Assorted Notes on Radial Basis Functions

Andreas Stöckel
Centre for Theoretical Neuroscience
University of Waterloo

November 29, 2020

Abstract

We discuss a minimal, unconstrained log-Cholesky parametrisation of radial basis functions (RBFs) and the corresponding partial derivatives. This is useful when using RBFs as part of neural network that is either trained in a supervised fashion via error backpropagation, or unsupervised using a homeostasis mechanism. We perform some experiments and discuss potential caveats when using RBFs in this way. Furthermore, we compare RBFs to the Spatial Semantic Pointer similarity that can be used to construct networks with sparse hidden representations resembling those found in RBF networks.

1 Introduction

Radial basis functions (RBFs) are a family of nonlinearities used in artificial neural networks. The activity of an RBF unit (or "neuron") $a(\vec{x}; \vec{\mu})$ depends on the distance of the input $\vec{x} \in \mathbb{R}^m$ to a centre $\vec{\mu} \in \mathbb{R}^m$ assigned to each unit in the network. Typically, RBFs are constructed in such a way that the smaller the distance between $\vec{x}$ and $\vec{\mu}$, the higher the neural activity. Furthermore, there is only a small region in the input space where an individual neuron has significant activity. This makes RBFs particularly useful when constructing sparse, distributed representations of an input $\vec{x}$.

Mathematically, the most general form of a radial basis function is

$$a(\vec{x}; \vec{\mu}) = \varphi\big(d(\vec{x}, \vec{\mu})\big)\,.$$

Here, the function $d(\vec{x}, \vec{y})$ is an arbitrary metric, i.e., a function that is symmetric, obeys the triangle inequality, and is zero for $\vec{x} = \vec{y}$. The function $\varphi : [0, \infty) \longrightarrow \mathbb{R}$ translates the distance returned by d into a neural activity. This results in an activation function $a(\vec{x}; \vec{\mu})$ that—for typical choices of d—is characteristically rotation-symmetric around its centre $\vec{\mu}$.

Gaussian RBFs The canonical choice for φ and d is

$$\varphi(\xi) = \exp(-\xi^2)\,, \text{ and } d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}\,, \tag{1}$$

where d is the so-called Mahalanobis distance (or Mahalanobis metric) with a covariance matrix $\Sigma \in \mathbb{R}^{m \times m}$. The covariance matrix can be used to stretch the input space along arbitrarily rotated coordinate axes. Thus, the neuron can be made more or less sensitive to changes in certain directions of the input space. For $\Sigma = I$ the Mahalanobis distance is equivalent to the Euclidian norm.

An important interpretation of these choices for φ and d is that the corresponding $a(\vec{x})$ is proportional to the probability density function of a multivariate Gaussian distribution:

$$a(\vec{x}; \vec{\mu}, \Sigma) \propto \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \exp\big(-(\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu}) \big)\,. \tag{2}$$

In this interpretation, the covariance matrix Σ describes the rotation and elongation of the iso-density ellipsoid of the Gaussian. The inverse of the covariance matrix, Σ^{-1}, is the so-called *precision* matrix.

Conditions for $\varphi(\xi)$ Of course, the exponential discussed above is not the only possible choice for $\varphi(\xi)$; in principle, this function is completely arbitrary. However, in an RBF network that is meant to be used as a universal function approximator, $\varphi(\xi)$ must be continuous "almost everywhere"[1] and have a non-zero indefinite integral $\int_0^\infty \varphi(\xi)\, d\xi$ (Park and Sandberg, 1991).[2] Note that this formalisation explicitly includes monotonically increasing $\varphi(\xi)$. This can for example result in neurons with minimal activity for $\vec{x} = \vec{\mu}$.

Some authors prefer formalisations of RBFs with a more "natural" set of restrictions. For example, Karayiannis (1999) requires $\varphi(\sqrt{\xi})$ to be strictly positive over $\xi \in (0, \infty)$, its derivative to be strictly negative (i.e., monotonically decreasing), and its second derivative to be as well strictly positive (i.e., the rate of decrease increases over time).

Applications of RBFs Classically, RBFs have been used in single-layer neural networks in which the RBF parameters remain unchanged; only the output layer is learned (Broomhead and Lowe, 1988). In such networks, the RBF parameters may be selected using techniques such as k-means and expectation maximisation (Wu et al., 2012). A simple technique for initialising the means $\vec{\mu}$ is to simply choose them randomly from input samples $\vec{x}_i$ (Wu et al., 2012); other reasonable initialisations include sampling $\vec{\mu}$ uniformly across the represented space or, in our eyes even better yet, selecting $\vec{\mu}$ from a Halton sequence (Chi, Mascagni, and Warnock, 2005).

Outside of neural networks, the so called "RBF kernel" is used in techniques such as support vector machines (Wu et al., 2012).

[1] Being "continuous almost everywhere" is a technical term describing that the set of discontinuities has measure zero, i.e., there are only countably many discontinuities.

[2] The proof by Park and Sandberg is more general, listing requirements for the multivariate kernel function $K(\vec{x} - \vec{\mu})$, and not just $\varphi(\xi)$.

Outline and contributions of this paper When including RBFs in a neural network, where we would like to learn $\vec{\mu}$ and Σ, we need to consider two minor challenges. First, we need to find a minimal and unconstrained parametrisation of Σ that preserves its covariance nature. Second, we need to find the derivatives of $a(\vec{x})$ with respect to $\vec{\mu}$ and Σ^{-1}. We discuss solutions to these challenges in the next two sections. Note that while we did not find these particular solutions in the literature, they are neither complicated nor overly original, so we would be surprised if this had not already been discussed elsewhere.

We follow with a discussion of how backpropagation can be used in conjunction with these gradients to learn the parameters. Furthermore, we discuss experiments that highlight some caveats of learning RBF parameters in this way. We close with a short discussion comparing RBFs to the spatial semantic pointer (SSP) similarity from a theoretical perspective. The SSP similarity shares some properties with RBFs and can similarly be interpreted as the activation of a hidden unit in a neural network.

2 Parametrisation

Representing covariance or precision? A first decision we need to make when parametrising $a(\vec{x})$ is whether we should represent Σ or its inverse Σ^{-1}. Representing Σ^{-1} has the advantage of eliminating a computationally expensive matrix inversion whenever we evaluate $a(\vec{x})$ or any of its derivatives. However, we need to make sure that there is a one-to-one mapping between our parametrisation of Σ^{-1} and the space of all possible covariance matrices Σ. As we will show in a moment, this is indeed possible, so we will pursue a parametrisation that directly represents Σ^{-1}.

Minimal and unconstrained parametrisations Optimally, our parametrisation $\vec{\theta} = (\theta_1, \ldots, \theta_n)$ should be *unconstrained* and *minimal*. "Unconstrained" means that we can set any individual parameter θ_i to an arbitrary value on the real number line without violating the restrictions imposed on the parameters (such as Σ being a covariance matrix). Such an unconstrained representation would be nice, because it eliminates potential post-processing steps after a parameter update that constrain the parameters to their respective subspace.

By "minimal" we mean that the number of parameters n is equal to the degrees of freedom of the equation. We would like to find the smallest number of parameters such that the function space spanned by this parametrisation is equal to the function space spanned by our original parametrisation. In other words, and more loosely, there should be a one-to-one mapping between the parameters and the actual function that is being realized.[3] Such a minimal

[3] In general, requiring a one-to-one mapping may be too strict, since there may be poles in the parametrisation where, when fixing one set of the parameter space, other parameters can be varied without changing the output of the function. One parametrisation of the covariance matrix with this problem would be using rotation angles and radii to describe the ellipsoid (e.g., the "Givens" representation discussed below). When all radii are equal (i.e., the ellipsoid

representation is desirable when performing parameter optimisation. A non-minimal representation implies that the same function can be represented using different parameter combinations. This results in multiple local minima in the loss function and may thus slow down convergence.

