ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

Non-Markovian Processes Modeling with Echo State Networks

Xavier Dutoit[1], Benjamin Schrauwen[2], Hendrik Van Brussel[1] *

1- Katholieke Universiteit Leuven - Mobile Learning Robots Research Group
Celestijnenlaan 300b, 3000 Leuven - Belgium

2- Universiteit Gent - Dept of Electronics and Information Systems
Sint Pietersnieuwstraat 41, 9000 Gent - Belgium

Abstract. Reservoir Computing (RC) is a relatively recent architecture for using recurrent neural networks. It has shown interesting performance in a wide range of tasks despite the simple training rules. We use it here in a logistic regression (LogR) framework. Considering non-Markovian time series with a hidden variable, we show that RC can be used to estimate the transition probabilities at each time step and also to estimate the hidden variable. We also show that it outperforms classical LogR on this task. Finally, it can be used to extract invariants from a stochastic series.

1 Introduction

The everyday life is full of phenomena which, most of the times, can be described by a *process* which undergoes a series of discrete and finite *states*.

With those phenomena, it is interesting to try and estimate what the next state will be. It is even more informative to assign probabilities to all the possible next states at a given time. Logistic regression (LogR) is a mathematical tool which, given an actual series of states, computes the probabilities associated with each state transition.

This work is a preliminary study on using Reservoir Computing (RC) [1, 2, 3, 4] with LogR. The idea underlying RC is to use a large recurrent neural network and to train only the output layer. This training consists here of LogR in order to estimate the transition probabilities of a non-Markovian process. When trained on the reservoir activity, LogR can use the short-term memory and the non-linearities introduced by the reservoir.

2 Task

We consider stochastic time series where a process is at each time step $n = 1, \ldots, N_n$ in a state $s[n] \in \mathcal{S} = \{s_1, s_2, \ldots, s_{N_s}\}$. The transition from a state $s[n]$ to the next state $s[n+1]$ is described by a set of memory-dependent rules $\mathcal{R}$ and, if no rule applies, by a transition probability. A rule consists of a set of 3 states $\{s_i, s_j, s_k\}$. If the system was in the first two states s_i and s_j consecutively, then the next state will be s_k with probability 1[a]. If, for the current states, no

*This work was partially funded by the FWO Flanders project G.0317.05.

[a]Note that there are no incompatible rules in $\mathcal{R}$: if $\{s_i, s_j, s_k\} \in \mathcal{R}$, then $\forall l \neq k : \{s_i, s_j, s_l\} \notin \mathcal{R}$; so there are at most N_s^2 rules.

rule applies, then the transition probability depends only on the current state and on a binary hidden state $\tilde{s}[n] \in \{0, 1\}$. The transition probability is one out of two predefined probabilities, depending on the current hidden state. This hidden state is described by a Poisson random variable: at time step 1, it is 0 or +1 with equal probability, and the probability that it stays in this state until time step n is given by $e^{-\mu} \cdot \mu^n / n!$; if the state changes, the probability that it stays in this new state for another n time steps is given by the same formula.

This could for instance model the (discretized) activities of a human user where the transitions depends also on her/his mood (the hidden state) and some invariants (the predefined rules).

A time series generation is then completely described by two transition matrices P_1 and P_2, a parameter μ and a set of rules. Formally, the probability $P(n, k)$ that the process is in state k at time step n is given by:

$$\begin{aligned}
&P(1,k) := P(s[1] = s_k) = 1/N_s \\
&P(2,k) := P(s[2] = k | s[1] = j) = P_1(j,k) \cdot \tilde{s}[1] + P_2(j,k) \cdot (1 - \tilde{s}[1]) \\
&P(n,k) := P(s[n] = s_k | s[n-1] = s_j, s[n-2] = s_i) = \\
&\quad = \begin{cases} 1 \text{ if } (s_i, s_j, s_k) \in \mathcal{R} \\ 0 \text{ if } \exists l \neq k \text{ such that } (s_i, s_j, s_l) \in \mathcal{R} \\ P_1(j,k) \cdot \tilde{s}[n-1] + P_2(j,k) \cdot (1 - \tilde{s}[n-1]) \text{ otherwise} \end{cases} \qquad \forall n > 2. \qquad (1)
\end{aligned}$$

