# Computing with population codes

Peter E. Latham
Gatsby Computational Neuroscience Unit, University College London
pel@gatsby.ucl.ac.uk

## Introduction

One of the few things in systems neuroscience we are fairly certain about is that information is encoded in population activity. For example, a population of neurons coding for the orientation of a bar ($\theta$), might have firing rates $r_i = f(\theta - \theta_i) + \text{noise}$. Here $f(\theta)$ is a smooth, bell shaped function and $\theta_i$ is the preferred orientation of neuron $i$.

Representing variables in populations codes, however, is only one step – just as important is *computing* with population codes. Sensorimotor transformations are a natural example: the brain receives information about the outside world; that information is represented in population activity at the sensory level; and to perform an action, such as reaching for an object, population codes in motor cortex must be generated to drive the appropriate joint movements. The transformation from the activity in sensory cortex to the activity in motor cortex is a computation based on population codes. Here we discuss how networks of neurons might implement such computations.

## Computing, invariance, and throwing away information

Before we start, it's worth saying a few words about computing in general. Many, if not most, of the computations carried out by the brain involve *throwing away* information. For example, when computing a function, say $z = \phi(x, y)$, information about $x$ and $y$ is thrown away: $z$ tells us nothing about the individual values of of $x$ and $y$. This is a simple kind of invariance: all values of $x$ and $y$ such that $z = \phi(x, y)$ are effectively identical. These values thus form a manifold (in this case a one-dimensional curve) in the full two-dimensional $x$-$y$ space. One can easily think of much more complex invariants. For example, the same face viewed from different angles, at different distances, in different lightings, and with various hair length, degree of sunburn, style of glasses, etc. is always recognized as belonging to the same person. Or, in terms of invariant manifolds, there are many different combinations of angle, distance, lighting, etc. that lead to the same outcome ("hi Fred"). Invariances like this, of course, live in high dimensions, and it is an open question how the brain computes them.

A second, less obvious, kind of invariance has to do with population activity. Because of neuronal noise, many different patterns of activity code for the same thing (e.g., the same value of a variable), and networks should be invariant with respect to these difference. In particular, they should do as good a job as possible extracting the relevant part of the activity (e.g., the position of the peak of a tuning curve) and ignoring the irrelevant part (e.g., the overall amplitude). The goal of this chapter is to develop an understanding of how networks of neurons handle both kinds of invariance.

## Computing functions with networks of neurons: a general algorithm

Networks of neurons implement input-output functions: they take as input population activity from one set of neurons and produce as output population activity on another. If the input and output populations code for a set of variables, then we can think of the networks

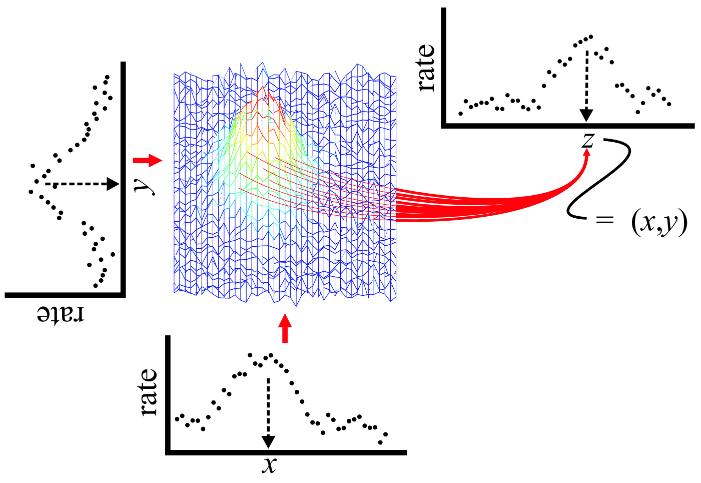

**Figure 1**: Feedforward network that computes $z = \phi(x, y)$. The input layers consist of populations that code for $x$ and $y$, and thus peak near $x$ and $y$. Feedforward connections to the intermediate layer, combined with recurrent connections in that layer, produced a bump at position $(x, y)$. Connections from the position of that bump to $z = \phi(x, y)$ in the output layer completes the network.

as computing functions of those variables. In this section we describe how to construct networks that can compute essentially any smooth function.

