ^2,

$$\mathbb{E}_{m(k),\mathbf{E}^k}\left\langle \frac{d}{d\mathbf{X}}f(\mathbf{W}^*,\mathbf{X})\Big|_{\mathbf{X}_{m(k)}+\mathbf{E}^k} - \frac{d}{d\mathbf{X}}f(\mathbf{W}^k,\mathbf{X})\Big|_{\mathbf{X}_{m(k)}}, \mathbf{E}^k\right\rangle \tag{C.31}$$

$$= \mathbb{E}_{\mathbf{E}^k}\sum_i \left(\mathbf{w}_i^{*\top}\boldsymbol{\varepsilon}_i - \frac{1}{N}\sum_j \mathbf{w}_j^{*\top}\varepsilon_j\right)\mathbf{w}_i^{*\top}\boldsymbol{\varepsilon}_i = \left(1-\frac{1}{N}\right)\mathbb{E}_{\mathbf{E}^k}\sum_i\left(\mathbf{w}_i^{*\top}\boldsymbol{\varepsilon}_i\right)^2 \tag{C.32}$$

$$= \left(1-\frac{1}{N}\right)\mathbb{E}_{\mathbf{E}^k}\sum_i \operatorname{Tr}\left(\mathbf{w}_i^*\mathbf{w}_i^{*\top}\boldsymbol{\varepsilon}_i\boldsymbol{\varepsilon}_i^\top\right) \leq \sigma^2\|\mathbf{W}^*\|_F^2\,. \tag{C.33}$$

So the whole second term becomes

$$-2\eta_k\mathbb{E}_{m(k),\mathbf{E}}\left\langle \mathbf{W}^k - \mathbf{W}^*,\ \frac{d}{d\mathbf{W}}f(\mathbf{W},\mathbf{X}_{m(k)}+\mathbf{E}^k)\Big|_{\mathbf{W}^k}\right\rangle \tag{C.34}$$

$$\leq -\gamma\eta_k\mathbb{E}_{m(k),\mathbf{E}^k}\|\mathbf{W}^k-\mathbf{W}^*\|_F^2 + \eta_k\sigma^2\|\mathbf{W}^*\|_F^2\,. \tag{C.35}$$

Third term. First, observe that

$$\frac{d}{d\,\mathbf{w}_i}f(\mathbf{W},\mathbf{X}) = \mathbf{x}_i\mathbf{x}_i^\top\mathbf{w}_i - \mathbf{x}_i\frac{1}{N}\sum_j \mathbf{x}_j^\top\mathbf{w}_j + \gamma\mathbf{w}_i - \gamma\mathbf{w}_i^{\text{init}} \tag{C.36}$$

$$= \left(1-\frac{1}{N}\right)\mathbf{A}_i\mathbf{w}_i - \mathbf{B}_i\mathbf{W} + \gamma\mathbf{w}_i - \gamma\mathbf{w}_i^{\text{init}}\,, \tag{C.37}$$

where $\mathbf{A}_i = \mathbf{x}_i\mathbf{x}_i^\top$ and $(\mathbf{B}_i)_j = \mathbb{I}[i\neq j]\mathbf{x}_i\mathbf{x}_j^\top/N$.

Therefore, using $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ twice, properties of the matrix

2-norm, and $(1-1/N) \leq 1$,

$$\left\| \frac{d}{d\,\mathbf{w}_i} f(\mathbf{W}, \mathbf{X}) \right\|^2 \leq 4 \left\|\mathbf{A}_i\right\|_2^2 \left\|\mathbf{w}_i\right\|^2 + 4 \left\|\mathbf{B}_i\right\|_2^2 \left\|\mathbf{W}\right\|^2 + 4\gamma^2 \left\|\mathbf{w}_i\right\|^2 + 4\gamma^2 \left\|\mathbf{w}_i^{\text{init}}\right\|^2 . \tag{C.38}$$

In our particular case, bounding the 2 norm with the Frobenius norm gives

$$\begin{aligned} \mathbb{E}_{m(k),\mathbf{E}} \left\|\mathbf{A}_i\right\|_2^2 &\leq \mathbb{E}_{m(k),\mathbf{E}} \left\| (\mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i)(\mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i)^\top \right\|_F^2 && \text{(C.39)} \\ &= \mathbb{E}_{m(k),\mathbf{E}} \left\| \mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i \right\|^4 \leq c_{x\varepsilon} . && \text{(C.40)} \end{aligned}$$

Similarly,

$$\mathbb{E}_{m(k),\mathbf{E}} \left\|\mathbf{B}_i\right\|_2^2 \leq \mathbb{E}_{m(k),\mathbf{E}} \left\|\mathbf{B}_i\right\|_F^2 \leq \frac{1}{N^2} \mathbb{E}_{m(k),\mathbf{E}} \sum_{j \neq i} \left\| \mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_i \right\|^2 \left\| \mathbf{x}_{m(k)} + \boldsymbol{\varepsilon}_j \right\|^2 \leq \frac{c_{x\varepsilon}}{N} . \tag{C.41}$$

Therefore, we can bound the full gradient by the sum of individual bounds (as it's the Frobenius norm) and using $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ again,

$$\mathbb{E}_{m(k),\mathbf{E}} \left\| \frac{d}{d\,\mathbf{W}} f(\mathbf{W}, \mathbf{X}_{m(k)+\mathbf{E}^k}) \Big|_{\mathbf{W}^k} \right\|_F^2 \leq 4(2c_{x\varepsilon} + \gamma^2) \left\|\mathbf{W}^k\right\|_F^2 + 4\gamma^2 \left\|\mathbf{W}^{\text{init}}\right\|_F^2 \tag{C.42}$$

$$\leq 8(2c_{x\varepsilon} + \gamma^2) \left\|\mathbf{W}^k - \mathbf{W}^*\right\|_F^2 + 8(2c_{x\varepsilon} + \gamma^2) \left\|\mathbf{W}^*\right\|_F^2 + 4\gamma^2 \left\|\mathbf{W}^{\text{init}}\right\|_F^2 . \tag{C.43}$$

