LETTER Communicated by Erkki Oja

A Hebbian/Anti-Hebbian Neural Network for Linear Subspace Learning: A Derivation from Multidimensional Scaling of Streaming Data

Cengiz Pehlevan
cpehlevan@simonsfoundation.org
Janelia Research Campus, Howard Hughes Medical Institute, Ashburn, VA 20147, and Simons Center for Analysis, Simons Foundation, New York, NY 10010, U.S.A.

Tao Hu
taohu@tees.tamus.edu
Texas A&M University, College Station, TX 77843, U.S.A.

Dmitri B. Chklovskii
dchklovskii@simonsfoundation.org
Simons Center for Analysis, Simons Foundation, New York, NY 10010, U.S.A.

Neural network models of early sensory processing typically reduce the dimensionality of streaming input data. Such networks learn the principal subspace, in the sense of principal component analysis, by adjusting synaptic weights according to activity-dependent learning rules. When derived from a principled cost function, these rules are nonlocal and hence biologically implausible. At the same time, biologically plausible local rules have been postulated rather than derived from a principled cost function. Here, to bridge this gap, we derive a biologically plausible network for subspace learning on streaming data by minimizing a principled cost function. In a departure from previous work, where cost was quantified by the representation, or reconstruction, error, we adopt a multidimensional scaling cost function for streaming data. The resulting algorithm relies only on biologically plausible Hebbian and anti-Hebbian local learning rules. In a stochastic setting, synaptic weights converge to a stationary state, which projects the input data onto the principal subspace. If the data are generated by a nonstationary distribution, the network can track the principal subspace. Thus, our result makes a step toward an algorithmic theory of neural computation.

1 Introduction

Early sensory processing reduces the dimensionality of streamed inputs (Hyvärinen, Hurri, & Hoyer, 2009), as evidenced by a high ratio of input to output nerve fiber counts (Shepherd, 2003). For example, in the human

Neural Computation **27, 1461–1495** (2015)
doi:10.1162/NECO_a_00745

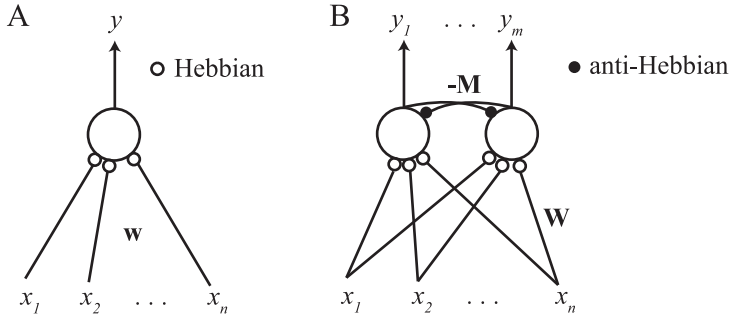

Figure 1: An Oja neuron and our neural network. (A) A single Oja neuron computes the principal component, y, of the input data, $\mathbf{x}$, if its synaptic weights follow Hebbian updates. (B) A multineuron network computes the principal subspace of the input if the feedforward connection weight updates follow a Hebbian and the lateral connection weight updates follow an anti-Hebbian rule.

retina, information gathered by approximately 125 million photoreceptors is conveyed to the lateral geniculate nucleus through 1 million or so ganglion cells (Hubel, 1995). By learning a lower-dimensional subspace and projecting the streamed data onto that subspace, the nervous system denoises and compresses the data simplifying further processing. Therefore, a biologically plausible implementation of dimensionality reduction may offer a model of early sensory processing.

For a single neuron, a biologically plausible implementation of dimensionality reduction in the streaming, or online, setting has been proposed in the seminal work of Oja (1982; see Figure 1A). At each time point, t, an input vector, $\mathbf{x}_t$, is presented to the neuron, and, in response, it computes a scalar output, $y_t = \mathbf{w}\mathbf{x}_t$, where $\mathbf{w}$ is a row-vector of input synaptic weights. Furthermore, synaptic weights $\mathbf{w}$ are updated according to a version of Hebbian learning called Oja's rule,

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_t(\mathbf{x}_t^\top - \mathbf{w}y_t), \tag{1.1}$$

where η is a learning rate and $^\top$ designates a transpose. Then the neuron's synaptic weight vector converges to the principal eigenvector of the covariance matrix of the streamed data (Oja, 1982). Importantly, Oja's learning rule is local, meaning that synaptic weight updates depend on the activities of only pre- and postsynaptic neurons accessible to each synapse and therefore biologically plausible.

Oja's rule can be derived by an approximate gradient descent of the mean squared representation error (Cichocki & Amari, 2002; Yang, 1995), a so-called synthesis view of principal component analysis (PCA) (Pearson,

1901; Preisendorfer & Mobley, 1988):

$$\min_{\mathbf{w}} \sum_t \|\mathbf{x}_t - \mathbf{w}^\top \mathbf{w}\mathbf{x}_t\|_2^2. \tag{1.2}$$

Computing principal components beyond the first requires more than one output neuron and motivated numerous neural networks. Some well-known examples are the generalized Hebbian algorithm (GHA) (Sanger, 1989), Földiak's network (Földiak, 1989), the subspace network (Karhunen & Oja, 1982), Rubner's network (Rubner & Tavan, 1989; Rubner & Schulten, 1990), Leen's minimal coupling and full coupling networks (Leen, 1990, 1991), and the APEX network (Kung & Diamantaras, 1990; Kung, Diamantaras, & Taur, 1994). We refer to Becker and Plumbley (1996), Diamantaras and Kung (1996), and Diamantaras (2002) for a detailed review of these and further developments.

However, none of the previous contributions was able to derive a multineuronal single-layer network with local learning rules by minimizing a principled cost function, in a way that Oja's rule, equation 1.1, was derived for a single neuron. The GHA and the subspace rules rely on nonlocal learning rules: feedforward synaptic updates depend on other neurons' synaptic weights and activities. Leen's minimal network is also nonlocal: feedforward synaptic updates of a neuron depend on its lateral synaptic weights. While Földiak's, Rubner's, and Leen's full coupling networks use local Hebbian and anti-Hebbian rules, they were postulated rather than derived from a principled cost function. APEX network perhaps comes closest to our criterion: the rule for each neuron can be related separately to a cost function that includes contributions from other neurons. But no cost function describes all the neurons combined.

At the same time, numerous dimensionality-reduction algorithms have been developed for data analysis needs, disregarding the biological plausibility requirement. Perhaps the most common approach is again principal component analysis (PCA), which was originally developed for batch processing (Pearson, 1901) but later adapted to streaming data (Yang, 1995; Crammer, 2006; Arora, Cotter, Livescu, & Srebro, 2012; Goes, Zhang, Arora, & Lerman, 2014). (For a more detailed collection of references, see, e.g., Balzano, 2012.) These algorithms typically minimize the representation error cost function:

$$\min_{\mathbf{F}} \|\mathbf{X} - \mathbf{F}^\top \mathbf{F}\mathbf{X}\|_F^2, \tag{1.3}$$

where $\mathbf{X}$ is a data matrix and $\mathbf{F}$ is a wide matrix (for detailed notation, see below). The minimum of equation 1.3 is when rows of $\mathbf{F}$ are orthonormal

and span the m-dimensional principal subspace, and therefore $\mathbf{F}^\top\mathbf{F}$ is the projection matrix to the subspace (Yang, 1995).[1]

A gradient descent minimization of such cost function can be approximately implemented by the subspace network (Yang, 1995), which, as pointed out above, requires nonlocal learning rules. While this algorithm can be implemented in a neural network using local learning rules, it requires a second layer of neurons (Oja, 1992), making it less appealing.

In this letter, we derive a single-layer network with local Hebbian and anti-Hebbian learning rules, similar in architecture to Földiak's (1989) (see Figure 1B), from a principled cost function and demonstrate that it recovers a principal subspace from streaming data. The novelty of our approach is that rather than starting with the representation error cost function traditionally used for dimensionality reduction, such as PCA, we use the cost function of classical multidimensional scaling (CMDS), a member of the family of multidimensional scaling (MDS) methods (Cox & Cox, 2000; Mardia, Kent, & Bibby, 1980). Whereas the connection between CMDS and PCA has been pointed out previously (Williams, 2001; Cox & Cox, 2000; Mardia et al., 1980), CMDS is typically performed in the batch setting. Instead, we developed a neural network implementation of CMDS for streaming data.

The rest of the letter is organized as follows. In section 2, by minimizing the CMDS cost function, we derive two online algorithms implementable by a single-layer network, with synchronous and asynchronous synaptic weight updates. In section 3, we demonstrate analytically that synaptic weights define a principal subspace whose dimension m is determined by the number of output neurons and that the stability of the solution requires that this subspace corresponds to top m principal components. In section 4, we show numerically that our algorithm recovers the principal subspace of a synthetic data set and does it faster than the existing algorithms. Finally, in section 5, we consider the case when data are generated by a nonstationary distribution and present a generalization of our algorithm to principal subspace tracking.

2 Derivation of Online Algorithms from the CMDS Cost Function

CMDS represents high-dimensional input data in a lower-dimensional output space while preserving pairwise similarities between samples (Young & Householder, 1938; Torgerson, 1952).[2] Let T centered input data samples in $\mathbb{R}^n$ be represented by column vectors $\mathbf{x}_{t=1,\ldots,T}$ concatenated into an

[1] Recall that in general, the projection matrix to the row space of a matrix $\mathbf{P}$ is given by $\mathbf{P}^\top\left(\mathbf{P}\mathbf{P}^\top\right)^{-1}\mathbf{P}$, provided $\mathbf{P}\mathbf{P}^\top$ is full rank (Plumbley, 1995). If the rows of $\mathbf{P}$ are orthonormal, this reduces to $\mathbf{P}^\top\mathbf{P}$.

[2] Whereas MDS in general starts with dissimilarities between samples that may not live in Euclidean geometry, in CMDS, data are assumed to have a Euclidean representation.

$n \times T$ matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$. The corresponding output representations in $\mathbb{R}^m$, $m \leq n$, are column vectors, $\mathbf{y}_{t=1,\ldots,T}$, concatenated into an $m \times T$-dimensional matrix $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_T]$. Similarities between vectors in Euclidean spaces are captured by their inner products. For the input (output) data, such inner products are assembled into a $T \times T$ Gram matrix $\mathbf{X}^\top\mathbf{X}$ ($\mathbf{Y}^\top\mathbf{Y}$).[3] For a given $\mathbf{X}$, CMDS finds $\mathbf{Y}$ by minimizing the so-called strain cost function (Carroll & Chang, 1972):

$$\min_{\mathbf{Y}} \|\mathbf{X}^\top\mathbf{X} - \mathbf{Y}^\top\mathbf{Y}\|_F^2. \tag{2.1}$$

For discovering a low-dimensional subspace, the CMDS cost function, equation 2.1, is a viable alternative to the representation error cost function, equation 1.3, because its solution is related to PCA (Williams, 2001; Cox & Cox, 2000; Mardia et al., 1980). Specifically, $\mathbf{Y}$ is the linear projection of $\mathbf{X}$ onto the (principal sub-)space spanned by m principal eigenvectors of the sample covariance matrix $\mathbf{C}_T = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t\mathbf{x}_t^\top = \mathbf{X}\mathbf{X}^\top$. The CMDS cost function defines a subspace rather than individual eigenvectors because left orthogonal rotations of an optimal $\mathbf{Y}$ stay in the subspace and are also optimal, as is evident from the symmetry of the cost function.

