Dynamic covariance estimation via positive semidefinite dictionary learning

Clara Fannjiang (clarafj@stanford.edu)

February 26, 2016

1 Introduction

Time series are among the most prevalent forms of data we encounter in the real world, whether it be a sequence of audio, biomedical, demographic, climate, or financial measurements. Covariance estimation, where we try to estimate the covariance matrix of a time series, is a classic and important problem that can provide us predictive power in many applications.

The vast majority of covariance estimation methods assume that time series are *stationary*, *i.e.*, the covariance matrix remains unchanged throughout the lifetime of the variables. However, in reality the covariance of most real time series changes over time—for example, the covariance of stock prices is conceivably different in periods of market growth and market decline. We could compute sample covariance matrices over successive windows of the time series, but the resulting sequence of sample covariance matrices offers no insight into what particular features and structures of the covariance are evolving over time. Given this sequence of sample covariance matrices, how can we learn a meaningful, compact parameterization of how the covariance is changing?

We take inspiration from *dictionary learning*, an approach where dictionary *atoms* $D_i \in \mathbb{R}^n$, $i = 1, \ldots, K$ for some small K, are learned via some optimization algorithm such that each of $N \gg K$ samples $x_j \in \mathbb{R}^n$, $j = 1, \ldots, N$, can be expressed as a linear combination of the dictionary atoms. Dictionary learning simultaneously 1) compresses the dataset since we only need the dictionary $D = [D_1 \ \ldots D_K]$ and the linear combination coefficients to restore it, and 2) provides a model for understanding the data, by parameterizing it in terms of the few "fundamental" atoms D_i. Each atom can be thought of as capturing a basic prototype of how the variables covary.

1.1 Problem formulation

We formalize the optimization problem as follows. The input is a sequence of sample covariance matrices computed from successive windows of a time series, each vectorized in column-major order such that we can form an input matrix $X = [x_1 \ \ldots \ x_N] \in \mathbb{R}^{n^2 \times N}$ where

x_j is the vectorized sample covariance matrix from the j-th time window. (We choose this format to deal with a matrix instead of an $n \times n \times N$ tensor.) We wish to solve the following optimization problem:

$$\begin{array}{ll} \underset{D,W}{\text{minimize}} & (1/2)\|X - DW\|_F^2 \\ \text{subject to} & D_i \in \mathbb{S}_{\text{vec}}^+, i = 1, \ldots, K \\ & W_{ij} \geq 0, i = 1, \ldots, K, j = 1, \ldots N \end{array} \tag{1}$$

where the j-th column of W gives the coefficients or *weights* for reconstructing x_j as a linear combination of dictionary atoms; the constraint $D_i \in \mathbb{S}_{col}^+$ ensures each dictionary atom is the vectorized version of a positive semidefinite matrix, *i.e.*, a legitimate covariance matrix; and the non-negativity constraint on the weights was added for interpretability. If the weights are non-negative, it becomes straightforward to understand how much each dictionary atom contributed to each sample covariance matrix.

When the individual variances of the variables are not important to us, we also consider dictionary learning over sequences of correlation matrices. Given a covariance matrix Y, the (i, j)-th entry of the corresponding correlation matrix is simply $Y_{ij}/Y_{ii}Y_{jj}$, or the covariance of the i-th and j-th variables normalized by their variances. We then replace the $\mathbb{S}_{\text{vec}}^+$ in the constraints with $\mathbb{U}_{\text{vec}}$, the set of vectorized versions of all positive semidefinite matrices whose diagonals are 1 and whose off-diagonals fall in $[-1, 1]$.

1.2 Roadmap

The optimization problem is non-convex in (D, W), so there is no general-purpose algorithm that is guaranteed to discover the true minimum in finite time. In Sections 3-6, we consider four different algorithms for solving (1): alternating least-squares (ALS), projected gradient method (PGM), Douglas-Rachford splitting (DR), and alternating directions method of multipliers (ADMM). Sections 7 and 8 describe how we initialize and evaluate the algorithms, respectively, and Section 9 compares the performance and runtime of ALS and ADMM (comparisons with PGM and DR, as well as convergence analysis, are in the works). We will then demonstrate the better algorithm on neural development data in Section 10, to showcase how our approach exposes the fundamental prototypes of the neurons' covariance and how the neurons' activity transitions from one to the other.