Combining all of this, and taking the expectation over all steps before $k+1$, gives us

$$\begin{aligned} \mathbb{E} \left\|\mathbf{W}^{k+1} - \mathbf{W}^*\right\|_F^2 &\leq \left(1 - \gamma\eta_k + \eta_k^2 8(2c_{x\varepsilon} + \gamma^2)\right) \mathbb{E} \left\|\mathbf{W}^k - \mathbf{W}^*\right\|_F^2 && \text{(C.44)} \\ &\quad + \eta_k \sigma^2 \|\mathbf{W}^*\|_F^2 + \eta_k^2 \left(8(2c_{x\varepsilon} + \gamma^2) \left\|\mathbf{W}^*\right\|_F^2 + 4\gamma^2 \left\|\mathbf{W}^{\text{init}}\right\|_F^2 \right) . && \text{(C.45)} \end{aligned}$$

If we choose η_k such that $\eta_k \cdot \left(8(2c_{x\varepsilon}+\gamma^2)\|\mathbf{W}^*\|_F^2 + 4\gamma^2\left\|\mathbf{W}^{\text{init}}\right\|_F^2\right) \leq \sigma^2$, we can simplify the result,

$$\mathbb{E}\|\mathbf{W}^{k+1}-\mathbf{W}^*\|_F^2 \leq \left(1-\gamma\eta_k+\eta_k^2 8(2c_{x\varepsilon}+\gamma^2)\right)\mathbb{E}\left\|\mathbf{W}^k-\mathbf{W}^*\right\|_F^2 + 2\eta_k\sigma^2\|\mathbf{W}^*\|_F^2 \tag{C.46}$$

$$\leq \left(\prod_{s=0}^{k}\left(1-\gamma\eta_s+\eta_s^2 8(2c_{x\varepsilon}+\gamma^2)\right)\right)\mathbb{E}\left\|\mathbf{W}^{\text{init}}-\mathbf{W}^*\right\|_F^2 \tag{C.47}$$

$$+ 2\sigma^2\sum_{t=0}^{k}\eta_t\left(\prod_{s=1}^{t}\left(1-\gamma\eta_s+\eta_s^2 8(2c_{x\varepsilon}+\gamma^2)\right)\right)\|\mathbf{W}^*\|_F^2 . \tag{C.48}$$

If we choose $\eta_k = O(1/k)$, the first term will decrease as $1/k$. The second one will stay constant with time, and proportional to σ^2.

□

C.1.4 Applicability to vision transformers

In vision transformers (e.g. [167]), an input image is reshaped into a matrix $\mathbf{Z} \in \mathbb{R}^{N\times D}$ for N non-overlapping patches of the input, each of size D. As the first step, $\mathbf{Z}$ is multiplied by a matrix $\mathbf{U} \in \mathbb{R}^{D\times 3D}$ as $\mathbf{Z}' = \mathbf{Z}\mathbf{U}$. Therefore, an output neuron $z'_{ij} = \sum_k z_{ik}u_{kj}$ looks at $\mathbf{z}_i$ with the same weights as $z'_{i'j} = \sum_k z_{i'k}u_{kj}$ uses for $\mathbf{z}_{i'}$ for any i'.

To share weights with dynamic weight sharing, for each k we need to connect all z_{ik} across i (input layer), and for each j – all z'_{ij} across i (output layer). After that, weight sharing will proceed just like for locally connected networks: activate an input grid j_1 (one of D possible ones) to create a repeating input pattern, then activate a grid j_2 and so on.

C.1.5 Details for convergence plots

Both plots in Fig. 4.5 show mean negative log SNR over 10 runs, 100 output neurons each. Initial weights were drawn from $\mathcal{N}(1,1)$. At every iteration, the new input $\mathbf{x}$ was drawn from $\mathcal{N}(1,1)$ independently for each compo-

nent. Learning was performed via SGD with momentum of 0.95. The minimum SNR value was computed from Eq. (4.5). For our data, the SNR expression in Eq. (4.6) has $\left(\frac{1}{N}\sum_i (\mathbf{w}_i)_j\right)^2 \approx 1$ and $\frac{1}{N}\sum_i \left((\mathbf{w}_i)_j - \frac{1}{N}\sum_{i'}(\mathbf{w}_{i'})_j\right)^2 \approx \gamma^2/(1+\gamma)^2$, therefore $-\log \mathrm{SNR}_{\min} = 2\log(\gamma/(1+\gamma))$.

For Fig. 4.5A, we performed 2000 iterations (with a new $\mathbf{x}$ each time). Learning rate at iteration k was $\eta_k = 0.5/(1000+k)$. For Fig. C.1A, we did the same simulation but for 10^4 iterations.

For Fig. 4.5B, network dynamics (Eq. (4.8b)) was simulated with $\tau = 30$ ms, $b = 1$ using Euler method with steps size of 1 ms. We performed 10^4 iterations (150 ms per iteration, with a new $\mathbf{x}$ each iteration). Learning rate at iteration k was $\eta_k = 0.0003/\sqrt{1+k/2} \cdot \mathbb{I}[k \geq 50]$.

The code for both runs is provided in the supplementary material.

C.2 Experimental details

Both convolutional and LC layers did not have the bias term, and were initialised according to Kaiming Normal initialisation [204] with ReLU gain, meaning each weight was drawn from $\mathcal{N}(0, 2/(c_{\text{out}}k^2))$ for kernel size k and c_{out} output channels.

All runs were done with automatic mixed precision, meaning that inputs to each layer (but not the weights) were stored as float16, and not float32. This greatly improved performance and memory requirements of the networks.

As an aside, the weight dynamics of sleep/training phases indeed followed Fig. 4.3A. Fig. C.2 shows $-\log \mathrm{SNR}_w$ (defined in Eq. (4.6)) for weight sharing every 10 iterations on CIFAR10. For small learning rates, the weights do not diverge too much in-between sleep phases.

