Lecture 7: Analyzing representations and transformations

7.1 Basis vectors and basis functions

7.1.1 Definitions

- A representational space is a specific kind of vector space.
- *Vectors* are simply collections of mathematical objects, be they numbers, functions, or other vectors.
- A vector *space* is a set of vectors that is closed under addition and multiplication (meaning that a sum or multiple of any two vectors in the set is also in the set).
- A *basis* is an *independent* set of vectors that *span* the vector space.
- A set of vectors $\mathbf{x}_n$ is *independent* if

  $$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = 0$$

  only when

  $$a_1 = a_2 = \ldots = a_n = 0,$$

  where a_n are scalars. As a result, n must be equal to the dimension of the vectors in $\mathbf{V}$ in order for $\mathbf{x}_n$ to be independent.
- A set of vectors *spans* a vector space if any vector in that space can be written as a linear sum of those vectors. That is, if for all $\mathbf{x} \in \mathbf{V}$ there are some a_n,

  $$a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_n\mathbf{x}_n = \mathbf{x},$$

  then the set of vectors $\mathbf{x}_n$ span the vector space $\mathbf{V}$.
- Example: The standard Cartesian basis in any number of dimensions is comprised of the unit vectors along the principle axes. All the vectors in the standard bases are orthogonal (meaning that the dot product of any two will be zero; i.e., $\mathbf{x} \cdot \mathbf{y} = \langle \mathbf{x}\mathbf{y} \rangle_n = \sum_n x[n]y[n] = 0$.
- if all the vectors in a basis are orthogonal, it is called an *orthogonal* basis
- if the length of all the vectors in an orthogonal basis is equal to one (i.e., if they are all unit vectors) then it is an *orthonormal* basis (e.g., the standard Cartesian basis).
- If we relax the constraint that the vectors have to be independent, we have what is called an *overcomplete* basis (or sometimes ‘frame’).
  - Overcomplete bases are (strictly speaking) redundant for defining the vector space.

  - descriptions employing overcomplete bases are are not as succinct as those employing complete, or orthogonal bases
  - But in the noisy world of physical systems, this redundancy can prove invaluable for error correction and the efficient use of available resources

7.1.2 An example

7.1.2.1 Orthogonal representation

- (see slide) Let the vector $\mathbf{x}$ be written in a standard Cartesian basis (e.g., in two dimensions, $\mathbf{x} = [x_1, x_2] = x_1\mathbf{i} + x_2\mathbf{j}$).
- If we have a system that uses a different basis, say one in which both basis vectors (i.e., $\mathbf{i}$ and $\mathbf{j}$) are rotated by 45 degrees (call these $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$), we need to re-write $\mathbf{x}$ in this basis in order to relate it to other objects in that system.
- We write the *encoding* of $\mathbf{x}$ into this basis as

$$a_i = \langle \mathbf{x}\boldsymbol{\phi}_i \rangle_n \,, \tag{7.1}$$

  which is simply the dot product projection of the vector onto the new basis.

- To find out where the point $\mathbf{a} = [a_1, a_2] = a_1\boldsymbol{\phi}_1 + a_2\boldsymbol{\phi}_2$ lies in the original space, we can *decode* $\mathbf{x}$ from this new basis using

$$\mathbf{x} = \sum_i a_i \boldsymbol{\phi}_i. \tag{7.2}$$

  This way, we can move back and forth between orthonormal bases (see figure **??**a).

- Notice that if we substitute (7.1) into (7.2), we recover the same vector we originally encoded.
- Thus we can think of the coefficients a_i as 'representing', or 'carrying the same information', or 'encoding' the original coefficients x_i

7.1.2.2 Overcomplete representation

- Suppose that we do not know what the decoding basis is, but we do know what the encoding basis is
- We can guarantee that the encoding basis is overcomplete by using redundant, non-orthogonal encoders (see slide)
- Let us choose the encoding basis that consists of *three* vectors in the Cartesian plane, equally spaced at 120 degree intervals (i.e., $\tilde{\boldsymbol{\phi}}_1 = [\frac{\sqrt{3}}{2},\, \frac{1}{2}]$, $\tilde{\boldsymbol{\phi}}_2 = [-\frac{\sqrt{3}}{2},\, \frac{1}{2}]$, and $\tilde{\boldsymbol{\phi}}_3 = [0,\, -1]$).

