Self-organized reservoirs and their hierarchies

Mantas Lukoševičius

Jacobs University Bremen, Campus Ring 1, 28759 Bremen, Germany
m.lukosevicius@jacobs-university.de,

Abstract. We investigate how unsupervised training of recurrent neural networks (RNNs) and their deep hierarchies can benefit a supervised task like temporal pattern detection. The RNNs are fully and fast trained by unsupervised algorithms and only supervised feed-forward readouts are used. The unsupervised RNNs are shown to perform better in a rigorous comparison against state-of-art random reservoir networks. Unsupervised greedy bottom-up trained hierarchies of such RNNs are shown being capable of big performance improvements over single layer setups.

1 Introduction

Here we will investigate a way of training unsupervised hierarchies from exclusively temporal signals, where internal dynamics of the models match in time the external dynamics of the input and output signals. We will use recurrent neural networks (RNNs) as the generic model to implement such systems.

There is a considerable body of research showing that even using untrained random or hand-crafted features at bottom layers of deep architectures works reasonably well, e.g., [16,7,17]. Reservoir Computing (RC) has demonstrated a similar effect in RNNs (see, e.g., [13] for an overview). This makes the untrained feature detectors a viable contender to the unsupervised learning and provides a performance baseline to evaluate the usefulness of the unsupervised learning.

To evaluate the different systems we will use errors from supervisedly trained readouts for two temporal pattern detection tasks.

2 Computational models of a single RNN

We will investigate RNNs of two types of simple neuron models: Weighted Sum and Nonlinearity (WSN) and Radial Basis Function (RBF).

Our RNN update equations for the WSN model is:

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

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

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 $\tilde{\mathbf{x}}(n) \in \mathbb{R}^{N_{\text{x}}}$ is its update, all at time step n, $\tanh(\cdot)$ is the neuron activation (nonlinearity) function applied element-wise, $[\cdot;\cdot]$ stands for a vertical vector concatenation, $\mathbf{W}^{\text{in}}$ is the input weight matrix, $\mathbf{W}$ is the recurrent weight

matrix, and $\gamma \in (0, 1]$ is the leaking rate. This is a classical model of an Echo State Network (ESN) reservoir with a leaky integration [5].

Our RNN update equation for the RBF model is:

$$\tilde{x}_i(n) = \exp\left(-\alpha\left\|{\mathbf{v}^{\text{in}}}_i - \mathbf{u}(n)\right\|^2 - \beta\|\mathbf{v}_i - \mathbf{x}(n-1)\|^2\right), \quad i = 1, \ldots, N_{\text{x}}, \quad (3)$$

instead of (1), and the same (2), where $\tilde{\mathbf{x}}(n) = [\tilde{x}_1(n), \ldots, \tilde{x}_{N_{\text{x}}}(n)]^{\text{T}} \in \mathbb{R}^{N_{\text{x}}}$, $\|\cdot\|$ stands for the Euclidean norm, ${\mathbf{v}^{\text{in}}}_i \in \mathbb{R}^{N_{\text{u}}}$ is the ith column of the input weight matrix $\mathbf{V}^{\text{in}}$, $\mathbf{v}_i \in \mathbb{R}^{N_{\text{x}}}$ is the ith column of the recurrent weight matrix $\mathbf{V}$, and α and β are scaling parameters for the input and the recurrent distances respectively. The RBF neurons have some very different properties from the WSN.

This model is most similar to Recursive Self-Organizing Maps (RSOMs) [20,18], but is different in that we use leaky integration (2), which makes it also resemble the earlier temporal Kohonen networks [1] and recurrent SOMs [19] that use leaky integration as the only type of recurrence (see [4] for a systematic review). Even though the unit activation (3) is in fact a Gaussian Radial Basis Function, 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 [14] which is similar to the temporal Hebbian SOM [9]. There seem to not be any citations between the recurrent RBF and SOM communities. We will call our model (3) (2) a Self Organizing Reservoir (SOR).

A recent biologically-motivated contribution with similar objectives was introduced in [10]. Also, RSOMs are used as a pre-processor in the context of reservoir computing in [3].

