Kernelized information bottleneck leads to biologically plausible 3-factor Hebbian learning in deep networks

Roman Pogodin
Gatsby Computational Neuroscience Unit
University College London
London, W1T 4JG
roman.pogodin.17@ucl.ac.uk

Peter E. Latham
Gatsby Computational Neuroscience Unit
University College London
London, W1T 4JG
pel@gatsby.ucl.ac.uk

Abstract

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 forward pass (computation) and a backward pass (learning); and the algorithm needs a large amount of labeled data. 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. Here 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 squeeze as much information as possible out of the input without sacrificing prediction of the output. The resulting rules have a 3-factor Hebbian structure: they require pre- and post-synaptic firing rates and a global error signal – the third factor – that can be supplied by a neuromodulator. Moreover, they do not require precise labels; instead, they rely on the similarity between the desired outputs. They thus solve all three implausibility issues of backpropagation. Moreover, to obtain good performance on hard problems and retain biologically plausible learning rules, our rules need divisive normalization – a known feature of biological networks. Finally, simulations show that our rule performs nearly as well as backpropagation on image classification tasks.

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 [1]. 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 (in terms of either timing or circuitry); and backprop needs a large amount of precisely labeled data.

While approximations to backprop such as feedback alignment [2, 3] can solve the weight transport problem, there have been no plausible solutions to either 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 [4], 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,

Preprint. Under review.

along with a global error signal. Layer-wise training removes the need for both weight transport and a backward pass, and there is a growing evidence that such an approach can work as well as backprop [5, 6, 7]. 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.

Most of our work focuses on finding a layer-wise learning rule that is biologically plausible. For that we turn to the information bottleneck principle. Initially formulated in [8], it was later suggested as an objective for layer-wise training [9] with the following motivation: every layer should minimize the mutual information between its own activity and the input to the network, while maximizing the mutual information between the activity and the correct output (e.g., a label). Estimating the mutual information is hard [10], so [11] proposed a kernelized version of the idea: instead of mutual information they used "kernelized cross-covariance" called Hilbert-Schmidt independence criterion (HSIC). HSIC was originally proposed [12] as a way to measure independence between distributions. Unlike mutual information, HSIC is easy to estimate from data [12], and the information bottleneck objective keeps its intuitive interpretation. Moreover, as we will see, 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 [13]) specific to each layer. In addition, to work on hard problems and remain biologically plausible, our update rules need divisive normalization, a computation done by the primary visual cortex and beyond [14, 15].

In experiments we show that plausible rules generated by pHSIC work nearly as well as backprop on MNIST [16], fashion-MNIST [17], Kuzushiji-MNIST [18] and CIFAR10 [19] datasets. This significantly improves the results in [11], which was the first to use HSIC as an objective function in deep networks.

2 Training deep networks with layer-wise objectives: a kernel methods approach and its plausible approximation

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

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

The standard training approach is to minimize 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 minimize 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 minimized only with respect to the output weights, $\mathbf{W}^{L+1}$.

To choose the layer-wise objective function, we turn to the information bottleneck [8], which minimizes the information between the input and the activity in layer k, while maximizing the information between the activity in layer k and the output [9]. Information, however, is notoriously hard to compute [10], and so [11] proposed an alternative based on the Hilbert-Schmidt Independence Criterion (HSIC). HSIC is a kernel-based method for measuring independence between probability distribution [12]. 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,

$$\min_{\mathbf{W}^k} \mathrm{HSIC}\left(X,\, Z^k\right) - \gamma\, \mathrm{HSIC}\left(Y,\, Z^k\right),\ k = 1, \ldots, L\,, \tag{2}$$

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 becomes

larger as dependence increases,

$$\mathrm{HSIC}\left(A,\, B\right) \,=\, \int \Delta P_{ab}(\mathbf{a}_1, \mathbf{b}_1)\, k(\mathbf{a}_1, \mathbf{a}_2)\, k(\mathbf{b}_1, \mathbf{b}_2)\, \Delta P_{ab}(\mathbf{a}_2, \mathbf{b}_2)\,, \tag{3}$$

where

$$\Delta P_{ab}(\mathbf{a}, \mathbf{b}) \,\equiv (p_{ab}(\mathbf{a}, \mathbf{b}) - p_a(\mathbf{a}) p_b(\mathbf{b}))\, d\mathbf{a}\, d\mathbf{b}\,. \tag{4}$$

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 Appendix A.

HSIC gives us a layer-wise cost function, which eliminates the need for backprop. However, there is a downside: estimating it from data requires memory. This becomes clear when we consider the empirical estimator of Eq. (3) given m observations [12],

$$\begin{aligned}\widehat{\mathrm{HSIC}}(A, B) = &\frac{1}{m^2} \sum_{ij} k(\mathbf{a}_i, \mathbf{a}_j) k(\mathbf{b}_i, \mathbf{b}_j) + \frac{1}{m^2} \sum_{ij} k(\mathbf{a}_i, \mathbf{a}_j) \frac{1}{m^2} \sum_{kl} k(\mathbf{b}_k, \mathbf{b}_l) \\ &- \frac{2}{m^3} \sum_{ijk} k(\mathbf{a}_i, \mathbf{a}_k) k(\mathbf{b}_j, \mathbf{b}_k)\,.\end{aligned} \tag{5}$$

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 (see Fig. 1); for the second, we accumulate and store the average over the kernels (see Eq. (11) 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

$$\frac{1}{m} \sum_{k} k(\mathbf{a}_i, \mathbf{a}_k) k(\mathbf{b}_j, \mathbf{b}_k) \approx \frac{1}{m^2} \sum_{kl} k(\mathbf{a}_i, \mathbf{a}_k) k(\mathbf{b}_j, \mathbf{b}_l)\,. \tag{6}$$

Essentially, we replace the third term in Eq. (5) with the second. This leads to "plausible" HSIC, which we call pHSIC,

$$\mathrm{pHSIC}\left(A,\, B\right) \,=\, \left(\mathbb{E}_{\mathbf{a}_1 \mathbf{b}_1} \mathbb{E}_{\mathbf{a}_2 \mathbf{b}_2} - \mathbb{E}_{\mathbf{a}_1} \mathbb{E}_{\mathbf{b}_1} \mathbb{E}_{\mathbf{a}_2} \mathbb{E}_{\mathbf{b}_2}\right) \left(k(\mathbf{a}_1, \mathbf{a}_2)\, k(\mathbf{b}_1, \mathbf{b}_2)\right)\,. \tag{7}$$

While pHSIC is easier to compute than HSIC, there is still a potential problem: computing $\mathrm{HSIC}\left(X,\, Z^k\right)$ requires $k(\mathbf{x}_i, \mathbf{x}_j)$, as can be seen in the above equation. But if 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 minimizing the dependence between X and Z^k (by minimizing the first term in Eq. (2)), we minimize the (kernelized) covariance of Z^k. To do this, we replace X with Z^k in Eq. (2), and define $\mathrm{pHSIC}\left(A,\, A\right)$ via Eq. (7) but with $p_{ab}(\mathbf{a}, \mathbf{b})$ set to $p_a(\mathbf{a}) \delta(\mathbf{a} - \mathbf{b})$,

$$\mathrm{pHSIC}\left(A,\, A\right) = \mathbb{E}_{\mathbf{a}_1} \mathbb{E}_{\mathbf{a}_2} \left(k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 - \left(\mathbb{E}_{\mathbf{a}_1} \mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2\,. \tag{8}$$

This gives us the new objective,

$$\min_{\mathbf{W}^k} \mathrm{pHSIC}\left(Z^k,\, Z^k\right) - \gamma\, \mathrm{pHSIC}\left(Y,\, Z^k\right),\; k = 1, \ldots, L\,. \tag{9}$$

The new objective preserves the intuition behind the information bottleneck, which is to throw away as much information as possible about the input.

