Biol Cybern (2011) 104:251–262
DOI 10.1007/s00422-011-0433-y

ORIGINAL PAPER

Biological Cybernetics

Normalization for probabilistic inference with neurons

Chris Eliasmith · James Martens

Received: 22 June 2010 / Accepted: 19 April 2011 / Published online: 14 May 2011

Abstract Recently, there have been a number of proposals regarding how biologically plausible neural networks might perform probabilistic inference (Rao, Neural Computation, 16(1):1–38, 2004; Eliasmith and Anderson, Neural engineering: computation, representation and dynamics in neurobiological systems, 2003; Ma et al., Nature Neuroscience, 9(11):1432–1438, 2006; Sahani and Dayan, Neural Computation, 15(10):2255–2279, 2003). To be able to repeatedly perform such inference, it is essential that the represented distributions be appropriately normalized. Past approaches have considered normalization mechanisms independently of inference, often leaving them unexplored, or appealing to a notion of divisive normalization that requires pooling across many neurons. Here, we demonstrate how normalization and inference can be combined into an appropriate connection matrix, eliminating the need for pooling or a division-like operation. We algebraically demonstrate that such a solution is available regardless of the inference being performed. We show that such a solution is relevant to neural computation by implementing it in a recurrent spiking neural network.

Keywords Neural computation · Probabilistic inference · Spiking network · Attractor network · Normalization · NEF

Electronic supplementary material The online version of this article (doi:10.1007/s00422-011-0433-y) contains supplementary material, which is available to authorized users.

C. Eliasmith (✉)
Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON N2L 3G1, Canada
e-mail: celiasmith@uwaterloo.ca

J. Martens
Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada
e-mail: james.martens@gmail.com

1 Introduction

There have been several suggestions for how to implement probabilistic inference in neurons (Rao 2004; Eliasmith and Anderson 2003; Ma et al. 2006; Sahani and Dayan 2003). However, these proposals do not satisfactorily address the issue of how the resulting inferred distributions are appropriately normalized. Most often, normalization is left to a separate mechanism, such as division by pooled activity, which may not be an appropriate mechanism in many circumstances (such as rapid feedforward inference).

Correct normalization is crucial because if each layer in a multilayer network performs inference, or if there are recurrent connections in a network which perform inference, any lack of normality will accumulate over repeated inferences resulting in a non-probabilistic function representation (Rao 2004; Ma et al. 2006). This may allow the representation to be lost in background noise because its amplitude is too low, or may result in saturation of neurons which represent the result because the norm will become too large. In either case, interpretation of the resulting representation as a probability density will not be warranted. In short, because neurons have a finite dynamic range, arbitrarily large or small signals cannot be represented with consistent accuracy. Mis-normalization, coupled with repeated inference, drives the system to one of these regimes.

In this article, we present a solution to this problem for approaches that explicitly represent the probability density which inference is performed on (Rao 2004; Eliasmith and Anderson 2003; Sahani and Dayan 2003). This solution, unlike divisive normalization, does not rely on a nonlinear operation (division), or on pooling activity over a population of neurons. Instead, it appropriately modifies the inference transformation performed by the connection weights to preserve the integral of the represented function. In past work,

we have shown how the Neural Engineering Framework (NEF), can be used to perform arbitrary statistical inference (Eliasmith and Anderson 2003). Consequently, we demonstrate our solution using the NEF method for performing inference in spiking neurons.

After briefly introducing the NEF formalism, we present the relevant mathematical background and solve the problem of normalization purely algebraically. We then demonstrate that the solution is neurally relevant by applying it to the NEF characterization of statistical inference. We conclude by discussing the consequences of this solution to the normalization problem, and compare and contrast it with divisive normalization.

2 The neural engineering framework (NEF)

This section briefly summarizes the relevant methods described in Eliasmith and Anderson (2003). The following three principles describe the NEF approach:

1. Neural representations are defined by the combination of nonlinear encoding (exemplified by neuron tuning curves, and neural spiking) and weighted linear decoding (over populations of neurons and over time).
2. Transformations of neural representations are functions of the variables represented by neural populations. Transformations are determined using an alternately weighted linear decoding.
3. Neural dynamics are characterized by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analyzed using control theory.

These principles have been applied to constructing models of a wide variety of neural systems, including the barn owl auditory system (Fischer 2005), the rodent navigation system (Conklin and Eliasmith 2005), escape and swimming control in zebrafish (Kuo and Eliasmith 2005), working memory systems (Singh and Eliasmith 2006), and others. Here, we consider each principle in the context of performing statistical inference.

2.1 Representation

Consider a population of neurons whose activities $a_i(\mathbf{x})$ encode some M-dimensional vector, $\mathbf{x} = [x_1, \ldots, x_M]$. These activities can be written

$$a_i(\mathbf{x}) = G_i\left[J_i(\mathbf{x})\right], \tag{1}$$

where G_i is the nonlinear function describing the neuron's response function, and $J_i(\mathbf{x})$ is the current entering the soma. The somatic current is defined by

$$J_i(\mathbf{x}) = \alpha_i \langle \mathbf{x} \cdot \mathbf{e}_i \rangle_m + J_i^{\text{bias}} \tag{2}$$

where $J_i(\mathbf{x})$ is the current in the soma, α_i is a gain and conversion factor, $\mathbf{x}$ is the vector variable to be encoded, $\mathbf{e}_i$ is the encoding vector which picks out the "preferred stimulus" of the neuron, and J_i^{bias} is a bias current that accounts for background activity. Throughout the article, the notation $\langle \cdot \rangle_m$ indicates the dot product (or integral) over the indicated elements (here just the vector elements). This equation provides a standard description of the current arriving at the soma of neuron i as a result of presenting a stimulus $\mathbf{x}$.

The nonlinearity G_i which describes the neuron's activity as a result of this current is determined by physiological properties of the neuron(s) being modeled. The result of applying G_i to the soma current $J_i(\mathbf{x})$ is neural activity in the form of a pattern of action potentials, or "spikes". Over the range of possible stimuli, this activity is often summarized by a neuron's "tuning curve" $a_i(\mathbf{x})$. Therefore, the function $a_i(\mathbf{x})$ defines the *encoding* of the stimulus into neural activity over the range of $\mathbf{x}$ of interest.

Given this encoding, the original stimulus vector can be estimated by decoding those activities, i.e.

$$\hat{\mathbf{x}} = \sum_i a_i(\mathbf{x})\mathbf{d}_i. \tag{3}$$

These decoding vectors, $\mathbf{d}_i$, can be found by a least-squares optimization (Salinas and Abbott 1994; Eliasmith and Anderson 2003). This optimization amounts to performing an SVD decomposition (i.e. pseudo-inverse) on the correlation matrix of the neural activities. The decoders resulting from this optimization thus define a "population code" over a set of neurons i for the representation of $\mathbf{x}$ by the combination of nonlinear encoding in Eq. 1 and linear decoding in Eq. 3.

As characterized to this point, the tuning curve is a time-averaged summary of neural activity. To explicitly incorporate a temporal code into this population code, we draw on work that has shown that most of the information in neural spike trains can be extracted by linear decoding (Rieke et al. 1997). Let us first consider the temporal code in isolation by taking the neural activities $a_i(t)$ to be decoded spike trains, i.e.

$$a_i(t) = \sum_n h_i(t) * \delta_i(t - t_n) = \sum_n h_i(t - t_n), \tag{4}$$

where $\delta_i(\cdot)$ are the spikes at times t_n for neuron i generated by G_i, and $h_i(t)$ are the linear decoding filters which, for reasons of biological plausibility, we can take to be the (normalized) post-synaptic currents (PSCs) in the subsequent neuron. Elsewhere, it has been shown that the information loss under this assumption compared to optimal linear filters is minimal, and can be alleviated by increasing population size (Eliasmith and Anderson 2003). As before, the encoding on which this linear decoding operates is defined as in Eq. 1, where G_i is defined by a spiking neuron model.

