Journal of Machine Learning Research 18 (2017) 1-43 Submitted 8/15; Revised 9/16; Published 3/17

Using Conceptors to Manage Neural Long-Term Memories for Temporal Patterns

Herbert Jaeger H.JAEGER@JACOBS-UNIVERSITY.DE
Dpt. of Computer Science and Electrical Engineering
Jacobs University Bremen
28759 Bremen, Germany

Editor: Nando de Freitas

Abstract

Biological brains can learn, recognize, organize, and re-generate large repertoires of temporal patterns. Here I propose a mechanism of neurodynamical pattern learning and representation, called *conceptors*, which offers an integrated account of a number of such phenomena and functionalities. It becomes possible to store a large number of temporal patterns in a single recurrent neural network. In the recall process, stored patterns can be morphed and "focussed". Parametric families of patterns can be learnt from a very small number of examples. Stored temporal patterns can be content-addressed in ways that are analog to recalling static patterns in Hopfield networks.

Keywords: Recurrent neural network, temporal pattern learning, neural long-term memory, neural dynamics

1. Introduction

In the cognitive and neurosciences as well as in neural computation it is customary to distinguish between various sorts of short-term memory and long-term memory (Fusi and Wang, 2016). This article is concerned with neural long-term memory (LTM) in the sense that some information becomes permanently coded in synaptic weights at learning time, to be somehow recalled at exploitation time without changing those weights.

The paradigmatic model of a neural LTM is the *associative memory* pioneered, among others, by Willshaw et al. (1969), Cooper (1973), Kohonen (1974), Palm (1980) and Hopfield (1982). This classical family of models explains how a number of different patterns can be stored in a neural network by an explicit calculation of the network weight matrix or by iterative learning processes. A rich mathematical theory affords insights into capacity bounds and noise robustness. But, these classical associative memories are mainly devised for storing *static* patterns, for instance bit vectors or images. Biological neural systems obviously can learn and recall rich repertoires of *temporal* patterns, for instance gestures, melodies, words. It would be desirable to have a neural network model which extends the capabilities of associative networks to the domain of temporal patterns. This challenge is still largely open. I am aware of only a few proposals for neural memories of temporal patterns:

1. The classical associative memory model for static patterns has been extended on several occasions to learn sequences of patterns. The key mechanism is to replace the

learning of auto-associations of stored patterns (which results in fixed-point attractors representing these) by a learning of hetero-associations of a temporal pattern frame with its successor frame. This leads to recall episodes which step through a discrete sequence of patterns (Amari, 1972). The basic discrete hetero-associative memory mechanism has been elaborated in various ways to accomodate analog values, smoother temporal development, or identical patterns at different time slices (examples: Sompolinsky and Kanter (1986); Rinkus (1996); Billard and Hayes (1999); Huang and Hagiwara (2002)). A recent extension of hetero-associative symbol sequence coding networks (Jiang et al., 2016) integrates methods from graph-based coding theory, which leads to improved memory capacities and temporal pattern completion robustness, and enables an incremental storing of patterns.

2. In the field of reservoir computing, a temporal pattern is trained into a recurrent neural network (RNN) by adapting only the weights from the ("reservoir") network to the output neuron. Thus an arbitrary number of temporal patterns can be learnt on the basis of a single reservoir network if individual patterns are represented by separate output neurons, as for example in Hinaut and Dominey (2011).

3. Since the advent of RNN models in cognitive modeling it has been recognized that RNNs can be trained to generate several temporal patterns when they are paired in training with individual pattern-addressing input signals (Kolen and Pollack, 1991; Jordan, 1997; Krause et al., 2010).

4. Another possibility to train and select several temporal patterns in a single RNN is to associate different patterns with different initial network states. An example is Paine and Tani (2005) where an RNN-based mobile robot controller was trained to steer the robot into different action sequences depending on the state initialization.

5. In machine learning, RNNs trained on sequence prediction/generation tasks are typically considered as models of a stochastic process, for instance the process whose realizations are Wikepedia texts, as in Sutskever et al. (2011), rather than as cases of neural memory. But it also makes good sense to view them as neural memories. One may say that such a network after training has memorized typical continuations of initial cue sequences. Even more generally, any neural system trained on some training data set D for whatever purpose (classification, control, ...) could be regarded as having "stored" the probabilistic structure of D. In this general sense, any network-internal, trained representation can be interpreted as a LTM.

The diversity of these examples shows that there is no unique definition of what makes a neural LTM. Here is a list of system properties that are variously attributed (or not) to a neural memory system:

Multiplicity. An neural memory system typically can store more than only one memory pattern. The number of storable pattern is a standard performance criterion for memory systems.

Addressability. When several patterns have been stored, they should be efficiently addressable at use-time, whether by content-addressing or pointer-addressing.

Generativity. Stored patterns can be re-generated at retrieval time with some degree of accuracy.

Organization and searchability. Stored items are organized in a way that brings relationships between them to the fore, for instance, abstraction or part-of relationships (as in semantic network models in AI). This organization enables a systematic navigation and search in memory space. None of the above examples of dynamical pattern memories has this characteristic, but a diversity of neural learning systems for static patterns have been proposed that have an exploitable internal organization of some sort, for instance self-organizing semantic maps (Ritter and Kohonen, 1989), connectionist-localist networks for semantic concept modeling (Shastri, 1999), or recursive auto-associative memories (Pollack, 1990).

Incremental extensibility. New memory items can be stored incrementally on top of already stored ones. Such an incremental extensibility is difficult or impossible for networks trained with backpropagation due to the "catastrophic forgetting" problem (appreciation in Douglas and Sejnowski (2008), partial solutions and further references in French (2003); Grossberg (2005); Goodrich and Arel (2014)) and for most associative memories (exception: Jiang et al. (2016)).

Differential extension. When adding a new item, store only that information about the new item that is not already present in the memory system. None of the examples has this property but arguably human LTM has it.

Forgetting. Allow the memory system to fade out memory items that are not retrieved for a long time. This characteristic is typical for biological neural networks but is absent in all examples above.

Erasibility. Selectively delete arbitrary items from the memory, freeing storage space. This is typical for digital computer memories but is not usually considered in neural network models.

In this article I describe a neuro-computational mechanism, *conceptors*, by which the dynamics of an RNN can be governed in a variety of ways. Conceptors are a general-purpose mechanism that can be used in a diversity of neural information processing tasks. In a very long techreport (Jaeger, 2014) I present elementary demonstrations of conceptors used for temporal pattern classification, one-shot learning, human motion pattern generation, de-noising and signal separation. The present article focusses on exploits of conceptors for neural LTM management. From the above list of typical LTM properties, conceptors enable or facilitate all except the last two, forgetting and erasability. To keep the length of this article within reasonable bounds, I have omitted a sizable portion of LTM-relevant material from Jaeger (2014), namely the topic of Boolean operations on conceptors and the options that such Boolean operations grant for incremental and differential memory extensibility without catastrophic forgetting.

Most of the technical content is adopted from Jaeger (2014), but assembled in a more compact and intuitive exposition, and backed up by improved simulations. For space economy I omit proofs and some mathematical detail and refer the reader to the full treatment

given in Jaeger (2014). The Matlab code for all simulations is available at the JMLR online paper repository.

The article is organized in three main sections. Section 2 introduces the basic model in a detailed step-by-step fashion. Basic pattern morphing and pattern generalization demos, a "focussing" mechanism and a real-world data demo (learn from human motion capture data to animate a body model with 61 degrees of freedom) illustrate the basic functionalities of conceptors for storing, re-generating and modulating temporal patterns in an RNN. Section 3 introduces a conceptor-based model of content-addressable RNN memories for dynamical patterns. The stored patterns can be recovered from short and possibly distorted cues by a process of conceptor auto-adaptation. In several respects this provides a temporal analog of Hopfield networks.

2. Storing a Multitude of Temporal Patterns in an RNN

This section gives a concise introduction to the basic ideas and formalism of conceptors, starting from a toy example (Subsection 2.1) and then moving on to phenomena of pattern morphing, generalization, and "focussing" in the remaining subsections.

2.1 Basic Idea and Formalism

In this subsection I explain the basic ideas of conceptors by stepping through an elementary demonstration. The task is to store two dissimilar periodic temporal patterns in a tiny RNN and subsequently retrieve them.

2.1.1 Preparations

Before patterns can be stored and recalled, an experimental set-up is installed as follows.

Procuring training patterns. In this article a "pattern" means a discrete-time signal $p(n)$, where $p(n) \in \mathbb{R}^M$ and $n \geq 0$. For this elementary demo I use two discrete-time scalar patterns $p^1(n)$ and $p^2(n)$. The first is a sinewave signal with an irrational period length sampled at integer time points. The irrational period length makes the sampled version quasi-periodic. The second is a periodic signal with integer period 2. Figure 1 (left) depicts the two patterns. I remark that later I will also be considering non-periodic and non-stationary patterns.

Network creation. An RNN with N neurons is installed by randomly creating an $N \times N$ matrix W^* of internal connection weights, an $N \times M$ input weight vector W^{in}, and a random $N \times 1$ bias vector b. Here I use an unrealistically small network with only $N = 3$ neurons, which allows me to illustrate relevant effects in 3-D graphics.

This network can be driven by a scalar pattern $p(n)$ by the following state update equation:

$$\mathbf{x}(n+1) = \tanh(W^* \, \mathbf{x}(n) + W^{\text{in}} \, p(n) + \mathbf{b}). \tag{1}$$

All parameters of W^*, W and $\mathbf{b}$ are sampled from a normal distribution and scaled such that an overall system dynamics is obtained that works well for this didactic demonstration. Details of the scaling procedure are not important here, but I remark that the internal weights W^* must be scaled small enough to ensure that the RNN has the *echo state property* with respect to the input signals that are fed to it. In intuitive terms, an RNN has the echo

state property with respect to an input signal $p(n)$ if any initial network state is "forgotten" (washed out) when the network is driven by $p(n)$ for a long enough time. The echo state property is a core concept in the field of reservoir computing (Jaeger, 2001; Lukosevicius, 2012; Manjunath and Jaeger, 2013). Because the conceptor approach borrows some ideas from reservoir computing, I refer to the RNN as the *reservoir*.

Training a readout. The conceptor-based procedures explained below lead to a memory functionality in the following sense:

- At storing time, the reservoir is driven by a to-be-stored pattern $p^j(n)$ through (1). This results in a pattern-driven network dynamics $\mathbf{x}^j(n)$ (I use upper indices to refer to patterns).

- At recall time, the reservoir is run without input (under the control of a conceptor C^j, to be explained below), resulting in a free-running network dynamics $\tilde{\mathbf{x}}^j(n)$.

- The network has successfully "memorized" $p^j(n)$ to the degree that the free-running dynamics $\tilde{\mathbf{x}}^j(n)$ is approximately the same as the original pattern-driven dynamics $\mathbf{x}^j(n)$.

Thus, the network does not in fact memorize the original pattern $p^j(n)$, but instead its own, high-dimensional dynamical response to the pattern input. I will refer to such input-driven reservoir state dynamics as *state patterns*. But for practical exploits one wishes to recall the original input pattern, not the induced neural state pattern. In order to transform state patterns $\tilde{\mathbf{x}}^j(n)$ back to the original input patterns $p^j(n)$, a network state observer $y(n)$ is trained. This is a linear output neuron which reads from the network state through output weights W^{out}. It should display the following behavior

$$\begin{aligned} \textbf{if} \quad \mathbf{x}(n+1) \quad &= \tanh(W^* \mathbf{x}(n) + W^{\text{in}} p(n) + \mathbf{b}) \\ \textbf{then} \quad y(n) \quad &:= W^{\text{out}} \mathbf{x}(n) \approx p(n) \end{aligned} \tag{2}$$

for *any* driving pattern $p(n)$. That is, the output signal read through W^{out} should simply re-generate any driving input from the excited reservoir state.

The output weights W^{out} for such a generic input-redisplayer neuron can be trained by first driving the reservoir with an M-dimensional white-noise input signal $\nu(n)$ via $\mathbf{x}^\nu(n+1) = \tanh(W^* \mathbf{x}^\nu(n) + W^{\text{in}} \nu(n) + \mathbf{b})$, then compute W^{out} by linear regression to minimize the quadratic loss $\sum_n (W^{\text{out}} \mathbf{x}^\nu(n) - \nu(n))^2$, following the rationale of reservoir computing. Since this readout neuron is trained on white-noise input $\nu(n)$, it will also be able to re-generate other input signals $p(n)$ in agreement with (2).

I point out that this pattern-generic output neuron is trained prior to, and independent of, the subsequent memory learning. It is also possible to train the output weights W^{out} on network states induced by the to-be-stored input patterns, not by a generic white-noise input. That procedure gives more accurate results at recall time and therefore will usually be the preferred method. However, it obscures the essential independence of the storing procedure from the observer training. Therefore in this didactic demo I used white noise input to train W^{out}.

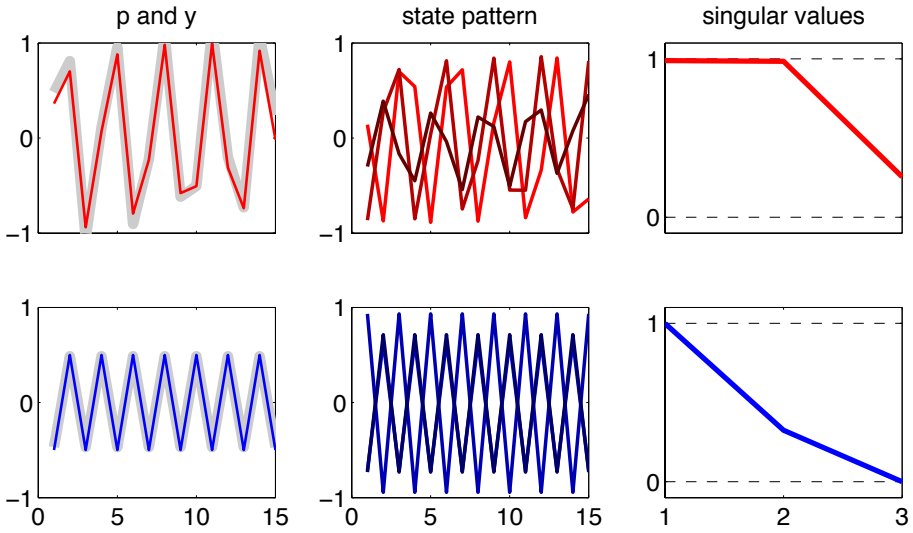

Figure 1: Left: the original patterns $p^1(n), p^2(n)$ (red/blue thin lines, 15 time steps are shown) and their recalled versions (gray thick lines; original and recalled patterns were phase-aligned for plotting). The patterns are discrete-time signals; connecting lines are drawn for better visual appearance. Center: traces of the three neurons' activations when the reservoir is driven with the respective pattern input. Right: singular value spectra of C^1 and C^2.

2.1.2 STORING PATTERNS

Loading the reservoir with the training patterns. I proceed to describe how the two patterns $p^1(n)$, $p^2(n)$ of our demo are stored in the reservoir. In intuitive terms, storing patterns p^j amounts to re-compute the initial random reservoir weights W^*, giving a new set of network weights W, such that the new reservoir can mimic the impact of drivers p^j in the absence of them. I refer to the new weights W as "input internalization weights" on the grounds that they incorporate the impact of an external input into the autonomous network update dynamics.

W is computed as follows. In two separate runs, the two patterns are fed into the initial reservoir via

$$\mathbf{x}^j(n+1) = \tanh(W^* \, \mathbf{x}^j(n) + W^{\text{in}} \, p^j(n) + \mathbf{b}), \quad j = 1, 2; \; n = 0, \ldots, L. \tag{3}$$

Figure 1 (middle panels) shows the activation traces of the three neurons when the reservoir is driven with either pattern.