As we show in Appendix A.2, $\mathrm{pHSIC}\left(A,\, A\right) \geq \mathrm{HSIC}\left(A,\, A\right)$, and when $\mathbb{E}_{\mathbf{a}_1} k(\mathbf{a}_1, \mathbf{a}_2) = 0$ or $\mathbb{E}_{\mathbf{b}_1} k(\mathbf{b}_1, \mathbf{b}_2) = 0$, $\mathrm{pHSIC}\left(A,\, B\right) = \mathrm{HSIC}\left(A,\, B\right)$. In our formulation we center $k(\mathbf{y}_1, \mathbf{y}_2)$, which ensures that $\mathbb{E}_{\mathbf{b}_1} k(\mathbf{b}_1, \mathbf{b}_2) = 0$. Consequently, the cost function in Eq. (9) is an upper bound on "true" HSIC objective (i.e. Eq. (2) with $\mathrm{HSIC}\left(Z^k,\, Z^k\right)$ in place of the first term).

When $\gamma = 2$, our objective is related to similarity matching. For the cosine similarity kernel $k(\mathbf{a}_1, \mathbf{a}_2) = \mathbf{a}_1^\top \mathbf{a}_2 / \|\mathbf{a}_1\| \|\mathbf{a}_2\|$, it is close to [6], and for the linear kernel, it is close to [20, 21]. However, the update rule for the cosine similarity kernel is implausible, and for the linear kernel the rule performs poorly (see below and in Appendix B). We thus turn to the Gaussian kernel.

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

3.1 General update rule

To derive the update rule for gradient descent, we need to estimate the gradient of Eq. (9). This is relatively straightforward, so we leave the derivation to Appendix B, and just report the update rule,

$$\Delta \mathbf{W}^k \propto \sum_{ij} \left(\gamma \mathring{k}(\mathbf{y}_i, \mathbf{y}_j) - 2\mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k) \right) \frac{d}{d\,\mathbf{W}^k} k(\mathbf{z}_i^k, \mathbf{z}_j^k)\,, \tag{10}$$

where the circle above k means empirical centering,

$$\mathring{k}(\mathbf{a}_i, \mathbf{a}_j) \equiv k(\mathbf{a}_i, \mathbf{a}_j) - \frac{1}{m^2} \sum_{i'j'} k(\mathbf{a}_{i'}, \mathbf{a}_{j'})\,. \tag{11}$$

This rule has a 3-factor form: for every pair of points i and j, every synapse needs the same multiplier, and this multiplier only needs the similarity between two labels. This is especially simple for the output $\mathbf{y}$: as we compare labels with the centered cosine similarity kernel, we have that $k(\mathbf{y}_i, \mathbf{y}_j) = 1$ if $\mathbf{y}_i = \mathbf{y}_j$ and $-1/(n-1)$ otherwise, where n is the number of classes (see Appendix A.3).

However, the derivative in Eq. (10) is not obviously Hebbian. In fact, it is the case Hebbian and biologically plausible only for specific kernels: below we show that it is for the Gaussian kernel, and in Appendix B.4 we show that it is not for the cosine similarity kernel (used in [6]).

3.2 Gaussian kernel: two-point update

The Gaussian kernel is given by $k(\mathbf{z}_i^k, \mathbf{z}_j^k) = \exp(-\|\mathbf{z}_i^k - \mathbf{z}_j^k\|^2/2\sigma^2)$. To compute updates from a stream of data (rather than batches), we approximate the sum in Eq. (10) with only two points, for which we'll again use i and j. If we take a linear fully connected layer (see Appendix B.2 for the general case), the update over two points is

$$\Delta \mathbf{W}^k \propto M_{ij}^k (\mathbf{z}_i^k - \mathbf{z}_j^k)(\mathbf{z}_i^{k-1} - \mathbf{z}_j^{k-1})^\top, \tag{12a}$$

$$M_{ij}^k = -\frac{1}{\sigma^2} \left(\gamma\, \mathring{k}(\mathbf{y}_i, \mathbf{y}_j) - 2\, \mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k) \right) k(\mathbf{z}_i^k, \mathbf{z}_j^k)\,, \tag{12b}$$

where 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 anti-Hebbian, which tends to move activity farther apart.

3.3 Gaussian kernel: synaptic details

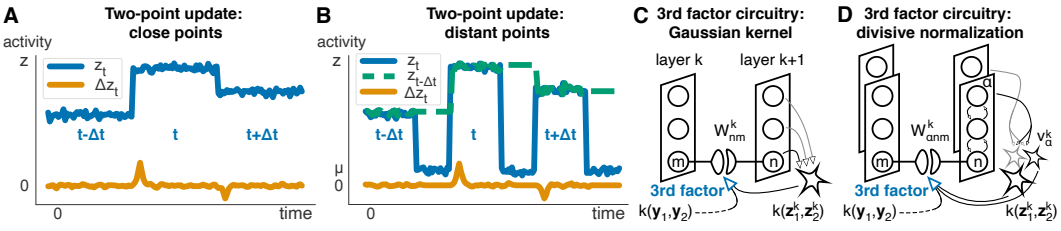

Figure 1: **A.** First scenario of the learning rule for the Gaussian kernel: plasticity (proportional to Δz_t, orange line) happens when the activity (blue line) switches from one data point to another. **B.** Second scenario: plasticity happens when the second point comes in some time after the first one, which uses memorized activity from the first point (dashed green line). **C.** Circuitry for the Gaussian kernel. **D.** Circuitry for the Gaussian kernel with grouping and divisive normalization.

In a realistic setting, the circuit would first see point j and then point i. To make this explicit, we replace i with t and j with $t - \Delta t$,

$$\Delta \mathbf{W}_t^k \propto M_{t,\,t-\Delta t}^k \, (\mathbf{z}_t^k - \mathbf{z}_{t-\Delta t}^k)(\mathbf{z}_t^{k-1} - \mathbf{z}_{t-\Delta t}^{k-1})^\top . \tag{13}$$

This could be implemented with a smoothed temporal derivative, but details depend on the size of Δt.

Two main scenarios are possible. In the first, Δt is fixed and relatively small. In that case, the jump in activity for a single neuron n can be computed by a smoothed temporal derivative, implemented by a convolution with a kernel κ (see Fig. 1A),

$$z_{n,\,t}^k - z_{n,\,t-\Delta t}^k \approx \Delta z_{n,\,t}^k \equiv (\kappa * z_n^k)(t); \quad \kappa(t) \propto -(t - c_1)\, e^{-c_2 |t - c_1|}\, \Theta(t)\,, \tag{14}$$

with c_1 and c_2 are positive.

In the second, Δt is large and potentially variable, and, more importantly, between trials z_n^k returns to some background activity μ_n^k (see Fig. 1B). Thus, the neuron needs to memorize the last significant deviation from the background (i.e., it needs to remember $z_{n,\,t-\Delta t}^k - \mu_{n,\,t-\Delta t}^k$). This can be done with a one-dimensional nonlinear differential equation with "memory", such as

$$\dot{\omega}_{n,\,t} = \left(z_{n,\,t}^k - \mu_{n,\,t}^k\right)^3 - \tanh\left(\left|z_{n,\,t}^k - \mu_{n,\,t}^k\right|^3\right) \omega_{n,\,t} - c\,\omega_{n,\,t} \tag{15}$$

with c small. The intuition is simple: if the neuron is in the background state, the right hand side of Eq. (15) is nearly zero (expect for the leak term $c\,\omega_t$). Otherwise, the deviation from the mean is large and the right hand side approaches $(z_{n,\,t}^k - \mu_{n,\,t}^k)^3 - (1 + c)\,\omega_{n,\,t}$, as long as $|z^k - \mu_{n,\,t}^k| \gg 1$ (due to tanh saturation). This quickly erases the previous $z_{n,\,t-\Delta t}^k$ and memorizes the new one (see Fig. 1B with no leak term). We can therefore compute the difference between the last and the current large deviations by convolving the (real) cube root $(\omega_{n,\,t}^k)^{1/3}$ with the same kernel as above, leading to

$$\mathbf{z}_{n,\,t}^k - \mathbf{z}_{n,\,t-\Delta t}^k \approx \Delta z_{n,\,t}^k \equiv \left(\kappa * (\omega_n^k)^{1/3}\right)(t)\,. \tag{16}$$

Both scenarios share the same circuitry of a third factor computation: they need access to the teaching signal, $\mathring{k}(\mathbf{y}_i, \mathbf{y}_j)$, and the magnitude of activity changes, $\|\mathbf{z}_i^k - \mathbf{z}_j^k\|^2$, to compute $k(\mathbf{z}_i^k, \mathbf{z}_j^k)$ and its centered version, $\mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k)$ (summarized in Fig. 1C, see a more detailed model in Appendix C).

