Notes on Tensor Decompositions

Alex Williams

September 2016

These are some informal notes for a chalk-talk tutorial given at Janelia as part of the junior scientist workshop on theoretical neuroscience.

The goal of these notes is to give a very direct and to-the-point introduction of these topics to scientists in biology and neuroscience. Many of these topics are reviewed in greater technical detail in [14] (for matrix decompositions) and [8] (for tensor decompositions).

1 Motivation

It is standard to represent neural dynamics in a two-dimensional table format. For example, suppose we record the fluorescence of N neurons in a calcium imaging experiment over T video frames. Then, we denote the fluorescence of neuron n at time t as x_{nt}, and the full data is a matrix $\mathbf{X}$ containing `neurons` × `time` (Fig 1A).

Many experimental studies in biology, neuroscience, and the social sciences follow a *factorial design* with repeated measurements made from the same subject over time or under a variety of experimental conditions. These cases do not always lend themselves to a simple matrix representation.

Suppose we repeat the recording above over K behavioral trials, and then denote the fluorescence of neuron n at time t on trial k as x_{ntk}. The full dataset is now a `neurons` × `time` × `trials` *tensor*, denoted $\boldsymbol{\mathcal{X}}$ (Fig 1B).

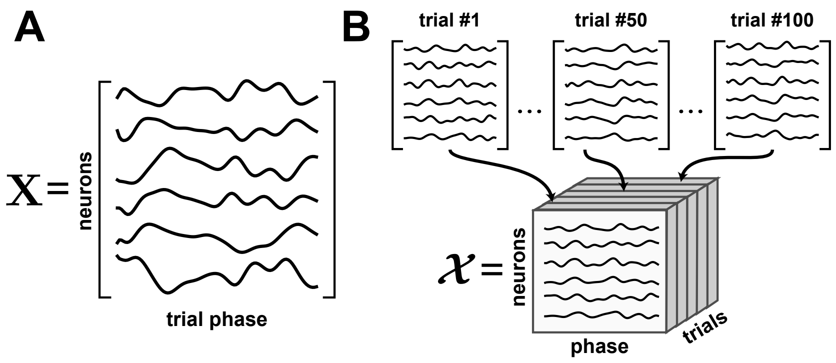

Figure 1: *Neural activity data represented in matrix and tensor formats.* **(A)** Activity of a neural population on a single trial. **(B)** Activity of a neural population across multiple trials.

Observations:

- We are interested in how neural dynamics change both within-trials (e.g., correlates of decision-making) and across-trials (e.g., correlates of learning).
- Each *mode* (dimension) of the tensor can be large.

Questions:

- How do we perform dimensionality reduction on higher-order (tensor) datasets?
- How do we separate and describe across-trial (learning-related?) dynamics from within-trial dynamics?
- What are the advantages of representing data as a tensor and working with this structure directly? Why not work with matrices?

Outline:

- Review dimensionality reduction in matrix case.
- Introduce canonical polyadic (CP) tensor decomposition.
- Time permitting, short overview of other techniques
  - Tucker decomposition
  - Demixed PCA

2 Matrix Decompositions

2.1 Basic notation:

- Uppercase-boldface letters are matrices. Lowercase-boldface letters are vectors. Non-boldface letters are scalars.
- For this section, let $\mathbf{X} \in \mathbb{R}^{N \times T}$ denote a matrix of `neurons` × `time`, holding firing rates.
- Denote row n of $\mathbf{X}$ by $\mathbf{x}_{n:}$ and denote column t of $\mathbf{X}$ by $\mathbf{x}_{:t}$.

2.2 Three views of matrix factorization

PCA, NMF, and many other classic techniques all fit within the framework of *matrix factorization.*[1] In particular, they all seek to approximate the full dataset as the product of two low-rank factor matrices $\mathbf{A} \in \mathbb{R}^{N \times R}$ and $\mathbf{B} \in \mathbb{R}^{T \times R}$

$$\mathbf{X} \approx \widehat{\mathbf{X}} = \mathbf{A}\mathbf{B}^T \tag{1}$$

In PCA, the elements of $\mathbf{A}$ are called *loadings* and the elements of $\mathbf{B}$ are called the *components.* We instead use more general terminology: we call $\mathbf{A}$ the *neuron factors* and $\mathbf{B}$ the *time factors*, since they respectively provide low-dimensional representations for each neuron and each time point in the full matrix.

NMF and other variants find $\mathbf{A}$ and $\mathbf{B}$ subject to different constraints and optimization objectives. They find different factors, but the relationship between the factors and the reconstructed dataset is the same.

It is useful to re-express Eq 1 in a couple of ways. First, consider reconstructing the activity of a single neuron n:

$$\widehat{\mathbf{x}}_{n:} = \sum_{r=1}^{R} a_{nr} \mathbf{b}_{:r}^T \tag{2}$$

Thus, each neuron is a weighted sum of R prototypical activity traces.