Parametrising the precision matrix Σ^{-1} While we can just use the m individual coefficients of the centre vector $\vec{\mu}$ as parameters, the m^2 coefficients of Σ^{-1} do not form an *unconstrained* or a *minimal* parametrisation. First of all, it cannot be unconstrained, because there are Σ^{-1} that are not the inverse of a covariance matrix; in fact there are some Σ^{-1} that cannot be the inverse of any matrix (i.e., if Σ^{-1} itself is non-invertible/singular). Second, it cannot be minimal, because covariance matrices are symmetric, and thus only have $M = \frac{m(m+1)}{2}$ degrees of freedom.

To find a minimal parametrisation of Σ^{-1} and to ensure that this choice is correct, we first need to review a few concepts from linear algebra. Specifically, we will use the fact that the concepts of covariance matrices and symmetric positive semidefinite matrices are equivalent, and that symmetry and positive definitiveness are preserved by inversion.

2.1 Mathematical background

Most proofs and definitions in this section can be found in any undergrad linear algebra textbook. Still, we have decided to include them here for clarity. Perhaps the most interesting lemma presented below is the equivalence of the concepts of "covariance matrix" and "symmetric positive semidefinite matrix".

Definition 1. Covariance and covariance matrix. The covariance $\mathrm{Cov}[X]$ of a set of vectors $X = \{\vec{x}_1, \ldots, \vec{x}_N\}$ is defined as

$$\mathrm{Cov}\left[X\right] = \frac{1}{N}\sum_{i=1}^{N}\left(\vec{x}_i - \bar{x}\right)\left(\vec{x}_i - \bar{x}\right)^T, \qquad \text{where } \bar{x} = \mathbb{E}[X] = \frac{1}{N}\sum_{i=1}^{N}\vec{x}_i\,.$$

In the following, we refer to any matrix that can be written as such an outer product as a covariance matrix. Note that a covariance matrix is square, positive, and symmetric.

Definition 2. Positive definite and positive semidefinite. Let Σ be a square matrix. Σ is positive definite if $\vec{y}^T\Sigma\vec{y} > 0$ for all $\vec{y} \in \mathbb{R}^m \setminus \{0\}$. Σ is positive semidefinite if the more relaxed condition $\vec{y}^T\Sigma\vec{y} \geq 0$ holds for all $\vec{y} \in \mathbb{R}^m$.

Lemma 1. *A matrix Σ is symmetric and positive semidefinite $\Leftrightarrow$ all eigenvalues of Σ are nonnegative.*

Proof. First consider the "$\Rightarrow$" direction. Let Σ be a symmetric positive semidefinite matrix and $\vec{x}$, λ be an eigenvector, eigenvalue pair of Σ. That is, it holds

is spherical), the ellipsoid is rotation-invariant. We will choose a parametrisation that does not have any such poles.

$\Sigma\vec{x} = \lambda\vec{x}$. Then by definition of positive semidefinite, $\vec{x}^T\Sigma\vec{x} = \lambda\vec{x}^T\vec{x} \geq 0$. Since Σ is symmetric, all eigenvalues and eigenvectors must be real and it follows $\lambda \geq 0$. For "$\Leftarrow$", since Σ is symmetric, we can write the matrix using the eigendecomposition as $\Sigma = V\Lambda V^T$, where V is a real matrix of Eigenvectors $\vec{v}_i$ and Λ a diagonal matrix of eigenvalues. Since Λ is nonnegative, we can write $V\Lambda V^T = (V\sqrt{\Lambda})(V\sqrt{\Lambda})^T$. It holds

$$\vec{y}^T\Sigma\vec{y} = \vec{y}^T\big(V\sqrt{\Lambda}\big)\big(V\sqrt{\Lambda}\big)^T\vec{y} = \sum_{i=1}^{m}\big\langle\vec{y}, \big(V\sqrt{\Lambda}\big)_i\big\rangle^2 \geq 0 \text{ for all } \vec{y} \in \mathbb{R}^m\,. \qquad \square$$

Note that for symmetric positive *definite* matrices (note the missing "semi") all eigenvalues are *strictly* positive, and, vice versa, if all eigenvalues of a matrix are strictly positive, then the matrix is positive definite. The proof is analogous to the above.

Lemma 2. *Σ is a covariance matrix $\Leftrightarrow$ Σ is symmetric and positive semidefinite.*

Proof. Again, first consider the "$\Rightarrow$" direction. Let Σ be a covariance matrix. By construction, the matrix is symmetric. For arbitrary $\vec{y}$

$$\begin{aligned}\vec{y}^T\Sigma\vec{y} &= \frac{1}{N}\sum_{i=1}^{N}\vec{y}^T\big(\vec{x}_i - \bar{x}\big)\big(\vec{x}_i - \bar{x}\big)^T\vec{y} = \frac{1}{N}\sum_{i=1}^{N}\big\langle\vec{y}, \big(\vec{x}_i - \bar{x}\big)\big\rangle\big\langle\big(\vec{x}_i - \bar{x}\big), \vec{y}\big\rangle \\ &= \frac{1}{N}\sum_{i=1}^{N}\big\langle\vec{y}, \big(\vec{x}_i - \bar{x}\big)\big\rangle^2 \geq 0\,.\end{aligned}$$

Hence $\vec{y}^T\Sigma\vec{y} \geq 0$ for all $\vec{y} \in \mathbb{R}^m$, which is the definition of positive semidefinite. For "$\Leftarrow$", symmetry and positive semidefiniteness implies that the eigendecomposition $\Sigma = V\Lambda V^T$ exists, where V are real, orthogonal matrices. Since Λ is non-negative (see the above lemma), we can again rewrite the decomposition as $\Sigma = (V\sqrt{\Lambda})\big(V\sqrt{\Lambda}\big)^T$. Let $X = \{\vec{x}_1, \ldots, \vec{x}_{2d}\}$ be a set of $N = 2d$ vectors defined as follows:

$$\vec{x}_{2i-1} = \frac{\sqrt{N\lambda_i}}{2}\vec{v}_i\,, \qquad \vec{x}_{2i} = -\frac{\sqrt{N\lambda_i}}{2}\vec{v}_i\,, \qquad \text{for all } i \in \{1, \ldots, d\}\,.$$

Since each vector is included twice, once with a positive and negative sign, the mean of X is zero and we have $\Sigma = \mathrm{Cov}[X]$. $\square$

Lemma 3. *All symmetric positive definite matrices are invertible. The inverse of a positive definite matrix is still positive definite.*

Proof. Let Σ be a symmetric positive definite matrix, and $\Sigma = V\Lambda V^T$ its eigendecomposition. Since, by the above lemma, the diagonal matrix Λ is strictly positive, it holds $\Sigma^{-1} = V\Lambda^{-1}V^T$. Since Λ^{-1} is still strictly positive Σ^{-1} must also be positive definite. $\square$

Note that one can trivially find an example of a positive semidefinite matrix that is not invertible—hence, to ensure invertibility of a covariance matrix, we must ensure that it is strictly positive definite. In other words, the concept of an invertible covariance matrix, and a symmetric, positive definite matrix are equivalent.

2.2 Representation of positive semidefinite matrices using the log-Cholesky representation

Given the concepts reviewed above, we can safely represent the precision matrix Σ^{-1} as a positive definite matrix. This ensures that, first, Σ^{-1} always corresponds to a covariance matrix. Second, every possible covariance matrix can be represented in this way.

Of course, the question we need to answer now is how to represent a symmetric positive definite matrix in a minimal and unconstrained manner. Symmetry implies that we only have $M = \frac{(m+1)m}{2}$ degrees of freedom, since $(\Sigma)_{ij} = (\Sigma)_{ji}$. We could thus, for example, represent the entire matrix by only storing the upper triangle, including the diagonal. However, this will not result in an unconstrained representation, since it does not ensure positive definiteness.

Minimal, unconstrained representations of invertible covariance matrices have previously been explored by Pinheiro and Bates (1996). In the following, we provide a quick overview of the representations presented in that paper.