Then W is computed to minimize the quadratic loss

$$\sum_{j=1,2} \sum_{n=n_0+1,\ldots,L} \| W^* \, \mathbf{x}^j(n) + W^{\text{in}} \, p^j(n) - W \, \mathbf{x}^j(n) \|^2, \tag{4}$$

where only network states for times after n_0 are used (in order to allow for washing out the arbitrary intial state $\mathbf{x}(0)$ according to the echo state property). This again amounts

to a linear regression. If this is achieved with a small training error, one will obtain similar network updates from states $\mathbf{x}^1(n)$ or $\mathbf{x}^2(n)$ with either the original input-driven update rule, or with an input-free update rule that employs W instead of W^*:

$$\tanh(W^* \mathbf{x}^j(n) + W^{\text{in}} p^j(n) + \mathbf{b}) \approx \tanh(W \mathbf{x}^j(n) + \mathbf{b}). \tag{5}$$

I call this procedure to transform the initial random weights W^* to W *loading* the patterns into the reservoir.

Linear regressions can be computed with various algorithms. In all simulations reported below I use ridge regression. Given a collection of L argument-target vector pairs $(\mathbf{a}_i, \mathbf{t}_i) \in \mathbb{R}^\nu \times \mathbb{R}^\mu$, ridge regression computes a transformation matrix $M \in \mathbb{R}^{\mu \times \nu}$ which minimizes the regularized mean square error $1/L \sum_i \|\mathbf{t}_i - M \mathbf{a}_i\|^2 + \|\varrho M\|^2_{\text{fro}}$, where $\| \cdot \|^2_{\text{fro}}$ is the squared Frobenius matrix norm and ϱ the Tychonov regularization coefficient.

In the case of (4), $\nu = \mu = N$ and the arguments are all $\mathbf{x}^j(n)$ and the targets are the corresponding $W^* \mathbf{x}^j(n) + W^{\text{in}} p^j(n)$.

The re-computation of initial weights of a RNN to obtain what I called here input internalization weights is a procedure that has been independently proposed several times in recent years under different names and for different purposes: as *self-predicting networks* for augmenting the performance of reservoir computing techniques (Mayer and Browne, 2004), as *equilibration* for enabling external controllability of RNN dynamics (Jaeger, 2010), as *reservoir regularization* for improved stability of neural motor controllers (Reinhart and Steil, 2011), as *self-sensing networks* for making RNN training methods more flexible (Sussillo and Abbott, 2012), and as *innate training* for reliable, noise-resistant reproducible chaotic patterns in short-term memory RNNs (Laje and Buonomano, 2013).

Lower cost variants of the loading procedure. In a variant of the loading procedure, only the input term $W^{\text{in}} p^j(n)$ is replaced, leaving W^* unchanged. That is, the input-driven network state $\tanh(W^* \mathbf{x}^j(n) + W^{\text{in}} p^j(n) + \mathbf{b})$ becomes emulated by the autonomous states $\tanh(W^* \mathbf{x}^j(n) + D \mathbf{x}^j(n) + \mathbf{b})$, where D is computed by linear regression to minimize the following variant of the quadratic loss (4):

$$\sum_{j=1,\ldots,K} \sum_{n=n_0+1,\ldots,L} \|W^{\text{in}} p^j(n) - D \mathbf{x}^j(n)\|^2. \tag{6}$$

I call the $N \times N$ matrix D *input simulation weights.* Another variant of the loading procedure is even more minimalistic and aims at replacing only the very input $p^j(n)$, by finding *input replacing weights* R (size $N \times M$) that minimize

$$\sum_{j=1,\ldots,K} \sum_{n=n_0,\ldots,L} \|p^j(n) - R \mathbf{x}^j(n)\|^2,$$

leading to autonomous states $\tanh(W^* \mathbf{x}^j(n) + W^{\text{in}} R \mathbf{x}^j(n) + \mathbf{b})$.

Computing conceptors. Considering (5), the loaded reservoir should be able to generate approximate versions of the two original state patterns. However, if the network were run just by iterating $\mathbf{x}(n+1) = \tanh(W \mathbf{x}(n) + \mathbf{b})$, the resulting input-free reservoir dynamics is entirely unpredictable because the reservoir can't "decide" which of the loaded pattern dynamics it should engage in.

Here conceptors enter the stage. Each loaded pattern p^j is associated with an $N \times N$ sized *conceptor* matrix C^j which at recall time is inserted into the state update loop via

$$\mathbf{x}(n+1) = C^j \ \tanh(W\,\mathbf{x}(n) + \mathbf{b}).$$

The matrix C^j acts as a filter that leaves states $\mathbf{x}^j(n)$ from the state pattern associated with pattern p^j essentially unchanged, but suppresses state components of states $\mathbf{x}^{j'}(n)$ associated with other patterns $p^{j'}$. Stated differently, C^j should act like the identity matrix for states $\mathbf{x}^j(n)$, but like the null matrix for state components that are not typical for states $\mathbf{x}^j(n)$.

This consideration leads to a quadratic loss function

$$L(C^j) = E[\|C^j\,\mathbf{x}^j(n) - \mathbf{x}^j(n)\|^2] + (\alpha^j)^{-2}\,\|C^j\|^2_{\mathrm{fro}}, \tag{7}$$

where the expectation E is taken over all states $\mathbf{x}^j(n)$ that arise in p^j-driven runs according to (3). Explanations:

- The first component $E[\|C^j\,\mathbf{x}^j(n) - \mathbf{x}^j(n)\|^2]$ of this loss function is minimal when C^j is the identity. This component reflects the objective that C^j should leave states $\mathbf{x}^j(n)$ unchanged.
- The second component $\|C^j\|^2_{\mathrm{fro}}$ becomes minimial for $C^j = \mathbf{0}$. This takes care of the objective that C^j should suppress state components which are untypical of states $\mathbf{x}^j(n)$, i.e. such state components which do not enter the expectation of the first component.
- The machine learning view on the loss (7) is to consider it as the loss for a *regularized identity function*. It is mathematically closely related to computing a *denoising autoencoder* for state patterns corrupted by Gaussian noise with variance $(\alpha^j)^{-2}$.
- The parameter $\alpha^j \geq 0$, called *aperture* for reasons that will soon become clear, negotiates between the two objectives. When the aperture is large, C^j will be close to the identity matrix I. Conversely, for small apertures C^j will shrink toward the null matrix $\mathbf{0}$. In our example I use $\alpha^1 = 12$ and $\alpha^2 = 20$.

Minimizing the loss $L(C^j)$ leads to the solution

$$C^j = R^j\,(R^j + (\alpha^j)^{-2}\,I)^{-1}, \tag{8}$$

where $R^j = E[\mathbf{x}^j(n)\mathbf{x}^j(n)']$ is the $N \times N$ correlation matrix of states $\mathbf{x}^j(n)$ obtained in the state dynamics driven by pattern p^j. C^j has the following properties:

- C^j is positive semi-definite, with eigenvalues (= singular values) $0 \leq \sigma^j_i \leq 1$ ($i = 1, \ldots, N$).
- The N eigenvectors $\mathbf{u}^j_i$ of C^j are the same as the eigenvectors of R^j, and can be arranged column-wise in an orthonormal matrix $U^j = (\mathbf{u}^j_1 \cdots \mathbf{u}^j_N)$. These eigenvectors are the same as the principal component vectors obtained from a principal component analysis of state sets $\mathbf{x}^j(n)$.

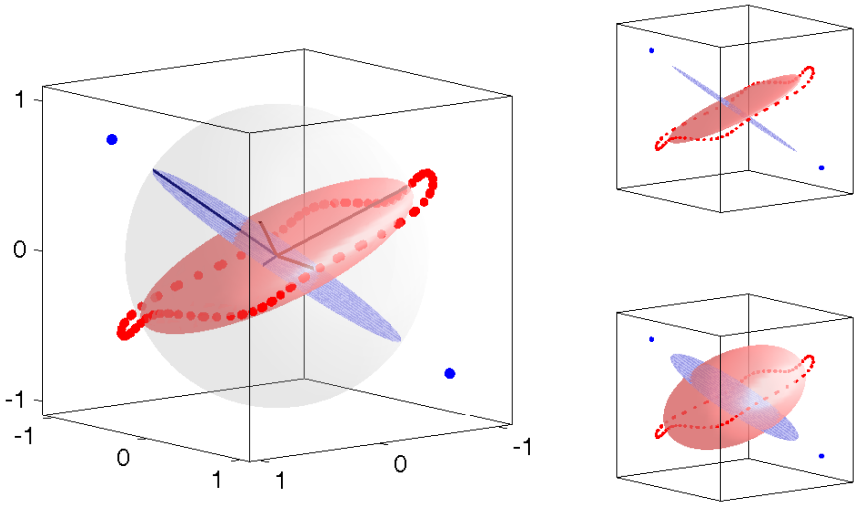

Figure 2: Conceptor geometry. Left: 3-dimensional state pattern $\mathbf{x}^1(n)$ (red dots) and $\mathbf{x}^2(n)$ (blue dots). Ellipsoids corresponding to conceptors C^1 and C^2 are rendered in red and blue respectively, with principal axes shown. They lie inside the unit sphere, shown in light gray. Right: effects of halving (top) and doubling (bottom) apertures.

I will often use the singular value decomposition (SVD) of $C^j = U^j\,S^j\,U^{j\prime}$, where S^j is the diagonal matrix containing the singular values σ_i^j on its diagonal, by convention arranged in descending order.

The derivation of (8) and the properties of C^j is elementary and can be found in Jaeger (2014). In practice, the correlation matrices R^j are estimated from states collected in training runs.

It is easily verified that $R\,(R+\alpha^{-2}I)^{-1} = \alpha^2 R\,(\alpha^2 R+I)^{-1}$. Because $\alpha^2 R = E[\alpha\mathbf{x}\,(\alpha\mathbf{x})']$, the aperture α can also be understood as a virtual scaling factor of reservoir states.

2.1.3 Geometry of Conceptors

It is instructive to contemplate the geometry of conceptors C^j and how it relates to the state dynamics $\mathbf{x}^j(n)$. The main panel in Figure 2 shows 100 instances of $\mathbf{x}^1(n)$ (red dots), which densely fill a cyclic path because of the irrational ratio between the driving sinewave period and the sampling interval. The resulting conceptor C^1 can be visualized by an ellipsoid centered at the origin. Its principal axes are the eigenvectors of C^1 scaled by their singular values (they are indicated in Figure 1, right panels). Since the singular values of conceptor matrices range in $[0, 1]$, such ellipsoids lie inside the unit sphere. The singular values of C^1 are all nonzero, hence the ellipsoid is non-degenerate and extends in three directions. The state pattern $\mathbf{x}^2(n)$ induced by the 2-periodic driver p^2 alternates between two states

(blue dots). These two states span a 2-dimensional subspace of $\mathbb{R}^3$: only two of the singular values of C^2 are nonzero, and the resulting ellipsoid is a degenerate (2-dimensional only).

The two small panels illustrate the geometrical effects of adjusting aperture. Increasing the aperture "widens" conceptors, while decreasing aperture lets conceptors contract.

2.1.4 Pattern Re-generation

The loaded reservoir will behave unpredictably when run via $\mathbf{x}(n+1) = \tanh(W\,\mathbf{x}(n) + \mathbf{b})$, but it will engage in the j-th state pattern $\mathbf{x}^j(n)$ when run with the corresponding conceptor in the loop via $\mathbf{x}(n+1) = C^j \tanh(W\,\mathbf{x}(n) + \mathbf{b})$. The conceptor C^j acts as a state filter which leaves states belonging to pattern j mostly unaffected, while other states that would belong to other patterns are projected into the ellipsoidal state volume typical for pattern j.

The original input pattern can be read from this dynamics by $W^{\text{out}}\,\mathbf{x}(n) \approx p^j(n)$. Figure 1 (left panels) shows the re-generated patterns under conceptor control overlaid with the original drivers.

2.1.5 Intuitive Summary, and Relationship to Reservoir Computing

Three kinds of matrices are trained in the conceptor-based pattern learning setup: the readout weights, the recurrent reservoir weights, and the conceptor matrices. How can the respective functional roles of these three components be intuitively understood?

In the loading process, when the random initial reservoir weights W^* are recomputed into W, information about the detailed dynamics of each training pattern is imprinted on the reservoir. A helpful metaphor is to liken the loading process to driving a vehicle over Sahara sands. Each pattern digs its individual track into the surface of the native sandscape. The original reservoir with weights W^* is like a virgin sandscape. After loading K patterns, K tracks have been engraved into the surface. These tracks are coded in the weight matrix W.

Still following that metaphor, the pattern tracks imprinted on the sand will have crossings. Sending a vehicle on a track-following mission, it wouldn't know where to turn at those crossings. If one would just run a network simulation with loaded weights W without further ado, the resulting network dynamics is unpredictable: the network doesn't know where to turn at the crossings; typically one observes a trajectory that appears like high-dimensional chaos or a trajectory that gets stuck in a point attractor.

This is where conceptors come into play. If conceptor C^j is inserted into the network update loop, it informs the network how to follow track number j. Figuratively speaking, the conceptor flattens out competing tracks but leaves track j mostly as it is. More technically speaking, this "flattening out" is effected by the large number of zero or close-to-zero singular values of C^j. In memory terminology, the conceptor acts as a *selector* for one of the loaded patterns. As we will see later on, as a side effect this "flattening out" endows the network dynamics with strong noise-suppression and dynamical stabilization characteristics.

Finally, the readout weights are not a crucial component in the conceptor approach to RNNs. This is different from the usual perspective of reservoir computing where the training of readout weights is the defining part of the entire paradigm. The focus of conceptor theory is *representation learning*, not output generation. It is all about creating representations

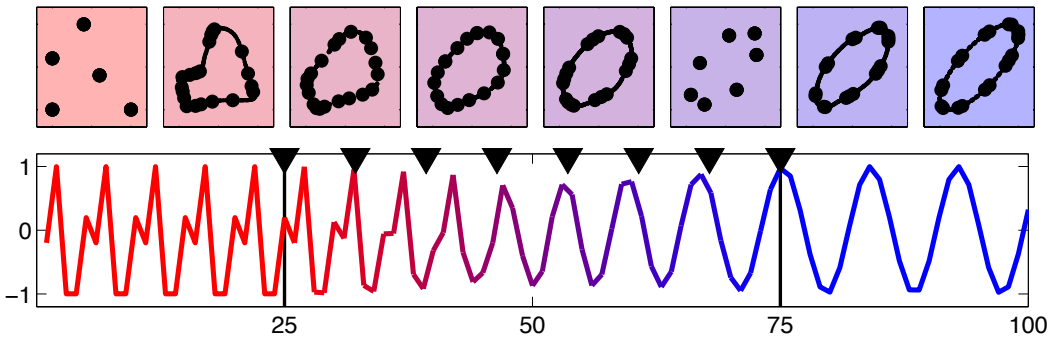

Figure 3: Morphing from an integer-5-periodic pattern p^1 to an irrational-period sine p^2 from time $n = 25$ to $n = 75$. Top: delay-embedding plots of signals $y(n)$ obtained with fixed conceptor mixes (indicated by triangle markers in center panel). Bottom: Output signal $y(n)$.

of dynamical patterns inside an RNN, where these representations (the tracks in the sand) become serviceable through conceptors. The readout weights merely provide an externally interpretable *observer* for the learnt representations, and a means to quantify the quality of the internal representations by measuring the mismatch between original patterns and re-generated output patterns.