- Since these encoders are not independent, (7.1) and (7.2) do not hold as they did before (which can be easily verified by substitution).
- So, we need to identify the set of vectors that span the space into which our vector $\mathbf{x}$ is being encoded (i.e., the decoding basis). To find this basis, we first assume, as before, that
$$\mathbf{x} = \sum_i a_i \boldsymbol{\phi}_i, \tag{7.3}$$
and
$$a_i = \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_i \right\rangle_n . \tag{7.4}$$
- We can now substitute (7.4) into (7.3) to give
$$\mathbf{x} = \sum_i \left\langle \mathbf{x}\tilde{\boldsymbol{\phi}}_i \right\rangle_n \boldsymbol{\phi}_i.$$
Writing the dot product explicitly, we get
$$\begin{aligned} x[m] &= \sum_{i,n} x[n]\tilde{\phi}_i[n]\phi_i[m] \\ &= \sum_n x[n] \sum_i \tilde{\phi}_i[n]\phi_i[m]. \end{aligned}$$
Since
$$x[m] = \sum_n x[n]\delta_{nm},$$
we know that
$$\delta_{nm} = \sum_i \tilde{\phi}_i[n]\phi_i[m]_i$$
or, in matrix notation,
$$\mathbf{I} = \tilde{\boldsymbol{\phi}}\boldsymbol{\phi}, \tag{7.5}$$
where $\mathbf{I}$ is the identity matrix, the columns of $\boldsymbol{\phi}$ are the $\boldsymbol{\phi}_i$, and the rows of $\tilde{\boldsymbol{\phi}}$ are the $\tilde{\boldsymbol{\phi}}_i$. We can thus solve for the decoders
$$\begin{aligned} \mathbf{I} &= \tilde{\boldsymbol{\phi}}\boldsymbol{\phi} \\ \tilde{\boldsymbol{\phi}}^T &= \tilde{\boldsymbol{\phi}}^T\tilde{\boldsymbol{\phi}}\boldsymbol{\phi}, \\ \boldsymbol{\phi} &= \left(\tilde{\boldsymbol{\phi}}^T\tilde{\boldsymbol{\phi}}\right)^{-1}\tilde{\boldsymbol{\phi}}^T, \end{aligned}$$
providing the desired vectors $\boldsymbol{\phi}_i$. Performing the calculations for this example gives $\boldsymbol{\phi}_1 = [\frac{\sqrt{3}}{3}, \frac{1}{3}]$, $\boldsymbol{\phi}_2 = [-\frac{\sqrt{3}}{3}, \frac{1}{3}]$, and $\boldsymbol{\phi}_3 = [0, -\frac{2}{3}]$, i.e.,
$$\boldsymbol{\phi}_i = \frac{2}{3}\tilde{\boldsymbol{\phi}}_i. \tag{7.6}$$

- These are the vectors that form the overcomplete basis for the vector space in which the a_i are coordinates. Thus, they are also the decoding vectors which help define the representation of $\mathbf{x}$ by the coefficients a_i.
- Vectors like these are called *biorthogonal* because together $\boldsymbol{\phi}$ and $\tilde{\boldsymbol{\phi}}$ act like an orthogonal basis.
- notably, the $\boldsymbol{\phi}$ and $\tilde{\boldsymbol{\phi}}$ we use do not, technically, satisfy the definitions provide here (because the encoding goes through a neural nonlinearity). *However*, they nearly satisfy these definitions, and the definitions provide a useful way of understanding what the relation between our encoders and decoders is.