3 Network generation and training

To test how much the different SOR computational model affects the reservoir computations compared to the standard ESNs, we will include random networks of both kinds in our empirical simulations. We will also include the SORs trained unsupervisedly. Unfortunately there are no really powerful successful unsupervised methods that fully train ESN reservoirs, despite numerous attempts (see [13] for an overview).

We used classical **random** ESN reservoirs (1) where the input weight matrix $\mathbf{W}^{\text{in}}$ is a randomly-generated matrix with elements uniformly distributed in a range set by the *input scaling* parameter, and the recurrent weight matrix $\mathbf{W}$ is a random sparse matrix with 20% connectivity and scaled to have a predefined spectral radius $\rho(\mathbf{W})$. For the random case of the (3) model the input $\mathbf{V}^{\text{int}}$ and recurrent $\mathbf{V}$ weights were all simply drawn from a uniform $[0, 1]$ distribution.

Most of the many **unsupervised training** methods available for RBF type of feed-forward NNs, combining competitive and collaborative learning, are also applicable to our recurrent model (3). One natural option is the classical SOM learning algorithm [8]:

$${\mathbf{v}^{\text{all}}}_i(n+1) = {\mathbf{v}^{\text{all}}}_i(n) + \eta(n)h(i, n)\left([\mathbf{u}(n); \mathbf{x}(n)] - {\mathbf{v}^{\text{all}}}_i(n)\right), \quad \mathbf{v}^{\text{all}} \equiv [\mathbf{v}^{\text{in}}; \mathbf{v}] \quad (4)$$

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

$$h(i,n) = \exp\left(-d_h(i, \mathrm{bmu}(n))^2 / b_h(n)^2\right), \tag{5}$$

where $\mathrm{bmu}(n) = \arg\max_i (x_i(n))$ 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 $b_h(n)$ 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. Note, that we are using $\mathbf{x}(n)$ for finding $\mathrm{bmu}(n)$ as in recurrent SOMs [19] as opposed to using $\tilde{\mathbf{x}}(n)$ as in temporal Kohonen networks [1]. $\eta(n)$ and $b_h(n)$ 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) [15] is a closely related alternative to SOM, that we used. It differs only in the gradient distribution function, which instead of (5) is

$$h_{ng}(i,n) = \exp\left(-d_{ng}(i,n)/b_h(n)\right), \tag{6}$$

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 SOMs, $d_{ng}(\mathrm{bmu}(n),n) \equiv 0$ and $h(\mathrm{bmu}(n),n) \equiv 1$. In our experiments we got similar results with both SOM and NG training.

To empirically evaluate the quality of the reservoirs for the given tasks, a **linear readout** $\mathbf{y}(n) \in \mathbb{R}^{N_\mathrm{y}}$ from the reservoir can be trained in a supervised way to match a desired output $\mathbf{y}_\mathrm{target}(n) \in \mathbb{R}^{N_\mathrm{y}}$:

$$\mathbf{y}(n) = \mathbf{W}^\mathrm{out}[1; \mathbf{x}(n)], \tag{7}$$

where $[\cdot;\cdot]$ stands for a vertical vector concatenation. The output weight matrix $\mathbf{W}^\mathrm{out}$ is learned using linear regression, common in reservoir computing [5]. The input $\mathbf{u}(n)$ can also be concatenated to $[1; \mathbf{x}(n)]$ in (7), expanding the $\mathbf{W}^\mathrm{out}$.

In this work we will not put much emphasis on designing elaborate output schemes for particular applications, but rather use the same simple readouts for all models to estimate the quality of the unsupervised adaptation in $\mathbf{x}(n)$.

4 Empirical comparison of SORs and ESNs

In our empirical simulations we used two different temporal pattern **datasets**.

To have a greater control over the task, a **synthetic dataset** was generated as a three-dimensional red noise background signal with up to five different short temporal patterns of the same nature embedded by cross-fading with smooth envelopes (Figure 1 left). The whole signals are in addition moderately time-warped. See Section 6.2 in [6] for technical details. We will use the envelopes as $\mathbf{y}_\mathrm{target}(n)$ to evaluate the quality of reservoir representations.

To have a more close to life task we generate "infinite" **strings of handwritten digits** taken from the USPS dataset[1] [11]. Each digit image is selected

[1] From Sam Roweis' homepage http://www.cs.nyu.edu/~roweis/data.html.

