Short Term Memory and Pattern Matching with Simple Echo State Networks

Georg Fette, Julian Eggert

2005

Preprint:

This is an accepted article published in ICANN. The final authenticated version is available online at: https://doi.org/[DOI not available]

Short Term Memory and Pattern Matching with Simple Echo State Networks

Georg Fette (*fette@in.tum.de*), Julian Eggert (*julian.eggert@honda-ri.de*)

Technische Universität München; Boltzmannstr. 3, 85748 Garching/München, Germany
HONDA Research I. Europe GmbH; Carl-Legien-Str.30, 63073 Offenbach/Main, Germany

Abstract. Two recently proposed approaches to recognize temporal patterns have been proposed by Jäger with the so called Echo State Network (ESN) and by Maass with the so called Liquid State Machine (LSM). The ESN approach assumes a sort of "black-box" operability of the networks and claims a broad applicability to several different problems using the same principle. Here we propose a simplified version of ESNs which we call Simple Echo State Network (SESN) which exhibits good results in memory capacity and pattern matching tasks and which allows a better understanding of the capabilities and restrictions of ESNs.

1 Introduction

Both, ESN and LSM, are three-layered networks consisting of input/hidden/output layers, which are used in matching tasks using input-output sequences. To solve the task the input signal is fed into a highly recurrent hidden layer. During the presentation of the signals the state-vector of the network's hidden layer is logged. After the complete input vector was fed into the system the logged activity is used to adjust the output weights in a way that the networks output matches the desired output pattern. Whereas the ESN proposed by Jaeger (see e.g. [1]) is a discrete time, nonlinear, recurrent network, the LSM proposed by Maass (see e.g. [2]) is a biologically inspired model with integrate-and-fire neurons and dynamically adapting synapses. Both approaches were used to detect temporal patterns with different time lags [1]-[4] between input and output. For reasons of brevity, in the following we just want to describe the main properties of the two models and refer to the specific publications for the details.

1) They both use a network with an input layer which is connected with a hidden layer. Maass connects about 30% of the hidden layer with the input layer, J¨ager fully connects the input towards the hidden layer. 2) The hidden layer is heavily recurrent. J¨ager uses a fully recurrent hidden layer, whereas Maass uses a spatial connection topology in which the neurons are more densely connected to their local neighbours than to remote neurons and about 10% of the connections are inhibitory. 3) The connectivity in both approaches is stochastic. 4) The weights to the output layer are set with a one-step learning rule, adjusting them in a way that the difference between the actual output and the desired output over the whole training period is minimized. To achieve this goal, both approaches use linear regression on the logged states of the hidden layer and the desired outputs.

2 SESN

The coarse structure of a SESN is similar to that of an ESN, also consisting of three layers. Input layer I contains just a single neuron u. We define $u(t)$ as the input to the network and $\mathbf{u} = [u(0), .., u(t_{max})]^T$ as the vector of inputs in the time interval $t \in [0, t_{max}]$. u is connected to every neuron x_v in the hidden layer $H(|H| = n)$ with weight $i_v = 1\,(v \in [1, n])$. Each unit x_v in H is just connected to itself with weight $d_v \in]0, 1[$ randomly taken from a uniform distribution ($d_i \neq d_j, \forall i, j \in [1, n], i \neq j$). All hidden units are fully connected to the output layer O. The output layer just contains a single neuron. The output weights $\mathbf{w} \in \mathcal{R}^n$ from H to O are set in a single learning step by linear regression as explained later. There are no connections from O to H or to I, so further readout neurons can be added and analyzed in parallel without affecting the rest of the system. All neurons have a linear transfer function. The state of the system is described by the vector $\mathbf{x}(t) = [x_1(t), .., x_v(t)]^T$ of the hidden layer. Every unit x_v is updated according to $x_v(t) = d_v x_v(t-1) + u(t)$. The output unit calculates its value by $o(t) = \mathbf{x}\,\mathbf{w}^T$. The desired output at every time step is defined as $\hat{o}(t)$. The vector $\hat{\mathbf{o}} = [\hat{o}(0), .., \hat{o}(t_{max})]^T$ describes the desired outputs for the time interval $t \in [0, t_{max}]$. When the proper output weights have to be calculated, the input signal $u(t)$ is fed into the system and the state vectors $\mathbf{x}(t)$ are logged forming the matrix $X \in \mathcal{R}^{n \times t_{max}}$

$$X_{v,t} = x_v(t) \tag{1}$$

The pseudoinverse $X^\dagger$ of this matrix is then used to calculate the weights by