7.1.3 Basis functions

- this discussion applies equally to basis *functions*
- Just as basis vectors define a vector space, so basis functions define a function space.
- Perhaps the most widely known basis functions are the sines and cosines. The Fourier decomposition is one that defines some function $f(x)$ in terms of coefficients, A_i, times a basis, $\cos(\omega_i x)$ and $\sin(\omega_i x)$ (see equation **??**). This is an orthonormal basis.
- just as in the vector case there are overcomplete basis functions as well. These are like the $\tilde{\phi}_i(\nu)$ and $\phi_i(\nu)$ we encountered in our characterization of function representation. Recall that these were used to characterize the tuning curve.
- So, the tuning curves are like a set of overcomplete basis functions for the space they represent.

7.2 Decomposing Γ

7.2.1 Matrix notation

- consider just the noise free $\gamma_{ij} = \langle a_i(\mathbf{x}) a_j(\mathbf{x}) \rangle_{\mathbf{x}}$.
- This is the Gram matrix (sometimes called a correlation matrix, but not really one - a correlation matrix should be a covariance matrix normalized by the standard deviations; a covariance matrix is $x_{ij} = (x_i - \mu_i)(x_j - \mu_j)$). That being said, we can think of it as measuring the similarity/overlap between the response of all neurons in the population. So, it tells us about the structure of the representation of the relevant vector space by the neurons.

- To rewrite our representation in vector notation, we assume the $\mathbf{x}$ is discretized by some step $\Delta\mathbf{x}$. Then the responses are:

$$\mathbf{A}^T = \begin{bmatrix} a_1(\mathbf{x}_{min}) & a_1(\mathbf{x}_{min}+\Delta_{\mathbf{x}}) & \cdots & a_1(\mathbf{x}_{max}-\Delta_{\mathbf{x}}) & a_1(\mathbf{x}_{max}) \\ a_2(\mathbf{x}_{min}) & & & & a_2(\mathbf{x}_{max}) \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{N-1}(\mathbf{x}_{min}) & & & & a_{N-1}(\mathbf{x}_{max}) \\ a_N(\mathbf{x}_{min}) & a_1(\mathbf{x}_{min}+\Delta_{\mathbf{x}}) & \cdots & a_1(x_{max}-\Delta_{\mathbf{x}}) & a_N(\mathbf{x}_{max}) \end{bmatrix}.$$

- This means we can write the estimate of $\mathbf{x}$ as:

$$\hat{\mathbf{X}}_{N_\Delta \times N_v} = \mathbf{A}_{N_\Delta \times N} \boldsymbol{\phi}_{N \times N_v},$$

  where the $\mathbf{A}$ matrix is the transpose of the neuron tuning curves, $\hat{\mathbf{X}}$ is a matrix in which each row is the estimate of $\mathbf{x}$ at the corresponding discretized value of $\mathbf{x}$, and $\boldsymbol{\phi}$ is a matrix in which each row is a decoding vector.

- we need the optimal $\boldsymbol{\phi}$ given a set of vectors, $\mathbf{X}$. So, take $\mathbf{X}$ as given and solve for $\boldsymbol{\phi}$ as follows:

$$\begin{aligned} \mathbf{X} &= \mathbf{A}\boldsymbol{\phi} \\ \mathbf{A}^T\mathbf{X} &= \mathbf{A}^T\mathbf{A}\boldsymbol{\phi} \\ (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X} &= \boldsymbol{\phi}. \end{aligned} \tag{7.7}$$

- Presuming we take the inverse of $\mathbf{A}^T\mathbf{A}$ in the right way, this is the same as minimizing the mean square error between the originally encoded value and our estimate. That is, we find

$$\begin{aligned} \gamma &= \mathbf{A}^T\mathbf{A} \qquad (7.8) \\ \Upsilon &= \mathbf{A}^T\mathbf{X}, \end{aligned}$$

  so,

$$\begin{aligned} \boldsymbol{\phi} &= \gamma^{-1}\Upsilon \qquad (7.9) \\ &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}. \end{aligned}$$

- We can now rewritethe estimate of $\mathbf{X}$ in matrix notation as

$$\begin{aligned} \hat{\mathbf{X}} &= \mathbf{A}\boldsymbol{\phi} \\ &= \mathbf{A}(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{X}. \end{aligned}$$

7.2.2 SVD on γ

- the major assumption we have made is that we can take the inverse of γ ‘in the right way’.