C.2.1 CIFAR10/100, TinyImageNet

Mean performance over 5 runs is summarised in Table C.1 (padding of 0), Table C.2 (padding of 4), and Table C.3 (padding of 8). Maximum minus minimum accuracy is summarised in Table C.4, Table C.5, and Table C.6. Hyperparameters for AdamW (learning rate and weight decay) are provided in Table C.7, Table C.8, and Table C.9.

Hyperparameters were optimised on a train/validation split (see Section 4.5)

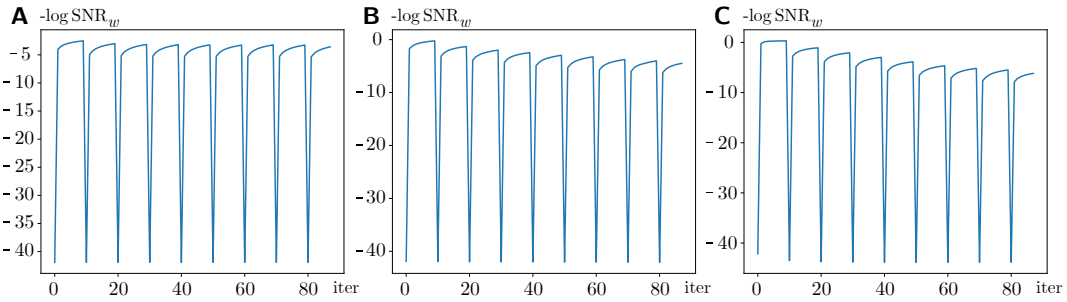

Figure C.2: Logarithm of inverse signal-to-noise ratio (mean weight squared over weight variance, see Eq. (4.6)) for weight sharing every 10 iterations for CIFAR10. **A.** Learning rate = 5e-4. **B.** Learning rate = 5e-3. **C.** Learning rate = 5e-2.

over the following grids. **CIFAR10/100.** Learning rate: [1e-1, 5e-2, 1e-2, 5e-3] (conv), [1e-3, 5e-4, 1e-4, 5e-5] (LC); weight decay [1e-2, 1e-4] (both). **TinyImageNet.** Learning rate: [5e-3, 1e-3, 5e-4] (conv), [1e-3, 5e-4] (LC); weight decay [1e-2, 1e-4] (both). The learning rate range for TinyImageNet was smaller as preliminary experiments showed poor performance for slow learning rates.

For all runs, the batch size was 512. For all final runs, learning rate was divided by 4 at 100 and then at 150 epochs (out of 200). Grid search for CIFAR10/100 was done for the same 200 epochs setup. For TinyImageNet, grid search was performed over 50 epochs with learning rate decreases at 25 and 37 epochs (i.e., the same schedule but compressed) due to the larger computational cost of full runs.

C.2.2 ImageNet

In addition to the main results, we also tested the variant of the locally connected network with a convolutional first layer (Table C.10). It improved performance for all configurations: from about 2% for weight sharing every 1-10 iterations, to about 5% for 100 iterations and for no weight sharing. This is not surprising, as the first layer has the largest resolution (224 by 224; initially, we performed these experiments due to memory constraints). Our result suggests that adding a "good" pre-processing layer (e.g. the retina) can also improve performance of locally connected networks.

Final hyperparameters. Learning rate: 1e-3 (conv, LC with w.sh. (1)), 5e-4 (all other LC; all LC with 1st layer conv), weight decay: 1e-2 (all). Hyperparameters were optimised on a train/validation split (see Section 4.5) over the following grids.

Table C.1: Performance of convolutional (conv) and locally connected (LC) networks for padding of 0 in the input images (mean accuracy over 5 runs). For LC, two regularisation strategies were applied: repeating the same image n times with different translations (n reps) or using dynamic weight sharing every n batches (ws (n)). LC nets additionally show performance difference w.r.t. conv nets.

Regulariser	Connectivity	CIFAR10		CIFAR100				TinyImageNet			
		Top-1 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
-	conv	84.1	-	49.5	-	78.2	-	26.0	-	51.2	-
	LC	67.2	-16.8	34.9	-14.6	62.2	-16.0	12.0	-14.1	30.4	-20.7
Weight Sharing	LC - ws(1)	74.8	-9.3	41.8	-7.7	70.1	-8.1	24.9	-1.2	49.1	-2.1
	LC - ws(10)	75.9	-8.1	44.4	-5.1	72.0	-6.2	28.1	2.0	52.5	1.3
	LC - ws(100)	75.4	-8.6	43.4	-6.1	71.9	-6.3	27.4	1.3	51.9	0.8

Table C.2: Mean performance over 5 runs. Same as Table C.1, but for padding of 4.

Regulariser	Connectivity	CIFAR10		CIFAR100				TinyImageNet			
		Top-1 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
-	conv	88.3	-	59.2	-	84.9	-	38.6	-	65.1	-
	LC	80.9	-7.4	49.8	-9.4	75.5	-9.4	29.6	-9.0	52.7	-12.4
Data Translation	LC - 4 reps	82.9	-5.4	52.1	-7.1	76.4	-8.5	31.9	-6.7	54.9	-10.2
	LC - 8 reps	83.8	-4.5	54.3	-5.0	77.9	-7.0	33.0	-5.6	55.6	-9.5
	LC - 16 reps	85.0	-3.3	55.9	-3.3	78.8	-6.1	34.0	-4.6	56.2	-8.8
Weight Sharing	LC - ws(1)	87.4	-0.8	58.7	-0.5	83.4	-1.6	41.6	3.0	66.1	1.1
	LC - ws(10)	85.1	-3.2	55.7	-3.6	80.9	-4.0	37.4	-1.2	61.8	-3.2
	LC - ws(100)	82.0	-6.3	52.8	-6.4	80.1	-4.8	37.1	-1.5	62.8	-2.3

Conv: learning rate [1e-3, 5e-4], weight decay [1e-2, 1e-4, 1e-6]. LC: learning rate [1e-3, 5e-4, 1e-4, 5e-5], weight decay [1e-2]. LC (1st layer conv): learning rate [1e-3, 5e-4], weight decay [1e-2, 1e-4, 1e-6]. For LC, we only tried the large weight decay based on an earlier experiment (LC (1st layer conv)). For LC (1st layer conv), we only tuned hyperparameters for LC and LC with weight sharing in each iteration, as they found the same values (weight sharing every 10/100 iterations interpolates between LC and LC with weight sharing in each iteration, and therefore is expected to behave similarly to both). In addition, for LC (1st layer conv) we only tested learning rate of 5e-4 for weight decay of 1e-2 as higher learning rates performed significantly worse for other runs (and in preliminary experiments).