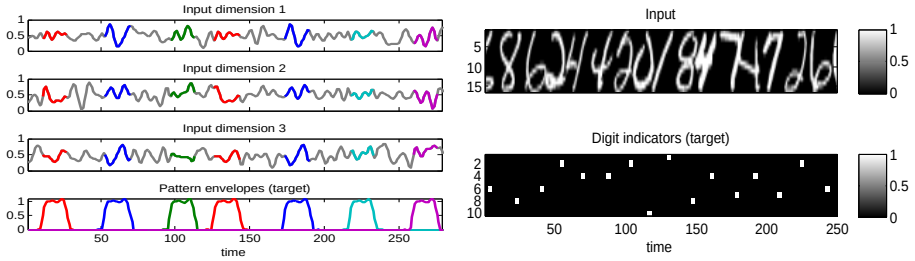

Fig. 1. A sample of a three-dimensional smooth pattern dataset with five different patterns highlighted in colors (left) and USPS concatenated data (right) with their respective supervised targets.

randomly, they are allowed to overlap or have gaps as a result of up to ±3 pixels random uniform displacement (Figure 1 right), making segmentation nontrivial. The generated long images are treated as time series, taking the rows as $N_\mathrm{u} = 16$ input dimensions, and columns as time n, presenting each pixel column as the current input $\mathbf{u}(n)$, going from left to right. For evaluation we also generate targets $\mathbf{y}_\mathrm{target}(n)$ ($N_\mathrm{y} = 10$) indicating classes of the images, i.e., digits in the images. The delay and duration of the indicators were manually adjusted to values that work well with all the investigated single reservoir models.

To avoid unfair comparison of the models by setting wrong parameters, the main three parameters of ESNs: γ (2), $\mathbf{W}^\mathrm{in}$ scaling, and spectral radius $\rho(\mathbf{W})$; and of SOR: γ (2), α, and β (3), are selected through **grid search**, separately for every scenario. We ran through eight values of each parameter over a reasonable interval. It took multiple trials to get the parameter ranges right. We use long enough data sequences (50k time steps for synthetic and 64k for USPS) so that overfitting was found not to be an issue, and selection of the best parameters was based on the average performance on the training sets.

To keep things fast and manageable all types of reservoirs used were of size $N_\mathrm{x} = 50$. All models in every task have the same number of parameters $\mathbf{W}^\mathrm{out} \in \mathbb{R}^{N_\mathrm{y} \times (N_\mathrm{x}+N_\mathrm{u}+1)}$ trained in the same supervised way, (7) with the concatenated input. Parameters trained in a supervised way can not by directly compared with the ones trained in an unsupervised way, because information about the desired output is not available in unsupervised training. In fact, it is not self-evident or always true that unsupervised training helps the supervised one at all, this is that we want to investigate. Thus, we only compare the models with equal number of supervisedly trained parameters.

The unsupervised training (SOM or NG) is done by passing through the training data once. The supervised readouts (7) are trained by passing through the training data once (again). Training of a single model took several seconds on a regular PC, unsupervised training taking about the same as supervised.

We use normalized root mean square error (NRMSE) between the supervised output $\mathbf{y}(n)$ (7) from $[\mathbf{x}(n); \mathbf{u}(n)]$ and the target $\mathbf{y}_\mathrm{target}(n)$ as the performance criterion of reservoir activations (a.k.a. "separation error").

Results. The pattern separation errors of the models with synthetic (different numbers of patterns) and USPS data are presented in Figure 2.

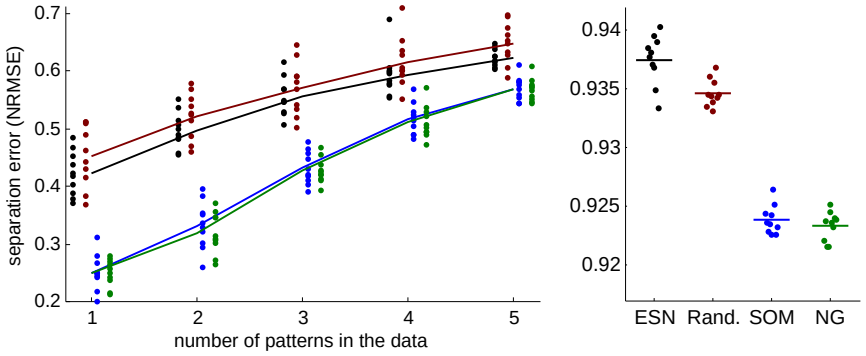

Fig. 2. Test separation NRMSE errors with best parameter values of regular ESNs (black), SORs: random (dark red), trained with SOM (blue) and NG (green) with 10 data instances (dots) and mean values (lines). Results with one to five different patterns in the synthetic data are on the left, and with concatenated USPS data on the right.