For simplicity we will assume that on the input side there are two populations, each coding for one variable, and on the output side there is one population, also coding for one variable (Fig. 1). As discussed in the introduction, we will also assume that the activity of each neuron is described by one number, $r$ (which we generally think of as firing rate). The value of $r$ is set by both a tuning curve and noise. Letting $x$ and $y$ refer to the input variables, the activity of the $i^{\text{th}}$ neuron in each population is thus given by

$$\begin{aligned} r_i^x &= f_x(x - x_i) + \eta_i^x && \text{(1a)} \\ r_i^y &= f_y(y - y_i) + \eta_i^y \,. && \text{(1b)} \end{aligned}$$

Here $r_i^x$ and $r_i^y$ are the firing rates of neuron $i$ of the $x$ and $y$ populations, $f_x$ and $f_y$ are tuning curves (often taken to be Gaussian), $x_i$ and $y_i$ are referred to as *preferred values*, and $\eta^x$ and $\eta^y$ represent noise, which is zero mean and typically correlated. The peaks of the tuning curves correspond approximately to the encoded variables (Fig. 1). Although the horizontal axis is index (which labels neuron), we can, and often do, also think of it as labeling the value of the encoded variable. Thus, the statement "the population peaks at $x$", is shorthand for "the preferred value of the neuron with the highest firing rate is $x$".

If the population activity given in Eq. (1) is fed into a network, the output is a new population which codes for the variable $z$,

$$r_i^z = f_z(z - z_i) + \eta_i^z \,. \tag{2}$$

The noise, $\eta^z$, is partially inherited from the input and is partially generated from internal noise in the network (stochastic transmitter release, for example). The value of $z$ in Eq. (2) (the approximate position of the peak of the population activity) depends on $x$ and $y$ (the approximate peaks of the two input populations). Thus, this network implements some function, which for definiteness we call $\phi$: $z = \phi(x, y)$. Our goal is to understand how to design a network that can implement an arbitrary function.

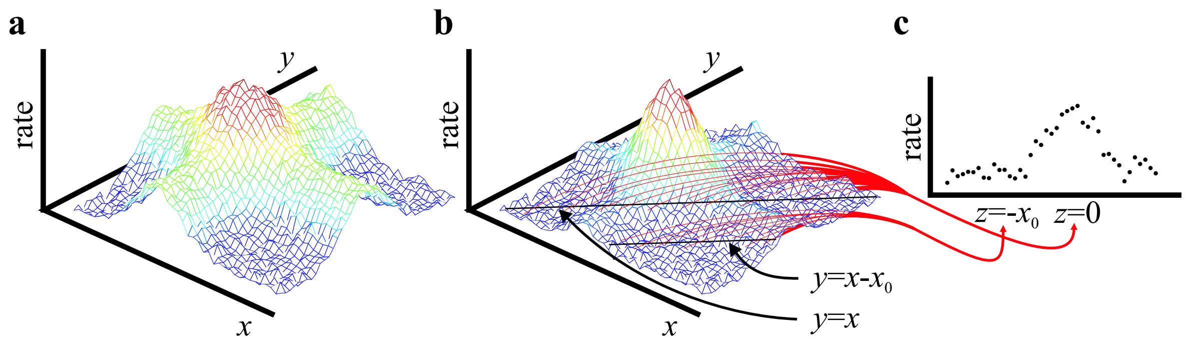

**Figure 2**: Intermediate layer in response to inputs like the ones shown in Fig. 1. **a**. Pattern of activity when there are only feedforward connections from the input. **b**. Pattern of activity when Mexican hat recurrent connectivity is introduced. Short range excitation enhances the peak; long range inhibition suppresses the ridges. **c**. Connectivity that implements the function $z = y - x$: neurons in diagonal strips in the intermediate layer connect preferentially to the appropriate neurons in the output layer. Two strips are shown, one corresponding to $y - x = 0$ and the other to $y - x = -x_0$.