Next, consider that the full approximation to the dataset can be expressed as the sum of R rank-one matrices. To appreciate this, extend Eq 2 across the rows of $\widehat{\mathbf{X}}$ (since the approximation for each row is decoupled).

$$\widehat{\mathbf{X}} = \sum_{r=1}^{R} \mathbf{a}_{:r} \circ \mathbf{b}_{:r}^T \tag{3}$$

Where $\mathbf{u} \circ \mathbf{v}$ denotes a vector outer product, $\mathbf{u}\mathbf{v}^T$. The outer product of two vectors produces a rank-one matrix:

$$\mathbf{Y} = \mathbf{u} \circ \mathbf{v} \iff y_{ij} = u_i v_j \tag{4}$$

[1] Dimension reduction techniques like tSNE that don't fit in this framework are often stupid ideas

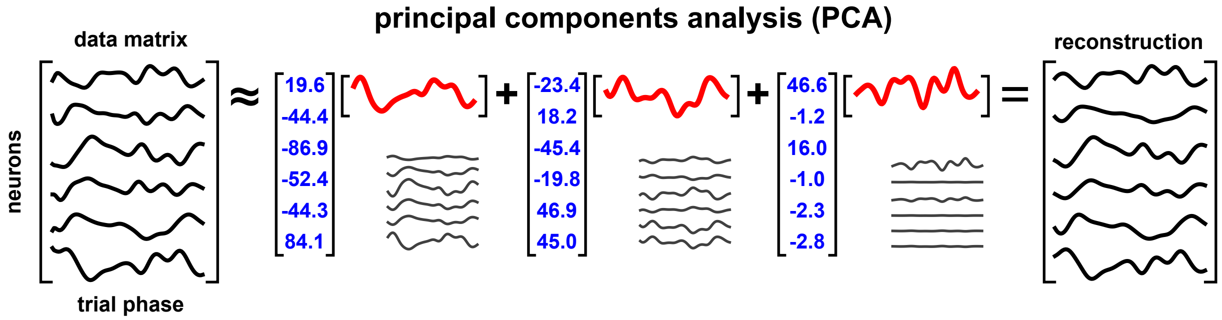

Figure 2: Using matrix decompositions to visualize single-trial, or trial-averaged data.

2.3 Popular matrix factorizations

2.3.1 PCA

PCA can be formulated as an optimization problem in two ways:

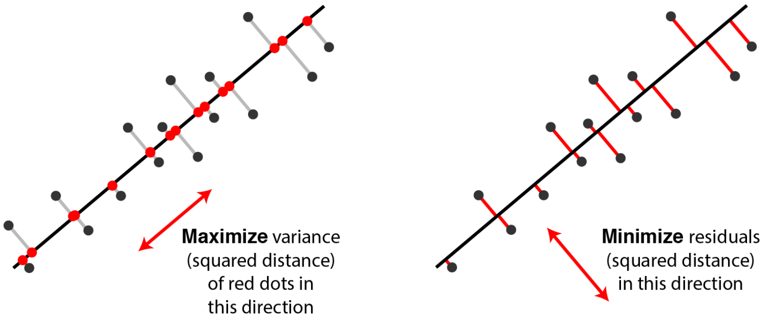
Maximize variance (squared distance) of red dots in this direction
Minimize residuals (squared distance) in this direction

Figure 3: Two equivalent perspectives of PCA.

In the first (Fig 3A) we find factor matrices that maximize the retained variance after projecting the data:

$$\begin{aligned} &\text{let} && \mathbf{A} = \mathbf{X}\mathbf{B}\ ,\ \text{then} \\ &\underset{\mathbf{B}}{\text{maximize}} && \sum_{r=1}^{R} \mathbf{b}_{:r}^T \mathbf{X}^T \mathbf{X} \mathbf{b}_{:r} \\ &\text{subject to} && \mathbf{b}_{:r}^T \mathbf{b}_{:r} = 1 \end{aligned} \tag{5}$$

In the second formulation (Fig 3B) we find factor matrices that minimize the squared reconstruction error:

$$\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \quad \left\| \mathbf{X} - \mathbf{A}\mathbf{B}^T \right\|_F^2 \tag{6}$$

This second perspective is usually simpler to think about and easier to generalize, but the first is still useful to know and we'll refer to it in the section on demixed PCA.

Note that the solution to these optimization problems is not unique. Any invertible linear transformation $\mathbf{F}$ can be used to produce a new pair of factor matrices $\{\mathbf{A}', \mathbf{B}'\}$ that produce an equivalent prediction:

$$\mathbf{A}\mathbf{B}^T = \mathbf{A}\mathbf{F}^{-1}\mathbf{F}\mathbf{B}^T = \mathbf{A}'\mathbf{B}'^T \tag{7}$$

This indeterminacy is typically handled by imposing an additional constraint that the columns of $\mathbf{A}$ and $\mathbf{B}$ be orthonormal. These (potentially disadvantageous) constraints are not necessary for tensor decompositions.