$$\mathbf{w} = X^\dagger\,\hat{\mathbf{o}} \tag{2}$$

3 SESNs for Memory Reconstruction

We now want to find the output weights $\mathbf{w}_s$ that allow to recover the past input $u(t - s)$ (shifted by s) at $o(t)$ for arbitrary inputs, so that the overall network acts as a sort of delay line. This task can be differently expressed by: The system shall map the input impulse $\mathbf{u} = [1, 0, .., 0]^T \in \mathcal{R}^{t_{max}}$ to the desired output sequence $\hat{\mathbf{o}}_s = [0, .., 0, 1, 0, .., 0]^T, (o_{s,s} = 1) \in \mathcal{R}^{t_{max}}$. This is a feasible approach because a continuous linear system can be completely described by its pulse response and the signals of an arbitrary input are processed by the system by superposition of the signal's individual composing pulses. For the time-discrete SESNs from 2 this still holds approximately as long as s is of the same order of magnitude as n. With the above assumptions on $\mathbf{u}$ and $\hat{\mathbf{o}}_s$ the matrix X simplifies to $X_{v,t} = {d_v}^t$. The output weights $\mathbf{w}_s$ are then calculated using (2). We call the discrete response $p_s(t)$ of the system on a discrete impulse $\delta(t)$ ($\delta(0) = 1; \delta(x) = 0, x \neq 0$) its kernel, with

$$p_s(t) = \mathbf{d}^t\,\mathbf{w}_s \tag{3}$$

and $\mathbf{d}^t$ the elementwise exponentiation of $\mathbf{d}$. When we now feed the system with an arbitrary input signal $u(t)$ the output unit's response can be directly calculated by folding the input with the kernel so that $o_s(t) = (p_s * u)(t)$ and training time can be reduced

by several orders of magnitude as compared to standard ESNs. Furthermore, we can easily calculate the partial ($mc(s)$) and total (MC) memory capacities for our network configuration. J¨ager defined the total memory capacity [3] as the ability of the network to extract the variation of former input from the system when it is fed with white noise input $u(t)$ [1]:

$$mc(s) = \frac{(cov_t(\hat{o}_s(t), o_s(t)))^2}{\sigma_t^2(\hat{o}_s(t))\,\sigma_t^2(o_s(t))} \quad MC = \sum_{s=0}^{\infty} mc(s) \tag{4}$$

For SESNs it holds that $MC = n$ and

$$mc(s) = p_s(s) \tag{5}$$

(for the proof see appendix of [4]) which means that the kernel p_s at time step s indicates how much of the signal $u(t-s)$ can still be retrieved by $o(t)$. We observe that the maximum peak $p_s(s)$ of the kernel response gets smaller with increasing s, meaning the memory capacity decreases when the time lag grows. This relationship can be seen in the top left plot of the figure at the end of the paper, were we show 3 kernels $p_s(t)$ that resulted from the system. When we equip the hidden neurons with a nonlinear transfer function like $f(x) = \tanh(x)$ the proper kernel p_s cannot be computed in the easy way explained above, but must be calculated extensively by propagating a white noise signal $u(t)$ through the network and using formulas (1), (2) and (3), since (5) does not hold any more. The nonlinearities lower the memory capacity of SESNs, because larger inputs are squashed and cannot be retrieved by the linear output weights in the same way smaller input values are retrieved. As a sigmoid behaves almost linearly for inputs around 0, memory capacity in this case gets dependent on the scaling of the input signal, being maximal for small scale inputs (linear limit) and decreasing for larger inputs. The reduction in memory capacity by non-linearities was already described by J¨ager [3] and can be observed in the bottom left plot of the figure, where we show the memory capacity as a function of the input scaling for linear and nonlinear SESNs. In the top right quadrant of our figure we also show a typical partial memory capacity plot which exhibits a characteristic drop at $t \approx n/4$, so that if a network with almost complete memory reconstruction capability of the last, say, $\hat{s}$ time steps is desired, one would have to use a network of size $n \approx 4\,\hat{s}$.

