Recurrent Kernel Machines: Computing with Infinite Echo State Networks

Michiel Hermans
michiel.hermans@ugent.be
Benjamin Schrauwen
Benjamin.schrauwen@ugent.be
Department of Electronics and Information Systems, Ghent University, 9000 Ghent, Belgium

Echo state networks (ESNs) are large, random recurrent neural networks with a single trained linear readout layer. Despite the untrained nature of the recurrent weights, they are capable of performing universal computations on temporal input data, which makes them interesting for both theoretical research and practical applications. The key to their success lies in the fact that the network computes a broad set of nonlinear, spatiotemporal mappings of the input data, on which linear regression or classification can easily be performed. One could consider the reservoir as a spatiotemporal kernel, in which the mapping to a high-dimensional space is computed explicitly. In this letter, we build on this idea and extend the concept of ESNs to infinite-sized recurrent neural networks, which can be considered recursive kernels that subsequently can be used to create recursive support vector machines. We present the theoretical framework, provide several practical examples of recursive kernels, and apply them to typical temporal tasks.

1 Introduction

Many schemes for training recurrent neural networks, powerful computational entities with a wide application domain, have been studied. Two main lines of research exist. The first focuses on gradient-based techniques to train all the parameters in the network, of which the best-known example is backpropagation through time (Rumelhart, Hinton, & Williams, 1986). Though often powerful, this approach is limited by problems such as slow convergence, many local optima, bifurcations, and high computational costs (Pearlmutter, 1995; Suykens, Moor, & Vandewalle, 2008). The second approach is to use large, randomly initiated neural networks. The internal parameters remain completely untrained, and instead an instantaneous linear readout layer is trained to optimally project the hidden state of the network onto the desired output by linear regression. This approach does not suffer from the problems typically found in error gradient methods.

Originally this idea had been separately developed for sigmoid nodes (Jaeger, 2001) and spiking neurons (Maass, Natschläger, & Markram, 2002), where the respective approaches were called echo state networks (ESN) and liquid state machines (LSM), but there are no limitations to the types of networks one could use (Fernando & Sojakka, 2003; Jones, Stekel, Rowe, & Fernando, 2007). Indeed, the network does not even have to be a true neural network in the common sense: any nonlinear, dynamical system with the right properties (most important is fading memory; Jaeger, 2001) can potentially be used in this approach. The umbrella term that encompasses all variants of this approach is *reservoir computing* (RC) (Lukosevicius & Jaeger, 2009; Verstraeten, Schrauwen, d'Haene, & Stroobandt, 2007), where the dynamical system is considered a reservoir of rich nonlinear dynamics.

This approach often performs very well on real-life problems such as speech recognition (Skowronski & Harris, 2007; Verstraeten, Schrauwen, & Stroobandt, 2006), phoneme recognition (Triefenbach, Jalalvand, Schrauwen, & Martens, 2010), robot navigation (Antonelo, Schrauwen, & Stroobandt, 2008), and time series prediction (Wyffels & Schrauwen, 2010), and it even yields state-of-the-art performance on the modeling of a chaotic attractor (Jaeger & Haas, 2004). At first, this was considered to be surprising, as the networks are random, and as such the neuron responses are random functions of the history of the input data. Subsequently a new point of view emerged roughly stating that if there exists a sufficiently broad set of functions on the input data stream, it is always possible to approach the desired output by finding the optimal combination of those functions. Indeed, as the size of the network goes to infinity, it has been shown theoretically that all possible functions on the input data can be approximated arbitrarily well (Maass et al., 2002; Maass, Joshi, & Sontag, 2007; Schäfer & Zimmerman, 2006).

A link has often been made between RC and kernel machines (Schmidhuber, Wierstra, Gagliolo, & Gomez, 2007; Shi & Han, 2007), since both techniques essentially map the input data to a high-dimensional space, called feature space, in which classification or regression is then performed linearly. In the case of reservoirs, this mapping is performed explicitly, as the hidden state of the reservoir is mapped directly onto the output. For kernel machines, such as support vector machines (SVMs; Boser, Guyon, & Vapnik, 1992), this mapping is not computed explicitly but is performed by the so-called kernel trick. Here, it is possible to define a dot product between two representations in feature space as a function that operates on two data points (called a kernel function). One can then use certain data points of a training set as so-called support vectors to define a linear map within feature space. Next, one can use quadratic programming approaches to find optimal support vectors and their corresponding weights. Interestingly, for certain simple kernel functions (such as the gaussian RBF kernel), feature space is infinite-dimensional. There is a link between this infinite-dimensional feature space and applying infinite-sized neural networks, as specified in Neal (1996) and Williams (1998). Specifically it is possible to associate a kernel function with an infinite feedforward neural network. In this work, we extend this idea to recurrent networks, and we define the associated kernel functions.

Usually when one applies SVMs on temporal problems, time is artificially represented in space by using a sliding time window of the data as input. This technique is related to the Markov property, which states that temporal dependencies are limited to a finite history of the time series. One important difference between RC and SVMs is the fact that the kernel in reservoirs is explicitly temporal: the states will depend on the recent history of the input, and not just the current input. This allows reservoirs to process information that is explicitly coded in time. In this letter, we derive a way to extend the concept of infinite neural networks to infinite recurrent neural networks. As such, we shall essentially bridge the gap between two machine learning techniques and define the "ultimate" ESN in the form of a kernel function that operates recursively on time series. We show the connection between the dynamics of ESNs and the evolution in the recursion in the kernels of the associated infinite neural networks and demonstrate this by investigating performance on temporal tasks.

We have structured this letter as follows. In sections 2 and 3, we respectively elaborate on ESNs and explain the concept of an infinite recurrent network and its associated kernel function. In section 4, we present a method to introduce recurrence in the previous definition and give some important examples of recursive kernels. After this, we investigate the link between known properties of the dynamics in ESNs and parameters of the recursive kernels in section 5. To validate our results, we test the recursive kernels by applying them on two temporal problems in section 6. Finally, in section 7, we discuss the results and draw overall conclusions.

2 Echo State Networks

Among the most widespread variants of RC are ESNs. Essentially, a recurrent network with randomly drawn internal connections and randomly drawn connections from input to hidden nodes is constructed, and the hidden state of the network evolves according to

$$\mathbf{a}(t+1) = f(\mathbf{W}\mathbf{a}(t) + \mathbf{V}\mathbf{s}(t+1)), \tag{2.1}$$

$$\mathbf{y}(t+1) = \mathbf{U}\mathbf{a}(t+1), \tag{2.2}$$

where f is the activation function, $\mathbf{a}$ is the hidden state vector, $\mathbf{s}(t)$ is the input signal at time t, and $\mathbf{W}$ and $\mathbf{V}$ are the internal connections and the input-to-network connections, respectively. The output weights $\mathbf{U}$ are the only weights that are trained and used to project the hidden state onto the output $\mathbf{y}(t)$. Typically the function f is a sigmoid function like the hyperbolic tangent. In that case, it is straightforward to characterize the dynamics by linearizing equation 2.2 around the origin. What is found is that the linearized system is asymptotically stable when the largest singular value of $\mathbf{W}$ is smaller than 1 (Jaeger, 2001). In practice, one uses the spectral radius ρ of the system, as this gives a better indication of the dynamics of the system. If ρ is greater than 1, the hidden state in the linearized system will start to grow exponentially. In the nonlinear version, this growth will be quenched by the saturating parts of the sigmoid function. Usually the system will go to a fixed point, start to oscillate, or become chaotic.

The rule of thumb for initializing reservoir weights is to keep the spectral radius smaller than or equal to 1.[1] If it is close to 1, the network states will decay to the fixed point only slowly, and as such, they will depend on a relatively long history of the input. If the spectral radius is significantly smaller than 1, the states will depend on only a short history of the input. When it is greater than 1, the states can in principle depend on the entire history of the input, which is usually considered undesirable.

The property of the network to depend on the recent history of the input signals is colloquially called fading memory (Boyd & Chua, 1985) and also echo state property (Jaeger, 2001), and is the key to the success of RC. One can tune the memory depth of the system by tuning the spectral radius of the connection matrix, where usually a trade-off has to be made between precision and the length of memory.

The other two main parameters that are identified as being of importance for ESNs are the scaling of input weights and the scaling of an optional bias term (not explicitly shown in equation 2.2). The input scaling will determine how far the hidden states are pushed away from the linear part of the activation function by the input; in other words, it will determine the overall nonlinearity of the reservoir. Typically an increase in nonlinearity is detrimental to the memory depth of the system, as the quenching parts of the activation function will decrease the effective spectral radius (Verstraeten et al., 2007), the mean spectral radius of the Jacobian of the system. The bias term is necessary for certain tasks where the desired output is not just an odd function of the input.[2]

Many ESN implementations also include leaky integrators in each neuron (Jaeger, Lukosevicius, & Popovici, 2007). This allows tuning the inherent timescale of the dynamics of the reservoir, which is another parameter of importance but not discussed in detail in this letter.

As we will show, all of these properties have counterparts in recursive kernels associated with infinite neural networks. Specifically, it is possible to identify a parameter that has a meaning equivalent to the spectral radius.

[1]In fact, the optimal value is heavily task dependent, and in many cases a spectral radius much smaller than 1, or in some cases even much greater than 1 will be optimal.

[2]A function $f(x)$ is odd when $f(-x) = -f(x)$. This is the case for hyperbolic tangents, the nonlinearity of choice in ESNs.