2.3.2 NMF

This is just the PCA objective function with the added constraint that the factor matrices be non-negative.

$$\begin{aligned} &\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \quad && \left\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\right\|_F^2 \\ &\text{subject to} && a_{nr} \geq 0,\ b_{tr} \geq 0 \end{aligned} \tag{8}$$

2.3.3 Sparse PCA

There are several other variants of sparse PCA, e.g. [4].

$$\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \quad \left\|\mathbf{X} - \mathbf{A}\mathbf{B}^T\right\|_F^2 \; + \; \lambda_{\mathbf{A}} \sum_{n=1}^{N} \left\|\mathbf{a}_{n:}\right\|_1 \; + \; \lambda_{\mathbf{B}} \sum_{t=1}^{T} \left\|\mathbf{b}_{t:}\right\|_1 \tag{9}$$

2.3.4 Logistic PCA

$$\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \quad \sum_{n=1}^{N}\sum_{t=1}^{T} \log\left(1 + \exp\left(-x_{nt} \cdot \textstyle\sum_{r=1}^{R} a_{nr} b_{tr}\right)\right) \tag{10}$$

2.3.5 Generalized Low-Rank Model

Described in [14], code available at: `https://github.com/madeleineudell/LowRankModels.jl`

$$\begin{aligned} &\underset{\mathbf{A},\mathbf{B}}{\text{minimize}} \quad && \sum_{i=1}^{I}\sum_{j=1}^{J} \ell_j\left(x_{ij}, \hat{x}_{ij}\right) \; + \; \sum_{i=1}^{I} \rho_i(\mathbf{a}_{i:}) \; + \; \sum_{j=1}^{J} \gamma_j(\mathbf{b}_{j:}) \\ &\text{subject to} && \widehat{\mathbf{X}} = \mathbf{A}\mathbf{B}^T \end{aligned} \tag{11}$$

2.4 Fitting matrix factorizations

- PCA can be solved by truncated SVD.

  - Rare and special case of a nonconvex problem that can be provably solved in polynomial time.

- All others, solve by *alternating gradient descent* and similar techniques.
  - $\widehat{\mathbf{X}} = \mathbf{A}\mathbf{B}^T$ is *biconvex*, meaning that it is convex if you fix either $\mathbf{A}$ or $\mathbf{B}$ and optimize over the other variable.
  - Alternate back-and-forth to solve (or approximately solve) convex subproblems.
  - In general, no guarantees that you reach a global optimum. However, for certain problems it has been proven that there are no local minima (i.e. all critical points are saddle points); see [13, 5]. And therefore gradient descent converges to the global minimum under *extremely* mild conditions [9].
- For certain problems, notably NMF [6], there are specialized algorithms that take advantage of the problem structure.

3 Canonical Polyadic (CP) Tensor Factorization

Extending matrix factorization techniques to tensor datasets is conceptually simple, but not widely used in systems neuroscience. (But see the cognitive and neuroimaging literature for some applications [10, 11, 12, 1].)

Since we are in an exploratory data analysis setting, interpretability is an important goal for us. Thus, we'll focus on CP tensor factorization, which is particularly simple and interpretable generalization of the ideas expressed in the previous section.

Some notation and basic terms (not too critical to memorize):

- The *order* of a tensor is the number of dimensions it has (`ndims(X)` would give the order in MATLAB/Julia). A vector is an order-1 tensor, and a matrix is an order-2 tensor.
- Each indexed dimension is also called a *mode* of the tensor. E.g. `size(X,1)` would give the size of a tensor along the first mode, while `size(X,2)` would give the size along the second mode in MATLAB/Julia.
- Order-3 tensors and higher-order tensors are denoted by bold letters in calligraphic font, e.g. $\boldsymbol{\mathcal{X}}$.

Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times K}$ denote a $\texttt{neurons} \times \texttt{time} \times \texttt{trials}$ data tensor holding neural activity. We will apply CP decomposition to achieve separate low-dimensional representations for within-trial dynamics and across-trial dynamics.

For a third-order tensor, CP decomposition finds a set of three factor matrices, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ that approximate the data set as the sum of R vector outer products:

$$\widehat{\boldsymbol{\mathcal{X}}} = \sum_{r=1}^{R} \mathbf{a}_{:r} \circ \mathbf{b}_{:r} \circ \mathbf{c}_{:r} \tag{12}$$

which is very similar to the formulation of matrix decomposition in Eq 3.

Eq 12 can be visualized as follows:

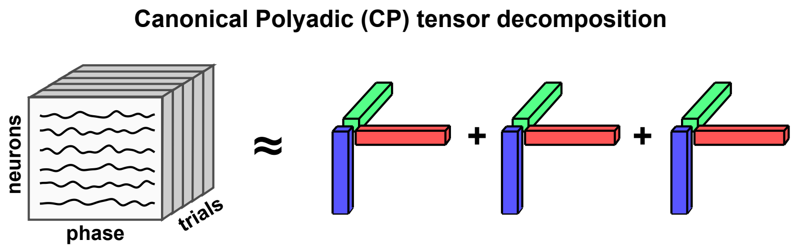

Figure 4: Canonical Polyadic (CP) tensor decomposition on a 3rd order tensor of neural data (see Fig ??B). Similar to Fig 2, the data are approximated as a sum of vector outer products. The first two ($\mathbf{a}_{:r}$, blue; and $\mathbf{b}_{:r}$, red) provide low-dimensional embeddings of the neurons and within-trial dynamics (again, compare to Fig 2). The third set of vectors ($\mathbf{c}_{:r}$, green) provide a low-dimensional embedding of the across-trial changes in dynamics (e.g. learning-related dynamics).

Note that the vector outer product readily generalizes to 3-dimensions and higher. The outer product of three vectors $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ produces a rank-one, third-order *tensor*:

$$\boldsymbol{\mathcal{Y}} = \mathbf{u} \circ \mathbf{v} \circ \mathbf{w} \iff y_{ijk} = u_i v_j w_k \tag{13}$$

CP decomposition introduces a third factor matrix, $\mathbf{C}$, which we call the *trial factors.*

Classically, the CP factors are fit in by minimizing squared error (similar to PCA):

$$\underset{\mathbf{A},\mathbf{B},\mathbf{C}}{\text{minimize}} \quad \sum_{n=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} (x_{ntk} - \widehat{x}_{ntk})^2 \tag{14}$$

But this approach can be adapted to different loss functions if desired (e.g. [3]).