The design we use is based on the algorithm described by (Pouget and Sejnowski 1997), which takes advantage of the properties of basis function networks. The main idea, which is illustrated in Fig. 1, is the following: Consider two population codes peaking at $x$ and $y$. Let these two populations project onto an intermediate network consisting of a two-dimensional array of neurons. With the appropriate connectivity (which we discuss below), the intermediate network can be made to peak at $(x, y)$. If the neurons associated with that peak project to the output layer such that they most strongly activate the neurons whose preferred value is $\phi(x, y)$, then the network will compute the function $z = \phi(x, y)$.

To understand this architecture in more detail, we construct this network step by step, starting with simplified connectivity and working up to the complete connectivity that will allow the network to compute the function $\phi$. Denoting the firing rates of the neurons in the intermediate layer $r_{ij}$, where $i$ and $j$ label position in a two-dimensional array, we first consider a network with only feedforward connections,

$$r_{ij} = \psi\left(\sum_k J_{ik} r_k^x + \sum_l J_{jl} r_l^y\right)$$

where the feedforward connection matrix, $J_{ij}$, is (relatively sharply) peaked at $i = j$ and $\psi$ is a sigmoidal function of its argument (e.g., $\psi(u) = \nu_0(1 + \exp(u_0 - u))^{-1}$). As illustrated in Fig. 2a, this feedforward connectivity produces perpendicular ridges of activity in the intermediate layer, with a peak where the ridges cross.

The second step is to get rid of the ridges so that we have only a bump of activity at position $(x, y)$. One way to do this is to add "Mexican hat" recurrent connectivity, which has the property that nearby neurons excite each other and distant neurons inhibit each other. Letting $W_{ij,kl}$ be the connection strength from neuron $kl$ to neuron $ij$, a standard implementation of Mexican hat connectivity is

$$W_{ij,kl} = W_E \exp\left(-[(i-k)^2 + (j-l)^2]/2\sigma_E^2\right) - W_I \exp\left(-[(i-k)^2 + (j-l)^2]/2\sigma_I^2\right).$$

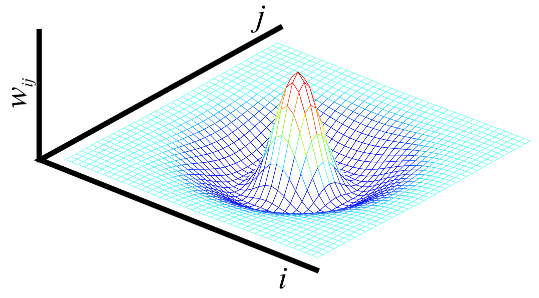

**Figure 3**: Weight matrix corresponding to Mexican hat connectivity.

The parameters $W_E, W_I, \sigma_E$ and $\sigma_I$ control the shape of the connectivity matrix; if we impose the conditions $W_E > W_I$ and $\sigma_E < \sigma_I$, then it will have the shape given in Fig. 3.

Introducing recurrent connectivity means that the network can no longer be characterized by a simple static nonlinearity, but instead must evolve in time. For this we use simple first order kinetics,

$$\frac{dr_{ij}}{dt} = \psi\left(\sum_{kl} W_{ij,kl} r_{kl} + \sum_k J_{ik} r_k^x + \sum_l J_{jl} r_l^y\right) - r_{ij} \,. \tag{3}$$

The effect of these recurrent connections is to enhance regions of highest firing rates – the intersection of the two grids in Fig. 2a – and suppress lower firing rates. This results in a bump of activity at position $(x, y)$. More accurately, the firing rate $r_{ij}$ is large for indices such that $x_i \approx x$ and $y_i \approx y$ where, recall, $x$ and $y$ correspond to the peaks of the input populations. A plot of the activity in the intermediate layer is shown in Fig. 2b.

The third step is to add the connectivity from the intermediate layer to the output layer. Conceptually, this is relatively straightforward: neurons in the intermediate layer whose preferred value is near $(x_i, y_j)$ should excite neurons in the output layer whose preferred value is near $\phi(x_i, y_j)$. Formally, we write

$$r_i^z = \sum_{jk} G\Big(z_i - \phi(x_j, y_k)\Big) r_{jk} \tag{4}$$

where $G(x)$ is a unimodal function with a peak at zero (e.g, a Gaussian) and the firing rates in the intermediate layer, $r_{jk}$, are taken to be the equilibrium rates ($t \to \infty$ in Eq. (3)). This connectivity, along with the pattern of activity it produces in the output layer, is shown in Fig. 2c for the function $\phi(x, y) = y - x$. For this function, neurons along diagonal strips in the intermediate layer connect preferentially to the appropriate neurons in the output layer. Specifically, neurons in the strip $y = x + a$ connect preferentially to the neurons in the output layer that code for $a$.