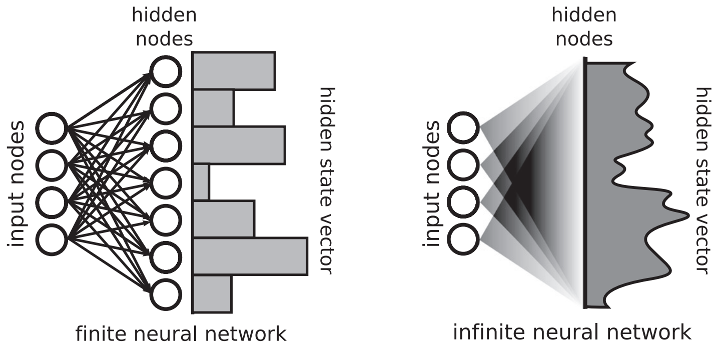

Figure 1: Schematic display of a finite versus an infinite neural network.

In the next section, we elaborate on the concept of infinite neural networks and give a formal definition of the associated kernel function.

3 Infinite Neural Networks

In this section, we first derive the kernel corresponding to an infinite feedforward neural network. In the next section, we extend this notion to recursive neural networks by introducing recursive kernels.

The state of the *i*th hidden neuron in a feedforward neural network is typically given by

$$a_i = f\left(\mathbf{w}_i, \mathbf{u}\right), \tag{3.1}$$

in which $\mathbf{w}_i$ is the vector of input weights for the *i*th neuron (optionally including a bias term, associated with an extra input dimension, which remains constant), $\mathbf{u}$ is the input vector, and f is the activation function. In a multilayered perceptron, $f\left(\mathbf{w}_i, \mathbf{u}\right) = f\left(\mathbf{w}_i \cdot \mathbf{u}\right)$, with f usually a sigmoid activation function. For radial basis function networks, the equation becomes $f\left(\mathbf{w}_i, \mathbf{u}\right) = f\left(||\mathbf{w}_i - \mathbf{u}||^2\right)$, with f usually an exponential function.

Extending the hidden layer to an infinite layer is straightforward, as has been shown in Williams (1998) and Neal (1996): all possible neurons correspond to all possible sets of input weights (and bias terms); hence, the neurons of the hidden layer will form a continuum that maps each point in the space of input weights to a neuron state, depending on the input vector. This concept is depicted schematically in Figure 1.

Obviously such a mapping can never be performed explicitly. However, it is possible to define a dot product in the Hilbert space that corresponds to the infinite-dimensional hidden state. This dot product is the corresponding kernel function for that type of network,

$$k(\mathbf{u}, \mathbf{v}) = \int_{\Omega_\mathbf{w}} d\mathbf{w} P(\mathbf{w}) f(\mathbf{u}, \mathbf{w}) f(\mathbf{v}, \mathbf{w}), \tag{3.2}$$

where $\Omega_{\mathbf{w}}$ is the space in which $\mathbf{w}$ is defined and $P(\mathbf{w})$ is the probability distribution of the input weights. Notice that this kernel is not always well defined and is not necessarily positive definite. Whether this can be a useful kernel function, that is, whether it fulfills the Mercer condition (Vapnik, 1995), will depend on P and f. Also, only a limited number of cases will give an analytically tractable solution.

Notice that equation 3.2 is similar to the typical procedure used in gaussian processes (Rasmussen & Williams, 2006), where the parameters are integrated out over a certain prior distribution function.

3.1 Gaussian Radial Basis Function Networks. One example of the previously introduced kernel type uses normalized gaussian radial basis functions as activation functions, and a distribution (in this case, an improper prior) $P(\mathbf{w}) = 1$:

$$k(\mathbf{u},\mathbf{v}) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{N}{2}} \int_{\Omega_{\mathbf{w}}} d\mathbf{w} \exp\left(\frac{-||\mathbf{w}-\mathbf{u}||^2 - ||\mathbf{w}-\mathbf{v}||^2}{2\sigma^2}\right), \tag{3.3}$$

which can be shown (Williams, 1998) to be equal to a gaussian kernel: $k(\mathbf{u},\mathbf{v}) = \exp(\frac{-||\mathbf{u}-\mathbf{v}||^2}{4\sigma^2})$.

3.2 Error Function Networks. Another important example has also been elaborated on in Williams (1998). It is possible to calculate an analytical solution for equation 3.2 if the neurons are perceptrons with an error function: $\text{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x \exp(-t^2)dt$, as nonlinearity. The distribution of the input weights is assumed gaussian, with a covariance matrix $\boldsymbol{\Sigma}$. Bias is taken into account by concatenating the input vectors with a constant element equal to 1. The error function has a sigmoid shape similar to the hyperbolic tangent (save for a different slope around the origin). The shape of both functions is plotted in Figure 2. This kernel can serve as a scaffold to extend typical ESNs to the infinite domain. The kernel, which we denote as the arcsine kernel throughout the rest of this letter, has the following expression:

$$k(\mathbf{u},\mathbf{v}) = \frac{2}{\pi}\arcsin\left(2\frac{\mathbf{u}\boldsymbol{\Sigma}\mathbf{v}^{\mathsf{T}}}{\sqrt{\left(1+2\mathbf{u}\boldsymbol{\Sigma}\mathbf{u}^{\mathsf{T}}\right)\left(1+2\mathbf{v}\boldsymbol{\Sigma}\mathbf{v}^{\mathsf{T}}\right)}}\right). \tag{3.4}$$

If we assume that $\boldsymbol{\Sigma}$ is diagonal, with σ^2 on the diagonal and σ_b^2 as the final element (corresponding to the variance of the bias distribution), this reduces to

$$k(\mathbf{u},\mathbf{v}) = \frac{2}{\pi}\arcsin\left(\frac{2\sigma^2\mathbf{u}\mathbf{v}^{\mathsf{T}} + 2\sigma_b^2}{\sqrt{\left(1+2\sigma^2\mathbf{u}\mathbf{u}^{\mathsf{T}} + 2\sigma_b^2\right)\left(1+2\sigma^2\mathbf{v}\mathbf{v}^{\mathsf{T}} + 2\sigma_b^2\right)}}\right). \tag{3.5}$$

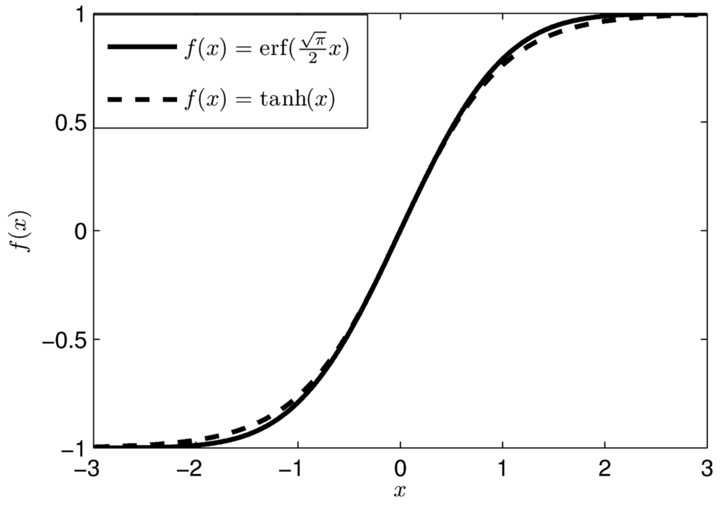

Figure 2: Shape of the error function compared to that of the hyperbolic tangent. The argument of the error function has been rescaled to match a slope of 1 around the origin.

3.3 Linear Rectifier Function Networks. Another important example that is analytically tractable is a feedforward network with powers of linear rectifier functions as activation functions. First worked out in Cho and Saul (2010), the setup is the same as the previous one, using a gaussian distribution of weights and an activation function $f(x) = \max\{0, x\}^p$. The resulting kernel function is given by

$$k_p(\mathbf{u}, \mathbf{v}) = \frac{1}{\pi} \|\mathbf{u}\|^p \|\mathbf{v}\|^p J_p(\theta), \tag{3.6}$$

with

$$J_p(\theta) = (-1)^p (\sin\theta)^{2p+1} \left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^p \left(\frac{\pi - \theta}{\sin\theta}\right), \tag{3.7}$$

and θ is the angle between $\mathbf{u}$ and $\mathbf{v}$:

$$\theta = \arccos\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\,\|\mathbf{v}\|}\right). \tag{3.8}$$

It is interesting to mention that these kernels can be stacked on top of each other in order to make successively more complex representations of the input data. Interestingly, this stacking is the infinite-dimensional equivalent of a multilayered neural network. The work done in Cho and Saul (2010) counts as an important inspiration for the work in this letter.

4 Recursive Kernels

4.1 Definition. To define a recursive kernel that is associated with an infinite recurrent network, we first mention that for any kernel function $k(\mathbf{u}, \mathbf{v})$ fulfilling the Mercer condition, there exists an implicit map $\mathbf{\Phi}$ such that $k(\mathbf{u}, \mathbf{v}) = \mathbf{\Phi}(\mathbf{u}) \cdot \mathbf{\Phi}(\mathbf{v})$. In our case, $\mathbf{\Phi}$ obviously corresponds to the infinite-dimensional hidden state. We now wish to define a kernel function that recursively operates on two discrete time series $\mathbf{x}(n)$ and $\mathbf{y}(n)$, $n \in \mathbb{Z}$. This means that the implicit map $\mathbf{\Phi}$ will have two arguments: the recursive map of the time series up to the current time step and the current sample in the time series. To avoid confusion, we use the symbol κ rather than k to denote recursive kernels throughout this letter. The recursive kernel function at the nth time step is then given by

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \mathbf{\Phi}(\mathbf{x}(n), \mathbf{\Phi}(\mathbf{x}(n-1), \mathbf{\Phi}(\cdots))) \cdot \mathbf{\Phi}(\mathbf{y}(n), \mathbf{\Phi}(\mathbf{y}(n-1), \mathbf{\Phi}(\cdots))), \tag{4.1}$$

where we abbreviated the left side of this equation as

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{x}(n), \mathbf{x}(n-1), \mathbf{x}(n-2), \ldots, \mathbf{y}(n), \mathbf{y}(n-1), \mathbf{y}(n-2), \ldots). \tag{4.2}$$

Often $\mathbf{\Phi}$ will map to an infinite-dimensional space, and it might seem strange that there is both a finite- and an infinite-dimensional argument. However, since the kernel functions usually depend on norms and dot products, which are well defined in both cases, this does not pose any difficulties for the actual mathematical derivation in specific cases, as we shall see.