3.1 Interpretations and relation to PCA

Under a CP model, the activity of a single neuron on a particular trial is:

$$\widehat{\mathbf{x}}_{n:k} = \sum_{r=1}^{R} a_{nr} c_{kr} \mathbf{b}_{:r} \tag{15}$$

This is similar to PCA (compare to Eq 2), in that each neuron's activity is approximated as a combination of components ($\mathbf{b}_{:r}$) but each component is weighted by a neuron-specific coefficient (a_{nr}) and a trial-specific coefficient (c_{kr}).

Thus, the fundamental difference between CP and PCA is the introduction of trial-specific re-scaling. This is even more apparent in the following equation, which shows how the CP model approximates the entire population activity for a single trial (compare to Eq 1):

$$\widehat{\mathbf{X}}_{::k} = \mathbf{A}\,\mathrm{Diag}(\mathbf{c}_{k:})\,\mathbf{B}^T \tag{16}$$

where Diag(·) transforms a vector into a diagonal matrix.

Eq 16 can be visualized as follows:

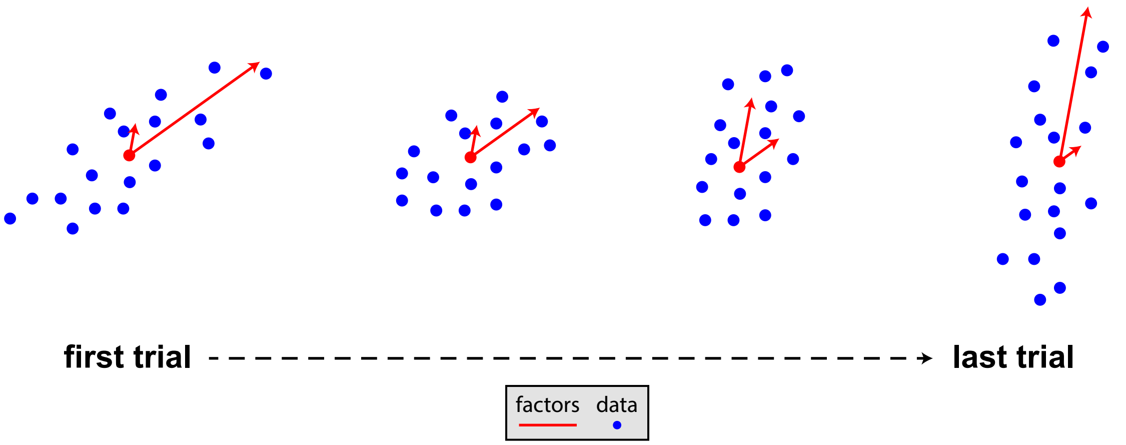

Figure 5: The direction of the factors (red lines) are fixed across trials, but their magnitude can change to reflect rotations/other changes in the data (blue dots). For example, each data point could correspond to a T-dimensional vector (holding a neuron's activity on a specified trial) and the factors would describe an R-dimensional embedding/approximation of these data.

Eq 16 also provides intuition for why CP factorizations — unlike PCA — are unique. Since the middle matrix, i.e. $\text{Diag}(\mathbf{c}_{k:})$, is constrained to be diagonal, an arbitrary linear transformation (i.e., matrix $\mathbf{F}$ in Eq 7) cannot be introduced without destroying the diagonal structure. However, we *can* sneak in permutation and scaling transformations and still preserve the diagonality. This non-uniqueness does not matter to much since we mostly care about the shape/direction of the latent factors.

It is useful to have a standard procedure for permuting and re-scaling the CP factors so that the factorization is unique:

- The factors (columns of $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$) are normalized to unit length. The scalings are absorbed into λ_i.
- The signs of the factors are flipped so that all λ_i are positive.
- The factors are permuted to order them from largest to smallest λ_r.

4 Tucker Decompositions

This section can be skipped unless you have a burning desire to be a tensor aficionado.

CP decomposition is a special case of a more general class called *Tucker decompositions*. Here, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$ is approximated based on a smaller core tensor, $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{R_i \times R_j \times R_k}$, and three factor matrices $\mathbf{A} \in \mathbb{R}^{I \times R_i}$, $\mathbf{B} \in \mathbb{R}^{J \times R_j}$, and $\mathbf{C} \in \mathbb{R}^{K \times R_k}$. Of course, the Tucker model readily generalizes to higher-order tensors, but we'll stick to three-dimensions to simplify presentation.

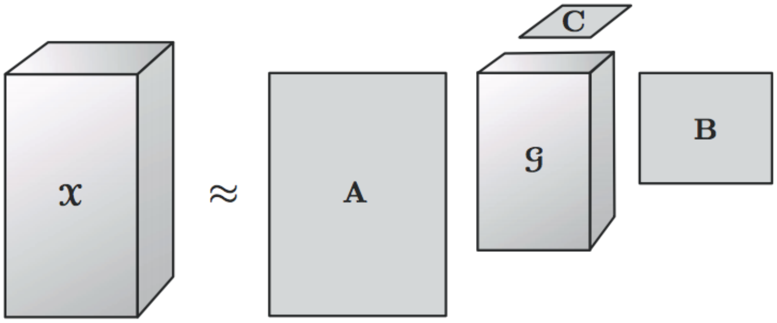

Figure 6: *Tucker decomposition of a third-order tensor.* Reproduced from [8].