2 Related work

Sra and Cherian [9] propose stochastic projected gradient method for a similar problem with an additional sparsity regularization. They then apply the results to the problem of nearest-neighbor retrieval. However, [5] found that directly applying projected gradient method to the related problem of nonnegative matrix factorization performed far worse than alternating least squares, which encouraged us to focus on ALS instead.

Harandi et al. [3] propose using Bregman divergences to solve the problem, an approach more similar in flavor to ADMM, though they also added sparsity regularization and did

not incorporate non-negativity. However, neither [9] nor [3] provided runtimes for their algorithms, which encouraged us to consider multiple algorithms and compare their speeds and accuracies.

3 Alternating least squares (ALS)

One natural way to solve (1) is to leverage its biconvexity: for fixed D the problem is convex in W, and vice versa. Let $f(D, W) = \|X - DW\|_F^2$ denote the objective. We can alternatively fix the value of the dictionary D and optimize $f(D, W)$ with respect to W, subject to the original constraints on W, then fix the value of the weights W and optimize $f(D, W)$ with respect to D, subject to the original constraints on D. Alternating between these two optimization subproblems gives a sequence of iterates (D^k, W^k) such that $f(D^k, W^{k+1}) \leq f(D^k, W^k)$ and $f(D^{k+1}, W^{k+1}) \leq f(D^k, W^{k+1})$. Looking into the two subproblems, we want to solve

$$\begin{array}{ll} \underset{W}{\text{minimize}} & (1/2)\|X - D^k W\|_F^2 \\ \text{subject to} & W_{jl} \geq 0, j = 1, \ldots, K, l = 1, \ldots N \end{array} \tag{2}$$

for W^{k+1}, and

$$\begin{array}{ll} \underset{D}{\text{minimize}} & (1/2)\|X - DW^{k+1}\|_F^2 \\ \text{subject to} & D_i \in \mathbb{S}_{\text{vec}}^+, i = 1, \ldots, K \end{array} \tag{3}$$

for D^{k+1}, both of which are constrained least-squares problems.

3.1 Nonnegative least-squares subproblem

To solve (2), we follow Lin [5], who solves the same subproblem using projected gradient method (see Alg. 4 in [5]). After each step along the negative gradient direction $-\nabla_W f(D^k, W) = (D^k)^T(D^k W - X)$, the algorithm projects to the feasible set by clipping all negative values of W to zero. We iterate until the norm of the projected gradient, or the difference between the previous iterate and the projection after a gradient step, falls below the threshold $\epsilon \nabla_W f(D^k, W^k)$ for some ϵ. (Recall that W^k is the previous iterate of ALS, not the previous iterate of the subproblem (2)). In bound-constrained optimization, this condition checks if the current point is near a stationary point [6].

3.2 Positive semidefinite least-squares subproblem

We treat subproblem (3) similarly, using projected gradient method. By transposing the variables within the objective, we see that the gradient with respect to D is $\nabla_D f(D, W^{k+1}) = (DW^{k+1} - X)(W^{k+1})^T$. To project onto the feasible set, we iterate through each column of D, reform it into a square matrix Y, project Y onto the closest positive semidefinite matrix, then re-vectorize Y. The projection has a closed-form solution: assuming we initialize D such that all columns are vectorized symmetric matrices, the gradient step preserves the

symmetry and we can compute the eigendecomposition $Y = Q\Lambda Q^T$, where the columns of Q are orthonormal eigenvectors and Λ is a diagonal matrix whose diagonal entries are eigenvalues in decreasing order. Let l be the index of the last nonnegative eigenvalue, *i.e.*, $\Lambda_{ii} < 0$ for $i > l$. By spectral theory, the closest positive semidefinite matrix Y_{psd} is given by

$$Y_{\text{psd}} = \sum_{i=1}^{l} \Lambda_{ii} Q_i (Q_i)^T. \tag{4}$$

As with subproblem (2), we iterate until the norm of the projected gradient fall below the threshold $\epsilon \nabla_D f(D^k, W^k)$.