2.2 Morphing and Generalization

When K "prototype" patterns $p^1, \ldots, p^K$ have been loaded, they can be *morphed* in recall by using linear blends of their conceptors via

$$\mathbf{x}(n+1) = (\sum_{j=1,\ldots,K} a^j\, C^j)\, \tanh(W\,\mathbf{x}(n) + \mathbf{b}), \quad y(n) = W^{\text{out}}\,\mathbf{x}(n), \tag{9}$$

where $\sum_{j=1,\ldots,K} a^j = 1$. For an elementary demonstration, an integer-5-periodic pattern p^1 and a sine pattern p^2 with an irrational period length of about 8.8 (sampled at integer time points) were loaded into a 100-neuron reservoir. Figure 3 (bottom) displays the output $y(n)$ of a recall run where for the first 25 steps, the pure conceptor C^1 was used, then for a morphing time of 50 steps C^1 was linearly blended into C^2, concluding the run with another 25 steps under the control of the pure C^2. The output signal exhibits a gradual blending from p^1 to p^2 during the morphing period from step 25 to step 75. In order to get more insight into the morphing dynamics, for eight mixing ratios (marked by triangle markers in the bottom panel), the system (9) was run separately with mixing coefficients frozen at these ratios. The panels in the top row show "fingerprints" of the resulting output signals $y_i(n)$, by plotting delay embedding state points $(y_i(n+1), y_i(n))$. Each panel shows 20 successive points as thick dots, and further 100 successive points as thin dots. The first panel, which corresponds to the pure 5-periodic pattern p^1, shows 5 cyclically repeated points, and the last panel a quasi-periodic oscillation, as expected. Intermediate dynamics are mostly quasi-periodic, but there are also mixture settings where there is a 6-, 7-, and

8-periodic behavior (of which only the 7-periodic instance happens to be captured by one of the displayed plots).

In a slightly more involved demonstration three representative patterns p^1, p^2, p^3 are loaded, which are taken from a two-parametric family of modulated sinewave patterns. One parameter changes the frequency, the other modulates the shape from "pointed upwards" to "pointed downwards" (details in the appendix). Figure 4 shows a close-up of the three loaded patterns in the separate top panels (red: slightly pointed downwards, high frequency; green: slightly pointed downwards, low frequency; blue: slightly pointed upwards, intermediate frequency). Note that three is the minimal number of patterns required to span a 2-parametric pattern space. To visualize the morphs, pattern samples generated with morphed conceptors $a^1\,C^1 + a^2\,C^2 + a^3\,C^3$ are arranged in a tesselated drawing plane. The "pure" recalled patterns (corresponding to mixtures $(a^1, a^2, a^3) = (1, 0, 0); (0, 1, 0); (0, 0, 1)$ respectively) are placed at the corners of an equilateral triangle (saturated red/green/blue tiles in the main figure panel). Tiles within the triangle spanned by the three loaded patterns correspond to interpolated conceptors, whereas patterns outside the triangle were obtained by *extrapolating* conceptor mixtures with coefficents outside [0 1].

By visual inspection of the signals shown in Figure 4 one finds that the system has acquired command over a sizeable portion of the parametrized pattern space by generalizing from three training instances. This generalization is effective not only in the "convex hull" of the three training examples (interpolation area within the central pattern triangle in the figure), but extends far into the extrapolation range. Going right/left from the three training patterns one sees that the mild frequency modulation indicated by the three training patterns reaches out into a wider frequency span; and going up/down one finds the "pointed up/down" characteristics, which is only feebly present in the training patterns, becomes fully expressed.

But one should not expect miracles. Upon closer inspection one will find that many regenerated patterns are not precise members of the family. Deviations grow as the mixture coefficients extend further away from the interpolation region. Also, there is no linear relationship between variations in mixing coefficients and geometric pattern characteristics. Far outside the interpolation region similiarities with correct patterns from the target family dissolve. If one wishes to obtain a wider-range faithful re-generation of patterns from the family, a larger number of reference patterns must be loaded. In Section 3.2 we will meet an altogether different way to employ conceptors to learn a pattern family from a small number of training examples.

2.3 A Real-World Data Example

In order to illustrate that the approach scales to more complex patterns, fifteen diverse human motion patterns were loaded: *slow walk*; *fast walk*; *walk with exaggerated stride*; *jog*; *sit*; *get up from stool*; *sit down on stool*; *kneel down*; *crawl*; *get up from crawl*; *waltz*; *cartwheeling*; and three *boxing punches*. These patterns are 61-dimensional signals (body pose and joint angles). Some are periodic (like *jog*), others are transient (like *get up from crawl*), yet others are irregular-stochastic (the *boxing* patterns); some patterns have multiple timescales. Training data consisted of short sequences clipped from publicly available human motion capture traces (CMU Graphics Lab)), one such clip per motion type. The length

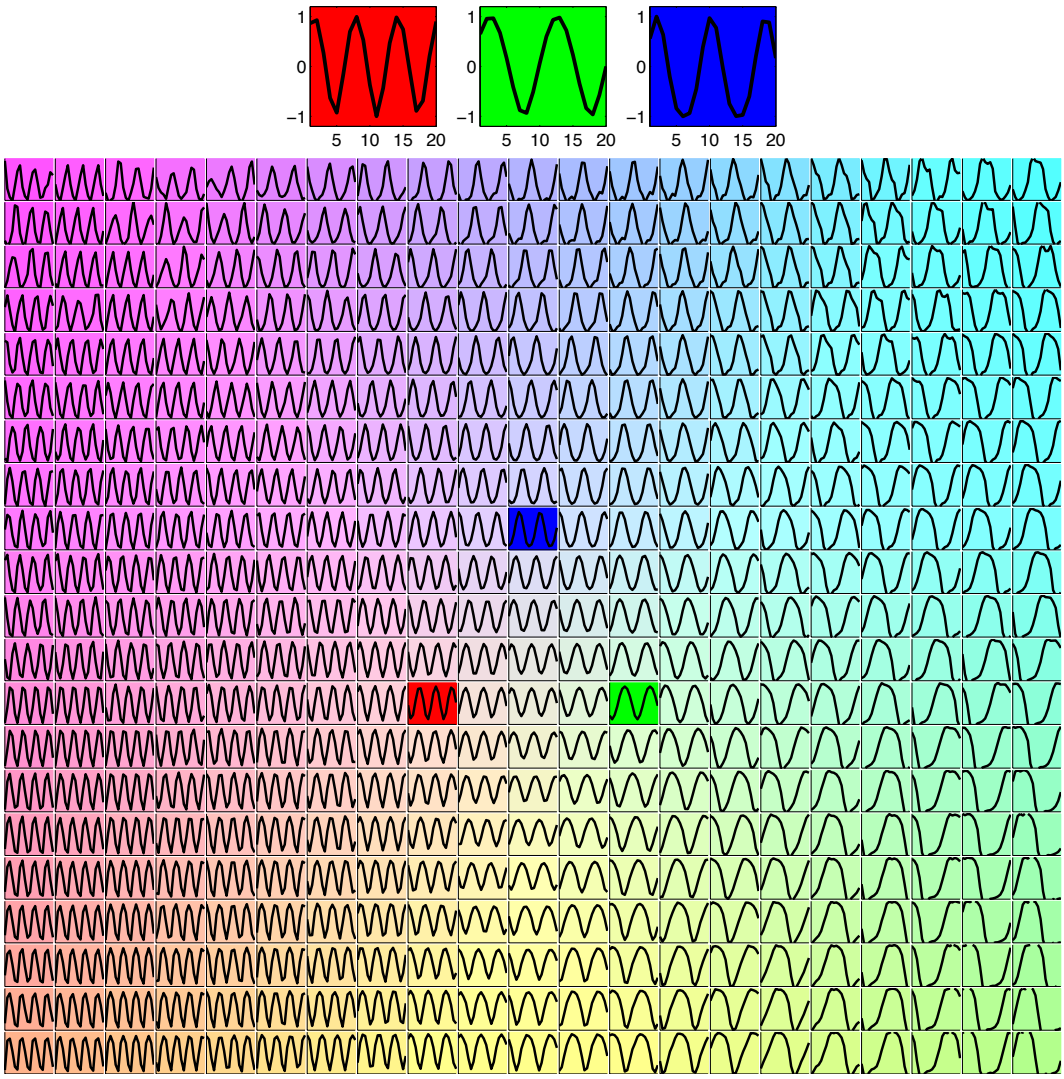

Figure 4: Morphing between and beyond three patterns from a 2-parametric family. For explanation see text.

of training patterns varied from 150 to 900 sampling points (sampling rate 120 / second). Data preprocessing and visualization was done with the help of a public Matlab motion capture toolbox (Burger and Toiviainen, 2013). Figure 5 gives an impression of the training data after preprocessing.

A 600-unit reservoir made from leaky-integrator neurons was loaded with the 15 training samples (details in the appendix). In a recall run, the 15 patterns were re-generated in a shuffled sequence according to a hand-designed choreography. In this sequence, a pattern p^j is reproduced by inserting the corresponding conceptor C^j into the reservoir update loop for some period of time. In order to achieve a smooth transition into the next motion $p^{j'}$, C^j was morphed into the following conceptor $C^{j'}$ for one simulated second before $C^{j'}$ takes over full

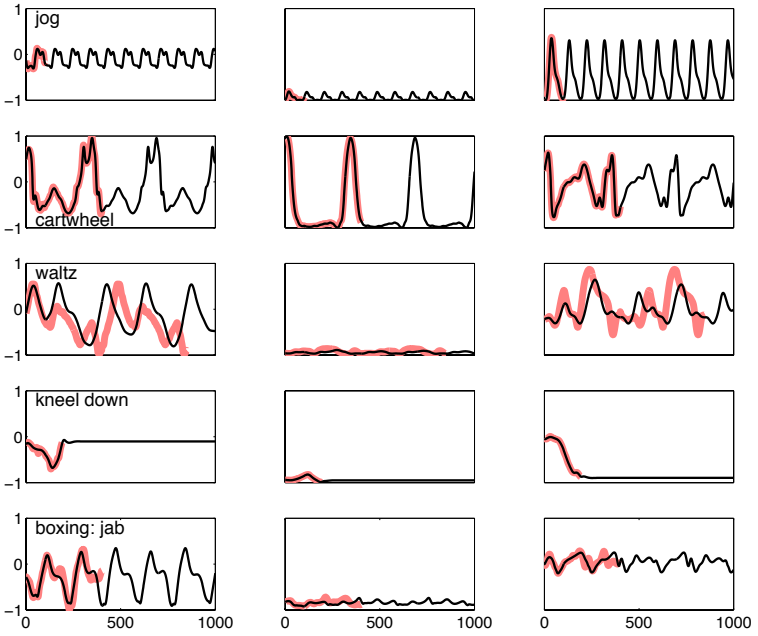

Figure 5: Exemplary human motion training data. Plots show the range-normalized versions used as input for the reservoir. The four rows show four out of the 15 loaded patterns. The three columns show three out of the 61 data dimensions. Thick red lines: training data; black lines: 1000-step free-running pattern recall under conceptor control. For explanation see text.

control. The resulting choreographed recall sequence of one minute duration was rendered into a video. The video is on YouTube (`www.youtube.com/watch?v=DkS_Yw1ldD4`). Figure 6 shows a few snapshots.

Morphing and generalizing dynamical patterns is a common but nontrivial task for training motor patterns in robots. It typically requires training demonstrations of numerous interpolating patterns (Reinhart and Steil, 2008; Coates et al., 2008; Lukic et al., 2012). Conceptor-based pattern learning and morphing appears promising for flexible robot motor pattern learning from a very small number of training patterns. I want to emphasize that this is only a method for "playback" of learnt trajectories with flexible sequence scheduling and motion blending with potential applications in computer game character animation; it is not a method for motor *control*. The procedure demonstrated here is visually superior to some existing neural-network based approaches (wyffels and Schrauwen, 2009; Sutskever et al., 2009; Boström et al., 2013; Wright and Jordanov, 2012) with respect to number of patterns, body model complexity, and smoothness of transitions. The mocap-trained human motion generation system presented by Taylor et al. (2011), which is based on restricted Boltzmann machines, comes close to the conceptor-based demo with respect to number of patterns and body model complexity, and has visually more appealing transitions between gaits.

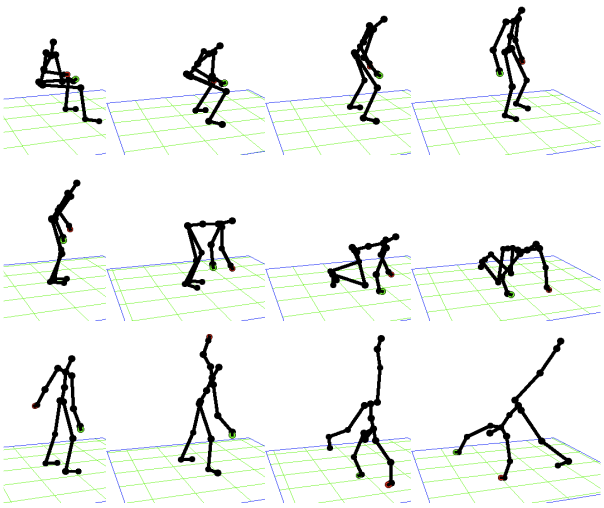

Figure 6: Snapshots from a human motion re-generation video created from a conceptor-controlled RNN. First row: standing up from a stool and starting a slow walk, second row: kneeling down and starting to crawl, third row: cartwheel.

I emphasize that this case study is solely meant as a visually appealing demo that conceptor-controlled pattern generation can scale to sizable numbers of high-dimensional, non-stationary patterns. In this it serves the same purpose as the other motion capture learning neural networks that I referenced above. It is neither a method for motor *control* in physical robots, nor does it aspire to compete with state-of-the-art game character animation engines, which each combine a diversity of sophisticated animation techniques, often including physics simulations (Gillies and Spanlang, 2010).

2.4 Aperture Adaptation

For good performance in recall, the aperture must be chosen appropriately. To illustrate this, I loaded four patterns into a $N = 500$ reservoir. The patterns were obtained from four classical chaotic attractors, p^1: Lorenz attractor; p^2: Rössler attractor; p^3: Mackey-Glass attractor; p^4: Hénon attractor (details in the appendix). Green plots in Fig. 7**A**,**B** visualize the clean training signals. In the reproduction stage, for each pattern p^j a number of different conceptors C^j with varied apertures were tried. Fig. 7**A** shows the effects for the Lorenz attractor. When α is too small, the reservoir-conceptor feedback loop becomes too constrained and the generated patterns de-differentiate. With α too large the feedback loop becomes over-excited. In this demonstration, a good aperture could be automatically found by minimizing a measurable that I call *attenuation*. This is the fraction of the reservoir signal energy which is suppressed by applying the conceptor. Formally, the attenuation $a_{C,\alpha}$ induced by a conceptor C at aperture α is

$$a_{C,\alpha} = E[\|\mathbf{r}(n) - \mathbf{z}(n)\|^2]/E[\|\mathbf{r}(n)\|^2],$$

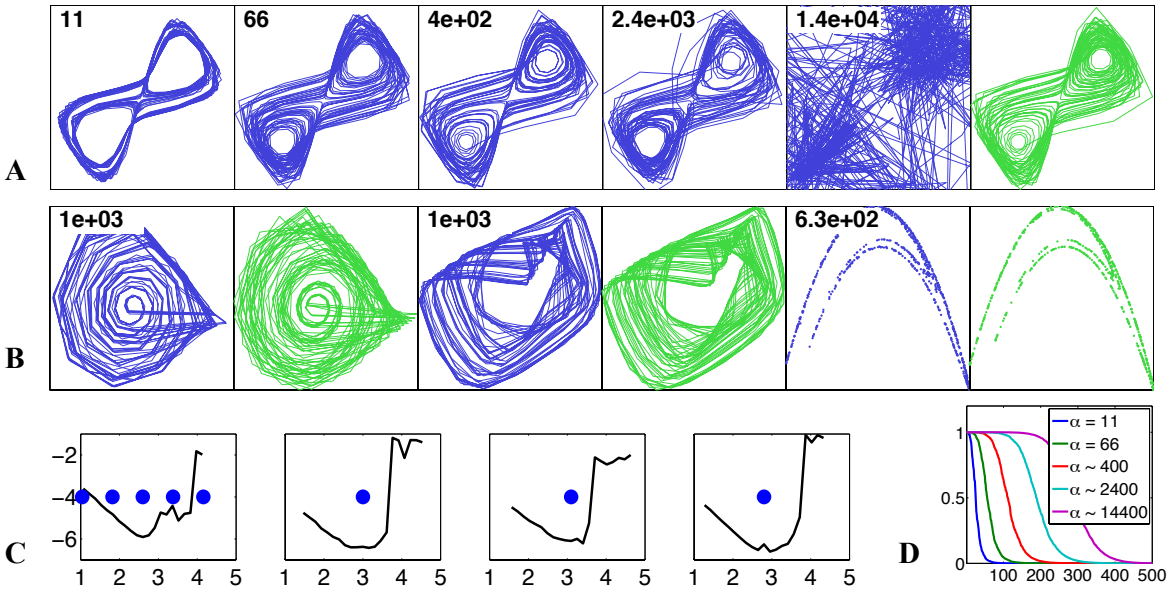

Figure 7: Reproducing four chaotic attractor patterns in a single network. Green: original training signals, blue: network-generated reproductions. The 2-dimensional plots are obtained from 1-dimensional timeseries by delay embedding, plotting $y(n)$ against $y(n-d)$ for a delay d chosen to yield instructive graphics. (**A**) The Lorenz attractor re-generated with five different aperture settings (α values inserted in panels). (**B**) From left to right: re-generations of the the Rössler, Mackey-Glass, and Hénon attractors. (**C**) Log10 of attenuation plotted against log10 of aperture. Dots mark the apertures used in the reproductions in (**A**,**B**). (**D**) Singular value spectra of conceptors used for the Lorenz attractor (compare panel **A**).

where $\mathbf{r}(n) = \tanh(W\mathbf{z}(n-1)+\mathbf{b})$, $\mathbf{z}(n) = C\,\mathbf{r}(n)$. This quantity can be cheaply estimated online and can be used to automatically adapt the aperture at recall time. Fig. 7**C** plots attenuation against aperture. The minimum of this curve marks a good aperture value, where "good" means "good visual agreement of recalled attractor with original attractor" (I did not implement an attractor comparison metric). The apertures used for the reproductions shown in Fig. 7**B** were obtained via this minimum-attenuation criterion.