We can combine this temporal code with the previously defined population code to give a general population-temporal code for vectors:

$$\text{Encoding } \delta(t - t_{in}) = G_i \left[\alpha_i \langle \mathbf{x} \cdot \mathbf{e}_i \rangle_m + J_i^{\text{bias}} \right] \tag{5}$$

$$\text{Decoding } \hat{\mathbf{x}} = \sum_{i,n} h_i(t - t_n)\mathbf{d}_i \tag{6}$$

Since we are here concerned with the representation and transformation of functions, it is useful to consider the relationship between vector and function representation. Often, external world relations between two continuous variables (e.g. light intensity and space, pitch and time, etc.) that are reflected in neural activity are naturally characterized as function representation. That is, neural activity that changes as some $x(\nu)$ changes can be said to represent the relation captured by x (e.g., change in brightness as a function of location ν).

As with any representation, we must specify the domain of that representation. In the case of vectors, this is the subspace of the vector space that is represented by the neurons of interest (e.g., the $\mathbf{x}$ vectors we performed the minimization over previously). To help us identify the relevant function domain, we parameterize the set of represented functions by a vector of coefficients $\mathbf{r}$ with components r_m. These define any function of interest on a basis $\Phi(\nu)$

$$x(\nu; \mathbf{r}) = \sum_m r_m \Phi_m(\nu) \text{ for } \mathbf{r} \sim \rho(\mathbf{r})$$

Defining a particular probability distribution $\rho(\mathbf{r})$ is a way of limiting the space spanned by the basis $\Phi(\nu)$ to some subspace of particular interest. The specific choice of domain will depend on the application. Importantly, this is also the domain over which the optimization to find the decoders will be performed as described for Eq. 3. We can now proceed to define a population encoding and decoding analogous to that in Eqs. 1 and 3, but for functions:

$$\begin{aligned}\text{Encoding: } a_i(x(\nu; \mathbf{r})) &= a_i(\mathbf{r}) \\ &= G_i \left[\alpha_i \langle x(\nu; \mathbf{r}) e_i(\nu) \rangle_\nu + J_i^{\text{bias}} \right]\end{aligned} \tag{7}$$

$$\text{Decoding: } \hat{x}(\nu; \mathbf{r}) = \sum_i a_i(\mathbf{r}) d_i(\nu) \tag{8}$$

where $e_i(\nu)$ and $d_i(\nu)$ are the encoding and decoding functions of the neurons.

We can proceed further, projecting these encoding and decoding functions onto the same basis used to identify the function space. Notably, the basis could be either bi-orthonormal (for an overcomplete basis) or orthonormal. For this example, we assume an orthonormal basis $\Phi_m(\nu)$ for simplicity, although an analogous derivation follows for a bi-orthonormal set as employed in Sect. 4.1. To begin, we can rewrite the encoding and decoding functions as:

$$d_i(\nu) = \sum_m^M d_{im} \Phi_m(\nu). \tag{9}$$

and

$$e_i(\nu) = \sum_m^M e_{im} \Phi_m(\nu). \tag{10}$$

where the e_{im} and d_{im} identify the M coefficients that represent the encoding and decoding functions in the orthonormal $\Phi(\nu)$ basis. We can now exploit these relations by substitution into Eq. 7:

$$\begin{aligned} a_i(\mathbf{A}) &= G_i \left[\alpha_i \left\langle \sum_{n,m} r_m \Phi_m(\nu) e_{in} \Phi_n(\nu) \right\rangle_\nu + J_i^{\text{bias}} \right] \\ &= G_i \left[\alpha_i \left(\sum_{n,m} r_m e_{in} \delta_{nm} \right) + J_i^{\text{bias}} \right] \\ &= G_i \left[\alpha_i \left(\sum_m r_m e_{im} \right) + J_i^{\text{bias}} \right] \\ &= G_i \left[\alpha_i \langle \mathbf{r} \mathbf{e}_i \rangle_m + J_i^{\text{bias}} \right]. \end{aligned}$$

We have thus expressed the problem of *function encoding* as one of *vector encoding*, given an orthonormal basis. Similarly, we can express the problem of *function decoding* as one of *vector decoding:*

$$\hat{\mathbf{r}} = \sum_i a_i(\mathbf{r}) \mathbf{d}_i. \tag{11}$$

In short, this is the observation that it is mathematically equivalent to talk in terms of function spaces or vector spaces (given an orthonormal basis). However, there are good reasons to discuss both in the context of neural systems. This is because some neural systems are more naturally thought of as representing functions. Describing the visual field as a two-dimensional function is far more intuitive than describing it as a 25-dimensional vector space, even though both might carry the same information. In other words, it may be the case than any 2-dimensional function of relevance to visual processing can be represented as a 25-dimensional vector over an orthonormal basis. But, visualization of the stimuli, the relevant tasks, and even the neural responses themselves is much easier to interpret as related to a 2-dimensional function space. However, it can be mathematically convenient to consider such function representations as vector representations to derive certain results and to characterize transformations of the representation. We exploit this relationship in our solution to the normalization problem.

2.2 Transformation

For such representations to be used by the system it must be possible to define *transformations* (i.e. functions of the vector

variables). Fortunately, we can again find (least-squares optimal) decoders $\mathbf{d}_i^{f(\mathbf{x})}$ to perform a transformation $f(\mathbf{x})$. Therefore, instead of finding the optimal decoders $\mathbf{d}_i$ to extract the originally encoded variable $\mathbf{x}$ from the encoding, we can "re-weight" the decoding to give some function $f(\mathbf{x})$ other than identity. Given this characterization, it is a simple matter to re-write the encoding and decoding equations for estimating some function of the vector variable:

$$\begin{aligned} \delta(t-t_{in}) &= G_i\left[\alpha_i \langle \mathbf{x}\cdot\mathbf{e}_i\rangle_m + J_i^{\text{bias}}\right] && \text{Encoding} \\ \hat{f}(\mathbf{x}) &= \sum_{i,n} h_i(t-t_n)\mathbf{d}_i^{f(\mathbf{x})} && \text{Decoding} \end{aligned}$$

Notably, both the linear and nonlinear functions of the encoded variable can be computed in this manner (Eliasmith and Anderson 2003).

Of particular interest here, is how we can perform statistical inference. Statistical inference is naturally thought of as a transformation of function representations. Suppose the system traffics in probability density function representations of the form $\rho(\mathbf{v}|\mathbf{r})$, where $\mathbf{v}$ is the property that needs to be inferred on the basis of data $\mathbf{r}$. Notice that here, the function $\rho()$ is analogous to the vector $\mathbf{x}$ in our previous discussions, since these are the objects taken to be represented by the neurons.

