JACOBS
UNIVERSITY

Mantas Lukoševičius

On self-organizing reservoirs and their hierarchies

Technical Report No. 25
October 2010

School of Engineering and Science

On self-organizing reservoirs and their hierarchies

Mantas Lukoševičius[1]

School of Engineering and Science
Jacobs University Bremen gGmbH
Campus Ring 12
28759 Bremen
Germany

http://www.jacobs-university.de/

Summary

Current advances in reservoir computing have demonstrated that fixed random recurrent networks with only readouts trained often outperform fully-trained recurrent neural networks. While full supervised training of such networks is problematic, intuitively there should also be something better than a random network. In this contribution we investigate a different approach which is in between the two. We use reservoirs derived from recursive self-organizing maps that are trained in an unsupervised way and later tested by training supervised readouts. This approach enables us to train greedy unsupervised hierarchies of such dynamic reservoirs. We demonstrate in a rigorous way the advantage of using the self-organizing reservoirs over the traditional random ones and using hierarchies of such over single reservoirs with a synthetic handwriting-like temporal pattern recognition dataset.

[1]Corresponding author, e-mail: m.lukosevicius@jacobs-university.de

1 Introduction

Despite the biological plausibility and theoretical computational universality of artificial Recurrent Neural Networks (RNNs), their practical applications are still scarce. This should arguably be attributed to their training being far from trivial. While many algorithms for training RNNs exist, they usually require a high level of expertise and do not scale up well to large networks. There are likely even theoretical limitations to using gradient descent training techniques in such networks [1] [2]. One fresh strain of RNN training approaches has even abandoned training the recurrent part of the network at all. The strain was pioneered by Echo State Networks (ESNs) [3] in machine learning and Liquid State Machines [4] in computational neuroscience and is increasingly referred to as Reservoir Computing (RC) [5] [6]. The fact that a simple ESN having a *randomly* generated recurrent part (called *reservoir*) and only a readout from it trained is outperforming sophisticated RNN training algorithms in many tasks [7] [8] [9] is in a way odd if not embarrassing. Intuitively, there should be something better than a random network. It is also not well understood what makes a reservoir good for a particular task. Dissatisfied with this situation many researchers are looking for such qualities and for novel RNN (reservoir) adaptation algorithms (see [6] for an overview).

A part of our contribution is along the lines of these efforts. We focus our attention on an *unsupervised* pre-training of the reservoir followed by a supervised training of the readout. These are the middle grounds between a fixed reservoir and a fully in a supervised way trained RNN, hopefully mitigating the shortcomings of both. Among the benefits of the unsupervised pre-training are the ability to utilize unlabeled data (input signals with no corresponding target outputs) that are abundant in many applications and the ability to use such reservoirs as building blocks for more complex hierarchical architectures since they don't require an external error signal back-propagated through other complex components and diluted in the process.

The latter is the main original motivation to this research and is further explored in this paper. We investigate architectures of multiple layers of such reservoirs that are greedily trained in an unsupervised way. It has been recently argued [10] and widely believed in the machine learning community that such deep architectures are necessary for more challenging tasks.

The article is organized as follows. We specify our self-organizing reservoir in Section 2, together with a short discussion on the properties of the neuron model used in Section 2.2, and the training algorithms used in Section 3, both the unsupervised training in Section 3.1 and the readout mechanism in Section 3.2. We compare the self-organizing reservoir with simple ESNs in Section 4 and show that it is consistently better in the investigated task. More concretely, we specify the baseline ESN, the task used, and the technical details of the simulations in Sections 4.1, 4.2, and 4.3 respectively. We then analyze the results of the comparison in Section 4.4. Utilizing the benefits of the self-organizing reservoirs we build multi-

layered hierarchies of them in Section 5, giving the technical details in Section 5.1 and analyzing the success in Section 5.2.

2 Computational model

2.1 Self-organizing reservoir

There are quite some unsupervised adaptation techniques for the "weighted sum and nonlinearity" (WSN) type of neurons suggested and recently investigated in the RC context, however often the improvements they offer are minute and the adaptation can be non-converging (see Section 6 in [6] for an overview). Instead, we focused our attention to the tradition of Self-Organizing (also called Kohonen) Maps (SOMs) [11], probably the most classic unsupervised neural network training method, and their recurrent extensions. While there are many extensions of SOMs to temporal data suggested (see [12] for a systematic overview), the one truly fully recurrent model, as in normal WSN fully recurrent networks, was introduced as Recursive Self-Organizing Maps (RSOMs) in [13]. In this work we use a similar model for the reservoir with the state update equations

$$\tilde{x}_i(n) = \exp(-\alpha \left\| {\mathbf{W}^{\text{in}}}_i - \mathbf{u}(n) \right\|^2 - \beta \left\| \mathbf{W}_i - \mathbf{x}(n-1) \right\|^2), \quad i = 1, \ldots, N_{\text{x}}, \quad (1)$$

$$\mathbf{x}(n) = (1-\gamma)\mathbf{x}(n-1) + \gamma \tilde{\mathbf{x}}(n), \quad (2)$$

where $\mathbf{u}(n) \in \mathbb{R}^{N_{\text{u}}}$ is the input signal, $\mathbf{x}(n) \in \mathbb{R}^{N_{\text{x}}}$ is a vector of reservoir neuron activations and $\tilde{\mathbf{x}}(n) = [\tilde{x}_1(n), \ldots, \tilde{x}_{N_{\text{x}}}(n)]^{\text{T}} \in \mathbb{R}^{N_{\text{x}}}$ is its update, all at time step n, $\|\cdot\|$ stands for the Euclidean norm, ${\mathbf{W}^{\text{in}}}_i$ is the ith column of the input weight matrix $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_{\text{u}} \times N_{\text{x}}}$, $\mathbf{W}_i$ is the ith column of the recurrent weight matrix $\mathbf{W} \in \mathbb{R}^{N_{\text{x}} \times N_{\text{x}}}$, $\gamma \in (0, 1]$ is the leaking rate, and α and β are scaling parameters for the input and the recurrent distances respectively.

Since we use leaky integration (2), our model also resembles the earlier temporal Kohonen networks [14] and recurrent self-organizing maps [15] that use leaky integration as the only type of recurrence. The unit activation function (1) is a Gaussian and in fact can be seen as a Radial Basis Function (RBF). However, to the best of our knowledge, this type of fully recurrent systems has not been investigated in the RBF literature, the closest of such being the recurrent RBF network [16] which is similar to the temporal Hebbian SOM [17]. There seem to not be any citations between the two communities.

A recent biologically-motivated contribution with similar objectives was introduced in [18]. Also recursive SOMs are used as a pre-processor in the context of reservoir computing in [19].