3.4 Gaussian kernel with divisive normalization

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, [6] computed the variance of the response over each channel in each convolutional layers, and then used it for cosine similarity.

We will use the same approach, which starts by arranging neurons into c^k groups, labeled by α, so that $z_{\alpha n}^k$ is the response of neuron n in group α of layer k. We'll characterize each group by its "smoothed" variance (meaning we add a positive offset δ), denoted u_α^k,

$$u_\alpha^k \equiv \frac{\delta}{c_\alpha^k} + \frac{1}{c_\alpha^k} \sum_{n'} \left(\mathring{z}_{\alpha n'}^k\right)^2; \quad \mathring{z}_{\alpha n}^k \equiv z_{\alpha n}^k - \frac{1}{c_\alpha^k} \sum_{n'} z_{\alpha n'}^k, \tag{17}$$

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

$$v_\alpha^k = (u_\alpha^k)^{1-p} - \frac{1}{c^k} \sum_{\alpha'} (u_{\alpha'}^k)^{1-p}\,, \tag{18}$$

and use it in the Gaussian kernel,

$$k(\mathbf{z}_i^k, \mathbf{z}_j^k) = \exp\left(-\frac{1}{2\sigma^2} \left\|\mathbf{v}_i^k - \mathbf{v}_j^k\right\|^2\right), \tag{19}$$

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

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

$$\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,W_{\alpha nm}^k} \propto -\frac{1}{\sigma^2} k(\mathbf{z}_i^k, \mathbf{z}_j^k)\,(v_{\alpha,\,i}^k - v_{\alpha,\,j}^k) \left(\frac{\mathring{z}_{\alpha n,\,j}^k}{(u_{\alpha,\,i}^k)^p} z_{m,\,i}^{k-1} - \frac{\mathring{z}_{\alpha n,\,j}^k}{(u_{\alpha,\,j}^k)^p} z_{m,\,j}^{k-1} \right). \tag{20}$$

Because of the term u_α^k 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 $\mathring{z}_{\alpha n,\,j}^k / (u_{\alpha,\,j}^k)^p$,

$$z_{\alpha n}^k = f\left(\sum_m W_{\alpha nm}^k r_m^{k-1} \right); \quad r_{\alpha n}^k = \frac{\mathring{z}_{\alpha n}^k}{(u_\alpha^k)^p}\,. \tag{21}$$

This automatically introduces divisive normalization into the network, a common "canonical computation" in the brain [14], and in our case makes the update 3-factor Hebbian. It also changes the network from Eq. (1) to Eq. (21), but that turns out to improve performance. The resulting update rule (again for a linear network; see Appendix B.3 for the general case) is

$$\begin{aligned} \Delta W_{\alpha nm}^k &\propto M_{\alpha,\,ij}^k \left(r_{\alpha n,\,i}^k r_{m,\,i}^{k-1} - r_{\alpha n,\,j}^k r_{m,\,j}^{k-1} \right) \\ M_{\alpha,\,ij}^k &= -\frac{1}{\sigma^2} \left(\gamma\, k(\mathbf{y}_i, \mathbf{y}_j) - 2\, k(\mathbf{z}_i^k, \mathbf{z}_j^k) \right) k(\mathbf{z}_i^k, \mathbf{z}_j^k)\,(v_{\alpha,\,i}^k - v_{\alpha,\,j}^k)\,. \end{aligned} \tag{22}$$

The circuitry to implement divisive normalization would be recurrent, but is out of scope of this work (see, however, [22] for network models of divisive normalization). For a convolutional layer, α would denote channels (or groups of channels), and the weights would be summed within each channel for weight sharing.

Note that the rule is slightly different from the one for the basic Gaussian kernel (Eq. (12)): the weight change is no longer proportional to differences of pre- and post-synaptic activities; instead, it is proportional to differences in their product times the global factor for the channel. The circuitry is more complicated as well (see Fig. 1D): we need to track changes in the normalization $v_{\alpha,\,i}^k$ and the whole Hebbian term $r_{\alpha n,\,i}^k r_{m,\,i}^{k-1}$, and every channel needs recurrent connections to implement divisive normalization. However, because we introduced divisive normalization, the rule stays Hebbian.

4 Experiments

4.1 Experimental setup

We trained the networks with stochastic gradient descent (SGD) with momentum, but without adaptive optimizers or batch normalization. While networks with adaptive optimizers (e.g. Adam [23]) and batch normalization (or batchnorm, [24]) perform better on deep learning tasks and are often used for biologically plausible algorithms (e.g. [2, 6]), these features imply non-trivial ciruitry (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. (10)) 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 [6]. (Note, however, that the cosine similarity kernel gives implausible update rules, see Appendix B.) For both kernels we tested 2 variants of the rules: plain (without grouping), and with grouping and divisive normalization (as in Eq. (22)). (Grouping *without* divisive normalization performed as well or worse, and we don't report it here as the resulting update rules are less plausible; see Appendix D.) We also tested backprop, and learning of only the output layer in small-scale experiments to have a baseline result (also with divisive normalization in the network). We compared centered labels with the cosine similarity kernel, which in our case ensures the pHSIC objective Eq. (9) is an upper bound on the HSIC objective (see Appendix A).

The nonlinearity for small networks was leaky ReLU (LReLU [25]; with a slope of 0.1 for negative input), but for convolutional networks trained with SGD we changed it to SELU [26] as it performed better. All parameters (including learning rates, learning rate schedules, grouping dimensionality, kernel parameters) were tuned on a validation set (10% of the training set). Optimizing

HSIC instead of our approximation, pHSIC, didn't improve performance, and the original formulation of the kernelized bottleneck (Eq. (2)) performed much worse (not shown). The datasets were: MNIST [16], fashion-MNIST [17], Kuzushiji-MNIST [18] and CIFAR10 [19]. We used standard data augmentation for CIFAR10 [19], but no augmentation for the other datasets. All simulation parameters are provided in Appendix D. The implementation is available on GitHub: https://github.com/romanpogodin/plausible-kernelized-bottleneck.

4.2 Small fully connected network

We start with a 3-layer fully connected network (1024 neurons in each layer) to show how the proposed rules compare to each other and to backprop on a variety of tasks. The models were trained for 100 epochs with dropout [27] of 0.05, batch size of 256, and $\gamma = 2$ (other values of γ performed worse); other parameters are provided in Appendix D.

Our results (summarized in Table 1 for mean test accuracy; see Appendix D for deviations between max and min) have a few clear trends across all four datasets: the kernel-based approaches with grouping and divisive normalization perform similarly to backprop on easy datasets (MNIST and its slightly harder analogues), but not on CIFAR10; grouping and divisive normalization had little effect on the cosine similarity performance; and the plain Gaussian kernel performs only marginally better than training just the output layer. However, we'll see that the poor performance on CIFAR10 is not fundamental to the method; it's because the network is too small.

Table 1: Mean test accuracy over 5 random seeds for a 3-layer fully connected net (1024 neurons per layer). The range of rules includes backprop, SGD over the last layer, cosine similarity (cossim) and the Gaussian kernel (all either plain, or with grouping and divisive normalization, denoted "grp+div").

	backprop		last layer		cossim		Gaussian	
		grp+div		grp+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

4.3 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 [6]: **conv128-conv256-maxpool-conv256-conv512-maxpool-conv512-maxpool-conv512-maxpool-fc1024** and its version with double the convolutional channels (denoted 2x; the size of the fully connected layer remained the same). Other similar architectures or deeper networks performed about as well (not shown). For better convergence we used the SELU nonlinearity [26]. All networks were trained for 500 epochs with $\gamma = 2$, dropout of 0.05 and batch size of 128 (accuracy jumps in Fig. 2 indicate learning rate decreases); the rest of the parameters are provided in Appendix D.

We compared the Gaussian kernel with backprop and the cosine similarity kernel, all with grouping and divisive normalization (summarized in Table 2 for mean test accuracy; see Appendix D for deviations between max and min). On the 1x net (see Fig. 2A), the Gaussian kernel achieved decent performance (86.2% accuracy average over 5 runs), but was significantly worse than backprop (91.0% average) and cosine similarity (89.8% average). However, on a wider net (Fig. 2B) the Gaussian kernel nearly matched performance of both rules (90.4% vs. 90.9% - 91.9% for backprop and 91.3% for cosine similarity), and also had nearly perfect training accuracy. For backprop without grouping and divisive normalization (solid pink in Fig. 2), the learning curve behaves unusually due to the high learning rate. We can achieve a monotonic increasing learning curve if we use a smaller learning rate, but at the cost of slightly worse performance.