Manipulating the feedforward connectivity – adjusting $\phi(x, y)$ in Eq. (4) – allows any sufficiently smooth function $\phi$ to be computed. Moreover, networks of this type are not restricted to computations of only two variables; by increasing the dimension of the intermediate layer, functions of multiple variables can be computed. Thus, this network architecture provides us with a general algorithm for computing smooth functions of multiple variables.

## Efficient computing; qualitative analysis

In spite of the power and flexibility of these kinds of networks, there is room for improvement in at least two ways. First, the network architecture was purely feedforward, so information flowed only from the input to the output. In many situations, however, the distinction

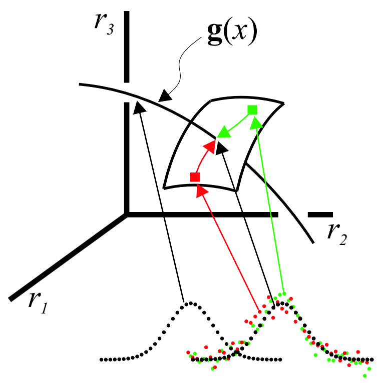

**Figure 4**: Cartoon of an attractor network; the three dimensions represent the full $n$-dimensional space. The line labeled $\mathbf{g}(x)$ represents a line attractor; each point on it corresponds to a smooth bump. The plane perpendicular to this line is invariant under network dynamics. Shown are two initial conditions (red and green squares) that correspond to two different realizations of noisy populations codes (red and green populations). Under the network dynamics, both initial conditions evolve to the same point on the line attractor (red and green arrows on the invariant manifold), and thus provide the same estimate, $\hat{x}$, for the encoded variable.

between input and output is not so clear. For example, when computing the location of an object relative to your head based on visual cues, the inputs are the location of the object on your retina and the location of your eyes in your head, and the output is the location of the object relative to your head. If the object makes noise, however, neurons in auditory cortex code directly for the location of the object, and so act as input. Second, the networks were not necessarily optimal; there was nothing preventing them from throwing away large amounts of information (an issue we will return to below). Both of these considerations are related to *efficient* computation – computation that makes use of all the available information. In this section we discuss how to build networks that compute efficiently.

Our starting point is the class of networks described above, augmented in two ways. First, we add feedback connections from the output layers to the intermediate layer and from the intermediate layer to the input layers, and we add recurrent connections within the input layers. This puts the input and output layers on equal footing, so we call all layers except the intermediate layer "input layers". Second, we construct the networks so that they exhibit multi-dimensional attractors; that is, so that the input and intermediate layers can display bumps of activity in the absence of input.

Replacing the largely feedforward network analyzed in the previous section with an attractor network is a key step, and the reason we do it was alluded to in the section "Computing, invariance, and throwing away information": we want to construct networks that pay attention only to the relevant activity. To see why an attractor network might do this, consider for a moment just a single population that codes for one variable, say $x$. In this case there is a unique mapping from the population activity, $\mathbf{r}^x$, to an estimate of the encoded variable, denoted $\hat{x}$. In other words, there is some function $h$ such that $\hat{x} = h(\mathbf{r}^x)$. This is a rather extreme kind of invariance: assuming there are $n$ neurons, then there is an $(n-1)$-dimensional manifold that maps to the same value of $\hat{x}$. A very natural way to implement this in a network of neurons is to ensure that the dynamics collapses onto a one-dimensional manifold; that is, ensure that the network implements a one-dimensional attractor. Such an attractor has the property that all the points on an $(n-1)$-dimensional manifold are attracted to the same point on the attractor. That point can thus code directly for $\hat{x}$, and if the network is designed optimally it will provide an optimal estimate of $\hat{x}$. This idea is illustrated in Fig. 4, and in the remainder of the section we expand on it from a quantitative (and fairly high-level) perspective.