2.2 Properties of the neuron

The difference between this model and a more conventional RNN as, for example, in ESNs is the model of neuron: the WSN type $x = f(\mathbf{W}\mathbf{u})$ versus the RBF type $x = f(\|\mathbf{W} - \mathbf{u}\|)$, where $f(\cdot)$ stands for a nonlinearity, typically a $\tanh(\cdot)$ sigmoid in the first case and a Gaussian in the second. Even more specifically it is how the inputs to the neurons are combined. As a result RBF neurons have some very different properties from the WSN neurons:

- Locality. By the virtue of calculating Euclidean distance $\|\mathbf{W} - \mathbf{u}\|$ between the vectors of its inputs $\mathbf{u}$ and the input weights $\mathbf{W}$ (as opposed to dot product $\mathbf{W}\mathbf{u}$ in WSN units), responses of RBF units are intrinsically local. The response is centered around $\mathbf{W}$ as the "prototype" input pattern that excites the unit most. The excitation drops sharply when $\mathbf{u}$ moves away from $\mathbf{W}$.

- Prototype input pattern. The prototype input pattern $\mathbf{u} = \mathbf{W}$ is bounded for RBF units as opposed to WSN units where it is asymptotic: the bigger the scalar projection of $\mathbf{u}$ on $\mathbf{W}$, the bigger the output.

- Quasi-sparse coding. A group of RBF units receiving the same input $\mathbf{u}$ and having different weights $\mathbf{W}$ produces a sparse spatial coding of $\mathbf{u}$ in a sense that only a few units, $\mathbf{W}$ of which are closest to $\mathbf{u}$, have higher activation. This however is not sparse coding in the strict sense of the term [20] but something in between a sparse coding and a local coding. On one hand, units are excited by their local prototype patterns, on the other hand they have continuous activations and in general any input can be reconstructed from their activations as long as the number of units in the population is greater than the dimensionality of the input and they are not on a lower-dimensional hyperplane in the input space.

- Greater nonlinearity. The response of a RBF unit is more nonlinear than that of the sigmoidal activation function of WSN units, which is monotonic and changes only along one direction of the input space.

- Signal value invariance. Prototype inputs of RBF units can be placed anywhere in the input space, treating any value on the axis of real numbers like any other. For example, an RBF unit can have its maximal response to the input with all values 0 (if $\mathbf{W} = \mathbf{0}$), while an WSN unit will have a zero activation independent of its $\mathbf{W}$. This gives more flexibility and requires less care when encoding the inputs.

- No signal energy metaphor. A closely related issue is that with RBF units the notion of signal energy is lost. Higher energy, or amplitude, input signals $\mathbf{u}$ do not automatically result in higher energy, or amplitude, output signals $\mathbf{x}$ and vice versa. In particular, because the zero input signal has no special meaning, there is no "dying-out" of the signals due to zero input.

Some of these features are desirable in a reservoir while others are daunting. In particular, recurrent networks of such units are hard to analyze analytically, because many insights on the WSN type of reservoirs, such as the role of the spectral radius of the recurrent connection matrix [3], are based on their local linearization, which is hard to apply here. The derived stability conditions for RSOMs [21] are not as elegant as for reservoirs of WSN units.

In a way the RBF units in a recurrent network work as detectors of the state of the dynamical system while at the same time constituting the next state by representing the dimensions of it. Each of them "detects" a certain situation in the combined space of the input and its history. This makes such a reservoir a natural candidate for temporal pattern detection or classification tasks.

The specific properties of the RBF units allow them to be trained by some state-of-art unsupervised methods.

3 Training

The training of our model consists of the unsupervised pre-training of the reservoir (1) and a supervised training of a readout from it to assess the effects of unsupervised adaptation.

3.1 Unsupervised training of the reservoir

Based on the fact that input weights of the RBF units "dwell" in the same space as their inputs, there is a large range of unsupervised training methods available for them. Most of them combine competitive and collaborative aspects of learning to make the units nicely span the input space. Virtually all such methods for static data are also applicable to our recurrent model (1). One natural option for training (1) is the classical SOM learning algorithm [22]:

$$\begin{aligned} \mathbf{W}^{\text{in}}{}_i(n+1) &= \mathbf{W}^{\text{in}}{}_i(n) + \eta(n) h(i,n) \left(\mathbf{u}(n) - \mathbf{W}^{\text{in}}{}_i(n) \right), \\ \mathbf{W}_i(n+1) &= \mathbf{W}_i(n) + \eta(n) h(i,n) \left(\mathbf{x}(n) - \mathbf{W}_i(n) \right), \end{aligned} \tag{3}$$

with $\eta(n)$ being the learning rate and the learning gradient distribution function

$$h(i,n) = \exp\left(-\frac{d_h(i, \text{bmu}(n))^2}{w_h(n)^2} \right), \tag{4}$$

where $\text{bmu}(n) = \text{argmax}_i \left(x_i(n) \right)$ is the index of the "best matching unit" (BMU), $d_h(i,j)$ is the distance between units with indices i and j in the additionally defined topology for reservoir units, and $w_h(n)$ is the neighborhood width of the gradient distribution. In our experiments we use a 2D rectangular lattice where $d_h(i,j)$ is the Manhattan distance between nodes i and j on it. Intuitively, $h(i,n)$ distributes the error gradient in (3) so that the BMU is updated with the biggest individual learning rate (since $h(\text{bmu}(n), n) \equiv 1$) and this rate drops as a smooth Gaussian

going further away from the BMU in the defined topology of units. Note, that we are using $\mathbf{x}(n)$ for finding $\mathrm{bmu}(n)$ as in recurrent SOMs [15] as opposed to using $\tilde{\mathbf{x}}(n)$ as in temporal Kohonen networks [14]. $\eta(n)$ and $w_h(n)$ control the *schedule of the training* process by varying the overall learning rate and amount of learning done outside the BMU at each time step respectively.

Neural gas (NG) is another closely related alternative learning method to SOMs that we tried. It differs only in the gradient distribution function, which instead of (4) is

$$h_{ng}(i,n) = \exp\left(-\frac{d_{ng}(i,n)}{w_h(n)}\right), \tag{5}$$

where $d_{ng}(i,n)$ is the zero-based index of the node i in the descending ordering of nodes by their $x_i(n)$. As in the case of RSOM, $d_{ng}(\mathrm{bmu}(n),n) \equiv 0$ and $h(\mathrm{bmu}(n),n) \equiv 1$. In our experiments we got similar results with both RSOM and NG training and will only report on the former.

3.2 Supervised training of the readouts

After unsupervised adaptation of such a reservoir to the given input $\mathbf{u}(n)$ a readout $\mathbf{y}(n) \in \mathbb{R}^{N_\mathrm{y}}$ from such a reservoir can be trained in a supervised way to match a desired output $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$. A natural and straightforward option is to use a linear readout

$$\mathbf{y}(n) = \mathbf{W}^\mathrm{out}[1;\mathbf{x}(n)], \tag{6}$$

where $[\cdot;\cdot]$ stands for a vertical vector concatenation. The output weight matrix $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y}\times(N_\mathrm{x}+1)}$ is learned using linear regression, a standard technique in reservoir computing [23]. The input $\mathbf{u}(n)$ can also be concatenated to $[1;\mathbf{x}(n)]$ in (6), making $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y}\times(N_\mathrm{x}+N_\mathrm{u}+1)}$.