To specify how we define a recursive kernel, we base ourselves again on neural networks. Notice that we can rewrite equation 2.2 as[3]

$$f(\mathbf{Wa}(t) + \mathbf{Vs}(t)) = f\left([\mathbf{W}|\mathbf{V}]\begin{bmatrix}\mathbf{a}(t)\\ \mathbf{s}(t)\end{bmatrix}\right), \tag{4.3}$$

which means that the "input" of the network in the current time step is the concatenation of the input signal and the previous hidden state. If we

[3]This way of thinking of recurrent neural networks can probably be attributed to Elman (1990).

extend this idea to our situation, we finally arrive at the main realization of this letter that makes the definition of a recursive kernel possible: the input vector of $\mathbf{\Phi}$ can be chosen to be simply the concatenation of the current input vector with the previous recursive mapping:

$$\mathbf{\Phi}\left(\mathbf{x}\left(n\right), \mathbf{\Phi}\left(\mathbf{x}(n-1), \mathbf{\Phi}\left(\cdots\right)\right)\right) = \mathbf{\Phi}([\mathbf{x}(n) \mid \mathbf{\Phi}\left([\mathbf{x}(n-1)|\mathbf{\Phi}(\cdots)]\right)]). \tag{4.4}$$

This will allow us to easily extend the definition of most common kernel functions into recursive equivalents. Especially, it is possible to find recursive versions of all kernel functions in which

$$k(\mathbf{u}, \mathbf{v}) = f\left(\|\mathbf{u} - \mathbf{v}\|^2\right) \tag{4.5}$$

or

$$k(\mathbf{u}, \mathbf{v}) = f\left(\mathbf{u} \cdot \mathbf{v}\right). \tag{4.6}$$

The first case can be worked out as follows. If **u** and **v** are concatenations of two vectors, that is, $\mathbf{u} = \left[\mathbf{u}_1|\mathbf{u}_2\right]$ and $\mathbf{v} = \left[\mathbf{v}_1|\mathbf{v}_2\right]$, we can write

$$k(\mathbf{u}, \mathbf{v}) = f\left(\|\mathbf{u}_1 - \mathbf{v}_1\|^2 + \|\mathbf{u}_2 - \mathbf{v}_2\|^2\right). \tag{4.7}$$

In our case, when $\mathbf{u}_1$ and $\mathbf{v}_1$ correspond to the current inputs $\mathbf{x}(n)$ and $\mathbf{y}(n)$, and $\mathbf{u}_2$ and $\mathbf{v}_2$ to the recursive maps, this becomes

$$\kappa_n(\mathbf{x}, \mathbf{y}) = f\left(\|\mathbf{x}(n) - \mathbf{y}(n)\|^2 + \kappa_{n-1}(\mathbf{x}, \mathbf{x}) + \kappa_{n-1}(\mathbf{y}, \mathbf{y}) - 2\kappa_{n-1}(\mathbf{x}, \mathbf{y})\right). \tag{4.8}$$

Equivalently we find that the second case leads to

$$\kappa_n(\mathbf{x}, \mathbf{y}) = f\left(\mathbf{x}(n) \cdot \mathbf{y}(n) \; + \kappa_{n-1}(\mathbf{x}, \mathbf{y})\right). \tag{4.9}$$

Note that we do not need to have an explicit infinite-dimensional form of the kernel to work out its recursive version. All that is necessary is the kernel function.

4.2 Examples. Here we will give a small number of examples of the recursive forms of some of the most commonly used kernel functions. Later we find that the parameter σ in the previously mentioned kernel functions is in fact the parameter that will determine the dynamics of the recursive kernels. However, we wish to define this parameter separately from the scaling of the data. Therefore, we will scale the two parts of the concatenated vector in equation 4.4 differently. We shall use σ for the infinite-dimensional state vector and σ_i for the current input.

We provide the following list of recursive kernels for reference throughout the rest of this letter:

- **Linear kernel.** The linear kernel has a trivial recursive extension: $k(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$, gives

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \sigma_i^2 \mathbf{x}(n) \cdot \mathbf{y}(n) + \sigma^2 \kappa_{n-1}(\mathbf{x}, \mathbf{y}). \tag{4.10}$$

  This kernel is especially useful as the linear approximation of the recursive arcsine kernel (see section 6.3). It is also obvious that the scaling term σ will have to be smaller than 1 to ensure asymptotic stability. Another property that can be observed clearly in this kernel is the fading memory property of the recursive kernels. We can write the kernel as

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \sigma_i^2 \sum_{j=0}^{\infty} \sigma^{2j} \mathbf{x}(n-j) \cdot \mathbf{y}(n-j), \tag{4.11}$$

  which clearly shows how the dependence on the previous input samples drops off exponentially as they lay further in the past.
- **Polynomial kernel.** As another example, the polynomial kernel $k(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^q$, gives

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \left[\sigma_i^2 \mathbf{x}(n) \cdot \mathbf{y}(n) + \sigma^2 \kappa_{n-1}(\mathbf{x}, \mathbf{y})\right]^q. \tag{4.12}$$

- **Gaussian kernel.** The gaussian kernel, which is very important in many applications, is given by $k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$. If we extend this to a recursive kernel function, we get

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}(n) - \mathbf{y}(n)\|^2}{2\sigma_i^2}\right) \exp\left(\frac{\kappa_{n-1}(\mathbf{x}, \mathbf{y}) - 1}{\sigma^2}\right). \tag{4.13}$$

  Notice that here we use a different definition for the two scaling parameters, more akin to the regular definition of kernel width.
- **Arcsine kernel.** More complicated kernels, like the arcsine kernel, require the recursive calculation of the kernels applied on the time series individually. The recursive version is given by

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \frac{2}{\pi} \arcsin\left(\frac{2\left(\sigma_i^2 \mathbf{x}(n) \cdot \mathbf{y}(n) + \sigma^2 \kappa_{n-1}(\mathbf{x}, \mathbf{y}) + \sigma_b^2\right)}{\sqrt{g_n(\mathbf{x}) g_n(\mathbf{y})}}\right), \tag{4.14}$$

  with

$$g_n(\mathbf{x}) = 1 + 2\left(\sigma_i^2 ||\mathbf{x}(n)||^2 + \sigma^2 \kappa_{n-1}(\mathbf{x}, \mathbf{x}) + \sigma_b^2\right), \tag{4.15}$$

and

$$\kappa_n(\mathbf{x},\mathbf{x}) = \frac{2}{\pi}\arcsin\left(1 - \frac{1}{g_n(\mathbf{x})}\right), \tag{4.16}$$

and similar for $g_n(\mathbf{y})\kappa_n(\mathbf{y},\mathbf{y})$.

- **Linear rectifier kernel.** The recursive extension of this kernel is as follows:

$$\kappa_{p,n}(\mathbf{x},\mathbf{y}) = \frac{1}{\pi}\left(h_n(\mathbf{x})h_n(\mathbf{y})\right)^{\frac{p}{2}} J_p(\theta_n), \tag{4.17}$$

with

$$h_n(\mathbf{x}) = \kappa_{p,n-1}(\mathbf{x},\mathbf{x}) + \|\mathbf{x}(n)\|^2, \tag{4.18}$$

$$\theta_n = \arccos\left(\frac{\kappa_{p,n-1}(\mathbf{x},\mathbf{x}) + \mathbf{x}(n)\cdot\mathbf{y}(n)}{\sqrt{h_n(\mathbf{x})h_n(\mathbf{y})}}\right), \tag{4.19}$$

and

$$\kappa_{p,n}(\mathbf{x},\mathbf{x}) = \frac{1}{\pi}\left(h_n(\mathbf{x})\right)^p J_p(0). \tag{4.20}$$

5 Stability Analysis of Recursive Kernels

Before we continue to practical examples, we need to do a stability analysis on the recursive kernels previously derived. More important, we will have to find a connection with the property of fading memory, as is known from RC to be important for good computational performance. In precise terms, when the input of both time series goes to 0, we want the recursive kernel to converge to a unique fixed point rather than behaving chaotically or going to a limit cycle. If possible, we also want to identify the speed (in terms of number of time steps) in which the kernel function would converge to this fixed point, because this will give an indication of the memory depth of the system.

Suppose $\kappa_0(\mathbf{x},\mathbf{y})$ is the output value of the kernel at $n = 0$, which can be any value within the range allowed by the kernel function. We now define the input series $\mathbf{x}(n)$ and $\mathbf{y}(n)$ to be equal to 0 for $n > 0$, such that their norms or dot products no longer change. The recursive kernel then reduces to an iterated function, which can be analyzed mathematically or visually with a cobweb diagram. We use the Banach fixed-point theorem to prove the existence of this fixed point for recursive gaussian kernels and recursive arcsine kernels.