We terminate ALS when the norm of the projected gradient of the original problem (1), which is the concatenation of the projected gradients of subproblems (2) and (3), falls below the threshold $\epsilon \nabla f(D^k, W^k)$.

3.3 Hyperparameters

To search for a good gradient step size, we use the trick in Alg. 4 of [5], which involves a hyperparameter $0 < \beta < 1$ for updating the step size. We have β_{nls} for the nonnegative least-squares subproblem, and β_{psdls} for the positive semidefinite least-squares subproblem.

3.4 Projection onto nearest correlation matrix

When we consider dictionary learning on correlation rather than covariance matrices, the analogous projection onto $\mathcal{U}_{\text{vec}}$ unfortunately has no closed-form solution. Instead, we compute the projection using Dykstra's alternating projections (see Alg 3.3 in [4]).

4 Projected gradient method (PGM)

We can also apply the projected gradient method directly to the original problem (1), using the gradients $\nabla_W f(D, W)$, $\nabla_D f(D, W)$, and projections to the feasible set described in Sections 3.1 and 3.2. The trick in Alg. 4 of [5] for finding a step size gives us the single hyperparameter $0 < \beta_{\text{pgm}} < 1$.

5 Douglas-Rachford (DR)

Douglas-Rachford splitting is a method for solving problems of the form

$$\text{minimize } f(x) + g(x)$$

using the proximal operator, a generalization of the projection operator (see [8] and [7]). The appeal of DR is that it requires only very minimal conditions on f and g, *e.g*, it can handle the indicator $g(D, W)$ in (5) and other non-differentiable, non-smooth functions.

We reformulate (1) as follows:

$$\underset{D,W}{\text{minimize}} \quad (1/2)\|X - DW\|_F^2 + g(D, W) \tag{5}$$

where

$$g(D,W) = \begin{cases} 0, & D_i \in \mathbb{S}^+_{\text{vec}}, i = 1, \ldots, K, \\ & W_{ij} \geq 0, i = 1, \ldots, K, j = 1, \ldots N \\ \infty, & \text{otherwise} \end{cases}$$

5.1 Proximal operators

To derive the proximal operators for DR, we first note that the proximal operator simplifies to the projection operator when the function is an indicator over a feasible set. We therefore use the projections described in Sections 3.1 and 3.2 to compute the proximal operator of $g(D, W)$.

We now consider the proximal operator of $f(D, W) = (\lambda/2)\|X - DW\|_F^2$. Given dictionary and weights D and W, we want to solve

$$\underset{Y,Z}{\text{minimize}} \quad (\gamma/2)\|X - YZ\|_F^2 + (1/2)\|Y - D\|_F^2 + (1/2)\|Z - W\|_F^2,$$

a problem that is biconvex in (Y, Z). Fixing either produces a least-squares problem, so we employ alternating least-squares using the following iterates:

$$\begin{aligned} Y^k &= (\gamma X (Z^{k-1})^T + D)(\gamma Z^{k-1}(Z^{k-1})^T + I)^{-1} \\ Z^k &= (\gamma (Y^k)^T Y^k + I)^{-1}(\gamma (Y^k)^T X + W). \end{aligned}$$

6 Alternating directions method of multipliers (ADMM)

Our last algorithm employs ADMM, a framework for solving problems of the form

$$\begin{aligned} \underset{x \in \mathcal{X}, y \in \mathcal{Y}}{\text{minimize}} \quad & g(x) + h(y) \\ \text{subject to} \quad & Ax + By = c. \end{aligned}$$