Applying Eqs. 7 and 8, we can define ensembles of neurons as representing such functions as follows:

$$\begin{aligned} a_i(\mathbf{r}) &= G_i\left[\alpha_i \langle e_i(\mathbf{v})\rho(\mathbf{v}|\mathbf{r})\rangle_{\mathbf{v}} + J_i^{\text{bias}}\right] \\ b_j(\mathbf{r}) &= G_j\left[\alpha_j \left\langle e_j(\mathbf{u})\rho(\mathbf{u}|\mathbf{r})\right\rangle_{\mathbf{u}} + J_j^{\text{bias}}\right], \end{aligned} \tag{12}$$

with the corresponding decoding rules

$$\begin{aligned} \hat{\rho}(\mathbf{v}|\mathbf{r}) &= \sum_i d_i(\mathbf{v})a_i(\mathbf{r}) \\ \hat{\rho}(\mathbf{u}|\mathbf{r}) &= \sum_j d_j(\mathbf{u})b_j(\mathbf{r}). \end{aligned} \tag{13}$$

One transformation of interest for relating such representations is simple conditional inference. Assuming $\mathbf{r}$ is conditionally independent of $\mathbf{u}$ given $\mathbf{v}$:

$$\rho(\mathbf{u}|\mathbf{r}) = \int \rho(\mathbf{u}|\mathbf{v})\rho(\mathbf{v}|\mathbf{r})\mathrm{d}\mathbf{v} \tag{14}$$

Substituting Eq. 14 into Eq. 12, and then letting $\hat{\rho}(\mathbf{v}|\mathbf{r}) = \rho(\mathbf{v}|\mathbf{r})$ gives

$$\begin{aligned} b_j(\mathbf{r}) &= G_j\left[\alpha_j \left\langle e_j(\mathbf{u})\rho(\mathbf{u}|\mathbf{v})\rho(\mathbf{v}|\mathbf{r})\right\rangle_{\mathbf{v},\mathbf{u}} + J_j^{\text{bias}}\right] \\ &= G_j\left[\alpha_j \left\langle e_j(\mathbf{u})\rho(\mathbf{u}|\mathbf{v})\sum_i d_i(\mathbf{v})a_i(\mathbf{r})\right\rangle_{\mathbf{v},\mathbf{u}} + J_j^{\text{bias}}\right] \\ &= G_j\left[\sum_i \omega_{ji}a_i(\mathbf{r}) + J_j^{\text{bias}}\right], \end{aligned} \tag{15}$$

where

$$\omega_{ji} = \alpha_j \left\langle e_j(\mathbf{u})\rho(\mathbf{u}|\mathbf{v})d_i(\mathbf{v})\right\rangle_{\mathbf{v},\mathbf{u}}. \tag{16}$$

Note that the expression within the angle brackets is integrated over both $\mathbf{v}$ and $\mathbf{u}$, but for different reasons. The integral over $\mathbf{v}$ is due to the inference in Eq. 14, while the integral over $\mathbf{u}$ is from the neural encoding, Eq. 12. Within this framework, Eqs. 15 and 16 tell us how to implement simple feedforward statistical inference in a neurobiologically plausible network. In particular, the connection weights between neurons in these networks can be understood as the projection of the encoding functions of the output neurons, $e_j(\mathbf{u})$, on the condition, $\rho(\mathbf{u}|\mathbf{v})$, weighted by the decoding function, $d_i(\mathbf{v})$, of each input neuron.

Notably, for the final representation to be probability density, it should have an integral equal to 1. However, even if all of the distributions are appropriately defined there is nothing in this formulation that will guarantee us of that fact. This is because our representations are approximate, and hence likely to introduce error. In addition, they are defined over finite subspaces, which necessarily leads to violations of standard probabilistic representation assumed by these equations. If we repeat such a transformation many times in a row, such errors will accumulate, and it would be highly unlikely that the integral of our function representation will remain 1. This is also a consideration for more sophisticated forms of inference, such as Bayesian inference, where the computation is more difficult, given the explicit normalization by the marginal. In the remainder of the article, we concern ourselves with solving the normalization problem for simple statistical inference.

2.3 Dynamics

As we have discussed in detail in other work, the dynamics of neural systems can be described using the previous characterizations of representation and transformation by employing modern linear control theory (Eliasmith and Anderson, 2003). Specifically, we can allow the "higher-level" vector variables represented by a neural population to be control theoretic state variables, specifically, the vector space $\mathbf{r}$ decoded in equation 11 can be taken to be the relevant state variable. Since we have shown how function representations can also be considered as vector representations, we consider the vector case here.

Let us first consider linear, time-invariant (LTI) systems. Recall that the state equation in modern linear control theory describing LTI dynamics is

$$\dot{\mathbf{r}}(t) = \mathbf{A}\mathbf{r}(t) + \mathbf{B}\mathbf{u}(t). \tag{17}$$

Notably, the input matrix $\mathbf{B}$ and the dynamics matrix $\mathbf{A}$ completely describe the dynamics of the LTI system, given the state variables $\mathbf{r}(t)$ and the input $\mathbf{u}(t)$. Taking the Laplace

transform of (17) gives:

$$\mathbf{r}(s) = h(s)\left[\mathbf{A}\mathbf{r}(s) + \mathbf{B}\mathbf{u}(s)\right],$$

where $h(s) = \frac{1}{s}$. Any LTI control system can be written in this form.

In the case of the neural system, the transfer function $h(s)$ is not $\frac{1}{s}$, but is determined by the intrinsic properties of the component cells. Since it is reasonable to assume that the dynamics of the synaptic PSC dominate the dynamics of the cellular response as a whole (Eliasmith and Anderson, 2003), it is reasonable to characterize the dynamics of neural populations based on their synaptic dynamics, i.e. using $h_i(t)$ from Eq. 4.

A simple model of a synaptic PSC is given by

$$h'(t) = \frac{1}{\tau}e^{-t/\tau}, \tag{18}$$

where τ is the synaptic time constant. The Laplace transform of this filter is:

$$h'(s) = \frac{1}{1+s\tau}.$$

Given the change in filters from $h(s)$ to $h'(s)$, we now need to determine how to change $\mathbf{A}$ and $\mathbf{B}$ in order to preserve the dynamics defined in the original system (i.e. the one using $h(s)$). In other words, letting the neural dynamics be defined by $\mathbf{A}'$ and $\mathbf{B}'$, we need to determine the relation between matrices $\mathbf{A}$ and $\mathbf{A}'$ and matrices $\mathbf{B}$ and $\mathbf{B}'$ given the differences between $h(s)$ and $h'(s)$. To do so, we can solve for $\mathbf{r}(s)$ in both cases and equate the resulting expressions for $\mathbf{r}(s)$. Doing so gives

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I} \tag{19}$$

$$\mathbf{B}' = \tau\mathbf{B}. \tag{20}$$

This procedure assumes nothing about $\mathbf{A}$ or $\mathbf{B}$, so we can construct a neurobiologically realistic implementations of any dynamical system defined using the techniques of modern linear control theory applied to LTI systems (constrained by the neurons' intrinsic dynamics). For example, if we want to implement an integrator, we would set $\mathbf{A}$ =0 and $\mathbf{B}$ = 1 in Eq. 17. As a result, we find $\mathbf{A}' = \mathbf{I}$ and $\mathbf{B}$ =τ for the neural dynamics from Eqs. 19 and 20. Importantly, the same approach can be used to characterize the broader class of time-varying and nonlinear control systems, though for repeated statistical inference we can consider only the linear case.

To be more specific, we can reconsider the statistical inference defined by Eq. 15. The connection weights defined for this inference determine the feedforward connection between a first and second population of cells, a and b. If we want to perform repeated inference, then we need to use the result of a single inference, found in population b, to drive the next inference. A natural way to do this is to project the b neurons back to a. This results in a recurrent network, where the dynamics of the whole network are similar to an integrator, since the output of the network is also its recurrent input (see Fig. 5 for the architecture, and Eqs. 24 and 25 for the connection weights).