- because this matrix has no noise term, and because *some* tuning curves are likely to be similar for a large population, the matrix $\boldsymbol{\gamma}$ is likely to be singular, or nearly singular so it is *not* invertible
- There exists a general, and very powerful, technique for analyzing singular matrices called singular value decomposition (SVD).
- SVD decomposition of an $M \times N$ matrix, $\mathbf{B}$, results in three matrices whose product gives $\mathbf{B}$, i.e.,

  $$\mathbf{B}_{M\times N} = \mathbf{U}_{M\times N}\mathbf{S}_{N\times N}\mathbf{D}^T_{N\times N}.$$

  The matrix $\mathbf{S}$ is a diagonal matrix whose entries are called the *singular values* of $\mathbf{B}$.
- In the case when $\mathbf{B}$ is square and symmetrical (as with $\boldsymbol{\gamma}$), this simplifies to

  $$\boldsymbol{\gamma} \quad = \quad \mathbf{USU}^T,$$

  or equivalently, in summation notation,

  $$\gamma_{ij} = \sum_m U_{im} S_m U_{jm}.$$
- In the case where γ is singular (or nearly so), some elements of $\mathbf{S}$ are zero (or very small), so the inverse of $\mathbf{S}$ includes infinite (or very large) terms, meaning the inverse of γ is ill-defined (as expected for singular or near singular matrices). In this case, the SVD 'pseudo-inverse' is defined where for $S_i = 0$, the inverse is set to 0.
- Even in the case when a matrix is singular (or nearly so), SVD can be very informative. the columns of $\mathbf{U}$ whose corresponding singular values are non-zero form an orthonormal basis that spans the range and the corresponding zero elements form an orthonormal basis that spans the null space.
- This decomposition is useful for characterizing representation and transformation for a number of reasons.
  - First, as already mentioned, the relevant $\mathbf{U}$ matrix provides an orthogonal basis for both the range and nullity of $\boldsymbol{\gamma}$. Because $\boldsymbol{\gamma}$ tends to be singular, both bases are important.
  - Second, when a vector in Υ lies in the range of $\boldsymbol{\gamma}$, the SVD pseudo-inverse guarantees that the corresponding vector from $\boldsymbol{\phi}$ found from (7.9) *minimizes* the length of that $\boldsymbol{\phi}$ vector. This is important because given that $\boldsymbol{\gamma}$ is singular, there are an infinite number of solutions for $\boldsymbol{\phi}$. The solution that provides the shortest vector is a natural and compact choice from the set.

- Third, when a vector in Υ lies in the nullity of $\mathbf{A}$, the SVD pseudo-inverse guarantees that the best (in the least squares sense) ϕ given Υ will be found. In other words, this 'pseudo-inverse' minimizes the error. Thus, we can use SVD to find the optimal decoding functions, which we can now write as

$$\boldsymbol{\phi} = \mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}. \tag{7.10}$$

Given the properties of SVD, we know that this is the same solution we would find by constructing the error explicitly (i.e., $E = \left\langle \left[\mathbf{X} - \hat{\mathbf{X}}\right]^2 \right\rangle_{\mathbf{X}}$), taking the derivative, and setting it to zero, as we have previously done.

7.3 Possible transformations

7.3.1 Theory

- let's do a similar analysis to find an expression for the decoding vectors needed to decode any *transformation* of $\mathbf{x}$:

$$\begin{aligned} f(\mathbf{X}) &= \mathbf{A}\boldsymbol{\phi}^f \\ \mathbf{A}^T f(\mathbf{X}) &= \mathbf{A}^T\mathbf{A}\boldsymbol{\phi}^f \\ \boldsymbol{\phi}^f &= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T f(\mathbf{X}) \\ &= \boldsymbol{\gamma}^{-1}\Upsilon \end{aligned}$$

Performing SVD on $\mathbf{A}^T\mathbf{A}$ to find the inverse, as before, gives

$$\boldsymbol{\phi}^f = \mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T f(\mathbf{X}), \tag{7.11}$$

where $\boldsymbol{\phi}^f$ are the linear decoders for estimating the transformation $f(\mathbf{X})$.