Often $A = I$, $B = -I$, and $c = 0$, in which case x and y represent parallel copies of a single variable that must satisfy both constraints $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. The *augmented Lagrangian* of this problem, or the Lagrangian appended with quadratic term that acts as regularization [2], is

$$\mathcal{L}_{\text{aug}}(x, y, \lambda) = g(x) + h(y) + \lambda^T(Ax + By - c) + (\alpha/2)\|Ax + By - c\|_2^2$$

where λ is the dual variable controlling the cost of violating the constraint $Ax + By = c$. ADMM operates by iterating through the following sequence: fixing y and λ and minimizing $\mathcal{L}_{\text{aug}}(x, y, \lambda)$ with respect to x, fixing x and λ and minimizing $\mathcal{L}_{\text{aug}}(x, y, \lambda)$ with respect to y, and updating λ via gradient ascent on $\mathcal{L}_{\text{aug}}(x, y, \lambda)$ (see [2] for discussions on convergence and optimality guarantees).

6.1 ADMM for covariance dictionary learning

We reformulate our optimization problem into the equivalent ADMM-compatible problem:

$$\begin{array}{rl} x \in \mathcal{X}, y \in \mathcal{Y} & (1/2)\|X - DW\|_F^2 \\ \text{subject to} & D = U, W = V \\ & U_i \in \mathbb{S}_{\text{vec}}^+, i = 1, \ldots, K \\ & V_{jl} \geq 0, j = 1, \ldots, K, l = 1, \ldots N, \end{array}$$

which has the augmented Lagrangian

$$\mathcal{L}_{\text{aug}}(x, y, \Lambda, \Gamma) = (1/2)\|X - DW\|_F^2 + \sum_{i,j} \Lambda_{ij}(D_{ij} - U_{ij}) + \sum_{i,j} \Gamma_{ij}(W_{ij} - V_{ij}) + (\alpha/2)\|D - U\|_F^2 + (\beta/2)\|W - V\|_F^2$$

Minimizations with respect to D and W can each be rewritten as least-squares problems, while minimizations with respect to U and V are projections onto the feasible sets $\mathbb{S}_{\text{vec}}^+$ and $\mathbb{R}^+$, respectively. Altogether, the k-th iteration of ADMM computes the following updates:

$$\begin{array}{rl} D^k & = (X(W^{k-1})^T + \alpha U^{k-1} - \Lambda^{k-1})(W^{k-1}(W^{k-1})^T + \alpha I)^{-1} \\ W^k & = ((D^k)^T D^k + \beta I)^{-1}(D^k X + \beta V^k - \Gamma^k) \\ U_i^k & = \text{Proj}_{\mathbb{S}_{\text{vec}}^+}((D_i)^k + \Lambda^{k-1}/\alpha),\ i = 1, \ldots, K \\ V_{ij}^k & = \max\{W_{ij}^k + \Gamma^{k-1}/\beta, 0\} \\ \Lambda^k & = \Lambda^{k-1} + \rho\alpha(D - U^k) \\ \Gamma^k & = \Gamma^{k-1} + \rho\beta(W^k - V^k) \end{array}$$

where ρ is a gradient step size we tune. The projection of a vector U_i onto $\mathbb{S}_{\text{vec}}^+$ is described above (see (4)). We terminate ADMM when either $f(D^k, W^k) < \epsilon$ or

$$\frac{|f(D^k, W^k) - f(D^{k-1}, W^{k-1})|}{\max\{1, f(D^{k-1}, W^{k-1})\}} < \epsilon$$

for some ϵ, *i.e.*, when the objective or the relative decrease in objective falls below some threshold.

As with ALS, when we consider dictionary learning on correlation rather than covariance matrices, we use Dykstra's alternating projections to compute the nearest correlation matrix (see Alg 3.3 in [4]).

6.2 Hyperparameters

Our hyperparameters are α and β, the regularization coefficients in the augmented Lagrangian, and the gradient step size ρ. To reduce the number of hyperparameters, we use the heuristic $\beta = (N/n^2)\alpha$ in [12].

7 Algorithm initialization

For both ALS and ADMM, we compared accuracy and runtime with two different initialization strategies for the dictionary D. The first is to initialize each column of D to a random linear combination of the columns of X, where the linear combination coefficients are nonnegative. Nonnegative linear combination preserves positive semidefiniteness, so this creates a feasible initial D.