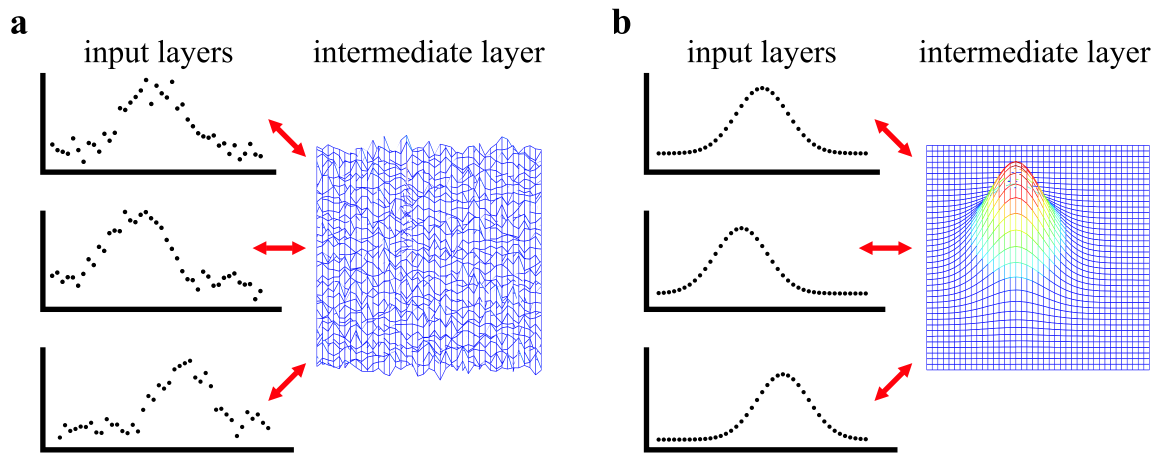

**Figure 5**: Evolution from noisy to smooth hills. **a.** Initial conditions: noisy populations activities. Note that information has not yet propagated into the intermediate layer, so the neurons in that layer fire at the background rate. **b.** After a long time the network has evolved to a set of smooth hills, including a smooth bump in the intermediate layer. The peaks act as network estimates of the encoded variables.

**Efficient computing; quantitative analysis**

To see how an attractor network can be used to estimate variables encoded in population activity, consider the following scenario. At time $t = 0$, the network is initialized, typically by transient input. This initialization, which provides all the information about the encoded variables that the network will receive, results in a noisy pattern of activity like the one shown in Fig. 5a. After that the network evolves deterministically[1] until it settles into a state with smooth hills of activity in both the input and intermediate layers, as shown in Fig. 5b. The peaks of these hills act as network estimates of the variables encoded in the initial conditions. The question we are interested in is: how accurate are those estimates? Or, more specifically, what is the variance of the estimates?

This question was first considered by (Deneve, Latham, and Pouget 2001), and discussed in gory detail by (Latham, Deneve, and Pouget 2003). Here we give a simplified answer for the case of a single variable encoded in population activity (although our analysis applies to multiple variables). This corresponds to the situation shown in Fig. 6: a noisy hill of activity encoding a single variable, $x$, relaxes under the intrinsic network dynamics to a smooth hill whose peak is an estimate of the encoded variable. Because of trial-trial variability, the smooth peak is never in the same place twice. What we do now is compute the variance of the position of the peak; that is, the variance of the network estimate of the encoded variable.

This analysis requires more advanced math than we have been using so far. For a somewhat expanded (but also more general, and thus more difficult) exposition see (Latham, Deneve, and Pouget 2003). A quick review of all that linear algebra that used to put you to sleep would also be useful.

Our starting point is network dynamics of the form

[1]The deterministic evolution is an approximation, but it is useful in that it allows detailed analysis of the network. See (Latham, Deneve, and Pouget 2003) for a further discussion of this point.