For all runs, the batch size was 256. For all final runs, learning rate was divided by 4 at 100 and then at 150 epochs (out of 200). Grid search was performed over 20 epochs with learning rate decreases at 10 and 15 epochs (i.e., the same schedule but compressed) due to the large computational cost of full runs.

Table C.3: Mean performance over 5 runs. Same as Table C.1, but for padding of 8.

Regulariser	Connectivity	CIFAR10		CIFAR100				TinyImageNet			
		Top-1 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
-	conv	88.7	-	59.6	-	85.4	-	42.6	-	68.7	-
	LC	80.7	-8.0	47.7	-11.8	74.8	-10.6	31.9	-10.7	55.4	-13.3
Data Translation	LC - 4 reps	82.8	-6.0	50.6	-9.0	76.2	-9.2	35.5	-7.1	58.6	-10.1
	LC - 8 reps	83.6	-5.1	53.0	-6.6	77.4	-8.0	35.8	-6.7	59.0	-9.7
	LC - 16 reps	85.0	-3.8	55.6	-4.0	78.4	-7.0	37.9	-4.7	60.3	-8.4
Weight Sharing	LC - ws(1)	87.8	-0.9	59.2	-0.4	84.0	-1.4	43.6	1.0	67.9	-0.9
	LC - ws(10)	84.3	-4.5	53.7	-5.8	80.4	-5.0	39.6	-2.9	64.5	-4.3
	LC - ws(100)	79.5	-9.3	50.0	-9.6	78.6	-6.8	39.2	-3.4	64.8	-3.9

Table C.4: Max minus min performance over 5 runs; padding of 0.

Regulariser	Connectivity	CIFAR10	CIFAR100		TinyImageNet	
		Top-1 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)
-	conv	0.5	1.0	1.7	1.0	0.4
	LC	0.4	1.6	1.5	1.0	1.7
Weight Sharing	LC - ws(1)	0.5	1.3	1.3	1.2	2.0
	LC - ws(10)	0.8	1.0	0.7	1.8	2.1
	LC - ws(100)	0.9	0.7	0.9	1.0	1.3

Table C.5: Max minus min performance over 5 runs; padding of 4.

Regulariser	Connectivity	CIFAR10	CIFAR100		TinyImageNet	
		Top-1 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)
-	conv	0.7	1.5	0.2	1.2	1.1
	LC	0.8	1.1	0.4	0.7	0.8
Data Translation	LC - 4 reps	0.8	1.3	0.8	0.5	0.8
	LC - 8 reps	0.3	1.4	1.3	0.7	1.2
	LC - 16 reps	0.7	0.7	0.6	0.9	0.5
Weight Sharing	LC - ws(1)	0.5	1.1	0.9	0.9	0.6
	LC - ws(10)	0.6	1.1	0.3	0.6	1.2
	LC - ws(100)	0.7	1.0	0.6	0.2	0.9

Table C.6: Max minus min performance over 5 runs; padding of 8.

Regulariser	Connectivity	CIFAR10	CIFAR100		TinyImageNet	
		Top-1 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)	Top-1 accuracy (%)	Top-5 accuracy (%)
-	conv	0.9	1.5	1.2	1.7	1.0
	LC	0.5	0.6	0.5	0.5	0.9
Data Translation	LC - 4 reps	0.4	0.9	0.3	0.6	0.8
	LC - 8 reps	0.6	0.9	0.5	0.5	0.6
	LC - 16 reps	0.9	0.9	0.6	0.5	1.1
Weight Sharing	LC - ws(1)	0.4	1.2	1.5	0.7	0.7
	LC - ws(10)	0.2	1.4	0.9	1.4	1.2
	LC - ws(100)	0.4	0.5	0.7	0.7	0.9

Table C.7: Hyperparameters for padding of 0.

Regulariser	Connectivity	CIFAR10		CIFAR100		TinyImageNet	
		Learning rate	Weight decay	Learning rate	Weight decay	Learning rate	Weight decay
-	conv	0.01	0.01	0.01	0.01	0.005	0.01
	LC	0.001	0.01	0.001	0.01	0.001	0.0001
Weight Sharing	LC - ws(1)	0.001	0.01	0.001	0.01	0.001	0.0001
	LC - ws(10)	0.0005	0.01	0.0005	0.0001	0.0005	0.01
	LC - ws(100)	0.0001	0.01	0.0001	0.01	0.001	0.0001

Table C.8: Hyperparameters for padding of 4.

Regulariser	Connectivity	CIFAR10		CIFAR100		TinyImageNet	
		Learning rate	Weight decay	Learning rate	Weight decay	Learning rate	Weight decay
-	conv	0.01	0.0001	0.01	0.01	0.005	0.0001
	LC	0.001	0.0001	0.0005	0.01	0.0005	0.0001
Data Translation	LC - 4 reps	0.001	0.01	0.001	0.01	0.0005	0.01
	LC - 8 reps	0.0005	0.01	0.0005	0.0001	0.0005	0.01
	LC - 16 reps	0.0005	0.01	0.0005	0.01	0.0005	0.01
Weight Sharing	LC - ws(1)	0.001	0.01	0.001	0.0001	0.001	0.01
	LC - ws(10)	0.0005	0.01	0.0005	0.01	0.001	0.0001
	LC - ws(100)	0.0005	0.01	0.0005	0.01	0.001	0.01

Table C.9: Hyperparameters for padding of 8.