The second strategy takes inspiration from Wild, Curry, & Dougherty [11], who initialize nonnegative matrix factorization by clustering the input using k-means and using the centroids. Here, we cluster the columns of X using k-means where $k = K$, and initialize the columns of D to the resulting centroids.

For both ALS and ADMM, initialization with k-means often converged many times faster than random initialization, and for ADMM this advantage could be three or more orders of magnitude. Random initialization was often as prohibitively slow as 10 minutes on problems as small as $n = 20$, $K = 2$, $N = 40$. We therefore use k-means initialization throughout the rest of the paper.

8 Evaluation metric for simulated data

Since the optimization problem is non-convex and our algorithms are not guaranteed to find the global minimum, we will compare the true and estimated dictionaries and weights on simulated data.

Note that we can only expect each column of our estimated $\hat{D}$ and each row of our estimated $\hat{W}$ to be accurate up to a scale factor, since scaling $\hat{D}_i$ by some c_i and scaling the i-th row of $\hat{W}$ by $1/c_i$ produces the same reconstruction $\hat{D}\hat{W}$. Our evaluation must also allow the columns of $\hat{D}_i$ and the rows of $\hat{W}$ to be arbitrarily permuted, since applying the same permutation to them also gives the same reconstruction $\hat{D}\hat{W}$. Therefore, we must evaluate $\hat{D}$ and $\hat{W}$ on a per-column and per-row basis, respectively, checking every possible permutation.

For every possible permutation of the columns of $\hat{D}$, we compute the correlation coefficient $\texttt{corrcoef}(D_i - \hat{D}_i)$, which is scale invariant as it quantifies the strength of the linear relationship between D_i and $\hat{D}_i$. We take the average across all columns

$$\texttt{accuracy}_D(D, \hat{D}) = (1/K) \sum_{i=1}^{K} \texttt{corrcoef}(D_i - \hat{D}_i)$$

to evaluate accuracy, and do the same to compare the rows of W and $\hat{W}$:

$$\texttt{accuracy}_W(W, \hat{W}) = (1/K) \sum_{i=1}^{K} \texttt{corrcoef}(W_{i,:}^T - \hat{W}_{i,:}^T)$$

where $W_{i,:}$ denotes the i-th row of W. We then report the maximum accuracy evaluated across all permutations, since we consider the permutation that gives the maximum accuracy

to be the "true" mapping from the atoms of D to the atoms of $\hat{D}$ (and the rows of W to the rows of $\hat{W}$).

We also compare the speed of the algorithms, by measuring the runtime to reach threshold distances from the global optimum.

9 ALS vs. ADMM on simulated data

We compare the accuracy and speed of ALS and ADMM using the k-means initialization scheme (comparisons with PGM and DR in the works). We create ground truth D by generating K random symmetric $n \times n$ matrices, projecting them to the nearest positive semidefinite matrices using (4), then vectorizing them to set the columns of D. We then create ground truth W by generating random nonnegative numbers, and simulate the data $X = DW$. We consider three different problem sizes: $(n, K, N) = (20, 2, 40)$, $(50, 5, 100)$, and $(100, 7, 200)$. (We limit ourselves to small K, since comparing $K!$ permutations of the dictionary explodes combinatorially.)

For the purpose of comparing ALS and ADMM fairly, we need to use equivalent stopping criteria. Since we know for the simulated data that the global optimum is zero, we terminate the algorithms when the objective falls below $\epsilon\|X\|$ for $\epsilon = 5 \times 10^{-2}$, 10^{-2}, and 10^{-3}. We report $\texttt{accuracy}(D, \hat{D})$ and $\texttt{accuracy}(W, \hat{W})$ averaged across 10 different instances of simulated data. The hyperparameters of ALS were set to $\beta_{\text{nls}} = \beta_{\text{psdls}} = 0.1$, and the hyperparameters of ADMM were set to $\alpha = 0.1$ and $\rho = 1$. The general trends remained the same when we tweaked these up and down an order of magnitude.