5.1 Stability of the Recursive Gaussian RBF Kernel. Banach's fixed-point theorem (Banach, 1922) states that given X, a nonempty, complete metric space with a distance metric d, the contractive mapping f has one and only one fixed point c^* if there exists a number $0 < q < 1$ such that

$$d\big(f(a), f(b)\big) \leq q d(a, b), \tag{5.1}$$

with a and b any two elements from X. As a metric, we use $d(a, b) = |a - b|$. The space on which the kernel is defined is $[0, 1]$. We can then write the condition for the recursive gaussian kernel as

$$\left|\exp\left(\frac{a-1}{\sigma^2}\right) - \exp\left(\frac{b-1}{\sigma^2}\right)\right| \leq q\,|a-b|\,. \tag{5.2}$$

The function $\exp\left(\frac{a-1}{\sigma^2}\right)$ increases monotonically, so if we assume $a > b$, we can omit the absolute value operator. This allows us to rewrite the condition as

$$\exp\left(\frac{a-1}{\sigma^2}\right) - qa \leq \exp\left(\frac{b-1}{\sigma^2}\right) - qb. \tag{5.3}$$

This condition is automatically fulfilled if we can show that the function

$$\exp\left((a-1)/\sigma^2\right) - qa$$

decreases monotonically or that the derivative is nonpositive in $[0, 1]$:

$$\frac{1}{\sigma^2}\exp\left(\frac{a-1}{\sigma^2}\right) - q \leq 0. \tag{5.4}$$

The exponent in this equation is always smaller than or equal to 1. This means that as long as $\sigma > 1$, there always exists a $q < 1$ that fulfills this condition. If $\sigma < 1$, the condition will no longer hold for all $a \in [0, 1]$, and the fixed point at $a = 1$ will become unstable.

Clearly, for $\sigma > 1$, the stable fixed point is $a = 1$. We are interested in the speed of convergence asymptotically close to this fixed point, as this will give us an upper limit to how long a perturbation in the input still influences the kernel function. We find that if we linearize around $a = 1$, the distance between two orbits recedes with a factor $1/\sigma^2$. Associating this with an exponential decay time τ gives us a typical timescale for the kernel. Assuming $\kappa_n = \exp(-n/\tau)\kappa_0$, we get $\tau = \frac{1}{2\ln(\sigma)}$. This means that for $\sigma = 1$, the fading memory of the system goes to infinity, and the definition no longer applies when $\sigma < 1$.

Figure 3 shows a cobweb diagram of the iterated map for the three situations. For $\sigma < 1$, all orbits still converge to a fixed point, but there is an unstable zone in which orbits diverge. Notice that σ^{-1} is equivalent to the spectral radius ρ in ESNs. When $\rho < 1$, it will similarly dictate the speed at which the reservoir states converge to their fixed points. In the linear

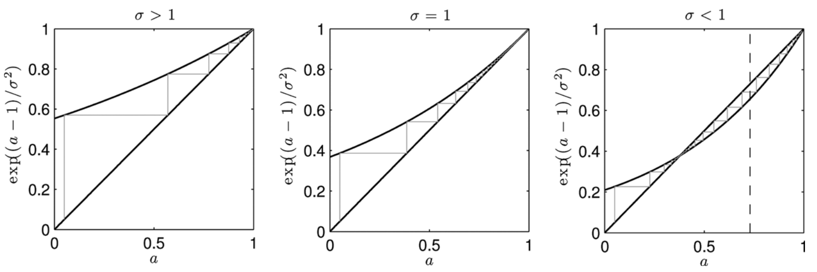

Figure 3: Cobweb plots for $\sigma > 1$, $\sigma = 1$, and $\sigma < 1$ associated with the recursive gaussian RBF kernel. The gray lines are example orbits that converge to a stable fixed point. The dashed line in the right panel shows the separation between the stable and unstable region. Left of the dashed line, all orbits will converge, but on the right, they will diverge until they reach the stable region.

approximation (asymptotically close to the fixed point around the origin), a spectral radius equal to 1 corresponds to the edge of stability. When the spectral radius is greater than 1, the states will converge to another fixed point, start to oscillate, or become chaotic. However, another stable region then exists when the states are pushed into the saturating parts of their nonlinearity, corresponding with the stable region around the second fixed point in the right panel of Figure 3.

5.2 Stability of the Recursive Arcsine Kernel. The case of the recursive arcsine kernel is more complicated, since each iteration will require mapping $\kappa_n(\mathbf{x}, \mathbf{x})$, $\kappa_n(\mathbf{y}, \mathbf{y})$, and $\kappa_n(\mathbf{x}, \mathbf{y})$, that is, the iterative function is a mapping of the form $\mathbb{R}^3 \to \mathbb{R}^3$. However, the iterative map $\kappa_{n+1}(\mathbf{x}, \mathbf{x}) = f(\kappa_n(\mathbf{x}, \mathbf{x}))$ does not depend on $\kappa_n(\mathbf{y}, \mathbf{y})$, and $\kappa_n(\mathbf{x}, \mathbf{y})$. Therefore, we first focus on the behavior of this map and further on look at the behavior of $\kappa_n(\mathbf{x}, \mathbf{y})$.

We again focus on the situation where $\mathbf{x}(n), \mathbf{y}(n) = 0$ for $n > 0$, and for simplicity, we assume $\sigma_b = 0$. We need to consider the case where the starting point of the recursion is in the range [0, 1], since the right side of equation 4.16 is positive and smaller than 1. The recursive formula can be written as

$$a \to \frac{2}{\pi} \arcsin\left(1 - \frac{1}{1 + 2a\sigma^2}\right). \tag{5.5}$$

The same line of reasoning as before applies: this function rises monotonically in [0, 1], so we need only to look at its derivative. The condition becomes

$$\frac{2}{\pi} \frac{2\sigma^2}{\sqrt{1 + 4a\sigma^2}(1 + 2a\sigma^2)} - q \leq 0. \tag{5.6}$$

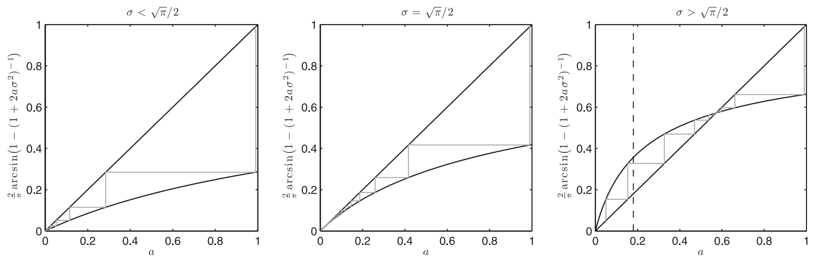

Figure 4: Cobweb plots for $\sigma < \sqrt{\pi}/2$, $\sigma = \sqrt{\pi}/2$, and $\sigma > \sqrt{\pi}/2$ associated with the recursive arcsine kernel. The gray lines are example orbits that converge to a stable fixed point. The dashed line in the right panel shows the separation between the stable and unstable region. On the right of the dashed line, all orbits will converge. On the left, they will diverge until they reach the stable region.

This expression reaches the highest value in its range for $a = 0$, which leads to the condition that

$$\sigma < \frac{\sqrt{\pi}}{2}. \tag{5.7}$$

If this condition holds, the fixed point will be $a = 0$.

We have now proved that for $\sigma < \sqrt{\pi}/2$, the contractive mapping determined by $\kappa_n(\mathbf{x}, \mathbf{x})$ will have a unique stable fixed point. To prove that this also applies to the contractive mapping of $\kappa_n(\mathbf{x}, \mathbf{y})$, we start by realizing that $\kappa_n(\mathbf{x}, \mathbf{y}) = \sqrt{\kappa_n(\mathbf{x}, \mathbf{x})\kappa_n(\mathbf{y}, \mathbf{y})}\cos(\theta)$, with θ the angle between the two infinite-dimensional mappings.[4] Because both $\kappa_n(\mathbf{x}, \mathbf{x})$ and $\kappa_n(\mathbf{y}, \mathbf{y})$ converge to 0, so will $\kappa_n(\mathbf{x}, \mathbf{y})$.

It seems that again, an edge of stability as in RC can be defined. This time it is in fact defined by the distribution of recursive weights for the infinite-dimensional neural network. An interesting remark is the fact that the value of $\sqrt{\pi}/2$ is the inverse of the amplification of the error function around the origin. If another sigmoid nonlinearity were to be used to define a recursive kernel, the edge of stability would be given by σ equal to the inverse of the slope round the origin.

In Figure 4, we show cobweb plots for the evolution of $\kappa_n(\mathbf{x}, \mathbf{x})$. We can see that $a = 0$ is a stable fixed point for as long as $\sigma < \sqrt{\pi}/2$ and becomes unstable for $\sigma > \sqrt{\pi}/2$ and another stable fixed-point forms.

[4]This equality derives from the property that for any two vectors $\mathbf{u}$ and $\mathbf{u}$, the inner product is given by $\mathbf{u} \cdot \mathbf{v} = \sqrt{||\mathbf{u}||^2||\mathbf{v}||^2}\cos\theta$. In the case of the kernel functions, the vectors are the infinite-dimensional mappings, and the inner product and norms are respectively given by $\kappa_n(\mathbf{x}, \mathbf{y})$, $\kappa_n(\mathbf{x}, \mathbf{x})$, and $\kappa_n(\mathbf{y}, \mathbf{y})$.

5.3 Spectral Radius. We have now analyzed the stability of the recursive kernels by considering them as iterated functions, but this is not directly related to ESNs. It is possible to formally link the concept of the spectral radius with the definition given by equation 3.2 and as such generalize the definition of spectral radius. We will do this for the case where the activation function $f(\mathbf{u}, \mathbf{w})$ is of the form $f(\mathbf{u} \cdot \mathbf{w})$.