3 An algebraic solution

Since we have characterized neural representation of functions as equivalent to representations in a vector space, here we consider a solution to the normalization problem strictly in a mathematical context first. We subsequently demonstrate it in the context of simulations of spiking neural networks.

Following the introduction of some mathematical background, we derive an expression for a bias function that can be used to modify any linear transformation (such as conditional inference), to guarantee that this transformation preserves the integral of the function represented by the vector being transformed. That is, the bias function tells us how to perform inference while normalizing the result.

3.1 Mathematical background

Let $(V, \langle,\rangle)$ and $(W, \langle,\rangle)$ be Hilbert spaces over $\Re$. Finite dimensional inner product spaces, like those used to characterize representation earlier, over $\Re$ are necessarily Hilbert spaces.

Definition 1 We say that a linear transformation $T : V \to W$ respects (a, b), where $a \in V, b \in W$, if we have $\langle v, a\rangle = \langle T(v), b\rangle\ \forall v \in V$.

That is, if inner product between the transformation of v and b is the same as that between v and a, then T respects (a, b). This is a standard definition.

Definition 2 T^* is the unique linear transformation that satisfies $\langle T(v), w\rangle = \langle v, T^*(w)\rangle\ \forall v \in V, \forall w \in W$

Often T^* is called the adjoint of T.

Proposition 3 *Let $x, y \in V$, then $\langle v, x\rangle = \langle v, y\rangle\ \forall v \in V$ iff $x = y$.*

Proof The reverse direction is trivial. For the forward direction we have $\langle v, x\rangle = \langle v, y\rangle\ \forall v{\in}V \Rightarrow \langle v, x - y\rangle = 0\,\forall v{\in}V$. In particular, $\langle x - y, x - y\rangle = 0$ so $x = y$. □

Proposition 4 *Let $T : V \to W$ be a linear transformation and $a \in V, b \in W$, then T respects (a, b) iff $T^*(b) = a$.*

Proof T respects (a, b) iff $\langle v, a\rangle = \langle T(v), b\rangle = \langle v, T^*(b)\rangle$ $\forall v \in V$ from Definitions 1 and 2. But this happens iff $T^*(b) = a$ by Proposition 3.

Suppose $T : V \to W$ is linear and continuous and $S : V \to \Re$, $R : W \to \Re$ are bounded linear functionals. By the Riesz Representation Theorem there is $a \in V, b \in W$ such that $\langle f, a\rangle = S(f) \forall f \in V$ and $\langle f, b\rangle = R(f) \forall f \in W$. We then have that $S(f) = R(T(f)) \forall f \in V$ iff $\langle f, a\rangle = \langle T(f), b\rangle \forall f \in V$ (i.e. T respects (a, b) iff $T^*(b) = a$). □

3.2 Application to probability function representation in the NEF

As described earlier, we need to ensure that the integral of probability densities represented in neural populations is 1. In the Neural Engineering Framework (NEF), probability densities are represented as linear combinations of a finite set of functions (see Eq. 8). Applying a linear transformation to such a linear representation will not necessarily preserve this property, even when the transformation corresponds to probabilistic inference.

Let $(F, \langle , \rangle)$ be the Hilbert space of continuous functions on some compact interval $[x_1, x_2]$.

Definition 5 1_F is the function that is identically 1 over $[x_1, x_2]$.

Let A *and* B be closed subspaces of F (closed subspaces of Hilbert spaces are themselves Hilbert spaces). Let $T : A \to B$ be linear (and continuous). Let $O_A : A \to F$ and $O_B : B \to F$ be projection maps (i.e. $O_A(f) = f$). These are projections like those defined in Eqs. 9 and 10, in our discussion of neural representation of functions.

Definition 6 $I : F \to R$ is normal integration. Normal integration is a bounded linear functional.

Proposition 7 *For a closed subspace C of F, the Riesz representation c of integration in C is given by* $O_C^*(1_F)$ *where* $O_C : C \to F$ *is the identity map.*

Proof $I(f) = \langle O_C(f), 1_F\rangle = \langle f, O_C^*(1_F)\rangle \forall f \in C$. □

For T to preserve integrals (which is a necessary and sufficient condition for T to map integral 1 functions to integral 1 functions), we require that $I(f) = I(T(f)) \forall f \in A$ which by the discussion in the last section is equivalent to requiring $T^*(b) = a$ where a and b are the Riesz representations for integration in A and B, respectively.

Since $a = O_A^*(1_F)$ and $b = O_B^*(1_F)$, we have that T is integral preserving iff $T^*(b) = a$ iff $T^* O_B^*(1_F) = O_A^*(1_F)$.

If $P : F \to F$ corresponds to probabilistic inference then P necessarily preserves integrals, which is equivalent to the statement that P respects $(1_F, 1_F)$. This is in turn equivalent to $P^*(1_F) = 1_F$.

However, if A and B are strict subspaces of F (finite dimensional in the case of the NEF) and T is given by $O_B^+ P O_A$ where $^+$ denotes the pseudo-inverse (as in the NEF optimization), then T will not necessarily preserve integrals.

Definition 8 We define for a closed subspace C *of* F, the "bias vector of C" which we will denote bias(C) to be $O_C^{+*} O_C^*(1_F) = (O_C O_C^+)^*(1_F)$.

Next, we give a proposition which essentially states that bias(C) defines how encoding functions and vectors from F into the subspace C will change their integrals.

Proposition 9 $I(O_C^+(f)) = \langle f, bias(C)\rangle \forall f \in V$

Proof $I(O_C^+(f)) = \langle O_C^+(f), O_C^*(1_F)\rangle = \langle f, O_C^{+*} O_C^*(1_F)\rangle = \langle f, \text{bias}(C)\rangle \forall f \in V$. □

Since bias(C) is not in general equal to 1_F there will be functions in F whose integral will not be preserved when encoded into the C space.

Applying this to the example of NEF representation, we have:

$$I(T(f)) = I(O_B^+ P O_A) = \langle P O_A(f), \text{bias}(B)\rangle = \langle f, O^* P^* \text{bias}(B)\rangle \forall f \in A$$

This is equal to $I(f) = \langle f, O_A^*(1_F)\rangle \forall f \in A$ iff $O_A^* P^* \text{bias}(B) = O_A^*(1_F)$ which is true iff $O_A^*(P^* \text{bias}(B) - 1_F) = 0_F$ which holds iff $P^* \text{bias}(B) - 1_F \in null(A)$.

Definition 10 Define the "discrepancy" as $\text{disc}(A, B, P) = O_A^*(P^* \text{bias}(B) - 1_F)$ for spaces A and B and transformations P.

If $\text{bias}(B) = 1_F$ then since $P^*(1_F) = 1_F$ we have $\text{disc}(A, B, P) = 0_F$ so P is integral preserving. This is a sufficient condition but it is not strictly required condition for T to be integral preserving.

Proposition 11 $bias(C) = 1_F$ *iff* $1_F \in C$.

Proof Since bias(C) is a vector in C by definition the forward direction is trivial. Suppose that $1_F \in C$. Noting that $O_C : C \to F$ is one-to-one we have $O_C^+ O_C = I_C$ and in particular $O_C^+(1_F) = 1_F$. Taking O_C of both sides then gives $O_C O_C^+(1_F) = O_C(1_F) = 1_F$. The result then follows from the fact that $O_C O_C^+ = (O_C O_C^+)^* = \text{bias}(C)$. □

This is the only proposition in which we used specific properties of the pseudo-inverse, which we use to determine the decoding functions. Indeed, the bias can be defined somewhat independently of the decoding transformation used, and all the previous results discussed in this paper will hold, save perhaps this one.