Tables 1, 2, and 3, give the averaged runtimes and accuracies, rounded to four decimal points. Blank entries denote prohibitively long runtimes that we terminated after 20 minutes. For all three problem sizes, ADMM has an advantage of about an order of magnitude in runtime on the most liberal stopping criterion, while maintaining accuracies comparable to those of ALS. To get tighter convergence with $\epsilon < 5 \times 10^{-2}$ and improve accuracy, however, it is clear that ADMM is many orders of magnitude faster and the only tractable approach. We therefore use ADMM in the following section.

	Time (sec)			$\texttt{accuracy}(D, \hat{D})$			$\texttt{accuracy}(W, \hat{W})$		
ϵ	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}
ALS	0.1960	35.2750		0.9765	0.9954		1.000	1.000	
ADMM	0.03414	0.0910	0.3428	0.9855	0.9998	0.9993	1.000	1.000	1.000

Table 1: Runtimes and accuracies of ALS and ADMM averaged over 10 instances of simulated data with size $(n, K, N) = (20, 2, 40)$.

	Time (sec)			$\texttt{accuracy}(D, \hat{D})$			$\texttt{accuracy}(W, \hat{W})$		
ϵ	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}
ALS	2.3450			0.9209			1.000		
ADMM	0.1458	0.6113	1.5844	0.9242	0.9983	0.9996	1.000	1.000	1.000

Table 2: Runtimes and accuracies of ALS and ADMM averaged over 10 instances of simulated data with size $(n, K, N) = (50, 5, 100)$.

	Time (sec)			$\texttt{accuracy}(D, \hat{D})$			$\texttt{accuracy}(W, \hat{W})$		
ϵ	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}	5×10^{-2}	10^{-2}	10^{-3}
ALS	13.9395			0.8962			1.000		
ADMM	1.1798	5.2510	8.6574	0.8958	0.9986	0.9995	1.000	1.000	1.000

Table 3: Runtimes and accuracies of ALS and ADMM averaged over 10 instances of simulated data with size $(n, K, N) = (100, 7, 200)$.

10 Analyzing neural development data

During the course of embryonic development, how do the covariances between different neurons' firing activities evolve? Understanding this can lend insight into how the function of the brain, and more broadly the nervous system, emerges throughout development. We apply our dictionary learning approach to calcium-imaging recordings of neural activity in developing zebrafish spinal cord (many thanks to Yinan Wan and Philipp Keller at Janelia Research Campus for the data). In calcium imaging, absolute magnitude of a neuron's recording is not as reliable as its relative fluctuations, so we perform dictionary learning on the sample correlation rather than sample covariance matrices.

Fig. 1 shows the positions of the 95 recorded neurons, which occupy the two sides of the spinal cord. After splitting the time series of the neurons' activities into minute-long windows, yielding 287 windows, we computed the sample correlation matrix per window and applied our dictionary learning approach with $K = 2$ atoms using ADMM with k-means initialization.

Fig. 2 shows the learned weights, revealing a stark transition between two correlation atoms in the first and second halves of the experiment. Fig. 3 reveals the learned correlation atoms, where the brightness of the (i, j)-th entry gives the correlation between the i-th and j-th neurons. The neuron indices are ordered such that the first 48 indices correspond to neurons on the right side of the spinal cord, and the last 47 indices correspond to neurons on the left side. The top left corner of atom 0 therefore corresponds to neurons on the right side generally being correlated with each other, and similarly with neurons on the left side.

Fig. 4 provides a graph-based visualization of these correlation atoms, exposing an intriguing trend as the neurons progress through development: in the first half of the experiment, there is no clear pattern of which neurons are correlated to each other. As development continues, however, neurons on each side of the spinal cord simultaneously become 1) more correlated to other neurons on the same side, and 2) more anti-correlated to neurons on the

opposite side. Without incorporating any prior biological information, our covariance dictionary learning approach uncovered a phenomenon known to exist in the developing zebrafish spinal cord, as reported in [10].

