Kernel similarity matching with Hebbian Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent works have derived neural networks with online correlation-based learning rules to perform *kernel similarity matching*. These works applied existing linear similarity matching algorithms to nonlinear features generated with random Fourier methods. In this paper attempt to perform kernel similarity matching by directly learning the nonlinear features. Our algorithm proceeds by deriving and then minimizing an upper bound for the sum of squared errors between output and input kernel similarities. The construction of our upper bound leads to online correlation-based learning rules which can be implemented with a 1 layer recurrent neural network. In addition to generating high-dimensional linearly separable representations, we show that our upper bound naturally yields representations which are sparse and selective for specific input patterns. We compare the approximation quality of our method to neural random Fourier method and variants of the popular but non-biological "Nyström" method for approximating the kernel matrix. Our method appears to be comparable or better than randomly sampled Nyström methods when the outputs are relatively low dimensional (although still potentially higher dimensional than the inputs) but less faithful when the outputs are very high dimensional.

1 Introduction

Brain inspired learning algorithms have a long history in the field of neural networks and machine learning [Rosenblatt, 1958, Olshausen and Field, 1996, Lee and Seung, Riesenhuber and Poggio, 1999, Hinton, 2007, Lillicrap et al., 2016]. While many algorithms have diverged from their biological roots, the motivation to study biology remains clear: the human brain is such a powerful learning agent, there must be insights to be gained by making our algorithms look "brain-like". This paper is focused on merging biological constraints with the well-established field of kernel-based machine learning.

A common assumption in brain-inspired models of learning is that synaptic update rules should be a) online, meaning the algorithm only has access to a single input pattern at a time and b) local, meaning synapses should only be modified using information immediately available to the synapse, often just the pre- and post-firing rates of the neurons to which it is connected. Learning rules with these properties are commonly referred to as Hebbian learning rules [Chklovskii, 2016].

Recent works have devised neural networks with Hebbian learning rules that perform linear similarity matching. These networks map every input $\mathbf{x}_t$ to a representation $\mathbf{y}_t$ such that linear output similarities match linear input similarities $\mathbf{y}_s \cdot \mathbf{y}_t \approx \mathbf{x}_s \cdot \mathbf{x}_t$. These networks are interesting as models for real brains because they display a number of interesting biological properties: they are recurrent networks with correlation-based learning rules [Pehlevan et al., 2018] and can be modified to include non-negativity [Pehlevan and Chklovskii, 2014], sparsity, and convolutional structure [Obeid et al., 2019].

However there is a problem if one believes these networks should ultimately generate representations which are useful for downstream tasks. If similarities are actually matched, that is if $\mathbf{y}_s \cdot \mathbf{y}_t = \mathbf{x}_s \cdot \mathbf{x}_t$,

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

then the outputs are simply an orthogonal transformation of the inputs, $\mathbf{y}_t = \mathbf{Q}\mathbf{x}_t$, which is unlikely to have significant impact on downstream tasks. Bahroun et al. [2017] identified this problem and proposed a solution: instead of matching linear input similarities, one can match nonlinear input similarities: $\mathbf{y}_s \cdot \mathbf{y}_t \approx f(\mathbf{x}_s, \mathbf{x}_t)$. The authors provided a method that can be applied to any shift-invariant kernel. They applied the random Fourier feature method of Rahimi et al. [2007] to map inputs to nonlinear feature vectors $\mathbf{x} \to \boldsymbol{\psi}$ and then applied the linear similarity matching framework of Pehlevan et al. [2018] to these nonlinear features.

In this paper, tackle the same neural kernel similarity matching problem with a different approach. Instead of using random nonlinear features, we directly optimize for the features with Hebbian learning rules that resemble the learning rules derived in the original works on linear similarity matching. To derive our learning rules, we show that for any kernel we can upper bound the sum of squared errors $|\mathbf{y}_s \cdot \mathbf{y}_t - f(\mathbf{x}_s, \mathbf{x}_t)|^2$ with a correlation-based energy. Gradient-based optimization of our upper bound with will lead to a neural network with correlation-based learning rules.

2 Correlation-based bound for kernel similarity matching

Roadmap for this section We first define the kernel similarity matching problem (Eq. 1). We then derive a correlation-based optimization (Eq. 6) which is an upper bound for to Eq. 1 (up to a constant that does not depend on the representations). We then use a Legendre transform to derive an equivalent (except for the numerical stability parameter λ) optimization problem in Eq. 9 that will lend itself towards online updates.

Kernel similarity matching Assume we are given a set of input vectors $\{\mathbf{x}^t \in \mathbb{R}^M\}_{t=1}^T$ and a positive semi-definite kernel function $f : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ which defines the similarity between input vectors. The goal is to find a corresponding set of representations $\{\mathbf{y}^t \in \mathbb{R}^N\}_{t=1}^T$ such that for all pairs (s, t) we have $\mathbf{y}^s \cdot \mathbf{y}^t \approx f(\mathbf{x}^s, \mathbf{x}^t)$. We will assume that $T \gg N > M$: the dimensionalities of $\mathbf{x}, \mathbf{y}$ are much lower than the number of samples T, but $\mathbf{y}$ are still higher dimensional than the inputs. This is formalized by minimizing the sum of squared errors:

$$\min_{\{\mathbf{y}^t\}} \frac{1}{T^2} \sum_{s,t}^{T} \left[f(\mathbf{x}^s, \mathbf{x}^t) - \mathbf{y}^s \cdot \mathbf{y}^t \right]^2 \tag{1}$$

This is known as the classical multidimensional scaling objective [Borg and Groenen, 2005]. For arbitrary nonlinearity f this can be solved exactly by finding the top N eigenvectors of the $T \times T$ input similarity matrix [Borg and Groenen, 2005], and is therefore closely related to kernel PCA [Schölkopf et al., 1997]. However, this requires computing and storing similarities for all pairs of input vectors which breaks the online constraint that we require for biological realism. The purpose of this paper is to find an online algorithm, with correlation-based computations, that can at least approximately minimize Eq. 1.

Correlation based upper bound In this section we provide an upper bound to Eq. 1 which does not require computing $f(\mathbf{x}_s, \mathbf{x}_t)$ for any (s, t). The first step is to expand the square in Eq (1) to yield:

$$\frac{1}{T^2} \sum_{s,t}^{T} \left[f(\mathbf{x}^s, \mathbf{x}^t) - \mathbf{y}^s \cdot \mathbf{y}^t \right]^2 = -\frac{2}{T^2} \sum_{s,t} f(\mathbf{x}^s, \mathbf{x}^t) \mathbf{y}^s \cdot \mathbf{y}^t + \frac{1}{T^2} \sum_{s,t} (\mathbf{y}^s \cdot \mathbf{y}^t)^2 + \text{const} \tag{2}$$

We will now show how to bound the first term on the right hand side.

Theorem 1. *If f is a positive semidefinite kernel function, then*

$$\frac{1}{2T^2} \sum_{s,t} y^s y^t f(\mathbf{x}^s, \mathbf{x}^t) \geq \frac{1}{T} \sum_{t} q y^t f(\mathbf{x}^t, \mathbf{w}) - \frac{1}{2} q^2 f(\mathbf{w}, \mathbf{w}) \tag{3}$$

for all q and $\mathbf{w}$.