In order to reduce the dimensionality of streaming data, we minimize the CMDS cost function, equation 2.1, in the stochastic online setting. At time T, a data sample, $\mathbf{x}_T$, drawn independently from a zero-mean distribution is presented to the algorithm, which computes a corresponding output, $\mathbf{y}_T$, prior to the presentation of the next data sample. Whereas in the batch setting, each data sample affects all outputs, in the online setting, past outputs cannot be altered. Thus, at time T, the algorithm minimizes the cost depending on all inputs and ouputs up to time T with respect to $\mathbf{y}_T$ while keeping all the previous outputs fixed:

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T} \|\mathbf{X}^\top\mathbf{X} - \mathbf{Y}^\top\mathbf{Y}\|_F^2 = \arg\min_{\mathbf{y}_T} \sum_{t=1}^{T}\sum_{t'=1}^{T}(\mathbf{x}_t^\top\mathbf{x}_{t'} - \mathbf{y}_t^\top\mathbf{y}_{t'})^2, \tag{2.2}$$

where the last equality follows from the definition of the Frobenius norm. By keeping only the terms that depend on current output $\mathbf{y}_T$, we get

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T} \left[-4\mathbf{x}_T^\top\left(\sum_{t=1}^{T-1}\mathbf{x}_t\mathbf{y}_t^\top\right)\mathbf{y}_T + 2\mathbf{y}_T^\top\left(\sum_{t=1}^{T-1}\mathbf{y}_t\mathbf{y}_t^\top\right)\mathbf{y}_T - 2\|\mathbf{x}_T\|^2\|\mathbf{y}_T\|^2 + \|\mathbf{y}_T\|^4 \right]. \tag{2.3}$$

[3] When input data are pairwise Euclidean distances, assembled into a matrix $\mathbf{Q}$, the Gram matrix, $\mathbf{X}^\top\mathbf{X}$, can be constructed from $\mathbf{Q}$ by $\mathbf{HZH}$, where $Z_{ij} = -1/2Q_{ij}^2$, $\mathbf{H} = \mathbf{I}_n - (1/n)\mathbf{1}\mathbf{1}^\top$ is the centering matrix, $\mathbf{1}$ is the vector of n unitary components, and $\mathbf{I}_n$ is the n-dimensional identity matrix (Cox & Cox, 2000; Mardia et al., 1980).

In the large-T limit, expression 2.3 simplifies further because the first two terms grow linearly with T and therefore dominate over the last two. After dropping the last two terms, we arrive at

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T} \left[-4\mathbf{x}_T^\top \left(\sum_{t=1}^{T-1} \mathbf{x}_t \mathbf{y}_t^\top \right) \mathbf{y}_T + 2\mathbf{y}_T^\top \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{y}_t^\top \right) \mathbf{y}_T \right]. \tag{2.4}$$

We term the cost in expression 2.4 the online CMDS cost. Because the online CMDS cost is a positive semidefinite quadratic form in $\mathbf{y}_T$, this optimization problem is convex. While it admits a closed-form analytical solution via matrix inversion, we are interested in biologically plausible algorithms. Next, we consider two algorithms that can be mapped onto single-layer neural networks with local learning rules: coordinate descent leading to asynchronous updates and Jacobi iteration leading to synchronous updates.

2.1 A Neural Network with Asynchronous Updates. The online CMDS cost function, equation 2.4, can be minimized by coordinate descent, which at every step finds the optimal value of one component of $\mathbf{y}_T$ while keeping the rest fixed. The components can be cycled through in any order until the iteration converges to a fixed point. Such iteration is guaranteed to converge under very mild assumptions: diagonals of $\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{y}_t^\top$ have to be positive (Luo & Tseng, 1991), meaning that each output coordinate has produced at least one nonzero output before current time step T. This condition is almost always satisfied in practice.

The cost to be minimized at each coordinate descent step with respect to the ith channel's activity is

$$y_{T,i} = \arg\min_{y_{T,i}} \left[-4\mathbf{x}_T^\top \left(\sum_{t=1}^{T-1} \mathbf{x}_t \mathbf{y}_t^\top \right) \mathbf{y}_T + 2\mathbf{y}_T^\top \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{y}_t^\top \right) \mathbf{y}_T \right].$$

Keeping only those terms that depend on $y_{T,i}$ yields

$$y_{T,i} = \arg\min_{y_{T,i}} \left[-4 \sum_k x_{T,k} \left(\sum_{t=1}^{T-1} x_{t,k} y_{t,i} \right) y_{T,i} + 4 \sum_{j \neq i} y_{T,j} \left(\sum_{t=1}^{T-1} y_{t,j} y_{t,i} \right) y_{T,i} + 2 \left(\sum_{t=1}^{T-1} y_{t,i}^2 \right) y_{T,i}^2 \right].$$

By taking a derivative with respect to $y_{T,i}$ and setting it to zero, we arrive at the following closed-form solution:

$$y_{T,i} = \frac{\sum_k \left(\sum_{t=1}^{T-1} y_{t,i} x_{t,k}\right) x_{T,k}}{\sum_{t=1}^{T-1} y_{t,i}^2} - \frac{\sum_{j \neq i} \left(\sum_{t=1}^{T-1} y_{t,i} y_{t,j}\right) y_{T,j}}{\sum_{t=1}^{T-1} y_{t,i}^2}. \tag{2.5}$$

To implement this algorithm in a neural network, we denote normalized input-output and output-output covariances,

$$W_{T,ik} = \frac{\sum_{t=1}^{T-1} y_{t,i} x_{t,k}}{\sum_{t=1}^{T-1} y_{t,i}^2}, \quad M_{T,i,j \neq i} = \frac{\sum_{t=1}^{T-1} y_{t,i} y_{t,j}}{\sum_{t=1}^{T-1} y_{t,i}^2}, \quad M_{T,ii} = 0, \tag{2.6}$$

allowing us to rewrite the solution, equation 2.5, in a form suggestive of a linear neural network,

$$y_{T,i} \leftarrow \sum_{j=1}^{n} W_{T,ij} x_{T,j} - \sum_{j=1}^{m} M_{T,ij} y_{T,j}, \tag{2.7}$$

where $\mathbf{W}_T$ and $\mathbf{M}_T$ represent the synaptic weights of feedforward and lateral connections respectively (see Figure 1B).

Finally, to formulate a fully online algorithm, we rewrite equation 2.6 in a recursive form. This requires introducing a scalar variable $D_{T,i}$ representing the cumulative squared activity of a neuron i up to time $T-1$,

$$D_{T,i} = \sum_{t=1}^{T-1} y_{t,i}^2, \tag{2.8}$$

Then at each time point, T, after the output $\mathbf{y}_T$ is computed by the network, the following updates are performed:

$$\begin{aligned} D_{T+1,i} &\leftarrow D_{T,i} + y_{T,i}^2, \\ W_{T+1,ij} &\leftarrow W_{T,ij} + y_{T,i}(x_{T,j} - W_{T,ij} y_{T,i})/D_{T+1,i}, \\ M_{T+1,i,j \neq i} &\leftarrow M_{T,ij} + y_{T,i}(y_{T,j} - M_{T,ij} y_{T,i})/D_{T+1,i}. \end{aligned} \tag{2.9}$$

Equations 2.7 and 2.9 define a neural network algorithm that minimizes the online CMDS cost function, equation 2.4, for streaming data by alternating between two phases: neural activity dynamics and synaptic updates. After a data sample is presented at time T, in the neuronal activity phase, neuron activities are updated one by one (i.e., asynchronously; see equation 2.7) until the dynamics converges to a fixed point defined by the following

equation:

$$\mathbf{y}_T = \mathbf{W}_T\mathbf{x}_T - \mathbf{M}_T\mathbf{y}_T \quad \Rightarrow \quad \mathbf{y}_T = (\mathbf{I}_m + \mathbf{M}_T)^{-1}\mathbf{W}_T\mathbf{x}_T, \tag{2.10}$$

where $\mathbf{I}_m$ is the m-dimensional identity matrix.

In the second phase of the algorithm, synaptic weights are updated, according to a local Hebbian rule, equation 2.9, for feedforward connections and, according to a local anti-Hebbian rule (due to the minus sign in equation 2.7), for lateral connections. Interestingly, these updates have the same form as the single-neuron Oja's rule, equation 1.1 (Oja, 1982), except that the learning rate is not a free parameter but is determined by the cumulative neuronal activity $1/D_{T+1,i}$.[4] To the best of our knowledge, such a single-neuron rule (Hu, Towfic, Pehlevan, Genkin, & Chklovskii, 2013) has not been derived in the multineuron case. An alternative derivation of this algorithm is presented in section A.1 in the appendix.

Unlike the representation error cost function, equation 1.3, the CMDS cost function, equation 2.1, is formulated only in terms of input and output activity. Yet the minimization with respect to $\mathbf{Y}$ recovers feedforward and lateral synaptic weights.

2.2 A Neural Network with Synchronous Updates. Here, we present an alternative way to derive a neural network algorithm from the large-T limit of the online CMDS cost function, equation 2.4. By taking a derivative with respect to $\mathbf{y}_T$ and setting it to zero, we arrive at the following linear matrix equation:

$$\left(\sum_{t=1}^{T-1}\mathbf{y}_t\mathbf{y}_t^\top\right)\mathbf{y}_T = \left(\sum_{t=1}^{T-1}\mathbf{y}_t\mathbf{x}_t^\top\right)\mathbf{x}_T. \tag{2.11}$$

We solve this system of equations using Jacobi iteration (Strang, 2009) by first splitting the output covariance matrix that appears on the left side of equation 2.11 into its diagonal component $\mathbf{D}_T$ and the remainder $\mathbf{R}_T$,

$$\left(\sum_{t=1}^{T-1}\mathbf{y}_t\mathbf{y}_t^\top\right) = \mathbf{D}_T + \mathbf{R}_T,$$

[4] The single-neuron Oja's rule derived from the minimization of a least squares optimization cost function ends up with the identical learning rate (Diamantaras, 2002; Hu et al., 2013). Motivated by this fact, such learning rate has been argued to be optimal for the APEX network (Diamantaras & Kung, 1996; Diamantaras, 2002) and used by others (Yang, 1995).

where the ith diagonal element of $\mathbf{D}_T$, $D_{T,i} = \sum_{t=1}^{T-1} y_{t,i}^2$, as defined in equation 2.8. Then equation 2.11 is equivalent to

$$\mathbf{y}_T = \mathbf{D}_T^{-1} \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{x}_t^\top \right) \mathbf{x}_T - \mathbf{D}_T^{-1} \mathbf{R}_T \mathbf{y}_T.$$

Interestingly, the matrices obtained on the right side are algebraically equivalent to the feedforward and lateral synaptic weight matrices defined in equation 2.6:

$$\mathbf{W}_T = \mathbf{D}_T^{-1} \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{x}_t^\top \right) \quad \text{and} \quad \mathbf{M}_T = \mathbf{D}_T^{-1} \mathbf{R}_T. \tag{2.12}$$

Hence, the Jacobi iteration for solving equation 2.11,

$$\mathbf{y}_T \leftarrow \mathbf{W}_T \mathbf{x}_T - \mathbf{M}_T \mathbf{y}_T, \tag{2.13}$$

converges to the same fixed point as the coordinate descent, equation 2.10.

Iteration 2.13 is naturally implemented by the same single-layer linear neural network as for the asynchronous update, Figure 1B. For each stimulus presentation the network goes through two phases. In the first phase, iteration 2.13 is repeated until convergence. Unlike the coordinate descent algorithm, which updated the activity of neurons one after another, here, activities of all neurons are updated synchronously. In the second phase, synaptic weight matrices are updated according to the same rules as in the asynchronous update algorithm, equation 2.9.

Unlike the asynchronous update, equation 2.7, for which convergence is almost always guaranteed (Luo & Tseng, 1991), convergence of iteration 2.13 is guaranteed only when the spectral radius of $\mathbf{M}$ is less than 1 (Strang, 2009). Whereas we cannot prove that this condition is always met, the synchronous algorithm works well in practice. While in the rest of the letter, we consider only the asynchronous updates algorithm, our results hold for the synchronous updates algorithm provided it converges.

3 Stationary Synaptic Weights Define a Principal Subspace

What is the nature of the lower-dimensional representation found by our algorithm? In CMDS, outputs $y_{T,i}$ are the Euclidean coordinates in the principal subspace of the input vector $\mathbf{x}_T$ (Cox & Cox, 2000; Mardia et al., 1980). While our algorithm uses the same cost function as CMDS, the minimization is performed in the streaming, or online, setting. Therefore, we cannot take for granted that our algorithm will find the principal subspace of the input. In this section, we provide analytical evidence, by a stability analysis

in a stochastic setting, that our algorithm extracts the principal subspace of the input data and projects onto that subspace. We start by previewing our results and method.