The results show clearly that unsupervised training benefits the pattern separation in the reservoir space. The performance of both SOM and NG training are virtually indistinguishable for synthetic, and slightly worse for SOM on USPS (the 2D lattice becomes a hindrance with this more complex data).

The benefit of unsupervised SORs is bigger with fewer synthetic patterns N_y in the data, because of the limited expressive capacity of the unsupervised SOR, while the random reservoirs are universal and the readouts are virtually independent. Still, the performance decreases with N_y for all models due to each pattern appearing more rarely in the input. In all cases recognition of the synthetic patterns would be virtually perfect.

The performance of the random SOR model is similar, to the classical ESN (slightly worse in synthetic and better in USPS), showing it as a viable alternative RNN. For random SORs good α and β in (3) had to be considerably smaller to compensate bigger distances in (3), since vectors in $\mathbf{V}^{\text{in}}$ and $\mathbf{V}$ are distributed randomly and not centered on the typical respective $\mathbf{u}(n)$ and $\mathbf{x}(n)$ values by unsupervised training.

5 Hierarchies of self-organizing reservoirs

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, the training effort of the hierarchy scales linearly with the number of layers, with no additional complexity.

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. In our experiments, if a hierarchy has k layers and the fixed training effort is l epochs, then each layer receives l/k epochs of training.

For the experiments with the hierarchies we used the same synthetic data as in Figure 1 (left) with $N_y = 5$ different patterns. In these experiments, however, we have gone through the training data multiple times (*epochs*). We again used reservoirs of $N_x = 50$ units and the same unsupervised SOM algorithm (4)(5). The same more gentle 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. For every architecture the readout (7) was trained only from the activations of the top-most layer, thus number of supervised parameters is constant $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_y \times (N_x+1)} = \mathbb{R}^{5\times 51}$. We used the same pattern separation error performance criterion.

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 3. The performance is plotted against the total number of unsupervised training epochs. Each curve 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. 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.

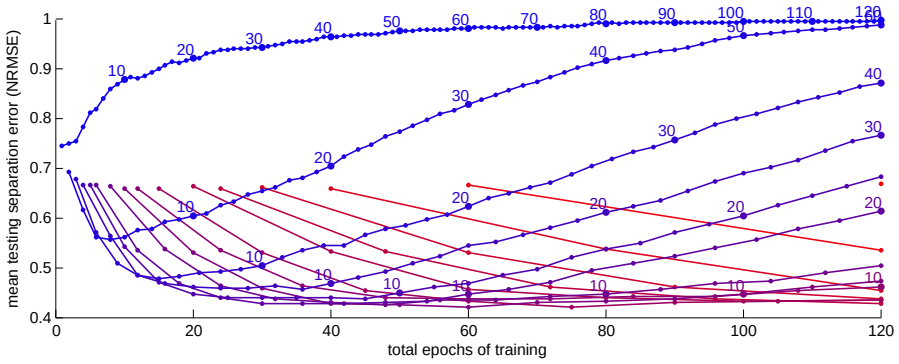

Fig. 3. Mean testing separation errors in layers of differently trained hierarchies plotted against the total epochs of training. See text for the details.

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.