Proof. Because f is a positive semi-definite kernel, we can assign to any set of M-dimensional vectors $\{\mathbf{w}, \mathbf{x}_1, \ldots, \mathbf{x}_T\}$, a corresponding set of (at most) $T + 1$-dimensional vectors $\{\boldsymbol{\phi}_\mathbf{w}, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_T\}$ whose inner products yield the similarity defined by f:

$$\boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{t'} = f(\mathbf{x}_t, \mathbf{x}_{t'}) \qquad \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{x}_t, \mathbf{w}) \qquad \boldsymbol{\phi}_\mathbf{w} \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{w}, \mathbf{w}) \tag{4}$$

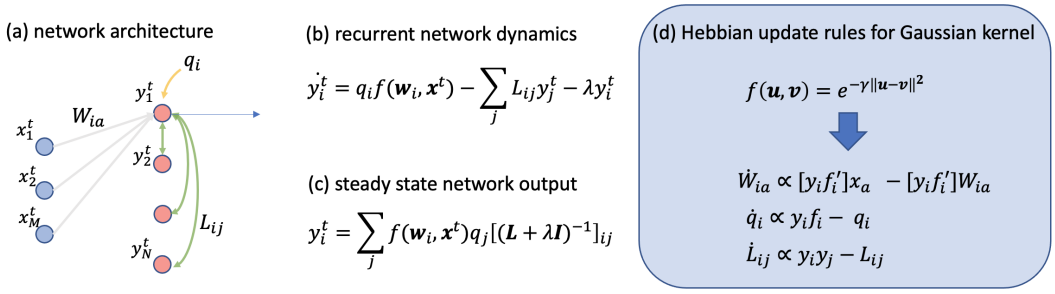

Figure 1: Neural network implementation of the optimization in Eq. 9 (a) network architecture (b) recurrent network dynamics (c) steady state network response (d) Hebbian update rules for the special case of Gaussian kernels (the precise form of these updates will be depend on the kernel)

Now consider the vector difference $\frac{1}{T}\sum_t y_t \boldsymbol{\phi}_t - q\boldsymbol{\phi}_w$. The squared norm of this difference is of course non-negative. Additionally we can expand out this square:

$$0 \leq \frac{1}{2}\left\| \frac{1}{T}\sum_t y_t \boldsymbol{\phi}_t - q\boldsymbol{\phi}_{\mathbf{w}} \right\|^2 = \frac{1}{2T^2}\sum_{s,t} y_s y_t \boldsymbol{\phi}_s \cdot \boldsymbol{\phi}_t - \frac{1}{T}\sum_t q y_t \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{\mathbf{w}} + \frac{1}{2} q^2 \boldsymbol{\phi}_{\mathbf{w}} \cdot \boldsymbol{\phi}_{\mathbf{w}} \quad (5)$$

At this point we can simply replace all dot products with the equivalent nonlinear similarities $f(\cdot,\cdot)$ in Equation 4 and rearrange the terms to yield our key inequality (Eq. 3). □

Our inequality still holds if we maximize the right hand side with respect to q and $\mathbf{w}$. For every index i of $\mathbf{y}$, we find the optimal $\mathbf{w}_i, q_i$, and then replace the first pairwise sum in Eq. (2) with our upper bounds. Additionally we rearrange the order of the summations in second term on the right hand side of Eq. (2) to yield the following upper bound for the y-dependent terms in Eq. (2):

$$\min_{\mathbf{y}^t} \min_{q_i, \mathbf{w}_i} -\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\left[q_i y_i^t f(\mathbf{w}_i, \mathbf{x}^t) - \frac{1}{2} q_i^2 f(\mathbf{w}_i, \mathbf{w}_i)\right] + \frac{1}{4}\sum_{i,j=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T} y_i^t y_j^t\right)^2 \quad (6)$$

Online focused reformulation We can further remove the square of the correlation matrix $\frac{1}{T}\sum_t y_i^t y_j^t$ (another impediment to online learning) by introducing a Legendre transformation: $\frac{1}{2}C_{ij}^2 \to \max_{L_{ij}} C_{ij}L_{ij} - \frac{1}{2}L_{ij}^2$:

$$\min_{\mathbf{W},\mathbf{Y},\mathbf{q}} \max_{\mathbf{L}} \frac{1}{T}\sum_{t=1}^{T}\left[-\sum_{i=1}^{N}\left[q_i y_i^t f(\mathbf{w}_i, \mathbf{x}^t) - \frac{1}{2} q_i^2 f(\mathbf{w}_i, \mathbf{w}_i)\right] + \frac{1}{2}\sum_{i,j=1}^{N}\left[L_{ij} y_i^t y_j^t - \frac{1}{2}L_{ij}^2\right]\right] \quad (7)$$

We can swap the order of the y and L optimizations, because the objective obeys the strong min-max property with $\mathbf{W}, \mathbf{q}$ fixed (Appendix Section A of Pehlevan et al. [2018]). We add one final term $\frac{\lambda}{NT}\sum_{t=1}^{T}\sum_{i=1}^{N}(y_i^t)^2$ to the objective, which can be important for numerical stability of our resulting algorithm. In our experiments $\lambda = 0.001$. Finally, to better motivate our online algorithm, we define the "per-sample-energy":

$$e^t := \sum_{i=1}^{N} -\left[q_i y_i^t f(\mathbf{w}_i, \mathbf{x}^t) - \frac{1}{2} q_i^2 f(\mathbf{w}_i, \mathbf{w}_i)\right] + \frac{1}{2}\sum_{i,j=1}^{N}\left[L_{ij} y_i^t y_j^t - \frac{1}{2}L_{ij}^2\right] + \frac{\lambda}{2}\sum_{i=1}^{N}(y_i^t)^2 \quad (8)$$

where $e^t := e(\mathbf{y}^t, \mathbf{x}^t; \mathbf{W}, \mathbf{q}, \mathbf{L})$. The final optimization we will perform, which is equivalent to the optimization in Eq. 6, and is derived as an upper bound to Eq. 1, is thus:

$$\min_{\mathbf{W},\mathbf{q}} \max_{\mathbf{L}} \min_{\mathbf{Y}} \frac{1}{T}\sum_{t=1}^{T} e(\mathbf{y}^t, \mathbf{x}^t; \mathbf{W}, \mathbf{q}, \mathbf{L}) \quad (9)$$

3 Neural network optimization

Applying a stochastic gradient descent-ascent algorithm to Eq. (9) yields a neural network (Fig. 1) in which y_i^t is the response of neuron i to input pattern t, $\mathbf{w}_i$ is the vector of incoming connections to

neuron i from the input layer, q_i is a term which modulates the strength of these incoming connections, and L_{ij} is a matrix of lateral recurrent connections between outputs.

Specifically the neural algorithm procedes as follows. We initialize $W_{ia} \leftarrow \mathcal{N}(0,1)$, $q_i \leftarrow 1$ and $L_{ij} \leftarrow \mathbf{I}_{ij}$. At each iteration sample a minibatch of inputs $\{\mathbf{x}^b\}$. Using Eq.12 we compute the $\{\mathbf{y}^b\}$ which minimize Eq. 9 for fixed synapses. Using these optimal $\{\mathbf{y}^b\}$, we compute the minibatch energy $e = \frac{1}{B}\sum_b e(\mathbf{x}^b, \mathbf{y}^b; \mathbf{W}, \mathbf{q}, \mathbf{L})$ and take a gradient descent step for $\mathbf{q}$, a rescaled gradient descent step for $\mathbf{w}$ and a gradient ascent step for $\mathbf{L}$:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \frac{\eta_w}{q_i^2}\frac{\partial e}{\partial \mathbf{w}_i} \qquad q_i \leftarrow q_i - \eta_q \frac{\partial e}{\partial q_i} \qquad L_{ij} \leftarrow L_{ij} + \eta_l \frac{\partial e}{\partial L_{ij}} \tag{10}$$

Convergence of the neural algorithm We treat convergence of this gradient descent ascent algorithm as an empirical issue. We adopt the "two time scale" strategy that has shown empirical successes for training generative adversarial networks [Heusel et al., 2017]. We choose the learning rates such that $\eta_q, \eta_w \ll \eta_l$. Intuitively when choosing η_l to be large, the L_{ij} can approximately maximize Eq. 9 for any particular $\mathbf{q}, \mathbf{W}$ so that the min-max ordering is roughly preserved. In practice this ratio is important for convergence. We do not observe convergence when the ratios η_w/η_l or η_q/η_l are large. Unfortunately it is an empirical question of what is "too large". If we could show that the objective were concave in L, it can be gradient descent ascent with smaller learning rates for W, q would indeed converge to a saddle point [Lin et al., 2020, Seung, 2019]. However, this question will have to be left for future work.

Empirically it is sometimes observed that q_i quickly shrinks to a small value early in training, which subsequently leads to small gradients for $\mathbf{w}$. The rescaling of the $\mathbf{w}_i$ updates provides an adaptive learning rate that appeared to improve training times in practice. We have attached the main portion of the training code, written using PyTorch, in the appendix.