Cholesky representation Every symmetric positive definite matrix Σ can be decomposed as $\Sigma = L^T L$, where $(L)_{ij} = \ell_{ij}$ is an upper triangular matrix. This is known as the Cholesky decomposition. We can directly use the upper triangle of L as our parameter vector $\vec{\theta}$. For example, one way of unpacking L into $\vec{\theta}$ is to successively move along the diagonals

$$\vec{\theta} = \big(\underbrace{\ell_{11}, \ell_{22}, \ldots, \ell_{m,m}}_{\text{Main diagonal}}, \underbrace{\ell_{12}, \ell_{23}, \ldots \ell_{m-1,m}}_{1^{\text{st}}\text{ off-diagonal}}, \ldots, \ell_{1,m}\big)\,.$$

One short-coming of this representation is that the diagonal elements of Σ are the sum of squares of each row in L, i.e., $(\Sigma)_{ii} = \sum_{j=1}^{m} \ell_{ij}^2$. We can thus find 2^m different ways in which the same Σ can be encoded by systematically flipping the signs of ℓ_{ij}. As mentioned above, the additional local minima induced by these ambiguities can be problematic when using such a representation for optimization. Furthermore, setting one or more ℓ_{ii} to zero could result in a non-invertible matrix. We could of course constrain the ℓ_{ii} to strictly positive values, but we were explicitly looking for an unconstrained representation.

log-Cholesky representation A solution to both problems mentioned above is to simply represent the diagonal elements of L in terms of their logarithm. The remaining coefficients can be stored as-is. That is, our parameter vector $\vec{\theta}$

takes on the following form

$$\vec{\theta} = \big(\underbrace{\log(\ell_{11}), \log(\ell_{22}), \ldots, \log(\ell_{m,m})}_{\text{Main diagonal}}, \underbrace{\ell_{12}, \ell_{23}, \ldots \ell_{m-1,m}}_{1^{\text{st}}\text{ off-diagonal}}, \ldots, \ell_{1,m}\big)\,. \quad (3)$$

This representation of covariance matrices is simple, fully unconstrained, minimal, and ensures a one-to-one mapping between the parameter space and every symmetric positive definite matrix. This is also the representation we will use in the following; however, for completeness, we will first review the remaining three representations proposed by Pinheiro et al.

Spherical and Givens representation The intuition for these two representations stems from the eigendecomposition of a symmetric positive definite matrix, i.e., $\Sigma = V\Lambda V^T$. Since V is orthogonal, it can, as already alluded to in the introduction, be seen as a rotated coordinate system, and Λ as scaling factors stretching each individual dimension. In an m-dimensional space there are $M - m$ primitive rotations (so called "Givens" matrices) and m independent scaling factors. This would result in a parameter vector $\vec{\theta}$ of the following form:

$$\vec{\theta} = \big(\underbrace{s_1, \ldots, s_m}_{\text{Scaling}}, \underbrace{\alpha_{12}, \alpha_{23}, \ldots, \alpha_{m-1,m}, \ldots, \alpha_{1,m}}_{\text{Rotations}}\big)$$

Pinheiro et al. present an efficient way to compute the entry ℓ_{ij} in the Cholesky factorization without requiring the multiplication of $M - m$ Givens matrices, although this requires some changes to the representation that make it a little less geometrically interpretable. Pinheiro et al. call this the "Spherical representation". Alternatively, one can of course directly use the parameter vector to construct the eigenvectors V and eigenvalues Λ. This is called the "Givens" representation.

These two representations have the upside of being geometrically intuitive. However, a naive representation of the angles and scaling factors would not result in a unique representation—all scaling factors need to be constrained to strictly positive values and all angles need to be restricted to a range of 180°. However, as Pinheiro et al. discuss, both scaling factors and angles can be represented using logarithms to overcome this limitation.

Matrix logarithm The idea behind this representation is to set θ to the the upper triangle of the matrix logarithm $\log(\Sigma)$. To see why this is reasonable, let the eigendecomposition of Σ be $\Sigma = V\Lambda V^T$. Then, it holds $\log(\Sigma) = V \log(\Lambda) V^T$. Hence, as long as we ensure that $\log(\Sigma)$ is symmetric (e.g., by only storing the upper triangle in $\vec{\theta}$), there is a one-to-one mapping between $\log(\Sigma)$ and Σ without putting any constraints on the parameter vector.

The upside of this representation is that it is mathematically elegant, the downside is that it requires a costly matrix exponentiation operation in order to convert $\vec{\theta}$ back to Σ. Requiring matrix exponentiation also makes computing the differentials we need for gradient descent more complicated.

3 Derivatives

We have now established that we can parametrise the precision matrix Σ^{-1} as an M-dimensional parameter vector $\vec{\theta}$ using the log-Cholesky representation. We further need m coefficients to represent the RBF centre $\vec{\mu}$.

When using gradient-based methods to learn these parameters, we must compute the derivative of the neural activities with respect to the parameters $\vec{\theta}$ and $\vec{\mu}$. Additionally, it is sometimes useful to be able to compute the gradient of the function with respect to the input $\vec{x}$—which, due to symmetry of the metric d is equivalent in structure to the derivative with respect to $\vec{\mu}$.

In the following, we derive all three derivatives of the RBF activity $a(\vec{x}; \vec{\mu}, \vec{\theta})$ for an RBF with an arbitrary non-linearity $\varphi(\xi)$ and the Mahalanobis norm, i.e.,

$$a(\vec{x}; \vec{\mu}, \vec{\theta}) = \varphi\big(d(\vec{x}, \vec{\mu}; \vec{\theta})\big) \quad \text{where} \quad d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1}(\vec{\theta})(\vec{x} - \vec{y})}\,.$$

To simplify things, we use the squared metric d in most equations, denoted as

$$d^2(\vec{x}, \vec{y}) = (\vec{x} - \vec{y})^T \Sigma^{-1}(\vec{\theta})(\vec{x} - \vec{y})\,.$$

Applying the chain rule The first step to computing any derivative of $a(\vec{x}; \vec{\mu}, \vec{\theta})$ with respect to $\vec{x}$, $\vec{\mu}$, or $\vec{\theta}$ is to apply the chain rule twice—once for φ and once for the square root. Taking, for example, the derivative with respect to $\vec{x}$ we get

$$\begin{aligned}
\frac{\partial}{\partial \vec{x}} \varphi\big(d(\vec{x}, \vec{\mu}; \vec{\theta})\big) &= \varphi'\big(d(\vec{x}, \vec{\mu}; \vec{\theta})\big) \left(\frac{\partial}{\partial \vec{x}} d(\vec{x}, \vec{\mu}; \vec{\theta}) \right) \\
&= \varphi'\big(d(\vec{x}, \vec{\mu}; \vec{\theta})\big) \left(\frac{\frac{\partial}{\partial \vec{x}} d^2(\vec{x}, \vec{\mu}; \vec{\theta})}{2 d^2(\vec{x}, \vec{\mu}; \vec{\theta})} \right), \quad \text{where } \varphi'(\xi) = \frac{d}{d\xi} \varphi(\xi)\,.
\end{aligned}$$

Of course, the denominator in the above fraction disappears if φ is already defined in terms of the squared metric.

3.1 Derivative with respect to $\vec{x}$, $\vec{\mu}$

The differential with respect to the input vector $\vec{x}$ can be expressed as a $1 \times d$ Jacobian matrix, where each column i corresponds to the derivative with respect to the input dimension x_i. Expanding the partial differential of d^2 with respect

to $\vec{x}$ we obtain

$$\begin{aligned}
\frac{\partial}{\partial \vec{x}} d^2(\vec{x}, \vec{\mu}; \vec{\theta}) &= \frac{\partial}{\partial \vec{x}} (\vec{x} - \vec{\mu})^T \Sigma^{-1}(\vec{\theta})(\vec{x} - \vec{\mu}) \\
&= \underbrace{\left(\frac{\partial}{\partial \vec{x}} (\vec{x} - \vec{\mu})^T \right)}_{= K_{m,1}} \Sigma^{-1}(\vec{\theta})(\vec{x} - \vec{\mu}) + (\vec{x} - \vec{\mu})^T \left(\frac{\partial}{\partial \vec{x}} \Sigma^{-1}(\vec{\theta})(\vec{x} - \vec{\mu}) \right) \\
&= (\vec{x} - \vec{\mu})^T \left(\Sigma^{-1}(\vec{\theta}) \right)^T + (\vec{x} - \vec{\mu})^T \Sigma^{-1}(\vec{\theta}) \\
&= (\vec{x} - \vec{\mu})^T \left(\left(\Sigma^{-1}(\vec{\theta}) \right)^T + \Sigma^{-1}(\vec{\theta}) \right) \\
&= 2(\vec{x} - \vec{\mu})^T \Sigma^{-1}(\vec{\theta}) \,,
\end{aligned}$$

where $K_{m,1}$ is the so called "commutation matrix" with $K_{m,1}\vec{x} = \vec{x}^T$ (see Lütkepohl, 1997, pp. 9, 183). As mentioned above, the derivative with respect to $\vec{\mu}$ is the similar in structure except for the sign. It holds

$$\frac{\partial}{\partial \vec{\mu}} d^2(\vec{x}, \vec{\mu}; \vec{\theta}) = -2(\vec{x} - \vec{\mu})^T \Sigma^{-1}(\vec{\theta}) \,.$$