3 Approach

To estimate the transition probabilities at each time step, we use an Echo State Network (ESN)[1]. An ESN is described by an *input matrix* $\mathbf{W}_{\mathtt{i}}^{\mathtt{r}}$, a *reservoir matrix* $\mathbf{W}_{\mathtt{r}}^{\mathtt{r}}$, a *bias matrix* $\mathbf{W}_{\mathtt{b}}^{\mathtt{r}}$ and an *output matrix* $\mathbf{W}_{\mathtt{o}}^{\mathtt{r}}$. At each time step t, the reservoir activity $\mathbf{r}[t]$ is computed according to:

$$\mathbf{r}[t] = f\big(\mathbf{W}_{\mathtt{i}}^{\mathtt{r}} \cdot \mathbf{i}[t-1] + \mathbf{W}_{\mathtt{r}}^{\mathtt{r}} \cdot \mathbf{r}[t-1] + \mathbf{W}_{\mathtt{b}}^{\mathtt{r}}\big), \qquad (2)$$

where $\mathbf{i}[t]$ is the input at time step t, $f(.)$ is a non-linear function (we use here a hyperbolic tangent), and the initial activity is $\mathbf{r}[0] = \mathbf{0}$. The output $\hat{\mathbf{o}}[t]$ is given by $\hat{\mathbf{o}}[t] = f_{\text{out}}(\mathbf{W}_{\mathtt{o}}^{\mathtt{r}} \cdot \mathbf{r}[t])$, where we use LogR as f_{out} (see later, Equation (4)). For more details on the reservoir implementation, please refer for instance to [1].

It has been seen in the experiments that the performance could be improved by using the ESN at a faster rate than the process. In practice, the same input is repeated for t_{ext} time steps. We will thus distinguish process time steps $n = 1, 2, \ldots, N_n$ and reservoir time steps $t = 1, 2, \ldots, N_n \cdot t_{\text{ext}}$. At reservoir time step t, the input $\mathbf{i}[t]$ encodes the state $s[n]$ of process time step $n = \lceil t/t_{\text{ext}} \rceil$[b]. Reciprocally, a state $s[n]$ is encoded in inputs $i[(n-1) \cdot t_{\text{ext}} + 1]$ to $i[n \cdot t_{\text{ext}}]$.

With this trick, the reservoir can get closer to the attractor corresponding to the current process state. However, the reservoir activity should include information not only about the current process state, but also about the past

[b] $\lceil x \rceil$ denotes the smallest integer i such that $x \leq i$.

P_1	P_2	$\mathcal{R}$
$\begin{bmatrix} 0.0 & 0.9 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.3 & 0.0 & 0.7 & 0.0 \\ 0.0 & 0.3 & 0.7 & 0.0 & 0.0 \\ 0.9 & 0.0 & 0.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.4 & 0.6 & 0.0 \end{bmatrix}$	$\begin{bmatrix} 0.0 & 0.2 & 0.8 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.8 & 0.0 & 0.2 \\ 0.7 & 0.0 & 0.0 & 0.3 & 0.0 \\ 0.0 & 0.7 & 0.0 & 0.3 & 0.0 \\ 0.4 & 0.0 & 0.0 & 0.0 & 0.6 \end{bmatrix}$	$\begin{matrix} \{4,2,1\} \\ \{3,3,2\} \\ \{4,5,3\} \\ \{3,1,4\} \\ \{2,1,5\} \end{matrix}$

Table 1: Transition matrices and set of rules used to generate the time series

state (as the process transitions depend on the last 2 states); so one should also not get too close to an attractor, which would mean loosing all memory.

The input $\mathbf{i}[t]$ is a $(N_s) \times 1$ column vector representing the state $s[\lceil t/t_{\text{ext}} \rceil]$ ($i_k[t]$ is 1 if $s[\lceil t/t_{\text{ext}} \rceil] = k$, and -1 otherwise).