Our algorithm performs a linear dimensionality reduction since the transformation between the input and the output is linear. This can be seen from the neural activity fixed point, equation 2.10, which we rewrite as

$$\mathbf{y}_T = \mathbf{F}_T \mathbf{x}_T, \tag{3.1}$$

where $\mathbf{F}_T$ is a matrix defined in terms of the synaptic weight matrices $\mathbf{W}_T$ and $\mathbf{M}_T$:

$$\mathbf{F}_T := \left(\mathbf{I}_m + \mathbf{M}_T\right)^{-1} \mathbf{W}_T. \tag{3.2}$$

Relation 3.1 shows that the linear filter of a neuron, which we term a neural filter, is the corresponding row of $\mathbf{F}_T$. The space that neural filters span, the row space of $\mathbf{F}_T$, is termed a filter space.

First, we prove that in the stationary state of our algorithm, neural filters are indeed orthonormal vectors (see section 3.2, theorem 1). Second, we demonstrate that the orthonormal filters form the basis of a space spanned by some m eigenvectors of the covariance of the inputs $\mathbf{C}$ (see section 3.3, theorem 2). Third, by analyzing linear perturbations around the stationary state, we find that stability requires these m eigenvectors to be the principal eigenvectors and therefore the filter space to be the principal subspace (see section 3.4, theorem 3).

These results show that even though our algorithm was derived starting from the CMDS cost function, equation 2.1, $\mathbf{F}_T$ converges to the optimal solution of the representation error cost function, equation 1.3. This correspondence suggests that $\mathbf{F}_T^\top \mathbf{F}_T$ is the algorithm's current estimate of the projection matrix to the principal subspace. Further, in equation 1.3, columns of $\mathbf{F}^\top$ are interpreted as data features. Then columns of $\mathbf{F}_T^\top$, or neural filters, are the algorithm's estimate of such features.

Rigorous stability analyses of PCA neural networks (Oja, 1982, 1992; Oja & Karhunen, 1985; Sanger, 1989; Hornik & Kuan, 1992; Plumbley, 1995) typically use the ODE method (Kushner & Clark, 1978). Using a theorem of stochastic approximation theory (Kushner & Clark, 1978), the convergence properties of the algorithm are determined using a corresponding deterministic differential equation.[5]

[5] Application of stochastic approximation theory to PCA neural networks depends on a set of mathematical assumptions. See Zufiria (2002) for a critique of the validity of these assumptions and an alternative approach to stability analysis.

Unfortunately the ODE method cannot be used for our network. While the method requires learning rates that depend only on time, in our network, learning rates ($1/D_{T+1,i}$) are activity dependent. Therefore we take a different approach. We directly work with the discrete-time system, assume convergence to a stationary state, to be defined below, and study the stability of the stationary state.

3.1 Preliminaries. We adopt a stochastic setting where the input to the network at each time point, $\mathbf{x}_t$, is an n-dimensional independent and identically distributed random vector with zero mean, $\langle \mathbf{x}_t \rangle = 0$, where brackets denote an average over the input distribution, and covariance $\mathbf{C} = \langle \mathbf{x}_t \mathbf{x}_t^\top \rangle$.

Our analysis is performed for the stationary state of synaptic weight updates; that is, when averaged over the distribution of input values, the updates on $\mathbf{W}$ and $\mathbf{M}$ average to zero. This is the point of convergence of our algorithm. For the rest of the section, we drop the time index T to denote stationary state variables.

The remaining dynamical variables, learning rates $1/D_{T+1,i}$, keep decreasing at each time step due to neural activity. We assume that the algorithm has run for a sufficiently long time such that the change in learning rate is small and it can be treated as a constant for a single update. Moreover, we assume that the algorithm converges to a stationary point sufficiently fast such that the following approximation is valid at large T,

$$\frac{1}{D_{T+1,i}} = \frac{1}{\sum_{t=1}^{T} y_{t,i}^2} \approx \frac{1}{T\langle y_i^2 \rangle},$$

where $\mathbf{y}$ is calculated with stationary state weight matrices.

We collect these assumptions into a definition:

Definition 1 *(Stationary State). In the stationary state,*

$$\langle \Delta W_{ij} \rangle = \langle \Delta M_{ij} \rangle = 0,$$

and

$$\frac{1}{D_i} = \frac{1}{T\langle y_i^2 \rangle},$$

with T large.

The stationary state assumption leads us to define various relations between synaptic weight matrices, summarized in the following corollary:

Corollary 1. *In the stationary state,*

$$\langle y_i x_j \rangle = \langle y_i^2 \rangle W_{ij}, \tag{3.3}$$

and

$$\langle y_i y_j \rangle = \langle y_i^2 \rangle (M_{ij} + \delta_{ij}), \tag{3.4}$$

where δ_{ij} is the Kronecker delta.

Proof. The stationarity assumption when applied to the update rule on **W**, equation 2.9, leads immediately to equation 3.3. The stationarity assumption applied to the update rule on **M**, equation 2.9, gives

$$\langle y_i y_j \rangle = \langle y_i^2 \rangle M_{ij}, \quad i \neq j.$$

The last equality does not hold for $i = j$ since diagonal elements of **M** are zero. To cover the case $i = j$, we add an identity matrix to **M**, and hence one recovers equation 3.4.

Remark. Note that equation 3.4 implies $\langle y_i^2 \rangle M_{ij} = \langle y_j^2 \rangle M_{ji}$—that lateral connection weights are not symmetrical.

3.2 Orthonormality of Neural Filters. Here we prove the orthonormality of neural filters in the stationary state. First, we need the following lemma:

Lemma 1. *In the stationary state, the following equality holds:*

$$\boldsymbol{I}_m + \boldsymbol{M} = \boldsymbol{W}\boldsymbol{F}^\top. \tag{3.5}$$

Proof. By equation 3.4, $\langle y_i^2 \rangle \left(M_{ik} + \delta_{ik}\right) = \langle y_i y_k \rangle$. Using $\mathbf{y} = \mathbf{Fx}$, we substitute for y_k on the right-hand side: $\langle y_i^2 \rangle \left(M_{ik} + \delta_{ik}\right) = \sum_j F_{kj} \langle y_i x_j \rangle$. Next, the stationarity condition, equation 3.3, yields $\langle y_i^2 \rangle \left(M_{ik} + \delta_{ik}\right) = \langle y_i^2 \rangle \sum_j F_{kj} W_{ij}$. Canceling $\langle y_i^2 \rangle$ on both sides proves the lemma.

Now we can prove our theorem:

Theorem 1. *In the stationary state, neural filters are orthonormal:*

$$\boldsymbol{F}\boldsymbol{F}^\top = \boldsymbol{I}_m. \tag{3.6}$$

Proof. First, we substitute for $\mathbf{F}$ (but not for $\mathbf{F}^\top$) its definition (see equation 3.2): $\mathbf{F}\mathbf{F}^\top = (\mathbf{I}_m + \mathbf{M})^{-1}\mathbf{W}\mathbf{F}^\top$. Next, using lemma 1, we substitute $\mathbf{W}\mathbf{F}^\top$ by $(\mathbf{I}_m + \mathbf{M})$. The right-hand side becomes $(\mathbf{I}_m + \mathbf{M})^{-1}(\mathbf{I}_m + \mathbf{M}) = \mathbf{I}_m$.

Remark. Theorem 1 implies that $\text{rank}(\mathbf{F}) = m$.

3.3 Neural Filters and Their Relationship to the Eigenspace of the Covariance Matrix. How is the filter space related to the input? We partially answer this question in theorem 2, using the following lemma:

Lemma 2. *In the stationary state, $\boldsymbol{F}^\top\boldsymbol{F}$ and $\boldsymbol{C}$ commute:*

$$\boldsymbol{F}^\top\boldsymbol{F}\boldsymbol{C} = \boldsymbol{C}\boldsymbol{F}^\top\boldsymbol{F}. \tag{3.7}$$

Proof. See section A.2.

Now we can state our second theorem.

Theorem 2. *At the stationary state state, the filter space is an m-dimensional subspace in $\mathbb{R}^n$ that is spanned by some m eigenvectors of the covariance matrix.*

Proof. Because $\mathbf{F}^\top\mathbf{F}$ and $\mathbf{C}$ commute (see lemma 2), they must share the same eigenvectors. Equation 3.6 of theorem 1 implies that m eigenvalues of $\mathbf{F}^\top\mathbf{F}$ are unity and the rest are zero. Eigenvectors associated with unit eigenvalues span the row space of $\mathbf{F}$ and are identical to some m eigenvectors of $\mathbf{C}$.[6]

Which m eigenvectors of $\mathbf{C}$ span the filter space? To show that these are the eigenvectors corresponding to the largest eigenvalues of $\mathbf{C}$, we perform a linear stability analysis around the stationary point and show that any other combination would be unstable.

3.4 Linear Stability Requires Neural Filters to Span a Principal Subspace. The strategy here is to perturb $\mathbf{F}$ from its equilibrium value and show that the perturbation is linearly stable only if the row space of $\mathbf{F}$ is the space spanned by the eigenvectors corresponding to the m highest eigenvalues of $\mathbf{C}$. To prove this result, we need two more lemmas.

Lemma 3. *Let $\boldsymbol{H}$ be an $m \times n$ real matrix with orthonormal rows and $\boldsymbol{G}$ an $(n - m) \times n$ real matrix with orthonormal rows, whose rows are chosen to be orthogonal to the rows of $\boldsymbol{H}$. Any $n \times m$ real matrix $\boldsymbol{Q}$ can be decomposed as*

$$\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{H} + \boldsymbol{S}\boldsymbol{H} + \boldsymbol{B}\boldsymbol{G},$$

[6] If this fact is not familiar, we recommend Strang's (2009) discussion of singular value decomposition.

where $\mathbf{A}$ is an $m \times m$ skew-symmetric matrix, $\mathbf{S}$ is an $m \times m$ symmetric matrix, and $\mathbf{B}$ is an $m \times (n-m)$ matrix.

Proof. Define $\mathbf{B} := \mathbf{Q}\mathbf{G}^\top$, $\mathbf{A} := \frac{1}{2}(\mathbf{Q}\mathbf{H}^\top - \mathbf{H}\mathbf{Q}^\top)$ and $\mathbf{S} := \frac{1}{2}(\mathbf{Q}\mathbf{H}^\top + \mathbf{H}\mathbf{Q}^\top)$. Then $\mathbf{A}\mathbf{H} + \mathbf{S}\mathbf{H} + \mathbf{B}\mathbf{G} = \mathbf{Q}(\mathbf{H}^\top\mathbf{H} + \mathbf{G}^\top\mathbf{G}) = \mathbf{Q}$.

We denote an arbitrary perturbation of $\mathbf{F}$ as $\delta\mathbf{F}$, where a small parameter is implied. We can use lemma 3 to decompose $\delta\mathbf{F}$ as

$$\delta\mathbf{F} = \delta\mathbf{A}\,\mathbf{F} + \delta\mathbf{S}\,\mathbf{F} + \delta\mathbf{B}\,\mathbf{G}, \tag{3.8}$$

where the rows of $\mathbf{G}$ are orthogonal to the rows of $\mathbf{F}$. Skew-symmetric $\delta\mathbf{A}$ corresponds to rotations of filters within the filter space; it keeps neural filters orthonormal. Symmetric $\delta\mathbf{S}$ keeps the filter space invariant but destroys orthonormality. $\delta\mathbf{B}$ is a perturbation that takes the neural filters outside the filter space.

Next, we calculate how $\delta\mathbf{F}$ evolves under the learning rule, $\langle\Delta\delta\mathbf{F}\rangle$.

Lemma 4. *A perturbation to the stationary state has the following evolution under the learning rule to linear order in perturbation and linear order in T^{-1}:*

$$\langle\Delta\delta F_{ij}\rangle = \frac{1}{T}\sum_k \frac{\left(I_m + M\right)^{-1}_{ik}}{\langle y_k^2\rangle}\left[\sum_l \delta F_{kl}C_{lj} - \sum_{lpr}\delta F_{kl}F_{rp}C_{lp}F_{rj} - \sum_{lpr}F_{kl}\delta F_{rp}C_{lp}F_{rj}\right] - \frac{1}{T}\delta F_{ij}. \tag{3.9}$$

Proof. The proof is provided in section A.3.