11 Conclusion

We have formulated two different optimization algorithms for covariance or correlation dictionary learning, ALS and ADMM with k-means initialization, and have compared their accuracy and speed on simulated datasets. Though the two are on par in terms of accuracy, ADMM converges far more quickly than ALS and is our algorithm of choice. We then demonstrated the power of correlation dictionary learning by applying it to measurements of zebrafish spinal cord neural activity during development. Our learned dictionary revealed two fundamental "states" of neural correlation: one that appears to lack a clear correlation structure, dominant at the beginning of development, and another in which neurons are highly correlated to other neurons on the same side and highly anti correlated to those on the opposite side. This second state grows in strength as development progresses, and is confirmed by the findings in [10].

We hope to extend these results by considering other algorithms often applied to multiconvex problems, including the approach used by Harandi et al. [3]. We'd also like to apply the covariance dictionary learning approach to more examples of real time series, such as financial assets or the behavior of animals in groups, to see what other real-world "network phenomena" it can uncover.

12 Figures

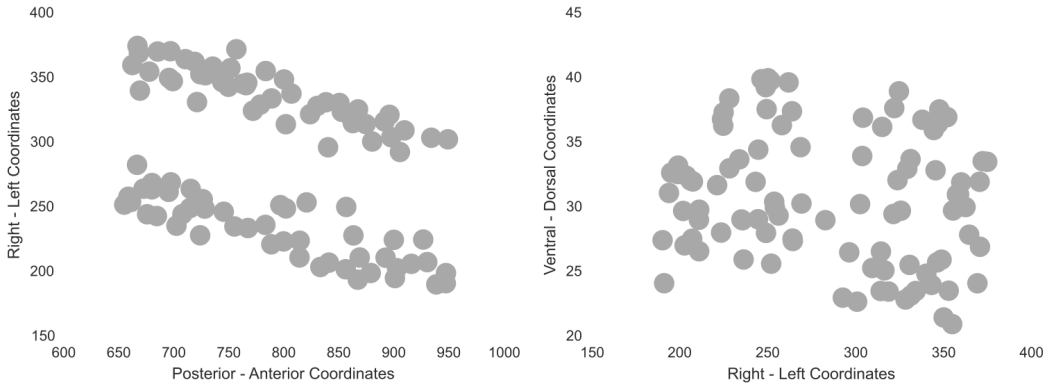

Figure 1: The positions of the recorded spinal cord neurons from two different views, birds-eye (left) and facing the front of the fish (right).

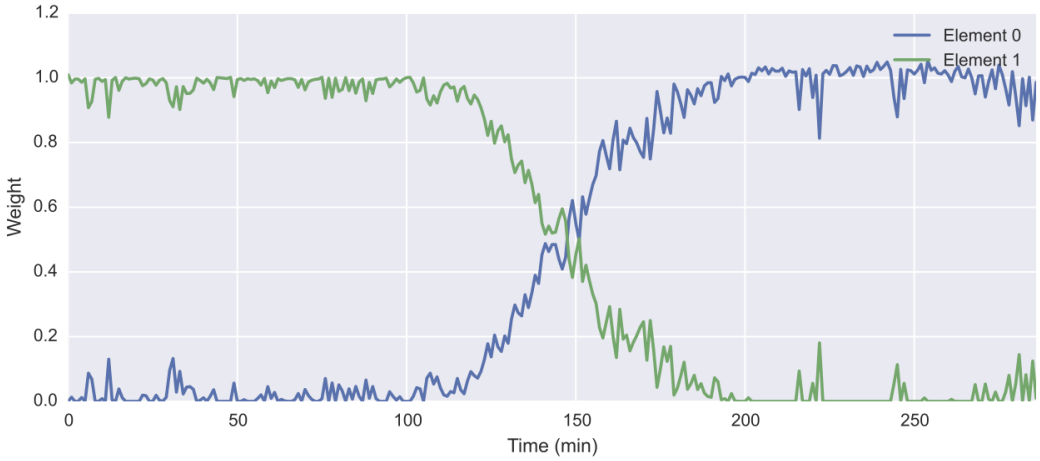

Figure 2: Learned weights on two correlation dictionary atoms using ADMM, over a period of embryonic development.