Regulariser	Connectivity	CIFAR10		CIFAR100		TinyImageNet	
		Learning rate	Weight decay	Learning rate	Weight decay	Learning rate	Weight decay
-	conv	0.01	0.01	0.01	0.01	0.005	0.01
	LC	0.001	0.01	0.0005	0.0001	0.001	0.01
Data Translation	LC - 4 reps	0.0005	0.01	0.001	0.0001	0.0005	0.01
	LC - 8 reps	0.001	0.01	0.0005	0.0001	0.0005	0.0001
	LC - 16 reps	0.0005	0.0001	0.0005	0.01	0.0005	0.01
Weight Sharing	LC - ws(1)	0.001	0.0001	0.001	0.01	0.001	0.01
	LC - ws(10)	0.0005	0.01	0.0005	0.0001	0.001	0.0001
	LC - ws(100)	0.0005	0.01	0.0005	0.0001	0.001	0.0001

Table C.10: Performance of convolutional (conv), locally connected (LC) and locally connected with convolutional first layer (LC + 1st layer conv) networks on ImageNet (1 run). For LC, we also used dynamic weight sharing every *n* batches. LC nets additionally show performance difference w.r.t. the conv net.

Model	Connectivity	Weight sharing frequency	ImageNet			
			Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
0.5x ResNet18	conv	-	63.5	-	84.7	-
	LC	-	46.7	-16.8	70.0	-14.7
	LC	1	61.7	-1.8	83.1	-1.6
	LC	10	59.3	-4.2	81.1	-3.6
	LC	100	54.5	-9.0	77.7	-7.0
0.5x ResNet18 (1st layer conv)	LC	-	52.2	-11.3	75.1	-9.6
	LC	1	63.6	0.1	84.5	-0.2
	LC	10	61.6	-1.9	83.1	-1.6
	LC	100	59.1	-4.4	81.1	-3.6

C.3 Brain-Score details

Hyperparameters: weight decay for all networks was set to 1e-2. Learning rates were chosen on a validation set, and are provided in Table C.11.

Table C.11: ImageNet top-1 accuracy and Brain-Score for several ResNet-18 networks: convolutional, locally connected (LC), LC with a convolutional first layer, LC with a weight sharing sleep phase every 1/10/100 iterations, and with random and convolutional (conv) initialisation.

Connectivity	Weight sharing frequency	Initialisation	Top-1 acc., %	Brain-Score		Learning rate
				average score	link	
conv	-	conv	69.9	0.426	brain-score.org/model/1071	1e-3
LC	-	random	44.9	0.383	brain-score.org/model/1072	5e-4
LC	1	random	62.0	0.409	brain-score.org/model/1073	1e-4
LC	10	random	60.9	0.418	brain-score.org/model/1074	5e-4
LC	100	random	56.5	0.424	brain-score.org/model/1075	5e-4
LC	-	conv	46.1	0.374	brain-score.org/model/1076	1e-4
LC	1	conv	64.4	0.433	brain-score.org/model/1077	1e-3
LC	10	conv	63.0	0.429	brain-score.org/model/1078	1e-3
LC	100	conv	58.7	0.416	brain-score.org/model/1079	1e-3
LC + 1st conv	-	random	52.5	0.370	brain-score.org/model/1157	1e-4
LC + 1st conv	-	conv	52.5	0.379	brain-score.org/model/1094	1e-4
conv (untrained)	-	conv	-	0.227	brain-score.org/model/1159	-
LC (untrained)	-	random	-	0.137	brain-score.org/model/1158	-

Bibliography

[1] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

[2] Pablo E Castillo, Chiayu Q Chiu, and Reed C Carroll. Long-term plasticity at inhibitory synapses. *Current opinion in neurobiology*, 21(2):328–338, 2011.

[3] Romain Brette. Philosophy of the spike: rate-based vs. spike-based theories of the brain. *Frontiers in systems neuroscience*, page 151, 2015.

[4] Donald Olding Hebb. The organization of behavior: A neuropsychological theory. 1949.

[5] RE Brown. Do hebb and the origins of the organization of behavior. In *Society of Neuroscience Abstracts*, pages 22–23, 2001.

[6] RE Brown. Do hebb's "lost" ma thesis: The first draft of the hebb synapse. *Society of Neuroscience, Poster*, 21, 2002.

[7] Peter Milner. A brief history of the hebbian learning rule. *Canadian Psychology/Psychologie canadienne*, 44(1):5, 2003.

[8] Donald Olding Hebb. *Conditioned and unconditioned reflexes and inhibition*. PhD thesis, McGill University Montreal, Quebec, Canada, 1932.

[9] Elie L Bienenstock, Leon N Cooper, and Paul W Munro. Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, 2(1):32–48, 1982.

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

[11] Carlos SN Brito and Wulfram Gerstner. Nonlinear hebbian learning as a unifying principle in receptive field formation. *PLoS computational biology*, 12(9):e1005070, 2016.

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[13] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[14] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[15] John Carew Eccles. From electrical to chemical transmission in the central nervous system: the closing address of the sir henry dale centennial symposium cambridge, 19 september 1975. *Notes and records of the Royal Society of London*, 30(2):219–230, 1976.

[16] Piergiorgio Strata, Robin Harvey, et al. Dale's principle. *Brain research bulletin*, 50(5):349–350, 1999.

[17] Victor AF Lamme and Pieter R Roelfsema. The distinct modes of vision offered by feedforward and recurrent processing. *Trends in neurosciences*, 23(11):571–579, 2000.

[18] Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

[19] Jonathan Cornford, Damjan Kalajdzievski, Marco Leite, Amélie Lamarquette, Dimitri M Kullmann, and Blake Richards. Learning to live with dale's

principle: Anns with separate excitatory and inhibitory units. *bioRxiv*, pages 2020–11, 2021.