Reciprocally, the reservoir activity is down-sampled to align it back with the original process. For each process time step n, the corresponding reservoir activity are $\mathbf{r}[(n-1) \cdot t_{\text{ext}} + 1]$ to $\mathbf{r}[n \cdot t_{\text{ext}}]$. The first $t_{\text{ext}} - 1$ samples are discarded, and the down-sampled reservoir activity $\tilde{\mathbf{r}}[n]$ corresponding to state $s[n]$ is defined as $\tilde{\mathbf{r}}[n] = \mathbf{r}[n \cdot t_{\text{ext}}]$.

As the process is stochastic, the goal is not to estimate the next state, but rather the probability associated to each possible next state. The output is then computed by LogR [5, 6]. The main advantage of the ESN approach (and of RC in general) is that only the *output matrix* $\mathbf{W}^{\circ}_{\mathtt{r}}$ is trained (the other matrices are created randomly beforehand). It is trained by minimizing the penalized negative log-likelihood:

$$\mathbf{W}^{\circ}_{\mathtt{r}} = \arg\min_{\mathbf{W}} \left(-\log \Big(\prod_{n=1}^{N_n} \hat{p}(s[n], \tilde{\mathbf{r}}[n], \mathbf{W}^{\circ}_{\mathtt{r}}) \Big) + \lambda \sum_{k=1}^{N_s - 1} \|\mathbf{W}_k\|^2 \right), \tag{3}$$

where λ is a regularization parameter and $\hat{p}(s[n], \tilde{\mathbf{r}}[n], \mathbf{W}^{\circ}_{\mathtt{r}})$ is defined as:

$$\hat{p}(k, \tilde{\mathbf{r}}[n], \mathbf{W}^{\circ}_{\mathtt{r}}) = \frac{\exp\Big(\mathbf{W}^{\circ}_{\mathtt{r},k} \cdot \tilde{\mathbf{r}}[n]\Big)}{1 + \sum_{i=1}^{N_s - 1} \exp\Big(\mathbf{W}^{\circ}_{\mathtt{r},i} \cdot \tilde{\mathbf{r}}[n]\Big)} := \hat{P}(n,k) \; \forall k < N_s,$$

$$\text{and } \hat{p}(N_s, \tilde{\mathbf{r}}[n], \mathbf{W}^{\circ}_{\mathtt{r}}) = 1 - \sum_{i=1}^{N_s - 1} \hat{p}(i, \tilde{\mathbf{r}}[n], \mathbf{W}^{\circ}_{\mathtt{r}}) := \hat{P}(n, N_s), \tag{4}$$

$\mathbf{W}^{\circ}_{\mathtt{r},k}$ being the k-th row of $\mathbf{W}^{\circ}_{\mathtt{r}}$. Training the output matrix can then be done by using a iteratively re-weighted least square algorithm, as in e.g. [7].

4 Results

We consider time series of length $N_n = 100$ with $N_s = 5$ different states described by the two transition matrices and the set of 5 rules in Table 1 (and with $\mu = 10$).

At each time step, the goal is to estimate the transition probabilities to each of the possible next states. The training is done with 10 different time series; 8 are effectively used for training, one is used as a validation error to optimize the regularization parameter λ, and one is used for testing.

To account for the randomness in the creation of the reservoir, several different reservoirs of 50 neurons are created[c]. Several values have been tested for the t_{ext} parameter. For each possible value, 25 reservoirs have been tested. The average negative log-likelihood as a function of t_{ext} is shown in Figure 1(a), with the bars denoting the standard deviation. For the optimal value $t_{\text{ext}} = 5$, the average negative log-likelihood is 0.85 (with a standard deviation of 0.06).

When t_{ext} is increased, the error first decreases and then increases again. When the same input is repeated for several time steps, the reservoir activity goes toward an attractor. Most of the time, no rule applies and the probability transition depends only on the current state (and on the hidden state). So most of the time, the attractor corresponding to the current state can be mapped to the desired output. However, when a rule does apply, it is necessary to have information about the past state as well. There is thus a trade-off, which can be seen in the error curve.