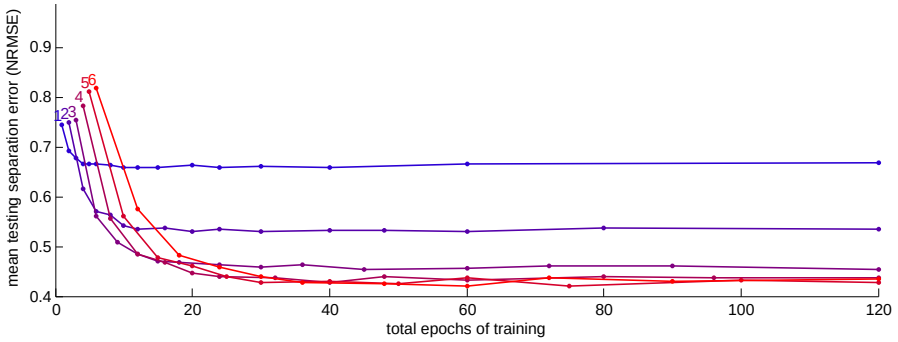

Fig. 4. Errors in the first six layers across differently trained hierarchies plotted against the total epochs of training. See text for the details.

In Figure 4 we look at the data from a different angle: we connect the dots representing layers of the same level across differently trained hierarchies. Here each of the six curves represents a hierarchy with a fixed number of layers. We can see that putting more effort in unsupervisedly training a single layer (the top-most blue curve) does improve the performance but only to a point. The additional layers are able to break this performance saturation achieving much smaller separation errors. This shows that with the same total number of training epochs hierarchical architectures clearly outperform a single reservoir.

The same hierarchical training with the concatenated USPS data was only a partial success, at least with the small $N_x = 50$ reservoirs: the performance already peaks at the second layer. Due to the richer nature of data, such bottom reservoirs are most probably inadequate to learn good representations, and thus we get a similar effect to the second blue curve in Figure 3.

6 Discussion

We have demonstrated that RBF-RNNs can learn unsupervised representations of the temporal data that enable better results in supervised tasks such as separating repeated slow patterns or recognizing handwritten digits. We have also demonstrated that hierarchies of our unsupervised reservoirs can improve pattern separations by a large margin, even though the data has no longer-term structure. This is in line with the results for static data [2]. Random RBF-RNNs are also introduced as viable alternative to classical ESN reservoirs.

These findings are, however, not universal for just any task. A more complete understanding of the mechanisms and limitations involved is still lacking, but is partially addressed in [12].

Acknowledgment This research was supported by EU FP7 project ORGANIC.

References

1. G. J. Chappell and J. G. Taylor. The temporal Kohonen map. *Neural Networks*, 6(3):441–445, 1993.
2. D. Erhan, Y. Bengio, A. C. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.
3. I. Farkaš and M. W. Crocker. Systematicity in sentence processing with a recursive self-organizing neural network. In *Proceedings of ESANN 2007*, pages 49–54, 2007.
4. B. Hammer, A. Micheli, A. Sperduti, and M. Strickert. Recursive self-organizing network models. *Neural Networks*, 17(8-9):1061 – 1085, 2004.
5. H. Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007.
6. H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.
7. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of ICCV'09*. IEEE, 2009.
8. T. Kohonen and T. Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007.
9. J. Koutník. Inductive modelling of temporal sequences by means of self-organization. In *Proceeding of IWIM 2007*, pages 269–277. CTU in Prague, 2007.
10. A. Lazar, G. Pipa, and J. Triesch. SORN: a self-organizing recurrent neural network. *Frontiers in Computational Neuroscience*, 4(0):12, 2010.
11. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.
12. M. Lukoševičius. *Reservoir Computing and Self-Organized Neural Hierarchies*. PhD thesis, Jacobs University Bremen, Bremen, Germany, 2011.
13. M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.
14. M. W. Mak. A learning algorithm for recurrent radial basis function networks. *Neural Processing Letters*, 2(1):27–31, 1995.
15. T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. *Artificial Neural Networks*, 1:397–402, 1991.
16. A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Proceedings of NIPS 2008*, pages 1313–1320. MIT Press, Cambridge, MA, 2009.
17. A. Saxe, P. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of ICML 2011*, 2011.
18. P. Tiňo, I. Farkaš, and J. van Mourik. Dynamics and topographic organization of recursive self-organizing maps. *Neural Computation*, 18(10):2529–2567, 2006.
19. M. Varsta, J. D. R. Millán, and J. Heikkonen. A recurrent self-organizing map for temporal sequence processing. In *Proceedings of ICANN 1997*, 1997.
20. T. Voegtlin. Recursive self-organizing maps. *Neural Networks*, 15(8-9):979–991, 2002.