3.1 Network dynamics

Assuming fixed parameters $\mathbf{q}, \mathbf{W}, \mathbf{L}$, the gradient for $\mathbf{y}$ can be computed for any input pattern $\mathbf{x}$. Gradient descent can be used to perform the inner loop minimization in Equation 9:

$$\dot{y}_i = \eta_y \left[q_i f(\mathbf{w}_i, \mathbf{x}) - \sum_{j=1}^{N} L_{ij} y_j - \lambda y_i \right] \tag{11}$$

Like previous works on linear similarity matching, these dynamics can be interpreted as the dynamics of a one-layer recurrent neural network with all-to-all inhibition $\sum_{j=1}^{N} L_{ij} y_j$ between units. A diagram of this network is shown in Figure 1. Note that we can analytically perform the inner loop minimization with a non-neural algorithm:

$$y_i \leftarrow \sum_j [\mathbf{L} + \lambda \mathbf{I}]^{-1}_{ij} q_j f(\mathbf{w}_j, \mathbf{x}) \tag{12}$$

This is useful both conceptually and for speeding up the training process in our experiments. This formula shows us that $\mathbf{y}$ is a linear function of the non-linear feedforward input $f(\mathbf{w}_i, \mathbf{x}_t)$. This is different from Seung and Zung [2017], Pehlevan and Chklovskii [2014] where the the neurons are non-linear functions (due to non-negativity constraints) of linear feedforward input $\mathbf{w}_i \cdot \mathbf{x}_t$.

3.2 Synaptic learning rules: arbitrary kernel

In the previous section we saw how the $\mathbf{W}$ could be interpreted as feedforward synapses, $\mathbf{q}$ as feedforward regulatory terms, $\mathbf{L}$ as inhibitory synapses. Gradient descent on $\mathbf{W}, \mathbf{q}$ and gradient ascent on $\mathbf{L}$ provides an algorithm for performing the optimization in Equation 9. At each step, we compute the optimal $\mathbf{y}$. For simplicity, we consider the case with a single input, in which case we drop the index b on $\mathbf{x}^b, \mathbf{y}^b$. The stochastic gradients for $\mathbf{W}$ lead to the update:

$$\Delta \mathbf{w}_i \propto y_i \partial f(\mathbf{w}_i, \mathbf{x})/\partial \mathbf{w}_i - q_i \partial f(\mathbf{w}_i, \mathbf{w}_i)/\partial \mathbf{w}_i \tag{13}$$

Classically Hebbian rules have been defined so that the update is linear in the input $\mathbf{x}$ (although they can be nonlinear in the output $\mathbf{y}$ which is a function of $\mathbf{x}$) (Eq. 1 of Brito and Gerstner [2016]). This

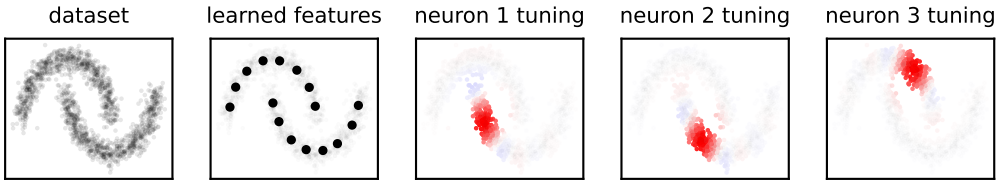

Figure 2: Overview of our Hebbian radial basis function network on the half moons dataset (a) dataset, (b) features $\{\mathbf{w}_i\}_{i=1}^{16}$ (c,d,e) response profiles of 3 neurons.

rule is more general as it is a nonlinear function of the input vector $\partial f(\mathbf{w}_i, \mathbf{x})/\partial \mathbf{w}_i = h(\mathbf{x}, \mathbf{w}_i)$. However we note that the spirit of Hebb is still here as this is an online, local, correlation-based learning rule.

The regulatory terms (essentially controlling the magnitude of the strength of feedforward input) can be updated with:

$$\Delta q_i \propto y_i f(\mathbf{w}_i, \mathbf{x}) - f(\mathbf{w}_i, \mathbf{w}_i) q_i \tag{14}$$

Here we have the correlation between the feedforward input and the neurons response. Finally gradients for the inhibitory synapses are:

$$\Delta L_{ij} \propto y_i y_j - L_{ij} \tag{15}$$

This is exactly the same "anti-hebbian" update seen in previous linear similarity matching works Pehlevan et al. [2018]. The inhibition grows in strength as the correlation between neurons grows.

3.3 Synaptic learning rules: radial basis function kernel

Before moving on, we'll consider the form of the update rules in Eq. 13,14 when the kernel is a radial basis function, i.e. when the kernel is a function of the Euclidean distance. For simplicity we'll also assume the kernel is normalized so that $f(\mathbf{v}, \mathbf{v}) = 1$:

$$f(\mathbf{u}, \mathbf{v}) := g(\|\mathbf{u} - \mathbf{v}\|) \quad \text{and} \quad g(0) = 1 \tag{16}$$

In this case we get the gradient updates for $\mathbf{w}_i, q_i$:

$$\Delta \mathbf{w}_i \propto [y_i g_i'] \mathbf{x} - [y_i g_i'] \mathbf{w} \quad \Delta q_i \propto y_i g_i - q_i \tag{17}$$

The update for $\mathbf{w}_i$ is proportional to the input $\mathbf{x}$, but modulated by the output response (y_i) and a function of the feedforward input (g_i'). The updates for L_{ij} do not depend on the form of the kernel.

4 Experiments

We train networks using a Gaussian kernel for the half moons dataset and a "power-cosine" kernel (defined in section 4.2) for the MNIST dataset. We compare the approximation error (Eq. 1) of our method to the approximation error given by a) the optimal eigenvector-based solution (which we label as kernel PCA) b) Nyström approximation with uniformly sampled landmarks c) Nyström approximation using KMeans to generate the landmarks d) Nyström approximation using our generated features (w_i) as the landmarks and e) random Fourier feature method (This method is not applicable to the cosine-based kernel we use for the MNIST dataset). The "dimensionality" refers to the number of components for the PCA method, the number of landmarks for the Nyström methods, and the number of Fourier features for the Fourier method. See the appendix for more details regarding each of these 5 methods. Method (e) is the only other explicitly neural method.

4.1 Half Moons Dataset

We train our algorithm on a simple half moons dataset (which can be generated with Pedregosa et al. [2011]), shown in Figure 2. It consists of 1600 input vectors $\mathbf{x} = [x_1, x_2]$ drawn from a distribution of two noisy interleaving half circles. We use a Gaussian kernel with $\sigma = 0.3$ to measure input

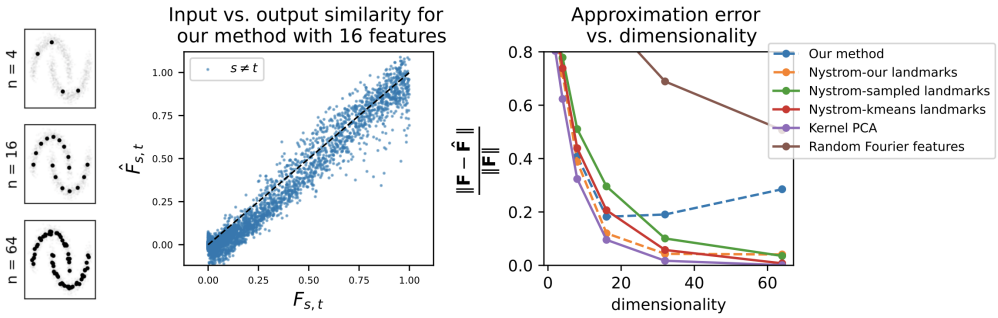

Figure 3: Approximation error vs. dimensionality for the half moons dataset (a) learned features for $n = 4, 16, 64$ (b) input-output similarities for 16 dimensional networks (c) normalized root-mean-square error between true kernel matrix and various approximation methods. The neural method (dashed blue) that we derive in Section 3 performs well for $n <= 16$, but the approximation actually gets worse as we increase the dimensionality.

similarities: $f(\mathbf{u}, \mathbf{v}) = e^{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}}$. We compare various number of neurons $n \in \{2, 4, 8, 16, 32, 64\}$. See the appendix for training details.

Emergence of sparse, template-tuned neurons In Fig. 2 we show the learned features $\{\mathbf{w}_i\}$ when we train our algorithm with 16 neurons. We observe that the features appear to tile the input space. We also show the tuning properties of 3 of the output neurons over the dataset. To generate these figures, we color code each sample in the dataset with the response of neuron y_i. Gray indicates zero response, red indicates a positive response and blue indicates a negative response. We observe that neurons appear to respond with large positive values centered around a small localized region of the input dataset. The features closely resemble the cluster centers return by KMeans.

Kernel approximation error In panel (a) of Fig. 3 we show the learned features for $n = \{4, 16, 64\}$. In panel (b) we plot the input similarities vs. output similarities generated by our neural algorithm with 16 dimensional outputs. In panel (c), we plot the normalized mean squared error for our method compared to the neural random Fourier method of Bahroun et al. [2017], non-neural Nyström methods, and non-neural but optimal kernel PCA method.