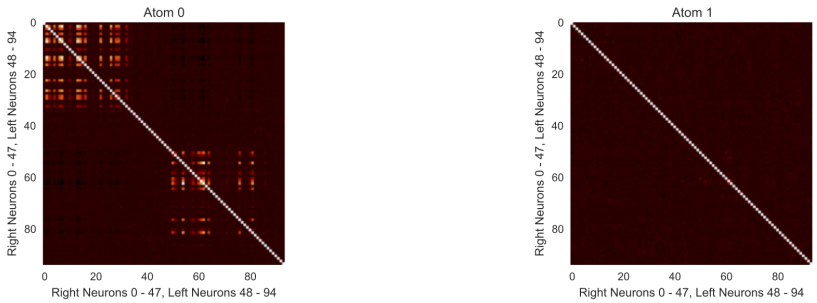

Figure 3: Two correlation dictionary atoms learned using ADMM, where the lower half of the variable indices correspond to neurons on the right side of the spinal cord, and the upper half corresponds to neurons on the left side of the spinal cord.

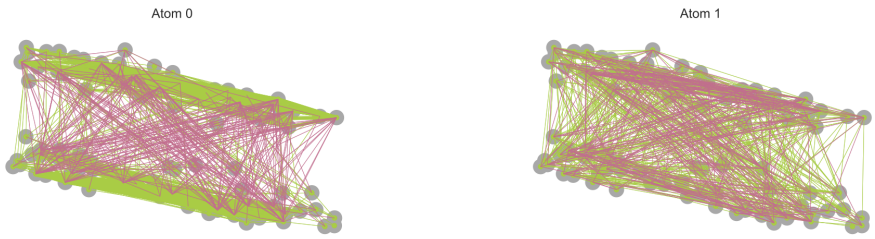

Figure 4: Graph visualization of Fig. 3, where green edges link positively correlated neurons, red edges link anti-correlated neurons, and edge width gives correlation magnitude.

References

[1] Aharon, M., Elad, M., & Bruckstein, A. (2006). K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Sig. Proc.*, 54:4311-4322.

[2] Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein J. (2010). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. and Trends in Machine Learning*, 3(1):1-122.

[3] Harandi, M.T., Hartley, R., Lovell, B., & Sanderson, C. (2015). Sparse coding on symmetric positive definite manifolds using Bregman divergences. *IEEE Transactions on Neural Networks and Learning Systems*, forthcoming.

[4] Higham, N. (2002). Computing the nearest correlation matrix: a problem from finance. *IMA Journal of Numerical Analysis*, 22:329-343.

[5] Lin, C.-J. (2007). Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756 - 2779.

[6] Lin, C.-J. & Moré, J. (1999). Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100-1127.

[7] Combettes, P.L. & Pesquet J.-C. (2011). Proximal splitting methods in signal processing.

[8] Parikh, N. & Boyd, S. (2013). Proximal algorithms.

[9] Sra, S. & Cherian, A. (2011). Generalized dictionary learning for symmetric positive definite matrices with application to nearest neighbor retrieval. *Machine Learning and Knowledge Discovery in Databases*, 6913:318-332.

[10] Warp, E., Agarwal, G., Wyart, C., Friedmann, D., Oldfield, C.S., Conner, A., Del Bene, F., Arrenberg, A.B., Baier, H., & Isacoff, E. (2012). Emergence of patterned activity in the developing zebrafish spinal cord. *Current Biology*, 22(2):93-102.

[11] Wild, S., Curry, J., & Dougherty, A. (2004). Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37(11): 2217-2232.

[12] Xu, Y., Yin, W., Wen, Z., & Zhang, Y. (2012). An alternating direction algorithm for matrix completion with nonnegative factors. *Frontiers of Mathematics in China*, 7(2):365-384.