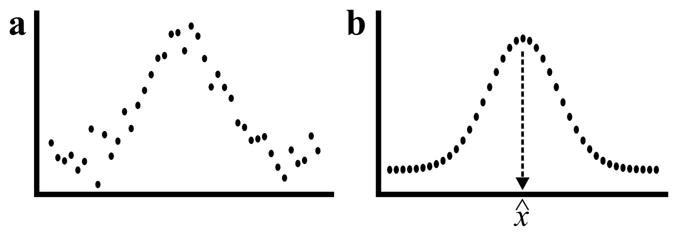

**Figure 6**: Same as Fig. 5 but for a single hill of activity. **a.** Initial conditions: a noisy hill of activity. **b.** After a long time the network has evolved to a smooth hill. The position of the peak is the network estimator, $\hat{x}$, of the encoded variable.

$$\frac{d\mathbf{r}}{dt} = \mathbf{H}(\mathbf{r}) - \mathbf{r}\,. \tag{5}$$

Here $\mathbf{r}$ represent the population activity encoding the variable $x$ (since there is only one population, we drop the superscript that used to appear on $\mathbf{r}$) and $\mathbf{H}(\mathbf{r})$ contains all the information about the network dynamics. This function is in principle arbitrary, but in practice it typically has the form $H_i(\mathbf{r}) = \psi\left(\sum_j W_{i-j} r_j\right)$ where, as above, $\psi$ is sigmoidal. A key aspect of this network is that it admits an attractor, and for the simple one-population model we are considering here it admits a line attractor. What this means is that there is some function $\mathbf{g}(x)$ that satisfies

$$\mathbf{g}(x) = \mathbf{H}\big(\mathbf{g}(x)\big)\,.$$

For example, $\mathbf{g}(x)$ might satisfy the equation $g_i(x) = \psi\left(\sum_j W_{i-j} g_j(x)\right)$ where $g_i(x)$ versus $i$ is a smooth hill of activity.

As in Eq. (1), the activity is initialized according to a set of tuning curves, $r_i = f(x - x_i) + \eta_i$. It is convenient to write this in vector notation, and also to explicitly take into account the fact that the network is initialized with noisy activity,

$$\mathbf{r}(t=0) = \mathbf{f}(x) + \mathbf{N} \tag{6}$$

where $\mathbf{f}(x) \equiv (f(x-x_1), f(x-x_2), ...)$ is the set of tuning curves and $\mathbf{N} \equiv (\eta_1, \eta_2, ...)$ is the noise. The noise has zero mean, $\langle \mathbf{N} \rangle = 0$ (angle brackets represent an average over trials), and covariance matrix $\mathbf{R}$,

$$\langle \mathbf{N}\mathbf{N} \rangle = \mathbf{R}\,. \tag{7}$$

We can now explicitly state the problem we are interested in: If a network that evolves according to Eq. (5) is initialized according to Eq. (6), it will evolve onto a point on the attractor; that is, in the limit $t \to \infty$, $\mathbf{r}(t)$ will approach $\mathbf{g}(\hat{x})$. Under this scenario, $\hat{x}$ is the network estimate of the encoded variable. Because of the noise, $\hat{x}$ will vary from trial to trial. However, we will assume the network is unbiased, so that when $\hat{x}$ is averaged over many trials it is equal to $x$: $\langle \hat{x} \rangle = x$. The first question we wish to answer is: what is the variance of $\hat{x}$? The second is: how does that variance compare to the variance of an optimal estimator?

To answer these questions we will assume that the noise is small, linearize the equations of motion around an equilibrium on the attractor, and solve explicitly for the variance. The only subtlety is deciding which point on the attractor to linearize around. It turns out to be convenient to linearize around $\hat{x}$, so we write

$$\mathbf{r}(t) = \mathbf{g}(\hat{x}) + \delta\mathbf{r}(t) . \tag{8}$$

The reason for this choice is that as $t \to \infty$ the network evolves to $\mathbf{g}(\hat{x})$ (i.e., $\mathbf{r}(t = \infty) = \mathbf{g}(\hat{x})$), which in turn means that $\delta\mathbf{r}(\infty) = 0$. This will greatly simplify our analysis.