Grouping and divisive normalization was crucial for the Gaussian kernel, which achieved below 40% test accuracy without it, but not for cosine similarity (see Appendix D). We also trained the networks with AdamW [28] and batchnorm [24]; although not biologically plausible, these are common in deep learning applications (for discussion see Appendix D).

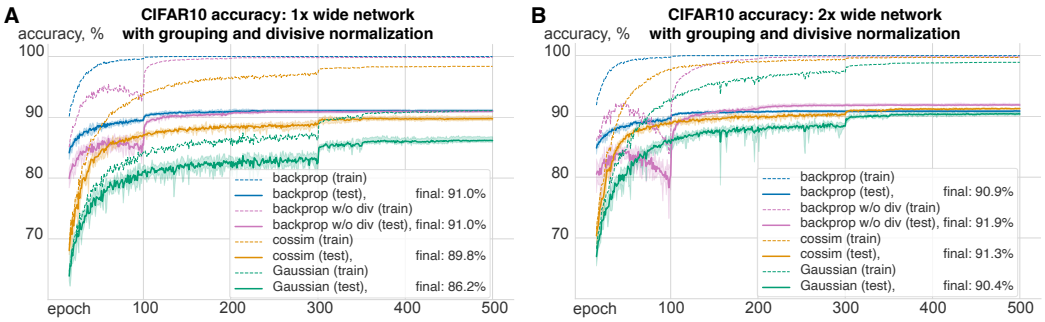

Figure 2: Performance of backprop, cosine similarity kernel (cossim) and Gaussian kernel on CIFAR10 with grouping and divisive normalization (and without for backprop; in pink). Solid lines: mean test accuracy over 5 random seeds; shaded areas: min/max test accuracy over 5 seeds; dashed lines: mean training 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.

Table 2: Mean test accuracy on CIFAR10 over 5 random seeds for the 1x and 2x wide networks. Same range of rules as Table 1, but without the plain version for the cossim or Gaussian kernels.

	backprop		cossim	Gaussian
		grp+div	grp+div	grp+div
1x wide net	91.0	91.0	89.8	86.2
2x wide net	91.9	90.9	91.3	90.4

5 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 works 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 learning rules do not need precise labels, but instead a simple binary signal – whether or not the previous and the current points have the same label. This allows networks to build representations with weaker supervision than those trained with backprop. 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.

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 object 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 activity within the layer. This could be done by cells that influence plasticity, such as dopaminergic neurons [13] or glia [29]. Such cells could also provide a plausible way to implement convolutional-like layers: if synapses receive the same modulation (the third factor), they would have similar weights, and therefore a convolutional structure. However, that would require an even more structured implementation of the third factor.

How do our results compare to other work on biologically plausible learning rules? Most of that work has focused on feedback alignment, with [2] and without [30] adaptive feedback. The former essentially performs backprop; the latter does not, but still exhibits good performance. However, both need a backward pass and precise labels; the backward pass especially is not biologically plausible. The similarity matching objective in [6], which corresponds to the cosine similarity kernel with grouping, is a layer-wise objective, and thus potentially biologically plausible. However, in our hands we could not find a biologically plausible implementation (see Appendix B).

It should be possible to scale our learning rules to larger datasets, such as CIFAR100 [19] and ImageNet [31], as suggested by results from other layer-wise rules [5, 6, 7]. The layer-wise objectives can also make the theoretical analysis of deep learning easier. In fact, recent work analyzed a similar type of kernel-based objectives, showing its optimality with one hidden layer and backprop-comparable performance in deeper networks [32].

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 for solving 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.

Acknowledgments

This work was supported by the Gatsby Charitable Foundation.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

[7] 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.

[8] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[9] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[10] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

[11] Wan-Duo Kurt Ma, JP Lewis, and W Bastiaan Kleijn. The hsic bottleneck: Deep learning without back-propagation. *arXiv preprint arXiv:1908.01580*, 2019.

[12] 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.

[13] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

[14] Matteo Carandini and David J Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51, 2012.

[15] Shawn R Olsen, Vikas Bhandawat, and Rachel I Wilson. Divisive normalization in olfactory population codes. *Neuron*, 66(2):287–299, 2010.

[16] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[18] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature, 2018.

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] 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.

[21] Shanshan Qin, Nayantara Mudur, and Cengiz Pehlevan. Supervised deep similarity matching. *arXiv preprint arXiv:2002.10378*, 2020.

[22] 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.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[25] 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.

[26] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[27] 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.

[28] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[29] 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.

[30] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[31] 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.

[32] Shiyu Duan, Shujian Yu, and Jose Principe. Modularizing deep learning via pairwise learning with kernels. *arXiv preprint arXiv:2005.05541*, 2020.

Appendices

A Kernel methods, HSIC and pHSIC

A.1 Kernels and HSIC

A kernel is a symmetric function $k(\mathbf{a}_1, \mathbf{a}_2) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ that is also positive-definite,

$$\forall\, \mathbf{a}_i \in \mathbb{R}^n,\, \forall c_i \in \mathbb{R}, \quad \sum_{ij} c_i c_j k(\mathbf{a}_i, \mathbf{a}_j) \geq 0\,. \tag{23}$$

In other words, a matrix $\mathbf{K}$ with $K_{ij} = k(\mathbf{a}_i, \mathbf{a}_j)$ needs to be positive-semidefinite.

We will use the following kernels,

$$\text{linear:} \qquad k(\mathbf{a}_i, \mathbf{a}_j) = \mathbf{a}_i^\top \mathbf{a}_j\,; \tag{24}$$

$$\text{cosine similarity:} \qquad k(\mathbf{a}_i, \mathbf{a}_j) = \mathbf{a}_i^\top \mathbf{a}_j / (\|\mathbf{a}_i\|_2 \, \|\mathbf{a}_j\|_2)\,; \tag{25}$$

$$\text{Gaussian:} \qquad k(\mathbf{a}_i, \mathbf{a}_j) = \exp(-\|\mathbf{a}_i - \mathbf{a}_j\|_2^2 / (2\sigma^2))\,. \tag{26}$$

The Hilbert-Schmidt Independence Criterion (introduced in [12]) measures independence between two random variable, A and B, with kernels k_a and k_b,

$$\mathrm{HSIC}\,(A,\, B) = (\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2\mathbf{b}_2} - 2\,\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2} + \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2})\,(k(\mathbf{a}_1, \mathbf{a}_2)\,k(\mathbf{b}_1, \mathbf{b}_2))\,, \tag{27}$$

where $\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}$ denotes expectation over the joint distribution.

When $B \equiv A$, meaning that $p_{ab}(\mathbf{a}, \mathbf{b}) = p_a(\mathbf{a})\,\delta(\mathbf{b} - \mathbf{a})$, HSIC becomes

$$\mathrm{HSIC}\,(A,\, A) = \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)^2 - 2\,\mathbb{E}_{\mathbf{a}_1}\left(\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 + \left(\mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2\,. \tag{28}$$

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

$$\mathrm{HSIC}\,(A,\, B) = \|\mathbf{C}_{\mathbf{ab}}\|_F^2\,, \quad \mathbf{C}_{\mathbf{ab}} = \mathbb{E}_{\mathbf{ab}}\,\mathbf{a}\mathbf{b}^\top\,. \tag{29}$$

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

A.2 pHSIC

We define the "plausible" HSIC by substituting $\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2}$ in Eq. (27) by $\mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2}$ so neurons don't have to remember many data points to compute this expectation,

$$\mathrm{pHSIC}\,(A,\, B) = (\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2\mathbf{b}_2} - \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{b}_1}\mathbb{E}_{\mathbf{a}_2}\mathbb{E}_{\mathbf{b}_2})\,(k(\mathbf{a}_1, \mathbf{a}_2)\,k(\mathbf{b}_1, \mathbf{b}_2))\,. \tag{30}$$

Therefore, $\mathrm{HSIC}\,(A,\, B) = \mathrm{pHSIC}\,(A,\, B)$ when $\mathbf{a}$ and $\mathbf{b}$ are independent $\mathbb{E}_{\mathbf{a}_1\mathbf{b}_1} = \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{b}_1}$ (which is not useful in our case), and when $\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2) = 0$ or $\mathbb{E}_{\mathbf{b}_2} k(\mathbf{b}_1, \mathbf{b}_2) = 0$.