While placing the aperture roughly in the right range is important, fine-tuning is unnecessary. The Lorenz attractor shown in Fig. 7**A** is quite well reproduced with apertures ranging over two orders of magnitude. This is also true for the other chaotic attractor signals, as well as for all other simulation studies that I undertook so far. While this 4-pattern demo is an academic exercise, it underlines the functionality of conceptors to stabilize neural pattern generation.

3. Autoconceptors

The definition and usage of conceptors, as described so far, invites an obvious critique: conceptor matrices have the same size as the reservoir weight matrix. Thus, for representing a pattern p^j by a conceptor matrix C^j one has to create and store an object that is as heavyweight as the hosting reservoir—one might say that each new pattern adds another brain. This is clearly inappropriate for computational neuroscience modeling, and it may become too expensive in machine learning applications. The storing cost of adding a conceptor matrix can be reduced by storing only economical SVDs of conceptor matrices, which often have low numerical rank. This however only mitigates but does not solve the principal problem.

I can offer two answers to this important critique:

Conceptor auto-adaptation and content-addressable memories. At loading time, when pattern internalization or pattern simulation weights are computed, no conceptors are created. Pattern recall functions by content-addressing: a short / incomplete cue of the target pattern is presented and an appropriate conceptor is generated on the fly by an auto-adaptation process. This leads to a memory model which is reminiscent of Hopfield networks. The present section is devoted to such auto-adapting conceptors.

Diagonal conceptors with single-neuron representations. When conceptor matrices are constrained to diagonal form, filtering a reservoir state by a conceptor reduces to unit-wise scalar multiplications, and a conceptor can be mathematically represented by a vector, which in turn could possibly be biologically implemented by a single neuron. This venue is explored in some detail in Jaeger (2014), where also proof-of-principle demos are presented of multi-reservoir hierarchical memory systems that are based on such diagonal conceptors. I am however not satisfied with the robustness properties of diagonal conceptors (they are too parameter-sensitive), and I am currently working on improved versions. Therefore I will not report on the available material from Jaeger (2014) this article.

3.1 Conceptor Auto-Adaptation

In the preceding sections I have defined conceptors as transforms $C^j = R^j\,(R^j + \alpha^{-2} I)^{-1}$ of reservoir state correlation matrices R^j (Equation 8) obtained from driving the reservoir with pattern p^j. The conceptor C^j could then be exploited for pattern re-generation via $\mathbf{x}(n+1) = C^j\ \tanh(W\mathbf{x}(n)+\mathbf{b})$ (using input internalization weights) or its variant $\mathbf{x}(n+1) = C^j\ \tanh(W^*\,\mathbf{x}(n) + D\,\mathbf{x}(n) + \mathbf{b})$ (using input simulation weights).

This way of using conceptors, however, requires that the conceptor matrices C^j are computed at loading time, and they have to be stored in some way for usage at exploitation time. Such a procedure may be useful in some practical engineering respects (for instance, for dynamical pattern morphing and pattern stabilization), but otherwise invites the critique pointed out at the beginning of this section.

This motivates to look for ways of how conceptors can be used for constraining reservoir dynamics without the necessity to create and store conceptor matrices beforehand. The network would have to create the requisite conceptors on the fly by some auto-adaptation process while it is performing some relevant task. In this section I investigate auto-adapted

conceptors for the task of realizing a content-addressable memory. They are also useful for other tasks—in Jaeger (2014) (Section 3.15) I demonstrate the use of such auto-adapted conceptors for the task of simultaneous signal denoising and classification.

For ease of discussion, I call conceptors created by auto-adaptation at use-time *autoconceptors*, as opposed to the conceptors which are created at pattern loading time and which are externally inserted into the reservoir update loop at use-time—I will refer to those as *alloconceptors*. Autoconceptors, like alloconceptors, are positive semidefinite matrices with singular values in the unit interval. Aperture operations are identical for allo- and autoconceptors. However, the way how autoconceptors are generated by auto-adaptation leads to additional constraints on their algebraic characteristics. The set of autoconceptor matrices is a proper subset of the set of alloconceptor matrices.

3.1.1 Basic Equations

The basic system equation for autoconceptor systems is

$$\mathbf{x}(n+1) = C(n)\ \tanh(W^*\,\mathbf{x}(n) + W^{\mathrm{in}} p(n+1) + \mathbf{b}) \tag{10}$$

or variants thereof, like

$$\mathbf{x}(n+1) = C(n)\ \tanh(W\,\mathbf{x}(n) + \mathbf{b}) \tag{11}$$

or

$$\mathbf{x}(n+1) = C(n)\ \tanh(W^*\mathbf{x}(n) + D\mathbf{x}(n) + \mathbf{b}), \tag{12}$$

the latter two capturing the situation after having patterns loaded by input internalization or input simulation weights. The novel element in these equations is that $C(n)$ is time-dependent. Its evolution is governed by an adaptation rule that I will describe presently. $C(n)$ need not be positive semidefinite at all times; only after convergence the $C(n)$ matrices will have the algebraic properties of conceptors.

It is convenient to formally split the network state $\mathbf{x}$ in (10) into a "reservoir state" $\mathbf{r}$ measured directly after the neuron nonlinearity, and a "filtered" state $\mathbf{z}$ obtained after passing the reservoir state through the conceptor. This turns (10) into

$$\begin{aligned} \mathbf{r}(n+1) &= \tanh(W^*\mathbf{z}(n) + W^{\mathrm{in}} p(n+1) + \mathbf{b}) && (13)\\ \mathbf{z}(n+1) &= C(n)\,\mathbf{r}(n+1). && (14) \end{aligned}$$

The auto-adaptation of $C(n)$ aims at minimizing a loss that at first sight looks the same as the loss (7):

$$L(C) = E[\|C\,\mathbf{z}(n) - \mathbf{z}(n)\|^2] + \alpha^{-2}\,\|C\|^2_{\mathrm{fro}}, \tag{15}$$

which for alloconceptors was solved by $C = R\,(R + \alpha^{-2} I)^{-1}$ with $R = E[\mathbf{x}\,\mathbf{x}']$ being the autocorrelation matrix of reservoir states. Similarly, (15) is minimized by

$$C = R\,(R + \alpha^{-2} I)^{-1},\ \text{with } R = E[\mathbf{z}\,\mathbf{z}']. \tag{16}$$

The crucial difference is that now the state correlation matrix R depends on C:

$$R = E[\mathbf{z}\,\mathbf{z}'] = E[C\mathbf{r}\,(C\mathbf{r})'] = C\,E[\mathbf{r}\,\mathbf{r}']\,C =: CQC,$$

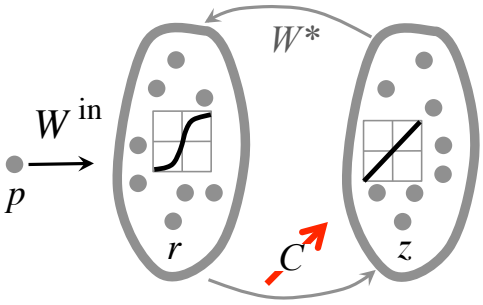

Figure 8: Network representation of a basic autoconceptor system. Bias b and readout mechanisms are omitted. The red arrow indicates that C is adapted online. For explanation see text.

where we introduce $Q = E[\mathbf{r}\,\mathbf{r}']$. This transforms the explicit solution formula (8) into a fixed-point equation:

$$C = CQC\,(CQC + \alpha^{-2}\,I)^{-1}. \tag{17}$$

Since Q depends on $\mathbf{r}$ states, which in turn depend on $\mathbf{z}$ states, which in turn depend on C again, Q depends on C and should be more appropriately be written as Q_C. A formal analysis of solutions to the fixed-point equation $C = CQ_CC\,(CQ_CC + \alpha^{-2}\,I)^{-1}$ is involved (carried out in some detail in Jaeger (2014)). Explicit solution formulas are unlikely to exist because the nonlinear reservoir is involved in Q_C. When one uses autoconceptors, however, one does not need to solve (17) explicitly. Instead, one can invoke stochastic gradient descent to minimize the loss (15). It is easily derived (Jaeger, 2014) that

$$C(n+1) = C(n) + \lambda\,\left((\mathbf{z}(n) - C(n)\,\mathbf{z}(n))\,\mathbf{z}'(n) - \alpha^{-2}\,C(n)\right) \tag{18}$$

implements stochastic gradient descent with respect to the loss (15). Here λ is an adaptation rate. Equations (13), (14) and (18) taken together—or versions where (13) is replaced by (11) or (12)—define the joint state update and conceptor adaptation working cycle.

In Jaeger (2014) I show that, if the driver $p(n)$ is a stationary process and if $C(n)$ converges under this auto-adaptation rule, the limit C is positive semidefinite with singular values in the set $(1/2, 1) \cup \{0\}$. Singular values of C asymptotically obtained under the evolution (18) are either greater than $1/2$ or they are zero. If the aperture α is fixed at increasingly smaller values, increasingly many singular values converge to zero. Furthermore, the analysis in Jaeger (2014) reveals that among the nonzero singular values, the majority will be close to 1. Both effects together endow autoconceptors with singular value spectra that are typically approximately rectangular.

3.1.2 Basic Demonstrations

In order to display how autoconcepters can be used to realize a content-addressable memory, I ran simulations according to the following scheme:

1. *Loading.* A collection of 10 scalar patterns p^j was loaded in an N-dimensional reservoir, yielding an input simulation matrix D and readout weights W^{out}. More specifically, D was computed to minimize the loss (6), using states $\mathbf{x}^j(n)$ obtained from driven runs $\mathbf{x}^j(n+1) = \tanh(W^* \mathbf{x}^j(n) + W^{\text{in}} p^j(n) + \mathbf{b})$. No conceptors were involved or computed in the loading procedure.

2. *Recall.* For each pattern p^j, a recall run was executed which consisted of three stages:

   (a) *Initial washout.* Starting from a zero network state, the reservoir was driven with p^j for n_{washout} steps, in order to obtain a task-related reservoir state.

   (b) *Cueing.* The reservoir was continued to be driven with p^j for another n_{cue} steps. During this cueing period, C^j was adapted through $\mathbf{z}(n+1) = \tanh(W^* \mathbf{z}(n) + W^{\text{in}} p(n) + \mathbf{b})$, $C^j(n+1) = C^j(n) + \lambda^{\text{cue}} ((\mathbf{z}(n) - C^j(n)\,\mathbf{z}(n))\,\mathbf{z}'(n) - \alpha^{-2}\, C^j(n))$. At the beginning of this cueing period, $C^j(0)$ was initialized to the zero matrix. Notice that during cueing, the nascent conceptor $C^j(n)$ was not inserted in the reservoir state update loop. At the end of this period, a conceptor $C^{j\,\text{cue}}$ was obtained.

   (c) *Autonomous auto-adaptation and pattern recall.* The network run was continued with the external input switched off, using $\mathbf{z}(n+1) = C^j(n) \tanh(W^* \mathbf{z}(n) + D\mathbf{z}(n) + \mathbf{b})$, while continuing to auto-adapt the conceptor through $C^j(n+1) = C^j(n) + \lambda^{\text{recall}} ((\mathbf{z}(n) - C^j(n)\,\mathbf{z}(n))\,\mathbf{z}'(n) - \alpha^{-2}\, C^j(n))$. At three test time points t_1, t_2, t_3 the conceptor $C^j(t_i)$ in its current adaptation stage was recorded for an offline quality assessment.

3. *Measuring the quality of conceptors.* The quality of the conceptors $C^{j\,\text{cue}}$ and $C^j(t_i)$ (where $i = 1, 2, 3$) was measured in separate offline runs without conceptor adaptation using $\mathbf{z}(n+1) = C \tanh(W^* \mathbf{z}(n) + D\mathbf{z}(n) + \mathbf{b})$, where C was one of $C^{j\,\text{cue}}, C^j(t_1), C^j(t_2), C^j(t_3)$. A reconstructed pattern $y(n) = W^{\text{out}}\,\mathbf{z}(n)$ was obtained and its similarity with the original pattern p^j was quantified in terms of a NRMSE.

I carried out two instances of this experiment, using two kinds of patterns:

5-periodic pattern. The patterns were random integer-periodic patterns of period 5. Experiment parameters: Reservoir size $N = 100$, spectral radius of W^* set to 1.5, input weight scaling 1.5, bias weight scaling 0.5, ridge regularization coefficients $\varrho^2_D, \varrho^2_{W^{\text{out}}}$ both set to 0.0001, aperture $\alpha = 1000$, $n_{\text{washout}} = 20$, $n_{\text{cue}} = 10$, $t_1, t_2, t_3 = 10, 50, 500$, auto-adaptation rates $\gamma^{\text{cue}} = 0.02$, $\gamma^{\text{recall}} = 0.01$. To make the task more challenging, during the 10-step cueing phase the cue input pattern was corrupted by additive uniform noise sampled from $[-0.05, 0.05]$, and during the autonomous recall time the reservoir states were *very strongly* perturbed by additive Gaussian noise (fed inside the tanh) whose variance equalled the average reservoir state component variance (a signal-to-noise ratio of 1).

2-parametric mix of 2 irrational-period sines. The 10 patterns were were taken from the 2-parametric family of patterns governed by

$$p(n) = a\, \sin(2\,\pi\, n/P) + (1-a)\, \sin(4\,\pi\,(b + n/P)).$$

These signals are weighted sums of two sines, the first with period length P and the second with period length $P/2$. The weights of these two components are a and $(1-a)$, and the second component is phase-shifted relative to the first by a fraction b of its period length $P/2$. The reference period length P was fixed to $P = \sqrt{30}$. The parameters a, b were freshly sampled from the uniform distribution on $[0, 1]$ for each p^j. Network parameters: $N = 100$, reservoir spectral radius 1.5, input weight scaling 1.5, bias scaling 0.5, $\varrho_D^2 = \varrho_{W^{\text{out}}}^2 = 0.0001$, $\alpha = 200, n_{\text{washout}} = 100, n_{\text{cue}} = 12$, $t_1, t_2, t_3 = 20, 1000, 10000$, $\gamma^{\text{cue}} = \gamma^{\text{recall}} = 0.01$. Cue patterns were corrupted by the same amount of additive noise as the 5-periodic patterns, and reservoir states were again perturbed with a signal-to-noise ratio of 1 during the autonomous recall.