Now we can state our main result in the following theorem:

Theorem 3. *The stationary state of neuronal filters is stable, in large-T limit, only if the m-dimensional filter space is spanned by the eigenvectors of the covariance matrix corresponding to the m highest eigenvectors.*

Proof. The Full proof is given in section A.4. Here we sketch the proof.

To simplify our analysis, we choose a specific $\mathbf{G}$ in lemma 3 without losing generality. Let $\mathbf{v}^{1,\dots,n}$ be eigenvectors of $\mathbf{C}$ and $v^{1,\dots,n}$ be corresponding eigenvalues, labeled so that the first m eigenvectors span the row space of $\mathbf{F}$ (or filter space). We choose rows of $\mathbf{G}$ to be the remaining eigenvectors: $\mathbf{G}' := [\mathbf{v}^{m+1}, \dots, \mathbf{v}^n]$.

By extracting the evolution of components of $\delta\mathbf{F}$ from equation 3.9 using equation 3.8, we are ready to state the conditions under which perturbations of $\mathbf{F}$ are stable. Multiplying equation 3.9 on the right by $\mathbf{G}^\top$ gives the

evolution of $\delta\mathbf{B}$:

$$\langle \Delta\delta B_i^j \rangle = \sum_k P_{ik}^j \delta B_k^j \quad \text{where} \quad P_{ik}^j \equiv \frac{1}{T}\left(\frac{\left(\mathbf{I}_m + \mathbf{M}\right)_{ik}^{-1}}{\langle y_k^2 \rangle} v^{j+m} - \delta_{ik} \right).$$

Here we changed our notation to $\delta B_{kj} = \delta B_k^j$ to make it explicit that for each j, we have one matrix equation. These equations are stable when all eigenvalues of all $\mathbf{P}^j$ are negative, which requires, as shown in section A.4,

$$\left\{v^1, \ldots, v^m\right\} > \left\{v^{m+1}, \ldots, v^n\right\}.$$

This result proves that the perturbation is stable only if the filter space is identical to the space spanned by eigenvectors corresponding to the m highest eigenvalues of $\mathbf{C}$.

It remains to analyze the stability of $\delta\mathbf{A}$ and $\delta\mathbf{S}$ perturbations. Multiplying equation 3.9 on the right by $\mathbf{F}^\top$ gives

$$\langle \Delta\delta A_{ij} \rangle = 0 \quad \text{and} \quad \langle \Delta\delta S_{ij} \rangle = \frac{-2}{T}\delta S_{ij}.$$

$\delta\mathbf{A}$ perturbation, which rotates neural filters, does not decay. This behavior is inherently related to the discussed symmetry of the strain cost function, equation 2.1, with respect to left rotations of the $\mathbf{Y}$ matrix. Rotated $\mathbf{y}$ vectors are obtained from the input by rotated neural filters, and hence $\delta\mathbf{A}$ perturbation does not affect the cost. But $\delta\mathbf{S}$ destroys orthonormality, and these perturbations do decay, making the orthonormal solution stable.

To summarize our analysis, if the dynamics converges to a stationary state, neural filters form an orthonormal basis of the principal subspace.

4 Numerical Simulations of the Asynchronous Network

Here, we simulate the performance of the network with asynchronous updates, equations 2.7 and 2.9, on synthetic data. The data were generated by a colored gaussian process with an arbitrarily chosen "actual" covariance matrix. We choose the number of input channels, $n = 64$, and the number of output channels, $m = 4$. In the input data, the ratio of the power in the first four principal components to the power in the remaining 60 components was 0.54. $\mathbf{W}$ and $\mathbf{M}$ were initialized randomly, and the step size of synaptic updates was initialized to $1/D_{0,i} = 0.1$. The coordinate descent step is cycled over neurons until the magnitude of change in $\mathbf{y}_T$ in one cycle is less than 10^{-5} times the magnitude of $\mathbf{y}_T$.

We compared the performance of the asynchronous updates network, equation 2.7 and 2.9, with two previously proposed networks, APEX (Kung

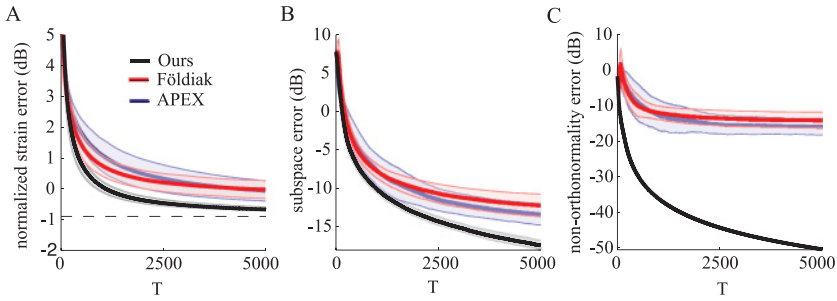

Figure 2: Performance of the asynchronous neural network compared with existing algorithms. Each algorithm was applied to 40 different random data sets drawn from the same gaussian statistics, described in text. Weight initializations were random. Solid lines indicate means, and shades indicate standard deviations across 40 runs. All errors are in decibels (dB). For formal metric definitions, see the text. (A) Strain error as a function of data presentations. The dotted line is the best error in batch setting, calculated using eigenvalues of the actual covariance matrix. (B) Subspace error as a function of data presentations. (C) Nonorthonormality error as a function of data presentations.

& Diamantaras, 1990; Kung et al., 1994) and Földiak's (1989), on the same data set (see Figure 2). The APEX network uses the same Hebbian and anti-Hebbian learning rules for synaptic weights, but the architecture is slightly different in that the lateral connection matrix, $\mathbf{M}$, is lower triangular. Földiak's network has the same architecture as ours (see Figure 1B) and the same learning rules for feedforward connections. However, the learning rule for lateral connections is $\Delta M_{ij} \propto y_i y_j$, unlike equation 2.9. For the sake of fairness, we applied the same adaptive step-size procedure for all networks. As in equation 2.9, the step size for each neuron i at time T was $1/D_{T+1,i}$, with $D_{T+1,i} = D_{T,i} + y_{T,i}^2$. In fact, such a learning rate has been recommended and argued to be optimal for the APEX network (Diamantaras & Kung, 1996; Diamantaras, 2002; see also note 4).

To quantify the performance of these algorithms, we used three different metrics. First is the strain cost function, equation 2.1, normalized by T^2 (see Figure 2A). Such a normalization is chosen because the minimum value of offline strain cost is equal to the power contained in the eigenmodes beyond the top m: $T^2 \sum_{k=m+1}^{n} (v^k)^2$, where $\{v^1, \ldots, v^n\}$ are eigenvalues of sample covariance matrix $\mathbf{C}_T$ (Cox & Cox, 2000; Mardia et al., 1980). For each of the three networks, as expected, the strain cost rapidly drops toward its lower bound. As our network was derived from the minimization of the strain cost function, it is not surprising that the cost drops faster than in the other two.

The second metric quantifies the deviation of the learned subspace from the actual principal subspace. At each T, the deviation is $\|\mathbf{F}_T^\top \mathbf{F}_T - \mathbf{V}^\top \mathbf{V}\|_F^2$,

where $\mathbf{V}$ is an $m \times n$ matrix whose rows are the principal eigenvectors, $\mathbf{V}^\top\mathbf{V}$ is the projection matrix to the principal subspace, $\mathbf{F}_T$ is defined the same way for APEX and Földiak networks as ours, and $\mathbf{F}_T^\top\mathbf{F}_T$ is the learned estimate of the projection matrix to the principal subspace. Such a deviation rapidly falls for each network, confirming that all three algorithms learn the principal subspace (see Figure 2B). Again, our algorithm extracts the principal subspace faster than the other two networks.

The third metric measures the degree of nonorthonormality among the computed neural filters. At each T, $\|\mathbf{F}_T\mathbf{F}_T^\top - \mathbf{I}_m\|_F^2$. The nonorthonormality error quickly drops for all networks, confirming that neural filters converge to orthonormal vectors (see Figure 2C). Yet again our network orthonormalizes neural filters much faster than the other two networks.

5 Subspace Tracking Using a Neural Network with Local Learning Rules

We have demonstrated that our network learns a linear subspace of streaming data generated by a stationary distribution. But what if the data are generated by an evolving distribution and we need to track the corresponding linear subspace? Using algorithm 2.9 would be suboptimal because the learning rate is adjusted to effectively "remember" the contribution of all the past data points.

A natural way to track an evolving subspace is to "forget" the contribution of older data points (Yang, 1995). In this section, we derive an algorithm with "forgetting" from a principled cost function where errors in the similarity of old data points are discounted:

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T} \sum_{t=1}^{T}\sum_{t'=1}^{T} \beta^{2T-t-t'}(\mathbf{x}_t^\top\mathbf{x}_{t'} - \mathbf{y}_t^\top\mathbf{y}_{t'})^2, \tag{5.1}$$

where β is a discounting factor $0 \le \beta \le 1$ with $\beta = 1$ corresponding to our original algorithm, equation 2.2. The effective timescale of forgetting is

$$\tau := -1/\ln\beta. \tag{5.2}$$

By introducing a $T \times T$-dimensional diagonal matrix β_T with diagonal elements $\beta_{T,ii} = \beta^{T-i}$ we can rewrite equation 5.1 in a matrix notation:

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T} \|\beta_T^\top\mathbf{X}^\top\mathbf{X}\beta_T - \beta_T^\top\mathbf{Y}^\top\mathbf{Y}\beta_T\|_F^2. \tag{5.3}$$

Yang (1995) used a similar discounting to derive subspace tracking algorithms from the representation error cost function, equation 1.3.

To derive an online algorithm to solve equation 5.3, we follow the same steps as before. By keeping only the terms that depend on current output $\mathbf{y}_T$ we get

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T}\left[-4\mathbf{x}_T^\top\left(\sum_{t=1}^{T-1}\beta^{2(T-t)}\mathbf{x}_t\mathbf{y}_t^\top\right)\mathbf{y}_T + 2\mathbf{y}_T^\top\left(\sum_{t=1}^{T-1}\beta^{2(T-t)}\mathbf{y}_t\mathbf{y}_t^\top\right)\mathbf{y}_T - 2\left\|\mathbf{x}_T\right\|^2\|\mathbf{y}_T\|^2 + \|\mathbf{y}_T\|^4\right]. \tag{5.4}$$

In equation 5.4, provided that past input-input and input-output outer products are not forgotten for a sufficiently long time (i.e., $\tau \gg 1$), the first two terms dominate over the last two for large T. After dropping the last two terms, we arrive at

$$\mathbf{y}_T = \arg\min_{\mathbf{y}_T}\left[-4\mathbf{x}_T^\top\left(\sum_{t=1}^{T-1}\beta^{2(T-t)}\mathbf{x}_t\mathbf{y}_t^\top\right)\mathbf{y}_T + 2\mathbf{y}_T^\top\left(\sum_{t=1}^{T-1}\beta^{2(T-t)}\mathbf{y}_t\mathbf{y}_t^\top\right)\mathbf{y}_T\right]. \tag{5.5}$$

As in the nondiscounted case, minimization of the discounted online CMDS cost function by coordinate descent, equation 5.5, leads to a neural network with asynchronous updates,

$$y_{T,i} \leftarrow \sum_{j=1}^{n} W_{T,ij}^{\beta} x_{T,j} - \sum_{j=1}^{m} M_{T,ij}^{\beta} y_{T,j}, \tag{5.6}$$

and by a Jacobi iteration to a neural network with synchronous updates,

$$\mathbf{y}_T \leftarrow \mathbf{W}_T^{\beta}\mathbf{x}_T - \mathbf{M}_T^{\beta}\mathbf{y}_T, \tag{5.7}$$

with synaptic weight matrices in both cases given by

$$W_{T,ij}^{\beta} = \frac{\sum_{t=1}^{T-1}\beta^{2(T-t)}y_{t,i}x_{t,j}}{\sum_{t=1}^{T-1}\beta^{2(T-t)}y_{t,i}^2}, \quad M_{T,i,j\neq i}^{\beta} = \frac{\sum_{t=1}^{T-1}\beta^{2(T-t)}y_{t,i}y_{t,j}}{\sum_{t=1}^{T-1}\beta^{2(T-t)}y_{t,i}^2},$$
$$M_{T,ii}^{\beta} = 0. \tag{5.8}$$

Finally, we rewrite equation 5.8 in a recursive form. As before, we introduce a scalar variable $D_{T,i}^{\beta}$ representing the discounted cumulative activity

of a neuron i up to time $T-1$,

$$D^{\beta}_{T,i} = \sum_{t=1}^{T-1} \beta^{2(T-t-1)} y^2_{t,i}. \tag{5.9}$$

Then the recursive updates are

$$\begin{aligned} D^{\beta}_{T+1,i} &\leftarrow \beta^2 D^{\beta}_{T,i} + y^2_{T,i}, \\ W^{\beta}_{T+1,ij} &\leftarrow W^{\beta}_{T,ij} + y_{T,i}(x_{T,j} - W^{\beta}_{T,ij} y_{T,i})/D^{\beta}_{T+1,i}, \\ M^{\beta}_{T+1,i,j\neq i} &\leftarrow M^{\beta}_{T,ij} + y_{T,i}(y_{T,j} - M^{\beta}_{T,ij} y_{T,i})/D^{\beta}_{T+1,i}. \end{aligned} \tag{5.10}$$

These updates are local and almost identical to the original updates, equation 2.9, except the $D^{\beta}_{T+1,i}$ update, where the past cumulative activity is discounted by β^2. For suitably chosen β, the learning rate, $1/D^{\beta}_{T+1,i}$, stays sufficiently large even at large T, allowing the algorithm to react to changes in data statistics.