[20] Jonas Kubilius, Martin Schrimpf, Ha Hong, Najib J. Majaj, Rishi Rajalingham, Elias B. Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, Aran Nayebi, Daniel Bear, Daniel L. K. Yamins, and James J. DiCarlo. Brain-Like Object Recognition with High-Performing Shallow Recurrent ANNs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. D'Alché-Buc, E. Fox, and R. Garnett, editors, *Neural Information Processing Systems (NeurIPS)*, pages 12785—-12796. Curran Associates, Inc., 2019.

[21] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[22] Martin Schrimpf, Jonas Kubilius, Ha Hong, Najib J. Majaj, Rishi Rajalingham, Elias B. Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Franziska Geiger, Kailyn Schmidt, Daniel L. K. Yamins, and James J. DiCarlo. Brain-score: Which artificial neural network for object recognition is most brain-like? *bioRxiv preprint*, 2018.

[23] Martin Schrimpf, Jonas Kubilius, Ha Hong, Najib J Majaj, Rishi Rajalingham, Elias B Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Franziska Geiger, et al. Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv*, page 407007, 2020.

[24] Jianghong Shi, Bryan Tripp, Eric Shea-Brown, Stefan Mihalas, and Michael Buice. Cnn mousenet: A biologically constrained convolutional neural network model for mouse visual cortex. *bioRxiv*, 2021.

[25] Shahab Bakhtiari, Patrick J Mineault, Tim Lillicrap, Christopher C Pack, and Blake Aaron Richards. The functional specialization of visual cortex emerges

from training parallel pathways with self-supervised predictive learning. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[26] Roman Pogodin and Peter E Latham. Kernelized information bottleneck leads to biologically plausible 3-factor hebbian learning in deep networks. *arXiv preprint arXiv:2006.07123*, 2020.

[27] Yazhe Li*, Roman Pogodin*, Danica J Sutherland, and Arthur Gretton. Self-supervised learning with kernel dependence maximization. *Advances in Neural Information Processing Systems*, 34, 2021. *Equal contribution.

[28] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[30] Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. MINE: mutual information neural estimation. In *ICML*. 2018.

[31] Ben Poole, Sherjil Ozair, Aäron van den Oord, Alexander A. Alemi, and George Tucker. On variational bounds of mutual information. In *ICML*, 2019.

[32] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International conference on algorithmic learning theory*, pages 63–77. Springer, 2005.

[33] Zoltán Szabó and Bharath K. Sriperumbudur. Characteristic and universal tensor product kernels. *Journal of Machine Learning Research*, 18(233):1–29, 2018.

[34] Arthur Gretton, Kenji Fukumizu, Choon Hui Teo, Le Song, Bernhard Schölkopf, and Alexander J Smola. A kernel statistical test of independence. In *NeurIPS*, 2007.

[35] Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization. *Journal of Machine Learning Research*, 13(5), 2012.

[36] Le Song, Alex Smola, Arthur Gretton, and Karsten M. Borgwardt. A dependence maximization view of clustering. In *ICML*, 2007.

[37] Matthew B. Blaschko and Arthur Gretton. Learning taxonomies by dependence maximization. In *NeurIPS*, 2009.

[38] Siddhartha Jain, Ge Liu, and David Gifford. Information condensing active learning, 2020.

[39] Kurt Wan-Duo Ma, J. P. Lewis, and W. Bastiaan Kleijn. The HSIC bottleneck: Deep learning without back-propagation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5085–5092. AAAI Press, 2020.

[40] Denny Wu, Yixiu Zhao, Yao-Hung Hubert Tsai, Makoto Yamada, and Ruslan Salakhutdinov. 'dependency bottleneck' in auto-encoding architectures: an empirical study, 2018.

[41] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

[42] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[43] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[44] W. Rudin. *Functional Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1991.

[45] 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.

[46] Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, pages 974–982, 2019.

[47] 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):1–10, 2016.

[48] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In *Advances in Neural Information Processing Systems*, pages 8721–8732, 2018.

[49] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to ImageNet. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 583–593. PMLR, 09–15 Jun 2019.

[50] Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3033–3045, 2019.

[51] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[52] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[53] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

[54] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51, 2012.

[55] Shawn R Olsen, Vikas Bhandawat, and Rachel I Wilson. Divisive normalization in olfactory population codes. *Neuron*, 66(2):287–299, 2010.

[56] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[57] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[58] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[59] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[60] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[61] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[62] Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? *arXiv preprint arXiv:1510.05067*, 2015.

[63] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso Poggio. Biologically-plausible learning algorithms can scale to large datasets. *arXiv preprint arXiv:1811.03567*, 2018.

[64] Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

[65] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[66] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

[67] Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *arXiv preprint arXiv:2006.03824*, 2020.

[68] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

[69] Joel Veness, Tor Lattimore, Avishkar Bhoopchand, Agnieszka Grabska-Barwinska, Christopher Mattern, and Peter Toth. Online learning with gated linear networks. *arXiv preprint arXiv:1712.01897*, 2017.

[70] Joel Veness, Tor Lattimore, David Budden, Avishkar Bhoopchand, Christopher Mattern, Agnieszka Grabska-Barwinska, Eren Sezener, Jianan Wang, Peter Toth, Simon Schmitt, et al. Gated linear networks. *arXiv preprint arXiv:1910.01526*, 2019.

[71] Shanshan Qin, Nayantara Mudur, and Cengiz Pehlevan. Supervised deep similarity matching. *arXiv preprint arXiv:2002.10378*, 2020.

[72] Cengiz Pehlevan, Anirvan M Sengupta, and Dmitri B Chklovskii. Why do similarity matching objectives lead to hebbian/anti-hebbian networks? *Neural computation*, 30(1):84–124, 2018.

[73] Dmitry Krotov and John J Hopfield. Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, 116(16):7723–7731, 2019.

[74] Gregor M Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cerebral cortex*, 24(3):677–690, 2014.

[75] Keith J Todd, Houssam Darabid, and Richard Robitaille. Perisynaptic glia discriminate patterns of motor nerve activity and influence plasticity at the neuromuscular junction. *Journal of Neuroscience*, 30(35):11870–11882, 2010.