Figure 9 illustrates the outcomes of these two experiments. For brevity I refer to the two experiments as the "IP5" and "PF" conditions. Observations and interpretations:

- The quality of the preliminary conceptor $C^{j\,\text{cue}}$ was always much improved by the subsequent auto-adaptation (panels **B**, **D**).

- The effects of autoconceptive adaptation are reflected in the singular value profiles of $C^{\text{j}\,\text{cue}}$ versus $C^j(t_3)$ (**A**, **C**). During the short cueing time, the online adaptation of the conceptor from a zero matrix to $C^{j\,\text{cue}}$ only manages to build a preliminary "nascent" profile, which then "matures" by auto-adaptation.

- The conceptors $C^j(t_3)$ have an almost rectangular singular value profile, in agreement with mathematical analyses (Jaeger, 2014).

- In PF the averge re-generation NRMSE with the final conceptor $C(t_3)$ was 0.11. For comparison I loaded and recalled the 10 patterns following the basic alloconceptor procedure with a manually optimized aperture. This gave an NRMSE of 0.087, that is, content-addressed autoconceptor recall from short and noisy cues here worked almost as well as re-generation with previously stored conceptors.

- For the 5-periodic patterns, a comparison simulation with the basic alloconceptor procedure yielded a mean NRMSE of .0033, much better than the mean NRMSE of 0.070 obtained after $t_3 = 500$ steps in the IP5 experiment. When perturbations to the cue signal and the reservoir states were switched off (not shown), the final NRMSE improved to 0.0037, almost as good as in the alloconceptor comparison. In summary, with cue and state noise the IP5 experiment worked out satisfactorily, and in a noise-free version essentially as well as with alloconceptors.

- In the FP experiment, unlike in IP5, state noise not only did not harm, but was conducive for fast adaptation. As a comparison I repeated the simulation with noiseless cues and without state noise insertion during autoconception (not shown). The outcome: in some cases the auto-adaptation failed altogether, and when it did function, it needed much longer runtimes (1,000,000 steps) to reach reconstruction qualities similar to what is reported in Figure 9 **C** **D**. State noise in auto-adaptation helps to drive the singular values of relevant state components toward 1 and considerably speeds up the adaptation.

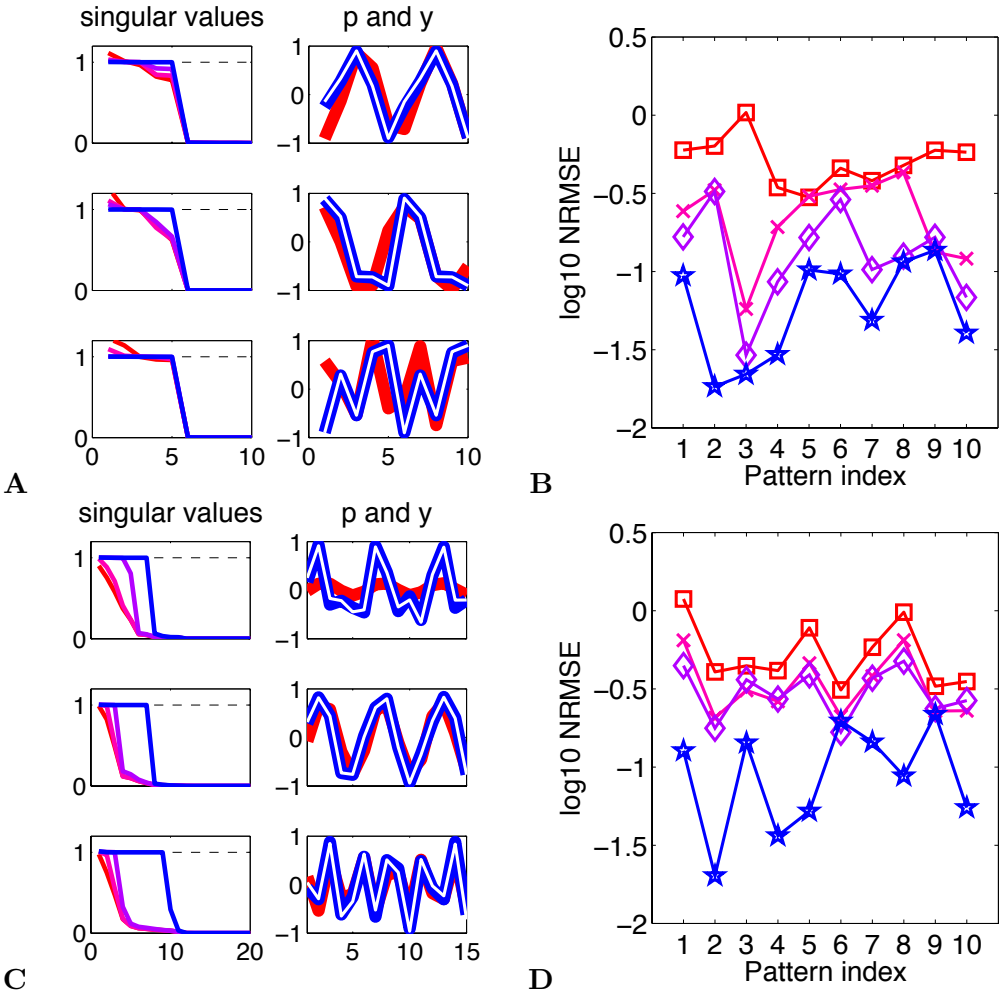

Figure 9: Basic content-addressable memory demos. **A**, **B**: 5-periodic pattern, **C**, **D**: mix of sines pattern. Panels **A**, **C** show the first three patterns, giving the first few singular values of the conceptors C^{cue}, $C(t_1)$, $C(t_2)$, $C(t_3)$. The "p and y" panels show the original patterns (white), the pattern re-generated with C^{cue} (thick red) and with the final $C(t_3)$ (thick blue) after optimal phase alignment with the original. **B**, **D**: recall log10 NRMSEs obtained from $C^{\text{cue}}, C(t_1), C(t_2), C(t_3)$.

- The robustness of auto-adaptation against very strong state noise may seem surprising at first, but is easily explained. Autoconceptor adaptation leads to singular value spectra with many zero values (Jaeger, 2014). Reservoir state noise components in directions of the nulled eigenvectors are entirely suppressed in the conceptor-reservoir loop, and state noise components within the nonzero conceptor eigenspace do not impede the development of a "clean" rectangular profile but in fact, as seen in FP, may even speed up the auto-adaptation process.

- In the simulations reported above, zero singular values were present from the start because the conceptor was initialized as the zero matrix. If it had been initialized differently (for instance, as the identity matrix), the auto-adaptation would only asymptotically pull (the majority of) singular values to zero, with noise robustness only gradually emerging to the degree that many singular values decrease toward zero. If noise robustness is desired, it can be reached by additional adaptation mechanisms for C. In particular, it is helpful to include thresholding: all singular values of $C(n)$ exceeding a suitable threshold are set to 1, all singular values dropping below a certain cutoff are zeroed (not shown).

- The simulations reported here were done with unfavorable settings on purpose, using short cueing times and strong noise. With larger reservoir networks, longer cueing times, less noise etc. much higher recall accuracies can be obtained (not shown).

An inquisitive reader may ask why I don't install auto-adaptation with respect to the reservoir output $\mathbf{r}$ instead of $\mathbf{z}$, that is, why not use the loss

$$L(C) = E[\|C\,\mathbf{r}(n) - \mathbf{r}(n)\|^2] + \alpha^{-2}\,\|C\|^2_{\mathrm{fro}}, \tag{19}$$

instead of the loss (15). This alternative loss (19) appears as natural as the loss (15) (and its solutions would be much easier to analyze mathematically). But in simulation experiments (not reported) I found that the performance of auto-adaptation based on (19) was far inferior to the version used above. Specifically, (i) all noise robustness was lost, (ii) during auto-adaptation the conceptors grew very heavy tails in their singular value spectra even for 5-periodic patterns where one would desire a spectrum with exactly 5 nonzero singular values, and (iii) singular values greater than 1 emerged under auto-adaptation, likewise undesirable within the conceptor philosophy.

The conceptor auto-adaptation dynamics in content-addressable memories is quite involved, with intriguing analogies and differences to Hopfield network adaptation dynamics. I discuss this in Section 3.3.

3.2 From Rote Learning to "Understanding"

When the cue represents a pattern that was not loaded, autoconceptors can be invoked to re-generate even such patterns. Specifically, if the reservoir was loaded with a few patterns $p^1, \ldots, p^k$ from a parametric family and then is cued with another pattern p^* from that family, an attempt can be made to re-generate this new pattern, too. This will work out provided that (i) enough patterns $p^1, \ldots, p^k$ had been loaded to instruct the reservoir about the characteristics of the pattern family, and (ii) the reservoir size is large enough to code the characteristics of the pattern family. For a demonstration I chose a 100-neuron reservoir and the mix-of-sines patterns from the 2-parametric family used before. A cue time of 12 steps and an auto-adaption time of 10,000 was used. The simulation was carried out as follows (detail in the appendix):

1. Create a random reservoir.

2. In separate trials, load this reservoir with an increasing number k of patterns (ranging from $k = 1$ to $k = 1000$), randomly taken from the parametric family.

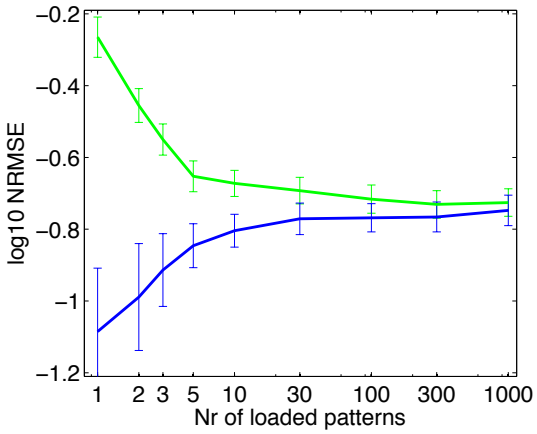

Figure 10: Class learning effect. Error bars indicate 95 % confidence intervals for mean NRMSEs. The curves are means over the 10 recall targets and the 10 experiment repetitions (blue: recall of patterns that were loaded; green: recall of patterns that were not loaded). Both axes are in logarithmic scale. For explanation see text.

3. After loading, cue (with noiseless cues) and re-generate the loaded patterns by auto-conceptors. Measure the final recall accuracy for the first 10 of the loaded patterns (when fewer than 10 were loaded, do it only for these).

4. In addition, per trial, also try to cue and "recall" 10 novel patterns that were drawn randomly from the family. Monitor the "recall" accuracy of these novel patterns as well.

5. Repeat this entire scheme 10 times, with freshly sampled reservoirs and patterns. Report averaged diagnostics.

Figure 10 shows the results. Observations:

- The recall NRMSE for patterns that have been expressedly loaded is lowest when only few patterns have been loaded. As the number of loaded patterns grows, the NRMSE first rises, then levels out for very large numbers of loaded patterns.

- The recall NRMSE for non-loaded patterns is high in conditions when only few patterns have been loaded. As the number of loaded patterns grows, this NRMSE decreases and levels out at about the same goodness as the NRMSE for loaded patterns.

Similar findings were obtained in other simulations (Jaeger, 2014). Pending a formal analysis, the following interpretation suggests itself:

- For small numbers of loaded patterns the system stores and recalls individual patterns. The input simulation matrix D represents **individual** patterns.

- For large numbers of loaded patterns, the system learns a representation of the parametrized pattern family and can re-generate any pattern from that family from a cue. The input simulation matrix D represents the **class** of patterns.

In suggestive wording, as more and more patterns are loaded this memory system changes its nature from "rote learning" to "understanding" the nature of the pattern family. With the aid of autoconceptors, any member of the family can be "recognized" from a cue and become re-generated. This family learning effect becomes effective already when only a single pattern has been loaded, and when as few as about 5 patterns are loaded, the recall log10 NRMSE for novel patterns is within 20% of the accuracy for loaded patterns. This is another display of the generalization powers of conceptor-based systems, which we encountered before in connection with conceptor morphing (Section 2.2).

3.3 Analysis of Auto-Adaptation Dynamics

The dynamics of autoadaptation—that is, the evolution of $C(n)$ under the simultaneous reservoir state update (13), (14) (or variants) and conceptor matrix adaptation rule (18)—is highly nonlinear and involves at least the two timescales of reservoir updates (fast, N variables) and C adaptation (slow, N^2 variables). A full analysis is out of reach, but some core aspects are analytically accessible. In Jaeger (2014) I investigate properties of fixed points of C auto-adaptation. Here I review the most important findings and attempt an interpretation of some phenomena found in simulation studies in the light of these analytical insights. This subsection assumes some familiarity with methods from dynamical systems theory. Readers not interested in technical detail may jump to the intuitive summary at the end of this subsection.

My analysis was based on assuming an adaptation rate λ that is small enough to admit temporal averaging over the fast $\mathbf{z}(n)$ state variables. The discrete update equation (18) can then be transformed into a continuous-time ODE which is easier to analyze. This ODE is

$$\tau \dot{C} = (I - C)\, C\, Q_C\, C' - \alpha^{-2}\, C, \tag{20}$$

where $Q_C = E_{\mathbf{z}}[\tanh(W\,\mathbf{z} + \mathbf{b})\tanh(W\,\mathbf{z} + \mathbf{b})'] = E_{\mathbf{r}}[\mathbf{r}\,\mathbf{r}']$ is the autocorrelation matrix of reservoir states (here in a version with input internalization weights, versions with input simulation weights are analog) when these states evolve under the control of C. The time constant τ is of no concern for the qualitative analysis. Specifically I investigated two objects:

Fixed points F of (20). These are candidates for conceptors that are asymptotically obtained for convergent autoadaptation runs.

Jacobians J_F of fixed points F. J_F is an $N^2 \times N^2$ matrix whose (i,j)-th element is the derivative $\partial \dot{F}_i / \partial F_j$, under the dynamics (20) linearized in F, and where F_i is the i-th element of F (enumerated row-wise). The eigenvalues of J_F characterize stability properties of F.

Here is a summary of insights from this analysis:

- Fixed points F are positive-semidefinite matrices with eigenvalues (= singular values) ranging in $[0, 1]$, and thus qualify as conceptors.

- When F has singular values that are smaller than $1/2$ and greater than 0, J_F has positive eigenvalues and hence F is not a stable fixed point. Such F cannot be obtained in empirical auto-adaptation runs. Conversely, this means that if auto-adaptation converges, the resulting conceptors C have singular values that are either exactly zero or larger than $1/2$.

- The rank of F is bounded from above by the number of singular values of Q_F which are greater than $4\alpha^{-2}$. This implies that reducing the aperture α forces an increasing number of singular values of F to become zero.

- Assume F has rank k. Then J_F has $k(N-k)$ zero eigenvalues, that is, a center manifold $\mathcal{K}$ of dimension $k(N-k)$. On this center manifold the stability properties of the dynamics (20) cannot be elucidated by linearization techniques; it may be stable, instable, or neutral.

- If F is a rank-k solution of the fixed-point equation (17) which is generic in the sense that the nonzero singular values of F are pairwise different, and if Q in (17) is fixed (that is, its dependency on C is ignored), then differential changes of F along a $k(N-k)$-dimensional subspace will also lead to rank-k solutions of (17) (Jaeger (2014), Section 3.13.4). The identity of the center manifold dimension and the dimension of this local solution space implies that the center manifold coincides with the local solution space. Hence the adaptation dynamics on the center manifold $\mathcal{K}$ is neutrally stable: $\mathcal{K}$ is a plane attractor. Note however that this interpretation is based on fixing Q at Q_F in the vicinity of F. Plane attractors are non-generic in randomly parametrized ODEs, and fixed point solutions F are generically isolated (not embedded in a plane attractor). When the dependence of Q on C is included in the picture, and F is a fixed-point solution of (17), we would generically expect a separation of timescales in the autoadaptation dynamics in the vicinity of F: fast convergence toward F in directions orthogonal to $\mathcal{K}$ and (very) slow drift within $\mathcal{K}$. Below I present a numerical simulation that corroborates this intuition.

