ONLINE APPROXIMATE FACTORIZATION OF A KERNEL MATRIX BY A HEBBIAN NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

We derive an online algorithm for unsupervised learning based on representing every input $\mathbf{x}_t$ by a high dimensional vector $\mathbf{y}_t$ with pairwise inner products that approximately match input similarities as measured by a kernel function: $\mathbf{y}_s \cdot \mathbf{y}_t \approx f(\mathbf{x}_s, \mathbf{x}_t)$. The approximation is formulated using the objective function for classical multidimensional scaling. We derive an upper bound for this objective which only involves correlations between output vectors and nonlinear functions of input vectors. Minimizing this upper bound leads to a minimax optimization, which can be solved via stochastic gradient descent-ascent. This online algorithm can be interpreted as a recurrent neural network with Hebbian and anti-Hebbian connections, generalizing previous work on linear similarity matching. Numerical experiments demonstrate that unsupervised learning can be aided by the nonlinearity inherent in our kernel method, which also makes learning high dimensional representations meaningful. We show that heavy-tailed representation vectors emerge from the learning even though no sparseness prior is used, lending further biological plausibility to the model. Our upper bound employs a series of rank-one Nystrom approximations to the kernel function, with the novelty of leading to an online algorithm that optimizes landmark placement.

1 INTRODUCTION

Kernel methods in machine learning are used to map input vectors $\mathbf{x}$ to high-dimensional feature vectors $\Phi(\mathbf{x})$ that are more linearly separable and generally better suited for downstream learning than the original inputs (Hofmann et al., 2008). Instead of specifying $\Phi(\mathbf{x})$ directly, it is often convenient to specify a kernel function $\mathcal{K}(\cdot,\cdot)$ such that $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = \mathcal{K}(\mathbf{x}, \mathbf{x}')$. Kernel algorithms are written in terms of the Gram matrix $\mathcal{K}(\mathbf{x}^s, \mathbf{x}^t)$, where the indices s and t run over the input vectors in a dataset (Cortes & Vapnik, 1995; Zhu & Hastie, 2002). There is no need to explicitly compute the feature vectors $\Phi(\mathbf{x}^s)$. However, this approach becomes unwieldy for large datasets, because the size of the Gram matrix scales quadratically with the size of the dataset, a problem that is exacerbated by dataset augmentations.

Subsequent methods were developed to find explicit feature vectors specified by the kernel and then train standard algorithms using these explicit vectors. These methods still have the benefit of improved linear separability and the simplicity of only needing to specify a kernel but they can dramatically speed up learning compared to conventional kernel methods if the feature vector dimensionality is much smaller than the number of samples. Two popular methods for constructing explicit feature vectors are random Fourier features (Rahimi et al., 2007) and the Nyström method (Williams & Seeger, 2001).

In other related work, it has been shown that neural networks with Hebbian and anti-Hebbian learning rules can be used to match similarities in the special case where the feature vector is linear, $\Phi(\mathbf{x}^s) = \mathbf{x}^s$ or $\mathcal{K} = \mathbf{x}^s \cdot \mathbf{x}^t$ (Hu et al., 2014; Pehlevan & Chklovskii, 2014; Pehlevan et al., 2015; 2018). This operation is unlikely to have much (or any) impact on downstream learning tasks which can be kernelized (nearest neighbor classification, logistic regression, support vector machines) for the simple reason that these features do not modify the input similarity matrix (assuming the approximation is close).

However these neural algorithms are still interesting as models of learning in real brains. They are recurrent networks with correlation-based learning rules (Pehlevan et al., 2018) and can be modified to

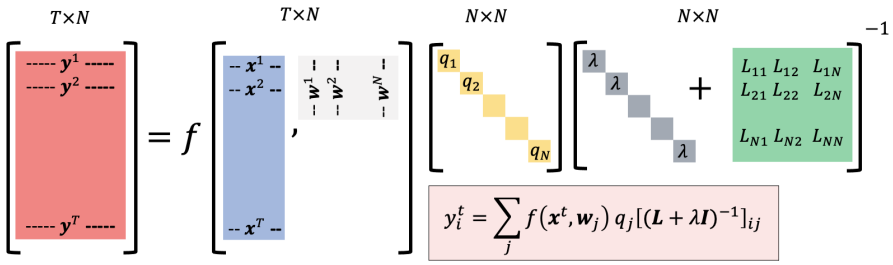

Figure 1: We learn an explicit feature representation which approximates the high dimensional kernel representations: $\mathbf{y}_s \cdot \mathbf{y}_t \approx f(\mathbf{x}_s, \mathbf{x}_t)$. Stochastic gradient descent-ascent on Eq. 8 yields online Hebbian update rules for the parameters $\mathbf{w}_i, q_i, L_{ij}$ and λ is a fixed small parameter used for numerical stability.

include non-negativity Pehlevan & Chklovskii (2014), sparsity, and convolutional structure (Obeid et al., 2019). Additionally these algorithms are qualitatively different from the random Fourier and Nyström methods because they formulate the matrix factorization as a minimax optimization which allows for online algorithms. It is therefore a natural question if these linear matrix factorization algorithms can be extended to the more general setting of nonlinear kernel to yield more powerful features.