In addition, $\mathrm{pHSIC}\,(A,\, A)$ becomes the variance of $k(\mathbf{a}_1, \mathbf{a}_2)$ with respect to $p_a(\mathbf{a}_1)p_a(\mathbf{a}_2)$,

$$\mathrm{pHSIC}\,(A,\, A) = \mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2}\left(k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 - \left(\mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 = \mathbb{V}\mathrm{ar}\left(k(\mathbf{a}_1, \mathbf{a}_2)\right)\,, \tag{31}$$

and we can show that $\mathrm{HSIC}\,(A,\, A) \leq \mathrm{pHSIC}\,(A,\, A)$ by combining Eq. (28) and Eq. (31) and denoting $\mu_a(\mathbf{a}_1) = \mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)$,

$$\begin{aligned}
\mathrm{pHSIC}\,(A,\, A) - \mathrm{HSIC}\,(A,\, A) &= 2\,\mathbb{E}_{\mathbf{a}_1}\left(\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 - 2\,\left(\mathbb{E}_{\mathbf{a}_1}\mathbb{E}_{\mathbf{a}_2} k(\mathbf{a}_1, \mathbf{a}_2)\right)^2 \\
&= 2\left(\mathbb{E}_{\mathbf{a}_1}\mu_a(\mathbf{a}_1)^2 - \left(\mathbb{E}_{\mathbf{a}_1}\mu_a(\mathbf{a}_1)\right)^2\right) \\
&= 2\,\mathbb{V}\mathrm{ar}\left(\mu_a(\mathbf{a}_1)\right) \geq 0.
\end{aligned} \tag{32}$$

As a result, our objective,

$$\mathrm{pHSIC}\left(Z^k,\, Z^k\right) - \gamma\,\mathrm{pHSIC}\left(Y,\, Z^k\right)\,, \tag{33}$$

is an upper bound on the "true" objective with HSIC if $\mathbb{E}_{\mathbf{y}_2} k(\mathbf{y}_1, \mathbf{y}_2) = 0$ and consequently $\mathrm{pHSIC}\left(Y,\, Z^k\right)$ is equal to $\mathrm{HSIC}\left(Y,\, Z^k\right)$.

Finally, the empirical estimate of pHSIC (which can be derived just like for HSIC in [12]) is

$$\widehat{\mathrm{pHSIC}}(A, B) = \frac{1}{m^2} \sum_{ij} k(\mathbf{a}_i, \mathbf{a}_j) k(\mathbf{b}_i, \mathbf{b}_j) - \frac{1}{m^2} \sum_{ij} k(\mathbf{a}_i, \mathbf{a}_j) \frac{1}{m^2} \sum_{ql} k(\mathbf{b}_q, \mathbf{b}_l)\,. \tag{34}$$

A.3 How much label information do we need?

The label information in our rules 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).

In our analysis we used the cosine similarity kernel with $\mathbf{y}$ centered. 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

$$k(\mathbf{y}_i, \mathbf{y}_j) = \frac{\left(\mathbf{y}_i - \frac{1}{n}\mathbf{1}_n\right)^\top \left(\mathbf{y}_j - \frac{1}{n}\mathbf{1}_n\right)}{\left\|\mathbf{y}_i - \frac{1}{n}\mathbf{1}_n\right\| \left\|\mathbf{y}_j - \frac{1}{n}\mathbf{1}_n\right\|} = \frac{\mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] - \frac{1}{n}}{1 - \frac{1}{n}} = \begin{cases} 1, & \mathbf{y}_i = \mathbf{y}_j\,, \\ -\frac{1}{n-1}, & \text{otherwise}\,. \end{cases} \tag{35}$$

If there are many classes, this signal approaches $\mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right]$, which is what we get for the uncentered linear kernel.

Eq. (35) 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 centering of $\mathbf{y}$ doesn't make the normalized vector centered. However, taking the linear kernel gives an almost binary signal,

$$\begin{aligned} k(\mathbf{y}_i, \mathbf{y}_j) &= (\mathbf{y}_i - \mathbf{p})^\top (\mathbf{y}_j - \mathbf{p}) = \mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] + \sum_k p_k^2 - p_i - p_j \\ &= \mathbb{I}\left[\mathbf{y}_i = \mathbf{y}_j\right] + \mathcal{O}\left(\frac{1}{n}\right), \end{aligned} \tag{36}$$

as long as the probability of each class k is $p_k = \mathcal{O}(1/n)$ (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.

B Derivations of the update rules for plausible kernelized information bottleneck

B.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 normalization). 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},

$$\mathbb{E}_{\mathbf{y}\mathbf{z}^k} g(\mathbf{y}, \mathbf{z}^k) = \mathbb{E}_{\mathbf{y}\mathbf{z}^{k-1}} g\left(\mathbf{y}, f\left(\mathbf{W}^k, \mathbf{z}^{k-1}\right)\right)\,. \tag{37}$$

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

$$\begin{aligned} \frac{d\,\mathrm{pHSIC}\left(Y,\, Z^k\right)}{d\,\mathbf{W}^k} &= \left(\mathbb{E}_{\mathbf{y}_1 \mathbf{z}_1^{k-1}} \mathbb{E}_{\mathbf{y}_2 \mathbf{z}_2^{k-1}} - \mathbb{E}_{\mathbf{y}_1} \mathbb{E}_{\mathbf{z}_1^{k-1}} \mathbb{E}_{\mathbf{y}_2} \mathbb{E}_{\mathbf{z}_2^{k-1}}\right) \\ &\qquad k(\mathbf{y}_1, \mathbf{y}_2)\, \frac{d\, k\left(f\left(\mathbf{W}^k, \mathbf{z}_1^{k-1}\right), f\left(\mathbf{W}^k, \mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k}\,, \end{aligned} \tag{38a}$$

$$\begin{aligned} \frac{d\,\mathrm{pHSIC}\left(Z^k,\, Z^k\right)}{d\,\mathbf{W}^k} &= 2\, \mathbb{E}_{\mathbf{z}_1^{k-1}} \mathbb{E}_{\mathbf{z}_2^{k-1}} k(\mathbf{z}_1^k, \mathbf{z}_2^k) \frac{d\, k\left(f\left(\mathbf{W}^k, \mathbf{z}_1^{k-1}\right), f\left(\mathbf{W}^k, \mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k} \\ &\quad - 2\, \left(\mathbb{E}_{\mathbf{z}_1^{k-1}} \mathbb{E}_{\mathbf{z}_2^{k-1}} k(\mathbf{z}_1^k, \mathbf{z}_2^{k-1})\right) \left(\mathbb{E}_{\mathbf{z}_1^{k-1}} \mathbb{E}_{\mathbf{z}_2^{k-1}} \frac{d\, k\left(f\left(\mathbf{W}^k, \mathbf{z}_1^{k-1}\right), f\left(\mathbf{W}^k, \mathbf{z}_2^{k-1}\right)\right)}{d\,\mathbf{W}^k}\right)\,. \end{aligned} \tag{38b}$$

To compute these derivates from data, we take empirical averages (see Eq. (34)),

$$\begin{aligned}
&\frac{d\left(\widehat{\mathrm{pHSIC}}(Z^k, Z^k) - \gamma\,\widehat{\mathrm{pHSIC}}(Y, Z^k)\right)}{d\,\mathbf{W}^k} = \\
&\quad 2\,\frac{1}{m^2}\sum_{ij} k(\mathbf{z}_i^k, \mathbf{z}_j^k)\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,\mathbf{W}^k} - 2\,\frac{1}{m^2}\sum_{ql} k(\mathbf{z}_q^k, \mathbf{z}_l^k)\frac{1}{m^2}\sum_{ij}\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,\mathbf{W}^k} \\
&\quad - \gamma\,\frac{1}{m^2}\sum_{ij} k(\mathbf{y}_i, \mathbf{y}_j)\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,\mathbf{W}^k} + \gamma\,\frac{1}{m^2}\sum_{ql} k(\mathbf{y}_q, \mathbf{y}_l)\frac{1}{m^2}\sum_{ij}\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,\mathbf{W}^k}.
\end{aligned} \tag{39}$$

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

$$\begin{aligned}
&\frac{d\left(\widehat{\mathrm{pHSIC}}(Z^k, Z^k) - \gamma\,\widehat{\mathrm{pHSIC}}(Y, Z^k)\right)}{d\,\mathbf{W}^k} \\
&\qquad = \frac{1}{m^2}\sum_{ij}\left(2\,\mathring{k}(\mathbf{z}_i^k, \mathbf{z}_j^k) - \gamma\,\mathring{k}(\mathbf{y}_i, \mathbf{y}_j)\right)\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,\mathbf{W}^k}\,.
\end{aligned} \tag{40}$$

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.