The Tucker model is:

$$\widehat{x}_{ijk} = \sum_{r_i}^{R_i} \sum_{r_j}^{R_j} \sum_{r_k}^{R_k} s_{r_i r_j r_k} a_{i r_i} b_{j r_j} c_{k r_k} \tag{17}$$

Unlike CP decomposition, the Tucker model allows for there to be a different number of factors (i.e. $\{R_i, R_j, R_k\}$) along each mode of the tensor.

CP decomposition maps onto a Tucker decomposition where the core tensor is constrained to be super-diagonal. To see this, start with Eq 19 and impose that the core tensor is square ($R_i = R_j = R_k = R$) and zero unless $r_i = r_j = r_k = r$, then three sums simplify to the CP model:

$$\widehat{x}_{ijk} = \sum_{r}^{R} \lambda_r a_{ir} b_{jr} c_{kr} \qquad \Longleftrightarrow \qquad \widehat{\boldsymbol{\mathcal{X}}} = \sum_{r}^{R} \lambda_r \mathbf{a}_{:r} \circ \mathbf{b}_{:r} \circ \mathbf{c}_{:r}$$

There are two major disadvantages to using Tucker decompositions for exploratory data analysis.

- The core tensor is difficult to visualize
- The decomposition is invariant to rotations in the factor matrices (similar to Eq 7 in the case of PCA)

To appreciate this second point, let $\boldsymbol{\mathcal{S}} \times_m \mathbf{F}$ denote multiplication along mode-m of a tensor $\boldsymbol{\mathcal{S}}$ with a matrix $\mathbf{F}$, defined for the first three modes as:

$$\begin{aligned} (\boldsymbol{\mathcal{S}} \times_1 \mathbf{F})_{i r_j r_k} &= \sum_{r_i} s_{r_i r_j r_k} f_{r_i i} \\ (\boldsymbol{\mathcal{S}} \times_2 \mathbf{G})_{r_i j r_k} &= \sum_{r_j} s_{r_i r_j r_k} g_{r_j j} \\ (\boldsymbol{\mathcal{S}} \times_3 \mathbf{H})_{r_i r_j k} &= \sum_{r_k} s_{r_i r_j r_k} h_{r_k k} \end{aligned} \tag{18}$$

Which means that the Tucker model is:[2]

$$\widehat{x}_{ijk} = \boldsymbol{\mathcal{S}} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \tag{19}$$

[2] The order of this multiplication is irrelevant

We can now make a formal statement for the lack of uniqueness for the Tucker model. Let $\{\boldsymbol{\mathcal{S}}, \mathbf{A}, \mathbf{B}, \mathbf{C}\}$ and $\{\boldsymbol{\mathcal{S}}', \mathbf{A}', \mathbf{B}', \mathbf{C}'\}$ denote the parameters for two Tucker models. These models produce the same $\widehat{\boldsymbol{\mathcal{X}}}$ if:

$$\begin{aligned} \boldsymbol{\mathcal{S}}' &= \boldsymbol{\mathcal{S}} \times_1 \mathbf{F} \times_2 \mathbf{G} \times_3 \mathbf{H} \\ \mathbf{A}' &= \mathbf{A}\mathbf{F}^{-1} \\ \mathbf{B}' &= \mathbf{B}\mathbf{G}^{-1} \\ \mathbf{C}' &= \mathbf{C}\mathbf{H}^{-1} \end{aligned} \tag{20}$$

Which follows from a basic property of mode-m tensor-matrix multiplication:

$$\boldsymbol{\mathcal{S}} \times_1 \mathbf{U} \times_1 \mathbf{V} = \boldsymbol{\mathcal{S}} \times_1 \mathbf{V}\,\mathbf{U}$$

5 Matrix-based alternatives

5.1 Forcing tensor datasets into the matrix decomposition framework

Neuroscientists have been unwittingly working with tensor datasets for a long time. Since matrix decomposition techniques are easier to come by (e.g. PCA is a standard function in MATLAB/Python/Julia), the standard protocol in the field is to flatten the tensor into a matrix and then perform data analysis.

A common approach is to consider the unfolding in Fig **??**A to find a low-dimensional representation of neurons. Then, each trial can be projected onto this low-dimensional subspace:

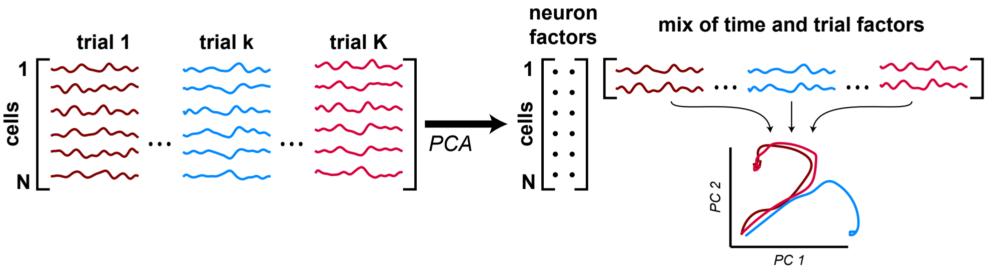

Figure 7: *Applying PCA to tensor unfoldings.* The data tensor can be unfolded or matricized along each of the three modes. (A) Unfolding the tensor along the first mode produces a matrix which we call the *neurons unfolding* of the tensor. (B) Unfolding along the third mode produces the *trials unfolding.* (C) Unfolding along the second mode produces the *time unfolding.* Fitting PCA (or any other matrix decomposition) to these unfoldings produces a low-dimensional embedding for the (A) neurons, (B) trials, and (C) within-trial dynamics/phase.