In this work we will not put much emphasis on designing elaborate output schemes for particular applications (which would be important for tasks like classification or detection), but rather use simple linear outputs trained on simple targets to estimate the quality of the unsupervised adaptation in $\mathbf{x}(n)$.

4 Comparing self-organizing reservoirs with ESNs

We made a systematic comparison between self-organizing reservoirs and classical random echo state networks. We will first specify the ESNs used here for the sake of completeness in Section 4.1, describe the data on which all the experiments are run in Section 4.2, give the technical details of the numerical simulations in Section 4.3, and analyze the results in Section 4.4.

4.1 Baseline ESN architecture

We compare our self-organizing reservoirs presented in Section 2.1 to reservoirs of classical echo state networks [3] [23] with the update equation

$$\tilde{\mathbf{x}}(n) = f(\mathbf{W}^{\text{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)), \tag{7}$$

where $f(\cdot) = \tanh(\cdot)$ is the neuron activation function applied element-wise, instead of (1). Here the input weight matrix $\mathbf{W}^{\text{in}} \in \mathbb{R}^{N_{\text{x}} \times (N_{\text{u}}+1)}$ is a randomly-generated matrix with elements uniformly distributed in a zero-centered range set by the *input scaling* parameter. The recurrent weight matrix $\mathbf{W} \in \mathbb{R}^{N_{\text{x}} \times N_{\text{x}}}$ is a random sparse matrix with 20% connectivity and scaled to have a predefined spectral radius $\rho(\mathbf{W})$. The rest of the model, including leaky integration (2) and readout (6), is the same.

4.2 Synthetic smooth temporal pattern dataset

To investigate the effects of unsupervised pretraining in more controlled conditions we used a synthetically generated smooth temporal pattern dataset (Figure 1) in our simulations, the same as in some of our previous work [24, 8]. It is essentially a multidimensional red noise background signal with smoothly embedded short temporal patterns. Both the background signal and the patterns are generated in the same way by low-pass filtering white noise. Both the background signal and the patterns have the same amplitude and frequency makeup. The patterns are embedded into the background signal by applying smooth envelopes that also have the same frequency makeup. At the places where the patterns are embedded the background signal is suppressed by the envelope and the pattern is accordingly multiplied by it, producing smooth cross-fades between the background and the pattern. The pattern signals have the same dimensionality as the background and appear in all dimensions of the background simultaneously. All the dimensions in the background and inside the patterns are generated as independent signals. The patterns are chosen randomly with equal probabilities and embedded at randomly varying intervals in the background. The patterns do not overlap. Almost half on the final signal is constituted by the patterns, the rest is the background. Thus, the more different patterns there are, the rarer they appear in the signal. The average length of the pattern in the final signal is about 20 time steps. The whole signal in addition is moderately time-warped: the original time step of size 1 can obtain values from the interval $[0.5, 1.5]$ during the transformation. See section 6.3 in [24] for more technical details on how the data were produced.

The dataset was originally designed having handwriting in mind. The background can be seen as the unknown handwriting characters and the patterns as frequently reappearing letters to be recognized in it. Everything can be seen as encoded in, for example, pen movement data, or features extracted from sequentially scanning the handwriting and treating it as a time series.

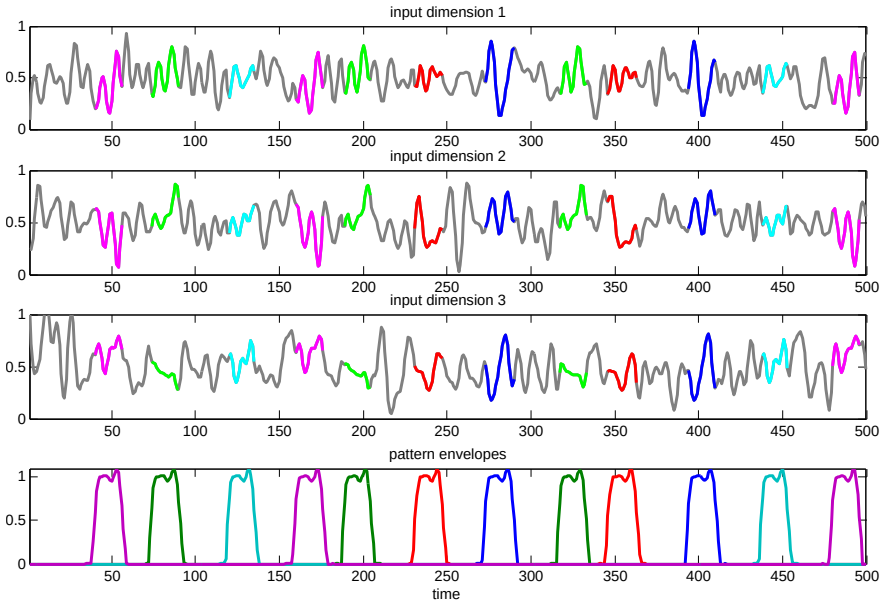


Figure 1: A sample of a three-dimensional smooth pattern dataset with five different patterns highlighted in colors and their corresponding envelopes.

In our experiments for unsupervised training we only use the data with no targets. The general idea is to test how well the unsupervised models learn the structure of the data. To evaluate this we estimate how well the patterns are separable form the rest of the signal in the reservoir activation space $\mathbf{x}(n)$. More concretely, we test how well the envelopes of the patterns can be recovered from it. For this we train a supervised output (6) using a signal containing the envelopes with which the N_y patterns were inserted as the training target $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$.

This data difficult in several aspects:

- The background can literally become very similar to the pattern. This makes perfect learning impossible. The lower the dimensionality of the signal the higher the probability of this happening.
- Because the background signal and the patterns (including transitions) have the same amplitude and frequency makeup there are no "cheap tricks" to spot them without looking at the exact shape of the signals. In fact the patterns are not easy to see in the signal by the naked eye if they are not highlighted.
- The background signal is very information-rich because it is not repeating itself. This can be a problem for some unsupervised models.
- Time warping and relative slowness of the signal can be problematic for some models.

4.3 Simulation details

Since our data are generated randomly and the difficulty of the task depends on the concrete instance, we run and average all our experiments with the same ten different instances of the data. We also run the experiments with a different number of patterns (corresponding to the dimensionality of the target N_y) in the data, ranging from one to five. The input dimension of the data is always $N_\mathrm{u} = 3$.