B.2 Gaussian kernel

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

$$\begin{aligned}
\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,W_{nm}^k} &= \frac{d}{d\,W_{nm}^k}\exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{z}_i^k - \mathbf{z}_j^k\right\|^2\right) \\
&= -\frac{k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{\sigma^2}\left(z_{n,\,i}^k - z_{n,\,j}^k\right)\frac{d\,\left(z_{n,\,i}^k - z_{n,\,j}^k\right)}{d\,W_{nm}^k} \\
&= -\frac{k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{\sigma^2}\left(z_{n,\,i}^k - z_{n,\,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1} - f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}\right).
\end{aligned} \tag{41}$$

For the linear network $f'(x) = 1$, and so the gradient w.r.t. $\mathbf{W}^k$ becomes an outer product (Eq. (12)).

B.3 Gaussian kernel with grouping and divisive normalization

For the circuit with grouping and divisive normalization, the kernel is a function of $\mathbf{v}$, not $\mathbf{z}$ (see Eqs. (17), (18) and (19)). This makes the derivative with respect to $\mathbf{z}$ more complicated than the above expression would suggest. Specifically, using Eq. (19) for the kernel with grouping, we have

$$\begin{aligned}
\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,W_{\alpha nm}^k} &= \frac{d}{d\,W_{\alpha nm}^k}\exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{v}_i^k - \mathbf{v}_j^k\right\|^2\right) \\
&= -\frac{k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{\sigma^2}\sum_{\alpha'}(v_{\alpha',\,i}^k - v_{\alpha',\,j}^k)\frac{d\,(v_{\alpha',\,i}^k - v_{\alpha',\,j}^k)}{d\,W_{\alpha nm}^k}\,,
\end{aligned} \tag{42}$$

where the sum over α' appears due to centering (so all α are coupled). Using Eq. (18) to express $\mathbf{v}$ in terms of $\mathbf{u}$, the above derivative is

$$\frac{d\,v_{\alpha',\,i}^k}{d\,W_{\alpha nm}^k} = \frac{d\,\left((u_{\alpha',\,i}^k)^{1-p} - \frac{1}{c_\alpha^k}\sum_{\alpha''}(u_{\alpha'',\,i}^k)^{1-p}\right)}{d\,W_{\alpha nm}^k} = \left(\delta_{\alpha\alpha'} - \frac{1}{c_\alpha^k}\right)\frac{d\,(u_{\alpha,\,i}^k)^{1-p}}{d\,W_{\alpha nm}^k}\,. \tag{43}$$

Then using Eq. (17) to express $\mathbf{u}$ in terms of $\mathbf{z}$, we have

$$\begin{aligned}
\frac{d\,(u^k_{\alpha,\,i})^{1-p}}{d\,W^k_{\alpha nm}} &= \frac{1-p}{(u^k_{\alpha,\,i})^p}\frac{d\left(\frac{\delta}{c^k_\alpha}+\frac{1}{c^k_\alpha}\sum_{n'}\left(\mathring{z}^k_{\alpha n',\,i}\right)^2\right)}{d\,W^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,\,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',\,i}\frac{d\,\mathring{z}^k_{\alpha n',\,i}}{d\,W^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,\,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',\,i}\frac{d\left(z^k_{\alpha n',\,i}-\frac{1}{c^k_\alpha}\sum_{n''}z^k_{\alpha n'',\,i}\right)}{d\,W^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,\,i})^p}\frac{2}{c^k_\alpha}\sum_{n'}\mathring{z}^k_{\alpha n',\,i}\left(\delta_{n'n}-\frac{1}{c^k_\alpha}\right)\frac{d\,z^k_{\alpha n,\,i}}{d\,W^k_{\alpha nm}} \\
&= \frac{1-p}{(u^k_{\alpha,\,i})^p}\frac{2}{c^k_\alpha}\mathring{z}^k_{\alpha n,\,i}\frac{d\,z^k_{\alpha n,\,i}}{d\,W^k_{\alpha nm}}\,.
\end{aligned} \tag{44}$$

Inserting this expression into Eq. (43), inserting that into Eq. (42), and performing a small amount of algebra, we arrive at

$$\frac{d\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{d\,W^k_{\alpha nm}} = -\frac{k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{\sigma^2}\sum_{\alpha'}(v^k_{\alpha',\,i}-v^k_{\alpha',\,j})\left(\delta_{\alpha\alpha'}-\frac{1}{c^k_\alpha}\right)\frac{2\,(1-p)}{c^k_\alpha} \tag{45}$$

$$\times\left(\frac{\mathring{z}^k_{\alpha n,\,i}}{(u^k_{\alpha,\,i})^p}\frac{d\,z^k_{\alpha n,\,i}}{d\,W^k_{\alpha nm}}-\frac{\mathring{z}^k_{\alpha n,\,j}}{(u^k_{\alpha,\,j})^p}\frac{d\,z^k_{\alpha n,\,j}}{d\,W^k_{\alpha nm}}\right). \tag{46}$$

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

$$\frac{d\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{d\,W^k_{\alpha nm}} = -\frac{2\,(1-p)\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{\sigma^2 c^k_\alpha}(v^k_{\alpha,\,i}-v^k_{\alpha,\,j}) \tag{47}$$

$$\times\left(\frac{\mathring{z}^k_{\alpha n,\,i}}{(u^k_{\alpha,\,i})^p}f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,\,i}-\frac{\mathring{z}^k_{\alpha n,\,j}}{(u^k_{\alpha,\,j})^p}f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j}\right). \tag{48}$$

B.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

$$\begin{aligned}
\frac{d\,k(\mathbf{z}^k_i,\mathbf{z}^k_j)}{d\,W^k_{nm}} &= \frac{d}{d\,W^k_{nm}}\frac{(\mathbf{z}^k_i)^\top\mathbf{z}^k_j}{\left\|\mathbf{z}^k_i\right\|\left\|\mathbf{z}^k_j\right\|} \\
&= \frac{1}{\left\|\mathbf{z}^k_i\right\|\left\|\mathbf{z}^k_j\right\|}\frac{d\left(z^k_{n,\,i}z^k_{n,\,j}\right)}{d\,W^k_{nm}}-\frac{(\mathbf{z}^k_i)^\top\mathbf{z}^k_j}{\left\|\mathbf{z}^k_i\right\|^2\left\|\mathbf{z}^k_j\right\|^2}\frac{d\left\|\mathbf{z}^k_i\right\|\left\|\mathbf{z}^k_j\right\|}{d\,W^k_{nm}}\,.
\end{aligned} \tag{49}$$

The first derivative is simple,

$$\frac{d\left(z^k_{n,\,i}z^k_{n,\,j}\right)}{d\,W^k_{nm}} = z^k_{n,\,i}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j}+z^k_{n,\,j}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,\,i}\,. \tag{50}$$

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 can re-arrange it into differences in activity:

$$\begin{aligned}
\frac{d\left(z^k_{n,\,i}z^k_{n,\,j}\right)}{d\,W^k_{nm}} &= \left(z^k_{n,\,i}-z^k_{n,\,j}\right)f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j}+z^k_{n,\,j}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j} \\
&\quad+\left(z^k_{n,\,j}-z^k_{n,\,i}\right)f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,\,i}+z^k_{n,\,i}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,i} \\
&= -\left(z^k_{n,\,i}-z^k_{n,\,j}\right)\left(f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,\,i}-f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j}\right) \\
&\quad+z^k_{n,\,i}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_i\right)_n z^{k-1}_{m,\,i}+z^k_{n,\,j}\,f'\left(\mathbf{W}^k\mathbf{z}^{k-1}_j\right)_n z^{k-1}_{m,\,j}\,.
\end{aligned} \tag{51}$$