3.2 Derivative with respect to $\vec{\theta}$

Computing the differential with respect to $\vec{\theta}$ is a little more involved—in particular, we need to take into account that the diagonal elements of Σ^{-1} are represented as a logarithm. Furthermore, each entry k in the vector $\vec{\theta}$ corresponds to a specific row i_k and column j_k. The exact mapping depends on how the matrix Σ^{-1} is vectorized—the mapping from vector index k to matrix cell indices (i_k, j_k) in eq. (3) are just a suggestion. In the following, we write the derivative in its most general form, using the mapped indices (i_k, j_k).

We first write down the matrix-valued derivative of $\Sigma^{-1}(\vec{\theta})$ with respect to $\vec{\theta}$

$$\begin{aligned}
\frac{d}{d\theta_k} \Sigma^{-1}(\vec{\theta}) &= \frac{d}{d\theta_k} L(\vec{\theta})^T L(\vec{\theta}) \\
&= \left(\frac{d}{d\theta_k} L(\vec{\theta})^T \right) L(\vec{\theta}) + L(\vec{\theta})^T \left(\frac{d}{d\theta_k} L(\vec{\theta}) \right) \\
&= \theta'_k \left(\Delta_{j_k, i_k} L(\vec{\theta}) + L(\vec{\theta})^T \Delta_{i_k, j_k} \right) \\
&= \theta'_k \left(\Delta_{j_k, i_k} L(\vec{\theta}) + \left(\Delta_{j_k, i_k} L(\vec{\theta}) \right)^T \right) \,, \qquad (4)
\end{aligned}$$

where $\vec{\theta}'$ and $\Delta_{i,j}$ are defined as

$$\theta'_k = \begin{cases} \exp(\theta_k) & \text{if } i_k = j_k \,, \\ 1 & \text{if } i_k \neq j_k \,, \end{cases} \qquad \left(\Delta_{i,j} \right)_{i',j'} = \begin{cases} 1 & \text{if } i = i' \text{ and } j = j' \\ 0 & \text{otherwise} \,. \end{cases}$$

In other words, $\Delta_{i,j}$ is the matrix with all-zero entries except for the cell at row i, and column j, which is set to one. In general, multiplying $\Delta_{i,j}$ with a

matrix A from the right can be used to take the jth row of A and copying it into the ith row of the result. Hence, the above expression can be interpreted as taking the i_kth row of $\theta'_k L(\vec{\theta})$ and placing it twice in the resulting matrix. Once as a row-vector in the j_kth result row, and once as a column-vector in the j_kth result column.

Multiplying eq. (4) by $(\vec{x} - \vec{\mu})$ from both sides and expanding yields

$$\begin{aligned}\frac{\partial}{\partial\theta_k} d^2(\vec{x}, \vec{\mu}; \vec{\theta}) &= (\vec{x} - \vec{\mu})^T \left(\frac{d}{d\theta_k}\Sigma^{-1}(\vec{\theta})\right)(\vec{x} - \vec{\mu}) \\ &= \theta'_k (\vec{x} - \vec{\mu})^T \left(\Delta_{j_k,i_k} L(\vec{\theta}) + \left(\Delta_{j_k,i_k} L(\vec{\theta})\right)^T\right)(\vec{x} - \vec{\mu}) \\ &= \theta'_k \left((\vec{x} - \vec{\mu})^T \Delta_{j_k,i_k} L(\vec{\theta})(\vec{x} - \vec{\mu}) + (\vec{x} - \vec{\mu})^T \left(\Delta_{j_k,i_k} L(\vec{\theta})\right)^T (\vec{x} - \vec{\mu})\right) .\end{aligned}$$

Both summands in the above equation evaluate to the same value. We get

$$\frac{\partial}{\partial\theta_k} d^2(\vec{x}, \vec{\mu}; \vec{\theta}) = 2\theta'_k \big(\vec{x} - \vec{\mu}\big)_{j_k} \left\langle \big(L(\vec{\theta})\big)_{i_k}, \vec{x} - \vec{\mu} \right\rangle .$$

When implementing these equations as a computer program, it may be convenient to compute the entire gradient vector at once. To this end, we replace the row-vector $(L(\vec{\theta}))_{i_k}$ with an $M \times m$ matrix $\tilde{L}(\vec{\theta})$, where the kth row of $\tilde{L}(\vec{\theta})$ is simply $(L(\vec{\theta}))_{i_k}$. Similarly, the vector coefficients of $\vec{x}$, $\vec{\mu}$ can be rearranged according to j_k into new M-dimensional vectors $\tilde{x}$, $\tilde{\mu}$. We get

$$\frac{\partial}{\partial\vec{\theta}} d^2(\vec{x}, \vec{\mu}; \vec{\theta}) = 2\vec{\theta'} \odot \big(\tilde{x} - \tilde{\mu}\big) \odot \tilde{L}(\vec{\theta})\big(\vec{x} - \vec{\mu}\big) .$$

Here, "$\odot$" denotes the Hadamard (element-wise) product.

4 Applications and Experiments

Backpropagation in a two-layer network Given the derivatives computed above we can implement algorithms such as error backpropagation. For example, consider a typical two-layer RBF network with m-dimensional input, n hidden units, and k outputs, as well as a set of N training samples $\{(\vec{x}_1, \vec{t}_1), \ldots, (\vec{x}_N, \vec{t}_N)\}$. We can optimize the least-squares loss

$$E = \frac{1}{N}\sum_{i=1}^{N} \big(\vec{y}(\vec{x}_i) - \vec{t}_i\big)^2 = \frac{1}{N}\sum_{i=1}^{N} \big(W\vec{a}(\vec{x}; M, \Theta) - \vec{t}_i\big)^2 .$$

Here, $\vec{y}(\vec{x})$ describes the network output, $W \in \mathbb{R}^{k\times n}$ the readout weights, and $\vec{a}$ the activities of the hidden layer. $M \in \mathbb{R}^{n\times m}$ and $\Theta \in \mathbb{R}^{n\times M}$ are matrices of RBF centres $\vec{\mu}_j$ and log-Cholesky precision matrix parameters $\vec{\theta}_j$.

The readout weights W can be trained using the delta learning rule. The RBF parameters are trained by backpropagating the error $\vec{\varepsilon}_j = (\vec{t}_i - \vec{y}_i)$ through

W^T, resulting in the per-hidden-unit error $\tilde{\varepsilon}_i = W^T \vec{\varepsilon}_i$. The update to $\vec{\theta}_j$ and $\vec{\mu}_j$ for $j \in \{1, \ldots, n\}$ is proportional to the product of $\tilde{\varepsilon}_{ij}$ and the above derivatives,

$$\Delta\vec{\mu}_j = -\frac{\eta_1}{N} \sum_{i=1}^{N} \tilde{\varepsilon}_{ij} \frac{\partial}{\partial \vec{\mu}_j} a_j(\vec{x}; \vec{\mu}_j, \vec{\theta}_j)\,, \quad \Delta\vec{\theta}_j = -\frac{\eta_2}{N} \sum_{i=1}^{N} \tilde{\varepsilon}_{ij} \frac{\partial}{\partial \vec{\theta}_j} a_j(\vec{x}; \vec{\mu}_j, \vec{\theta}_j)\,, \quad (5)$$

where η_1 and η_2 are two independent learning rates for the centres and precision matrix parameters.