We observe that for small dimensionality ($n \leq 16$) our method actually seems to marginally outperform the Nyström+KMeans method, which outperforms the Nyström+randomly sampled landmarks method. Additionally, using the Nyström approximation with our features seems to be uniformly better than the representations we generate with the neural net. Essentially, our algorithm learns useful landmarks, but for most faithful representation, it is better to just throw away the neural responses and simply use the Nyström approximation with our landmarks. It is worth mentioning that as you increase the dimensionality higher, the Random Fourier method ultimately does converge to zero error, unlike our method.

Utility of representations evaluated by KMeans clustering In Fig. 4 we visualize the principle components of the inputs $\mathbf{x}$ and 16D representations $\mathbf{y}$. Of course, the principle components of $\mathbf{x}$ are not too interesting, they are just a reflected version of the original 2D dataset. The top two components of $\mathbf{y}$ appear to be more linearly separable than the inputs and indicate that a strong nonlinear transformation has occured. Additionally, we run KMeans on $\mathbf{x}$ and on $\mathbf{y}$ (we use the implementation of scikit-learn, and take the lowest energy solution using 100 inits). We observe that the clustering yields the nearly perfect labels when performed on $\mathbf{y}$. The kernel similarity matching vectors appear to be better suited for downstream learning tasks than the original inputs.

4.2 MNIST Dataset

We train our algorithm on the MNIST handwritten digits dataset LeCun [1998]. The dataset consists of 70,000 images of 28x28 handwritten digits, which we cropped by 4 pixels on each side to yield 20x20 images (which become 400 dimensional inputs). We use kernels of the form $f(\mathbf{u}, \mathbf{v}) = \|\mathbf{u}\|\|\mathbf{v}\|(\hat{\mathbf{u}} \cdot \hat{\mathbf{v}})^\alpha$ and varying number of neurons. Training details are provided in the appendix.

The linear kernel is recovered by setting $\alpha = 1$. We are not aware of other works using this exact "power-cosine" kernel before, however it is motivated by the *arccosine kernel* studied in the context of

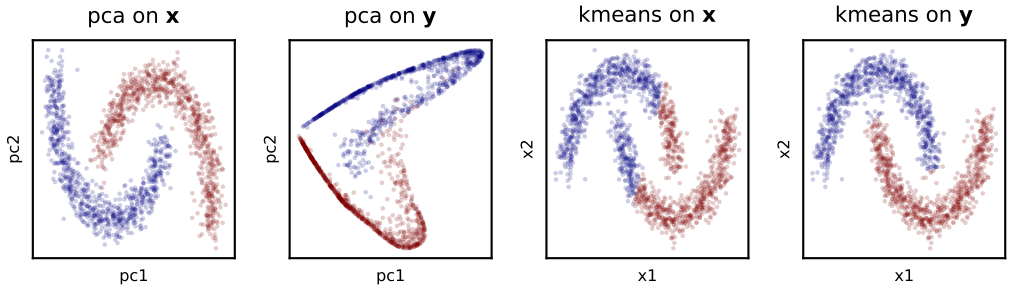

Figure 4: Utility of kernel similarity matching for downstream tasks. (a) principle components of the input vectors $\mathbf{x}$ (b) principle components of the 16 dimensional neural representations $\mathbf{y}$ (c) clustering generated by kmeans on $\mathbf{x}$ (d) clustering generated by kmeans on $\mathbf{y}$. For (a,b) the the colors are given by ground truth labels while in (c,d) the colors are given by the KMeans clustering.

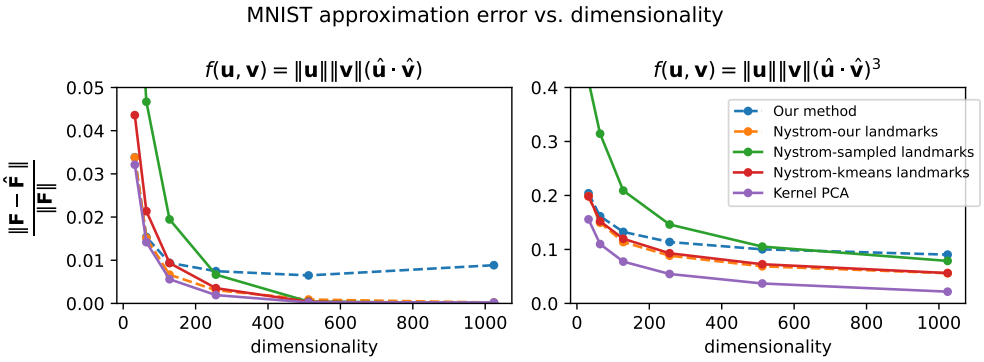

Figure 5: Approximation error vs. dimensionality for the MNIST dataset. (a) $f(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$ (linear kernel) (b) $f(\mathbf{u}, \mathbf{v}) = \|\mathbf{u}\|\|\mathbf{v}\|(\hat{\mathbf{u}} \cdot \hat{\mathbf{v}})^3$ (a nonlinear kernel). For the linear kernel all methods give relatively small approximation error once $n > 100$. Although yet again we see that the neural method does not continue to decrease as the dimensionality increases beyond 200, even in the linear setting.

wide random ReLU networks [Cho and Saul, 2009]. An important property of our kernel network is the linear input-output scaling, meaning that rescaling an input $\mathbf{x}' \leftarrow a\mathbf{x}$ will cause the corresponding representation to also be rescaled by the same factor $\mathbf{y}' \leftarrow a\mathbf{y}$. This will allow our nonlinear networks to have the same "contrast-invariant-tuning" properties that are thought to be displayed by simple cells in cat visual cortex [Skottun et al., 1987].

Approximation error We display the normalized approximation errors for $\alpha = 1$ and $\alpha = 3$ in Fig. 5. For the linear kernel ($\alpha = 1$) all methods yield a relatively small error even for low dimensionality. An error of 0.01 is hard to see by eye when plotting input-output similarity scatter plots as done in Fig. 3. For both $\alpha = 1, 3$ we observe again a strange behavior of our method: it seems to "bottom out" and the error stops decreasing and even begins to increase as the dimensionality increases. This may be related to unstable convergence properties of gradient descent ascent.

For $\alpha = 3$ we observe that the kernel PCA method largely outperforms all methods. We observe that Nyström with either our features or KMeans appears to outperform sampled Nyström methods. The sampled Nyström method is worse than our representations for low dimensionality but eventually catches up and surpasses ours neural representations.

Emergence of sparse representations We train networks with $\alpha = \{1, 2, 3, 4\}$ and $n = 800$ neurons (so the output dimensionality is exactly 2x the input dimensionality). There is a sign degeneracy when α is odd: we can multiply both $\mathbf{w}_i$ and y_i by -1 and the objective is unchanged. When we look at the response histogram for single neurons, we observe that for $\alpha = 3$, the response tends to be heavily skewed so that when the response has a high magnitude, it is either always positive or always negative. We remove this degeneracy by multiplying both $\mathbf{w}_i$ and y_i by the sign of $\langle y_i \rangle$. After removing this degeneracy we plot the neuron responses over all patterns in Figure 6.

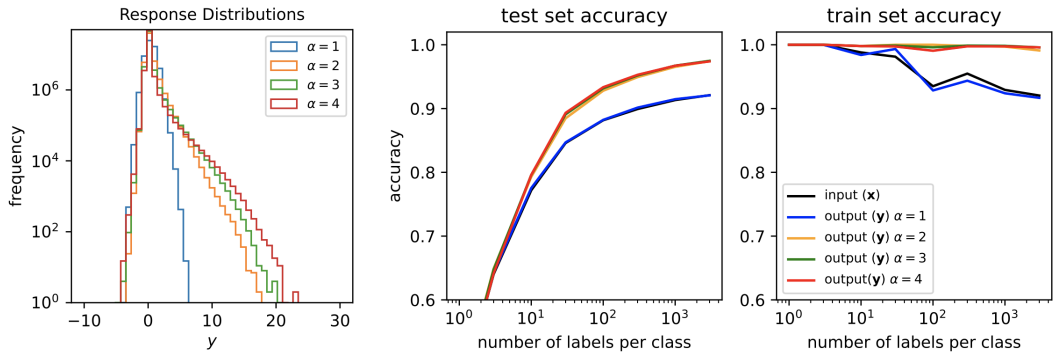

Figure 6: (a) response distribution (after the procedure we describe in the text for removing the sign degeneracy). The nonlinear kernels ($\alpha = 2, 3, 4$) naturally give rise to sparse distributions. (b) test set accuracy of a linear classifier classification for MNIST (c) train set accuracy of the corresponding linear classifier. Interestingly all nonlinear kernels give nearly identical train and test set results. The linear kernel gives nearly identical results to simply training the classifier directly on the pixels.