3.3 Bilinear functions

Since probabilistic inference often relies on the integral of products of functions, it is important to consider the more

general case of bilinear functions. Suppose C is a closed subspace of F and $P : F \times F \to F$ corresponds to probabilistic inference. In an analogous way to the linear case, we let $T : A \times B \to C$ be given by

$$T(f, g) = O_C^+ OP(O_A(f), O_B(g)).$$

We will say that T is integral preserving if $I(T(f, g)) = I(f)I(g)$ which is the natural definition to make for T to be both bilinear and to preserve integrals with respect to only the first or the second arguments. Since P corresponds to probabilistic inference, it preserve integrals in this sense.

For a fixed $f \in A$ we define $T_f : B \to C$ by $T_f(g) = T(f, g)$. Then, T_f is a continuous linear transformation and applying results from the previous section we have:

$$I(T_f(g)) = I(O_C^+ P_f) = \left\langle P_f O_B(g), \text{bias}(C) \right\rangle$$
$$= \left\langle g, O_B^* P_f^* \text{bias}(C) \right\rangle$$

Noting that $I(f)I(g) = \langle g, I(f)1_F \rangle$, we have that T_f preserves integrals iff $O_B^*(P_f^* \text{bias}(C) - I(f)1_F) = 0_F$.

If we have $I(f) \neq 0$, then an equivalent condition is:

$$\text{disc}(B, C, \frac{P_f}{I(f)}) = 0_F \tag{21}$$

And, since $P_f^*(1_F) = I(f)1_F$ we again have that a sufficient condition for T_f to preserve integrals is $\text{bias}(C) = 1_F$.

The intuitive interpretation of the discrepancy function is that it identifies areas in the function space that will be distorted by a linear transformation between closed subspaces of F (as given by the NEF formulation). Similarly, the bias function describes how optimal linear decoding into a given subspace will distort arbitrary functions from F. This distortion, among other things, affects the integral of the functions. Areas of the bias function that are below 1 correspond to areas that need to be boosted by the bias function. In short, computing the bias(C), and including it in the transformations defined earlier provides a method for guaranteeing that the result of a probabilistic transformation will be a function whose integral is 1.

4 Application to the NEF

Given this derivation of the bias function, we now present several numerical demonstrations that this function is useful in practice.

4.1 Idealized application

To begin, we consider the repeated application of the statistical inference described by Eq. 14, where the conditional probability is a Gaussian for all values of v. This distribution is shown in Fig. 1. To demonstrate the utility of a bias function in this context, we begin by employing the idealized neural responses shown in Fig. 2a.

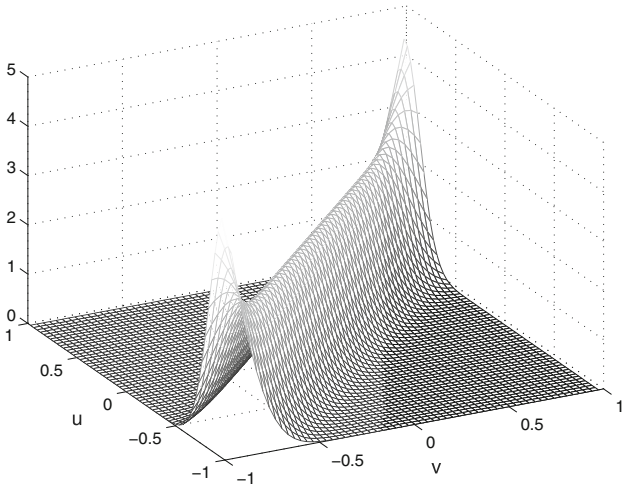

Fig. 1 The conditional probability distribution for the example inference problem. The distribution has been normalized so that $\int \rho(u|v) = 1 \forall v$, giving the higher values on either end

These responses form the encoding basis for the representation, i.e.

$$a_i(\rho(v)) = G_i \left[\alpha_i \langle e_i(v)\rho(v) \rangle_v + J_i^{\text{bias}} \right]$$

where all $\alpha_i = 1$, G_i is linear, and $J_i^{\text{bias}} = 0$. Consequently, $a_i(\rho(v)) = \langle e_i(v)\rho(v) \rangle_v$, which is shown for all 20 "neurons" in Fig. 2a where $\rho(v) = \delta(v)$ is assumed as input to give these neuron-like tuning curves.

The complementary optimal decoding functions are shown in Fig. 2b. As described earlier, these functions are found by minimizing the reconstruction error $\left[\rho(v) - \hat{\rho}(v)\right]^2$ using the pseudo-inverse, where:

$$\hat{\rho}(v) = \sum_i a_i(\rho(v)) d_i(v).$$

We have now defined a simplified problem so application of the algebraic solution to preserving integrals is straightforward. Specifically, recall from Definition 8, that the bias vector we need to compute can be found as follows

$$\text{bias}(C) = O_C^{+*} O_C^*(1_F) = (O_C O_C^+)^*(1_F).$$

Therefore, for the NEF the bias function is

$$\text{bias}(v) = \mathbf{DE}1_F \tag{22}$$

where $\mathbf{D}$ and $\mathbf{E}$ are matrices consisting of the decoding functions in columns and encoding functions in rows, respectively. The product of these matrices can be thought of as providing a kernel function of v and v', $K(v, v')$. Therefore, given the definition of 1_F (i.e., equal to 1 over the range), the bias can be written:

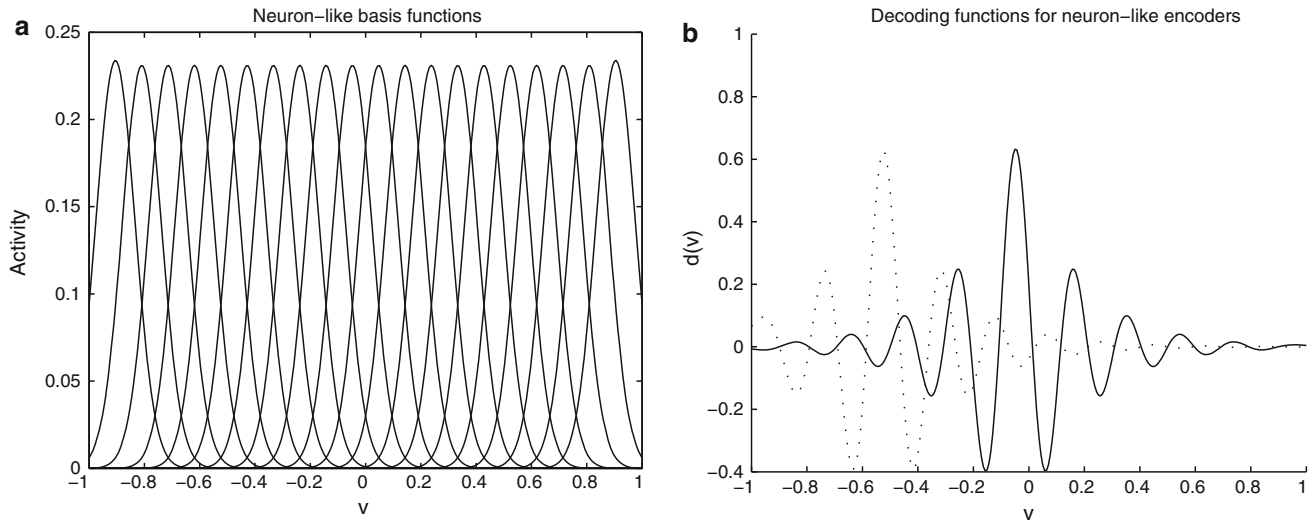

Fig. 2 Idealized neuron bi-orthonormal basis. **a** The 20 evenly spaced Gaussian neuron-like responses used in the example in this section. These form the encoding basis of the neural representation of $\rho(v)$. **b** Two example decoding functions. These do not have a neural correlate