To start, we estimate equation 3.2 by a Monte Carlo sampling,

$$k(\mathbf{u}, \mathbf{v}) \approx \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{u} \cdot \mathbf{w}_i) f(\mathbf{v} \cdot \mathbf{w}_i), \tag{5.8}$$

where each $\mathbf{w}_i$ is randomly drawn from the distribution $P(\mathbf{w})$. Notice that since we consider this an approximation of the dot product of two hidden state vectors, we can write $f(\mathbf{u} \cdot \mathbf{w}_i) = \tilde{a}_i^u$, which gives us

$$k(\mathbf{u}, \mathbf{v}) \approx \frac{1}{N} \sum_{i=1}^{N} \tilde{a}_i^u \tilde{a}_i^v = \sum_{i=1}^{N} \frac{\tilde{a}_i^u}{\sqrt{N}} \frac{\tilde{a}_i^v}{\sqrt{N}}. \tag{5.9}$$

This means we can define finite Monte Carlo approximations of the infinite-dimensional hidden states. If we wish to make a recurrent equivalent of these sampled Monte Carlo states, we get the following equation

$$\tilde{a}_i^u(t+1) = f\left(\frac{1}{\sqrt{N}} \sum_{j=1}^{N} \mathbf{W}_{ij} \tilde{a}_j^u(t) \right), \tag{5.10}$$

with $\mathbf{w}_{ij}$ signifying the jth element of the vector $\mathbf{w}_i$ or, more precisely, the element on the ith row and jth column of the matrix $\mathbf{W}$. Notice that this equation is nothing more than the update equation of a reservoir system. This means that reservoirs indeed can be considered as finite approximations of infinite-sized kernels.

The spectral radius of the connection matrix of this system is given by

$$\rho = \frac{\rho(\mathbf{W})}{\sqrt{N}}, \tag{5.11}$$

with $\rho(\mathbf{W})$ the spectral radius of $\mathbf{W}$. Let us now assume that $P(\mathbf{w}) = \prod_i P(w_i)$, that is, all elements are drawn independently and from the same distribution, and, furthermore, we assume the distribution has 0 mean and variance σ^2. In that case, it can be proved (Geman, 1986) that

$$\lim_{N\to\infty} \frac{\rho(\mathbf{W})}{\sqrt{N}} \leq \sigma, \tag{5.12}$$

such that we find that the corresponding spectral radius for infinite-sized neural networks is $\rho \leq \sigma$.

This result confirms our earlier finding that σ is the parameter in recursive arcsine kernels that corresponds to the spectral radius (apart from the factor $\sqrt{\pi}/2$, which comes from the slope of the error function).

6 Recurrent Kernel Machines for Applications

6.1 SVMs. Before considering tasks, we first explain how we apply recursive kernels in SVMs. For the first two tasks we considered, we decided to use least squares support vector machines (LS-SVM) (Suykens, Van Gestel, De Brabanter, De Moor, & Vandewalle, 2002). Contrary to the more common SVM, the LS-SVM uses a quadratic loss function instead of the hinge loss and is conceptually easier because training is reduced to finding the solution to a system of linear equations. Essentially there are four main reasons we chose LS-SVMs rather than SVMs:

- Two of the three tasks we considered are regression tasks, in which the error metric to evaluate performance is the quadratic error, and this is the error metric LS-SVMs minimize in the first place with ridge regression in the dual space. For classification tasks, normal SVMs would be the more natural choice.
- We attempted to use LIBSVM for the NARMA task (specified later), as it allows working with precalculated kernels.[5] We examined the two variants for regression problems—ν-SVR and ϵ-SVR—but neither gave a performance that came even close to that of LS-SVMs. Likely this is due to the L_1-norm optimization, which is apparently unsuited for this task.
- Reservoir computing also uses a quadratic loss function for training the output weights. This allowed us to compare the performance of recurrent neural networks in an RC context to their kernel machine equivalents and truly consider them to be the dual version of the normal reservoir training algorithm for.
- We also consider a classification task for which normal SVMs would seem the more natural choice. However, because we have to deal with a very large data set, common SVMs render impractical, and we will use the Newton-Raphson approximation of a quadratic hinge-loss function (more details are in section 6.5). Although not strictly LS-SVMs, their resulting systems of equations are equivalent.

An LS-SVM for regression operates as follows. Given a training set of N input features $\mathbf{x}_i$ with corresponding output targets y_i, the system outputs

[5]LIBSVM is available online from http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

a value $\tilde{y}(\mathbf{x})$ for an input vector $\mathbf{x}$ defined as

$$\tilde{y}(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + \beta, \tag{6.1}$$

where the parameters α_i and β are found by solving the system

$$\begin{bmatrix} 0 & \mathbf{1}_N^{\mathsf{T}} \\ \mathbf{1}_N & \mathbf{K} + \lambda \mathbf{I} \end{bmatrix} \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix}, \tag{6.2}$$

with $\mathbf{1}_N = [1, \ldots, 1]$, $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{y} = [y_1, \ldots, y_N]$. A scaled unity matrix is added to the Gramm matrix for regularization, with regularization parameter λ. For recursive kernels, we redefine the SVM as

$$\tilde{y}(t) = \sum_{i=1}^{N} \alpha_i \kappa_t(\mathbf{x}(t : t - \infty), \mathbf{x}_i(0 : -\infty)) + b, \tag{6.3}$$

in which $\mathbf{x}(t : t - \infty)$ denotes the full history of the input signal up to a time t and $\mathbf{x}_i(0 : -\infty)$ is the ith support vector, in this case also an infinitely long time series. Obviously the recursion will need to be cut off at a certain point for practical reasons, and we need to specify a maximum recursion depth τ. For this we use the following criterion:

$$\kappa_0(\mathbf{x}_i(0 : -\tau), \mathbf{x}_j(0 : -\tau)) \approx \kappa_0(\mathbf{x}_i(0 : -\infty), \mathbf{x}_j(0 : -\infty)); \tag{6.4}$$

that is, there should be only a small relative difference between the "correct" value (with an infinite recursion depth) and the practically attainable value. This is easily attainable by choosing a recursion depth that is much larger than the typical timescale $\tau = -\frac{1}{2\ln(\rho)}$. However, in reality, it seems that this criterion is too strict, and shorter recursion depths already give reasonable estimates for the asymptotic values. Finally, this leads to the following approximation for SVMs using recursive kernels

$$\tilde{y}(t) = \sum_{i=1}^{N} \alpha_i \kappa_t(\mathbf{x}(t : t - \tau), \mathbf{x}_i(0 : -\tau)) + \beta. \tag{6.5}$$

6.2 Relation Between ESNs and Recurrent Kernel Machines. Here we explain the relation between a classic ESN readout layer and the LS-SVM with recurrent kernels. For this, it is interesting to first consider the case where we would train a finite ESN the same way we train an LS-SVM. We select a set of time series $\mathbf{x}_i(0 : -\tau)$ as support vectors. Calculating the associated kernel function between two time series can now be done explicitly as an inner product of the hidden states. This has two implications. First, there is no point in storing the support vectors as time series because the hidden state of the final time step is known explicitly. If $\mathbf{a}_i$ is the last hidden state caused by time series $\mathbf{x}_i(0 : -\tau)$ and $\mathbf{a}(t)$ is the hidden state caused by $\mathbf{x}(t : -\infty)$, the kernel function between these two is nothing but $\mathbf{a}_i \cdot \mathbf{a}(t)$. Second, at any time t, the output of the system can be written as

$$\begin{aligned}\tilde{y}(t) &= \sum_{i=1}^{N} \alpha_i \mathbf{a}_i \cdot \mathbf{a}(t) + \beta \\ &= \underbrace{\alpha^{\mathsf{T}} \mathbf{A}^{\mathsf{T}}}_{\mathbf{u}^{\mathsf{T}}} \mathbf{a}(t) + \beta \\ &= \mathbf{u} \cdot \mathbf{a}(t) + \beta,\end{aligned}$$

where $\mathbf{A} = \left[\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N\right]$, the concatenation of the hidden states of the support vectors. Essentially this show that in the case of a finite ESN, a kernel machine would simply lead to a single linear readout layer as in normal ESN training.

It is possible to show that the readout weights $\mathbf{u}$ and bias β obtained in this matter are exactly the same readout weights that would be obtained by constructing the linear readout weights in the classic way by ridge regression and using the support vectors as training data. This stems from the fact that both training methods solve the same system of equations. Where classic reservoir training does this directly (in primal space), the LS-SVM method does this via the dual representation of the problem. (For more details, see chapter 3 in Suykens et al., 2002, or chapter 6 in Bishop, 2006.) This result stays valid for arbitrarily large network sizes, which shows explicitly that LS-SVMs using recurrent kernels are truly the equivalent of infinite ESNs trained with a finite training set.

6.3 Fading Memory

6.3.1 Memory Function and Memory Capacity. The first property we consider is the so-called memory capacity. This task is fully academic and is meant as a way to study fading memory in RC by investigating how well a certain input signal can be linearly reproduced after a delay. Assume $s(t)$ to be an independent and identically distributed (i.i.d.) variable, drawn from some distribution. Formally, one defines a memory function (Jaeger, 2001) as

$$m(k) = \frac{\operatorname{cov}\left(s(t-k), \tilde{s}_k(t)\right)^2}{\operatorname{var}(s(t-k))\operatorname{var}(\tilde{s}_k(t))}, \tag{6.6}$$

with $\tilde{s}_k(t)$ the optimal linear reconstruction of the signal $s(t-k)$. The memory function $m(k)$ is a number between 0 and 1 and essentially indicates the time window of the past that is visible to the network.

To quantify the total memory present in a network, one defines the memory capacity $M = \sum_{k=0}^{\infty} m(k)$. It is well known that this number is bounded by the number of neurons N in the reservoir (Ganguli, Huh, & Sompolinsky, 2008; Jaeger, 2001). A linear reservoir has the best possible memory, whereas any nonlinearity necessarily has a reducing effect (Ganguli et al., 2008; Jaeger, 2001).