The training data of 50'000 time steps was used, of which the first 500 were discarded from training and used to "wash out" the initialization transient in the reservoirs. For such a long dataset overfitting was found to not be an issue. Thus selection of the best parameters was performed on the same training set, choosing the ones that give the smallest training error averaged over the ten runs. Testing was also performed on continuations of the datasets of length 10'000 . Testing errors match very closely the training errors (both presented in the article) proving that overfitting is not happening. The exact same setup was used for both models.

All the three dimensions of input data were normalized to have 0.01 variance and zero mean. For self-organizing reservoirs the input data was then shifted to have a 0.5 mean, such that it lies almost entirely inside the $[0, 1]$ interval. This was done for convenience since the weights $\mathbf{W}^\mathrm{in}$ where standardly initialized for the $[0, 1]$ input intervals. For ESNs the data was left at zero mean, as this was found to give slightly better results.

As the performance criteria for all the models we use the normalized root mean square error (NRMSE) of the pattern envelopes reconstructed from $\mathbf{x}(n)$ (with $\mathbf{u}(n)$ appended) using linear regression (6), as mentioned in Section 4.2. This way both models have the same amount of parameters that are trained in a supervised way, namely $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y} \times (N_\mathrm{x}+N_\mathrm{u}+1)}$ (or $N_\mathrm{x} + N_\mathrm{u} + 1 = 54$ parameters per pattern). The pattern envelopes on which the outputs are trained (and performance evaluated) are delayed by one time step, as such shift was found to give the best performance.

The input weight matrix of the self-organizing reservoir was initialized as $\mathbf{W}^\mathrm{in}(0) = \mathbf{a}\mathbf{b}^\mathrm{T}$, where $\mathbf{a} \in \mathbb{R}^{N_\mathrm{u}}$ is a column-vector containing a linear progression starting at $1/N_\mathrm{u}$ and ending at 1 and $\mathbf{b} \in \mathbb{R}^{N_\mathrm{x}}$ is a column-vector containing a linear progression starting at $1/N_\mathrm{x}$ and ending at 1. This gives a matrix with an almost-zero value at the top left corner, value one at the bottom right corner, and all the elements gradually interpolated between these values, which spans the $[0, 1]$ intervals of the input space nicely. This is a rather simplistic initialization. A more sophisticated classical SOM initialization method would be making the two edges of the lattice of the reservoir units follow the two principal components of the input data. $\mathbf{W}$ was simply initialized as the identity matrix $\mathbf{W}(0) = \mathbf{I}_{N_\mathrm{x}}$.

The $N_\mathrm{x} = 50$ units of the self-organizing reservoirs were arranged into a 2D rectangular lattice topology of 10×5 units for the SOM type (4) of unsupervised initial training. The training schedule for the self-organizing reservoirs was set manually. The unsupervised training was done by passing through the training data once, that is in 49'500 time steps. The learning rate $\eta(n)$ (3) followed a

geometric progression from 0.04 to 0.01 and the neighborhood width $w_h(n)$ from 2 to 0.001. These parameters were found to be reasonable through some manual experimentation.

4.4 Simulation results

To answer the question whether and by how much the use of the self-organizing reservoir and its unsupervised pretraining benefits the pattern detection task, we compare it with the regular type of reservoirs (7). We use the same number of neurons $N_{\mathrm{x}} = 50$ in both types of reservoirs, so that the reservoir signal space $\mathbf{x}(n)$ has the same dimensionality. Both models have a number of parameters that need to be set and values of these parameters influence the performance of the models. To have a fair comparison we have run a grid search over the three parameters that affect the performance most in both of the models. For the self-organizing reservoir these three parameters are: input and recurrent distance scalings α and β in (1), and the leaking rate γ in (2). For the regular ESNs the three similar parameters are: the scaling of the input matrix $\mathbf{W}^{\mathrm{in}}$ (such that the elements of $\mathbf{W}^{\mathrm{in}}$ are uniformly distributed between plus and minus the value of the scaling) in (7), the spectral radius of the recurrent connection matrix $\rho(\mathbf{W})$ (7), and the same leaking rate γ in (2).

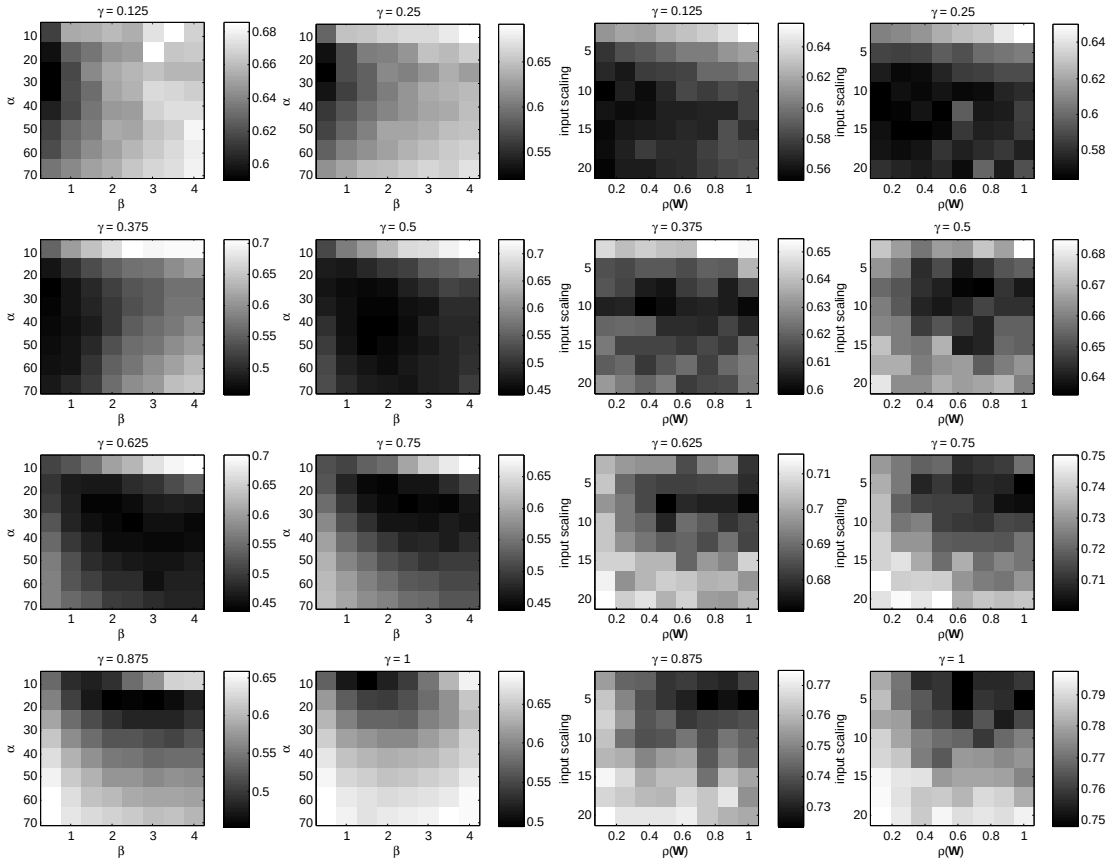
Table 1: Grid search parameters and best values found.