In this work we will show how to modify these online neural algorithms so they can compute explicit embeddings which are defined by more a general nonlinear kernel similarity. The key features of our algorithm will be a) an explicit feature embedding $\Phi(\mathbf{x})$ defined by the outputs of a recurrent neural network and b) online learning rules which update the weights of the network proportionally to nonlinear function of input and output neurons.

2 Kernel Matrix Factorization

Formally, 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. We formulate *kernel matrix factorization* as finding a corresponding set of representations $\{\mathbf{y}^t \in \mathbb{R}^N\}_{t=1}^T$ such that for all pairs (s, t):

$$\mathbf{y}^s \cdot \mathbf{y}^t \approx f(\mathbf{x}^s, \mathbf{x}^t) \tag{1}$$

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 goal can be formalized by minimizing the classical multidimensional scaling objective (Borg & Groenen, 2005):

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

This reduces to a previous "similarity matching" objective function in the special case of a linear feature vector $\Phi(\mathbf{x}) = \mathbf{x}$ (Williams, 2002; Pehlevan et al., 2018). OUr work generalizes to nonlinear or kernel similarity matching.

The minimization of Eq. (2) can be solved exactly by finding the top N eigenvectors of the $T \times T$ input similarity matrix (Borg & Groenen, 2005), and is therefore closely related to kernel PCA (Schölkopf et al., 1997). However, this algorithm requires computing and storing similarities for all pairs of input vectors which is the operation we are trying to avoid in the first place. The main goal of this paper is to find an online algorithm, which processes one input vector at a time. We will find an approximate solution to this problem in the form of a neural network with biologically plausible learning rules. We will further show with examples how the nonlinearity can aid unsupervised learning.

2.1 FACTORED UPPER BOUND

Kernel matrix factorization as in Eq. (2) can be formulated as minimizing

$$\frac{1}{T^2}\sum_{s,t}^{T}\left(f(\mathbf{x}^s,\mathbf{x}^t)-\mathbf{y}^s\cdot\mathbf{y}^t\right)^2=-\frac{2}{T^2}\sum_{s,t}f(\mathbf{x}^s,\mathbf{x}^t)\mathbf{y}^s\cdot\mathbf{y}^t+\frac{1}{T^2}\sum_{s,t}(\mathbf{y}^s\cdot\mathbf{y}^t)^2+\text{const} \tag{3}$$

with respect to the output vectors $\mathbf{y}$. To avoid explicitly computing the input kernel matrix, we introduce the following key inequality.

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

$$\frac{1}{2}\sum_{s,t}y^s y^t f(\mathbf{x}^s,\mathbf{x}^t)\geq\sum_t qy^t f(\mathbf{x}^t,\mathbf{w})-\frac{1}{2}q^2 f(\mathbf{w},\mathbf{w}) \tag{4}$$

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

The proof is given in the Appendix, and relies on Mercer's theorem to express the kernel function as $f(\mathbf{u},\mathbf{v})=\boldsymbol{\phi}_u\cdot\boldsymbol{\phi}_v$. The bound can be interpreted using a rank-1 Nystrom approximation for $f(\mathbf{x}^s,\mathbf{x}^t)$ (see Appendix). The key inequality still holds if we maximize the right hand side with respect to q and $\mathbf{w}$. For every index i of the vector $\mathbf{y}$, we insert this maximum back into the first term on the right hand side of Eq. (3). Additionally we rearrange the order of the summations in second term on the right hand side of Eq. (3) to yield the following upper bound for Eq. (3):

$$\min_{\mathbf{y}^t}\min_{q_i,\mathbf{w}_i}-\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\left[q_i y_i^t f(\mathbf{w}_i,\mathbf{x}^t)-\frac{1}{2}q_i^2 f(\mathbf{w}_i,\mathbf{w}_i)\right]+\frac{1}{4}\sum_{i,j=1}^{N}\left(\frac{1}{T}\sum_{t=1}^{T}y_i^t y_j^t\right)^2 \tag{5}$$

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: $\frac{1}{2}C_{ij}^2\rightarrow\max_{L_{ij}}C_{ij}L_{ij}-\frac{1}{2}L_{ij}^2$:

$$\min_{\mathbf{W},\mathbf{Y},\mathbf{q}}\max_{\mathbf{L}}\frac{1}{T}\sum_{t=1}^{T}\left[-\sum_{i=1}^{N}\left[q_i y_i^t f(\mathbf{w}_i,\mathbf{x}^t)-\frac{1}{2}q_i^2 f(\mathbf{w}_i,\mathbf{w}_i)\right]+\frac{1}{2}\sum_{i,j=1}^{N}\left[L_{ij}y_i^t y_j^t-\frac{1}{2}L_{ij}^2\right]\right] \tag{6}$$