$$\text{bias}(v) = \int K(v, v')dv'.$$

Essentially, this bias captures the distortion to the representation that results from projecting it into the neuron-like encoding. Consequently, because we want the discrepancy to be equal to zero, as shown in Eq. 21, the transformation of interest must be modified as follows:

$$\rho(u|v)_{\text{bias}} = \rho(u|v)/\text{bias}(v) \forall v. \tag{23}$$

To be clear, this division by the bias function must be done for each row in the conditional distribution separately and pointwise. This new transformation $\rho(u|v)_{\text{bias}}$ will now preserve integrals given the properties of the representational space, as captured by bias(v).

To demonstrate the utility of this bias, it is important to repeatedly apply the transformation, as the distortion can be reasonably small for a single inference. In this idealized application, the starting $\rho(v)$ is a Gaussian centered at zero, so repeated application of the given conditional results in a flattening of that Gaussian to an even distribution over the range of interest as evident in Fig. 3. This figure also demonstrates the important effect of the bias function over repeated applications. As can be seen, with the bias function, the representation, though distorted, remains a good estimate of ideal inference. In contrast, without the bias function (i.e. applying $\rho(u|v)$ to the neuron representation), the estimate quickly decays.

An explicit calculation of the integral of these representations over the 100 iterations of the simulation is shown in Fig. 4. As can be seen from this graph, the integral is preserved well in both the ideal and biased cases, but not in the unbiased case. In short, the inference is being appropriately normalized in the neuron-like representation by the

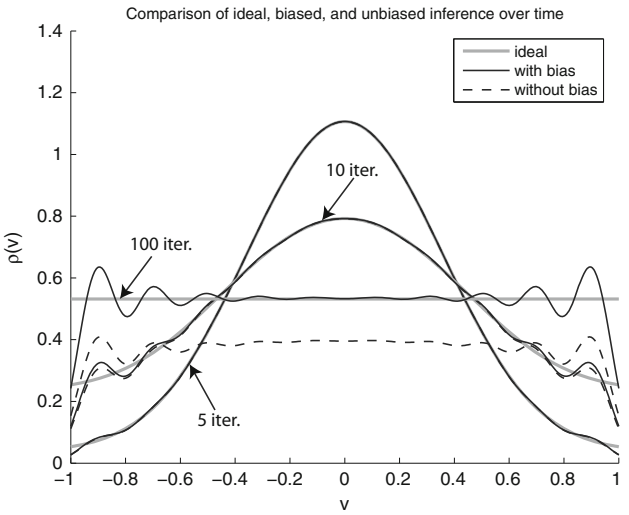

Fig. 3 Repeated application of inference. Each line represents a distribution after a given number of inferences (5, 10, and 100 iterations). The ideal, which goes from a Gaussian to an even distribution is shown in gray. Without the bias function, the integral is not preserved as can be seen from the decaying dashed line. With the bias function, the integral is preserved and the represented distribution approximates the ideal well, as shown by the solid black line

bias. The slight increase in the integral of both the ideal and biased cases is due to the end effects evident in the conditional matrix in Fig. 1. This change in the integral can be removed from the inference, but only at the cost of no longer inferring to a constant distribution. In either situation, the biased inference normalizes the representation as well as the ideal case.

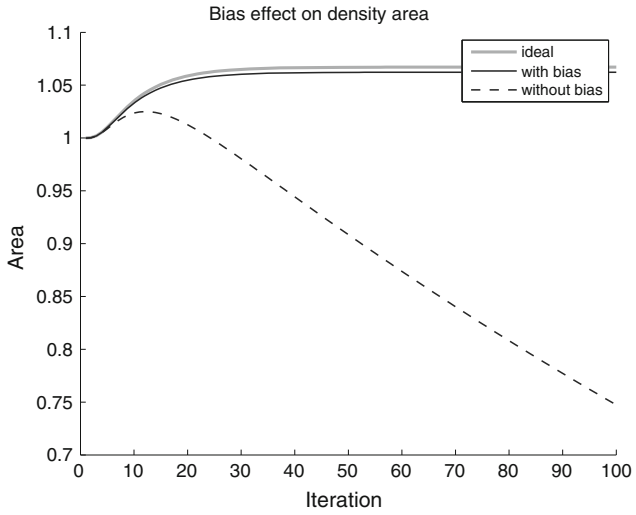

Fig. 4 The integral of the representations during repeated inference. The ideal and biased inference preserve integrals similarly over all 100 iterations. In contrast, the unbiased inference decays quickly

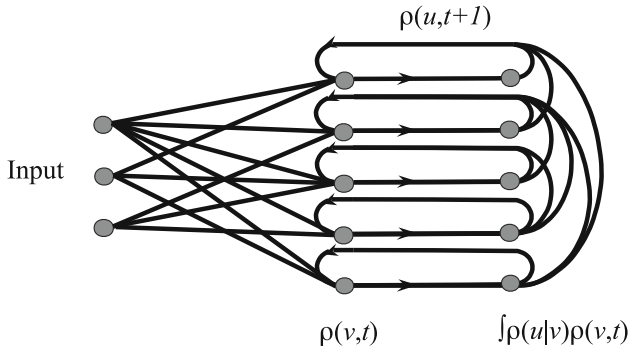

Fig. 5 The network architecture of the spiking network implementing repeated inference. The input signal is projected into the first layer, which represents the presented function $\rho(v, t)$. Inference is then performed in the connection weights between the first and second layers, resulting in a representation of $\rho(u, t)$, which is then projected back to the first layer for subsequent inference

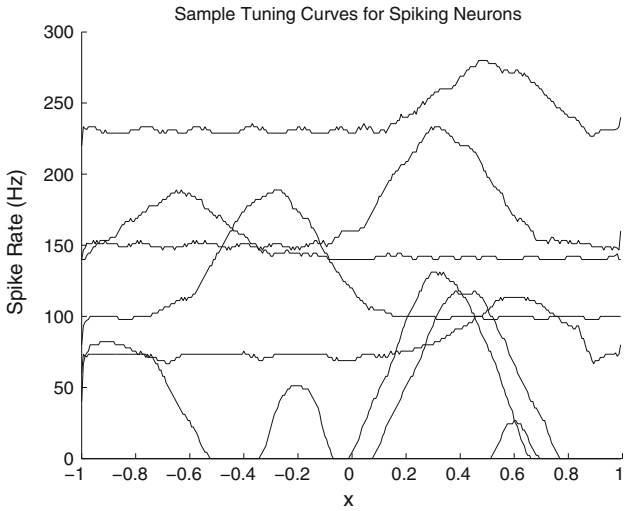

Fig. 6 Sample tuning curves of neurons in the recurrent network. The tuning curves were generated by moving a delta function input across the possible range of v. The resulting spikes were binned (50 ms) and smoothed (10 point moving average). The neurons shown were randomly selected from the output population. A variety of background firing, gain, and input selectivity is evident

4.2 A biologically relevant application

Considering the ideal neural representation helps to distinguish the effects of the bias function from the effects of the neural representation itself. Here, we use the same solution, but in the context of a recurrent neural network implemented in spiking neurons. This demonstrates that the solution may be biologically relevant.