Potential advantages comapared to CP decomposition:

- Imposing orthogonality amongst the factors can be helpful to visualize data in 2d or 3d plots. Although CP factors can also be orthogonalized for visualization.
- In the case of PCA, there is no worry about getting stuck in a local minima.

- Greater familiarity in the field.

Possible disadvantages:

- There is no connection/matching between the low-dimensional factors. In CP decomposition, the factors come in a set: each neuron factor (column r of $\mathbf{A}$) is matched to a within-trial time factor (column r of $\mathbf{B}$) and across trial factor (column r of $\mathbf{C}$). So, e.g., if you find a particular across-trial trend, it is trivial to go back and see which neurons participate in this trend and what their within-trial dynamical pattern is.
- If using PCA, the factors are not unique due to Eq 7. Thus, PCA is useful for finding a linear subspace for visualizing the data, whereas CP decomposition goes beyond that and tries to identify latent factors that are directly interpretable/meaningful.

5.2 Demixed PCA (dPCA)

5.2.1 Notation

As before, let x_{ntk} denote the firing rate of neuron n at time t on trial k. Define the trial-averaged firing rate for neuron n as:

$$\overline{x}_{nt\cdot} = \frac{1}{K}\sum_{k=1}^{K} x_{ntk}$$

This "dot notation" for averaging can be extended to average over any other indexed variable (or comibination). For example, if each trial is one of several experimental conditions indexed by $c \in \{1, ..., C\}$, then x_{ntck} denotes the firing rate of neuron n across trials and conditions. Assuming an equal number of trials for each condition,[3] the average firing rate across all trials and conditions is:

$$\overline{x}_{nt\cdot\cdot} = \frac{1}{KC}\sum_{c=1}^{C}\sum_{k=1}^{K} x_{ntck}$$

5.2.2 Overview

DPCA [7] is a supervised dimension reduction technique[4] that aims to tradeoff between two competing aims:

- Find components that retain as much variance in the data as possible
- Have each component explain variance only along a specified indexed variable.

DPCA is a tradeoff between PCA (which finds a linear projection retaining maximal variance) and linear discriminant analysis (LDA, which finds a linear projection that maximally separates class labels). Geometrically, this looks like the following:

[3] This actually ends up being an important assumption that is often violated, see methods of [7] for dealing with this scenario.

[4] Demixed PCA was also described in an earlier publication from the same group [2]. There are small technical details between the reports, so we focus on the most recent publication.

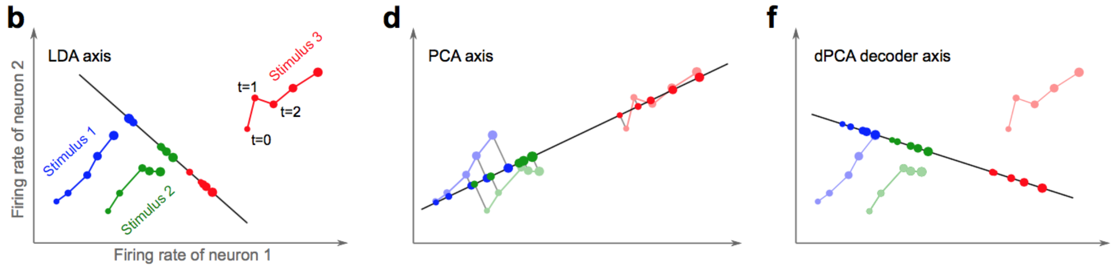

Figure 8: Geometric intuition of DPCA as a tradeoff between PCA and LDA. Figure reproduced from [7].

5.2.3 Sum-of-squares partition

The central observation of DPCA is that the total variance in the dataset can be partioned as shown in Fig 9. This is often called the *sum-of-squares partition* ([7] call it the *marginalization procedure*).