- In order to obtain a better understanding of convergence properties in directions orthogonal to $\mathcal{K}$ one needs to analyse the negative eigenvalues of the Jacobian J_F. Assume again that F has rank k, and furthermore the k nonzero singular values $s_1, \ldots, s_k$ of F are greater than $1/2$. Then the $N^2 - k(N-k)$ nonzero eigenvalues of J_F are all negative. They come in three sorts (Proposition 16 in Jaeger (2014)):

  1. $N(N-k)$ eigenvalues are equal to $-\alpha^{-2}$ ("sort 1"),
  2. k eigenvalues are associated with the k singular values s_i and have the form $-\alpha^{-2}\,(2s_i - 1)/(1 - s_i)$ ("sort 2"),
  3. the remaining $k(k-1)$ eigenvalues ("sort 3") come in pairs

$$\lambda_{1,2} = \frac{-\alpha^{-2}}{2}\left(\frac{s_l}{1-s_m} + \frac{s_m}{1-s_l} \pm \sqrt{\left(\frac{s_l}{1-s_m} - \frac{s_m}{1-s_l}\right)^2 + 4}\right),$$

     where $m < l \leq k$.

When $k \ll N$, which is often the case, the majority of all eigenvalues of F is of sort 1, revealing that in the majority of directions orthogonal to the center manifold the convergence rate toward F is inversely proportional to α^2. A relevant message here is that increasing the aperture substantially slows down convergence. An eigenvalue of sort 2 is also inversely proportional to α^2, but furthermore the term $(2s_i - 1)/(1 - s_i)$ indicates that as $s_i \downarrow 1/2$, the convergence rate in this direction slows down arbitrarily much, while conversely for $s_i \uparrow 1$, convergence becomes arbitrarily fast. Finally, eigenvalues of sort 3 are products of two factors too, the first factor being again inversely proportional to α^2. The second factor, which depends on a pair s_l, s_m, may become arbitrarily large or arbitrarily small depending on s_l, s_m and the $\pm$ choice. Specifically, this second factor approaches zero if both s_l, s_m approach $1/2$ from above and the "$-$" option is taken. In summary, outside the center manifold convergence rates are always co-determined by a factor proportional to α^2, and by factors that depend on the singular values s_i of F, where these latter factors may shrink toward zero if associated singular values s_i approach $1/2$.

- As we have just seen, singular values s_i of F which are close to $1/2$ lead to slow convergence (which implies weak stability) in some directions of C. It is therefore relevant to investigate into typical values of s_i. Unfortunately I can only provide a plausibility argument to the effect that most s_i will be close to 1 and only very few (if any at all) close to $1/2$. This argument goes as follows. Recall from (16) that F, because it is a minimizer of (15), can be written as $F = R(R + \alpha^{-2} I)^{-1}$, where $R = E[\mathbf{z}\,\mathbf{z}']$. This implies that the singular values s_i of F relate to the singular values s_i^R of R by $s_i = s_i^R \,/\, (s_i^R + \alpha^{-2})$. Furthermore, nonzero s_i must be greater than $1/2$. Assume the "worst case" that the smallest nonzero singular value of the (rank k) matrix F is $s_k = 1/2$, which translates to $s_k^R = \alpha^{-2}$. Let $s_{k-1}^R = s_k^R + \varepsilon$ be the next greater singular value of R. A simple argument based on the derivative $d\, s_i / d\, s_i^R$ evaluated at s_k^R yields that for small increments ε we get $s_{k-1} \approx 1/2 + 1/4\, \alpha^2\, \varepsilon$. In words, the singular values s_i of F grow away from $1/2$ by increments $1/4\, \alpha^2\, \varepsilon$. Since apertures are often in the order of 10 (or much larger), this means that we may expect that most if not all nonzero singular values of F will be close to 1. This is in fact what I consistently observe in simulations, like in the demos above (Figure 9). It is a desirable effect because according to the considerations made in this subsection before, singular values $s_i \approx 1$ or $s_i = 0$ foster noise robustness of auto-adaptation.

- The analyses available so far can explain the robustness of conceptor auto-adaptation to state noise. When at some stage of auto-adaptation the matrix $C(n)$ has k nonzero singular values and $N - k$ singular values equal to zero, the latter will be preserved under auto-adaptation because the states $\mathbf{z}(n) = C(n)\, \mathbf{r}(n)$ that enter the adaptation of $C(n)$ will have zero components in the $N - k$ directions associated with the zero singular values. The effect of state noise on the nonzero $s_i(n)$ is to drive them closer to 1 than they would go without state noise. Conceptors eventually converged to with state noise are not identical, but similar to variants that would be obtained without noise: the difference being that the former are (even) more rectangular than the latter. This favourable situation however hinges on the condition that the post-cue C^{cue} has (many) zero singular values. In the simulations in previous subsections

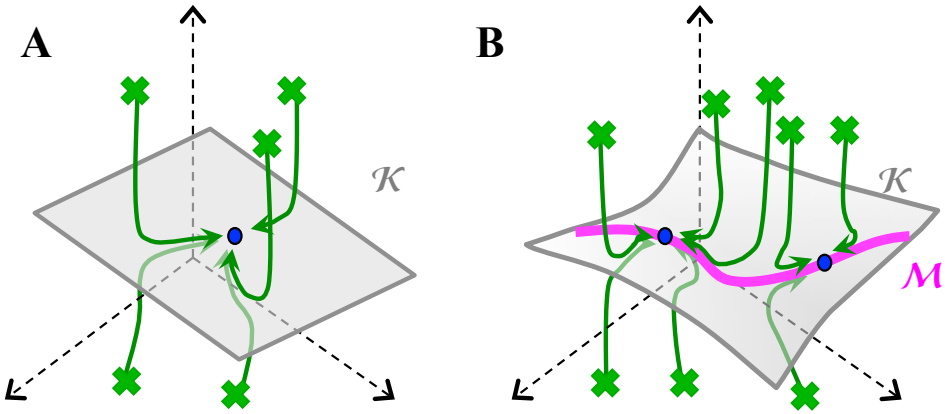

Figure 11: Hypothetical phase portraits of C autoadaptation in the parameter space of C (schematic). Blue points show stable fixed point solutions C. The gray plane in **A** represents the center manifold $\mathcal{K}$ and in **B** the merged center manifolds of neighboring fixed points C. Green arrows show sample trajectories of late stages of C adaptation. Green crosses mark the starting points of the adaptation trajectories set by the cueing procedure. **A**: When a small number of patterns has been loaded, individual stable fixed point conceptors C are created. **B**: In the case of learning a d-parametric pattern class, fixed point solutions C_i become located within a d-dimensional pattern manifold $\mathcal{M}$ (magenta line). For explanation see text.

this was achieved by (i) initializing $C(0) = \mathbf{0}$ and (ii) not inserting state noise during the cueing—state noise insertion in that sensitive period would have led to a large number of nonzero singular values in C^{cue}. If state noise or other perturbations are unavoidable during cueing, it will be a practical idea to simply zero small singular values of the developing $C(n)$ through a thresholding mechanism.

Based on these insights I tentatively offer the following qualitative account of conceptor auto-adaptation dynamics.

This dynamics evolves in the N^2-dimensional parameter space of conceptor (or conceptor candidate) matrices $C(n)$ under the assumption of a small enough adaptation rate λ, such that we may use the ODE (20). I distinguish two cases depending on whether few or many patterns have been loaded. Figure 11 gives a graphical impression.

Representing individual patterns. This corresponds to the situation called "rote learning" in Section 3.2. Here I assume that a small number of patterns p^j have been loaded, and that they become represented by corresponding isolated attracting fixed points F^j of (20). The dynamics in a neighborhood of F^j then should be characterized by a (very) slow timescale of attraction in the directions of the center manifold $\mathcal{K}$ and faster attraction in orthogonal directions (Figure 11 **A**). The attraction orthogonal to

$\mathcal{K}$ may in turn be governed by a wide range of different timescales due to what we have learnt about the dependency of eigenvalues of J_{F^j} on the singular values of F^j.

Representing a pattern family. When many patterns p^j from a family characterized by d parameters have been loaded, we have seen in Section 3.2 that any pattern from that family will be re-generated upon cue equally well. I hypothesize that during the loading procedure a d-dimensional manifold $\mathcal{M}$ is created which represents the pattern family and which superficially appears like a line/plane attractor: when by virtue of the cueing procedure the auto-adaptation is initialized into the vicinity of $\mathcal{M}$, one will observe an auto-adaptation toward $\mathcal{M}$ (Figure 11 **B**). The dimension d of $\mathcal{M}$ will typically be much smaller than the dimension $k(N-k)$ of the center manifold of conceptors in $\mathcal{M}$. I hypothesize that $\mathcal{M}$ is embedded as a submanifold in a $k(N-k)$-dimensional manifold $\mathcal{K}$ which can be understood as a merge of the (locally defined) center manifolds of conceptors sitting in $\mathcal{M}$. The convergence toward $\mathcal{M}$ leads to a good reconstruction of the cue pattern after a "reasonable" adaptation time. It is this convergence which was exploited for 10,000 step runtimes in the experiment shown in Figure 10. However, as remarked earlier, line/plane attractors are non-generic objects and unlikely to exist in empirical or randomly created systems. Therefore, for very long runtimes I would predict a very slow drift within $\mathcal{M}$ that lets auto-adaptation ultimately end up in some isolated attracting fixed point within $\mathcal{M}$. According to this view, line/plane attractors only appear phenomenally on some intermediate timescale, and hence the class learning effect described in Section 3.2 should degrade for very long adaptation times.

In order to substantiate this interpretation of an only "ephemeral" appearance of a line/plane attractor, I carried out a long-duration auto-adaptation simulation. Ten 5-periodic patterns were loaded into a small ($N = 50$) reservoir. These patterns represented ten stages of a linear morph between two similar patterns p^1 and p^{10}, resulting in a morph sequence $p^1, p^2, \ldots, p^{10}$ where $p^i = (1 - (i-1)/9)\, p^1 + ((i-1)/9)\, p^{10}$, thus representing instances from a 1-parametric family. Considering what was found in Section 3.2, loading these ten patterns should enable the system to re-generate by auto-adaptation any linear morph p_{test} between p^1 and p^{10} after being cued with p_{test}.

After loading, the system was cued for 20 steps with 20 different cues. In each of these $j = 1, \ldots, 20$ conditions, the cueing pattern p^j_{test} was the j-th linear interpolation between the loaded p^1 and p^{10}. At the end of the cueing, the system will be securely driven into a state $\mathbf{z}$ that is very accurately connected to re-generating the pattern p^j_{test}, and the conceptor matrix that has developed by the end of the cueing would enable the system to re-generate a close simile of p^j_{test} (a post-cue log10 NRMSE of about -2.7 was obtained in this simulation).

After cueing, the system was left running in conceptor auto-adaptation mode for 1 Mio timesteps, with an adaptation rate of $\lambda = 0.01$. At times $n = 1, 1000, 10000, 1e^6$ the stage of convergence was assessed as follows. The pairwise distances between the twenty autoconceptors $C^j(n)$ were compared, resulting in a 20×20 distance matrix $D(n) = (\|C^k(n) - C^l(n)\|_{\text{fro}})_{k,l=1,\ldots,20}$. Figure 12 shows color plots of these distance matrices. The outcome: at the beginning of autoadaptation ($n = 1$), the 20 autoconceptors are spaced from each other by distances proportional to their morphing distances. In terms of the

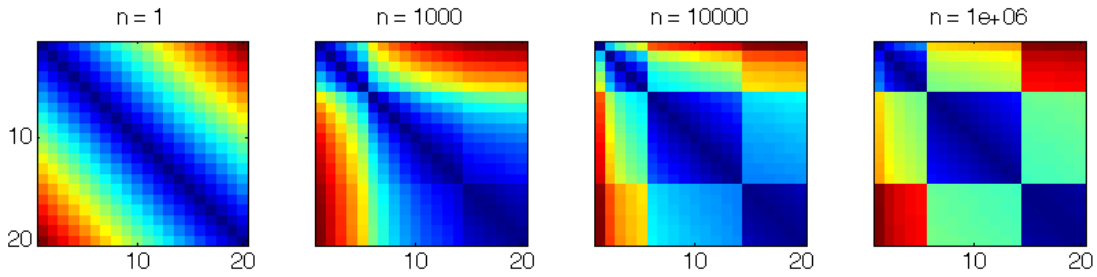

Figure 12: Numerical exploration of fixed point solutions under C auto-adaptation. Each panel shows pairwise distances of 20 conceptor matrices obtained after n auto-adaptation steps, after being cued along a 20-step morph sequence of cue signals. Color coding: blue: zero distance; red: maximum distance. For explanation see text.

schematic in Figure 11 **B**, they would all be almost equi-distantly lined up on the manifold $\mathcal{M}$, which is a line in this case of a 1-parametric pattern family. Then, as the adaptation time n grows, they contract toward three point attractors within $\mathcal{M}$ (which would correspond to a version of 11 **B** with three isolated fixed point attractors within $\mathcal{M}$). These three point attractors correspond to the three dark blue squares on the diagonal of the last distance matrix shown in Figure 12.

This singular simulation cannot, of course, provide conclusive evidence that the qualitative picture proposed in Figure 11 is correct. A rigorous mathematical characterization of the hypothetical manifold $\mathcal{M}$ and its relation to the center manifolds of fixed point solutions of the adaptation dynamics remains to be worked out.

Plane attractors have been proposed as models for a number of biological neural adaptation processes (summarized in Eliasmith (2005)). Specifically, the fact that animals can fix their gaze in arbitrary (continuously many) directions has been modelled by plane attractors in the oculomotoric neural control system. Each gaze direction corresponds to a (controlled) constant neural activation profile. It would be interesting to carry out in-vivo experiments of gaze direction fixation where the animal (or human) is cued with a direction target, which is then removed, under the instruction to preserve the gaze direction after target removal for an extended timespan. The mathematical model outlined above would predict that the gaze direction would start drifting, possibly becoming ultimately arrested in one of a number of isolated, "preferred" gaze directions.

3.4 Single-Step Content-Addressing by Thresholding

We observed above that approximately rectangular conceptors evolved in auto-adaptation and that directions in $\mathbf{z}$ states which are nulled by C^{cue} remain nulled during auto-adaptation. This suggests a drastic shortcut of the recall procedure: take the preliminary C^{cue} which is available immediately after the cue period, then transform its singular value profile to a binary 0-1-rectangular shape by thresholding to give $\overline{C^{\text{cue}}}$, and use this conceptor for pattern re-generation. Accordingly, I re-ran the 5-periodic and mix-of-sines demos from

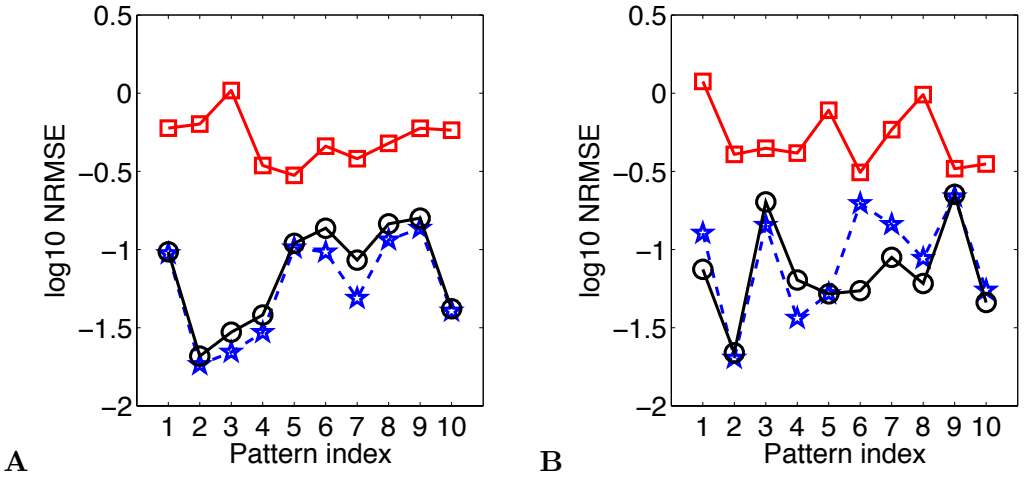

Figure 13: Using single-step adapted conceptors $\overline{C^{\text{cue}}}$ for cued pattern recall. **A**: 5-periodic patterns, **B**: mix-of-sines. Red and blue plots are identical to Figure 9 and represent post-cue NRMSE from C^{cue} and $C(t_3)$. The black plot gives the NRMSE obtained from $\overline{C^{\text{cue}}}$.