We can swap the order of the y and L optimizations, because the objective is "nearly" convex-concave and as such, there is only one saddle point of the optimization with $\mathbf{W},\mathbf{q}$ fixed. 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":

$$e^t:=\sum_{i=1}^{N}-\left[q_i y_i^t f(\mathbf{w}_i,\mathbf{x}^t)-\frac{1}{2}q_i^2 f(\mathbf{w}_i,\mathbf{w}_i)\right]+\frac{1}{2}\sum_{i,j=1}^{N}\left[L_{ij}y_i^t y_j^t-\frac{1}{2}L_{ij}^2\right]+\frac{\lambda}{2}\sum_{i=1}^{N}(y_i^t)^2 \tag{7}$$

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. 5, and is derived as an upper bound to Eq. 2 is thus:

$$\min_{\mathbf{W},\mathbf{q}}\max_{\mathbf{L}}\min_{\mathbf{Y}}\ \frac{1}{T}\sum_{t=1}^{T}e(\mathbf{y}^t,\mathbf{x}^t;\mathbf{W},\mathbf{q},\mathbf{L}) \tag{8}$$

3 NEURAL NETWORK OPTIMIZATION

Applying a stochastic gradient descent-ascent algorithm to Eq. (8) yields a neural network (Fig. 2) 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.11 we compute the $\{\mathbf{y}^b\}$ which minimize Eq. 8 for fixed synapses. Using these optimal $\{\mathbf{y}^b\}$, we compute the minibatch

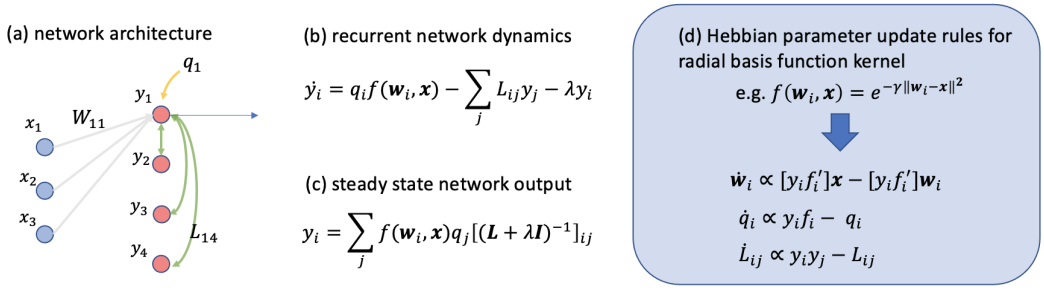

Figure 2: Neural network interpretation of the optimization procedure used to find explicit kernel feature vectors (a) network architecture (b) recurrent network dynamics (c) steady state network response (d) online hebbian update rules for radial basis function kernels (the precise form of these updates will be different when the kernel is not an RBF, but nonlinear correlation-based learning rules can still be derived (Eq. 12,13)

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 $\mathbf{L}$:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i - \frac{\eta_w}{q_i^2}\frac{\partial e}{\partial \mathbf{w}_i} \qquad q_i \leftarrow q_i - \eta_q \frac{\partial e}{\partial q_i} \qquad L_{ij} \leftarrow L_{ij} + \eta_l \frac{\partial e}{\partial L_{ij}} \tag{9}$$

In practice 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 be maximizing Eq. 8 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". Although under certain conditions it can be guaranteed that choosing the learning rate ratio small enough will gaurantee convergence (Seung, 2019).

It is sometimes the case 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 8:

$$\dot{y}_i = \eta_y \left[q_i f(\mathbf{w}_i, \mathbf{x}) - \sum_{j=1}^{N} L_{ij} y_j - \lambda y_i \right] \tag{10}$$

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 2. Note that we can analytically perform the inner loop minimization with a non-neural algorithm:

$$y_i \leftarrow \sum_j [\mathbf{L} + \lambda \mathbf{I}]_{ij}^{-1} q_j f(\mathbf{w}_j, \mathbf{x}) \tag{11}$$

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 & Zung (2017), Pehlevan & 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

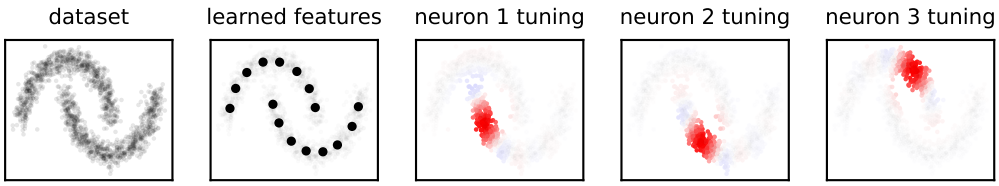

Figure 3: Overview of our Hebbian radial basis function network on half moons dataset (a) dataset, (b) features $\{\mathbf{w}_i\}_{i=1}^{16}$ (c,d,e) response profiles of 3 neurons

ascent on $\mathbf{L}$ provides an algorithm for performing the optimization in Equation 8. 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:

$$\Delta \mathbf{w}_i \propto y_i \partial f(\mathbf{w}_i, \mathbf{x})/\partial \mathbf{w}_i - q_i \partial f(\mathbf{w}_i, \mathbf{w}_i)/\partial \mathbf{w}_i \tag{12}$$

Classically Hebbian rules are 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 & Gerstner (2016)). This 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. We'll show in the next section that common kernels like Gaussian kernel give rise to the standard "linear-in-the-input" update rules.