4 SESNs for Pattern Matching

With the presented mechanism we now want the network to detect an arbitrary binary temporal pattern $g(t) \in \{0,1\}$ ($t \in [0, t_g]$). We feed the network with the pattern of length t_g. At time t_g, after the pattern presentation ends, we want $\hat{o}(t_g) = 1$ at the

[1] $\mu_x(f(x)) = \langle f(i)\rangle_i \quad \sigma_x^2(f(x)) = \langle (f(i) - \mu_x(f(x)))^2\rangle_i \quad cov_x(f(x), h(x)) = \langle f(i)\,h(i)\rangle_i$

output if the learned pattern was presented and otherwise $\hat{o}(t_g) = 0$. To accomplish this task we take a network of proper size (considering the arguments gained from the empirical results for memory reconstruction from the previous section we use $n = 4\,t_g$), so that the ability to recover the pattern signals at the entire length of the presented patterns is almost 1. We now superpose the output weights $\mathbf{w}_s$, for every time shift $s \in [0, t_g]$ for which we want to map the input vector $\mathbf{u} = [1, 0, .., 0]^T$ to the output vector $\hat{\mathbf{o}}_s = [0, .., 0, k_s, 0, .., 0]^T$, with $k_s = \frac{2g(s)-1}{t_g}$. By performing this superposition also the different kernels p_s are summed to a pattern-detecting kernel $\hat{p}$. The summed weight vector $\mathbf{w} = \sum_{s=0}^{t_g} \mathbf{w}_s$ now reacts on g with the output $o(t_g) = 1$ and on every other pattern with a lower excitation. In addition, the systems output is continuous so that small changes from the original pattern only lead to small reductions in the output excitation.

Nevertheless in this mode of operation the system performs approximately as a linear filter after learning. To overcome this penalty without adding non-linearities into the system itself, we can add another pool of hidden units of the same size as the first one, which is connected to a further input neuron, which supplies the square $(u(t))^2$ of the input signal. We also need a further bias unit u_b supplying a permanent bias signal $u_b(t) = 1$ which is connected to the output unit with the weight w_b. If we set the kernel of the first hidden pool to the negative inverse pattern, i.e., $p_1(t) = -2\,g(t - t_g)$, the second pool to $p_2(t) = 1$ and the bias weight to the summed squared pattern values ($w_b = \sum_{i=0}^{t_g} g(i)^2$), we get the following output:

$$o(t) = (u * p_1)(t) + (u^2 * p_2)(t) + \sum_{i=0}^{t_g} g(i)^2 = \sum_{i=0}^{t_g} u(t-i)\,(-2)\,g(t-i) +$$
$$\sum_{i=0}^{t_g} u(t)^2\,1 + \sum_{i=0}^{t_g} g(t-i)^2 = \sum_{i=0}^{t_g} (u(t-i) - g(t-i))^2 = \sum_{i=0}^{t_g} (u(i) - g(i))^2 \quad (6)$$

The resulting total kernel calculates the summed squared distance between the input pattern and the learned pattern. Thus only the learned pattern produces a maximum excitation at the output and every other pattern produces less activity.

Another possibility to make SESN capable to detect arbitrary patterns is to add non-linearities into the system itself by introducing non-linear activation functions. To train the network with $n = 100$ the system is fed by white noise in which every 50 time steps a pattern $g(t) = -(-2+\frac{4\,t}{t_g})^2$ with $t_g = 20$ is inserted. Again only at the end of a pattern presentation the output shall be 1 and otherwise 0. The size of the training set was 2000 time steps. The calculation of $\mathbf{w}_s$ has to be done by formulas (1) and (2). As can be observed in the figure bottom right where as non-linearity was introduces by a transfer function which depolarizes the neurons after the treshold 2 is surpassed performance rises significantly. This suggests that the enhanced capabilities of the nonlinear network originate from exploiting the nonlinearities in a way that the network computes higher orders of the input signal internaly and computes a similar distance measure as in (6).

5 Comparing SESN to ESN

When we want to compare ESNs with SESNs it is better to describe the differences between the two models, because they are very similar: 1) In standard ESNs the hidden units have a non-linear, sigmoid transfer function. As we have seen, by introducing non-linearities in SESNs, the memory capacity is reduced with respect to the input scaling, but on the other hand the ability to match patterns is enhanced. In our experiments we have seen that pattern matching performance again decreases, when the input is drastically scaled down [4]. 2) In ESNs the input weights are drawn randomly from a normal distribution. In linear SESNs the input weights to each hidden neuron can be multiplied into the output weight and the input weight itself can be set to 1. 3) In ESNs the hidden units are highly interconnected with each other. A linearized version of the ESN can straightforwardly be transformed into a SESN by diagonalizing the ESNs hidden unit connectivity matrix and multiplying the resulting matrices D and D^{-1} which occur during this process into the input-output-signals. In SESNs, there is no interconnectivity between the units. 4) In ESNs all recurrent connections are drawn randomly from a normal distribution, whereas SESNs get their recurrent connections as specified in section 2.