Section 3.1.2 with the same reservoirs and cueing conditions, and directly at the end of the cueing phase I computed $\overline{C^{\text{cue}}}$ from C^{cue} as follows:

1. compute the SVD $USU' = C^{\text{cue}}$,
2. recompute the entries s_i on the diagonal of S to new, binary values by thresholding, obtaining $\overline{S}$ with diagonal values

$$\overline{s_i} = \begin{cases} 1, & \text{if } s_i > \tau \\ 0, & \text{else} \end{cases}$$

   for some threshold τ,
3. recombine into $\overline{C^{\text{cue}}} = U\overline{S}U'$,
4. immediately use $\overline{C^{\text{cue}}}$ for re-generating a pattern and report the matching NRMSE with the original pattern that was used for the cue.

Figure 13 displays the outcome. I used thresholds τ of 0.5 and 0.01 respectively for the 5-periodic and mix-of-sines experiments. For both kinds of patterns the re-generation quality obtained from $\overline{C^{\text{cue}}}$ conceptors is essentially the same as the quality obtained from the auto-adapted $C(t_3)$ after 500 resp. 10,000 steps for the two kinds of patterns.

Considering that thresholded $\overline{C^{\text{cue}}}$ are much faster and cheaper to create than online-adapted conceptors, yet (in these simulations) are of the same quality, it is natural to ask why one should bother about online-adapted autoconceptors at all instead of focussing all attention on thresholded conceptors. The reasons why I spent much effort on the former (and used much of the reader's patience) are manifold:

- There are other applications of online-adapted autoconceptors besides pattern recall from a memory. In Jaeger (2014) I employ conceptor auto-adaptation in hierarchical RNN demonstrator systems for tasks of signal denoising, pattern classification, and pattern mixture recognition, and in ongoing work I am using them for single-shot training of a nonlinear channel equalizer and blind signal separation. In all of these cases, an adaptive RNN-based system has to be able to smoothly adjust its reactions to a non-stationary input stream, requiring a continuously ongoing gradual adaptation.

- A serious drawback of the above auto-adaptation mechanism is that it is based on stochastic gradient descent in a high-dimensional space, incurring the well-known stability issues which mandate small adaptation rates, hence yield slow convergence. In Jaeger (2014) I introduce a simplified version of conceptors which basically constrains conceptor matrices C to diagonal form, leading to decoupled adaptation rules for the diagonal parameters which admits individual and very large adaptation rates. While systems of this kind work well in some specific scenarios (demos in Jaeger (2014), Section 3.15), I am not satisfied with them in general, mainly because the value range for well-working apertures becomes very narrow. My current research concentrates on new types of conceptors which combine fast online adaptivity with the robustness properties found in matrix conceptors.

- In computational neuroscience, attractor-like phenomena of all sorts are widely investigated as neural mechanisms for representing and processing information in neural systems (partial overviews in Durstewitz et al. (2000); Fusi and Wang (2016); Jaeger (2012)). The multiple-timescale attractor phenomenology of conceptor auto-adaptation discussed in the previous section sheds some further light on this field, especially with respect to the role of line/plane attractors in the representation and addressing of continuous memory items.

- Biological implausibility. The fast thresholding mechanism invokes a matrix SVD computation. Much research has been devoted to determine biologically plausible neural algorithms for the closely related task of computing a PCA or of finding some cost-optimal projection subspaces (Oja, 1982; Pehlevan et al., 2015). However, these neural algorithms share with my stochastic gradient conceptor adaptation the shortcoming of slow convergence, and biological plausibility remains debatable in my view.

3.5 Comparison with Hopfield Networks

As I mentioned in the Introduction, Hopfield networks and its relatives (Willshaw et al., 1969; Cooper, 1973; Kohonen, 1974; Palm, 1980; Hopfield, 1982) are the paradigmatic model of content-addressable neural long-term memories. For brevity I will use the term *Hopfield networks* (HNs) as an umbrella term for this family of models, and refer to the networks introduced in this section as autoconceptor networks (ACNs). ACNs are analogous to HNs in some ways and different in others:

Nature of memory items. Patterns stored in (auto-associative) HNs are static (often images), though hetero-associative HN variants can be trained to re-generate sequences of static patterns—I briefly discussed this in the Introduction. ACNs intrinsically

host temporal patterns. The pattern sequences of static patterns learnt by sequence-representing HNs differ in kind from the temporal patterns in ACNs. The former are "jump" transitions between a finite number of explicitly trained "waypoint" patterns, and in principle any transition order can be learnt. Temporal patterns in ACNs have what one might call intrinsic temporality; the reservoir state sequence cannot in general be re-trained in other orders. Specifically, ACNs can host non-periodic patterns sampled from ODE evolutions which is not possible with hetero-associative HNs. One might word this as, "hetero-associative HNs can store pattern sequences, and ACNs can store sequence patterns".

States and patterns. In HNs patterns are identified with certain network states. In ACNs patterns are state evolution processes.

Symmetric vs. directed synaptic connections. Standard auto-associative HNs have symmetric connections, which admits an analysis of HN dynamics in terms of descent in an energy landscape. Reservoir networks in ACNs with their asymmetric weight matrices do not admit an energy-based interpretation.

Pattern restauration. A hallmark of HNs is their ability to restore patterns perfectly and quickly from highly corrupted cues. This striking capability results from the fact that HN training creates well-defined local minima in the HN energy landscape which correspond 1-1 to target patterns; from any arbitrary cue the HN *must* settle in one of these perfect target patterns (setting aside the issue of spurious attractors in HNs). Furthermore, perfection of restauration is aided by the circumstance that HNs often are installed with binary neurons, representing binary patterns. In contrast, the continuous-valued states in ACNs and the vastly more complex multi-timescale dynamics of auto-adaptation limit the perfection and speed of pattern re-generation.

Class learning. ACNs can learn entire continuous-parametric pattern classes (with the caveats pointed out in Section 3.3), which HNs cannot.

All in all, while both HNs and ACNs are models of content-addressable neural long-term memories, their underlying neuro-dynamical working mechanisms and their performance characteristics are fundamentally—and interestingly—different.

4. Discussion

This article is the first peer-reviewed publication about conceptors and thereby assumes the role of an "official" introduction of this concept and name. New scientific concepts and terminology should be introduced with care and circumspection. So, what is a "conceptor"?

This article dealt with conceptors that come in the specific form of positive-semidefinite matrices derived from reservoir state correlation matrices R via $C = R\,(R + \alpha^{-2} I)^{-1}$. In Jaeger (2014) I also describe conceptors that are realized by a forward- and backprojection to/from a higher-dimensional random feature space. In my initial explorations (unpublished) I also used affine maps, as well as pure linear subspace projectors. Currently I am investigating a kind of conceptors built on the basis of a variant of self-organizing maps. In

the light of this diversity I wouldn't want to make any specific mathematical or algorithmical format a defining part of "conceptors".

This article focussed on neural long-term memory for temporal patterns. In Jaeger (2014) however I use conceptors also for static patterns (where an input pattern is transformed to a single neural state, not a sequence of states), and without loading (which is not necessary when conceptors are employed for non-generative tasks like pattern classification). I am also aware of ongoing work by others in static pattern recognition where backprop-trained deep feedforward neural networks are combined with conceptors. Thus, neither temporality of patterns, nor loading them persistently, nor the use of random networks are necessary attributes of "conceptors", the way I want them to be seen.

Then what is left? Here is an outline of how I would like the concept of conceptors to be understood:

1. *Set-up: patterns encoded by states.* Given: some computational framework where a diversity of **patterns** p^j are **encoded** by probability distributions P^j over a metric state space X. *Comment: patterns can be temporal or static. The state space is typically high-dimensional, and encodings would typically be "distributed".*

2. *Filtering of encodings.* A **conceptor** C^j is a **map** associated with a pattern p^j which transforms state vectors $x \in X$ to state vectors $C^j(x) \in X$, and which is optimized in some way to preserve states x^j that are typical for pattern p^j: if $P^j(x)$ is large, then the distance $d(x, C^j(x))$ is small. Furthermore, C^j projects off-pattern states closer to pattern-typical regions of X: if $P^j(x)$ is small, then $P^j(C^j(x)) > P^j(x)$. *Comment: The map C^j can be instantiated by a mathematical formula, or by some neural circuit, or in any other way.*

3. *Conceptual operations.* Given a collection $\{p^j\}$ of patterns, on the associated collection of conceptors $\{C^j\}$ one has available an assortment of operations which can **construct** new conceptors from the conceptors contained in the original collection $\{C^j\}$. These operations should be interpretable as "cognitive-level" operations on concepts, and it is these operations that connect conceptors with concepts. *Comment: this is obviously vague. This article featured the following such operations: (i) conceptual blending by conceptor morphing, (ii) "focussing" by aperture adaptation. A large part of Jaeger (2014) is devoted to a third kind of conceptual operations on conceptors, namely, (iii) Boolean combinations of conceptors.*

My ultimate motivation to investigate conceptors is to establish an effective link between "low-level", "subsymbolic", "distributed" neural representations and processing mechanisms on the one hand, and a "high-level", "conceptual" organization and interpretability of such processing on the other hand. This intended use of conceptors, which is expressed in item 3. in the list above, is to some degree independent of the "mechanics" of conceptors, which is stated in item 2. It turns out that conceptors also can serve relevant "low-level" functionalities, especially neural noise suppression and stabilization of neural state dynamics.

Conceptors can be engineered into artificial neural architectures, and they might be instantiated in one way or the other by neural circuitry in biological systems. In such

cases they become procedurally effective in the concerned systems, implementing *conceptor mechanisms*. This was the perspective adopted in this article. But conceptors can also be used "epistemologically" by a researcher who investigates some (likely neural) information processing system which may or may not itself have have effective conceptor mechanisms inside. Then conceptors may become elements of a descriptive scientific language that talks *about* neural encodings of concepts. In Jaeger (2014) I describe in detail such scientific languages in the format of formal *conceptor logics*.

Acknowledgments

I am indebted to Emre Neftci for bringing the work of Sompolinsky and Kanter (1986) to my attention. The first ideas of conceptors were triggered by research within the European FP7 project AMARSi (contract 248311, `amarsi-project.eu`) on the modulation of neural pattern generators for humanoid robot motor control.

Appendix A. Documentation of Simulation Detail

Training and test errors are given as normalized root mean square error (NRMSE) between a signal $s(n)$ and its target $t(n)$, that is, by the square root of the mean square error divided by the variance of the target.

A.1 Detail for Section 2.1

Pattern definition: Pattern p^1 is a sinewave sampled at intervals of about 1.9944. Pattern p^2 alternates between -0.5 and 0.5.

Network set-up: Initial weights W^* were sampled from the normal distribution, then the weight matrix was rescaled to a spectral radius (largest absolute eigenvalue) of 1.3. Input and bias weights were sampled from the normal distribution, then scaled by 1.4 and 0.1 respectively.

Learning: For training W^{out} the reservoir was driven by iid input sampled from the normal distribution scaled by 1.5 for 200 steps (plus an initial washout of 100 steps). The linear regression was computed without regularization, resulting in a training NRMSE of 0.39 (training W^{out} from state patterns $\mathbf{x}^1(n), \mathbf{x}^2(n)$ would have given an NRMSE of 0.14). For the (not regularized) linear regression leading to W, again 200 states from each of the two patterns were used, leading to an NRMSE of 0.093 (average over the three neurons).

Recall: In order to numerically compare the re-generated patterns $y(n) = W^{\text{out}}\,\mathbf{x}^j(n)$ to the original patterns $p^j(n)$, sample trajectories of both were phase-aligned by (i) super-sampling with cubic spline interpolation by a factor of 20, (ii) finding the best-fitting phase shift, (iii) subsampling back to the original sampling rate, (iv) computing the NRMSE, which was 0.15 for the first and 0.13 for the second pattern.

Note: Parameters and the random seed for network set-up were hand-selected for good visual appearance of this demo example. Better recall accuracies were obtained with smaller apertures, but these would have given less instructive graphics. At any rate, a network size of $N = 3$ is far too small for high-precision results.

A.2 Detail for Section 2.2

Interpolation from 5-periodic to irrational sine (Fig. 3): A 100-unit reservoir was used, W^* scaled to a spectral radius of 1.6, input weights scaled to 1.6, bias vector to 0.3. Linear regression for W was regularized by a Tychonov regularizer of size $\varrho^2 = 0.0001$. Apertures for the two conceptors: 10 and 100.

Morphing between and beyond three patterns from a family (Fig. 4): Patterns were taken from the parametrized family with parameters A, B

$$p^{A,B}(n) = 2\left(\frac{1}{2}\sin(2\pi n/(P+A)) + \frac{1}{2}\right)^{\exp(B)} + 1, \quad A, B \in \mathbb{R}; |A| < P.$$

This family contains modulated versions of sines whose period lengths vary around a reference period P by differences A and whose shape is modulated by B. Figure 4 (top panels) shows the three stored patterns, which were defined by $P \approx 8.2$, $A = -1.1\sqrt{3}; 1.1\sqrt{3}/2; 0$, $B = -0.2, -0.2, 0.4$. Going upward by one tile changes a mixture from (a^1, a^2, a^3) to $(a^1 - 1/8, a^2 - 1/8, a^3 + 1/4)$, going right by one tile to $(a^1 - 1/4, a^2 + 1/4, a^3)$. The reservoir had 100 units, spectral radius of initial weight matrix: 2.2, input weight scaling: 2.2, bias scaling; 1.5, aperture of all conceptors 2.0. Ridge regression coefficient for loading: $\varrho = 0.001$.

A.3 Detail for Section 2.3

Preparation of training data. Fifteen human motion capture (mocap) sequences were retrieved from the public-domain mocap repository maintained by the Graphics Lab at Carnegie Mellon University (`http://mocap.cs.cmu.edu/`). Using the public mocap toolsuite provided by Burger and Toiviainen (2013), the marker trace format of the original mocap data was transformed into joint angle data in the following way. A human segment model with 17 segments was chosen. The original marker trace data was transformed into metric 3D trajectories of the segment endpoints (joints). The joint between the two hip joints was designated as "root". Let x, y denote the two ground coordinates and z the height coordinate (measured as distance from ground), and let δ (measured in rad) denote the horizontal pointing direction (in x, y) of the root.

The motion of the root joint was coded in the following four variables:

1. $h(n)$: simply the z coordinate of the Euclidean trajectory of the root,
2. $\dot{\delta}(n)$: the angular velocity (in rad/frame) of the root pointing direction,
3. $d_1(n), d_2(n)$: travel distance of root joint in the x, y plane of the current frame, relative to the horizontal pointing direction (Fig. 14), measured in mm/frame.

The locations of the other joints were coded relative to their predecessor joints in the kinematic chain tree, with the root joint as the top node, as follows:

1. The current frame n is horizontally shifted and rotated such that the root joint has (x, y)-position $(0, 0)$ and is pointing in the direction of the x-axis. For every joint j this results in three Euclidean coordinates $x_j(n), y_j(n), z_j(n)$.