- So, the representational decoder, $\boldsymbol{\phi}$, is found in the special case where $f(\mathbf{X}) = \mathbf{X}$.

- Notice that regardless of which transformation we need decoders for, we always perform SVD on the same matrix, $\gamma = \mathbf{A}^T\mathbf{A}$. This suggests that understanding the properties of γ can provide general insight into all possible decodings of the population, a_i.

- The singular values are useful because they tell us the *importance* of the corresponding $\mathbf{U}$ vector. There are a number of ways of thinking about 'importance' in this case.

  - related to the error that would result if we left a particular vector out of the mapping.
  - related to the variance of population firing along the vectors in the γ matrix.
  - being the amount of (independent) information about changes in population firing that can be extracted by looking only at data projected onto the corresponding $\mathbf{U}$ vector.

  - In general, we can think of the magnitude of the singular value as telling us how relevant the dimension defined by the corresponding $\mathbf{U}$ vector is to the identity of the matrix we have decomposed. Since the matrix we have decomposed is like the correlation matrix of the neuron tuning curves, the large singular values are most important for accounting for the structure of those correlations.

- Notice also that the vectors in $\mathbf{U}$ are orthogonal: they provide an (ordered) orthogonal *basis* for that matrix. This is very useful because the original γ matrix was generated by a non-ordered non-orthogonal basis; the neuron tuning curves.

- to understand this, define a point in the 'neuron space' (i.e., the space spanned by the overcomplete neuron tuning curves) as

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \ldots + a_N\mathbf{e}_N.$$

  In this notation, the vectors $\mathbf{e}_i$ serve as axes for the state space of the neural population. A point in this space is defined by the neuron firing rates from each neuron in the population (which, taken together, form the vector $\mathbf{a}$).

- Because the neural responses are non-independently driven by some variable, $\mathbf{x}$, only a *subspace* of the space spanned by the $\mathbf{e}_i$ vectors is ever *actually* occupied by the population.

- The γ matrix, because it tells us the correlations between all neurons in the population, provides us with the information we need to determine what that subspace is. When we find the $\mathbf{U}$ vectors in the SVD decomposition, we have characterized that subspace because those are the orthogonal vectors that span it.

- Let's see how we can use this to determine what functions can be computed by the particular encoding of $\mathbf{x}$ found in the a_i population:

$$\hat{\mathbf{X}} = \mathbf{A}\mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X},$$

  or, more simply

$$\hat{\mathbf{X}} = \chi\Phi, \tag{7.12}$$

  where

$$\chi = \mathbf{A}\mathbf{U},$$

  so

$$\chi_m(\mathbf{x}) = \sum_i a_i(\mathbf{x})U_{im}, \tag{7.13}$$

  and

$$\begin{aligned}\Phi &= \mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}\\ &= \mathbf{U}^T\mathbf{U}\mathbf{S}^{-1}\mathbf{U}^T\mathbf{A}^T\mathbf{X}\\ &= \mathbf{U}^T\boldsymbol{\phi},\end{aligned}$$

  so

$$\Phi_m = \sum_i U_{mi}\boldsymbol{\phi}_i.$$

- Notice that χ and Φ in (7.12) are rotated versions of $\mathbf{A}$ and ϕ respectively. Specifically, they are rotated into the coordinate system defined by $\mathbf{U}$. So we can think of $\mathbf{U}$ as the rotation matrix that aligns the first axis of the coordinate system along the dimension with the greatest variance in the encoding of $\mathbf{x}$, the second axis along the dimension with the second greatest variance, and so on. As a result, and as shown in appendix **??**, the

- the χ vectors also end up being orthogonal and ordered by importance. That is, they tell us what can be extracted, and how well it can be extracted, from the neural population.