Model	**Self-organizing reservoir**			**ESN**		
Parameter	γ	α	β	γ	input scaling	$\rho(\mathbf{W})$
Min. value	0.125	25/3	0.5	0.125	2.5	0.125
Step size	0.125	25/3	0.5	0.125	2.5	0.125
Max. value	1	200/3	4	1	20	1
Best values						
1 pattern	0.75	100/3	2.5	0.25	10	0.75
2 patterns	0.875	50/3	2	0.25	10	0.125
3 patterns	0.625	100/3	2.5	0.125	10	0.125
4 patterns	0.5	125/3	1.5	0.125	15	0.125
5 patterns	0.5	100/3	1.5	0.125	20	0.125

The ranges, step sizes, and the best found values of the parameters are presented in Table 1. We take eight values of each parameter over a reasonable interval. It took multiple trials to get the parameter ranges right. We found that good α and β values in (1) should account for the normalization over the dimensionality of the input N_{u} and the reservoir N_{x}. This is logical, because the two $\|\cdot\|^2$ terms in (1) are summations over N_{u} and N_{x} dimensions respectively, and a bigger dimensionality gives a bigger sum. As a result the intervals of α and β look very different in Table 1 because they are normalized. If we denormalize them, we see that αN_{u} and βN_{x} have the exact same intervals of $[25, 200]$. ESNs gave the

best performance with quite surprisingly big input scalings, together with small leaking rates and spectral radii.

Training and testing of a self-organizing network took about 7.6 seconds on an average modern computer, so it is quite fast. Training and testing an ESN took about 3.0 seconds. Thus self-organizing maps received more pure computational time. On the other hand we have selected not only the best parameters from the grid search for ESNs, but took the same ten randomly generated reservoirs with these parameters (that gave the best average performing) for testing. Thus the performance fluctuations caused by randomness of the reservoirs were used to the advantage of ESNs.



(a) Self-organizing reservoir (b) ESN

Figure 2: Mean training error surfaces for data with three different patterns.

Figure 2 illustrates in more details how the mean training error (averaged over the ten runs) depends on the three parameters set by the grid search for both of the models. The case with three patterns is shown here. The surfaces for ESNs are a bit more rough because the use of random reservoirs introduces some additional variance in performance. But the ranges of the ESN performance are smaller. The self-organizing reservoirs are a bit more sensitive to the set parameters, at

least in these ranges. We can see that the mean error surfaces are rather smooth, not abrupt, indicating that the right parameters do not need to be picked very accurately to get reasonably good performance.

The pattern separation errors for the two models and different numbers of patterns in the data are presented in Figure 3. The best parameters found using the grid search (Table 1) were used for every number of patterns in both models. The bigger spread of the ESN errors can be explained by the the use of randomly generated reservoirs.

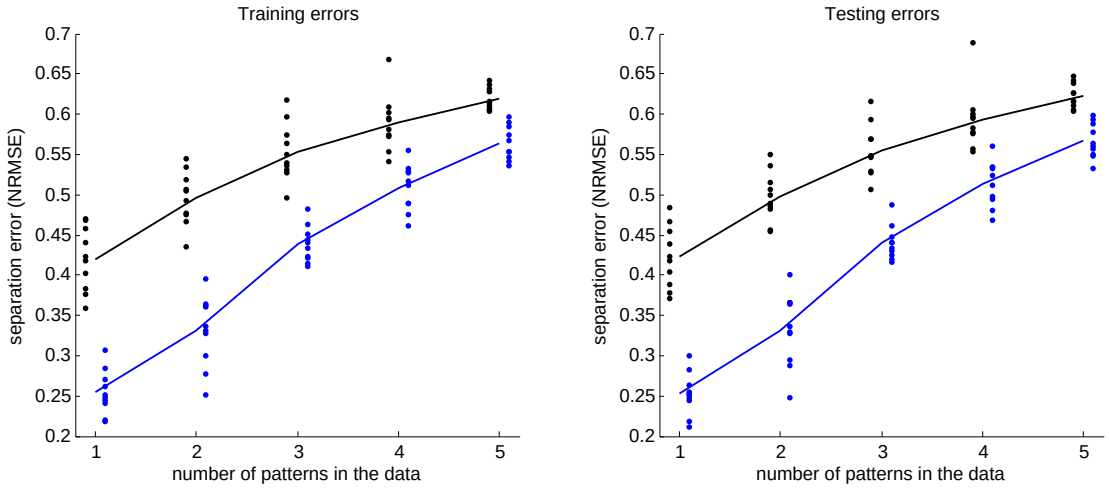


Figure 3: Separation errors with best parameter values of self-organizing reservoirs (blue) and ESNs (black) for different number of patterns in the data. The values of the ten data instances are represented by dots (slightly displaced for visual clarity) and mean values by lines.

Figure 3 shows clearly that unsupervised training benefits the patterns separation in the reservoir space. This improvement is statistically significant, present with different numbers of patterns in the data, and, because of the grid search, is not caused by parameter settings favoring one of the methods. We also see that training and testing errors are almost identical in all the cases, justifying choosing the parameters based on the training error.

Looking at Figure 3, we see that the benefit of the self-organizing reservoirs is bigger in the cases where there are fewer different patterns in the data. The reason for this is that given the limited capacity of the self-organizing reservoir it can learn to better represent fewer different patterns than more, while the random reservoir of ESN is universal and the readouts for the different patterns are virtually independent. The drop in performance with the number of different patterns is only due to the fact that each pattern appears more rarely in the input and thus is harder to learn.

To visualize how different number of patterns are represented in the signal space $\mathbf{x}(n)$ of the self-organizing reservoir, in Figure 4 we plot the two principal

components of $\mathbf{x}(n)$ with activations corresponding to the patterns highlighted in colors.

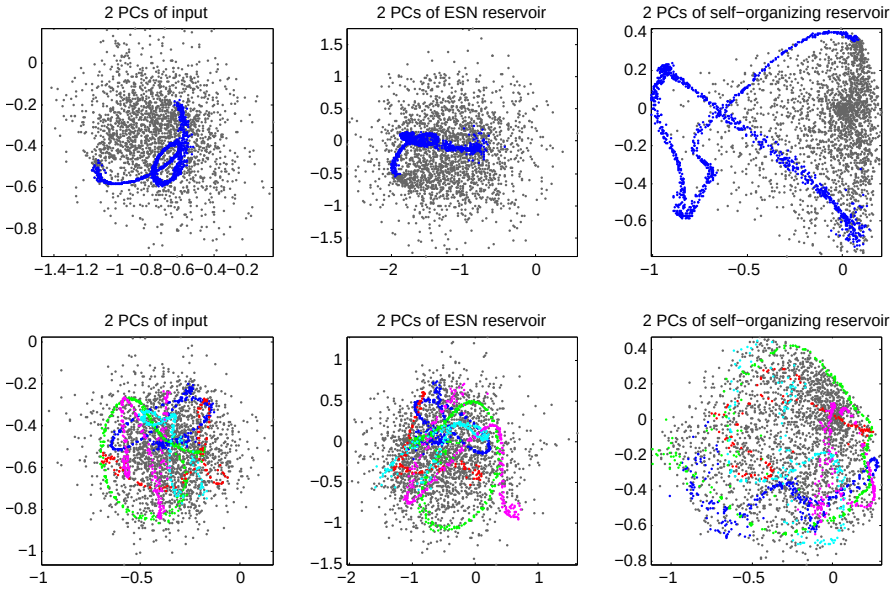


Figure 4: 1 pattern and 5 patterns highlighted in the two principal components of $\mathbf{u}(n)$, and in $\mathbf{x}(n)$ of both ESN and the self-organizing reservoir.