[76] Alfonso Araque, Giorgio Carmignoto, Philip G Haydon, Stéphane HR Oliet, Richard Robitaille, and Andrea Volterra. Gliotransmitters travel in time and space. *Neuron*, 81(4):728–739, 2014.

[77] Marta Navarrete and Alfonso Araque. Endocannabinoids potentiate synaptic transmission through stimulation of astrocytes. *Neuron*, 68(1):113–126, 2010.

[78] Olivier Pascual, Kristen B Casper, Cathryn Kubera, Jing Zhang, Raquel Revilla-Sanchez, Jai-Yoon Sul, Hajime Takano, Stephen J Moss, Ken McCarthy, and Philip G Haydon. Astrocytic purinergic signaling coordinates synaptic networks. *Science*, 310(5745):113–116, 2005.

[79] Agnieszka Grabska-Barwińska, Simon Barthelmé, Jeff Beck, Zachary F Mainen, Alexandre Pouget, and Peter E Latham. A probabilistic approach to demixing odors. *Nature neuroscience*, 20(1):98, 2017.

[80] Mengye Ren, Renjie Liao, Raquel Urtasun, Fabian H Sinz, and Richard S

Zemel. Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*, 2016.

[81] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[82] Javier R Movellan. Contrastive hebbian learning in the continuous hopfield model. In *Connectionist models*, pages 10–17. Elsevier, 1991.

[83] Xiaohui Xie and H Sebastian Seung. Equivalence of backpropagation and contrastive hebbian learning in a layered network. *Neural computation*, 15(2):441–454, 2003.

[84] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[85] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[86] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[87] Bernd Illing, Wulfram Gerstner, and Johanni Brea. Biologically plausible deep learning—but how far can we go with shallow networks? *Neural Networks*, 118:90–101, 2019.

[88] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[89] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[90] Michael Laskin, Luke Metz, Seth Nabarro, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates. *arXiv preprint arXiv:2012.03837*, 2020.

[91] Stefano Zappacosta, Francesco Mannella, Marco Mirolli, and Gianluca Baldassarre. General differential hebbian learning: Capturing temporal relations between events in neural networks and the brain. *PLoS computational biology*, 14(8):e1006227, 2018.

[92] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[93] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020.

[94] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[95] Shiyu Duan, Shujian Yu, and Jose Principe. Modularizing deep learning via pairwise learning with kernels. *arXiv preprint arXiv:2005.05541*, 2020.

[96] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[97] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[98] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(12), 2010.

[99] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2016.

[100] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*. 2020.

[101] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*, pages 561–580. Springer, 2012.

[102] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*. 2017.

[103] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *ICLR*, 2019.

[104] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*. 2015.

[105] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, pages 69–84. Springer, 2016.

[106] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *ICLR*, 2018.

[107] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, pages 649–666. Springer, 2016.

[108] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In *ECCV*, pages 577–593. Springer, 2016.

[109] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *CVPR*, pages 2536–2544, 2016.

[110] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *NeurIPS*, 2019.

[111] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *ICML*, 2020.

[112] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2019.

[113] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning, 2020.

[114] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. In *NeurIPS*. 2020.

[115] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021.

[116] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere, 2020.

[117] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning, 2020.

[118] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop*. 2015.

[119] Michael Tschannen, Josip Djolonga, Paul K Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. In *ICLR*, 2020.

[120] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS*, 2007.

[121] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. In *NeurIPS*, 2020.

[122] Yao-Hung Hubert Tsai, Shaojie Bai, Louis-Philippe Morency, and Ruslan Salakhutdinov. A note on connecting barlow twins with negative-sample-free contrastive learning. *arXiv preprint arXiv:2104.13712*, 2021.

[123] Ting Chen and Lala Li. Intriguing properties of contrastive losses, 2020.

[124] Matthew B. Blaschko, Wojciech Zaremba, and Arthur Gretton. Taxonomic prediction with tree-structured covariances. In *Machine Learning and Knowledge Discovery in Databases*, pages 304–319. Springer Berlin Heidelberg, 2013.

[125] Dino Sejdinovic, Bharath Sriperumbudur, Arthur Gretton, and Kenji Fukumizu. Equivalence of distance-based and RKHS-based statistics in hypothesis testing. *The Annals of Statistics*, 41(5):2263 – 2291, 2013.

[126] Gábor J. Székely, Maria L. Rizzo, and Nail K. Bakirov. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769 – 2794, 2007.

[127] Nello Cristianini, John Shawe-Taylor, André Elisseeff, and Jaz Kandola. On kernel-target alignment. In *NeurIPS*, volume 14, 2002.

[128] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

[129] Roland S. Zimmermann, Yash Sharma, Steffen Schneider, Matthias Bethge, and Wieland Brendel. Contrastive learning inverts the data generating process, 2021.

[130] Kai Zhang, Ivor W Tsang, and James T Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th international conference on Machine learning*, pages 1232–1239, 2008.

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

[132] Bernd Illing, Jean Ventura, Guillaume Bellec, and Wulfram Gerstner. Local plasticity rules can learn deep representations using self-supervised contrastive predictions. *Advances in Neural Information Processing Systems*, 34:30365–30379, 2021.

[133] Yang You, Igor Gitman, and Boris Ginsburg. Scaling SGD batch size to 32k for ImageNet training, 2017.

[134] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour, 2017.

[135] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S^4l: Self-supervised semi-supervised learning. In *ICCV*, 2019.

[136] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. In *CVPR Workshop*, 2004.

[137] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[138] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, 2013.

[139] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew B. Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft.

[140] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *ECCV*, pages 446–461. Springer, 2014.

[141] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[142] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *CVPR*, 2012.

[143] Jonathan Krause, Jia Deng, Michael Stark, and Li Fei-fei. Collecting a large-scale dataset of fine-grained cars. the second workshop on fine-grained visual categorization, 2013.

[144] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[145] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *IJCV*, 88(2):303–338, 2010.

[146] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better ImageNet models transfer better? In *CVPR*, 2019.

[147] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[148] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.