In the previous ideal case, repeated inference was performed by simply taking the representation after one pass through the neuron-like nodes and using it as the starting point for the next iteration. However, in a more biologically plausible setting, repeated inference must be done by either embedding the inference in a many layered feedforward network, or through a recurrent connection. To demonstrate the generality of the solution, we have chosen the latter approach. This brings with it the extra challenge of ensuring that the natural dynamics of a spiking network are accounted for in the model. We can employ the results of Sect. 2.3 to convert both our inference transformation and the recurrent connection to ensure they are compatible with the post-synaptic current dynamics in the model. The network structure is shown in Fig. 5.

In these simulations, each population consists of 3,000 leaky integrate-and-fire (LIF) neurons, which define the G_i in Eq. 7. Sample tuning curves for neurons in this network are shown in Fig. 6. As can be seen from this figure, the tuning curves are very heterogeneous, unlike the ideal case. The neurons used in the network have their thresholds, gains, and encoding vectors chosen randomly from even distributions over appropriate ranges.

The connection weights from the first to second layer are determined as in Eq. 16, with the condition being modified as in Eq. 23 to include the bias. The resulting weights for this connection are thus:

$$\omega_{ji} = \alpha_j \left\langle e_j(u)\rho(u|v)_{\text{bias}} d_i(v) \right\rangle_{v,u} \tag{24}$$

where the encoders and decoders are from the post-synaptic and pre-synaptic populations, respectively. The recurrent weights from the second layer to the first are also computed as in Eq. 16, where the transformation is simply the identity map, so:

$$\omega_{ji} = \alpha_j \left\langle e_j(v) d_i(u) \right\rangle_{v,u}. \tag{25}$$

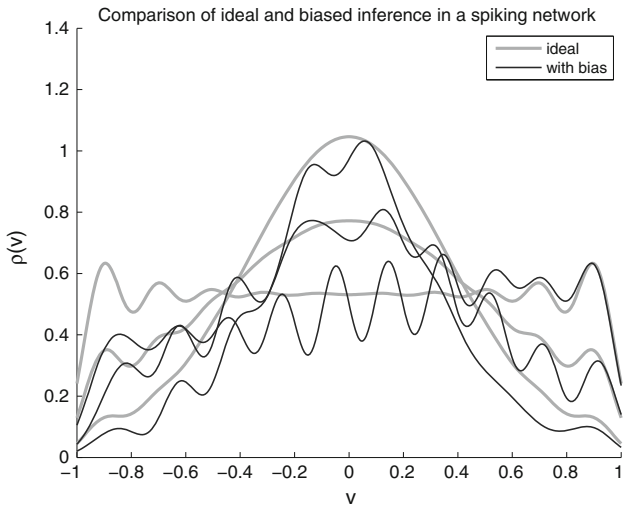

Fig. 7 Performance of the spiking inference network over time with the bias included in the connection weights. Some distortion due to the neural representation is evident, but the overall integral is preserved as demonstrated in Fig. 9. The spiking representation is computed by filtering the spikes over a 50 ms window to smooth the results. The example functions are plotted at points in time equivalent to the 5, 10, and 100 iterations in Fig. 3

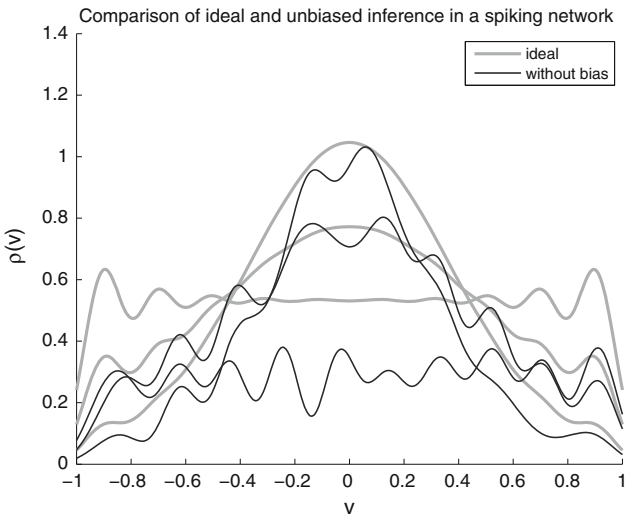

Fig. 8 Performance of the spiking inference network without bias in the connection weights. The final representation has a much smaller area than earlier representations as demonstrated by Fig. 9

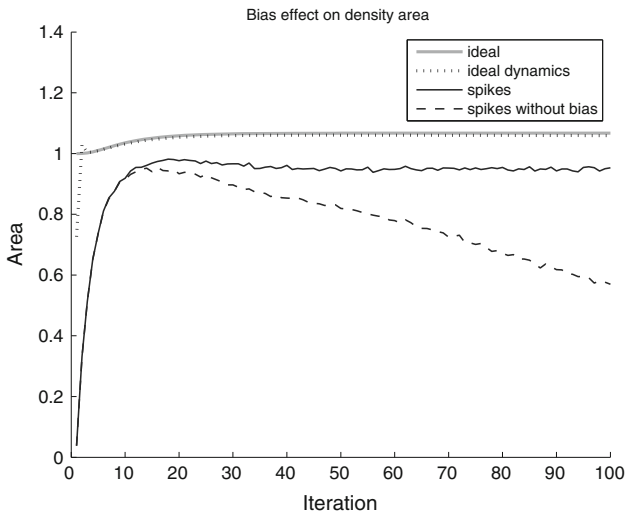

Fig. 9 A comparison of the integral of the various simulations. Included is the ideal case with bias from Fig. 4, the ideal case in a recurrent dynamical setting, and the results of the recurrent spiking network with and without the bias included in the connection weights. It is evident here that the bias results in preservation of the integral in the spiking network. The integral is slightly lower than the ideal due to error introduced through the neural representation

In essence, this connection projects the current representation of $\rho(u)$ to the future state of $\rho(v)$. This is very similar to performing repeated inference in a many layered feedforward network. The main difference is that the neurons used to represent $\rho(v, t)$ and $\rho(v, t+\delta t)$ are the same in the recurrent case, and would not be in the feedforward case.

We have chosen our time constants and scaled the weight matrices so that the 100 iterations in the ideal, feedforward simulation is equivalent to the 2 s run times used for the recurrent network. Specifically, post-synaptic currents are modeled as in Eq. 18 with a time constant of 10ms, and scaling is determined by Eqs. 19 and 20.

The behavior of this network with the bias included in the connection weights is shown in Fig. 7. The behavior of the same network without the bias in the connection weights is shown in Fig. 8. These figures can be compared to Fig. 3. It is evident from this comparison that the spiking network is performing similarly to the the ideal inference (shown on both figures in gray).

Finally, Fig. 9 demonstrates that the spiking network preserves the integral of the represented probability distribution appropriately. Comparing the two "ideal" cases demonstrates that the switch to a dynamic characterization of the problem does not change the ability of the network to preserve the integrals. More importantly, comparing these ideals to the spiking network responses demonstrates that the connection weights that include the bias term is able to preserve the integral, while the network without the bias is not able to do so. Given the similarity between the dynamical and non-dynamical characterizations of repeated inference, the difference between the spiking network performance and the ideal is likely due to the error in the neural representation evident in Fig. 7. Nevertheless, the integral itself is preserved. Thus, including the bias in the connection weights is a possible method for ensuring normalization during repeated inference in a spiking neural network.