We can see that a single pattern gets a special representation in $\mathbf{x}(n)$ of the self-organizing reservoir which is already clearly visible in the two principal components. With more patterns we can see that they are spread more than in the ESN or in $\mathbf{u}(n)$ but are in no way easily separable.

5 Hierarchies of self-organizing reservoirs

As mentioned in the introduction, one of the main benefits of unsupervised learning is that components trained this way can be easier assembled into more complex architectures. Here we investigate a simple layered hierarchy of such reservoirs where the bottom reservoir receives the external input $\mathbf{u}(n)$ and every reservoir above receives the activations $\mathbf{x}(n)$ of the reservoir directly below it as the input. Such an architecture features only bottom-up interactions and can be trained in a greedy layer-by-layer way starting from the bottom. Since every layer is trained independently from the rest, this hierarchical structure in essence does not introduce additional difficulties in training, except more of it needs to be done, because there are more layers.

When comparing a hierarchy to a single reservoir, a natural question to ask is whether it is better to invest the additional effort in training many layers of the hierarchy or in better training of the single reservoir. More concretely, we take the training time measured in epochs as the "effort". As a generalization of this

question, we investigate how the performance depends on the number of layers in the hierarchy for a given fixed training effort. By this we mean that if a hierarchy has k layers and the fixed training effort is l epochs, then each layer receives l/k epochs of training.

5.1 Simulation details

For the experiments with the hierarchies we used the same data described in Section 4.2. The same ten data instances with $N_\mathrm{y} = 5$ temporal patterns in it were reused as in Section 4.3 with the same normalization and splitting into the initialization, training, and testing sequences. In these experiments, however, we have gone through the 49'500 time steps of training data multiple times (*epochs*), each time first initializing the model with the initialization sequence of 500 time steps during which the training was not happening.

We again used reservoirs of $N_\mathrm{x} = 50$ units and all the other parameters: $\gamma = 0.75$, $\alpha = 100/N_\mathrm{u}$, and $\beta = 50/N_\mathrm{x} = 1$, the same in every layer. Note, however, that $N_\mathrm{u} = 3$ for the bottom layer and $N_\mathrm{u} = N_\mathrm{x} = 50$ for the others, which affects α accordingly.

For training *all* of our reservoirs we used the same SOM algorithm (3)(4) with the reservoir units again organized into a 10×5 lattice, the same weight initialization, and the same but slightly more subtle training schedule, where $\eta(n)$ followed a geometric progression from 0.01 to 0.001 and the neighborhood width $w_h(n)$ again from 2 to 0.001. The same training schedule was used independently of the length of the training: if the training is taking more epochs, the learning parameters are simply changing slower, but the end-points remain the same.

The same performance criterion is also used: a linear readout (6) is trained on the pattern envelopes as the teacher and the error (NRMSE) of the reconstruction computed. In this case the input $\mathbf{u}(n)$ was not included as part of $\mathbf{x}(n)$ in (6). For every architecture the readout was trained only from the activations of the top-most layer. This way the signal space from which the pattern separation is learned always have the same dimensionality $N_\mathrm{x} = 50$ and every model has the same amount of parameters trained in a supervised way, namely $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y} \times (N_\mathrm{x}+1)}$ (or $N_\mathrm{x} + 1 = 51$ parameters per pattern). The target signal $\mathbf{y}_\mathrm{target}(n)$ for each layer was delayed by the number of time steps equal to the number of layer (1 for the bottom layer, 2 for the second, and so on) as this was found to be optimal at least for a couple of first layers.

5.2 Simulation results

The results showing how different numbers of layers and different numbers of training epochs per layer affect the testing performance are presented in Figure 5. The performance is plotted against the total number of epochs spent in training. Each curve here represents a hierarchy trained with the same amount of epochs

per layer. The points on the curves represent the mean test separation errors in different layers. Every tenth layer is annotated. The hierarchies are trained with 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, or 120 epochs per layer. They are colored from blue (the top-most curve, 120 layers, each trained with 1 epoch) to red (a single point in the middle right, a single layer trained with 120 epochs) as the two extreme cases.

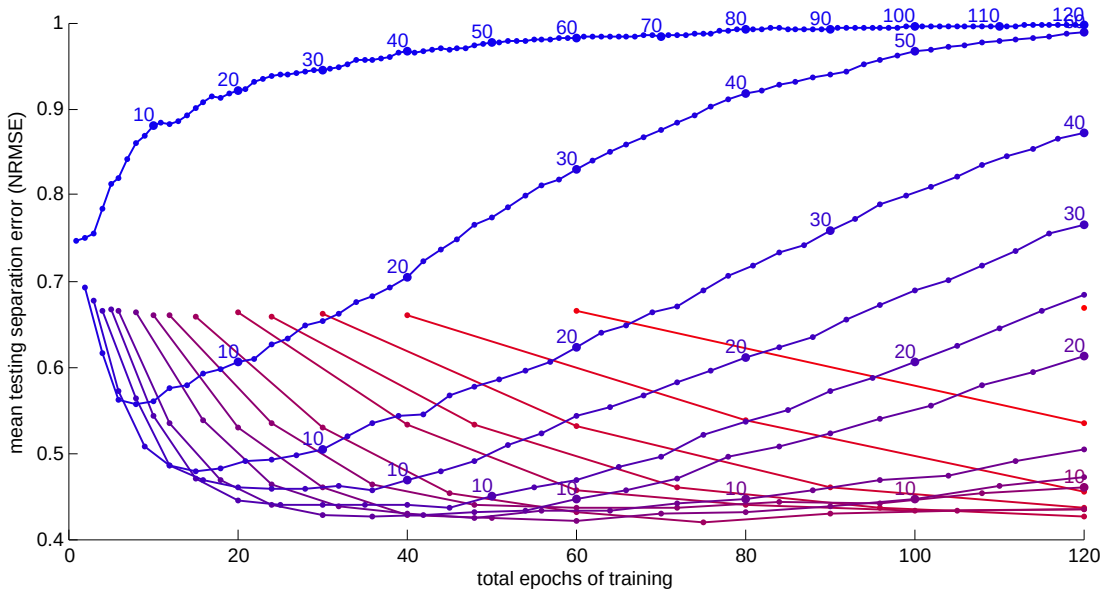


Figure 5: Mean testing separation errors in layers of differently trained hierarchies plotted against the total epochs of training. See text for the details.