As before, we have a two-phase algorithm for minimizing the discounted online CMDS cost function, equation 5.5. For each data presentation, first the neural network dynamics is run using equation 5.6 or 5.7, until the dynamics converges to a fixed point. In the second step, synaptic weights are updated using equation 5.10.

In Figure 3, we present the results of a numerical simulation of our subspace tracking algorithm with asynchronous updates similar to that in section 4 but for nonstationary synthetic data. The data are drawn from two different gaussian distributions: from $T = 1$ to $T = 2500$, with covariance $\mathbf{C}_1$, and from $T = 2501$ to $T = 5000$, with covariance $\mathbf{C}_2$. We ran our algorithm with four different β factors: $\beta = 0.998, 0.995, 0.99, 0.98$ ($\tau = 499.5, 199.5, 99.5, 49.5$).

We evaluate the subspace tracking performance of the algorithm using a modification of the subspace error metric introduced in section 4. From $T = 1$ to $T = 2500$, the error is $\|\mathbf{F}_T^\top \mathbf{F}_T - \mathbf{V}_1^\top \mathbf{V}_1\|_F^2$, where $\mathbf{V}_1$ is an $m \times n$ matrix whose rows are the principal eigenvectors of $\mathbf{C}_1$. From $T = 2501$ to $T = 5000$, the error is $\|\mathbf{F}_T^\top \mathbf{F}_T - \mathbf{V}_2^\top \mathbf{V}_2\|_F^2$, where $\mathbf{V}_2$ is an $m \times n$ matrix whose rows are the principal eigenvectors of $\mathbf{C}_2$. Figure 3A plots this modified subspace error. Initially the subspace error decreases, reaching lower values with higher β. Higher β allows for smaller learning rates, allowing a fine-tuning of the neural filters and hence lower error. At $T = 2501$, a sudden jump is observed corresponding to the change in principal subspace. The network rapidly corrects its neural filters to project to the new principal subspace, and the error falls to before jump values. It is interesting to note that higher β now leads to a slower decay due to extended memory in the past.

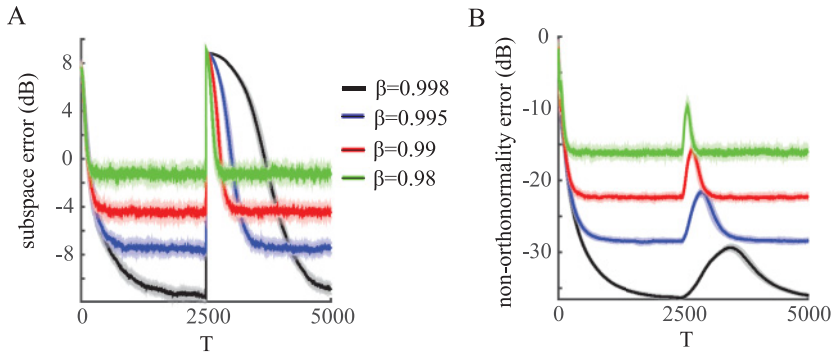

Figure 3: Performance of the subspace tracking asynchronous neural network with nonstationary data. The algorithm with different β factors was applied to 40 different random data sets drawn from the same nonstationary statistics, described in the text. Weight initializations were random. Solid lines indicate means, and shades indicate standard deviations. All errors are in decibels (dB). For formal metric definitions, see the text. (A) Subspace error as a function of data presentations. (B) Nonorthonormality error as a function of data presentations.

We also quantify the degree of nonorthonormality of neural filters using the nonorthonormality error defined in section 4. Initially the nonorthonormality error decreases, reaching lower values with higher β. Again, higher β allows for smaller learning rates, allowing a fine-tuning of the neural filters. At $T = 2501$, an increase in orthonormality error is observed as the network adjusts its neural filters. Then the error falls to before change values, with higher β leading to a slower decay due to extended memory in the past.

6 Discussion

In this letter, we made a step toward a mathematically rigorous model of neuronal dimensionality reduction satisfying more biological constraints than was previously possible. By starting with the CMDS cost function, equation 2.1, we derived a single-layer neural network of linear units using only local learning rules. Using a local stability analysis, we showed that our algorithm finds a set of orthonormal neural filters and projects the input data stream to its principal subspace. We showed that with a small modification in learning rate updates, the same algorithm performs subspace tracking.

Our algorithm finds the principal subspace but not necessarily the principal components themselves. This is not a weakness since both the representation error cost, equation 1.3, and CMDS cost, equation 2.1, are minimized

by projections to principal subspace and finding the principal components is not necessary.

Our network is most similar to Földiak's (1989) network, which learns feedforward weights by a Hebbian Oja rule and the all-to-all lateral weights by an anti-Hebbian rule. Yet the functional form of the anti-Hebbian learning rule in Földiak's network, $\Delta M_{ij} \propto y_i y_j$, is different from ours, equation 2.9, resulting in three interesting differences. First, because the synaptic weight update rules in Földiak's network are symmetric, if the weights are initialized symmetric (i.e., $M_{ij} = M_{ji}$), and learning rates are identical for lateral weights, they will stay symmetric. As mentioned above, such symmetry does not exist in our network (see equations 2.9 and 3.4). Second, while in Földiak's network neural filters need not be orthonormal (Földiak, 1989; Leen, 1991), in our network they will be (see theorem 1). Third, in Földiak's (1989) network, output units are decorrelated, since in its stationary state, $\langle y_i y_j \rangle = 0$. This need not be true in our network. Yet correlations among output units do not necessarily mean that information in the output about the input is reduced.[7]

Our network is similar to the APEX network (Kung & Diamantaras, 1990) in the functional form of both the feedforward and the lateral weights. However the network architecture is different because the APEX network has a lower-triangular lateral connectivity matrix. Such difference in architecture leads to two interesting differences in the APEX network operation (Diamantaras & Kung, 1996): (1) the outputs converge to the principal components, and (2) lateral weights decay to zero and neural filters are the feedforward weights. In our network, lateral weights do not have to decay to zero and neural filters depend on both the feedforward and lateral weights (see equation 3.2).

In numerical simulations, we observed that our network is faster than Földiak's and APEX networks in minimizing the strain error, finding the principal subspace and orthonormalizing neural filters. This result demonstrates the advantage of our principled approach compared to heuristic learning rules.

Our choice of coordinate descent to minimize the cost function in the activity dynamics phase allowed us to circumvent problems associated with matrix inversion: $\mathbf{y} \leftarrow (\mathbf{I}_m + \mathbf{M})^{-1}\mathbf{W}\mathbf{x}$. Matrix inversion causes problems for neural network implementations because it is a nonlocal operation. In

[7] As pointed out before (Linsker, 1988; Plumbley, 1993, 1995; Kung, 2014), PCA maximizes mutual information between a gaussian input, $\mathbf{x}$, and an output, $\mathbf{y} = \mathbf{F}\mathbf{x}$, such that rows of $\mathbf{F}$ have unit norms. When rows of $\mathbf{F}$ are principal eigenvectors, outputs are principal components and are uncorrelated. However, the output can be multiplied by a rotation matrix, $\mathbf{Q}$, and mutual information is unchanged, $\mathbf{y}' = \mathbf{Q}\mathbf{y} = \mathbf{Q}\mathbf{F}\mathbf{x}$. $\mathbf{y}'$ is now a correlated gaussian, and $\mathbf{Q}\mathbf{F}$ still has rows with unit norms. Therefore, one can have correlated outputs with maximal mutual information between input and output as long as rows of $\mathbf{F}$ span the principal subspace.

the absence of a cost function, Földiak (1989) suggested implementing matrix inversion by iterating $\mathbf{y} \leftarrow \mathbf{Wx} - \mathbf{My}$ until convergence. We derived a similar algorithm using Jacobi iteration. However, in general, such iterative schemes are not guaranteed to converge (Hornik & Kuan, 1992). Our coordinate descent algorithm is almost always guaranteed to converge because the cost function in the activity dynamics phase, equation 2.4, meets the criteria in Luo and Tseng (1991).

Unfortunately, our treatment still suffers from the problem common to most other biologically plausible neural networks (Hornik & Kuan, 1992): a complete global convergence analysis of synaptic weights is not yet available. Our stability analysis is local in the sense that it starts by assuming that the synaptic weight dynamics has reached a stationary state and then proves that perturbations around the stationary state are stable. We have not made a theoretical statement on whether this state can ever be reached or how fast such a state can be reached. Global convergence results using stochastic approximation theory are available for the single-neuron Oja rule (Oja & Karhunen, 1985), its nonlocal generalizations (Plumbley, 1995), and the APEX rule (Diamantaras & Kung, 1996); however, applicability of stochastic approximation theory was questioned recently (Zufiria, 2002). Although a neural network implementation is unknown, Warmuth and Kuzmin's (2008) online PCA algorithm stands out as the only algorithm for which a regret bound has been proved. An asymptotic dependence of regret on time can also be interpreted as convergence speed.

This letter also contributes to the MDS literature by applying the CMDS method to streaming data. However, our method has limitations in that to derive neural algorithms, we used the strain cost, equation 2.1, of CMDS. Such cost is formulated in terms of similarities, inner products to be exact, between pairs of data vectors and allowed us to consider a streaming setting where a data vector is revealed at a time. In the most general formulation of MDS, pairwise dissimilarities between data instances are given rather than data vectors themselves or similarities between them (Cox & Cox, 2000; Mardia et al., 1980). This generates two immediate problems for a generalization of our approach. First, a mapping to the strain cost function, equation 2.1, is possible only if the dissimilarites are Euclidean distances (see note 3). In general, dissimilarities do not need to be Euclidean or even metric distances (Cox & Cox, 2000; Mardia et al., 1980) and one cannot start from the strain cost, equation 2.1, for derivation of a neural algorithm. Second, in the streaming version of the general MDS setting, at each step, dissimilarities between the current and all past data instances are revealed, unlike our approach where the data vector itself is revealed. It is a challenging problem for future studies to find neural implementations in such generalized setting.

The online CMDS cost functions, equations 2.4 and 5.5, should also be valuable for subspace learning and tracking applications where biological plausibility is not a necessity. Minimization of such cost functions could

be performed much more efficiently in the absence of constraints imposed by biology.[8] It remains to be seen how the algorithms presented in this letter and their generalizations compare to state-of-the-art online subspace tracking algorithms from machine learning literature (Cichocki & Amari, 2002).