- (see slide) We can think of the components of χ as *basis functions* of the space that includes the ensemble of transformations definable on $\mathbf{x}$ using the encoding in the population a_i. This is more evident in (7.13) where we write χ in summation notation. There it is quite obvious that $\chi_m(\mathbf{x})$ at a particular value of $\mathbf{x}$ is the neuron firing rates at that value of $\mathbf{x}$ projected onto the mth orthonormal $\mathbf{U}$ vector (see figure **??**).

- Whichever $\chi(\mathbf{x})$ functions have reasonably large associated singular values, are exactly the functions that we can do a good job of extracting from our encoding of the input space, $\mathbf{x}$. Of course, we can also extract any linear combinations of those $\chi(\mathbf{x})$ functions quite well. But, because these functions are ordered, the more useful the 'first' $\chi(\mathbf{x})$ function is for reconstructing some transformation, $f(\mathbf{x})$, the better we can extract that transformation, $f(\mathbf{x})$.

- In practice, we can look at the resulting $\chi(\mathbf{x})$ functions and determine what sort of basis we seem to have. For example, if the $\chi(\mathbf{x})$ look like sines and cosines, we have a Fourier basis.

7.3.2 Real populations

7.3.2.1 Linear population

- so, let's do this to a population encoding a scalar with broad functions (see slides)

- the result? a standard basis: One of the most common polynomial bases used in mathematics is the Legendre basis, $l_i(x)$, which is defined over the interval [-1,1] and results from the orthogonalization of x^n.[1] Scaled versions of the first five elements of this basis are plotted in figure **??**b.

- the similarity between $\chi_m(x)$ and $l_i(x)$ means that this neural population supports the extraction of functions that can be well-estimated using the standard Legendre basis. But the $\chi_m(x)$ functions are ordered by their singular values. Thus, the higher-order polynomial terms are not as well encoded by our population as the lower-order ones. So, computing functions that depend strongly on precise high-order terms will be prone to error.

[1]One way of expressing the basis is: $l_i(x) = \frac{(-1)^i}{2^i i!} \frac{d^i}{dx^i} \left[\left(1 - x^2\right)^i \right]$.

- This is a natural basis to be found from the tuning curves in the linear population. The tuning curves are very broad, and the polynomial basis is also very broad. These tuning curves are approximately linear, and the more linear basis functions are also the first ones. The Legendre polynomial basis is ordered by decreasing linearity so it should not be too surprising that this population supports the functions in precisely that order.

- However, none of this would be true if the population did not do a good job of evenly tiling the input space. If, for example, there were only high gain neurons, whose slopes were the same sign as their x-intercepts (i.e., if the 'on' and 'off' sub-populations were 'clumped' near x_{max} and x_{min} respectively), we would not expect the linear term to be better supported than the quadratic term. In this sense, the heterogeneity of the population helps it support the more natural ordering of the polynomial basis; clumping would defeat this ordering. Thus, this particular set of $\chi_m(x)$ functions does not just depend on the general 'shape' of the neuron tuning curves, but also on which neurons are included in a population, i.e., the degree of heterogeneity.

- Vectors: there are additional analyses we can perform for vectors of two or more dimensions.

- When computing transformations of populations encoding n-dimensional vectors, we must realize that there are additional cross terms (e.g., x_1x_2) that introduce variations in the encoding that are not present in scalar transformations. The expansions are now of the form

$$\begin{aligned} f(\mathbf{x}) &= c_0 + c_1x_1 + c_2x_2 + c_3x_1^2 + c_4x_2^2 + c_5x_1x_2 + \ldots \\ &= \sum_{l=0}^{N_{order}} \sum_{n=0}^{l} c_{n,l-n} x_1^n x_2^{l-n}, \end{aligned}$$

  where N_{order} is the highest order term in the transformation and l indexes the lth order terms in the expansion (all terms whose exponents *sum* to l are considered lth order terms). As we can see, the cross terms (i.e., $x_1^n x_2^{l-n}$) are quite common. In order to characterize how well an arbitrary function can be decoded from a representation of $\mathbf{x}$, we need to know how big the singular values of these cross terms are as well.