For $\alpha = 1$ (linear neurons), neuron responses are roughly centered around zero: neuron responses are neither sparse not skewed. For $\alpha = 2, 3, 4$, neurons appear to have a heavy tailed distribution, they frequently have small responses, but occasionally have large positive responses. Neurons become increasingly sparse and heavy tailed as we increase α, although this effect is not that strong.

Evaluating the representations via linear classification We train a linear classifier on the inputs $(\mathbf{x})$ and the outputs $(\mathbf{y})$ for $\alpha = 1, 2, 3, 4$ and $n = 800$. We train every configuration using $k \in \{1, 3, 10, 30, 100, 300, 1000, 3000\}$ labels per class. We train all configuration with a weight decay parameter $\lambda \in \{1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1\}$ which yields the highest test accuracy. We average the accuracy for every configuration over 5 random seeds. The results are show in Fig. 6.

As expected, the performance of the inputs and $\alpha = 1$ (linear similarity matching) is nearly identical on both test and train sets. Surprisingly, the test performance of $\alpha = 2, 3, 4$ is nearly identical. Perhaps these curves can be partially explained by the spectra of the output similarity matrix which we show in Figure 10 of the Appendix. While the shapes of the spectra are different in every case, $\alpha = 1$ has roughly 200 nonzero eigenvalues while $\alpha = 2, 3, 4$ all have nearly 800 nonzero eigenvalues. Perhaps the number of nonzero eigenvalues is more influential for the linear classification performance than the detailed shapes of these spectra.

5 Related work

Kernel similarity matching with random Fourier features The most closely related work to ours is kernel similarity matching with random Fourier features [Bahroun et al., 2017]. The key difference between our methods is that instead of learning the features $\mathbf{w}$, they use random Fourier features to directly generate nonlinear feature vectors $\boldsymbol{\phi}^t = \sqrt{2/n}\cos(\mathbf{W}\mathbf{x}^t + \mathbf{b})$ which they then feed into a standard linear similarity matching network. This leads them to a different architecture (one feedforward layer + one recurrent layer, instead of our single recurrent layer net) and a different set of learning rules. A benefit of the random feature approach is that it will theoretically lead to perfect matching, so long as the number of random features is sufficiently large.

However, the feature learning aspect of our algorithm naturally led to a sparse set of responses which lends our model an added degree of biological plausibility. Additionally, our method generalizes to non-shift invariant kernels and empirically it seemed that to yield better approximation error when the dimensionality of the output is not too high. Our method can be seen as a biased method for approximation, which can be useful when the dimensionality is low, but ultimately will underperform compared to non-biased methods such as random Fourier methods or Nyström methods.

Nyström Approximation While not obviously biological, Nyström methods are perhaps the most commonly used methods for approximating kernel matrices. The Nyström approximation uses a set of landmarks $\{\mathbf{w}_i : i = 1, 2, \ldots, N\}$ to construct a low rank approximation of the original kernel matrix [Williams and Seeger, 2001]. To more clearly see the relationship between this method to

ours, one can slightly modify the original formulation to generate "Nyström features":

$$\text{"Nyström features"} \;\; y_j^t = \sum_i f(\mathbf{x}^t, \mathbf{w}_i) M_{ij} \text{ where } M_{ij} = [(\mathbf{B}^\dagger)^{1/2}]_{ij} \text{ and } B_{ij} = f(\mathbf{w}_i, \mathbf{w}_j) \tag{18}$$

$\mathbf{B}^\dagger$ indicates the psuedo-inverse. Multiplying two such vectors togethers yields the Nyström approximation $\hat{F}_{st} = \mathbf{y}^s \cdot \mathbf{y}^t = \sum_{ij} f(\mathbf{x}^s, \mathbf{w}_i)[\mathbf{B}^\dagger]_{ij} f(\mathbf{x}^t, \mathbf{w}_j)$. Our method produces representations of the same functional form but our M matrix is derived from parameters learned by the correlations:

$$\text{Our features} \;\; y_j^t = \sum_i f(\mathbf{x}^t, \mathbf{w}_i) M_{ij} \text{ where } M_{ij} = [\mathbf{L} + \lambda \mathbf{I}]_{ij}^{-1} q_j \tag{19}$$

As measured by squared error, the Nyström approximation was actually a better approximation than our representations, when we used the same set of landmarks (Figs. 3, 5). The variation in Nyström methods primarily come from the method used to generate the landmarks. Two broad categories of landmark selection can be defined: template vs. pseudo-landmark. Template based approaches choose landmarks as a subset of the inputs $\mathbf{w} \in \{\mathbf{x}_1, \mathbf{x}_w, \ldots, \mathbf{x}_T\}$ typically chosen via sampling schemes [Williams and Seeger, 2001, Drineas et al., 2005, Musco and Musco, 2016]. Pseudo-landmark approaches do not require the landmarks to be inputs. Zhang et al. [2008] used the cluster centers generated by KMeans as the landmarks. Fu [2014] formulate landmark selection as an optimization problem in the reproducing Hilbert space. Our method can be seen as a pseudo-template approach as our landmarks are directly generated via Hebbian learning rules and in general will not be exactly equal to any particular input pattern. Our method is similar in spirit to the approach of Fu [2014]. A key difference is that we use a different objective, a correlation-based upper bound to the sum of squared errors, which gives rise to correlation-based learning rules.

6 Discussion

We have extended the neural random Fourier feature method of Bahroun et al. [2017] for kernel similarity matching to instead be applicable to arbitrary differentiable kernels. Rather than using random nonlinear features, we learned the features with Hebbian learning rules. Both this work and that of Bahroun et al. [2017] can be seen as extensions of the linear similarity matching works written in Hu et al. [2014], Pehlevan and Chklovskii [2014], Pehlevan et al. [2015, 2018]. By using a nonlinear input similarity, the representations learned by our network are capable of learning high-dimensional nonlinear functions of the input, without requiring any constraints such as non-negativity.

To our knowledge this is the first work that attempts to directly optimize the sum of squared errors (Eq. 1) without relying on sampling schemes or direct computation of the input similarity matrix. It would be interesting to relax the correlation-based constraint we have imposed on ourselves. This might allow for a variety of different types of bounds (Eq. 3) to be derived which in turn could lead to more faithful approximations than the one presented in our paper.

A key obstacle faced by users of this algorithm is the stochastic gradient descent ascent procedure. Empirically the convergence our algorithm is quite sensitive to the learning rate choices. This method does not provide the same sorts of theoretically guarantees or empirically observed robustness of sampling based methods. Generation of more robust ascent-descent optimization methods could be useful for making this class of algorithms more accessible for the practitioner.

References

Yanis Bahroun, Eugénie Hunsicker, and Andrea Soltoggio. Neural networks for efficient nonlinear online clustering. In *International Conference on Neural Information Processing*, pages 316–324. Springer, 2017.

I. Borg and P. J. F. Groenen. *Classical Scaling*, pages 261–267. Springer New York, New York, NY, 2005. ISBN 978-0-387-28981-6. doi: 10.1007/0-387-28981-X_12. URL `https://doi.org/10.1007/0-387-28981-X_12`.

Carlos SN Brito and Wulfram Gerstner. Nonlinear hebbian learning as a unifying principle in receptive field formation. *PLoS computational biology*, 12(9):e1005070, 2016.

Dmitri Chklovskii. The search for biologically plausible neural computation: The conventional approach, 2016. URL http://www.offconvex.org/2016/11/03/MityaNN1/.

Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper/2009/file/5751ec3e9a4feab575962e78e006250d-Paper.pdf.

Petros Drineas, Michael W Mahoney, and Nello Cristianini. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6(12), 2005.

Zhouyu Fu. Optimal landmark selection for nyström approximation. In Chu Kiong Loo, Keem Siah Yap, Kok Wai Wong, Andrew Teoh, and Kaizhu Huang, editors, *Neural Information Processing*, pages 311–318, Cham, 2014. Springer International Publishing. ISBN 978-3-319-12640-1.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Geoffrey E Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

Tao Hu, Cengiz Pehlevan, and Dmitri B. Chklovskii. A hebbian/anti-hebbian network for online sparse dictionary learning derived from symmetric matrix factorization. In *2014 48th Asilomar Conference on Signals, Systems and Computers*, pages 613–619, 2014. doi: 10.1109/ACSSC.2014.7094519.

Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by nonnegative matrix factorization. *Nature*, 401:788–791, 1999.

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.

Tianyi Lin, Chi Jin, and Michael Jordan. On gradient descent ascent for nonconvex-concave minimax problems. In *International Conference on Machine Learning*, pages 6083–6093. PMLR, 2020.

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2): 129–137, 1982.

Cameron Musco and Christopher Musco. Recursive sampling for the nystr\" om method. *arXiv preprint arXiv:1605.07583*, 2016.

Dina Obeid, Hugo Ramambason, and Cengiz Pehlevan. Structured and deep similarity matching via structured and deep hebbian networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

B.A. Olshausen and D.J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, June 1996.

Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997. ISSN 0042-6989. doi: https://doi.org/10.1016/S0042-6989(97)00169-7. URL https://www.sciencedirect.com/science/article/pii/S0042698997001697.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Cengiz Pehlevan and Dmitri B. Chklovskii. A hebbian/anti-hebbian network derived from online non-negative matrix factorization can cluster and discover sparse features. In *2014 48th Asilomar Conference on Signals, Systems and Computers*, pages 769–775, 2014. doi: 10.1109/ACSSC.2014.7094553.

Cengiz Pehlevan, Tao Hu, and Dmitri B. Chklovskii. A hebbian/anti-hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data. *Neural Computation*, 27(7):1461–1495, 2015. doi: 10.1162/NECO_a_00745.

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. doi: 10.1162/neco_a_01018.

Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

H. Sebastian Seung. Convergence of gradient descent-ascent analyzed as a newtonian dynamical system with dissipation, 2019.

H. Sebastian Seung and Jonathan Zung. A correlation game for unsupervised learning yields computational interpretations of hebbian excitation, anti-hebbian inhibition, and synapse elimination. *CoRR*, abs/1704.00646, 2017. URL `http://arxiv.org/abs/1704.00646`.

Bernt C Skottun, Arthur Bradley, Gary Sclar, Izumi Ohzawa, and Ralph D Freeman. The effects of contrast on visual orientation and spatial frequency discrimination: a comparison of single cells and behavior. *Journal of neurophysiology*, 57(3):773–786, 1987.

Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2001. URL `https://proceedings.neurips.cc/paper/2000/file/19de10adbaa1b2ee13f77f679fa1483a-Paper.pdf`.

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.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] In the abstract we that we provide Hebbian learning rules to perform kernel similarity matching, and we derive these rules in Section 2,3
   (b) Did you describe the limitations of your work? [Yes] See discussion
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] We prove our results in Section 2,3, and the Appendix
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will include code in supplemental material

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In section 4 we provide these details

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] These are small scale experiments taking only a few minutes on a GPU

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We cite scikit-learn for generating the halfmoons dataset, and the creators the mnist dataset.

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use public datasets

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use public datasets

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Lemma 1

Proof of Lemma 1. In this section we will derive a correlation-based upper bound for the nonlinear pairwise sum $\sum_{t,t'} f(\mathbf{x}_t, \mathbf{x}_{t'})\mathbf{y}_t \cdot \mathbf{y}_{t'}$ in Equation 2. The key to creating this bound will be to replace all nonlinear similarities $f(\mathbf{u}, \mathbf{v})$ with the dot product between high dimensional vectors $\boldsymbol{\phi}_u \cdot \boldsymbol{\phi}_v$. This is allowed because we have assumed that f is a positive semidefinite kernel. Formally this assumption means that for any set of M-dimensional vectors $\{\mathbf{w}, \mathbf{x}_1, \ldots, \mathbf{x}_T\}$, there exists a corresponding set of (at most) $T+1$-dimensional vectors $\{\boldsymbol{\phi}_\mathbf{w}, \boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_T\}$ whose inner products yield the similarity defined by f:

$$\boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{t'} = f(\mathbf{x}_t, \mathbf{x}_{t'}) \text{ and } \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{x}_t, \mathbf{w}) \text{ and } \boldsymbol{\phi}_\mathbf{w} \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{w}, \mathbf{w}) \tag{20}$$

Assume we have some corresponding set of $T+1$ scalars $\{q, y_1, \ldots, y_T\}$. Now consider the vector difference $\frac{1}{T}\sum_t y_t \boldsymbol{\phi}_t - q\boldsymbol{\phi}_w$. The squared norm of this difference is of course non-negative. Additionally we can expand out this square:

$$0 \leq \frac{1}{2}\left\|\frac{1}{T}\sum_t y_t \boldsymbol{\phi}_t - q\boldsymbol{\phi}_\mathbf{w}\right\|^2 = \frac{1}{2T^2}\sum_{s,t} y_s y_t \boldsymbol{\phi}_s \cdot \boldsymbol{\phi}_t - \frac{1}{T}\sum_t q y_t \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_\mathbf{w} + \frac{1}{2}q^2 \boldsymbol{\phi}_\mathbf{w} \cdot \boldsymbol{\phi}_\mathbf{w} \tag{21}$$

At this point we can simply replace all dot products with the equivalent nonlinear similarities $f(\cdot, \cdot)$ in Equation 20 and rearrange the terms to yield our key inequality (Eq. 3) which we write again:

$$\frac{1}{2T^2}\sum_{s,t} y^s y^t f(\mathbf{x}^s, \mathbf{x}^t) \geq \frac{1}{T}\sum_t q y^t f(\mathbf{x}^t, \mathbf{w}) - \frac{1}{2}q^2 f(\mathbf{w}, \mathbf{w}) \tag{22}$$

□

B Interpretation using rank-1 Nyström approximation

The bound in Equation 22 can be interpreted using a rank-1 Nyström approximation for $f(\mathbf{x}_t, \mathbf{x}_{t'})$. By holding $\mathbf{w}$ fixed and maximizing for q in the right hand side of Equation 22, we get $q^* =$

```
# train loop
for i in range(n_iter):
    # inference
    x = next(loader)
    y = self.forward(x)

    # gradients
    e = self.energy(x,y)
    gq, gw, gl = torch.autograd.grad(e, [self.q, self.w, self.l])

    # updates
    with torch.no_grad():
        self.q += etaq * gq
        self.w += etaw * gw / (self.q**2).unsqueeze(1)
        self.l -= etal * gl
```

Figure 7: Training loop to perform the GDA-based optimazation of Eq. 9 written using PyTorch

$f(\mathbf{w},\mathbf{w})^\dagger \sum_t y_t f(\mathbf{x}_t,\mathbf{w})$ where $f(\mathbf{w},\mathbf{w})^\dagger$ indicates the pseudo-inverse.[1] We can insert this optimal q^* back into the right hand side to yield:

$$\sum_t q^* y_t f(\mathbf{x}_t,\mathbf{w}) - \frac{1}{2}(q^*)^2 f(\mathbf{w},\mathbf{w}) = \frac{1}{2}\sum_{t,t'} y_t f^{\text{Nyström}}_{t,t'} y_{t'} \tag{23}$$

where we have defined they Nyström matrix:

$$f^{\text{Nyström}}_{t,t'} := f(\mathbf{x}_t,\mathbf{w}) f(\mathbf{w},\mathbf{w})^\dagger f(\mathbf{w},\mathbf{x}_{t'}) \tag{24}$$

The matrix $f^{\text{Nyström}}_{t,t'}$ is a rank-1 Nyström approximation for the full similarity matrix $f(\mathbf{x}_t,\mathbf{x}_{t'})$ Williams and Seeger [2001]. Note that for every dimension i of the representation vector $\mathbf{y}$, we have a different landmark vector $\mathbf{w}^i$ so we are using a different rank-1 approximation of the matrix $f(\mathbf{x}_t,\mathbf{x}_{t'})$ for every to the pairwise sum $\frac{1}{2}\sum_{t,t'} y^i_t y^i_{t'} f(\mathbf{x}_t,\mathbf{x}_{t'})$.

Typically the weight vector $\mathbf{w}$, often called a "landmark", used in the Nyström approximation is set either by setting it to a random input or by more sophisticated schemes like setting it with KMeans. In our case, we are directly optimizing the landmarks via Equation 6. To our knowledge the only other work to do this was performed in Fu [2014].

C PyTorch code for training

The code used in the main training loop of our algorithm is shown in Fig. 7.