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

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

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

$$\Delta L_{ij} \propto y_i y_j - L_{ij} \tag{14}$$

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. 12,13 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 $f(\mathbf{v}, \mathbf{v}) = 1$:

$$f(\mathbf{u}, \mathbf{v}) := g(\|\mathbf{u} - \mathbf{v}\|) \quad \text{and} \quad g(0) = 1 \tag{15}$$

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

$$\Delta \mathbf{w}_i \propto [y_i g_i'] \mathbf{x} - [y_i g_i'] \mathbf{w} \quad \Delta q_i \propto y_i g_i - q_i \tag{16}$$

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 Experiment with Half-moons dataset

We train our algorithm on a simple half moons dataset, shown in Figure 3. 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 similarities:

$$f(\mathbf{u}, \mathbf{v}) = e^{-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}} \tag{17}$$

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 $\eta_l = 0.01$.

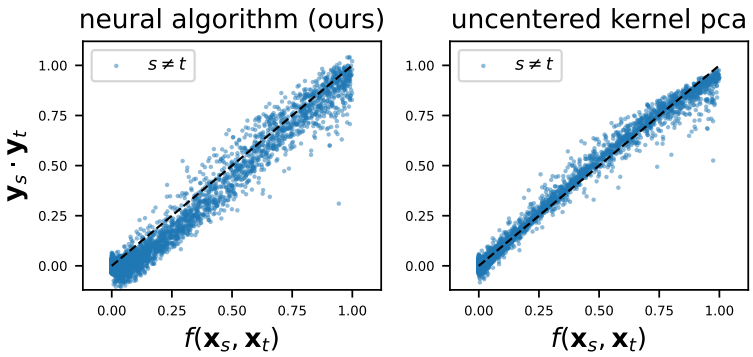

Figure 4: Input-output similarities for 16 dimensional vectors generated by (left) our neural algorithm (right) uncentered kernel PCA

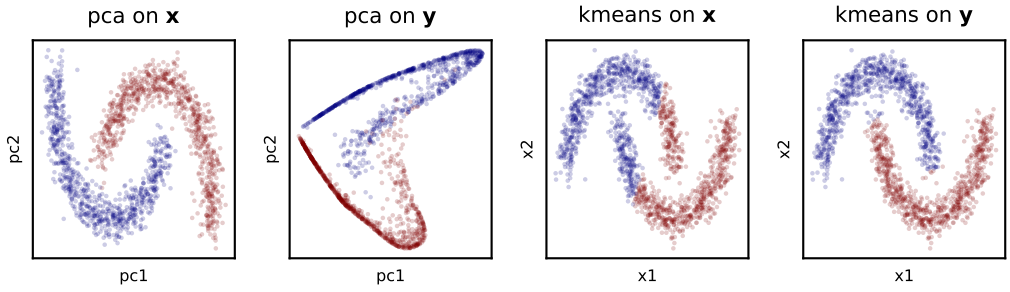

Figure 5: (a) principle components of $\mathbf{x}$ (b) principle components of $\mathbf{y}$. For (a,b), the colors are given by ground truth labels (c) clustering generated by kmeans on $\mathbf{x}$ (d) clustering generated by kmeans on $\mathbf{y}$

4.1 Results

In Fig. 3 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 neurons learn localized responses of the input.

In Fig. 4 we plot the input similarities vs. output similarities generated by our neural algorithm and by the non-Hebbian, but optimal 16 dimensional outputs generated by uncentered Kernel PCA Borg & Groenen (2005). We see that the output similarities roughly match the input similarities, although the optimal kernel PCA solution does seem to have tighter matching.

In Fig. 5 we visualize the principle components of $\mathbf{x}$ and $\mathbf{y}$. 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 correct labels when performed on $\mathbf{y}$. For this dataset, the vectors generated by kernel matrix factorization do in fact appear to be better suited for downstream learning tasks than the original inputs.

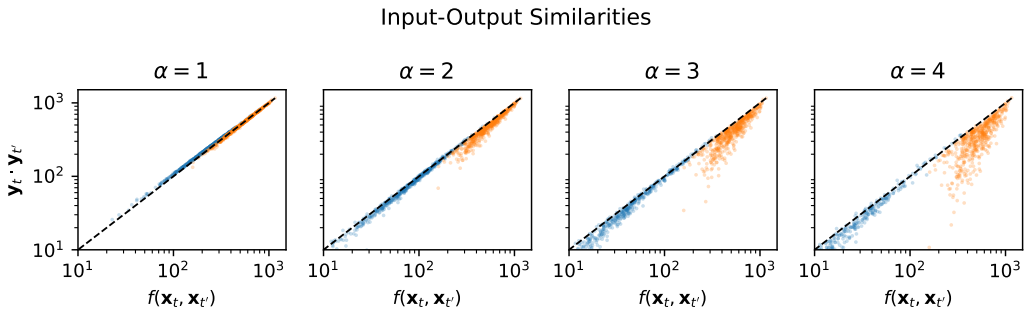

Figure 6: Input-output similarities for nets trained on mnist with cosine-based kernel. As α (the sharpness of the kernel), the error appears to grow. Blue points are $s \neq t$ and orange points are $s = t$.