[149] Roman Pogodin, Yash Mehta, Timothy Lillicrap, and Peter Latham. Towards biologically plausible convolutional networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[150] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[151] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[152] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[153] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[154] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[155] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[156] Grace W. Lindsay. Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. *Journal of Cognitive Neuroscience*, pages 1–15, 02 2020.

[157] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[158] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[159] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the national academy of sciences*, 111(23):8619–8624, 2014.

[160] Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS computational biology*, 10(11):e1003915, 2014.

[161] Santiago A Cadena, George H Denfield, Edgar Y Walker, Leon A Gatys, Andreas S Tolias, Matthias Bethge, and Alexander S Ecker. Deep convolutional models improve predictions of macaque v1 responses to natural images. *PLoS computational biology*, 15(4):e1006897, 2019.

[162] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[163] Stéphane d'Ascoli, Levent Sagun, Joan Bruna, and Giulio Biroli. Finding the

needle in the haystack with convolutions: on the benefits of architectural bias. *arXiv preprint arXiv:1906.06766*, 2019.

[164] Martin Schrimpf, Jonas Kubilius, Michael J Lee, N Apurva Ratan Murty, Robert Ajemian, and James J DiCarlo. Integrative benchmarking to advance neurally mechanistic models of human intelligence. *Neuron*, 2020.

[165] Behnam Neyshabur. Towards learning convolutions from scratch. *arXiv preprint arXiv:2007.13657*, 2020.

[166] Gamaleldin Elsayed, Prajit Ramachandran, Jonathon Shlens, and Simon Kornblith. Revisiting spatial invariance with low-rank local connectivity. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2868–2879. PMLR, 13–18 Jul 2020.

[167] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[168] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[169] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[170] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

[171] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

[172] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[173] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021.

[174] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015.

[175] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[176] Jordan Ott, Erik J. Linstead, Nicholas LaHaye, and Pierre Baldi. Learning in the machine: To share or not to share? *Neural networks : the official journal of the International Neural Network Society*, 126:235–249, 2020.

[177] Jeremy Freeman, Corey M. Ziemba, David J. Heeger, Eero P. Simoncelli, and J. Anthony Movshon. A functional and perceptual signature of the second visual area in primates. *Nature Neuroscience*, 16(7):974–981, Jul 2013.

[178] Tiago Marques, Martin Schrimpf, and James J DiCarlo. Multi-scale hierarchical neural network models that bridge from single neurons in the primate primary visual cortex to object recognition behavior. *bioRxiv*, 2021.

[179] Najib J. Majaj, Ha Hong, Ethan A. Solomon, and James J. DiCarlo. Simple learned weighted sums of inferior temporal neuronal firing rates accurately predict human core object recognition performance. *Journal of Neuroscience*, 35(39):13402–13418, 2015.

[180] Rishi Rajalingham, Elias B. Issa, Pouya Bashivan, Kohitij Kar, Kailyn Schmidt, and James J. DiCarlo. Large-scale, high-resolution comparison of the core visual object recognition behavior of humans, monkeys, and state-of-the-art deep artificial neural networks. *bioRxiv*, 2018.

[181] Maximilian Riesenhuber and Tomaso A. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[182] Peter Földiák. Learning invariance from transformation sequences. *Neural computation*, 3(2):194–200, 1991.

[183] Guy Wallis, Edmund Rolls, and Peter Foldiak. Learning invariant responses to the natural transformations of objects. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 2, pages 1087–1090. IEEE, 1993.

[184] Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

[185] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[186] Sushil K Jha, Brian E. Jones, Tammi Coleman, Nick Steinmetz, Chi-Tat Law, Gerald D. Griffin, Joshua D. Hawk, Nooreen Dabbish, Valery A. Kalatsky,

and Marcos G. Frank. Sleep-dependent plasticity requires cortical activity. *The Journal of Neuroscience*, 25:9266 – 9274, 2005.

[187] Carlos Puentes-Mestril and Sara J Aton. Linking network activity to synaptic plasticity during sleep: hypotheses and recent data. *Frontiers in neural circuits*, 11:61, 2017.

[188] Jonathan J. Nassi and Edward M. Callaway. Parallel processing strategies of the primate visual system. *Nature Reviews Neuroscience*, 10:360–372, 2009.

[189] Farran Briggs and Edward M. Callaway. Layer-specific input to distinct cell types in layer 6 of monkey primary visual cortex. *The Journal of Neuroscience*, 21:3600 – 3608, 2001.

[190] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[191] William H. Bosking, Ying Zhang, Brett Schofield, and David Fitzpatrick. Orientation selectivity and the arrangement of horizontal connections in tree shrew striate cortex. *J. Neurosci.*, 15, 1997.

[192] Dan D. Stettler, Aniruddha Das, Jean Bennett, and Charles Gilbert. Lateral connectivity and contextual interactions in macaque primary visual cortex. *Neuron*, 36:739–750, 2002.

[193] Roman Pogodin and Peter E Latham. Locally connected networks as ventral stream models. In *Brain-Score Workshop*, 2022.

[194] Ghislain St-Yves, Emily J Allen, Yihan Wu, Kendrick Kay, and Thomas Naselaris. Brain-optimized neural networks learn non-hierarchical models of representation in human visual cortex. *bioRxiv*, 2022.

[195] J KEVIN O'REGAN and Tatjana A Nazir. Some results on translation invariance in the human visual system. *Spatial vision*, 5(2):81–100, 1990.

[196] Loukas Grafakos. *Classical Fourier Analysis*, volume 2. Springer, 2008.

[197] Robert Piessens. The Hankel transform. *The Transforms and Applications Handbook*, 2(9), 2000.

[198] Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*, 2013. `http://mpmath.org/`.

[199] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[200] Olivier J. Hénaff, Skanda Koppula, Jean-Baptiste Alayrac, Aäron van den Oord, Oriol Vinyals, and João Carreira. Efficient visual pretraining with contrastive detection, 2021.

[201] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017.

[202] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[203] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. Sgd: General analysis and improved rates. In *International Conference on Machine Learning*, pages 5200–5209. PMLR, 2019.

[204] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.