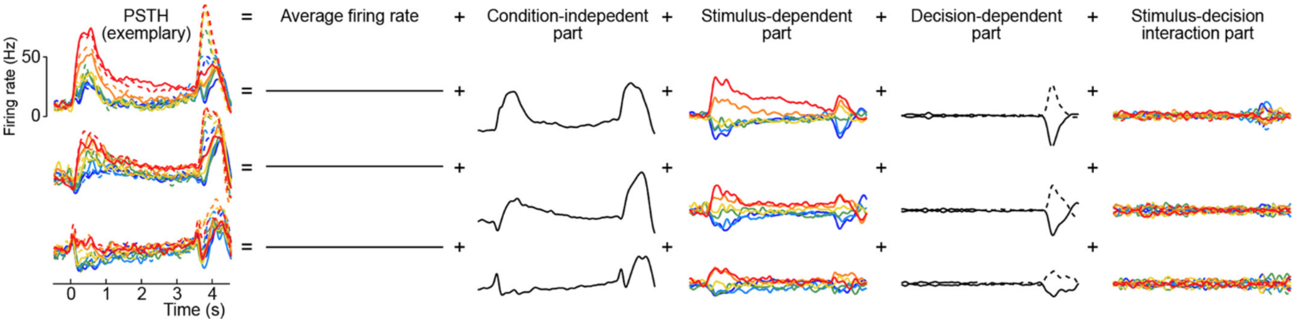

Figure 9: DPCA marginalization procedure for three example neurons. Firing rates across all trials are shown at left. Each trial is associated with a stimulus condition (line colors) and a behavioral decision the animal made (dotted vs solid lines), which are analogous to independent variables in regression. The firing rate for a neuron on any particular trial is modeled as a linear sum of the average firing rate for that condition and decision. This figure is reproduced from [7].

The partitioning of variance generalizes to having multiple independent variables (e.g., across experimental conditions, behavioral outcomes/decisions, see [7]). We will prove a simple case for a single neuron:

$$\underbrace{\sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk}-\overline{x}_{n\cdot\cdot})^2}_{\substack{\text{total variability}\\ \text{in neuron } n}} = \underbrace{\sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk}-\overline{x}_{nt\cdot})^2}_{\substack{\text{variability of } n\\ \text{about its trial-averaged}\\ \text{firing rate}}} + \underbrace{\sum_{t=1}^{T}\sum_{k=1}^{K}(\overline{x}_{nt\cdot}-\overline{x}_{n\cdot\cdot})^2}_{\substack{\text{variability of } n\text{'s}\\ \text{trial-averaged firing rate}}}$$

Proof:

$$\begin{aligned}
\text{total variance in neuron } n &\propto \sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{n\cdot\cdot})^2 \\
&= \sum_{t=1}^{T}\sum_{k=1}^{K}[(x_{ntk} - \overline{x}_{nt\cdot}) + (\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot})]^2 \\
&= \sum_{t=1}^{T}\sum_{k=1}^{K}\left[(x_{ntk} - \overline{x}_{nt\cdot})^2 + 2(x_{ntk} - \overline{x}_{nt\cdot})(\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot}) + (\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot})^2\right] \\
&= \sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{nt\cdot})^2 + 2\sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{nt\cdot})(\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot}) + \sum_{t=1}^{T}\sum_{k=1}^{K}(\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot})^2 \\
&= \sum_{t=1}^{T}\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{nt\cdot})^2 + 2\sum_{t=1}^{T}(\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot})\underbrace{\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{nt\cdot})}_{(*)} + \sum_{t=1}^{T}\sum_{k=1}^{K}(\overline{x}_{nt\cdot} - \overline{x}_{n\cdot\cdot})^2
\end{aligned}$$

The highlighted term $(*)$ is zero, which follows from a basic property of the arithmetic mean (i.e., the sum of absolute deviations from a sample mean is zero):

$$\begin{aligned}
\sum_{k=1}^{K}(x_{ntk} - \overline{x}_{nt\cdot}) &= \sum_{k=1}^{K}\left(x_{ntk} - \frac{1}{K}\sum_{k=1}^{K}x_{ntk}\right) \\
&= \sum_{k=1}^{K}x_{ntk} - K\left(\frac{1}{K}\sum_{k=1}^{K}x_{ntk}\right) \\
&= \sum_{k=1}^{K}x_{ntk} - \sum_{k=1}^{K}x_{ntk} = 0
\end{aligned}$$

This basic concept also arises in ordinary linear regression as schematized in Fig 10. This can be derived in a manner similar to above (instead of using a property of the average, use a property of the normal equations).

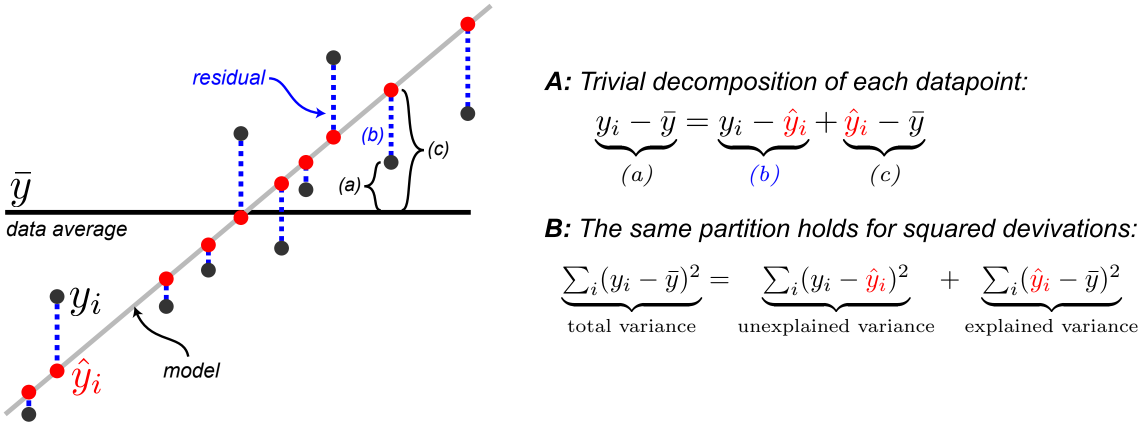
residual
$\bar{y}$
data average
(b)
(c)
(a)
y_i
$\hat{y}_i$
model
A: *Trivial decomposition of each datapoint:*
$\underbrace{y_i - \bar{y}}_{(a)} = \underbrace{y_i - \hat{y}_i}_{(b)} + \underbrace{\hat{y}_i - \bar{y}}_{(c)}$
B: *The same partition holds for squared devivations:*
$\underbrace{\sum_i (y_i - \bar{y})^2}_{\text{total variance}} = \underbrace{\sum_i (y_i - \hat{y}_i)^2}_{\text{unexplained variance}} + \underbrace{\sum_i (\hat{y}_i - \bar{y})^2}_{\text{explained variance}}$