To derive an expression for the time evolution of $\delta\mathbf{r}(t)$, we insert Eq. (8) into Eq. (5) and keep only terms up to first order in $\delta\mathbf{r}(t)$. This results in the evolution equation

$$\frac{d\delta\mathbf{r}}{dt} = \mathbf{J}(\hat{x}) \cdot \delta\mathbf{r} . \tag{9}$$

Here $\mathbf{J}(\hat{x})$ is the Jacobian,

$$J_{ij}(\hat{x}) \equiv \frac{\partial H_i(\mathbf{g}(\hat{x}))}{\partial g_j(\hat{x})} - \delta_{ij}$$

where $\delta_{ij}$ is the Kronecker delta ($\delta_{ij} = 1$ if $i = j$ and 0 otherwise), and "$\cdot$" indicates a dot product ($\mathbf{A} \cdot \mathbf{u} \equiv \sum_j A_{ij} u_j$). Equation (9) has the solution

$$\delta\mathbf{r}(t) = \exp[\mathbf{J}(\hat{x})t] \cdot \delta\mathbf{r}(0) . \tag{10}$$

To simplify this equation we use the expansion

$$\mathbf{J}(\hat{x}) = \sum_k \lambda_k(\hat{x}) \mathbf{v}_k(\hat{x}) \mathbf{v}_k^\dagger(\hat{x})$$

where $\mathbf{v}_k(\hat{x})$ and $\mathbf{v}_k^\dagger(\hat{x})$ are the eigenvectors and adjoint eigenvectors of $\mathbf{J}(\hat{x})$, respectively, with eigenvalue $\lambda_k(\hat{x})$. Using this expansion, Eq. (10) can be rewritten

$$\delta\mathbf{r}(t) = \sum_k \exp[\lambda_k(\hat{x})t] \mathbf{v}_k(\hat{x}) \mathbf{v}_k^\dagger(\hat{x}) \cdot \delta\mathbf{r}(0) . \tag{11}$$

Since the network dynamics admits a line attractor, one eigenvalue is zero and the rest have negative real part. Thus, in the limit $t \to \infty$, only one term in Eq. (11) survives – the one corresponding to the zero eigenvalue. Letting $k = 0$ label this eigenvalue ($\lambda_0 = 0$; $\mathrm{Re}\{\lambda_{k \neq 0}\} < 0$), we have

$$\delta\mathbf{r}(\infty) = \mathbf{v}_0(\hat{x}) \mathbf{v}_0^\dagger(\hat{x}) \cdot \delta\mathbf{r}(0) . \tag{12}$$

We now use the fact that $\mathbf{r}(\infty) = \mathbf{g}(\hat{x})$, which implies that $\delta\mathbf{r}(\infty) = 0$ (see discussion above). Combining this with Eq. (12) and using Eqs. (6) and (8) to express $\delta\mathbf{r}(0)$ in terms of $\mathbf{f}$ and $\mathbf{g}$, we find that

$$\mathbf{v}_0^\dagger(\hat{x}) \cdot [\mathbf{f}(x) - \mathbf{g}(\hat{x}) + \mathbf{N}] = 0\,. \tag{13}$$

To solve this equation we assume that the noise is small, let $\hat{x} = x + \delta x$, and expand Eq. (13) to first order in both $\delta x$ and the noise. This results in the expression

$$\mathbf{v}_0^\dagger(x) \cdot [\mathbf{f}(x) - \mathbf{g}(x)] + \mathbf{v}_0^\dagger(x) \cdot \mathbf{N} + \delta x \partial_{\hat{x}} \Big\{ \mathbf{v}_0^\dagger(\hat{x}) \cdot [\mathbf{f}(x) - \mathbf{g}(\hat{x})] \Big\}_{\hat{x}=x}\,. \tag{14}$$

For the network to be unbiased, $\delta x$ must vanish in the limit $\mathbf{N} \to 0$. This happens only if $\mathbf{v}_0^\dagger(x) \cdot [\mathbf{f}(x) - \mathbf{g}(x)] = 0$, so we will assume that this holds (in practice it must be checked on a case-by-case basis). Using this condition, the second term in Eq. (14) reduces to $-\delta x \mathbf{v}_0^\dagger(x) \cdot \mathbf{f}'(x)$ where a prime denotes a derivative. Thus, Eq. (14) becomes

$$\delta x = \frac{\mathbf{v}_0^\dagger(x) \cdot \mathbf{N}}{\mathbf{v}_0^\dagger(x) \cdot \mathbf{f}'(x)}\,.$$

Since we are assuming zero mean noise, $\langle \delta x \rangle = 0$. The variance of $\delta x$ is then given by

$$\langle \delta x^2 \rangle = \frac{\mathbf{v}_0^\dagger(x) \cdot \mathbf{R} \cdot \mathbf{v}_0^\dagger(x)}{[\mathbf{v}_0^\dagger(x) \cdot \mathbf{f}'(x)]^2} \tag{15}$$

where $\mathbf{R}$, the noise covariance matrix, is defined in Eq. (7).