Similar backpropagation based techniques have been thoroughly explored by other researchers, though the publications encountered by the author of this technical report either do not learn the precision or covariance matrix at all (Karayiannis, 1999), or assume that Σ is a diagonal matrix (i.e., only the variances are learned; Wu et al., 2012; Schwenker, Kestler, and Palm, 2001). In addition, and as mentioned in the introduction, RBFs are classically pre-trained in conjunction with specialised algorithms such as k-means and expectation maximisation (Schwenker, Kestler, and Palm, 2001). However, learning the centres and the full covariance matrices outside of a back-propagation step in such a specialised manner is only possible in a two-layer neural network.

Using linlog**- instead of** log**-Cholesky** One potential problem with the log-Cholesky representation is that exponentiation of the first m elements in the parameter vector $\vec{m}$ may lead to excessively large gradients, making the optimization problem less stable. This could theoretically be solved by replacing the logarithm with a piecewise linear-logarithmic function (hereby dubbed "linlog"):

$$\operatorname{linlog}(\xi) = \begin{cases} \xi - 1 & \text{if } \xi > 1\,, \\ \log(\xi) & \text{if } \xi \leq 1\,, \end{cases} \quad \operatorname{linlog}^{-1}(\xi) = \begin{cases} \xi + 1 & \text{if } \xi > 0\,, \\ \exp(\xi) & \text{if } \xi \leq 0\,. \end{cases} \quad \text{(Linlog)}$$

Our experiments (see below) indicate that there is virtually no difference between linlog and log with respect to the convergence behaviour during gradient descent. We correspondingly recommend using the more canonical log-Cholesky.

Caveats when learning $\vec{\mu}$ and $\vec{\theta}$ using supervised gradient descent Learning both $\vec{\mu}$ and the precision matrix parametrised by $\vec{\theta}$ at the same time comes with some caveats that are best explored in a simplified setup.

Consider learning the parameters $\vec{\mu}$, $\vec{\theta}$ of a single quadratic RBF with $\varphi(\xi) = \xi^2$. To learn these parameters we assume that there is a "supervising" RBF with ground-truth parameters $\vec{\mu}_{\text{gt}}$ and $\vec{\theta}_{\text{gt}}$. We sample $\vec{x}$ from the Gaussian distribution corresponding to the ground-truth parameters and use gradient descent to minimize the quadratic loss function between the activities of the ground-truth and learned unit

$$E = \frac{1}{N} \sum_{i=1}^{N} \left(a(\vec{x}; \vec{\mu}_{\text{gt}}, \vec{\theta}_{\text{gt}}) - a(\vec{x}; \vec{\mu}, \vec{\theta}) \right)^2, \quad \text{where } \vec{x} \sim \mathcal{N}\big(\vec{\mu}_{\text{gt}}, \Sigma(\vec{\theta}_{\text{gt}})\big)\,. \quad (6)$$

Put differently, we are learning the parameters of the probability distribution underlying $\vec{x}$ using the additional information encoded in the activities $a(\vec{x}; \vec{\mu}_{\mathrm{gt}}, \vec{\theta}_{\mathrm{gt}})$. Although this seems to be a very simple problem, doing this naively as outlined above tends to not work very well. In particular, the convergence of the system to the correct parameters is extremely slow.

An example demonstrating the slow convergence is depicted in fig. 1. It takes about 35 000 iterations to learn the five parameters describing the Mahalanobis distance in two-dimensional space using a minibatch size of 100 and a learning rate $\eta_1 = \eta_2 = 10^{-3}$. Using a smaller batch size results in slightly faster convergence (about 25 000 iterations), at the cost of being more unstable. This is illustrated in fig. 2, which reports the error statistics for this experiment over one thousand trials and the percentage of "failed" (diverged) trials over time.

The reason for this slow convergence is the mismatch between the covariance estimate $\Sigma(\vec{\theta})$ and the ground-truth covariance $\Sigma(\vec{\theta}_{\mathrm{gt}})$. In this particular learning setup, this mismatch leads to the estimated mean $\vec{\mu}$ being systematically biased towards a "phantom mean". This phenomenon is illustrated in fig. 3. If $\vec{\theta}_{\mathrm{gt}}$ and $\vec{\theta}$ are equal, the loss function E has a global minimum at $\vec{\mu}_{\mathrm{gt}}$ just after averaging three samples. For mismatched $\vec{\theta}_{\mathrm{gt}}$ and $\vec{\theta}$ this is not the case; a relatively large number of samples must be averaged for a reasonable estimate, and even then, it is not guaranteed that the global minimum will converge to the correct mean.

Making matters worse, an incorrect estimate of $\vec{\mu}$ in turn leads to an erroneous covariance estimate $\Sigma(\vec{\theta})$. Even in this simple case, this may lead to the parameters $\vec{\theta}$, $\vec{\mu}$ being "stuck" in a local minimum.

It should be noted that these results only have limited relevance when considering learning in a neural *network*. However, these considerations do highlight that the relatively large number of parameters per unit (compared to a single m-dimensional weight vector in a "standard" two-layer network) can induce sub-optimal local minima in the loss function. Some gentle source of stochasticity is needed in the optimizer to overcome these minima without causing divergence of the parameter set.

Unsupervised learning of $\vec{\mu}$ and $\vec{\theta}$ One creative solution to the particular learning problem considered above is unsupervised learning of the parameters $\vec{\mu}$ and $\vec{\theta}$. Remember that we sample $\vec{x}$ from the ground-truth distribution $\mathcal{N}\big(\vec{\mu}_{\mathrm{gt}}, \Sigma(\vec{\theta}_{\mathrm{gt}})\big)$. Hence, the parameters we are trying to estimate are implicitly encoded in the input time-series. We can try to extract these parameters without a supervised reference activity $a_{\mathrm{gt}} = a(\vec{x}; \vec{\mu}_{\mathrm{gt}}, \vec{\theta}_{\mathrm{gt}})$.

To learn the distribution of the input we must simply use the following (independent) errors for the mean and for theta in eq. (5)

$$\tilde{\varepsilon}_{\vec{\mu}} = a(\vec{x}; \vec{\mu}, \vec{\theta}) - \varphi(0)\,, \qquad \tilde{\varepsilon}_{\vec{\theta}} = a(\vec{x}; \vec{\mu}, \vec{\theta}) - \sigma\,, \tag{7}$$

where σ is a parameter that scales the learned covariance matrix. We found through experimentation that in the setup described above (i.e., $\varphi(\xi) = \xi^2$) a value of $\sigma \approx 4$ results in $\vec{\theta} \approx \vec{\theta}_{\mathrm{gt}}$; further mathematical investigation into the nature of this parameter is required.

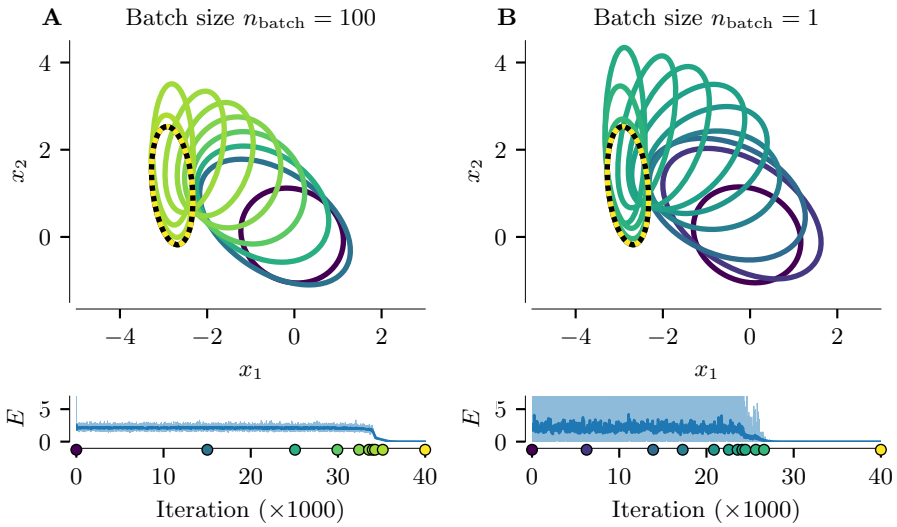

Figure 1: Learning $\vec{\mu}$ and $\vec{\theta}$ from a reference distribution parametrised by $\vec{\mu}_{\text{gt}}$ and $\vec{\theta}_{\text{gt}}$. *Top:* The black dotted ellipse is a visualisation of the reference distribution. Coloured ellipses depict the learning progress over time (ellipses are equidistant in space), colour indicates the iteration. *Bottom:* Light blue line is the mean error E over the minibatch. Darker blue line is a low-pass filtered version of the error. Coloured circles correspond to the iteration in which the corresponding ellipse was drawn. **(A)** Data for a minibatch size of 100, i.e., the error and gradient are averaged over 100 samples before updating the parameters. **(B)** Data for a minibatch size of 1, i.e., each sample $\vec{x}$ is processed individually.