Finally, we believe that formulating the cost function in terms of similarities supports the possibility of representation-invariant computations in neural networks.

Appendix: Derivations and Proofs

A.1 Alternative Derivation of an Asynchronous Network. Here, we solve the system of equations 2.11 iteratively (Strang, 2009). First, we split the output covariance matrix that appears on the left-hand side of equation 2.11 into its diagonal component $\mathbf{D}_T$, a strictly upper triangular matrix $\mathbf{U}_T$, and a strictly lower triangular matrix $\mathbf{L}_T$:

$$\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{y}_t^\top = \mathbf{D}_T + \mathbf{U}_T + \mathbf{L}_T. \tag{A.1}$$

Substituting this into equation 2.11, we get

$$\left(\mathbf{D}_T + \omega \mathbf{L}_T\right) \mathbf{y}_T = \left((1-\omega)\,\mathbf{D}_T - \omega \mathbf{U}_T\right) \mathbf{y}_T + \omega \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{x}_t^\top\right) \mathbf{x}_T, \tag{A.2}$$

where ω is a parameter. We solve equation 2.11 by iterating

$$\mathbf{y}_T \longleftarrow \left(\mathbf{D}_T + \omega \mathbf{L}_T\right)^{-1} \left[\left((1-\omega)\,\mathbf{D}_T - \omega \mathbf{U}_T\right) \mathbf{y}_T + \omega \left(\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{x}_t^\top\right) \mathbf{x}_T\right], \tag{A.3}$$

until convergence. If symmetric $\sum_{t=1}^{T-1} \mathbf{y}_t \mathbf{y}_t^\top$ is positive definite, the convergence is guaranteed for $0 < \omega < 2$ by the Ostrowski-Reich theorem (Reich, 1949; Ostrowski, 1954). When $\omega = 1$, the iteration, equation A.3, corresponds to the Gauss-Seidel method, and, when $\omega > 1$, to the succesive overrelaxation method. The choice of ω for fastest convergence depends on

[8] For example, matrix equation 2.11 could be solved by a conjugate gradient descent method instead of iterative methods. Matrices that keep input-input and output-output correlations in equation 2.11 can be calculated recursively, leading to a truly online method.

the problem, and we will not explore this question here. However, values around 1.9 are generally recommended (Strang, 2009).

Because in equation A.2, the matrix multiplying $\mathbf{y}_T$ on the left is lower triangular and on the right is upper triangular, iteration A.3 can be performed component-by-component (Strang, 2009):

$$y_{T,i} \longleftarrow (1-\omega)\, y_{T,i} + \omega \frac{\sum_k \left(\sum_{t=1}^{T-1} y_{t,i} x_{t,k}\right) x_{T,k}}{\sum_{t=1}^{T-1} y_{t,i}^2} - \omega \frac{\sum_{j\neq i} \left(\sum_{t=1}^{T-1} y_{t,i} y_{t,j}\right) y_{T,j}}{\sum_{t=1}^{T-1} y_{t,i}^2}. \tag{A.4}$$

Note that $y_{T,i}$ is replaced with its new value before moving to the next component.

This algorithm can be implemented in a neural network,

$$y_{T,i} \leftarrow (1-\omega)\, y_{T,i} + \omega \sum_{j=1}^{n} W_{T,ij} x_{T,j} - \omega \sum_{j=1}^{m} M_{T,ij} y_{T,j}, \tag{A.5}$$

where $\mathbf{W}_T$ and $\mathbf{M}_T$, as defined in equation 2.6, represent the synaptic weights of feedforward and lateral connections, respectively. The case of $\omega < 1$ can be implemented by a leaky integrator neuron. The $\omega = 1$ case corresponds to our original asynchronous algorithm, except that now updates are performed in a particular order. For the $\omega > 1$ case, which may converge faster, we do not see a biologically plausible implementation since it requires self-inhibition.

Finally, to express the algorithm in a fully online form, we rewrite equation 2.6 via recursive updates, resulting in equation 2.9.

A.2 Proof of Lemma 2

Proof of Lemma 2. In our derivation below, we use results from equations 3.2, 3.3, and 3.4 of the main text.

$$\begin{aligned}
\left(\mathbf{F}^\top \mathbf{F}\mathbf{C}\right)_{ij} &= \sum_{kl} F_{ki} F_{kl} \langle x_l x_j \rangle \\
&= \sum_k F_{ki} \langle y_k x_j \rangle && \text{(from 3.2)} \\
&= \sum_k F_{ki} \langle y_k^2 \rangle W_{kj} && \text{(from 3.3)}
\end{aligned}$$

$$= \sum_{kp} F_{ki} \langle y_k^2 \rangle \left(M_{kp} + \delta_{kp} \right) F_{pj} \qquad \text{(from 3.2)}$$

$$= \sum_{kp} F_{ki} \langle y_p^2 \rangle \left(M_{pk} + \delta_{pk} \right) F_{pj} \qquad \text{(from 3.4)}$$

$$= \sum_{p} W_{pi} \langle y_p^2 \rangle F_{pj} \qquad \text{(from 3.2)}$$

$$= \sum_{p} \langle y_p x_i \rangle F_{pj} \qquad \text{(from 3.3)}$$

$$= \sum_{pk} F_{pk} \langle x_k x_i \rangle F_{pj} = \sum_{pk} \langle x_i x_k \rangle F_{pk} F_{pj} = \left(\mathbf{C}\mathbf{F}^\top \mathbf{F} \right)_{ij}. \qquad \text{(from 3.2)}$$

A.3 Proof of Lemma 4. Here we calculate how $\delta\mathbf{F}$ evolves under the learning rule, $\langle \Delta\delta\mathbf{F} \rangle$, and derive equation 3.9.

First, we introduce some new notation to simplify our expressions. We define lateral synaptic weight matrix $\mathbf{M}$ with diagonals set to 1 as

$$\hat{\mathbf{M}} := \mathbf{I}_m + \mathbf{M}. \qquad \text{(A.6)}$$

We use $\tilde{}$ to denote perturbed matrices

$$\begin{aligned} \tilde{\mathbf{F}} &:= \mathbf{F} + \delta\mathbf{F}, & \tilde{\mathbf{W}} &:= \mathbf{W} + \delta\mathbf{W}, \\ \tilde{\mathbf{M}} &:= \mathbf{M} + \delta\mathbf{M}, & \hat{\tilde{\mathbf{M}}} &:= \mathbf{I} + \tilde{\mathbf{M}} = \hat{\mathbf{M}} + \delta\mathbf{M}. \end{aligned} \qquad \text{(A.7)}$$

Note that when the network is run with these perturbed synaptic matrices, for input $\mathbf{x}$, the network dynamics will settle to the fixed point,

$$\tilde{\mathbf{y}} = \hat{\tilde{\mathbf{M}}}^{-1} \tilde{\mathbf{W}} \mathbf{x} = \tilde{\mathbf{F}} \mathbf{x}, \qquad \text{(A.8)}$$

which is different from the fixed point of the stationary network, $\mathbf{y} = \hat{\mathbf{M}}^{-1}\mathbf{W}\mathbf{x} = \mathbf{F}\mathbf{x}$.

Now we can prove lemma 4.

Proof of lemma 4. The proof has the following steps.

1. Since our update rules are formulated in terms of $\mathbf{W}$ and $\mathbf{M}$, it will be helpful to express $\delta\mathbf{F}$ in terms of $\delta\mathbf{W}$ and $\delta\mathbf{M}$. The definition of $\mathbf{F}$, equation 3.2, gives us the desired relation:

$$(\delta\hat{\mathbf{M}})\mathbf{F} + \hat{\mathbf{M}}(\delta\mathbf{F}) = \delta\mathbf{W}. \qquad \text{(A.9)}$$

2. We show that in the stationary state,

$$\langle \Delta\delta \mathbf{F} \rangle = \hat{\mathbf{M}}^{-1} \left(\langle \Delta\delta \mathbf{W} \rangle - \langle \Delta\delta \mathbf{M} \rangle \mathbf{F} \right) + \mathcal{O}\left(\frac{1}{T^2} \right). \tag{A.10}$$

Proof. Average changes due to synaptic updates on both sides of equation A.9 are equal: $\langle \Delta[(\delta\hat{\mathbf{M}})\mathbf{F} + \hat{\mathbf{M}}(\delta\mathbf{F})] \rangle = \langle \Delta\delta\mathbf{W} \rangle$. Noting that the unperturbed matrices are stationary, that is, $\langle \Delta\mathbf{M} \rangle = \langle \Delta\mathbf{F} \rangle = \langle \Delta\mathbf{W} \rangle = 0$, one gets $\langle \Delta\delta\mathbf{M} \rangle \mathbf{F} + \hat{\mathbf{M}} \langle \Delta\delta\mathbf{F} \rangle = \langle \Delta\delta\mathbf{W} \rangle + \mathcal{O}(T^{-2})$, from which equation A.10 follows.

3. We calculate $\langle \Delta\delta\mathbf{W} \rangle$ and $\langle \Delta\delta\mathbf{M} \rangle$ using the learning rule, in terms of matrices $\mathbf{W}$, $\mathbf{M}$, $\mathbf{C}$, $\mathbf{F}$, and $\delta\mathbf{F}$, and plug the result into equation A.10. This manipulation is going to give us the evolution of $\delta\mathbf{F}$ equation, 3.9.

   First, $\langle \Delta\delta\mathbf{W} \rangle$:

$$\begin{aligned}
\langle \Delta\delta W_{ij} \rangle &= \langle \Delta\tilde{W}_{ij} \rangle \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\langle \tilde{y}_i x_j \rangle - \langle \tilde{y}_i^2 \rangle \tilde{W}_{ij} \right) \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\sum_k \tilde{F}_{ik} \langle x_k x_j \rangle - \sum_{kl} \tilde{F}_{ik} \tilde{F}_{il} \langle x_k x_l \rangle \tilde{W}_{ij} \right) \quad \text{(from A.8)} \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\sum_k \tilde{F}_{ik} C_{kj} - \sum_{kl} \tilde{F}_{ik} \tilde{F}_{il} C_{kl} \tilde{W}_{ij} \right) \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\sum_k F_{ik} C_{kj} - \sum_{kl} F_{ik} F_{il} C_{kl} W_{ij} + \sum_k \delta F_{ik} C_{kj} \right. \\
&\quad \left. - 2 \sum_{kl} \delta F_{ik} F_{il} C_{kl} W_{ij} - \sum_{kl} F_{ik} F_{il} C_{kl} \delta W_{ij} \right) \quad \text{(from A.7)} \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\sum_k \delta F_{ik} C_{kj} - 2 \sum_{kl} \delta F_{ik} F_{il} C_{kl} W_{ij} \right. \\
&\quad \left. - \sum_{kl} F_{ik} F_{il} C_{kl} \delta W_{ij} \right). \quad \text{(from 3.3)}
\end{aligned}$$