Equation (15) is important because it tells us that the efficiency of the network depends only on the adjoint eigenvector of the linearized dynamics whose eigenvalue is zero. Moreover, we can use this expression to find the adjoint eigenvector that minimizes the variance: differentiating Eq. (15) with respect to $\mathbf{v}_0^\dagger(x)$ and setting the resulting expression to zero, we find that the variance of the network is minimized when

$$\mathbf{v}_0^\dagger(x) \propto \mathbf{R}^{-1} \cdot \mathbf{f}'(x)\,. \tag{16}$$

To satisfy Eq. (16) – and thus find the optimal network – we most modify network parameters, as it is those parameters that determine $\mathbf{v}_0^\dagger(x)$. Note that we can adjust parameters until the network is optimal *without running any simulations*, although in practice the line attractor, $\mathbf{g}(x)$ (which also depends on network parameters), must be found numerically.

How well does the network compare to an optimal estimator. To answer that, we need to find the variance of the optimal network, which we do by inserting Eq. (16) into (15). This yields

$$\langle \delta x^2 \rangle = \frac{1}{\mathbf{f}'(x) \cdot \mathbf{R}^{-1} \cdot \mathbf{f}'(x)}\,.$$

This expression should be compared to the Cramér-Rao bound, which is $\langle \delta x^2 \rangle \geq 1/I$ where $I$ is the Fisher information. For a Gaussian distribution with covariance matrix $\mathbf{R}$, the Fisher information is given by (Abbott and Dayan 1999)

$$I = \mathbf{f}'(x) \cdot \mathbf{R}^{-1} \cdot \mathbf{f}'(x) + \frac{1}{2}\mathrm{Tr}\{\mathbf{R}^{-1} \cdot \partial_x \mathbf{R} \cdot \mathbf{R}^{-1} \cdot \partial_x \mathbf{R}\}\,.$$

If the second term is small, then the network comes close to the Cramér-Rao bound; otherwise it doesn't. Whether or not networks of the type discussed here are optimal, then, depends on the correlational structure. What the correlational structure actually is in realistic networks is an active area of research.

**Summary**

Our main thesis is that networks compute with population codes by taking as input one set of noisy population codes and producing as output another set. The art is figuring out how to build networks with interesting input/output relations and small information loss. Qualitatively, this means building networks such that the output population codes peak in the right places and the neurons in those populations aren't too noisy.

In this chapter we examined two related classes of networks. The first was a feedforward network that used basis functions; the second was an extension of the first, in which sufficient feedback and recurrent connections were added so that the network implemented a multi-dimensional attractor. The use of attractor networks allowed us to derive optimality conditions that told us when networks can compute with minimum information loss.

The networks discussed here work well in a few dimensions (when the input and output populations code for a few variables), but fail when the dimensionality becomes large, somewhere on the order of ten or so. This is because the number of neurons in the input layer increases with the dimension. Thus, networks of the type described here are not adequate to solve the really high-dimensional problems, such as those we encounter in vision or audition. Either new paradigms for computations will have to be invented, or we will have to figure out how to reduce high-dimensional problem to low-dimensional ones that can be solved using the network architecture described here.

## References

Abbott, L. and P. Dayan (1999). The effect of correlated variability on the accuracy of a population code. *Neural Comput. 11*, 91–101.

Deneve, S., P. Latham, and A. Pouget (2001). Efficient computation and cue integration with noisy population codes. *Nature Neurosci. 4*, 826–831.

Latham, P., S. Deneve, and A. Pouget (2003). Optimal computation with attractor networks. *J. Physiol. Paris 97*, 683–694.

Pouget, A. and T. Sejnowski (1997). Spatial transformations in the parietal cortex using basis functions. *J. Cog. Neurosci. 9*, 222–237.