- A quick inspection of the singular values of the population reveals that all terms of a given order have singular values of approximately the same magnitude (see slide).

- But there is an exponential decay in the magnitude of the singular values as a function of the order of the polynomial. This means that lower-order functions are *significantly* better supported by these kinds of broadly tuned neural populations. That is, the ability to extract higher-order functions drops more quickly with an increase in the order of the function than compared to the scalar case.

- In fact, we can determine exactly how many singular values, N_S, there should be for each order, l, in the polynomial for a input space of size D by using the equation for determining l-combinations of a multi-set (See slide):

$$N_S(l,D) = \frac{(l+D-1)!}{l!(D-1)!}. \tag{7.14}$$

7.3.2.2 Gaussian population

- Now we can do the same on a gaussian population (See slide)
- we find some differences:
  - the basis looks like a fourier basis, but only over [-1,1]
  - the basis is better for encoding localized functions, and not good a broad ones - we can see this by looking at the singular values (see slide)
- In general, then, we can examine the $\chi_m(x)$ functions, which span the active state space of a population, to determine what kinds of transformations are and are not well-supported by that population. This can be extremely useful for constraining hypotheses about the kinds of transformations we expect to see in a *real* neural population once we have a sense of what kind of encoding is being used.

7.3.3 Noise

- we've been ignoring noise. but most everything stays the same. Rather than the encoding defining a precise subspace (the black line in the projection slide), we can think of it as defining a cloud (e.g. a tube in the projection slide).
- However, the noise doesn't scale with the singular values, it is isometric in the vector space. This means that small singular values are more greatly affected by noise. We can derive this.
- We know the error with noise can be written in the vector case as

$$\begin{aligned} E &= \left\langle \left[\mathbf{x} - \sum_i \left(a_i(\mathbf{x}) + \eta_i\right) \boldsymbol{\phi}_i \right]^2 \right\rangle_{\mathbf{x},\eta} \\ &= \left\langle \left[\mathbf{x} - \sum_i a_i(\mathbf{x}) \boldsymbol{\phi}_i \right]^2 + \sigma_\eta^2 \sum_i \boldsymbol{\phi}_i^2 \right\rangle_{\mathbf{x}}. \end{aligned} \tag{7.15}$$

We can now use our expression SVD expressions to give

$$E = \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m \right]^2 + \sigma_\eta^2 \sum_{i,j} \boldsymbol{\phi}_i \delta_{ij} \boldsymbol{\phi}_j \right\rangle_{\mathbf{x}}$$

$$
\begin{aligned}
&= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m\right]^2 + \sigma_\eta^2 \sum_{i,j} \boldsymbol{\phi}_i \sum_m U_{im} U_{mj} \boldsymbol{\phi}_j \right\rangle_{\mathbf{x}} \\
&= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m\right]^2 + \sigma_\eta^2 \sum_m \Phi_m^2 \right\rangle_{\mathbf{x}} . \qquad (7.16)
\end{aligned}
$$

- We minimize this error by taking the derivative of (7.16) and setting it to zero to get an expression for the optimal Φ functions under noise:

$$
\begin{aligned}
\frac{dE}{d\Phi_n} &= \left\langle 2 \left[\mathbf{x} - \sum_m \chi_m \Phi_m\right] (-\chi_n) + 2\sigma_\eta^2 \Phi_n \right\rangle_{\mathbf{x}} \\
0 &= -2 \langle \chi_n \mathbf{x} \rangle_{\mathbf{x}} + 2 \left\langle \sum_m \chi_m \chi_n \Phi_n \right\rangle_{\mathbf{x}} + 2\sigma_\eta^2 \Phi_n \\
\langle \chi_n \mathbf{x} \rangle_{\mathbf{x}} &= S_n \Phi_n + \sigma_\eta^2 \Phi_n \\
\Phi_n &= \frac{\langle \chi_n \mathbf{x} \rangle_{\mathbf{x}}}{S_n + \sigma_\eta^2} . \qquad (7.17)
\end{aligned}
$$