We can see that if the layers are not trained well, stacking them in a hierarchy is not going to improve the result. The extreme case with each layer only trained in one epoch is the top-most blue curve. We can see that in this case the performance decreases with every additional layer and is approaching the worst NRMSE of 1. If a layer is not able to learn a good representation of its input, this bad representation is passed to the upper layers, information from the input is lost and the performance only decreases. Because we use quite small learning rates here the training time of one epoch per layer might simply be not enough.

However, when we give each layer enough time to learn a good representation of the input, we observe that adding additional layers improves the performance. The better the individual layers are trained, the better the improvement in the upper layers.

We can visualize the data of Figure 5 from a slightly different perspective. In Figure 6 we connect the dots representing layers of the same level across differently trained hierarchies. Here the six curves represent the errors in the first six layers across the architectures. This way we can see that putting more effort in training a single layer (the top-most blue curve) does improve the performance but only to a point where additional effort does not help anymore. The additional layers are able

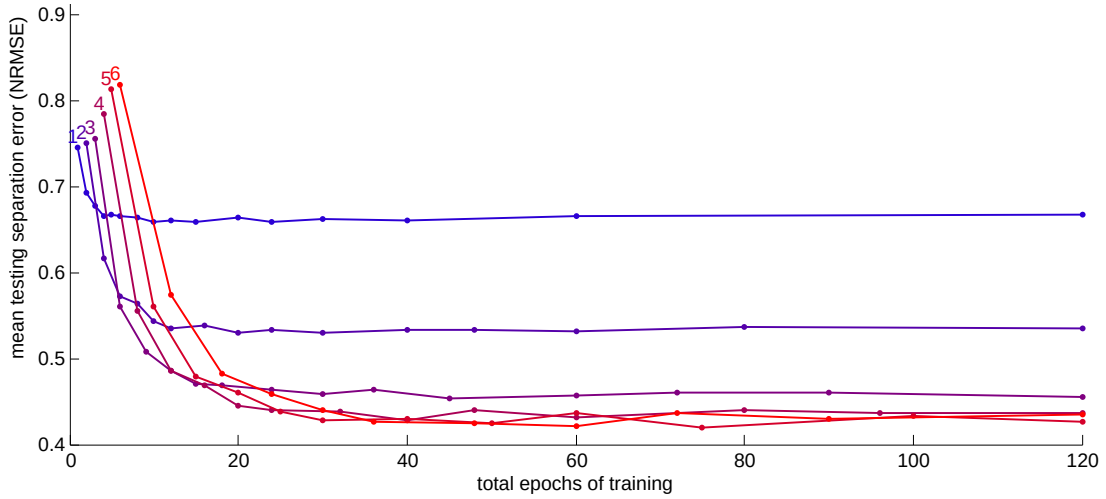


Figure 6: Errors in the first six layers across differently trained hierarchies plotted against the total epochs of training. See text for the details.

to break this performance ceiling achieving much smaller separation errors than a single layer could reach. We observe that when going up into the higher layers there is a significant drop in the error till about the fourth layer. This shows that with the same total number of training epochs hierarchical architectures clearly outperform a single reservoir.

The fact that the additional effort in training yields better results is not at all trivial in this case, because we train our reservoirs in an unsupervised way and test them on a supervised task. This proves that in this case the unsupervised pretraining does indeed improve the performance of a supervised task and there is a positive correlation between the quality of the unsupervised pretraining and the performance on the supervised task.

To have a more detailed view, we single out one case from the Figure 5 where every layer is trained using eight epochs of learning and present it in Figure 7. Here both the training and testing errors are shown in all the fifteen layers. The mean values are presented as well as the ten individual cases with the data instances to indicate the variability among them.

We see, that while the difference between the training and testing errors increases going up in the layers, it still remains quite small, and there is no real overfitting, because both errors reach minimal values at the same layer (6 in this case). One reason for this could be that we use long enough training data with small enough models. Another reason could be that most of the training we do is unsupervised, thus the model could overfit the input data but there is no training target to overfit. It might well be that the input data is too rich and without the target too open for interpretations to overfit.

The representation of five patterns in the two principal components of the activations $\mathbf{x}(n)$ of the first six layers in a hierarchy is presented in Figure 8. We

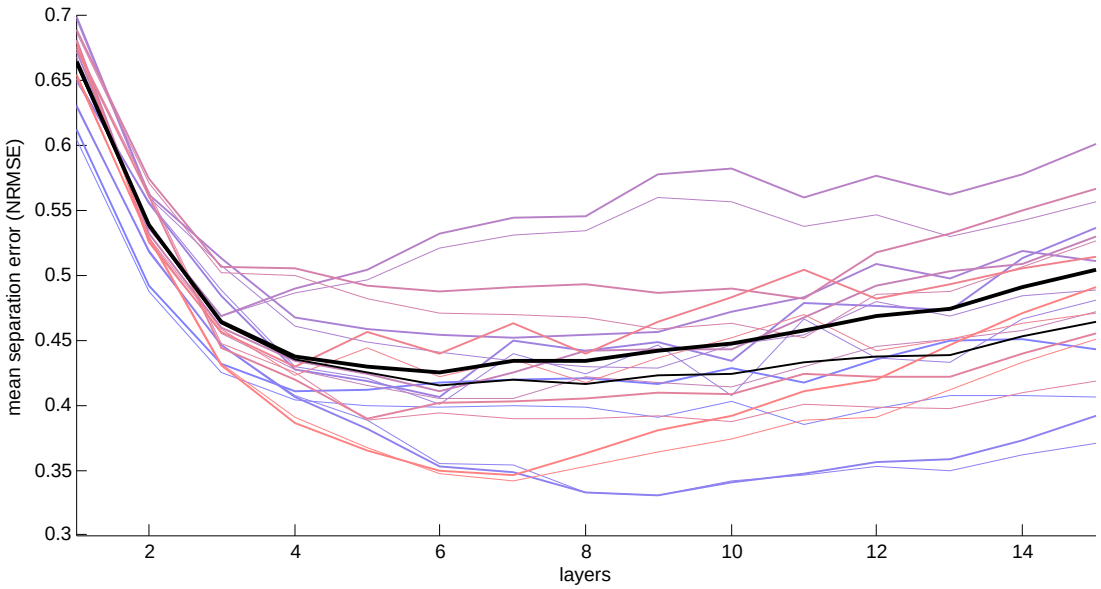


Figure 7: Errors in the layers each trained with eight epochs. Training errors are shown in thin and testing errors in bold lines. The mean values are shown in black and the ten separate instances are shown in light colors.