As a baseline, we try two other approaches. In the first one, no ESN is used and LogR is directly applied to the process state (later called Direct LogR or DLR). To have a more fair comparison, as the ESN has 50 neurons, the last 50 states are used as input, i.e. we use use the estimate $\hat{p}(k, \mathbf{I}_{50}[n], \mathbf{W})$ rather than $\hat{p}(k, \tilde{\mathbf{r}}[n], \mathbf{W}))$ (where $\mathbf{I}_{50}[n]$ is the vertical concatenation of $\mathbf{i}[n]$ to $\mathbf{i}[n-49]$). In the second one, an ESN is used but the readout is computed with standard Linear Regression (LinR). Although LinR is not meant to output probabilities but rather exact values (and nothing ensures that the LinR output will lie between 0 and 1), it is interesting to apply the same error measures as with LogR.

With the ESN, the expected values of $\hat{P}(n,k)$ averaged over the time steps when a fixed rule applies[d] and averaged over the time steps where no rule applies are very different. The expected values, averaged over all the states and all the time samples of a test run, are plotted in Figure 1(b). The upper plot is the expected value when a rule applies, and the lower one when no rule applies. Whenever a rule applies, the estimation of the ESN is close to one. In fact, $\hat{P}(n,k)$ is larger than 0.9 in 87% of the cases when a rule applies (whereas with DLR, the output is larger than 0.9 in only 31% of the cases and in 74% of the cases with LinR).

For $t_{\text{ext}} = 5$, the optimal regularization parameter is $2 \cdot 10^{-3}$. The output on a test sample of the process of a reservoir trained with such parameter is given in Figure 1(c). The solid lines represent the estimations $\hat{P}(n,k)$, and the dots the actual transition probabilities $P(n,k)$, for each of the 5 different states.

[c]A reservoir is created as follows: each element of $\mathbf{W}^{\mathtt{r}}_{\mathtt{r}}$ is sampled from a normal distribution, and the whole matrix is then rescaled to have a spectral radius of 0.99; each element of $\mathbf{W}^{\mathtt{r}}_{\mathtt{i}}$ and $\mathbf{W}^{\mathtt{r}}_{\mathtt{b}}$ is drawn from a 0-mean standard distribution with variance 0.1 and 0.01, respectively.

[d]i.e. time steps n s.t. $(s[n-2], s[n-1], s[n]) \in \mathcal{R}$, which represent 18.63% of the time steps in the test samples used here.