- We can use this expression to determine what the residual error will be (i.e., the expected error using these Φ functions).
- We can now determine the residual error as follows:

$$
\begin{aligned}
E_r &= \left\langle [\mathbf{x} - \hat{\mathbf{x}}]^2 \right\rangle_{\mathbf{x},\eta} \\
&= \left\langle \left[\mathbf{x} - \sum_m \chi_m \Phi_m\right]^2 \right\rangle_{\mathbf{x},\eta} \\
&= \left\langle \mathbf{x}^2 \right\rangle_{\mathbf{x},\eta} - 2 \left\langle \mathbf{x} \sum_m \chi_m \Phi_m \right\rangle_{\mathbf{x},\eta} + \left\langle \sum_m \chi_m \Phi_m \right\rangle_{\mathbf{x},\eta}^2 .
\end{aligned}
$$

Substituting the expression in (7.17) for Φ_m gives

$$
\begin{aligned}
E_r &= \left\langle \mathbf{x}^2 \right\rangle_{\mathbf{x},\eta} - 2 \left\langle \mathbf{x} \sum_m \chi_m \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}}{S_m + \sigma_\eta^2} \right\rangle_{\mathbf{x},\eta} + \left\langle \sum_m \chi_m \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}}{S_m + \sigma_\eta^2} \right\rangle_{\mathbf{x},\eta}^2 \\
&= \left\langle \mathbf{x}^2 \right\rangle_{\mathbf{x}} - 2 \sum_m \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}^2}{S_m + \sigma_\eta^2} + \sum_m (S_m + \sigma_\eta^2) \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}^2}{\left(S_m + \sigma_\eta^2\right)^2} \\
&= \left\langle \mathbf{x}^2 \right\rangle_{\mathbf{x}} - \sum_m \frac{\langle \chi_m \mathbf{x} \rangle_{\mathbf{x}}^2}{S_m + \sigma_\eta^2} .
\end{aligned}
$$

- We can see this how much the mth basis function, χ_m, reduces the error in our estimate under the influence of noise.

- Specifically, we know that as the singular value, S_m, approaches the value of the variance of the noise, σ_η^2, the corresponding mth element does not usefully contribute to the representation. This is because the S_m term acts to normalize the effects of the projection onto the non-normalized basis, χ_m.
- When the noise becomes near the magnitude of that normalization term (i.e., SNR = 1 or less), the projection onto the relevant χ_m becomes 'mis-normalized' and thus contributes incorrectly to the representation. That is, it will *introduce* error into the representation.
- This tells us that *those basis functions whose corresponding singular value is equal to or smaller than the noise, should not be used if we want a good representation*.
- So, the useful representational space is that space spanned by the basis functions, χ_m, whose corresponding singular values, S_m, are greater than the variance of the noise, σ_η^2, affecting a single neuron.
- So, we can simply 'lop off' some of the singular values when doing the inverse to get the same result as including a certain amount of noise in our calculation of Γ. This is approximately true, but we haven't done a careful analysis of this relation (between the amount of noise and the number of SVs perserved when inverting γ).

7.4 Heterogeneity

- One thing that becomes clearly important when doing these kinds of analysis is the precise nature of the tuning curves. In the book we show that heterogeneity is a useful balance between usefully tiling a space and ease of construction.
- Specifically, heterogeneity provides good reduction of error under noise (see slide). And, it is easy to construct compared to a perfect, lattice-like spacing of intercepts. (i.e., it's evolutionarily cheap).
- I haven't discussed representational capacity above, but both it and 'usefulness' (measured by resistance to noise) are good for heterogeneous populations. This is probably why real neural systems tend to have such tuning curves.