5 Experiment with MNIST handwritten digits

We train our algorithm on the MNIST handwritten digits dataset LeCun & Cortes (2010). 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). Our networks use 800 neurons, so we are learning 2x overcomplete representations. In our experiments we use kernels of the form:

$$f(\mathbf{u}, \mathbf{v}) = \|\mathbf{u}\|\|\mathbf{v}\|(\hat{\mathbf{u}} \cdot \hat{\mathbf{v}})^{\alpha} \tag{18}$$

We train networks with $\alpha = 1, 2, 3, 4$. Note that $\alpha = 1$ just yields the linear kernel. We are not aware of other works using this exact kernel before, however it is motivated by the *arccosine kernel* studied in the context of wide random ReLU networks (Cho & Saul, 2009). A key 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).

5.1 Training details

Because the kernels are homogeneous we use the trick described in Section A.3 and set $q_i = 1$ and $\eta_q = 0$. This is useful as there is just one less learning rate parameter to set. For all α, we have a "warmup period" where we train for 1k iterations with $\alpha = 1$ and $\eta_w = 0.01, \eta_l = 0.1$. We use this warmup to significantly improve training speed for the large α configurations. When we don't perform this warmup and try to directly train the networks with $\alpha > 2$, many weights quickly decay to nearly zero, and learning proceeds very slowly. In other words, this warmup provides a better initialization for large α networks than the random Gaussian weight initialization.

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.

5.2 Similarity alignment

In Fig. 6 we show scatter plots of input-output similarities $f(\mathbf{x}_s, \mathbf{x}_t)$ vs $\mathbf{y}_t \cdot \mathbf{y}_{t'}$ for $\alpha = 1, 2, 3, 4$. We examine both $s = t$ (the norm of each representation) and $s \neq t$ (different pairs of samples). If the similarities were perfectly matched all points would lie on the line $f(\mathbf{x}_s, \mathbf{x}_t) = \mathbf{y}_s \cdot \mathbf{y}_t$.

For linear kernel $\alpha = 1$, there is essentially no error. This makes sense, when $\alpha = 1$ the bound we derived is tight, and we have more output neurons than inputs. For higher α we do not have perfect matching. We see that for $s \neq t$ similarities are approximately matched. However, there is a systematic bias that the networks tend to underestimate the norm of each representation (orange points): $\|\mathbf{y}_t\|^2 < \|\mathbf{x}_t\|^2$. This error seems to grow with increasing α. There are two sources of

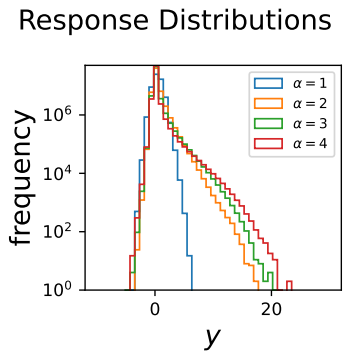

Figure 7: Histogram of all neuron responses over input patterns. See text for details regarding the sign degeneracy for $\alpha = 1, 3$. For linear similarity matching networks ($\alpha = 1$), neuron responses are roughly centered around zero. For $\alpha = 2, 3, 4$, every neuron tends to have a heavy tailed distribution skewed towards large positive values. Additionally, the neurons become increasingly sparse and heavy tailed as we increase α, although this effect seems rather small.

this error. One is that we are only minimizing an upper bound for the original similarity matching objective. Two, as we show in the next section, the dimensionality of the matrix kernel matrix $f(\mathbf{x}_t, \mathbf{x}_{t'})$ is larger than 800, so even the optimal PCA solution with 800 dimensional outputs cannot perfect matching similarities.

5.3 Sparsity of Neural Responses

We compute the distribution of each neuron over all input patterns. Note that 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 can remove this degeneracy by multiply both $\mathbf{w}_i$ and y_i by the sign of $\langle y_i \rangle$. After removing this degeneracy for $\alpha = 1, 3$, we visualize the responses of all neurons over all patterns in Figure 7.

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.

Previous works such as sparse coding (Olshausen & Field, 1996) have explicitly added sparsity and other works such as non-negative matrix factorization by Lee & Seung (1999) have explicity imposed non-negativity. In our network sparsity and approximately signed neuron responses are not imposed on the network, rather they are emergent properties of the algorithm. This sparsity is likely coming from the sharp feedforward input to each neuron $f(\mathbf{w}_i, \mathbf{x}) = \|\mathbf{w}_i\|\|\mathbf{x}\|(\mathbf{w}_i \cdot \mathbf{x})^\alpha$. As α increases, this feedforward input becomes increasingly sharply tuned, meaning the input pattern $\mathbf{x}$ has to closely match a neuron's weight $\mathbf{w}_i$ in order for $(\mathbf{w}_i \cdot \mathbf{x})^\alpha$ to be significantly larger than zero. Most input patterns will not match the weight $\mathbf{w}_i$ but a few will likely match it quite closely as the weights resemble digits.