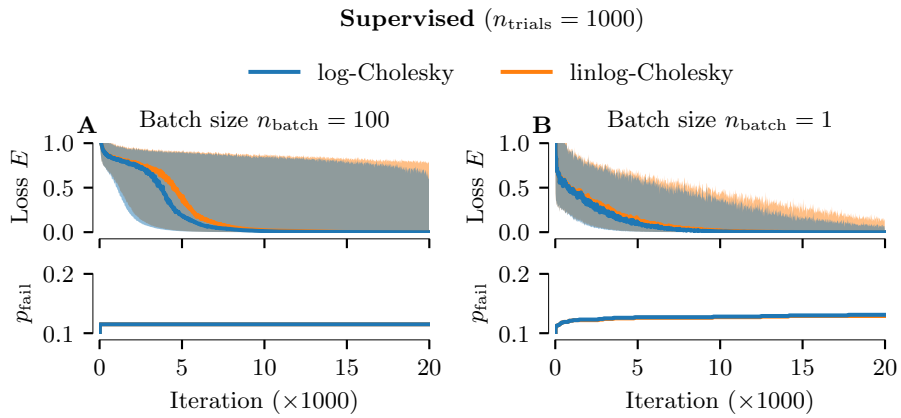

Figure 2: Experiment from fig. 1 repeated 1000 times. *Top:* Median loss. Filled regions indicate the quartiles (25%- and 75%-percentiles). *Bottom:* Fraction of trials p_{fail} that diverged up to this point, indicating the stability of the optimization. Small batch sizes tend to converge faster but are less stable.

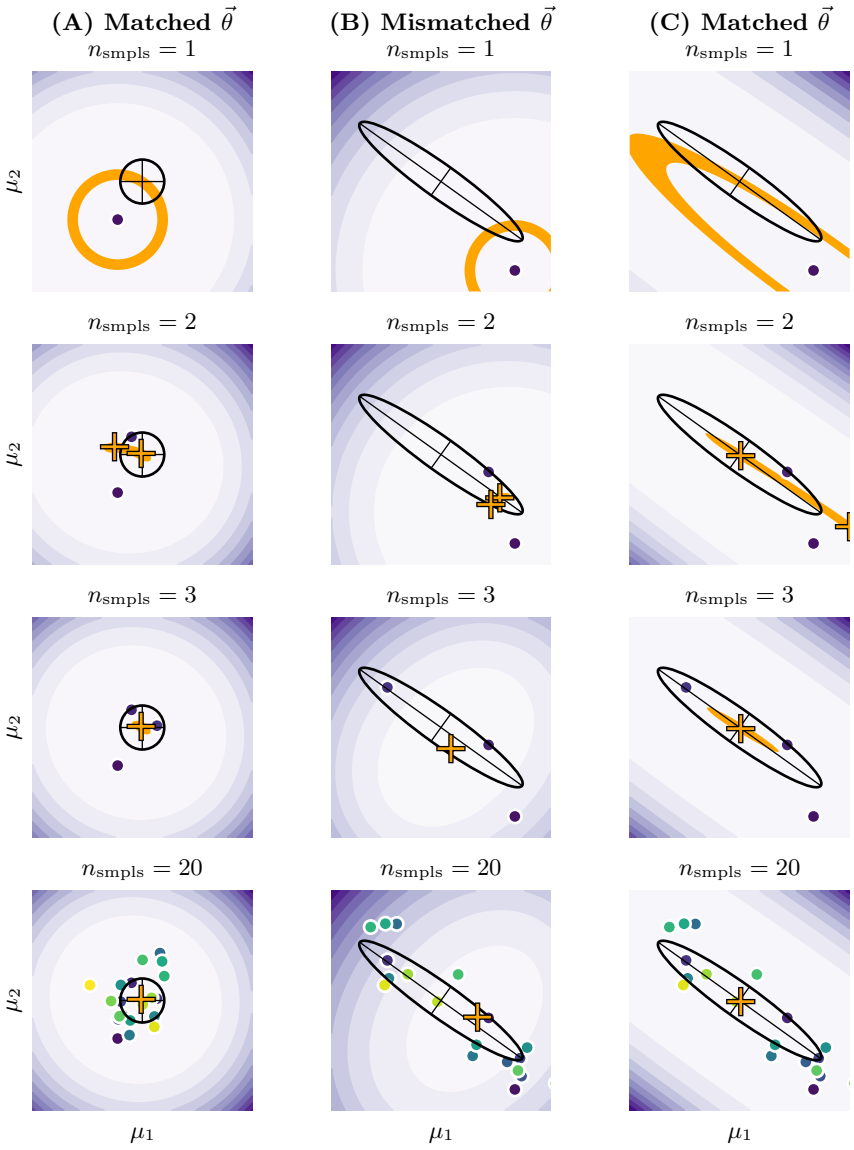

Figure 3: Mean estimation error when learning the RBF centre. The black ellipse is the Gaussian ground-truth distribution underlying the samples $\vec{x}$ (small purple, green, and yellow circles). Gradient in the background is the loss function $E(\vec{\mu})$ over all samples for a fixed $\vec{\theta}$ (eq. (6); lighter colours are smaller errors). Orange regions correspond to regions with minimum error, orange crosses are local minima (i.e., the estimated mean). **(A, C)** Matching estimated covariance $\vec{\theta}$ and ground-truth $\vec{\theta}_{\text{gt}}$. The estimated mean matches the true mean at $n_{\text{smpls}} = 3$. **(B)** Mismatched $\vec{\theta}$ and $\vec{\theta}_{\text{gt}}$. Convergence is much slower.

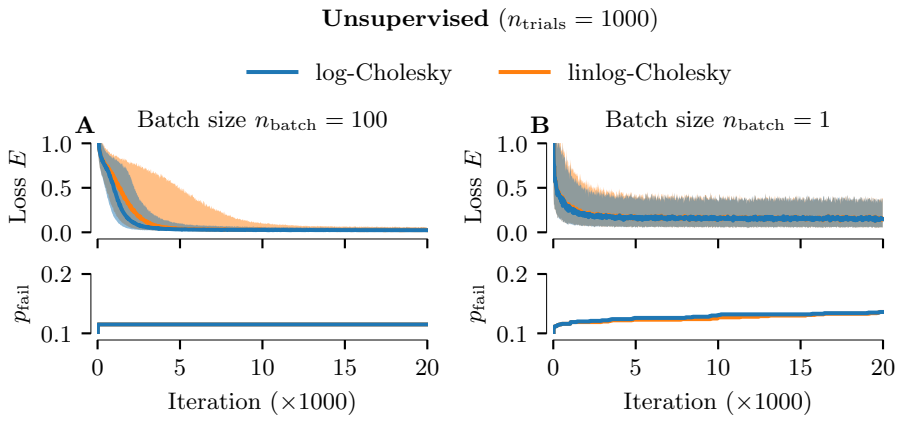

Figure 4: Experiment from fig. 2 repeated with the unsupervised learning rule from eq. (7). *Top:* Convergence of $\vec{\mu}$, $\vec{\theta}$ to the input distribution is much faster compared to the supervised learning rule, at least for larger batch sizes. *Bottom:* Noise can cause the learned parameters to diverge if $n_{\text{batch}} = 1$, even after a good parameter estimate has been reached (note the steady increase in p_{fail}).