   Next we calculate $\langle \Delta\delta\mathbf{M} \rangle$:

$$\begin{aligned}
\langle \Delta\delta M_{ij} \rangle &= \langle \Delta\tilde{\tilde{M}}_{ij} \rangle \\
&= \frac{1}{T\langle \tilde{y}_i^2 \rangle} \left(\langle \tilde{y}_i \tilde{y}_j \rangle - \langle \tilde{y}_i^2 \rangle \tilde{M}_{ij} \right) - \frac{1}{D_i} \delta_{ij} \langle \tilde{y}_i^2 \rangle
\end{aligned}$$

$$= \frac{1}{T\langle y_i^2\rangle}\left(\sum_{kl}\tilde{F}_{ik}\tilde{F}_{jl}\langle x_k x_l\rangle - \sum_{kl}\tilde{F}_{ik}\tilde{F}_{il}\langle x_k x_l\rangle\tilde{M}_{ij} - \delta_{ij}\sum_{kl}\tilde{F}_{ik}\tilde{F}_{il}\langle x_k x_l\rangle\right) \quad \text{(from A.8)}$$

$$= \frac{1}{T\langle y_i^2\rangle}\left(\sum_{kl}\tilde{F}_{ik}\tilde{F}_{jl}C_{kl} - \sum_{kl}\tilde{F}_{ik}\tilde{F}_{il}C_{kl}\tilde{M}_{ij} - \delta_{ij}\sum_{kl}\tilde{F}_{ik}\tilde{F}_{il}C_{kl}\right)$$

$$= \frac{1}{T\langle y_i^2\rangle}\left(\sum_{kl}F_{ik}F_{jl}C_{kl} - \sum_{kl}F_{ik}F_{il}C_{kl}M_{ij} - \delta_{ij}\sum_{kl}F_{ik}F_{il}C_{kl} + \sum_{kl}\delta F_{ik}F_{jl}C_{kl} + \sum_{kl}F_{ik}\delta F_{jl}C_{kl} - 2\sum_{kl}\delta F_{ik}F_{il}C_{kl}M_{ij} - \sum_{kl}F_{ik}F_{il}C_{kl}\delta M_{ij} - 2\delta_{ij}\sum_{kl}\delta F_{ik}F_{il}C_{kl}\right) \quad \text{(from A.7)}$$

$$= \frac{1}{T\langle y_i^2\rangle}\left(\sum_{kl}\delta F_{ik}F_{jl}C_{kl} + \sum_{kl}F_{ik}\delta F_{jl}C_{kl} - 2\sum_{kl}\delta F_{ik}F_{il}C_{kl}M_{ij} - \sum_{kl}F_{ik}F_{il}C_{kl}\delta M_{ij} - 2\delta_{ij}\sum_{kl}\delta F_{ik}F_{il}C_{kl}\right). \quad \text{(from 3.4)}$$

Plugging these in equation A.10, we get

$$\langle\Delta\delta F_{ij}\rangle = \sum_k \frac{\hat{M}^{-1}_{ik}}{T\langle y_k^2\rangle}\left[\sum_l \delta F_{kl}C_{lj} - 2\sum_{lp}\delta F_{kl}F_{kp}C_{lp}W_{kj} - \sum_{lp}F_{kl}F_{kp}C_{lp}\delta W_{kj} - \sum_{lpr}\delta F_{kl}F_{rp}C_{lp}F_{rj} - \sum_{lpr}F_{kl}\delta F_{rp}C_{lp}F_{rj} + 2\sum_{lpr}\delta F_{kl}F_{kp}C_{lp}M_{kr}F_{rj} + \sum_{lpr}F_{kl}F_{kp}C_{lp}\delta M_{kr}F_{rj} + 2\sum_{lpr}\delta_{kr}\delta F_{kl}F_{kp}C_{lp}F_{rj}\right] + \mathcal{O}\left(\frac{1}{T^2}\right).$$

M_{kr} and δM_{kr} terms can be eliminated using the previously derived relations, equations 3.2 and A.9. This leads to a cancellation of some

of the terms given above, and finally we have

$$\langle \Delta \delta F_{ij} \rangle = \sum_k \frac{\hat{M}^{-1}_{ik}}{T \langle y_k^2 \rangle} \left[\sum_l \delta F_{kl} C_{lj} - \sum_{lpr} \delta F_{kl} F_{rp} C_{lp} F_{rj} - \sum_{lpr} F_{kl} \delta F_{rp} C_{lp} F_{rj} - \sum_{lpr} F_{kl} F_{kp} C_{lp} \hat{M}_{kr} \delta F_{rj} \right] + \mathcal{O}\left(\frac{1}{T^2}\right).$$

To proceed further, we note that

$$\langle y_k^2 \rangle = (\mathbf{F}\mathbf{C}\mathbf{F}^\top)_{kk}, \tag{A.11}$$

which allows us to simplify the last term. Then we get our final result:

$$\langle \Delta \delta F_{ij} \rangle = \frac{1}{T} \sum_k \frac{\hat{M}^{-1}_{ik}}{\langle y_k^2 \rangle} \left[\sum_l \delta F_{kl} C_{lj} - \sum_{lpr} \delta F_{kl} F_{rp} C_{lp} F_{rj} - \sum_{lpr} F_{kl} \delta F_{rp} C_{lp} F_{rj} \right] - \frac{1}{T} \delta F_{ij} + \mathcal{O}\left(\frac{1}{T^2}\right).$$

A.4 Proof of Theorem 3. For ease of reference, we remind that in general $\delta\mathbf{F}$ can be written as in equation 3.8:

$$\delta\mathbf{F} = \delta\mathbf{A}\,\mathbf{F} + \delta\mathbf{S}\,\mathbf{F} + \delta\mathbf{B}\,\mathbf{G}.$$

Here, $\delta\mathbf{A}$ is an $m \times m$ skew symmetric matrix, $\delta\mathbf{S}$ is an $m \times m$ symmetric matrix, and $\delta\mathbf{B}$ is an $m \times (n-m)$ matrix. $\mathbf{G}$ is an $(n-m) \times n$ matrix with orthonormal rows. These rows are chosen to be orthogonal to the rows of $\mathbf{F}$. Let $\mathbf{v}^{1,\ldots,n}$ be the eigenvectors $\mathbf{C}$ and $v^{1,\ldots,n}$ be the corresponding eigenvalues. We label them such that $\mathbf{F}$ spans the same space as the space spanned by the first m eigenvectors. We choose rows of $\mathbf{G}$ to be the remaining eigenvectors: $\mathbf{G}^\top := [\mathbf{v}^{m+1}, \ldots, \mathbf{v}^n]$. Then, for future reference,

$$\mathbf{F}\mathbf{G}^\top = 0, \quad \mathbf{G}\mathbf{G}^\top = \mathbf{I}_{(n-m)}, \quad \text{and} \quad \sum_k C_{ik} G^\top_{kj} = \sum_k C_{ik} v_k^{j+m} = v^{j+m} G^\top_{ij}. \tag{A.12}$$

We also refer to the definition, equation A.6:

$$\hat{\mathbf{M}} := \mathbf{I}_m + \mathbf{M}.$$

Proof of Theorem 3. Below, we discuss the conditions under which perturbations of $\mathbf{F}$ are stable. We work to linear order in T^{-1} as stated in theorem 3.

We treat separately the evolution of $\delta\mathbf{A}$, $\delta\mathbf{S}$, and $\delta\mathbf{B}$ under a general perturbation $\delta\mathbf{F}$.

1. Stability of $\delta\mathbf{B}$
   1.1 Evolution of $\delta\mathbf{B}$ is given by

$$\langle \Delta\delta B_{ij} \rangle = \frac{1}{T} \sum_k \left(\frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle} v^{j+m} - \delta_{ik} \right) \delta B_{kj}. \tag{A.13}$$

Proof. Starting from equation 3.8 and using equation A.12,

$$\begin{aligned} \langle \Delta\delta B_{ij} \rangle &= \sum_k \langle \Delta\delta F_{ik} \rangle G_{kj}^{\top} \\ &= \frac{1}{T} \sum_k \frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle} \sum_{lp} \delta F_{kl} C_{lp} G_{jp} - \frac{1}{T} \delta B_{ij}. \end{aligned}$$

Here the last line results from equation A.12 applied to equation 3.9. We look at the first term again using equations A.12 and then 3.8:

$$\begin{aligned} \frac{1}{T} \sum_k \frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle} \sum_{lp} \delta F_{kl} C_{lp} G_{jp} &= \frac{1}{T} \sum_k \frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle} \sum_l \delta F_{kl} v^{j+m} G_{jl} \\ &= \frac{1}{T} \sum_k \frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle} v^{j+m} \delta B_{kj}. \end{aligned}$$

Combining these gives equation A.13.

   1.2 When is equation A.13 stable? Next, we show that stability requires

$$\{v^1, \ldots, v^m\} > \{v^{m+1}, \ldots, v^n\}.$$

For ease of manipulation, we express equation A.13 as a matrix equation for each column of $\delta\mathbf{B}$. For convenience, we change our notation to $\delta B_{kj} = \delta B_k^j$,

$$\langle \Delta\delta B_i^j \rangle = \sum_k P_{ik}^j \delta B_k^j$$

$$\text{where} \quad P_{ik}^j \equiv \frac{1}{T} \left(O_{ik} v^{j+m} - \delta_{ik} \right), \quad \text{and} \quad O_{ik} \equiv \frac{\hat{M}_{ik}^{-1}}{\langle y_k^2 \rangle}.$$

We have one matrix equation for each j. These equations are stable if all eigenvalues of all $\mathbf{P^j}$ are negative;

$$\{\mathrm{eig}(\mathbf{P})\} < 0 \quad \Rightarrow \quad \{\mathrm{eig}(\mathbf{O})\} < \frac{1}{v_j}, \quad j = m+1, \ldots, n.$$

$$\Rightarrow \quad \{\mathrm{eig}(\mathbf{O}^{-1})\} > v_j, \quad j = m+1, \ldots, n.$$

1.3 If one could calculate eigenvalues of $\mathbf{O}^{-1}$, the stability condition can be articulated. We start this calculation by noting that

$$\sum_k O_{ik}\langle y_k y_j\rangle = \sum_k \hat{M}^{-1}_{ik} \frac{\langle y_k y_j\rangle}{\langle y_k^2\rangle}$$

$$= \sum_k \hat{M}^{-1}_{ik}\hat{M}_{kj} = \delta_{ij}. \qquad \text{(from 3.4).} \quad \text{(A.14)}$$

Therefore,

$$\mathbf{O}^{-1} = \langle \mathbf{y}\mathbf{y}^\top\rangle = \mathbf{F}\mathbf{C}\mathbf{F}^\top. \tag{A.15}$$

Then we need to calculate the eigenvalues of $\mathbf{F}\mathbf{C}\mathbf{F}^\top$. They are

$$\mathrm{eig}(\mathbf{O}^{-1}) = \{v^1, \ldots, v^m\}.$$

Proof. We start with the eigenvalue equation:

$$\mathbf{F}\mathbf{C}\mathbf{F}^\top\boldsymbol{\lambda} = \lambda\boldsymbol{\lambda}.$$

Multiply both sides by $\mathbf{F}^\top$:

$$\mathbf{F}^\top\mathbf{F}\mathbf{C}\mathbf{F}^\top\boldsymbol{\lambda} = \lambda\left(\mathbf{F}^\top\boldsymbol{\lambda}\right).$$

Next, we use the commutation of $\mathbf{F}^\top\mathbf{F}$ and $\mathbf{C}$, equation 3.7, and the orthogonality of neural filters, $\mathbf{F}\mathbf{F}^\top = \mathbf{I}_m$, equation 3.6, to simplify the left-hand side:

$$\mathbf{F}^\top\mathbf{F}\mathbf{C}\mathbf{F}^\top\boldsymbol{\lambda} = \mathbf{C}\mathbf{F}^\top\mathbf{F}\mathbf{F}^\top\boldsymbol{\lambda} = \mathbf{C}\left(\mathbf{F}^\top\boldsymbol{\lambda}\right).$$

This implies that

$$\mathbf{C}(\mathbf{F}^\top\boldsymbol{\lambda}) = \lambda(\mathbf{F}^\top\boldsymbol{\lambda}). \tag{A.16}$$

Note that by the orthogonality of neural filters, the following is also true:

$$\mathbf{F}^\top\mathbf{F}(\mathbf{F}^\top\boldsymbol{\lambda}) = (\mathbf{F}^\top\boldsymbol{\lambda}). \tag{A.17}$$