Figure 10: *Partitioning data into aspects that are explainable and unexplainable by a linear model.*

5.2.4 DPCA algorithm

We'll be brief here, since [7] provides a nice and understandable description in their methods.

Consider the following variables associated with each measurement of population activity:

- time within trial ($t \in \{1, ..., T\}$)
- stimulus condition ($s \in \{1, ..., S\}$)
- decision/behavior ($d \in \{1, ..., D\}$)

Partition the data $\mathbf{X}$ across trials indexed by $k \in \{1, ..., K\}$ using the basic methods from of the previous section.

$$\mathbf{X} = \mathbf{X}_{(t)} + \underbrace{\mathbf{X}_{(s)} + \mathbf{X}_{(ts)}}_{\mathbf{X}_{(ts)}} + \underbrace{\mathbf{X}_{(d)} + \mathbf{X}_{(td)}}_{\mathbf{X}_{(td)}} + \mathbf{X}_{(tsd)} + \underbrace{\mathbf{X}_{(tsdk)}}_{\mathbf{X}_{(\text{noise})}} \tag{21}$$

Note that we lump a few terms together.

Again, the important detail is that this decomposition preserves the total variance and distributes it among these terms.

Finally, for each term, the following reduced-rank regression problem is solved:

$$\underset{\mathbf{F}_\phi, \mathbf{D}_\phi}{\text{minimize}} \quad \left\| \mathbf{X}_\phi - \mathbf{F}_\phi \mathbf{D}_\phi^T \right\|_F^2 \tag{22}$$

Where $\phi \in (t), (td), (ts), (tsd)$ indexes over the data partitions, and the low-rank matrices $\mathbf{F}_\phi$ and $\mathbf{D}_\phi$ provide low-dimensional embedding of the data selectively for each data partition.

This reduced reduced-rank regression optimization problem can be solved exactly using the SVD.

5.2.5 Comparison to CP decomposition

Demixed PCA can be applied to tensor data (though the data need not be balanced) but still applies to a matrix unfolding of the data. Rather than finding low-dimensional factors along higher-order tensor dimensions, it finds PCA factors similar to Fig 7.

One particularly nice application the demixing procedure is to regress away the effects of nuisance variables or confounding factors.

References

[1] Evrim Acar, Canan Aykut-Bingol, Haluk Bingol, Rasmus Bro, and Blent Yener. Multiway analysis of epilepsy tensors. *Bioinformatics*, 23(13):i10–i18, 2007.

[2] Wieland Brendel, Ranulfo Romo, and Christian K Machens. Demixed principal component analysis. In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2011.

[3] Eric C Chi and Tamara G Kolda. On tensors, sparsity, and nonnegative factorizations. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1272–1299, 2012.

[4] Alexandre d'Aspremont, Laurent El Ghaoui, Michael I Jordan, and Gert RG Lanckriet. A direct formulation for sparse pca using semidefinite programming. *SIAM review*, 49(3):434–448, 2007.

[5] Rong Ge, Jason D. Lee, and Tengyu Ma. Matrix completion has no spurious local minimum, 2016.

[6] Jingu Kim, Yunlong He, and Haesun Park. Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.

[7] Dmitry Kobak, Wieland Brendel, Christos Constantinidis, Claudia E Feierstein, Adam Kepecs, Zachary F Mainen, Xue-Lian Qi, Ranulfo Romo, Naoshige Uchida, and Christian K Machens. Demixed principal component analysis of neural population data. *eLife*, 5:e10989, apr 2016.

[8] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[9] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. Gradient descent converges to minimizers, 2016.

[10] Fumikazu Miwakeichi, Eduardo Martnez-Montes, Pedro A. Valds-Sosa, Nobuaki Nishiyama, Hiroaki Mizuhara, and Yoko Yamaguchi. Decomposing {EEG} data into spacetimefrequency components using parallel factor analysis. *NeuroImage*, 22(3):1035 – 1045, 2004.

[11] Morten Mørup, Lars Kai Hansen, Christoph S Herrmann, Josef Parnas, and Sidse M Arnfred. Parallel factor analysis as an exploratory tool for wavelet transformed event-related eeg. *NeuroImage*, 29(3):938–947, 2006.

[12] Morten Mrup, Lars Kai Hansen, and Sidse M. Arnfred. Erpwavelab: A toolbox for multi-channel analysis of timefrequency transformed event related potentials. *Journal of Neuroscience Methods*, 161(2):361 – 368, 2007.

[13] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *ICML*, volume 3, pages 720–727, 2003.

[14] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.