Intuitively, Equation (7) can be interpreted as follows. The error term $\varepsilon_{\vec{\mu}}$ expresses that we would like to adjust the mean in such a way that, on average, the activity of the neuron is equal to that of a neuron with $\vec{\mu} = \vec{\mu}_{\text{gt}}$, i.e., $a(\mathbb{E}[\vec{x}]; \vec{\mu}_{\text{gt}}, \vec{\theta}) = \varphi(0)$. With respect to $\varepsilon_{\vec{\theta}}$, we adjust $\vec{\theta}$ such that, on average, the neural activity reaches a certain target value σ. This forces the covariance matrix represented by $\vec{\theta}$ to be scaled to a fixed value that is proportional to the variance of the input samples. With some goodwill such an unsupervised learning rule can be interpreted as a "homeostasis" mechanism that regulates the neuron parameters to reach a certain average activity.

Figure 4 depicts the results of an experiment demonstrating the feasibility of the unsupervised learning rule. Convergence of the parameters is much faster than the supervised version of the same setup, though for small minibatch sizes the learning rate likely needs to be reduced to prevent divergence of the parameter estimates due to random noise.

Still, keep in mind that we only considered a single unit; unsupervised learning of the input distribution in a single-layer neural *network* comes with its own set of challenges.

5 RBFs and the SSP similarity

Spatial Semantic Pointers (SSPs; Komer et al., 2019) encode m-dimensional spatial information in an m'-dimensional vector space, where $m' \gg m$. SSPs can be used to represent continuous values in vector-symbolic architectures (Gayler, 2003) and suggest interesting biological interpretations relating them to grid-

and place-cells (Komer et al., 2019; Komer, 2020; Dumont and Eliasmith, 2020).

A crucial property of SSPs is that summing two SSPs does not correspond to a transformation of the represented spatial information. Instead, summing two SSPs results in a new SSP representing both points. In other words, if an SSP $\vec{a} \in \mathbb{R}^{m'}$ represents the vector $\vec{x} \in \mathbb{R}^m$, and another SSP $\vec{b} \in \mathbb{R}^{m'}$ represents the vector $\vec{y} \in \mathbb{R}^m$, then $\vec{a} + \vec{b}$ will represent the set $\{\vec{x}, \vec{y}\}$. This concept can be extended to representing entire regions $\mathcal{R} \subset \mathbb{R}^m$ by using integration instead of summation.

We write the SSP representation of a region $\mathcal{R} \subset \mathbb{R}^m$ as $\mathrm{S}(\mathcal{R})$. However, note that S is not unique. Each vector-component of the m-dimensional represented space is encoded using a randomly chosen basis SSP vector (see Komer et al., 2019 for more information).

To query whether $\vec{x}$ is represented within an SSP $\vec{a} = \mathrm{S}(\mathcal{R})$, we can simply use the cosine similarity between $\vec{a}$ and $\vec{b} = \mathrm{S}(\{\vec{x}\})$. If $\langle \vec{a}, \vec{b} \rangle$ (assuming normalised $\vec{a}$, $\vec{b}$) is larger than a certain threshold, then $\mathcal{R}$ is likely to contain $\vec{x}$. The "likely" is necessary because of SSP periodicity; we discuss this in more detail in the context of hexagonal SSPs below.

Voelker (2020) proves that the dot product between two SSPs representing $\vec{x}$, $\vec{y}$, respectively, is a product of "sinc functions"

$$\mathbb{E}\left[\frac{\langle \mathrm{S}(\{\vec{x}\}), \mathrm{S}(\{\vec{y}\}) \rangle}{\|\vec{x}\| \|\vec{y}\|}\right] = \prod_{i=1}^{m} \mathrm{sinc}(x_i - y_i)\,. \tag{8}$$

This equation holds for $m' \to \infty$, and the expectation value is over all possible SSP bases. The "sinc" function is defined as

$$\mathrm{sinc}(\xi) = \begin{cases} \frac{\sin(\pi\xi)}{\pi\xi} & \text{if } \xi \neq 0\,, \\ 1 & \text{if } \xi = 0\,. \end{cases} \tag{Sinc}$$

Equation (8) is depicted in Figure 5A for a two-dimensional represented quantity.

5.1 Comparison between the SSP similarity and RBFs

Crucially, the SSP similarity can be interpreted as the activity of a single neuron within a neural network, just as we interpret a single RBF as the activity of an artificial neuron. Both RBFs and the SSP similarity can be used to construct sparse representations. Indeed, considering an RBF with $\varphi(\xi) = \mathrm{sinc}(\xi)$ and Eucledian d results in a response curve that superficially is similar to the SSP similarity (cf. fig. 5B). In particular, both nonlinearities feature a circular "bump" where $\vec{x} \approx \vec{\mu}$, and for axis-aligned $\vec{x}$ (i.e., $\vec{x}$ is a scaled version of one of the basis vectors), the sinc-RBF activities and the SSP similarity are equivalent.

Still, using the SSP similarity has two potential benefits over RBFs. First, individual neurons are able to represent arbitrary regions in space. Second, once all vectors have been encoded as SSP representations, computing the SSP similarity is merely a matter of computing a simple dot product.

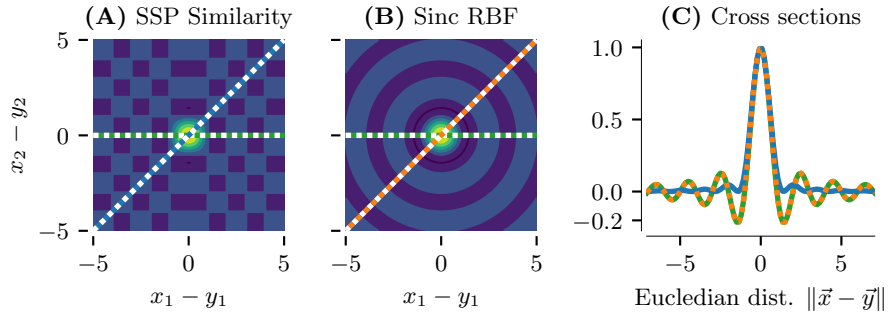

Figure 5: SSP cosine similarity compared to a sinc-RBF. **(A)** Similarity between two SSP vectors representing a two-dimensional quantity. **(B)** Activity of an RBF with sinc-nonlinearity and Euclidean metric. **(C)** Cross sections along the diagonal and x-axis, as highlighted in the previous two sub-figures. The defining property of the RBF is that both cross-sections are exactly the same when plotted over the underlying distance metric. This is not the case with the SSP similarity. Note the attenuation and the increase in frequency in the diagonal cross-section of the SSP similarity compared to the sinc-RBF.

Upon closer inspection however, it should be pointed out that there are stark qualitative differences between the RBF similarity and SSPs. Most importantly, the product of sinc functions is not rotation symmetric for arbitrary angles. Of course, changing the covariance matrix Σ in the Mahalanobis distance or using a metric other than L_2, such as L_1 or L_∞, would result in a pattern in fig. 5B that is not circular (and thus equally not arbitrarily rotation symmetric), but plotting the activity along cross-sections passing through $\vec{\mu}$ over the distance from the centre according to the underlying metric d would always result in the same graph (cf. fig. 5C).

In fact, we claim that the SSP similarity *cannot* be expressed in the very loose sense in which we defined RBFs in the introduction. That is, there exists no function $\varphi(\xi) : [0, \infty) \longrightarrow \mathbb{R}$ and no metric $d(\vec{x}, \vec{y}) : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow [0, \infty)$ such that $\varphi(d(\vec{x}, \vec{y}))$ is equal to eq. (8). As it stands, we do not have a rigorous proof for this. Intuitively however, consider eq. (8) for $d = 2$:

$$\frac{\sin\big(\pi(x_1 - y_1)\big)}{\pi(x_1 - y_1)} \frac{\sin\big(\pi(x_2 - y_2)\big)}{\pi(x_2 - y_2)} = \frac{\sin(\Delta_1)\sin(\Delta_2)}{\Delta_1 \Delta_2} \,.$$

Of both d and φ, only d has information about the difference between the two vector components, Δ_1 and Δ_2. There is no way to express $\sin(\Delta_1)\sin(\Delta_2)$ purely in terms of $\Delta_1\Delta_2$. Hence, d must compute something akin to the product of sines. However, such a function d would violate the triangle inequality and cannot be a metric.