All the relations above would hold true if $\lambda = 0$ and $(\mathbf{F}^\top\boldsymbol{\lambda}) = 0$, but this would require $\mathbf{F}(\mathbf{F}^\top\boldsymbol{\lambda}) = \boldsymbol{\lambda} = 0$, which is a

contradiction. Then equations A.16 and A.17 imply that $(\mathbf{F}^\top\boldsymbol{\lambda})$ is a shared eigenvector between $\mathbf{C}$ and $\mathbf{F}^\top\mathbf{F}$. $\mathbf{F}^\top\mathbf{F}$ and $\mathbf{C}$ were shown to commute before, and they share a complete set of eigenvectors. However, some $n-m$ eigenvectors of $\mathbf{C}$ have zero eigenvalues in $\mathbf{F}^\top\mathbf{F}$. We had labeled shared eigenvectors with unit eigenvalue in $\mathbf{F}^\top\mathbf{F}$ to be $\mathbf{v}^1, \ldots, \mathbf{v}^m$. The eigenvalue of $(\mathbf{F}^\top\boldsymbol{\lambda})$ with respect to $\mathbf{F}^\top\mathbf{F}$ is 1; therefore, $\mathbf{F}^\top\boldsymbol{\lambda}$ is one of $\mathbf{v}_1, \ldots, \mathbf{v}_m$. This proves that $\lambda = \{v^1, \ldots, v^m\}$ and

$$\mathrm{eig}(\mathbf{O}^{-1}) = \{v^1, \ldots, v^m\}.$$

1.4 From equation A.15, it follows that for stability

$$\{v^1, \ldots, v^m\} > \{v^{m+1}, \ldots, v^n\}.$$

2. Stability of $\delta\mathbf{A}$ and $\delta\mathbf{S}$. Next, we check the stabilities of $\delta\mathbf{A}$ and $\delta\mathbf{S}$:

$$\begin{aligned}
&\langle \Delta\delta A_{ij}\rangle + \langle \Delta\delta S_{ij}\rangle \\
&\quad = \sum_k \langle \Delta\delta F_{ik}\rangle F^T_{kj} \qquad \text{(from 3.8)} \\
&\quad = -\frac{1}{T}\sum_k \frac{\hat{M}^{-1}_{ik}}{\langle y_k^2\rangle}\sum_{lm} F_{kl}\delta F_{jm} C_{lm} - \frac{1}{T}(\delta A_{ij} + \delta S_{ij}) \\
&\quad = -\frac{1}{T}\sum_k \frac{\hat{M}^{-1}_{ik}}{\langle y_k^2\rangle}\sum_{l} (\mathbf{FCF}^T)_{kl}(\delta A^T_{lj} + \delta S^T_{lj}) - (\delta A_{ij} + \delta S_{ij}).
\end{aligned} \tag{A.18}$$

In deriving the last line, we used equations 3.8 and A.12. The k summation was calculated before equation A.14. Plugging this in equation A.18, one gets

$$\begin{aligned}
\langle \Delta\delta A_{ij}\rangle + \langle \Delta\delta S_{ij}\rangle &= -\frac{1}{T}(\delta A_{ij} + \delta A^T_{ij} + \delta S_{ij} + \delta S_{ij}) = \frac{-2}{T}\delta S_{ij} \\
&\Rightarrow \langle \Delta\delta A_{ij}\rangle = 0 \qquad \text{(from skew symmetry of } \mathbf{A}) \\
&\Rightarrow \langle \Delta\delta S_{ij}\rangle = \frac{-2}{T}\delta S_{ij}.
\end{aligned}$$

$\delta\mathbf{A}$ perturbation, which rotates neural filters to other orthonormal basis within the principal subspace, does not decay. On the other hand, $\delta\mathbf{S}$ destroys orthonormality and these perturbations do decay, making the orthonormal solution stable.

Collectively, the results above prove theorem 3.

A.5 Perturbation of the Stationary State due to Data Presentation. Our discussion of the linear stability of the stationary point assumed general perturbations. Perturbations that arise from data presentation,

$$\delta \mathbf{F} = \Delta \mathbf{F}, \tag{A.19}$$

form a restricted class of the most general case and have special consequences. Focusing on this case, we show that data presentations do not rotate the basis for extracted subspace in the stationary state.

We calculate perturbations within the extracted subspace. Using equations 3.8 and A.12,

$$\begin{aligned}
\delta \mathbf{A} + \delta \mathbf{S} &= \delta \mathbf{F}\, \mathbf{F}^\top \\
&= \Delta \mathbf{F}\, \mathbf{F}^\top && \text{(from A.19)} \\
&= \hat{\mathbf{M}}^{-1}(\Delta \mathbf{W} - \Delta \hat{\mathbf{M}}\, \mathbf{F})\mathbf{F}^\top && \text{(expand 3.2 to first order in } \Delta) \\
&= \hat{\mathbf{M}}^{-1}(\Delta \mathbf{W}\, \mathbf{F}^\top - \Delta \hat{\mathbf{M}}). && \text{(from 3.6)}
\end{aligned} \tag{A.20}$$

We look at $\Delta \mathbf{W}\, \mathbf{F}^\top$ term more closely:

$$\begin{aligned}
(\Delta \mathbf{W}\, \mathbf{F}^\top)_{ij} &= \sum_k \eta_i (y_i x_k - y_i^2 W_{ik}) F_{kj}^\top \\
&= \eta_i \left(y_i \sum_k F_{jk} x_k - y_i^2 \sum_k W_{ik} F_{kj}^\top \right) \\
&= \eta_i (y_i y_k - y_i^2 \hat{M}_{ij}) \\
&= \Delta \hat{M}_{ij}.
\end{aligned}$$

Plugging this back into equation A.20 gives

$$\delta \mathbf{A} + \delta \mathbf{S} = 0, \quad \Rightarrow \quad \delta \mathbf{A} = 0, \quad \& \quad \delta \mathbf{S} = 0. \tag{A.21}$$

Therefore, perturbations that arise from data presentation do not rotate neural filter basis within the extracted subspace. This property should increase the stability of the neural filter basis within the extracted subspace.

Acknowledgments

We are grateful to L. Greengard, S. Seung, and M. Warmuth for helpful discussions.

References

Arora, R., Cotter, A., Livescu, K., & Srebro, N. (2012). Stochastic optimization for PCA and PLS. In *Proceedings of the Allerton Conference on Communication, Control, and Computing* (pp. 861–868). Piscataway, NJ: IEEE.

Balzano, L. K. (2012). *Handling missing data in high-dimensional subspace modeling* Doctoral dissertation, University of Wisconsin–Madison.

Becker, S., & Plumbley, M. (1996). Unsupervised neural network learning procedures for feature extraction and classification. *Appl. Intell.*, *6*(3), 185–203.

Carroll, J., & Chang, J. (1972). *IDIOSCAL (individual differences in orientation scaling): A generalization of INDSCAL allowing idiosyncratic reference systems as well as an analytic approximation to INDSCAL.* Paper presented at the Psychometric Meeting, Princeton, NJ.

Cichocki, A., & Amari, S.-I. (2002). *Adaptive blind signal and image processing*. Hoboken, NJ: Wiley.

Cox, T., & Cox, M. (2000). *Multidimensional scaling*. Boca Raton, FL: CRC Press.

Crammer, K. (2006). Online tracking of linear subspaces. In G. Lugois & H. U. Simon (Eds.), *Learning theory* (pp. 438–452). New York: Springer.

Diamantaras, K. (2002). Neural networks and principal component analysis. In Y.-H. Hu & J.-N. Hwang (Eds.), *Handbook of neural networks in signal processing*. Boca Raton, FL: CRC Press.

Diamantaras, K., & Kung, S. (1996). *Principal component neural networks: Theory and applications*. Hoboken, NJ: Wiley.

Földiak, P. (1989). Adaptive network for optimal linear feature extraction. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 401–405). Piscataway, NJ: IEEE.

Goes, J., Zhang, T., Arora, R., & Lerman, G. (2014). Robust stochastic principal component analysis. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics* (pp. 266–274). http://jmlr.org/proceedings/papers/v33/

Hornik, K., & Kuan, C.-M. (1992). Convergence analysis of local feature extraction algorithms. *Neural Networks*, *5*, 229–240.

Hu, T., Towfic, Z., Pehlevan, C., Genkin, A., & Chklovskii, D. (2013). A neuron as a signal processing device. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers* (pp. 362–366). Piscataway, NJ: IEEE.

Hubel, D. H. (1995). *Eye, brain, and vision.* New York: Scientific American Library/Scientific American Books.

Hyvärinen, A., Hurri, J., & Hoyer, P. O. (2009). *Natural image statistics: A probabilistic approach to early computational vision.* New York: Springer.

Karhunen, J., & Oja, E. (1982). New methods for stochastic approximation of truncated Karhunen-Loeve expansions. In *Proc. 6th Int. Conf. on Pattern Recognition* (pp. 550–553). New York: Springer-Verlag.

Kung, S.-Y. (2014). *Kernel methods and machine learning*. Cambridge: Cambridge University Press.

Kung, S., & Diamantaras, K. (1990). A neural network learning algorithm for adaptive principal component extraction (apex). In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing* (pp. 861–864). Piscataway, NJ: IEEE.

Kung, S.-Y., Diamantaras, K., & Taur, J.-S. (1994). Adaptive principal component extraction (APEX) and applications. *IEEE T Signal Proces.*, *42*, 1202–1217.

Kushner H. J., & Clark D. S. (1978). *Stochastic approximation methods for constrained and unconstrained systems*. New York: Springer.

Leen, T. K. (1990). Dynamics of learning in recurrent feature-discovery networks. In D. Touretzky & R. Lippmann (Eds.), *Advances in neural information processing systems, 3* (pp. 70–76). San Mateo, CA: Morgan Kaufmann.

Leen, T. K. (1991). Dynamics of learning in linear feature-discovery networks. *Network*, *2*(1), 85–105.

Linsker, R. (1988). Self-organization in a perceptual network. *IEEE Computer*, *21*, 105–117.

Luo, Z. Q., & Tseng, P. (1991). On the convergence of a matrix splitting algorithm for the symmetric monotone linear complementarity problem. *SIAM J. Control Optim.*, *29*, 1037–1060.

Mardia, K., Kent, J., & Bibby, J. (1980). *Multivariate analysis*. Orlando, FL: Academic Press.

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *J. Math. Biol.*, *15*, 267–273.

Oja, E. (1992). Principal components, minor components, and linear neural networks. *Neural Networks*, *5*, 927–935.

Oja, E., & Karhunen, J. (1985). On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *J. Math. Anal. Appl.*, *106*(1), 69–84.

Ostrowksi A. M., (1954). On the linear iteration procedures for symmetric matrices. *Rend. Mat. Appl.*, *14*, 140–163.

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philos Mag.*, *2*, 559–572.

Plumbley, M. D. (1993). A Hebbian/anti-Hebbian network which optimizes information capacity by orthonormalizing the principal subspace. In *Proceedings of the International Conference on Artificial Neural Networks* (pp. 86–90). Piscataway, NJ: IEEE.

Plumbley, M. D. (1995). Lyapunov functions for convergence of principal component algorithms. *Neural Networks*, *8*(1), 11–23.

Preisendorfer, R., & Mobley, C. (1988). *Principal component analysis in meteorology and oceanography*. New York: Elsevier Science.

Reich, E. (1949). On the convergence of the classical iterative procedures for symmetric matrices. *Ann. Math. Statistics*, *20*, 448–451.

Rubner, J., & Schulten, K. (1990). Development of feature detectors by self-organization. *Biol. Cybern.*, *62*, 193–199.

Rubner, J., & Tavan, P. (1989). A self-organizing network for principal-component analysis. *Europhysics Letters*, *10*(7), 693.

Sanger, T. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, *2*(6), 459–473.

Shepherd, G. (2003). *The synaptic organization of the brain*. New York: Oxford University Press.

Strang, G. (2009). *Introduction to linear algebra*. Wellesley, MA: Wellesley-Cambridge Press

Torgerson, W. (1952). Multidimensional scaling: I. Theory and method. *Psychometrika*, *17*, 401–419.

Warmuth, M., & Kuzmin, D. (2008). Randomized online PCA algorithms with regret bounds that are logarithmic in the dimension. *J. Mach. Learn. Res.*, *9*(10), 2287–2320.

Williams, C. (2001). On a connection between kernel PCA and metric multidimensional scaling. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems, 13* (pp. 675–681). Cambridge, MA: MIT Press.

Yang, B. (1995). Projection approximation subspace tracking. *IEEE T Signal Proces.*, *43*, 95–107.

Young, G., & Householder, A. (1938). Discussion of a set of points in terms of their mutual distances. *Psychometrika*, *3*(1), 19–22.

Zufiria, P. J. (2002). On the discrete-time dynamics of the basic Hebbian neural network node. *IEEE Trans. Neural Netw.*, *13*, 1342–1352.

Received September 24, 2014; accepted February 28, 2015.