D Homogeneous kernels

Before moving on we note that a simplification can be made if we have a homogenous (scale free) kernel, i.e. if $f(a\mathbf{u},b\mathbf{v}) = (ab)^\alpha f(\mathbf{u},\mathbf{v})$. Examples of such kernels are the linear kernel $f(\mathbf{u},\mathbf{v}) = \mathbf{u}\cdot\mathbf{v}$, homogeneous polynomial kernels $f(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^\alpha$, and the cosine-based kernel we will use in one of our experiments $f(\mathbf{u},\mathbf{v}) = \|\mathbf{y}\|\|\mathbf{v}\|(\hat{\mathbf{u}}\cdot\hat{\mathbf{v}})^\alpha$. In this case, there is a degeneracy between q_i and the norm of $\mathbf{w}_i$. This means we can actually eliminate the minimization over $\mathbf{q}$ by setting $q_i = 1$. We prove this fact in the appendix. In the case of a homogeneous kernel, we are left with the simpler equivalent optimization:

$$\min_{\mathbf{W}}\max_{\mathbf{L}}\min_{\mathbf{Y}} -\sum_{i=1}^{N}\left[\langle y_i f(\mathbf{w}_i,\mathbf{x})\rangle - \frac{1}{2}f(\mathbf{w}_i,\mathbf{w}_i)\right] + \frac{1}{2}\sum_{i,j=1}^{N}\left[L_{ij}\langle y_i y_j\rangle - \frac{1}{2}L_{ij}^2\right] + \lambda\langle y_i^2\rangle \tag{25}$$

[1] The pseudo-inverse of the scalar $f(\mathbf{w},\mathbf{w})$ acts exactly like the regular inverse except it is defined to be 0 when $f(\mathbf{w},\mathbf{w})$ is zero, unlike the regular inverse which would be undefined.

This more clearly shows the relationship between the linear similarity matching objectives and the more general kernel similarity matching objective. When $f(\mathbf{w}_i, \mathbf{x}) = \mathbf{w}_i \cdot \mathbf{x}$, we are in fact left with the exact objective studied in previous works on linear similarity matching Pehlevan et al. [2018]. This simplification can be easily implemented in code by initializing $q_i = 1$ and setting the learning rate for η_q to be 0 for all iterations.

E Proof that the bound in Equation 6 is maximized when $q = 1$ and f is a homogeneous kernel

Assume that f is a homogenous kernel, so that $f(\lambda_1 \mathbf{u}, \lambda_2 \mathbf{v}) = (\lambda_1 \lambda_2)^\alpha f(\mathbf{u}, \mathbf{v})$ for any $\lambda_1, \lambda_2 > 0$. We will show that in this case we can simply set $q = 1$. Assume we have some pair $q, \mathbf{w}$. Then define $q' = 1$ and $\mathbf{w}' := q^{1/\alpha}\mathbf{w}$. Because our kernel is homogeneous, we have $f(\mathbf{w}', \mathbf{x}_t) = f(q^{1/\alpha}\mathbf{w}, \mathbf{x}_t) = qf(\mathbf{w}, \mathbf{x}_t)$ and similarly $f(\mathbf{w}', \mathbf{w}') = q^2 f(\mathbf{w}, \mathbf{w})$. In other words when we have a homogenous kernel, we can always just rescale the features $\mathbf{w}' \leftarrow q^{1/\alpha}\mathbf{w}$ so the following holds for any q:

$$\sum_t q y_t f(\mathbf{x}_t, \mathbf{w}) - \frac{1}{2} q^2 f(\mathbf{w}, \mathbf{w}) = \sum_t y_t f(\mathbf{x}_t, \mathbf{w}') - \frac{1}{2} f(\mathbf{w}', \mathbf{w}') \tag{26}$$

F Methods we compare to in our experiments

Kernel PCA The optimal (in terms of mean squared error) rank N approximation $\hat{f}$ to the kernel matrix f is given by the top n-dimensional subspace of the kernel matrix [Borg and Groenen, 2005]. Specifically, we perform an eigenvector decomposition on f then set $\hat{f}$ via:

$$f(\mathbf{x}_s, \mathbf{x}_t) = \sum_{i=1}^{T} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top \rightarrow \hat{f}_{st} = \sum_{i=1}^{N} \lambda_i \mathbf{v}_i \mathbf{v}_i^\top \tag{27}$$

For the mnist dataset, the kernel matrix is 70k x 70k entries so we use a randomized svd algorithm to compute the top components, rather than a full SVD. We use the PyTorch implementation "torch.svd_lowrank" with $q = 1024 + 256$ (so we estimate the top 1024+256 singular values and vectors) and we set niter=4 meaning we do 4 power iterations.

Nystrom methods Given a set of landmarks $\{\mathbf{w}_i : i = 1, 2, \ldots, N\}$, the nystrom method defines two matrices:

$$A_{ti} = f(\mathbf{x}^t, \mathbf{w}_i) \qquad B_{ij} = f(\mathbf{x}_i, \mathbf{w}_j) \tag{28}$$

These are used to approximate the kernel matrix via:

$$\hat{f}_{st} = \sum_{ij} A_{si} [\mathbf{B}^\dagger]_{ij} A_{tj}^\top \tag{29}$$

To calculate the pseudo-inverse of $\mathbf{B}$ we use double precision arithmetic and set first set all singular value of B smaller than $1e-10$ to zero. We compare 3 different methods of landmark generation in our paper.

Nystrom with uniformly sampled landmarks This is the simplest method, and was proposed in he original paper using the Nystrom method to approximate kernel matrices [Williams and Seeger, 2001]. We simply uniformly sample N landmarks without replacement from the dataset.

Nystrom with landmarks generated via KMeans This method was used by Zhang et al. [2008] and instead uses the cluster centers given by KMeans as the landmarks. We initialize our means with templates from the dataset and the use Lloyd's method to update our cluster centers [Lloyd, 1982]. This is run either until convergence, or 100 iterations of the algorithm occurs, whichever happens first.

Nystrom with landmarks generated via our method We use the N features learned with Hebbian update rules as the landmarks in the Nystrom approximation.

Random Fourier features For the half moons dataset using the Gaussian kernel, we also compare our method to the random Fourier feature method [Rahimi et al., 2007]. The authors in Bahroun et al.

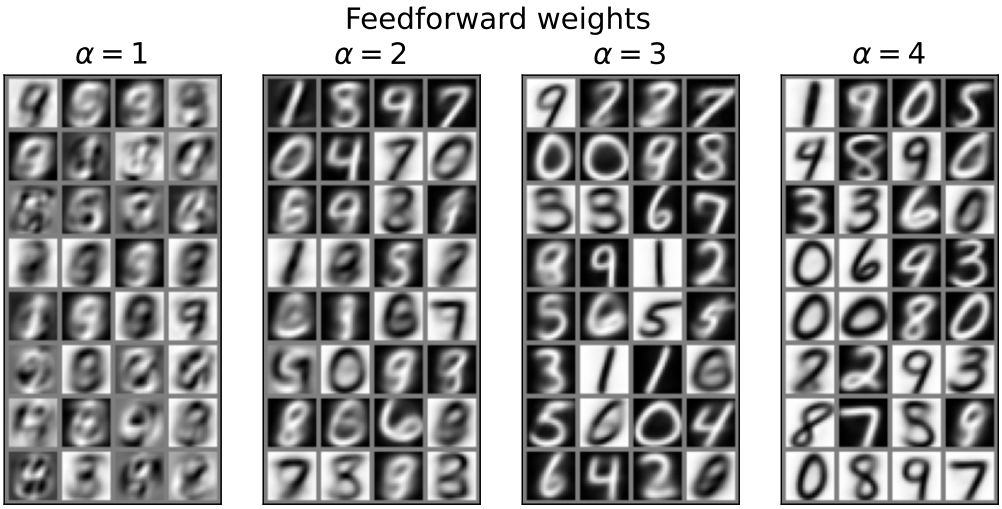

Figure 8: Feedforward weights $(\mathbf{W})$ learned by the network for $\alpha = 1, 2, 3, 4$. When $\alpha = 1$, the weights appear to be complicated linear combinations of input vectors. As α increases, the weights begin to resemble "templates", i.e. whole digits. In the main text, we argue this behavior results from the increasing sharpness of neural tuning as α increases.