For infinite ESNs, the memory capacity will obviously not be limited by the number of neurons. However, any usable kernel machine will have only a training set, which is limited in size. As kernels are defined on time series, training data in our case will consist of a set of (potentially infinitely long) time series. An SVM that would reconstruct the signal from k time steps ago will use a construction as follows:

$$\tilde{s}_k(t) = \sum_{i=1}^{N} \alpha_i^{(k)} \kappa_t(s(t:-\infty), z_i), \tag{6.7}$$

where $s(t:-\infty)$ is the input time series up to a time t, z_i are (potentially infinitely long) time series that serve as support vectors, N is the total number of support vectors, and $\alpha_i^{(k)}$ are optimal weights for reconstructing the signal from k time steps ago. Notice that we omitted the output bias term β. This is justified, as we will assume 0 mean for the input signal, which does not change the overall conclusion of this section. Also notice that in this setup, we wish to optimize covariance, which is equivalent to minimizing the mean square error and, hence, using a quadratic loss function is fully justified.

6.3.2 Linear Approximation. It is possible to show that in the case of a linear kernel (as defined in section 4.2), memory capacity is equal to the number of support vectors. Notice that if we linearize the equation for the recursive arcsine kernel around $\mathbf{x}, \mathbf{y} \approx 0$, assuming no bias we in fact end up with the linear kernel. To see this, we start by examining equation 4.15. Since we assume that the time series $\mathbf{x}$ and $\mathbf{y}$ are infinitesimally small, both $||\mathbf{x}||^2$ and $\kappa_{n-1}(\mathbf{x}, \mathbf{x})$ will be close to 0, and $g_n(\mathbf{x}) \approx 1$. The first-order approximation of the arcsine function around 0 is given by $\arcsin(z) \approx z$, such that we end up with

$$\kappa_n(\mathbf{x}, \mathbf{y}) = \frac{2}{\pi}\left(2\sigma_i^2 \mathbf{x}(n) \cdot \mathbf{y}(n) + 2\sigma^2 \kappa_{n-1}(\mathbf{x}, \mathbf{y})\right). \tag{6.8}$$

Using the same reasoning as in the example of the linear kernel in section 4.2, we can write this as an infinite sum. For simplicity, we take $\sigma_i = \sigma$ and end up with

$$\kappa_n(\mathbf{x},\mathbf{y}) = \sum_{i=0}^{\infty} \underbrace{\left(\frac{4\sigma^2}{\pi}\right)}_{\gamma}^{i+1} x(n-i)y(n-i)$$

$$= \gamma \sum_{i=0}^{\infty} \gamma^i x(n-i)y(n-i),$$

with γ acting as the square of the spectral radius of the system. We now need to determine the $\alpha_i^{(k)}$, which minimize the mean squared error (in which we assume the regularization parameter $\lambda = 0$):

$$E_k = \left\langle \left[\gamma \sum_{i=1}^{N} \alpha_i^{(k)} \sum_{j=0}^{\infty} \gamma^j z_i(-j)s(t-j) - s(t-k) \right]^2 \right\rangle_t . \tag{6.9}$$

Notice that we can write the first term under the brackets as

$$\gamma \sum_{i=1}^{N} \alpha_i^{(k)} \sum_{j=0}^{\infty} \gamma^j z_i(-j)s(t-j) = \gamma \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \sum_{j=0}^{\infty} \gamma^j \mathbf{z_j} s(t-j),$$

in which $\boldsymbol{\alpha}_\mathbf{k}$ is a column vector associated with elements $\alpha_i^{(k)}$, and $\mathbf{z_j}$ is a column vector with elements $z_i(-j)$. We can now write

$$E_k = \gamma^2 \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \left(\sum_{j=0}^{\infty} \sum_{l=0}^{\infty} \gamma^{j+l} \mathbf{z_j} \mathbf{z_l}{}^\mathsf{T} \left\langle s(t-j)s(t-l) \right\rangle_t \right) \boldsymbol{\alpha}_\mathbf{k}$$

$$-2\gamma \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \sum_{j=0}^{\infty} \mathbf{z_j} \left\langle s(t-j)s(t-k) \right\rangle_t + \left\langle s(t-k)^2 \right\rangle_t .$$

If we now use the fact that the signal values $s(t)$ are i.i.d. such that $\left\langle s(t_1), s(t_2) \right\rangle_t = \varsigma^2 \delta_{t_1,t_2}$, with ς^2 equal to signal variance, we can reduce this expression further to

$$E_k = \varsigma^2 \gamma^2 \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \underbrace{\left(\sum_{j=0}^{\infty} \gamma^{2j} \mathbf{z_j} \mathbf{z_j}{}^\mathsf{T} \right)}_{\mathbf{Z}} \boldsymbol{\alpha}_\mathbf{k} - 2\varsigma^2 \gamma^{k+1} \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \mathbf{z_k} + \varsigma^2$$

$$= \varsigma^2 \left[\gamma^2 \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \mathbf{Z} \boldsymbol{\alpha}_\mathbf{k} - 2\gamma^{k+1} \boldsymbol{\alpha}_\mathbf{k}{}^\mathsf{T} \mathbf{z_k} + 1 \right].$$

After deriving E with regard to the elements of $\boldsymbol{\alpha}_\mathbf{k}$, we find that the optimal weights are given by

$$\boldsymbol{\alpha}_\mathbf{k} = \gamma^{k-1}\mathbf{Z}^{-1}\mathbf{z}_\mathbf{k}.$$

Next, we can insert this result in equation 6.6. We find that

$$\operatorname{cov}\left(s(t-k), \tilde{s}_k(t)\right) = \operatorname{var}\left(\tilde{s}_k(t)\right) = \gamma^{2k}\varsigma^2 {\mathbf{z}_\mathbf{k}}^\mathsf{T}\mathbf{Z}^{-1}\mathbf{z}_\mathbf{k},$$

which leads to

$$m(k) = \gamma^{2k}{\mathbf{z}_\mathbf{k}}^\mathsf{T}\mathbf{Z}^{-1}\mathbf{z}_\mathbf{k}.$$

Summation over k then gives us the memory capacity:

$$M = \sum_{k=0}^{\infty} \gamma^{2k}{\mathbf{z}_\mathbf{k}}^\mathsf{T}\mathbf{Z}^{-1}\mathbf{z}_\mathbf{k} \tag{6.10}$$

$$= \operatorname{tr}\left(\mathbf{Z}^{-1}\sum_{k=0}^{\infty}\gamma^{2k}\mathbf{z}_\mathbf{k}{\mathbf{z}_\mathbf{k}}^\mathsf{T}\right) \tag{6.11}$$

$$= \operatorname{tr}(\mathbf{I}) = N, \tag{6.12}$$

with tr(...) indicating the trace of the matrix.

This result is interesting as it highlights the equivalence of the number of support vectors with the number of nodes. ESNs have their hidden state as degrees of freedom. The fact that memory capacity is fundamentally limited by the number of nodes is a reflection of this fact: the total amount of "information" (not used in the information-theoretical sense) that can be coded into the states of n hidden nodes is equal to n. For recursive SVMs, this role is taken over by the number of support vectors. Each support vector will correspond to a single "node" in an SVM. Obviously the advantage of SVMs in this case is the fact that the support vectors are not random but rather contain meaningful information of the distribution of the input data.

6.4 Nonlinear Autoregressive Moving Average. The second task we consider is the so called NARMA (nonlinear auto regressive moving average) task, which has been used for benchmarking in many papers that consider time-series processing (Atiya & Parlos, 2000; Jaeger, 2003; Steil, 2005). The task is a single-input, single-output system, with input $u(t)$, i.i.d. numbers drawn from a uniform distribution between 0 and 0.5. The desired output $y(t)$ is then constructed as

$$y(t+1) = 0.3y(t) + 0.05y(t)\sum_{i=0}^{9} y(t-i) + 1.5u(t)u(t-9) + 0.1. \quad (6.13)$$

As the error metric to evaluate performance on this task, we used the *normalized root mean square error* (NRMSE), defined as

$$\text{NRMSE} = \sqrt{\frac{\langle y(t) - \tilde{y}(t)\rangle_t^2}{\text{var}(y(t))}}, \quad (6.14)$$

in which $\tilde{y}(t)$ is the output of the trained system.

6.4.1 Experiments and Results. To compare different effects and results, we performed four experiments:

- We used a classic windowed gaussian RBF kernel as a reference value to compare our results against. We optimized both the window length and kernel width by a 2-dimensional grid search. Using a validation set, we found the optimal window length to be 27 frames and the kernel width $\sigma = 5$, although performance does not change much for a relatively broad range around this optimal value σ.
- We measured the performance of the recursive gaussian RBF kernel in relation to its corresponding spectral radius $\rho = \sigma^{-1}$. We limited the recursion depth to 50 frames, although a shorter time would likely give very similar results.
- We did the same for the arcsine kernel in relation to its corresponding spectral radius $\rho = \frac{2}{\sqrt{\pi}}\sigma$. We again used a recursion depth of 50 frames.
- Because arcsine kernels are strongly related to ESNs, we used the opportunity to compare their performances. We also measured the performance of ESNs with error function nonlinearities for an increasing number of nodes and in relation to the corresponding spectral radius.

In all of the experiments, we used a training set of 500 frames, a validation set of 2000 frames used to determine the optimal regularization parameter, and a test set of 5000 frames. For the recursive kernels and the ESNs, performance in relation to input scaling has a broad, shallow optimum (data not shown), but nevertheless the scaling factors were optimized by a grid search at a corresponding spectral radius of 0.9, leading to $\sigma_i = 0.1$ for the arcsine kernels and the ESNs, and $\sigma_i = 0.4$ for the recursive gaussian RBF. For the arcsine kernels and ESNs, bias did not seem to improve performance and was therefore set to 0. All results were found by averaging over 100 different trials with newly generated data or reservoirs, or both.