The second one is slightly harder,

$$\begin{aligned}\frac{d\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}{d\,W_{nm}^k} &= \left\|\mathbf{z}_i^k\right\|\frac{d\sqrt{\sum_{n'}(z_{n',\,j}^k)^2}}{d\,W_{nm}^k}+\left\|\mathbf{z}_j^k\right\|\frac{d\sqrt{\sum_{n'}(z_{n',\,i}^k)^2}}{d\,W_{nm}^k}\\ &= \frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,\,j}^k\frac{d\,z_{n,\,j}^k}{d\,W_{nm}^k}+\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,\,i}^k\frac{d\,z_{n,\,i}^k}{d\,W_{nm}^k}\\ &= \frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,\,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}+\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,\,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1}\,.\end{aligned} \tag{52}$$

Grouping these results together,

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{nm}^k} &= -\frac{\left(z_{n,\,i}^k-z_{n,\,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1}-f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}\right)}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\\ &\quad+\frac{z_{n,\,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1}}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}+\frac{z_{n,\,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\\ &\quad-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|^2\left\|\mathbf{z}_j^k\right\|^2}\frac{\left\|\mathbf{z}_i^k\right\|}{\left\|\mathbf{z}_j^k\right\|}z_{n,\,j}^k f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}\\ &\quad-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_i^k\right\|^2\left\|\mathbf{z}_j^k\right\|^2}\frac{\left\|\mathbf{z}_j^k\right\|}{\left\|\mathbf{z}_i^k\right\|}z_{n,\,i}^k f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1}\,.\end{aligned} \tag{53}$$

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

$$\begin{aligned}\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|\frac{d\,k(\mathbf{z}_i^k,\mathbf{z}_j^k)}{d\,W_{nm}^k} &= -\left(z_{n,\,i}^k-z_{n,\,j}^k\right)\left(f'\left(\mathbf{W}^k\mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1}-f'\left(\mathbf{W}^k\mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}\right)\\ &\quad+\sum_{s=i,\,j}\left(1-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_s^k\right\|^2}\right)z_{n,\,s}^k f'\left(\mathbf{W}^k\mathbf{z}_s^{k-1}\right)_n z_{m,\,s}^{k-1}\,.\end{aligned} \tag{54}$$

Considering a linear network for simplicity, the weight update over two points i, j (negative of a single term in the sum in Eq. (40)) for the cosine similarity kernel becomes

$$\begin{aligned}\Delta W_{nm}^k &= M_{ij}^k\left(z_{n,\,i}^k-z_{n,\,j}^k\right)\left(z_{m,\,i}^{k-1}-z_{m,\,j}^{k-1}\right)-M_{ij}^k\sum_{s=i,\,j}\left(1-\frac{(\mathbf{z}_i^k)^\top\mathbf{z}_j^k}{\left\|\mathbf{z}_s^k\right\|^2}\right)z_{n,\,s}^k z_{m,\,s}^{k-1},\\ M_{ij}^k &= \frac{1}{\left\|\mathbf{z}_i^k\right\|\left\|\mathbf{z}_j^k\right\|}\left(2\,\mathring{k}(\mathbf{z}_i^k,\mathbf{z}_j^k)-\gamma\,\mathring{k}(\mathbf{y}_i,\mathbf{y}_j)\right).\end{aligned} \tag{55}$$

This rule is biologically implausible (or very hard to implement) for two reasons.

First, to compute M_{ij}^k the layer needs to track three signal simultaneously: $(\mathbf{z}_i^k)^\top\mathbf{z}_j^k$, $\left\|\mathbf{z}_i^k\right\|$ and $\left\|\mathbf{z}_j^k\right\|$ (cf. only one for the Gaussian kernel, $\left\|\mathbf{z}_i^k-\mathbf{z}_j^k\right\|$).

Second, assuming point i comes after j, the pre-factor of the Hebbian term (the sum in Eq. (55)) for $s=j$ can't be computed until i comes in. As a result, the update requires three independent plasticity pathways (one for $M_{ij}^k\left(z_{n,\,i}^k-z_{n,\,j}^k\right)\left(z_{m,\,i}^{k-1}-z_{m,\,j}^{k-1}\right)$ and two for the sum in Eq. (55)), because each term in Eq. (55) combines the pre- and post-synaptic activity on different timescales and with different third factors.

Adding grouping and divisive normalization to this rule is straightforward: we need to use the grouped response v_α^k (Eq. (18)) in the kernel as $k(\mathbf{z}_i^k,\mathbf{z}_j^k)=(\mathbf{v}_i^k)^\top\mathbf{v}_j^k/(\left\|\mathbf{v}_i^k\right\|\left\|\mathbf{v}_j^k\right\|)$, and repeat the calculation above (divisive normalization will appear here too as it results from differentiating v_α^k). 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 normalization, our objective resembles the "sim-bpf" loss in [6] (but it doesn't match it exactly, as we also introduce centering of the kernel and group convolutional layers over multiple channels rather than one).

B.5 Linear kernel

Derivation of the update for the linear kernel only needs Eq. (50), therefore by doing the same calculations we obtain

$$\begin{aligned}\frac{d\,k(\mathbf{z}_i^k, \mathbf{z}_j^k)}{d\,W_{nm}^k} &= \frac{d\,(\mathbf{z}_i^k)^\top \mathbf{z}_j^k}{d\,W_{nm}^k} = \frac{d\,\left(z_{n,\,i}^k z_{n,\,j}^k\right)}{d\,W_{nm}^k} \\ &= -\left(z_{n,\,i}^k - z_{n,\,j}^k\right)\,\left(f'\left(\mathbf{W}^k \mathbf{z}_i^{k-1}\right)_n z_{m,\,i}^{k-1} - f'\left(\mathbf{W}^k \mathbf{z}_j^{k-1}\right)_n z_{m,\,j}^{k-1}\right) \\ &\quad + \sum_{s=i,\,j} z_{n,\,s}^k\, f'\left(\mathbf{W}^k \mathbf{z}_s^{k-1}\right)_n z_{m,\,s}^{k-1}.\end{aligned} \tag{56}$$

It 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 need the same third factor). However, we couldn't achieve good performance with this kernel.

C Circuitry to implement the third factor

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

C.1 Gaussian kernel

The update equation for the Gaussian kernel (repeating Eq. (12) but in time rather that indices ij; assuming a linear network as it doesn't affect the third factor) is

$$\Delta W_{nm,\,t}^k \propto M_t^k\,(z_{n,\,t}^k - z_{n,\,t-\Delta t}^k)(z_{m,\,t}^{k-1} - z_{m,\,t-\Delta t}^{k-1})\,, \tag{57}$$

$$M_t^k = -\frac{1}{\sigma^2}\left(\gamma\,\mathring{k}(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,\mathring{k}(\mathbf{z}_t^k, \mathbf{z}_{t-\Delta t}^k)\right)\,k(\mathbf{z}_t^k, \mathbf{z}_{t-\Delta t}^k)\,. \tag{58}$$

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 $k(\mathbf{z}_t^k, \mathbf{z}_{t-\Delta t}^k)$ (and then center everything). For the Gaussian kernel, we thus need $\left\|\Delta \mathbf{z}_t^k\right\|^2$, which is given by

$$b_{1,\,t}^k = \sum_n (\Delta z_{n,\,t}^k)^2\,, \tag{59}$$

so that $k(\mathbf{z}_t^k, \mathbf{z}_{t-\Delta t}^k) = \exp(-b_{1,\,t}^k/(2\sigma^2))$. That gives us the uncentered component of the third factor, denoted $b_{2,\,t}^k$,

$$b_{2,\,t}^k = \gamma\,k(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,k(\mathbf{z}_t^k, \mathbf{z}_{t-\Delta t}^k) = \gamma\,k(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\,\exp\left(-\frac{1}{2\sigma^2} b_{1,\,t}^k\right)\,. \tag{60}$$