Still, given these theoretical discrepancies, it is uncertain whether there is a practical difference in terms of computational power or convergence behaviour

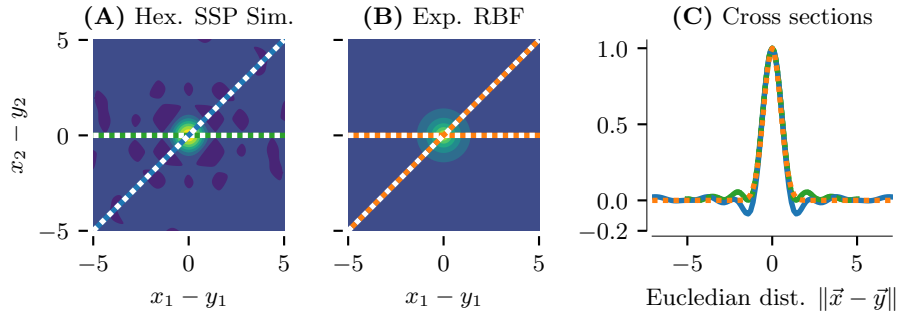

Figure 6: Hexagonal SSP similarity compared to an exponential RBF. See fig. 5 for a more detailed description. Data in **(A)** are the mean over the SSP similarity obtained for 100 random hexagonal SSP bases with $m' = 256$.

between a neural network built with units computing a product-of-sinc SSP similarity, and sinc-RBF networks. More research is needed in this direction.

5.2 Comparison between hexagonal SSPs and RBFs

A small point in favour of there being no practical difference between RBFs and SSP similarities—at least when considering biological constraints on SSP representations—are so-called hexagonal SSPs. To understand the purpose of hexagonal SSPs, we first need to review what we mean when saying that SSPs are "periodic", as well as the concept of "grid cells".

We mentioned above that SSPs are periodic (Komer, 2020, Section 3.2.3). Mathematically, we can express this periodicity by stating that for every $\varepsilon > 0$ and every $\vec{x} \in \mathbb{R}^m$ there exists a scalar $|\gamma| > 1$ such that $\|\mathrm{S}(\{\vec{x}\}) - \mathrm{S}(\{k\gamma\vec{x}\})\| < \frac{\varepsilon}{k}$ for all $k \in \mathbb{N} \setminus \{0\}$.[4] In other words, we cannot really distinguish between whether an SSP represents a vector $\vec{x}$ or a multiple of that vector $k\gamma\vec{x}$. The period γ depends on the ratio between m and m', as well as the particular choice of SSP basis vectors. For large m' the period is typically large enough to be negligible. If we deliberately set m' to a small value, we can exploit periodicity to generate what is known in the neuroscience literature as "grid cells".

Grid cells are neurons that are tuned to the location of an animal within an environment. They are active whenever the animal is located at a "grid point", where "grid" refers to a virtual tiling of the environment that is generated by the animal's brain. While different populations of grid cells are tuned to scaled or shifted versions of that tiling, the tiling itself tends to follow an hexagonal pattern, and such hexagonal patterns have been shown to be optimal

[4] This seemingly complicated formalisation is required since the ratio between two SSP basis vector components can be irrational, in which case strict equality between $\mathrm{S}(\{\vec{x}\})$ and $\mathrm{S}(\{k\gamma\vec{x}\})$ does not hold. However, the distance between $\mathrm{S}(\{\vec{x}\})$ and $\mathrm{S}(\{k\gamma\vec{x}\})$ at most increases linearly with k, hence the division by k on the right-hand-side of the inequality.

for location representation (Mathis, Herz, and Stemmler, 2012). In contrast, the grid generated by the periodicity of SSP representations is rectangular.

Komer (2020) (cf. Section 3.2.4) suggests a simple technique for encoding two-dimensional vectors as SSP representations that results in the periodic activity pattern to be hexagonal, matching the grid cell data. Crucially for the point of this paper, when using this encoding, the "ripples" of the sinc function are smoothed out. The mean hexagonal SSP similarity over 100 random bases is depicted in fig. 6 in comparison to a exponential RBF with $\varphi(\xi) = \exp(-\sqrt{2}\xi)$. Both functions look very similar, except for the hexagonal SSP similarity having a slight "Mexican hat" shape (i.e., negative overshoot surrounding the central "bump").

To summarise, when using a biologically motivated variant of SSP encoding for two-dimensional spaces, the SSP similarity and exponential RBFs are essentially equivalent for all practical purposes. However, it should be noted that this statement is very different from saying that SSP and RBF *representations* of a vector space $\mathbb{R}^m$ are equivalent; they clearly are not, and the reader is directed at Komer (2020) for comparisons between the two.

Acknowledgements

The author would like to thank Chris Eliasmith for his comments on an earlier draft of this document, as well as pointing out the stronger resemblance between the hexagonal SSP similarity and RBFs.

References

Broomhead, David S and David Lowe (1988). *Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks*. Royal Signals and Radar Establishment Malvern (United Kingdom).

Chi, H., M. Mascagni, and T. Warnock (2005). "On the Optimal Halton Sequence". In: *Mathematics and Computers in Simulation* 70.1, pp. 9–21. ISSN: 0378-4754. DOI: 10.1016/j.matcom.2005.03.004.

Dumont, Nicole Sandra-Yaffa and Chris Eliasmith (2020). "Accurate Representation for Spatial Cognition Using Grid Cells". In: *42nd Annual Meeting of the Cognitive Science Society*. Toronto, ON: Cognitive Science Society, pp. 2367–2373.

Gayler, Ross (2003). "Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience". In: *Proceedings of the ICCS/ASCS International Conference on Cognitive Science*. ICCS/ASCS International Conference on Cognitive Science. Sydney, Australia: University of New South Wales, pp. 133–138.

Karayiannis, N. B. (1999). "Reformulated Radial Basis Neural Networks Trained by Gradient Descent". In: *IEEE Transactions on Neural Networks* 10.3, pp. 657–671. DOI: 10.1109/72.761725.

Komer, Brent (2020). "Biologically Inspired Spatial Representation". PhD thesis. Waterloo, ON: University of Waterloo. URL: https://uwspace.uwaterloo.ca/handle/10012/16430.

Komer, Brent et al. (2019). "A Neural Representation of Continuous Space Using Fractional Binding". In: *41st Annual Meeting of the Cognitive Science Society*. Montreal, QC: Cognitive Science Society.

Lütkepohl, Helmut (1997). *Handbook of Matrices*. Chichester, England: John Wiley & Sons. 320 pp. ISBN: 978-0-471-97015-6.

Mathis, Alexander, Andreas V. M. Herz, and Martin Stemmler (2012). "Optimal Population Codes for Space: Grid Cells Outperform Place Cells". In: *Neural Computation* 24.9, pp. 2280–2317. DOI: 10.1162/NECO_a_00319.

Park, J. and I. W. Sandberg (1991). "Universal Approximation Using Radial-Basis-Function Networks". In: *Neural Computation* 3.2, pp. 246–257. DOI: 10.1162/neco.1991.3.2.246.

Pinheiro, José C. and Douglas M. Bates (1996). "Unconstrained Parametrizations for Variance-Covariance Matrices". In: *Statistics and Computing* 6.3, pp. 289–296. ISSN: 1573-1375. DOI: 10.1007/BF00140873.

Schwenker, Friedhelm, Hans A. Kestler, and Günther Palm (2001). "Three Learning Phases for Radial-Basis-Function Networks". In: *Neural Networks* 14.4, pp. 439–458. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(01)00027-2.

Voelker, Aaron R. (2020). *A Short Letter on the Dot Product between Rotated Fourier Transforms*. arXiv: 2007.13462 [q-bio.NC].

Wu, Yue et al. (2012). "Using Radial Basis Function Networks for Function Approximation and Classification". In: *ISRN Applied Mathematics* 2012. Ed. by M. Sun, E. Kita, and E. Gallopoulos, p. 324194. DOI: 10.5402/2012/324194.