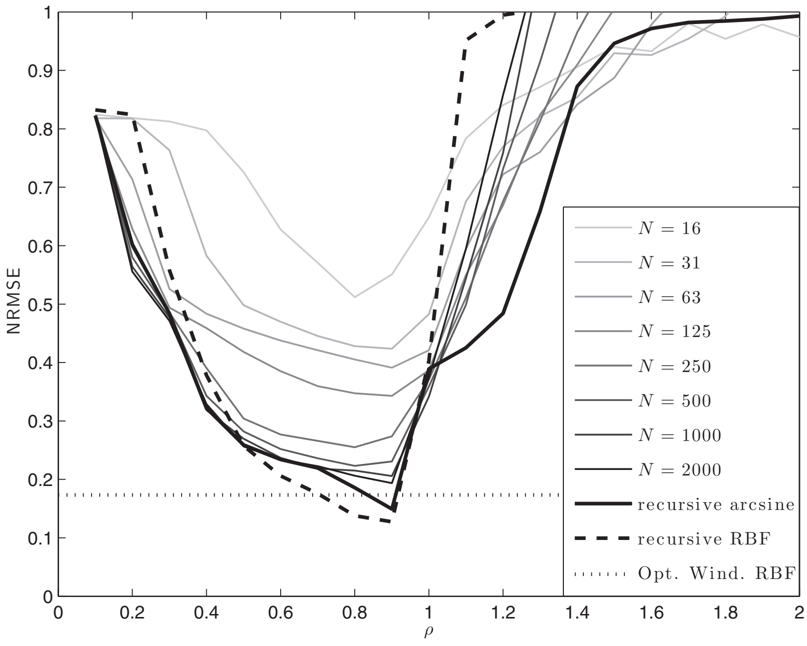

Figure 5: Mean NRMSE of the NARMA task for different setups in relation to the corresponding spectral radius ρ. The thin light to dark gray lines are NRMSEs for ESNs with increasing numbers of nodes N (specified in the legend). The thick black line is for the arcsine kernel, that is, $N \to \infty$. The dashed line is the performance of the recursive gaussian RBF kernel, and the dotted line (independent of ρ) is the mean NRMSE for optimized windowed gaussian RBF kernels.

Results of the experiments are shown in Figure 5. Optimal performance can be found around $\rho = 0.9$. Performance of the ESNs gradually increases with the number of nodes, converging slowly to the performance of the arcsine kernel. As ρ becomes greater than 1, performance rapidly deteriorates. The recursive gaussian RBF kernel performs best, and both recursive kernels perform better than the classic time window RBF kernel.

6.5 Phoneme Recognition. The second task we consider is a speech recognition task in which the goal is to classify phonemes, the smallest segmental unit of sound employed to form meaningful contrasts between utterances. We use the internationally renowned TIMIT speech corpus (Garofolo et al., 1993), which consists of 5040 English spoken sentences from 630 different speakers representing eight dialect groups. About 70% of the speakers are male, and 30% are female.

The speech is labeled by hand for each of the 61 existing phonemes, which was reduced to 39 symbols as proposed by Lee and Hon (1989). The TIMIT corpus has a predefined train and test set with different speakers. The speech has been preprocessed using Mel frequency cepstral coefficient (MFCC) analysis (Davis & Mermelstein, 1980), which is performed on 25 ms Hamming windowed speech frames, and subsequent speech frames are shifted over 10 ms with respect to each other. Each frame contains a 39-dimensional feature vector, consisting of the log energy and the first 12 MFCC coefficients and their first and second derivatives (the so-called Δ and $\Delta\Delta$ parameters).

6.5.1 One-Versus-One Classifiers. In order to classify each frame into one of the 39 possible classes, we use a voting system that starts from a set of 741 one-versus-one classifiers. Each of these classifiers is trained to distinguish between two specific phonemes, and because there are 39 classes, there are $(39 \times 38)/2 = 741$ one-versus-one classifiers. Each one-versus-one classifier is trained only on data labeled with its corresponding phonemes and outputs either 1 or -1 (the sign of the output value). Final classification is performed by letting the classifiers each cast a vote.

6.5.2 Training Method. One of the difficulties of using SVMs to train on TIMIT is the fact that the data set is very large. The training set consists of 1,124,823 frames and the test set of 353,390 frames. The number of frames per one-versus-one classifier is of the order 10^4 to 10^5. Traditional SVM methods for classification would run into practical computational problems for such large data sets.

Various methods for handling large data sets exist. We use a technique based on Newton optimization (Chapelle, 2007). Explaining the algorithm in detail would go beyond the scope of this letter so we explain it only briefly and refer to Botton, Chappelle, DeCoste, and Weston (2007, chap. 2) for details.

Classic SVMs use a hinge loss function. Optimizing this system is a convex problem with a unique solution, and therefore is solvable with quadratic programming techniques. An LS-SVM has the advantage that the solution can be found by solving a single system of linear equations. There are, however, two strong downsides of using a quadratic loss function for classification problems. First, a quadratic error will assign a high loss to some vectors that are in fact classified correctly. Second, all the data in the training set will serve as support vectors, making the end solution nonsparse. To solve this problem, we use a loss function of the form $\max(0, 1 - y_i\tilde{y}_i)^2$, with y_i the target (-1 or 1), and $\tilde{y}_i$ the output of the SVM. Essentially this is a quadratic hinge loss function. If we optimize this system using the Newton-Raphson method, this comes down to solving the problem using a quadratic loss function and next selecting data points with $y_i\tilde{y}_i < 1$ as support vectors. This process is repeated until the set of support vectors no longer changes. Next, one can train on successively larger data sets and use the previously found set of support vectors as initial values.

We trained each one-versus-one classifier on a subset of 10^4 samples and chose a separate validation set of 2000 samples, both randomly drawn from the total training data set associated with the corresponding labels. If the total number of samples in the set was smaller than 12,000, we randomly drew 1000 samples as a validation set and used the rest for training.

6.5.3 Subsampling and Parameter Optimization. We tested on classic windowed gaussian RBF kernels, recursive gaussian RBF kernels, and arcsine kernels. For both recursive kernels, we also investigated the effect of subsampling the MFCC data. We found in Triefenbach et al. (2010) that large recurrent neural networks perform better on phone recognition if the nodes are leaky integrators, that is, when the effective timescale of the network dynamics is slowed down. Rather than incorporating this into our kernels, we subsampled the data by a factor of 2, 3, and 5. This essentially means that we speed up the data rather than slow down the dynamics of our system. For the nonsubsampled variants of the data, we classify on the third frame of the time window or recursion depth, that is, the SVM needs to classify the phoneme of two frames in the past. For the subsampled versions, we classify on the second frame (effectively the third, fourth, and sixth frames in the respective unsubsampled data sets). Rather than optimizing the parameters for each one-versus-one classifier, we looked for globally optimal parameters by randomly selecting 250 from the 820 classifiers and trained them on a small training, validation, and test set of 1000 samples each, drawn randomly from the corresponding full training set and measured the average test error over a relevant range of parameters. The window size of the gaussian RBF kernel was determined this way. Recursion depths of the recursive kernels were determined by making sure the kernel value differed on average less than 1% from its asymptotic value. We chose a bias equal to 0 for the arcsine kernels to reduce the number of parameters to optimize.

6.5.4 Results. Typically the performance on the TIMIT data set is evaluated based on the phoneme error rate. However, this requires an additional mechanism such as a hidden Markov model to segment the frames into groups corresponding to phonemes. Because we are interested only in the relative performance of the kernels, we limited ourselves to measuring only the frame error rate (FER), that is, the percentage of input windows classified incorrectly. The results (FER), average number of support vectors per one-versus-one classifier ($\overline{N_{sv}}$), window size or recursion depth (τ), and optimal parameters σ and σ_i as defined in section 4.2 for each variant are shown in Table 1. FER for the subsampled versions of the test set were determined by labeling the missing frames with the nearest classified frame in the case of subsampling $3\times$ and $5\times$. In the case of $2\times$ subsampling, the

Table 1: Results on TIMIT.

	FER	$\overline{N_{sv}}$	τ	σ	σ_i
Windowed RBF	31.5%	1465	9	16	
Recursive RBF	30.6%	1386	10	1	22
Recursive RBF 2× subsampling	29.4%	1499	10	1	12.5
Recursive RBF 3× subsampling	28.7%	1504	10	1	16.5
Recursive RBF 5× subsampling	28.5%	1100	5	0.8	8
Arcsine	30.5%	1105	15	$1.75\frac{\sqrt{\pi}}{2}$	0.026
Arcsine 2× subsampling	29.3%	1511	8	$2.25\frac{\sqrt{\pi}}{2}$	0.035
Arcsine 3× subsampling	28.6%	1377	8	$2\frac{\sqrt{\pi}}{2}$	0.04
Arcsine 5× subsampling	28.9%	1210	8	$2\frac{\sqrt{\pi}}{2}$	0.045
Results from the literature					
HMM (Cheng, Sha, & Saul, 2009)	39.3%				
PA (Crammer, 2011)	30.0%				
DROP (Crammer, 2011)	29.2%				
PAC-Bayes 1-frame (Kesher, McAllester, & Hazan, 2011)	27.7%				
PAC-Bayes 9-frame (Kesher et al., 2011)	26.5%				
Online LM-HMM (Cheng et al., 2009)	25.0%				

FER was calculated twice by using the classification of both the previous and next frame as labels for the missing frames, and we took the average of both FERs. The fact that most literature does not mention FER makes it hard to compare our results to the state of the art, but some papers actually do mention FER, and to give some idea of our performance, we have included some representative results in the table.