5.4 Evaluation via Linear Classification

We train a linear classifier on the inputs ($\mathbf{x}$) and the outputs ($\mathbf{y}$) for $\alpha = 1, 2, 3, 4$. 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.

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.

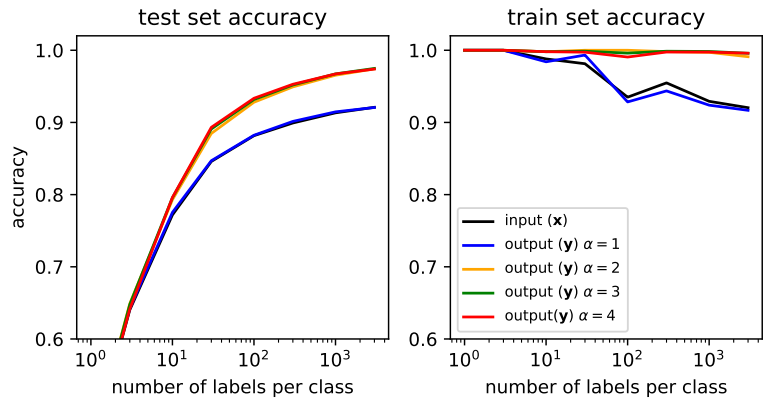

Figure 8: Linear classification results. The pixels (black) and linear network (blue) $(\alpha = 1)$ yield nearly identical performance. Interestingly any amount of nonlinearity $(\alpha = 2, 3, 4)$ seems to yield identical test set performance (yellow, green, red curves).

Perhaps these curves can be partially explained by the spectra of the output similarity matrix which we showed in Figure 12. 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.

6 Discussion

We have extended the works of Hu et al. (2014); Pehlevan & Chklovskii (2014); Pehlevan et al. (2015; 2018) on neural algorithms for linear similarity matching to the more general setting where the input similarity is defined by a nonlinear kernel function. 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 on the representations.

Unlike the works on linear similarity matching, we have resorted to minimizing an upper bound on the pairwise sum in the multidimensional scaling objective rather than a reformulation of this sum. In our experiments, this upper bound still gave surprisingly faithful similarity and spectral alignment between input and output similarities. It would be interesting to see how this result depends on the dataset and the form of the kernel function.

Our work is closely related to works on the Nyström approximation Williams & Seeger (2001). Most closely related is perhaps the work on "optimal" landmarks for the Nyström approximation Fu (2014). This method formulated this selection problem by "minimizing the error of reconstructing input vectors from landmark points in RKHS". The key differences between our work are (a) we derive stochastic online rules and (b) we start with a different objective, which leads us to correlation-based update rules.

It is also interesting view the reformulated kernel similarity matching objective (Equation 3) from the perspective of manifold learning. Popular manifold learning algorithms such as Laplacian eigenmaps Belkin & Niyogi (2001), T-SNE van der Maaten & Hinton (2008), and UMAP McInnes et al. (2018) first construct a sparse neighborhood graph, which only contains pairs of a small number of inputs which were extremely close in the input space. These algorithms differ in the details, but pairs in this graph "pull on each other", while there is some general repulsion to ensure not every pattern is pulled together. The objective in Equation 3 admits a similar interpretation. The inputs for which $f(\mathbf{x}_t, \mathbf{x}_{t'})$ is high yield a high "force", pulling $\mathbf{y}_t$ close to $\mathbf{y}_{t'}$ while there is a general repulsion between the correlations of the representation $\mathbf{y}$. As the kernel becomes increasingly sharp (so moving away from linear similarity matching by increasing α in the case of our cosine kernel), the number of sample pairs with a high force decreases, and $f(\mathbf{x}_t, \mathbf{x}_{t'})$ appears to more closely resemble these sparse neighborhood graphs.

7 REPRODUCIBILITY STATEMENT

To aid with reproducibility, we have added the PyTorch code used to implement our main algorithm in the Appendix. We have proven our core claims in the Appendix. We are also attaching as zip files for the source code used to train our networks and produce our figures.

REFERENCES

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pp. 585–591, Cambridge, MA, USA, 2001. MIT Press.

I. Borg and P. J. F. Groenen. *Classical Scaling*, pp. 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.

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

Corinna Cortes and Vladimir Vapnik. Support-vector networks. 20(3), 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

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

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171 – 1220, 2008. doi: 10.1214/009053607000000677. URL https://doi.org/10.1214/009053607000000677.

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*, pp. 613–619, 2014. doi: 10.1109/ACSSC.2014.7094519.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

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

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018. doi: 10.21105/joss.00861. URL https://doi.org/10.21105/joss.00861.

Dina Obeid, Hugo Ramambason, and Cengiz Pehlevan. *Structured and Deep Similarity Matching via Structured and Deep Hebbian Networks*. Curran Associates Inc., Red Hook, NY, USA, 2019.

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.

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*, pp. 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, pp. 5. Citeseer, 2007.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pp. 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.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

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

Christopher KI Williams. On a connection between kernel pca and metric multidimensional scaling. *Machine Learning*, 46(1):11–19, 2002.