5 Discussion

These results demonstrate that this method for ensuring the normalization of probability density functions is effective in the context of a spiking recurrent network. It is worth emphasizing several aspects of this solution to the normalization problem. First, the input signal is arbitrary. Though in the presented simulations it is a Gaussian distribution centered at zero, there are no assumptions about the kind of distribution being represented built into the solution. Therefore, while there are limitations on the kind of functions that can be represented by a neural population given the frequency content of their tuning curves, any function that can be represented by the neural basis can be normalized by this method.

Second, the conditional distribution that determines the inference is similarly arbitrary. We have chosen a particularly simple condition, so it would be obvious if repeated applications of the matrix were generating the correct results. As with the representation of the input function, the only constraints on the quality of the inference come from the quality of the neural representation of the space the inference is being performed on.

The quality of the neural representation used in the NEF has been discussed in detail elsewhere (Eliasmith and Anderson 2003, pp. 44–49). In particular, it has been shown that the root-mean-squared error of the representation decreases as $1/N$, where N is the number of neurons. In this same work, it has been demonstrated that there is a clear relationship between the quality of representation and the stability of a recurrent network employing such representations (Eliasmith and Anderson 2003, pp. 232–243). Specifically, the error in the approximation of the representation to the ideal vector space drive the dynamics of the network. When the approximation "crosses" the ideal with a negative gradient, there will be a dynamical fixed point. When the crossing has a positive gradient, there will be a dynamical unstable point. More generally, the speed of drift of the network is proportional to the magnitude of the error, with the direction of the drift determined by the error gradient. It should also be noted that the speed of drift is proportional to the synaptic time constant of the recurrent connections. Taken together, these observations demonstrate that the quality of repeated normalization is systematically sensitive to the number of neurons used to represent the underlying vector space. The preceding results were achieved with 3,000 neurons.

Third, the vector space underlying the neural representation is also arbitrary in the sense that the solution presented here does not depend on a particular choice of vector space. In the simulations presented, we have chosen a twenty dimensional space with a reasonably broad tuning to the basis functions. However, these choices are independent of the normalization solution, and can be tailored to the particular neural system that the solution is to be applied to.

It has often been noted that past approaches to performing statistical inference with neural representations demand some kind of normalization to ensure the neurons do not saturate and to keep recurrent networks stable (e.g., Rao 2004; Ma et al. 2006, p. 1436). Most commonly, these approaches allow for some kind of divisive normalization to be added to their networks to ensure normalization occurs. Divisive normalization has been suggested to take place in cortical circuits for several decades, and recent work has documented many examples of responses that are well-modeled by the presence of divisive normalization (Ringach 2010).

However, there remain several outstanding concerns with divisive normalization in the context of statistical inference. For instance, recent work suggests that such normalization cannot account for the distribution of observed neural responses (Shi et al. 2006). More importantly for statistical inference, the most common assumption is that a pooling of nearby neural responses divides (by an as-yet undetermined mechanism) the activity of the cell being normalized. However, the inclusion of this extra nonlinearity in cells is often not explicitly incorporated within inference models (Ma et al. 2006). Even if it is included, the time delays introduced by having to pool such information over the population of cells in a spiking network are ignored (Rao 2004). These delays can have serious impact on a recurrent network's stability, and in a feedforward implementation, would result in a non-normalized inference for at least some initial period of time.

The proposal presented here avoids such concerns. Since the normalization is included in the connection weight matrix, there is no need to introduce additional mechanisms to ensure that normalization takes place. Consequently, additional single cell nonlinearities, or the use of global inhibition for normalization, can be avoided. In addition, the preceding implementation in a recurrent spiking network demonstrates that the approach does not introduce unexpected stability issues.

Empirically distinguishing these two suggestions is likely to be challenging. Both can account for typical normalization effects (DeAngelis et al. 1992; Reynolds et al. 1999). Consequently, more subtle differences would need to be identified. For instance, the timing of the effect of normalization should be different for the two cases. Pooling models of normalization should predict a delay between the input signal and the normalization of that signal that accounts for the time it takes for the pooled responses to affect each individual cell's response. In contrast, the proposal we have presented would allow the normalization to occur as the signal propagates through the system. The difficulty with identifying this difference lies in the fact that there are many confounding effects, such as adaptation and feedback, that would be difficult to control for.

Another possible means of distinguishing these suggestions is to examine the differential effects of removing inhibition. If local inhibition alone could be blocked, then the current proposal would allow feedforward normalization to proceed as normal, though divisive normalization would be expected to fail. Similarly, even in a recurrent normalization situation, much of the recurrent information is excitatory under our proposal, so normalization should not be completely removed (even though the recurrent signal would not be the truly inferred distribution). Finally, sufficient random perturbation of the connection weights in the present proposal should essentially eliminate the effects of normalization. In contrast, a pooling model would continue to normalize the activity of the cells regardless of the specific computation being performed by the feedforward weights. While these hypotheses can be more carefully quantified by further modeling, testing them convincingly in vivo will be extremely challenging.

Despite these empirical challenges, there are clear theoretical differences between the present proposal and the more familiar divisive normalization. Future work may elucidate specific applications which would help distinguish these possibilities. Consequently, it remains useful to consider alternative biologically plausible methods for normalizing neural representations. This is especially relevant given the several recent suggestions for implementing statistical inference in neurally plausible networks.

6 Supplementary material

All Matlab® code and Nengo models for these simulations are available in the supplementary material.

References

Conklin J, Eliasmith C (2005) An attractor network model of path integration in the rat. J Comput Neurosci 18:183–203

DeAngelis GC, Robson JG, Ohzawa I, Freeman RD (1992) Organization of suppression in receptive fields of neurons in cat visual cortex. J Neurophysiol 68:144–163

Eliasmith C, Anderson CH (2003) Neural engineering: computation, representation and dynamics in neurobiological systems. MIT Press, Cambridge

Fischer BJ (2005) A model of the computations leading to a representation of auditory space in the midbrain of the barn owl. PhD thesis, Washington University, St. Louis

Kuo D, Eliasmith C (2005) Integrating behavioral and neural data in a model of zebrafish network interaction. Biol Cybern 93(3): 178–187

Ma WJ, Beck JM, Latham PE, Pouget A (2006) Bayesian inference with probabilistic population codes. Nat Neurosci 9(11):1432–1438

Rao R (2004) Bayesian computation in recurrent neural circuits. Neural Comput 16(1):1–38

Reynolds JH, Chelazzi L, Desimone R (1999) Competitive mechanisms sub serve attention in macaque areas V2 and V4. J Neurosci 19:1736–1753

Rieke F, Warland D, deRuytervan Steveninick R, Bialek W (1997) Spikes: exploring the neural code. MIT Press, Cambridge, MA

Ringach DL (2010) Population coding under normalization. Vis Res 50(22):2223–2232

Sahani M, Dayan P (2003) Doubly distributional population codes: Simultaneous representation of uncertainty and multiplicity. Neural Comput 15(10):2255–2279

Salinas E, Abbott LF (1994) Vector reconstruction from firing rates. J Comput Neurosci 1:89–107

Shi J, Wielaard J, Sajda P (2006) Analysis of a gain control model of V1: is the goal redundancy reduction? Conference proceedings: ... Annual international conference of the IEEE engineering in medicine and biology society. IEEE Eng Med Biol Soc 1:4991–4994.

Singh R, Eliasmith C (2006) Higher-dimensional neurons explain the tuning and dynamics of working memory cells. J Neurosci 26:3667–3678