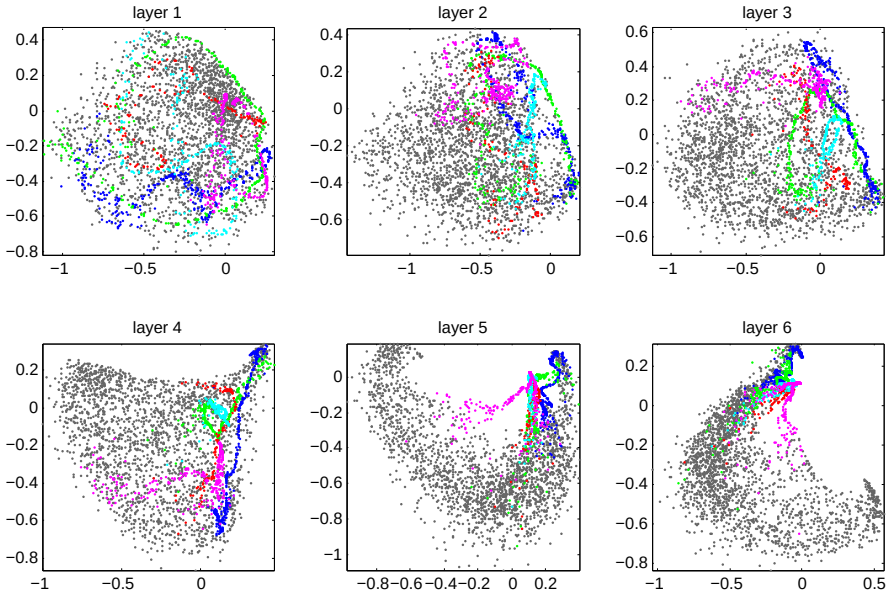


Figure 8: Patterns highlighted in the two principal components of $\mathbf{x}(n)$ in the first six layers of a hierarchy.

can see that the special representation of the patterns in data gets more expressed in the principal components of $\mathbf{x}(n)$ when going up in the hierarchy.

6 Discussion

We have demonstrated that the type of self-organizing recurrent neural network investigated here can learn in an unsupervised way the representations of the temporal input data that enable better results in a supervised task such as separating repeated slow patterns in the input. This was rigorously compared to an echo state network that produces random rich representations of the same dimensionality and found to be better. Parameter sweeps and averaging of results over different instances of data were performed for both models to exclude the possibility of an unfair comparison.

We also showed that longer unsupervised training results in better supervised performance, establishing a positive correlation between the two.

We do not have a rigorous explanation or analytical proof of this correlation, but only some intuitions. The competitive nature of the self-organizing learning diversifies the responses of the units in such a reservoir. Each unit during the training is "looking for" its "niche" input pattern to which it produces a high response. The reservoir tries to "inhabit" the input dynamics and intrinsically looks for patterns in it. The parts of the data that are more predictable get a more expressed representation in the reservoir space, as shown in Figure 4.

We have also demonstrated that hierarchies of such unsupervised reservoirs improve the performance by a large margin. The exact reasons for this need further investigation. Figure 8 gives an insight that the patterns in the data get more expressed in the principal components of the reservoir activations when going up in the layers. One reason for this could be that with more layers we simply get more powerful models having more unsupervisedly trained parameters. As a future work, it would be interesting to check if a single reservoir with a comparable number of parameters could achieve comparable performance. We have only compared the training effort so far. It could also be that the deep structure produces a more compact representation of the input which can not easily be achieved with a single layer (examples of such are presented in [10]). On one hand there is nothing really deep in our data, but on the other hand the fact that we get better representations in the reservoirs of the same dimension when going up the hierarchy is still spectacular.

There are still many improvements that could be done to the model, both to the structure and learning, as well as understanding it better. For example, an interesting and biologically motivated modification would be to also have top-down interactions in the hierarchies. It would also be interesting to see how the model scales up to more challenging real-world data. This is our ongoing work.

Acknowledgments

The research reported here is funded through the EC FP7 project ORGANIC (FP7-231267). The author would like to thank Herbert Jaeger, Vytenis Šakėnas, Michael Thon, Peter Tiňo, and Yoshua Bengio for stimulating discussions and valuable feedback on this research, and H.J. for also proof-reading this text.

References

[1] Kenji Doya. Bifurcations in the learning of recurrent neural networks. In *Proceedings of IEEE International Symposium on Circuits and Systems 1992*, volume 6, pages 2777–2780, 1992.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[3] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[4] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[5] David Verstraeten, Benjamin Schrauwen, Michiel D'Haene, and Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007.

[6] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.

[7] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. *Science*, pages 78–80, 2004.

[8] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

[9] David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Reservoir-based techniques for speech recognition. In *Proceedings of the IEEE International Joint Conference on Neural Networks, 2006 (IJCNN 2006)*, pages 1050 – 1053, 2006.

[10] Yoshua Bengio and Yann LeCun. Scaling learning algorithms toward AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*. MIT Press, Cambridge, MA, 2007.

[11] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, January 1982.

[12] Barbara Hammer, Alessio Micheli, Alessandro Sperduti, and Marc Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8-9):1061 – 1085, 2004.

[13] Thomas Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8-9):979–991, 2002.

[14] Geoffrey J. Chappell and John G. Taylor. The temporal Kohonen map. *Neural Networks*, 6(3):441–445, 1993.

[15] Markus Varsta, José Del R. Millán, and Jukka Heikkonen. A recurrent self-organizing map for temporal sequence processing. In *Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN 1997)*, volume 1327 of *LNCS*, pages 421–426. Springer, 1997.

[16] Man Wai Mak. A learning algorithm for recurrent radial basis function networks. *Neural Processing Letters*, 2(1):27–31, 1995.

[17] Jan Koutník. Inductive modelling of temporal sequences by means of self-organization. In *Proceeding of Internation Workshop on Inductive Modelling (IWIM 2007)*, pages 269–277. CTU in Prague, 2007.

[18] Andreea Lazar, Gordon Pipa, and Jochen Triesch. SORN: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, 4(0):12, 2010.

[19] Igor Farkaš and Matthew W. Crocker. Systematicity in sentence processing with a recursive self-organizing neural network. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 49–54, 2007.

[20] Peter Foldiak and Dominik Endres. Sparse coding. *Scholarpedia*, 3(1):2984, 2008.

[21] Peter Tiňo, Igor Farkaš, and Jort van Mourik. Dynamics and topographic organization of recursive self-organizing maps. *Neural Computation*, 18(10):2529–2567, 2006.

[22] Teuvo Kohonen and Timo Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007.

[23] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.

[24] Mantas Lukoševičius, Dan Popovici, Herbert Jaeger, and Udo Siewert. Time warping invariant echo state networks. Technical Report No. 2, Jacobs University Bremen, 2006.