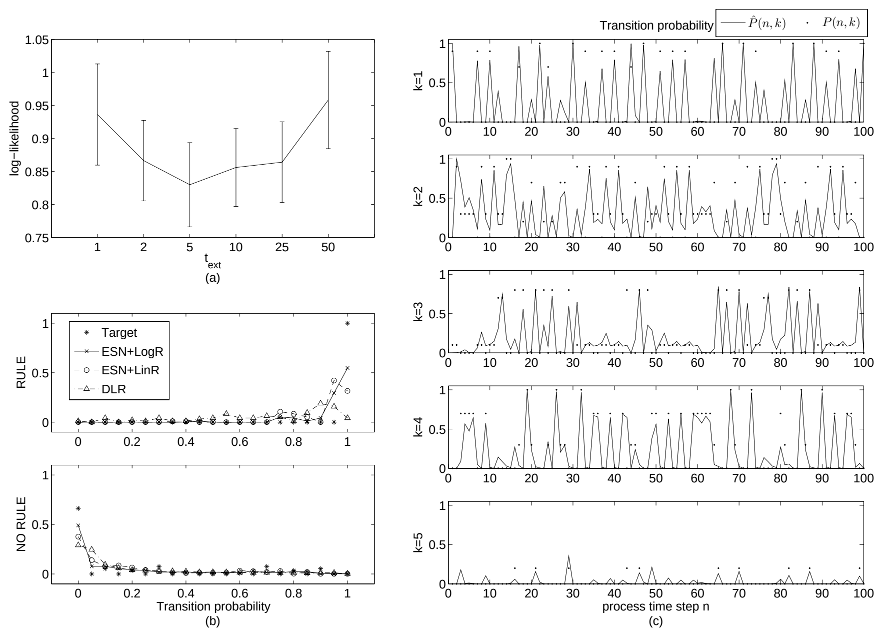

Fig. 1: (a) average negative log-likelihood as a function of t_{ext} (mind the y axis). (b) expected value of actual (stars) transition probability and the estimated ones (LogR: crosses, LinR: circles, DLR: triangles), when a rule applies (upper plot) or not (lower plot). (c) Estimation from the reservoir (solid line) and actual transition probability (dots), for each of the 5 different states.

We can see that the estimation is in general close to the actual probability. It is then possible, given only a few samples of a random process (in this case 10 sample of 100 transitions), to estimate correctly the actual transition probabilities underlying the process generation.

The negative log-likelihood, averaged over 10 new test time series, is 0.82. It is 1.24 for DLR and 1.06 for LinR. Another interesting error measure, when the actual probability distribution function $P(n, k)$ is known, is the Wasserstein distance $d_W(n) = \sum_{k=1}^{N_s} |\sum_{j=1}^{k} P(n, k) - \sum_{j=1}^{k} \hat{P}(n, k)|$. The average Wasserstein distance is $6.31 \cdot 10^{-2}$ (and $7.3 \cdot 10^{-2}$ with LinR and $1.26 \cdot 10^{-1}$ with DLR).

Moreover, the most likely estimated state corresponds with the actual most likely state[e] 79% of the time with LogR, 79.2% with LinR and 59% with DLR, and it corresponds to the actual state[f] 66.3% of the time with LogR, 68% with LinR and 54.5% with DLR (the actual state corresponds with the actual most

[e]i.e. $\arg\max_k P(n, k) = \arg\max_k \hat{P}(n, k)$

[f]i.e. $s[n] = s_{\arg\max_k \hat{P}(n,k)}$

likely state[g] 83.3% of the time). So ESN-based techniques can be used to estimate the next state of such a time series.

5 Conclusion

We considered time series where the transition from one state to the next one depends on the current time and also on a set of predefined rules. We considered 3 different techniques: an ESN with a readout trained with LogR, an ESN with a readout trained with LinR and a direct application of LogR on a windowed version of the input data.

When an ESN is used, it is possible to estimate the transition probabilities with a low error even when they are complex and involve memory and fixed rules. Moreover, it is possible to detect invariant features, i.e. events happening with a probability equal to 1. On the other hand, a direct application of LogR on the input is unable to detect such invariants.

To the best of the knowledge of the authors, this is the first time LogR is applied to an ESN. This is just a preliminary approach and the results are only slightly better than classical LinR. However, it shows than ESN can be used to estimate probabilities rather than deterministic values. ESN with LogR training could then be applied to a broader class of problems than the ones considered so far.

In the experiments, we have seen that the results can be improved by "extending" the reservoir time with respect to the actual time series time. This could possibly be done in a more natural way by using layered ESNs [8] or by increasing the weights of the input matrix. Both directions should be investigated in the future.

References

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

[2] W. Maass, T. Natschläger, and H. Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14:2531–2560, 2002.

[3] J. J. Steil. Backpropagation-Decorrelation: online reccurent learning with O(N) complexity. *Proceedings of the International Joint Conference on Neural Networks*, 1:843–848, 2004.

[4] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 4 2007.

[5] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, 1989.

[6] F.C. Pampel. *Logistic Regression: a Primer*. Sage, 2000.

[7] P. Karsmakers, K. Pelckmans, and J. Suykens. Multi-class kernel logistic regression: a fixed-size implementation. In *Proceedings of the IJCNN 2007*, pages 1–4, 2007.

[8] M. Lukoševičius. Echo State Networks with Trained Feedbacks. Technical report, Jacobs University Bremen, 2007.

[g] i.e. $s[n] = s_{\arg\max_k P(n,k)}$