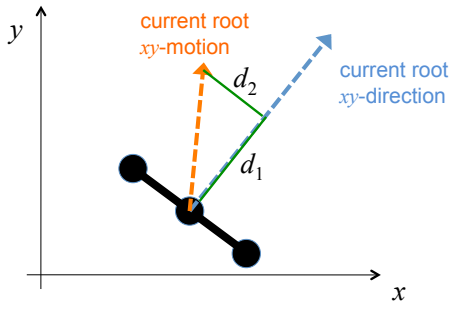

Figure 14: How the quantities d_1, d_2 geometrically relate to current root direction and root travel.

2. For each of the 19 non-root joints j, the current orientation was coded in three variables $s_j^x(n), s_j^y(n), s_j^z(n)$ as follows:
   (a) Determine the predecessor joint i of j (the adjacent joint that is closer to the root).
   (b) Compute the segment vector $s_0^j(n) = (x_j(n) - x_i(n), y_j(n) - y_i(n), z_j(n) - z_i(n))'$ and normalize it to $s^j(n) = s_0^j(n)/\|s_0^j(n)\|$. The three components of $s^j(n)$ are returned as coding the current spatial angle of the segment ending in joint j.

All in all this leads to a 61-dimensional coding of a body pose and root motion per frame. From such 61-dimensional sequences, joint trajectories in Euclidean task space can be recovered by a reversal of the coding procedure.

For each of the 15 original mocap timeseries, such a 61-dimensional encoding was produced. Each of the 61 signal components was shifted/scaled such that it ranged exactly in $[-1, 1]$, across all 15 patterns. These 15 range-normalized sequences were then loaded into the reservoir, and the readout weights were computed to recover them.

Reservoir setup. Different from all other simulations reported in the article, the motor pattern demonstration used a reservoir with leaky integration neurons:

$$\mathbf{x}(n+1) = (1-a)\,\mathbf{x}(n) + a\,\tanh(W\,\mathbf{x}(n) + W^{\text{in}}\,p(n+1) + \mathbf{b}),$$

where the leaking rate a is an additional tuning parameter. This alternative choice of network model does not affect any of the conceptor-related computations. Reservoirs based on leaky integration units effectively implement temporal smoothing which is indicated when slowly changing (relative to the network update cycle) patterns have to be processed.

Simulation parameters. Network size $N = 600$; leaking rate $a = 0.6$; spectral radius of W^* was 1; input and bias weights were sampled from the uniform distribution on $[-.8, .8]$. Loading each pattern used a washout of 50 network updates (corresponding to 5/12 seconds of simulated real time). Two of the input channels were found to act essentially only as noise sources (because they were essentially constant and after range normalization small fluctuations in the constant value were magnified); their input weights were nulled. Linear regression without regularization was used for computing W and W^{out}. The aperture was

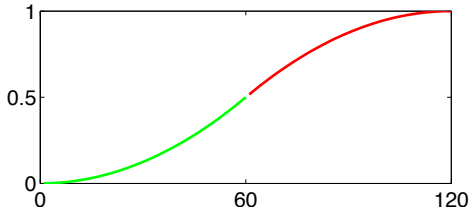

Figure 15: The morphing ramp $\varrho(n)$ used for smooth transitions between motor patterns.

set to $\alpha = 10$ for all fifteen conceptors except for the "slow walk" ($\alpha = 2$) and "boxing: jab" patterns ($\alpha = 20$; with $\alpha = 10$ the jab pattern would be reduced to a mildly active legwork without the jabs being triggered).

Training errors. The training NRMSE for W was 0.0068 and for W^{out} was 0.044.

Video creation. For creating the video, the durations of the individual patterns were set to yield an appealing overall choreography. Between every two successive pattern p^i, p^j evocations, a transition period of 120 network updates (= 1 simulated second) was inserted. In this transition period the preceding conceptor C^i was morphed into the succeeding conceptor C^j, using a morphing function $\varrho : \{1, \ldots, 120\} \to [0, 1]$ made from two suitably scaled and shifted branches of the square function (red and green lines in Fig. 15), via $C^{\text{morph}}(m) = (1 - \varrho(m))\, C^i(m) + \varrho(m) C^j(m) \quad [m = 1, \ldots, 120]$. While the reservoir was generating the behavioral sequence visualized in the video, the only external input to the system was the activation sequence of conceptors (shown in the left upper corner of the video).

The visualization of the re-generated motion sequence in the video keeps the simulated actor centered in the drawing box and illustrates its motion in the x, y plane by a moving floor tile pattern. The centering as well as the corresponding virtual floor motion is computed relative to the mean of the x, y coordinates of all 20 joints. This mean is therefore always centered on $x = y = 0$ in each displayed frame. The apparent foot slipping results from this "quick and dirty" centering method.

A.4 Detail for Section 2.4

Data generation. For the Rössler attractor, training time series were obtained from the standard Rössler ODE $\dot{x} = -(y + z), \;\; \dot{y} = x + a\, y, \;\; \dot{z} = b + x\, z - c\, z$ with $a = b = 0.2, c = 8$. The evolution of this system was Euler approximated with stepsize $1/200$ and the resulting discrete time series was then subsampled by 150. The x and y coordinates were assembled in a 2-dimensional driving sequence, where each of the two channels was shifted/scaled to a range of $[0, 1]$. For the Lorenz attractor, the ODE $\dot{x} = \sigma(y - x), \;\; \dot{y} = r\, x - y - x\, z, \;\; \dot{z} = x\, y - b\, z$ with $\sigma = 10, r = 28, b = 8/3$ was Euler-approximated with stepsize $1/200$ and subsequent subsampling by 15. The x and z coordinates were collected in a 2-dimensional driving sequence, again each channel normalized to a range of $[0, 1]$. The Mackey-Glass timeseries was obtained from the delay differential equation $\dot{x}(t) = \frac{\beta\, x(t-\tau)}{1 + x(t-\tau)^n} - \gamma\, x(t)$ with $\beta = 0.2, n = 10, \tau = 17, \gamma = 0.1$. An Euler approximation with stepsize $1/10$ was used. To obtain a 2-dim timeseries that could be fed to the reservoir through the same two input channels as the other attractor data, pairs $x(t), x(t - \tau)$ were combined into 2-dim

vectors. Again, these two signals were normalized to the $[0, 1]$ range. The Hénon attractor is governed by the iterated map $x(n+1) = y(n) + 1 - a\,x(n)$, $y(n+1) = b\,x(n)$, where I used $a = 1.4, b = 0.3$. The two components were filed into a 2-dim timeseries $(x(n), y(n))'$ with no resampling, and again normalization to a range of $[0, 1]$ in each component.

Reservoir setup. A 500-unit reservoir RNN was created with a normal distributed, 10%-sparse weight matrix W^* scaled to a spectral radius of 0.6. The bias vector b and input weights W^{in} (sized 400×2 for two input channels) were sampled from standard normal distribution and then scaled by 0.4 and 1.2, respectively. These scaling parameters were found by a coarse manual optimization of the performance of the pattern loading process. The network size was chosen large enough to warrant a robust trainability of the four chaotic patterns. Repeated executions of the experiment with different randomly initialized weights (not documented) showed no significant differences.

Pattern loading. The length of training signals used in loading was $L = 2500$ with a washout of $n_0 = 500$ for all four patterns. The ridge regression regularizer was $\varrho^2_W = 1e^{-6}$. Output weights were computed from the pattern-driven states with a regularizer $\varrho^2_{\text{out}} = 1e^{-8}$. The average (over neurons and the four patterns) NRMSEs obtained for the reservoir and readout weights were 0.0082 and 0.013, respectively.

A.5 Detail for Section 3.2

Scalings of reservoirs: spectral radius of W^*: 1.1; W^{in}: 1.1; b: 0.25. Aperture: 1000. Regularization coefficients $\varrho^2 = 0.01$ both for W^{out} and D. Data collection runlengths in loading: 500 (plus 100 washout) per pattern. Cueing: washout 100 steps, cue adaptation time 12 steps. No noise added to cue signal. State noise added during auto-adaptation with a signal-to-noise ratio of 1.

References

S.-I. Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Trans. on Computers*, C-21(11):1197–1207, 1972.

A. Billard and G. Hayes. DRAMA, a connectionist architecture for control and learning in autonomous robots. *Adaptive Behavior*, 7(1):35–63, 1999.

K. J. Boström, H. Wagner, M. Prieske, and M. de Lussanet. Model for a flexible motor memory based on a self-active recurrent neural network. *Human Movement Science*, 32: 880–898, 2013.

B. Burger and P. Toiviainen. MoCap Toolbox – A Matlab toolbox for computational analysis of movement data. In Roberto Bresin, editor, *10th Sound and Music Computing Conference*, pages 172–178, Stockholm, Sweden, 2013. KTH Royal Institute of Technology. URL `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mocaptoolbox`.

CMU Graphics Lab. Motion Capture Database. URL `http://mocap.cs.cmu.edu/`. retrieved Feb 2013.

A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *Proc. 25th ICML, Helsinki*, 2008.

L. N. Cooper. A possible organization of animal memory and learning. In B. Lundqvist and S. Lundqvist, editors, *Collective properties of physical systems: Medicine and Natural Sciences*, Nobel Symposia on Medicine and Natural Sciences, pages 252–264. Academic Press, New York and London, 1973.

R. Douglas and T. Sejnowski. Future challenges for the sciene and engineering of learning: Final workshop report. Technical report, National Science Foundation, 2008. URL `http://www.nsf.gov/sbe/SLCWorkshopReportjan08.pdf`.

D. Durstewitz, J. K. Seamans, and T. J. Sejnowski. Neurocomputational models of working memory. *Nature Neuroscience*, 3:1184–91, 2000.

C. Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural Computation*, 17:1276–1314, 2005.

R. M. French. Catastrophic interference in connectionist networks. In L. Nadel, editor, *Encyclopedia of Cognitive Science*, volume 1, pages 431–435. Nature Publishing Group, 2003.

S. Fusi and X.-J. Wang. Long-term, short-term and working memory. In M. Arbib and J. Bonaiuto, editors, *From Neuron to Cognition via Computational Neuroscience*, chapter 11, pages 319–344. MIT Press, 2016.

M. Gillies and B. Spanlang. Comparing and evaluating real time character engines for virtual environments. *Presence*, 19(2):95–117, 2010.

B. Goodrich and I. Arel. Unsupervised neuron selection for mitigating catastrophic forgetting in neural networks. In *Proc. IEEE 57th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 997–1000. IEEE, 2014.

S. Grossberg. Linking attention to learning, expectation, competition, and consciousness. In L. Itti, G. Rees, and J. Tsotsos, editors, *Neurobiology of attention*, chapter 107, pages 652–662. San Diego: Elsevier, 2005.

X. Hinaut and P. F. Dominey. A three-layered model of primate prefrontal cortex encodes identity and abstract categorical structure of behavioral sequences. *J. Physiology - Paris*, 105(1-3):16–24, 2011.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79:2554–2558, 1982.

J. Huang and M. Hagiwara. A combined multi-winner multidirectional associative memory. *Neurocomputing*, 48:369–389, 2002.

H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001. URL `http://minds.jacobs-university.de/pubs`.

H. Jaeger. Reservoir self-control for achieving invariance against slow input distortions. technical report 23, Jacobs University Bremen, 2010.

H. Jaeger. Long short-term memory in echo state networks: Details of a simulation study. Technical Report 27, Jacobs University Bremen, 2012. URL `http://minds.jacobs-university.de/pubs`.

H. Jaeger. Controlling recurrent neural networks by conceptors. Technical Report 31, Jacobs University Bremen, 2014. arXiv:1403.3369.

X. Jiang, V. Gripon, C. Berrou, and M. Rabbat. Storing sequences in binary tournament-based neural networks. *IEEE Trans. on Neural Networks and Learning Systems*, 27(5): 913–925, 2016.

M. I. Jordan. Serial order: a parallel distributed processing approach. In J. W. Donahoe and V. Packard Dorsel, editors, *Neural-Networks Models of Cognition*, chapter 25, pages 471–495. Elsevier, 1997. Abridged reprint of a technical report from 1986.

T. Kohonen. An adaptive associative memory principle. *IEEE Transactions on Computers*, 23(4):444–445, 1974.

J. F. Kolen and J. B. Pollack. Multiassociative memory. In *Proc. of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 785–789, 1991.

A. F. Krause, V. Dürr, B. Bl̈(a)sing, and T. Schack. Evolutionary optimization of echo state networks: multiple motor pattern learning. In *ANNIIP - 6th International Workshop on Artificial Neural Networks and Intelligent Information Processing*, 2010.

R. Laje and D. V. Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature Neuroscience*, 16(7):925–933, 2013.

L. Lukic, J. Santos-Victor, and A. Billard. Learning coupled dynamical systems from human demonstration for robotic eye-arm-hand coordination. In *IEEE-RAS International Conference on Humanoid Robots, Osaka 2012*, 2012.

M. Lukosevicius. A practical guide to applying echo state networks. In K.-R. Müller, G. Montavon, and G. Orr, editors, *Neural Networks Tricks of the Trade, Reloaded*, LNCS, pages 659–686. Springer Verlag, 2012.

G. Manjunath and H. Jaeger. Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks. *Neural Computation*, 25(3):671–696, 2013.

N. M. Mayer and M. Browne. Echo state networks and self-prediction. In *Biologically Inspired Approaches to Advanced Information Technology*, volume 3141 of *LNCS*, pages 40–48. Springer Verlag Berlin / Heidelberg, 2004.

E. Oja. A simplified neuron model as a principal component analyzer. *J. Math. Biol.*, 15: 267–273, 1982.

R. W. Paine and J. Tani. How hierarchical control self-organizes in artificial adaptive systems. *Adaptive Behaviour*, 13(3):211–225, 2005.

G. Palm. On associative memory. *Biol. Cybernetics*, 36(1):19–31, 1980.

C. Pehlevan, T. Hu, and D. B. Chklovskii. A Hebbian/anti-Hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data. *Neural Computation*, 27(7):1461–1495, 2015.

J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2):77–105, 1990.

F. R. Reinhart and J. J. Steil. Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant pa-10 robot. In A. Stoica, E. Tunsel, T. Huntsberger, T. Arslan, S. Vijayakumar, and A. O. El-Rayis, editors, *LAB-RS 2008, vol. 1*, pages 35–40, 2008.

R. F. Reinhart and J. J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, 19(1–2):27–46, 2011. DOI 10.1007/s12591-010-0067-x is an 2010 online pre-publication.

G. J. Rinkus. *A combinatorial neural network exhibiting episodic and semantic memory properties for spatiotemporal patterns*. Phd thesis, Boston University, 1996.

H. Ritter and T. Kohonen. Self-organizing semantic maps. *Biological Cybernetics*, 61: 241–254, 1989.

L. Shastri. Advances in Shruti – a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Artificial Intelligence*, 11:79–108, 1999.

H. Sompolinsky and I. Kanter. Temporal association in asymmetric neural networks. *Physical Review Letters*, 57(22):2861–2864, 1986.

D. Sussillo and L. Abbott. Transferring learning from external to internal weights in echo-state networks with sparse connectivity. *PLoS ONE*, 7(5):e37372, 2012.

I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted boltzmann machine. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS 08)*, pages 1601–1608, 2009.

I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *ICML 2011 (online)*, 2011. URL `http://www.icml-2011.org/papers.php`.

G. W. Taylor, G. E. Hinton, and S. T. Roweis. Two distributed-state models for generating high-dimensional time series. *Journal of Machine Learning Research*, 12(March):1025–1068, 2011.

D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222(June 7):960–962, 1969.

J. Wright and I. Jordanov. Intelligent approaches in locomotion. In *WCCI 2012 IEEE World Congress on Computational Intelligence*, pages 1–8, 2012. doi: dx.doi.org/10.1109/IJCNN.2012.6252537.

F. wyffels and B. Schrauwen. Design of a central pattern generator using reservoir computing for learning human motion. In *Advanced Technologies for Enhanced Quality of Life, 2009. AT-EQUAL '09*, pages 118–122. IEEE, 2009.