[2017] train a linear similarity matching on top of these features. But for simplicity, we just use the random features themselves, rather than the subsequent neurally generated features. This provides a best-case scenario for the neural random Fourier method. The neural algorithm is simply trying to matching similarities $\min y \|\mathbf{y}^s \cdot \mathbf{y}^t - \boldsymbol{\phi}^s \cdot \boldsymbol{\phi}^t\|$, and it should be able to provide zero error, given the same output dimension as input dimension. Although it practice it can be challenging to set the learning rates appropriately, so we evaluate $\boldsymbol{\phi}$ instead of $\mathbf{y}$ to avoid any possible issues with improper training.

To generate these features, we randomly sample $\mathbf{w}_i \sim \mathcal{N}(\text{mean} = 0, \text{variance} = \frac{1}{\sigma^2}\mathbf{I})$ where and $b_i \in \text{Uniform}[0, 2\pi]$ as set the features as

$$\phi_i^t = \sqrt{\frac{2}{n}} \cos(\mathbf{w}_i \cdot \mathbf{x}^t + b_i) \tag{30}$$

G Half Moons Experiment

G.0.1 Training details

We train with minibatch sizes of 64 input. We train for 10000 iterations with $\eta_w = \eta_q = 0.01$ and $\eta_l = 0.1$. Then we anneal the learning rates by a factor of 10x and train for 10000 more iterations $\eta_w = \eta_q = 0.001$ and $\eta_l = 0.01$.

H MNIST Experiment

H.1 Training details

We train with minibatch sizes of 64. After initialization and warmup, we set α and train for 10,000 iterations with $\eta_w = 0.001, \eta_l = 0.01$. We then decay the learning rates for W, L by 10x and train for with 5k more iterations with $\eta_w = 0.0001, \eta_l = 0.001$. This whole procedure takes approximately 4 minutes on an NVIDIA GTX 1080 GPU.

H.2 Learned features for MNIST dataset

In Figure 8 we show the weights $\mathbf{w}_i$, visualized as 20x20 images, that are learned. When $\alpha = 1$ (linear similarity matching), the features appear as complicated linear combinations of input digits.

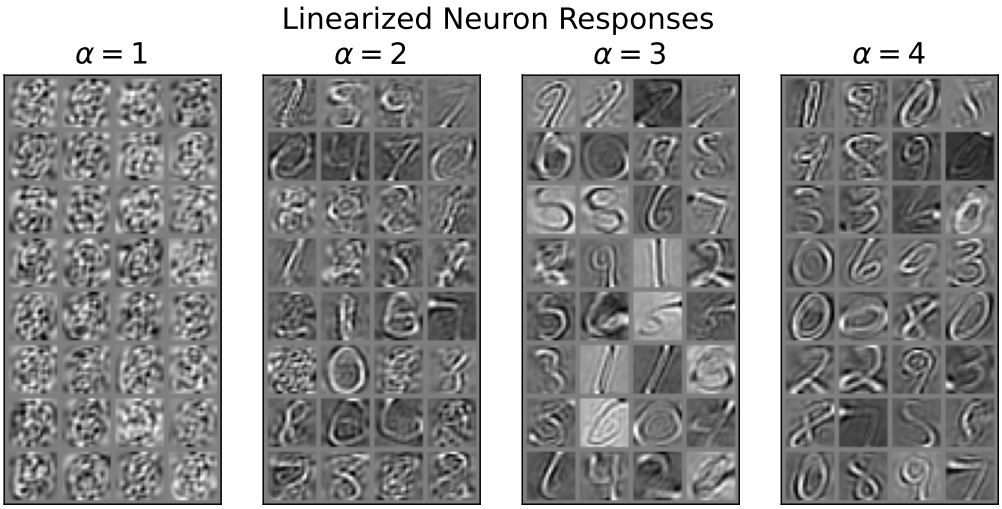

Figure 9: "Linearized responses" for a subset of neurons from networks with $\alpha = \{1, 2, 3, 4\}$. Specifically, for each neuron y_i we compute the vector $\mathbf{s}_i = \left[0.1\,\mathbf{I} + \langle \mathbf{x}\mathbf{x}^\top \rangle\right]^{-1} \langle y_i \mathbf{x} \rangle$ and visualize $\mathbf{s}_i$ as a 20×20 image. As α increases, it appears that neurons become increasingly selective to whole input digits.

However, with $\alpha = 2$ we see clear digits beginning to emerge. And with $\alpha = 4$ nearly all the features look like whole digits.

I Receptive field analysis (aka "linearized neuron responses")

A natural way to visualize what the networks learn is to examine the feedforward weights. However these visualizations are not as interpretable in this experiment as they were for the simple halfmoons dataset. In particular for $\alpha = 1, 2, 3$ the weights appear to be a blend of templates (whole digits) and more complicated linear combinations of digits. We show some examples from each network configuration in the appendix.

We can better understand and visualize the network responses by instead examining the linearized neuron responses. Specifically, for each neuron y_i we compute the vector $\mathbf{s}_i = \left[0.1\,\mathbf{I} + \langle \mathbf{x}\mathbf{x}^\top \rangle\right]^{-1} \langle y_i \mathbf{x} \rangle$. This vector can be thought of as a linear approximation to each neuron $y_i \approx \mathbf{s}_i \cdot \mathbf{x}$. We show these vectors, again visualized as images, in Figure 9.

These linearized responses actually highlight a behavior not seen by only considering feedforward weights. We see for $\alpha = 2$, it appears that many of the neurons appear to be selective for smaller regions of the input, sometimes interpretable as strokes and edges. This behavior is likely coming from some sort of cancellation between the feedforward input and lateral interactions. As α increases, the linear filters appear to grow in size to resemble whole digits.

For $\alpha = 1$ (aka linear similarity matching) the linearized responses do not in any way appear as whole digits, rather they appear to be high spatial frequency images. This is not a failure of the networks, as the input-output similarities are nearly perfectly matched. This behavior results from the fact that linearity is not enough to encourage parts or whole templates to be learned.

I.1 Spectral analysis of the representations

We examine the eigenvalue spectrum of the input similarity matrix $f(\mathbf{x}_t, \mathbf{x}_{t'})$ and the output similarity matrix $\mathbf{y}_t \cdot \mathbf{y}_{t'}$. We plot these spectra in Figure 10. Note that we normalize the spectra by dividing by the largest eigenvalue.

Without even considering the output representations, we can already observe interesting behavior just by considering the spectrum of the input similarity matrix. As we increase α, the "sharpness" of the kernel, the spectrum of f tends to flatten out. The effective rank of this matrix increases

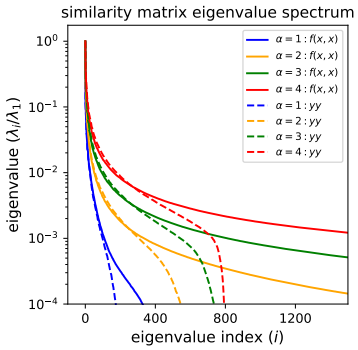

Figure 10: Eigenvalue spectrum of the input similarity matrix $f(\mathbf{x}_t, \mathbf{x}_{t'})$ and learned output similarity matrix $\mathbf{y}_t \cdot \mathbf{y}_{t'}$. If similarity matching were optimal, (i.e. we just performed uncentered kernel PCA on the input similarity matrix) the largest 800 eigenvalues would be exactly matched and subsequent eigenvalues would be zero. We see that increasing α brings up the tails of the spectrum, approximately "whitening" the responses. For $\alpha = 1$, because the inputs are 400 dimensional, the spectrum only has at most 400 nonzero eigenvalues.

with increasing kernel sharpness. This observation is is in part a motivation for kernel similarity matching. Matching a high rank matrix naturally requires high dimensional vectors. This increase in dimensionality may be useful for downstream tasks such as linear classification. It is also an important part of brain inspired modeling to use overcomplete representations of the input Olshausen and Field [1997].

For $\alpha = 1$, the spectrum of $\mathbf{y}_t \cdot \mathbf{y}_{t'}$ closely matches the spectrum $f(\mathbf{x}_t, \mathbf{x}_{t'})$ for the larger eigenvalues. However, it appears to fall off for smaller eigenvalues. This may be due in part to the training not being fully converged. For $\alpha > 1$, the spectrum of $\mathbf{y}_t \cdot \mathbf{y}_{t'}$ approximately matches for the larger eigenvalues (although not perfectly). However again the spectrum tends to fall off more rapidly for the learned representations than for the input similarity matrix. Note that because the dimensionality of $\mathbf{y}$ is 800 for all experiments, the spectrum necessarily must be zero for all eigenvalues smaller than the 800th largest eigenvalue.