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

$$b_{3,\,t}^k = \beta\,b_{2,\,t}^k + (1-\beta)\,b_{3,\,t}^k \tag{61}$$

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

$$M_t^k = -\frac{1}{\sigma^2}\left(b_{2,\,t}^k - b_{3,\,t}^k\right)\exp\left(-\frac{1}{2\sigma^2} b_{1,\,t}^k\right). \tag{62}$$

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

C.2 Gaussian kernel with grouping and divisive normalization

The update for the Gaussian kernel with grouping (repeating Eq. (22) but in time; assuming a linear network as it doesn't affect the third factor) is

$$\Delta W^k_{\alpha n m,\,t} \propto M^k_{\alpha,\,t} \left(r^k_{\alpha n,\,t} r^{k-1}_{m,\,t} - r^k_{\alpha n,\,t-\Delta t} r^{k-1}_{m,\,t-\Delta t} \right) , \tag{63a}$$

$$M^k_{\alpha,\,t} = M^k_t \left(v^k_{\alpha,\,t} - v^k_{\alpha,\,t-\Delta t} \right) , \tag{63b}$$

$$M^k_t = -\frac{1}{\sigma^2} \left(\gamma \mathring{k}(\mathbf{y}_t, \mathbf{y}_{t-\Delta t}) - 2\, \mathring{k}(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t}) \right) k(\mathbf{z}^k_t, \mathbf{z}^k_{t-\Delta t}) . \tag{63c}$$

The Hebbian term – the term in parentheses in Eq. (63a) – corresponds to the difference of pre- and post-synaptic product, 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. (14) or Eq. (16)), although this update rule requires a different interaction of the pre- and post-synaptic activity comparing to the plain Gaussian kernel (Eq. (57)).

The third factor is almost the same as for the plain Gaussian kernel, with two differences. First, we need to compute the centered normalization $v^k_{\alpha,\,t}$ (as in Eq. (18)) and its change over time for each group α:

$$\tilde{b}^k_{\alpha,\,t} = \Delta v^k_{\alpha,\,t} . \tag{64}$$

Second, M^k_t is computed for $\Delta v^k_{\alpha,\,t}$ rather than $\Delta z^k_{n,\,t}$, such that (cf. Eq. (59))

$$\tilde{b}^k_{1,\,t} = \sum_\alpha (\tilde{b}^k_{\alpha,\,t})^2 , \tag{65}$$

and $\tilde{b}^k_{2,\,t}$ (Eq. (60) but with $\tilde{b}^k_{1,\,t}$) and $\tilde{b}^k_{3,\,t}$ (Eq. (61) but with $\tilde{b}^k_{1,\,t}$ and $\tilde{b}^k_{2,\,t}$) stay the same.

The third factor becomes

$$M^k_{\alpha,\,t} = -\frac{1}{\sigma^2} \left(\tilde{b}^k_{2,\,t} - \tilde{b}^k_{3,\,t} \right) \exp\left(-\frac{1}{2\sigma^2} \tilde{b}^k_{1,\,t} \right) \tilde{b}^k_{\alpha,\,t} . \tag{66}$$

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

D Experimental details

D.1 Network architecture

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

Each layer had the following structure: linear/convolutional operation $\to$ batchnorm (if any) $\to$ nonlinearity $\to$ pooling (if any) $\to$ local loss computation (doesn't modify activity) $\to$ divisive normalization (if any) $\to$ 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.

D.2 Choice of kernels

The gradient over each batch was computed as in Eq. (40).

We used cosine similarity with centered labels (Eq. (35); as all datasets are balanced, we don't need to know the probability of a class to center). The kernels for $\mathbf{z}$ were plain Gaussian (Eq. (26)), Gaussian with grouping and divisive normalization ("grp+div"; Eq. (19) s.t. the next layer sees $r^k_{\alpha n} \equiv \mathring{z}^k_{\alpha n} / (u^k_\alpha)^p$) and grouping without divisive normalization ("grp"; also Eq. (19) but the next layer sees $z^k_{\alpha n}$), plain cosine similarity (Eq. (25)), and cosine similarity with grouping and with or without divisive normalization ($k(\mathbf{z}^k_i, \mathbf{z}^k_j) = (\mathbf{v}^k_i)^\top \mathbf{v}^k_j / (\left\|\mathbf{v}^k_i\right\| \left\|\mathbf{v}^k_j\right\|)$ with $\mathbf{v}$ as in Eq. (18)).

D.3 Pre-processing of datasets

MNIST, fashion-MNIST and Kuzushiji-MNIST images were centered by 0.5 and normalized by 0.5.

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

D.4 Shared hyperparameters for all experiments

We used the default parameters for AdamW, batchnorm, LReLU and SELU, and grouping without divisive normalization used $p = 0.5$ to be comparable with the objective in [6]. 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, for the final/backprop – 1e-6; the learning rates were multiplied by 0.25, with individual schedules described below. For SGD, the momentum was 0.95; for AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$1e-8. Batchnorm had momentum of 0.1 and $\epsilon = 1e-5$, with initial scale $\gamma = 1$ and shift $\beta = 0$. Leaky ReLU had the slope 0.01; $\mathrm{SELU}(x) = \mathrm{scale}(\max(0, x) + \min(0, \alpha(\exp(x) - 1)))$ had $\alpha \approx 1.6733$ and scale ≈ 1.0507 (precise values were found numerically in [26]; note that dropout for SELU was changed to alpha dropout, as proposed in [26]). Grouping with divisive normalization used $\delta = 1$, $p = 0.2$, but $\delta = 1$ and $p = 0.5$ without divisive normalization. Gaussian kernels used $\sigma = 5$. The balance parameter was set to $\gamma = 2$.

D.5 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 3; the final results are given in Table 4 (same as Table 1 but with "grp") and Table 5 (max - min accuracy).

D.6 Large network

The dropout for all experiments was 0.05, with LReLU for AdamW+batchnorm and SELU for SGD. The networks were trained for 100 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 6. The results are given in Table 7 (mean test accuracy) and Table 8 (max - min accuracy). We didn't find a successful set of parameters for the Gaussian kernel with grouping but without divisive normalization on SGD. Results on AdamW with batchnorm introduce a gap between performance of backprop and kernelized bottleneck, but the gradient updates in batch norm make the resulting learning rule implausible. Moreover, batchnorm is specific to backprop, and our rule can be improved in another (also implausible) way: [6] showed that passing the activity $\mathbf{z}^k$ through a convolutional layer before computing cosine similarity improves performance.

Table 3: Parameters for the 3-layer fully connected net (1024 neurons per layer). The range of rules includes backprop, SGD over the last layer, cosine similarity (cossim) and the Gaussian kernel (all either plain, or with grouping and divisive normalization, denoted "grp+div", or with grouping but without divisive normalization, denoted "grp").

	backprop		last layer		cossim			Gaussian		
		grp+div		grp+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 4: Mean test accuracy over 5 random seeds for a 3-layer fully connected net (1024 neurons per layer). The range of rules includes backprop, SGD over the last layer, cosine similarity (cossim) and the Gaussian kernel (all either plain, or with grouping and divisive normalization, denoted "grp+div", or with grouping but without divisive normalization, denoted "grp").

	backprop		last layer		cossim			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 5: Same as Table 4, but max minus min test accuracy over 5 random seeds.

	backprop		last layer		cossim			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 6: Parameters for the 1x and 2x wide networks. Same range of rules as Table 3, but without the plain version for the cossim or Gaussian kernels.

	backprop		cossim		Gaussian	
		grp+div	grp	grp+div	grp	grp+div
1x wide net						
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						
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

Table 7: Mean test accuracy on CIFAR10 over 5 random seeds for the 1x and 2x wide networks. Same range of rules as Table 4, but without the plain version for the cossim or Gaussian kernels.

	backprop		cossim		Gaussian	
		grp+div	grp	grp+div	grp	grp+div
1x wide net	91.0	91.0	88.8	89.8	<40	86.2
2x wide net	91.9	90.9	89.4	91.3	<40	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 8: Same as Table 7, but max minus min test accuracy over 5 random seeds.

	backprop		cossim		Gaussian	
		grp+div	grp	grp+div	grp	grp+div
1x wide net	0.4	0.4	0.7	0.7		0.9
2x wide net	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