All the techniques with recursive kernels outperform the classical windowed gaussian RBF kernels, even without subsampling, and it is obvious that subsampling gives a boost in performance. Remarkably, we found that the optimal spectral radius of the arcsine kernels is greater than 1. On examining the necessary recursion depth, we found that these kernels do indeed depend on only a finite history of the input time series. This is due to the relatively high variance of the input, which pushes the kernels into the saturating part of their nonlinearity.

It is interesting to note that in the case of the unsubsampled data set, we find that the number of support vectors is lower for the recursive kernels than for the windowed kernels. This seems to suggest that the recursive kernels are better at capturing the inherent structure of the speech data.[6]

[6]This comparison would be unfair for the subsampled data sets as these are smaller.

In Triefenbach et al. (2010), the same task was studied by (among other techniques) using a very large reservoir of 20,000 nodes. The FER found for this setup was 29.1% (FER is not mentioned in the paper, but we know this from personal communication with the authors), which is comparable to our own results.

7 Conclusion

In this letter, we have described a straightforward method to define a kernel function that is associated with a recurrent neural network with an infinite number of hidden nodes. We link this result with findings that have been made in the domain of reservoir computing, which employs large, randomly initiated neural networks. In our case (with analog sigmoid nodes), such networks are commonly called echo state networks. Infinite-sized neural networks can be considered as ESNs without a random factor and determined solely by the data and a small number of parameters.

It is possible to associate the parameters of the recursive kernel functions with properties known to play an important role in the dynamics of ESNs. Specifically, it is possible to define a spectral radius for recursive kernels, which determines the dynamical regime of both recurrent networks and recursive kernels. A second important parameter in reservoir computing, defined as the memory capacity, which is fundamentally limited by the number of nodes, has an equivalent for recursive kernels. For these, memory capacity is fundamentally limited by the number of support vectors.

We tested the performance of recursive kernels on two benchmarks in which relevant information is specifically coded in time. We found that for the tasks investigated, recursive kernels perform better than classical gaussian RBF kernels operating on a sliding time window of the input signal. This result suggests that the recursive nonlinearity and the fading memory of the recursive kernels are better suited to capture the temporal nature of the data than an artificial time window.

One of the main arguments we wish to convey in this letter is that there is a direct link between RC and kernel machines and that it is possible to view reservoirs as primal space approximations of an infinite-sized recursive kernel. More precisely, the random weights can be considered as Monte Carlo samples from a continuous distribution. Indeed, we found that the performance of reservoirs in the function of network size asymptotically approaches that of the recursive kernels (using the same number of training data).

Many potential directions for future work remain. One of the most interesting questions is why fading memory seems to work better than time windows for certain tasks. One potential explanation is that the fading dependence on input history is a more natural representation of the dependencies required for the task than an artificially cut-off time window.

Hence, one can argue that a fading memory acts as a natural regularizer on time series processing tasks. Interestingly, we also tried to apply recursive kernels on synthetic time series prediction and generation (where the input was defined by a differential equation), but we found that windowed kernels perform better than or as well as recursive kernels. Here, it seems a windowed approach is still preferable to the fading memory approach. This can partially be explained by the Takens embedding theorem (Takens, Rand, & Young, 1981), which states that all information for integrating an nth order differential equation is embedded in a time window of $2n + 1$ frames. Another interesting line of research would be trying to construct principle component approximations of the feature space of the recursive kernels (known as the Nyström approximation). This allows making a finite approximation, conforming to classical reservoirs, but which depends on the underlying structure of the data.

Acknowledgments

This work was partially funded by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) and the European FP7 project ORGANIC.

References

Antonelo, E. A., Schrauwen, B., & Stroobandt, D. (2008). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, *21*, 862–871.

Atiya, A. F., & Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, *11*, 697–709.

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. Math.*, *3*, 133–181.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (pp. 144–152). New York: ACM.

Botton, L., Chappelle, O., DeCoste, D., & Weston, J. (2007). *Large scale kernel machines*. Cambridge, MA: MIT Press.

Boyd, S., & Chua, L. O. (1985). Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Transactions on Circuits and Systems*, *32*(11), 1150–1171.

Chapelle, O. (2007). Training a support vector machine in the primal. In L. Bottou, O. Chappelle, D. DeCoste, & J. Weston (Eds.), *Large-scale kernel machines* (pp. 29–51). Cambridge, MA: MIT Press.

Cheng, C.-C., Sha, F., & Saul, L. (2009). A fast online algorithm for large margin training of online continuous density hidden Markov models. In *Proceedings of Interspeech 2009* (pp. 668–671). Piscataway, NJ: IEEE.

Cho, Y., & Saul, L. K. (2010). Large margin classification in infinite neural networks. *Neural Computation*, *22*(10), 2678–2697.

Crammer, K. (2011). Efficient online learning with individual learning-rates for phoneme sequence recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. Piscataway, NJ: IEEE.

Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *28*, 357–366.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Fernando, C., & Sojakka, S. (2003). Pattern recognition in a bucket. In *Proceedings of the 7th European Conference on Artificial Life* (pp. 588–597). New York: Springer.

Ganguli, S., Huh, D., & Sompolinsky, H. (2008). Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, *105*(48), 18970–18975.

Garofolo, J., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., Dahlgren, N. L., et al. (1993). *TIMIT acoustic-phonetic continuous speech corpus.* Philadelphia: Linguistic Data Consortium.

Geman, S. (1986). The spectral radius of large random matrices. *Annals of Probability*, *14*(4), 1318–1328.

Jaeger, H. (2001). *The "echo state" approach to analysing and training recurrent neural networks* (Tech. Rep. no. 148). St. Augustin: German National Research Center for Information Technology.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrün, & K. Obermayer (Eds.), *Advances in neural information processing systems, 15* (pp. 593–600). Cambridge, MA: MIT Press.

Jaeger, H., & Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless telecommunication. *Science*, *308*, 78–80.

Jaeger, H., Lukosevicius, M., & Popovici, D. (2007). Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, *20*, 335–352.

Jones, B., Stekel, D., Rowe, J., & Fernando, C. (2007). Is there a liquid state machine in the bacterium *Escherichia coli*? In *IEEE Symposium on Artificial Life* (pp. 187–191). Piscataway, NJ: IEEE.

Kesher, J., McAllester, D., & Hazan, T. (2011). Pac-Bayesian approach for minimization of phoneme error rate. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*. Piscataway, NJ: IEEE.

Lee, K.-F., & Hon, H.-W. (1989). Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, *37*(11), 1641–1648.

Lukosevicius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149.

Maass, W., Joshi, P., & Sontag, E. D. (2007). Computational aspects of feedback in neural circuits. *PLOS Computational Biology, 3*(1), e165, 1–20.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Neal, R. M. (1996). *Bayesian learning for neural networks*. Berlin: Springer.

Pearlmutter, B. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228.

Rasmussen, C. E., & Williams, C.K.I. (2006). *Gaussian processes for machine learning*. Cambridge, MA: MIT Press.

Rumelhart, D., Hinton, G., & Williams, R. (1986). *Learning internal representations by error propagation*. Cambridge, MA: MIT Press.

Schäfer, A. M., & Zimmerman, H. G. (2006). Recurrent neural networks are universal approximators. *Lecture Notes in Computer Science*, *4131*, 632–640.

Schmidhuber, J., Wierstra, D., Gagliolo, M., & Gomez, F. (2007). Training recurrent networks by Evolino. *Neural Computation*, *19*, 757–779.

Shi, Z., & Han, M. (2007). Support vector echo-state machine for chaotic time-series prediction. *IEEE Transactions on Neural Networks*, *18*(2), 359–372.

Skowronski, M. D., & Harris, J. G. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, *20*(3), 414–423.

Steil, J. J. (2005). Stability of backpropagation-decorrelation efficient O(N) recurrent learning. In *Proceedings of the 13th European Symposium on Artificial Neural Networks*. Brugge: d-side.

Suykens, J., Moor, B. D., & Vandewalle, J. (2008). Toward optical signal processing using photonic reservoir computing. *Optics Express*, *16*(15), 11182–11192.

Suykens, J., Van Gestel, T., De Brabanter, J., De Moor, B., & Vandewalle, J. (2002). *Least squares support vector machines*. Singapore: World Scientific.

Takens, F., Rand, D., & Young, L. S. (1981). Detecting strange attractors in turbulence. In D. Rand & L. S. Young (Eds.), *Dynamical systems and turbulence* (pp. 366–381). Berlin: Springer-Verlag.

Triefenbach, F., Jalalvand, A., Schrauwen, B., & Martens, J.-P. (2010). Phoneme recognition with large hierarchical reservoirs. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. S. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems 23* (pp. 2307–2315). Red Hook, NY: Curran Associates.

Vapnik, V. (1995). *The nature of statistical learning theory*. Berlin: Springer-Verlag.

Verstraeten, D., Schrauwen, B., d'Haene, M., & Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, *20*(3), 391–403.

Verstraeten, D., Schrauwen, B., & Stroobandt, D. (2006). Reservoir-based techniques for speech recognition. In *Proceedings of the International Joint Conference on Neural Networks* (pp. 1050–1053). Piscataway, NJ: IEEE.

Williams, C.K.A. (1998). Computation with infinite neural networks. *Neural Computation*, *10*, 1203–1216.

Wyffels, F., & Schrauwen, B. (2010). A comparative study of reservoir computing strategies for monthly time series prediction. *Neurocomputing*, *73*, 1958–1964.

Received December 21, 2010; accepted May 13, 2011.