We have seen that linear SESNs have the same (maximal) memory capacity as the (also linear) ESNs, as shown in the figure bottom left and proved in [3] and [4]. We have also seen that nonlinear SESNs and nonlinear ESNs behave very similarly in pattern matching tasks, suggesting that the ESN learning procedure may select the weights in a way that the ESNs are effectively reduced to the simplified network structures as suggested in this paper.

6 Comparing SESN to LSM

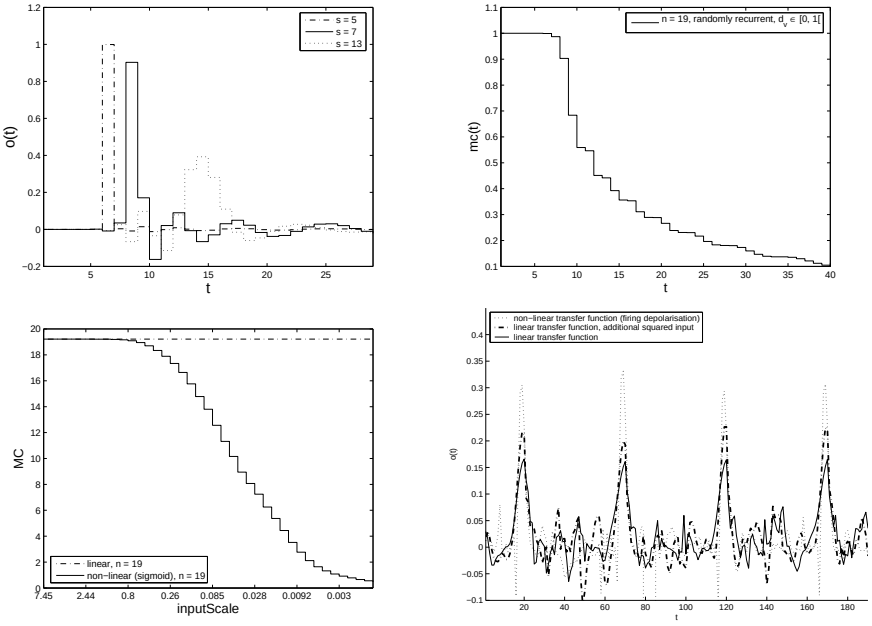
LSMs are based on a continuous model with biologically inspired integrate-and-fire neurons. Therefore they are influenced by heavy non-linearities from different sources (e.g. depolarization after firing or saturation of membrane potentials). As the non-linear transformations of incoming signals can only be examined with great difficulties, our conclusions about LSM are based mostly on empirical results.

For comparison, we have implemented SESNs using a continuous time model in which the hidden layer is replaced by a layer of neurons which are just connected with their input and their output and no other (also no recurrent) connections. Each neuron was equipped with a membrane constant that matched that of typical integrate-and-fire neurons from the LSMs (since the recurrent connection weights d define a kind of membrane constant τ in the discrete, linear SESN model). Non-linear effects lead to a memory capacity reduction. Therefore connections between the neurons with their non-linear signal exchanging mechanism would result in a loss of memory capacity. On the other hand, as non-linearities help in pattern matching tasks, a small number of non-linearities proved to be very useful. In figure bottom right we can see that the introduction of a non-linearity by depolarizing the neurons membrane potential when a threshold is surpassed is useful to improve the capability to recognize patterns. Again, a SESN implementation with a hidden layer of integrate-and-fire neurons which aren't

connected at all with each other performed quite well in memorization as well as in pattern matching tasks in our experiments [4], compared to simulations with LSMs.

7 Conclusions

To better understand the working principles of networks of the ESN and LSM type, we have introduced a recurrent network model called Simple Echo State Network which in comparison has a very reduced complexity and learning costs. In particular the emphasized role of the recurrent connectivity and the nonlinearities can be nicely studied with SESNs. Since SESNs perform comparably well to both ESNs and LSMs on memorization tasks in the linear operation mode and on pattern matching tasks in the nonlinear operation mode, we suggest that they provide an insight into understanding the properties of the two other, much more complex models, and that they give some hints on detecting the main mechanisms leading to their performance.



References

1. H. Jäger, The echo state approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.
2. W. Maass, T. Natschläger, H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation 14, 2002.
3. H. Jäger, Short term memory in echo state networks. GMD Report 152, GMD - German National Research Institute for Computer Science, 2002.
4. G. Fette, Signalverarbeitung in Neuronalen Netzen vom Typ Echo State Networks, diploma thesis (german), 2004 (http://atknoll1.informatik.tu-muenchen.de:8080/tum6/people/fette).