Ji Zhu and Trevor Hastie. Kernel logistic regression and the import vector machine. In T. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002. URL https://proceedings.neurips.cc/paper/2001/file/2eace51d8f796d04991c831a07059758-Paper.pdf.

A APPENDIX

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 3. 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:

$$\boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{t'} = f(\mathbf{x}_t, \mathbf{x}_{t'}) \text{ and } \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{x}_t, \mathbf{w}) \text{ and } \boldsymbol{\phi}_\mathbf{w} \cdot \boldsymbol{\phi}_\mathbf{w} = f(\mathbf{w}, \mathbf{w}) \tag{19}$$

```
# 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 9: Training loop to perform the GDA-based optimazation of Eq. 8 written using PyTorch

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

$$0 \leq \left\| \sum_t y_t \boldsymbol{\phi}_t - q\boldsymbol{\phi}_{\mathbf{w}} \right\|^2 = \frac{1}{2} \sum_{t,t'} y_t y_{t'} \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{t'} - \sum_t q y_t \boldsymbol{\phi}_t \cdot \boldsymbol{\phi}_{\mathbf{w}} + \frac{1}{2} q^2 \boldsymbol{\phi}_{\mathbf{w}} \cdot \boldsymbol{\phi}_{\mathbf{w}} \tag{20}$$

At this point we can simply replace all dot products with the equivalent nonlinear similarities $f(\cdot, \cdot)$ in Equation 19 to yield our key inequality: □

A.1 Interpretation using rank-1 Nystrom approximation

The bound in Equation 4 can be interpreted using a rank-1 Nystrom 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 4, we get $q^* = f(\mathbf{w}, \mathbf{w})^\dagger \sum_t y_t f(\mathbf{x}_t, \mathbf{w})$ 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:

$$\sum_t q^* y_t f(\mathbf{x}_t, \mathbf{w}) - \frac{1}{2}(q^*)^2 f(\mathbf{w}, \mathbf{w}) = \frac{1}{2} \sum_{t,t'} y_t f^{\text{nystrom}}_{t,t'} y_{t'} \tag{21}$$

where we have defined they Nystrom matrix:

$$f^{\text{nystrom}}_{t,t'} := f(\mathbf{x}_t, \mathbf{w}) f(\mathbf{w}, \mathbf{w})^\dagger f(\mathbf{w}, \mathbf{x}_{t'}) \tag{22}$$

The matrix $f^{\text{nystrom}}_{t,t'}$ is a rank-1 Nystrom approximation for the full similarity matrix $f(\mathbf{x}_t, \mathbf{x}_{t'})$ Williams & 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 Nystrom approximation is set either by setting it to a random input or by more sophistcated schemes like setting it with KMeans. In our case, we are directly optimizing the landmarks via Equation 5. To our knowledge the only other work to do this was performed in Fu (2014).

A.2 PyTorch code for training

The code used in the main training loop of our algorithm is shown in Fig. 9.

[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.

A.3 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 degenecary 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:

$$\min_{\mathbf{W}} \max_{\mathbf{L}} \min_{\mathbf{Y}} - \sum_{i=1}^{N} \left[\langle y_i f(\mathbf{w}_i, \mathbf{x}) \rangle - \frac{1}{2} f(\mathbf{w}_i, \mathbf{w}_i) \right] + \frac{1}{2} \sum_{i,j=1}^{N} \left[L_{ij} \langle y_i y_j \rangle - \frac{1}{2} L_{ij}^2 \right] + \lambda \langle y_i^2 \rangle \quad (23)$$

$$\min_{\mathbf{W}} \max_{\mathbf{L}} \min_{\mathbf{Y}} \frac{1}{T} \sum_{t=1}^{T} \left[- \sum_{i=1}^{N} \left[y_i^t f(\mathbf{w}_i, \mathbf{x}^t) - \frac{1}{2} f(\mathbf{w}_i, \mathbf{w}_i) \right] + \frac{1}{2} \sum_{i,j=1}^{N} \left[L_{ij} \langle y_i y_j \rangle - \frac{1}{2} L_{ij}^2 \right] + \frac{\lambda}{2} \sum_{i=1}^{N} (y_i^t)^2 \right] \quad (24)$$

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.

A.4 PROOF THAT THE BOUND IN EQUATION 5 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 $f(\mathbf{w}', \mathbf{x}_t) = f(q^{1/\alpha} \mathbf{w}, \mathbf{x}_t) = q f(\mathbf{w}, \mathbf{x}_t)$ 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:

$$\sum_t q y_t f(\mathbf{x}_t, \mathbf{w}) - \frac{1}{2} q^2 f(\mathbf{w}, \mathbf{w}) = \sum_t y_t f(\mathbf{x}_t, \mathbf{w}') - \frac{1}{2} f(\mathbf{w}', \mathbf{w}') \quad (25)$$

A.5 MNIST EXPERIMENT

A.5.1 LEARNED FEATURES FOR MNIST DATASET

In Figure 10 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. However, with $\alpha = 2$ we see clear digits beginning to emerge. And with $\alpha = 4$ nearly all the features look like whole digits.

A.6 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 11.

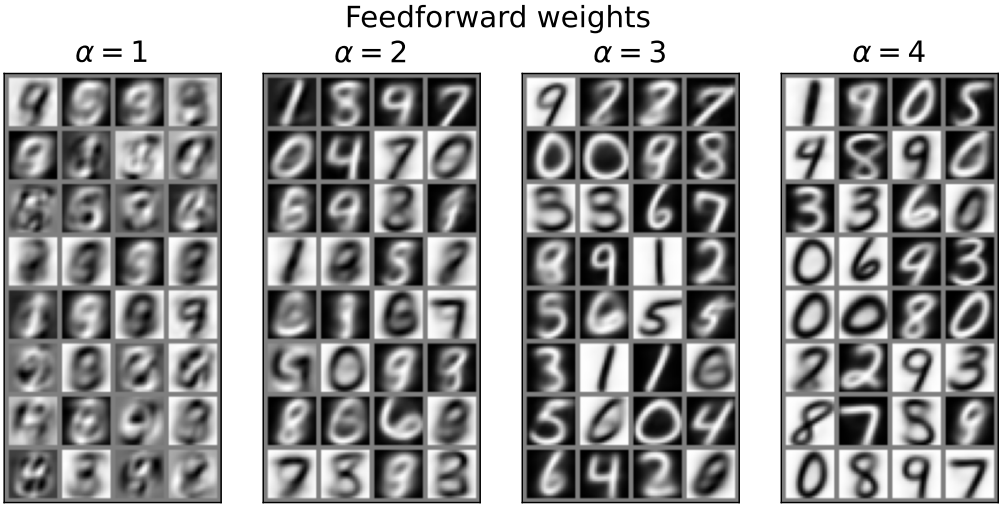

Figure 10: Feedforward weights $(\mathbf{W})$ learned by the network for $\alpha = 1, 2, 3, 4$. When $\alpha = 1$, the weights appear to be complicated linear combinations of input vectors. As α increases, the weights begin to resemble "templates", i.e. whole digits. In the main text, we argue this behavior results from the increasing sharpness of neural tuning as α increases.

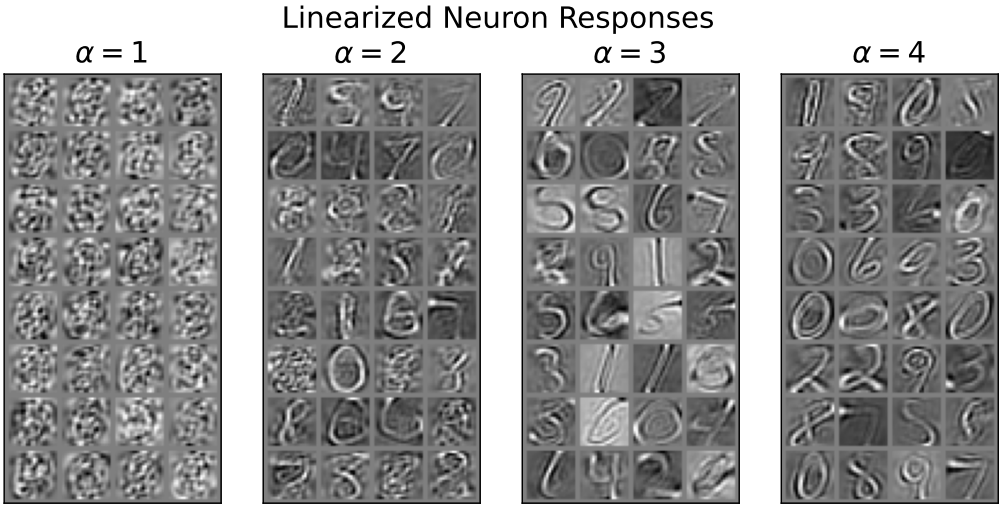

Figure 11: "Linearized responses" for a subset of neurons from networks with $\alpha = \{1, 2, 3, 4\}$. Specifically, for each neuron y_i we compute the vector $\mathbf{s}_i = \left[0.1\ \mathbf{I} + \langle \mathbf{x}\mathbf{x}^\top \rangle\right]^{-1} \langle y_i \mathbf{x} \rangle$ and visualize $\mathbf{s}_i$ as a 20×20 image. As α increases, it appears that neurons become increasingly selective to whole input digits.

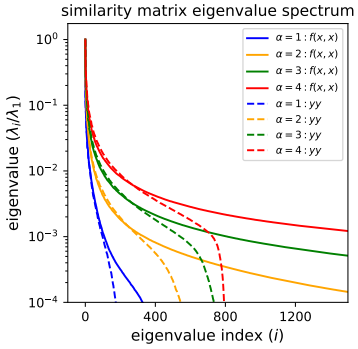

Figure 12: Eigenvalue spectrum of the input similarity matrix $f(\mathbf{x}_t, \mathbf{x}_{t'})$ and learned output similarity matrix $\mathbf{y}_t \cdot \mathbf{y}_{t'}$. If similarity matching were optimal, (i.e. we just performed uncentered kernel pca on the input similarity matrix) the largest 800 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.

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.

A.6.1 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 12. 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 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 & 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.