Echo State Incremental Gaussian Mixture Network for Spatio-Temporal Pattern Processing

Rafael C. Pinto[1], **Paulo M. Engel, Milton R. Heinen**

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{rcpinto,engel,mrheinen}@inf.ufrgs.br

Abstract. *This work introduces a novel neural network algorithm for online spatio-temporal pattern processing, called Echo State Incremental Gaussian Mixture Network (ESIGMN). The proposed algorithm is a hybrid of two state-of-the-art algorithms: the Echo State Network (ESN), used for spatio-temporal pattern processing, and the Incremental Gaussian Mixture Network (IGMN), applied to aggressive learning in online tasks. The algorithm is compared against the conventional ESN in order to highlight the advantages of the IGMN approach as a supervised output layer.*

Resumo. *Este trabalho introduz um novo algoritmo de redes neurais para processamento online de padrões espaço-temporais, chamado Echo State Incremental Gaussian Mixture Network (ESIGMN). O algoritmo proposto é um híbrido de dois algoritmos estado-da-arte: a Echo State Network (ESN), usada para processamento de padrões espaço-temporais, e a Incremental Gaussian Mixture Network (IGMN), aplicada ao aprendizado agressivo em tarefas online. O algoritmo é comparado com a ESN convencional a fim de destacar as vantagens da abordagem IGMN como camada supervisionada de saída.*

1. Introduction

This work presents a novel neural network algorithm for online spatio-temporal pattern processing, called Echo State Incremental Gaussian Mixture Network (ESIGMN). The proposed algorithm is a hybrid of two state-of-the-art algorithms: the Echo State Network (ESN) [Jaeger 2001], based on the Reservoir Computing (RC) paradigm, used for spatio-temporal pattern processing, and the Incremental Gaussian Mixture Network (IGMN, previously known as Incremental Probabilistic Neural Network or IPNN on early versions) [Heinen and Engel 2010] [Heinen 2011], applied to aggressive learning of online tasks.

The ESN is a recurrent neural network which can be applied to spatio-temporal supervised learning. While being able to learn incrementally through Recursive Least Squares (RLS) [Hayes 1996] [Jaeger 2003] or conventional Least Mean Squares (LMS) [Widrow 1966], it is really powerful in batch mode, i.e. when all training data is given at once. Its incremental versions, which can be applied to online tasks, learn slowly, requiring several presentations of each pattern.

The IGMN, on the other hand, is an one-shot incremental learning algorithm, needing only a single scan through the training data in order to build a consistent model. But the IGMN is essentially a static algorithm, meaning that it basically works only for static tasks, where any pattern is independent of past ones.

The ESIGMN merges those two algorithms, giving the best of the two worlds. The ESN gives temporal processing capabilities to the IGMN, while the IGMN gives one-shot incremental learning capabilities to the ESN.

The algorithm is compared against the conventional ESN in order to highlight the advantages of the IGMN approach as a supervised output layer, instead of the conventional ESN output layer.

This work is structured as follows: In section 2, the Reservoir Computing (RC) approach is described, presenting the ESN as an instantiation of that approach. The IGMN is described in section 3. Section 4 presents the new algorithm, the ESIGMN. In section 5, the ESIGMN is compared to a conventional ESN in time-series prediction tasks. Section 6 finishes this work with concluding remarks about the new algorithm and future works.

2. Reservoir Computing

Reservoir Computing (RC) is a recently coined term for a neural pattern processing paradigm where a random, non-linear, fixed and large hidden layer with recurrent connections, called a reservoir, is used as an excitable medium where interesting dynamic features of the data stream can be extracted. It's similar to a random filter bank, producing transformations over the input data. Albeit the reservoir not being trained, its output states are sufficient to train linear regression / classification algorithms on non-linear dynamic tasks successfully, thus potentially turning any static linear algorithm into a non-linear dynamic one. But since the reservoir is random, large reservoirs are required, to increase the chances of obtaining useful transformations.

This paradigm was found independently by different researchers at different times, also in distinct research fields like computational neuroscience and machine learning: Temporal Recurrent Neural Network [Dominey 1995]; Liquid State Machines [Natschläger et al. 2002]; Echo State Networks [Jaeger 2001]; Decorrelation-Backpropagation Learning [Steil 2004].

This work will incorporate the ESN as the base for the new algorithm, since it is composed by the default neuron model used in artificial neural networks (like the Multi-Layer Perceptron), is very simple to implement, is probably the most widely used RC artificial neural network in computer science and gave excellent results in previous works, e.g., predicting chaotic dynamics (three orders of magnitude improved accuracy [Jaeger and Haas 2004]), nonlinear wireless channel equalization (two orders of magnitude improvement [Jaeger and Haas 2004]), the Japanese Vowel benchmark (zero test error rate, previous best was 1.8% [Jaeger et al. 2007]), financial forecasting (winner of the international forecasting competition NN3), and in isolated spoken digits recognition (improvement of word error rate on benchmark from 0.6% of previous best system to 0.2% [Verstraeten et al. 2006]).

2.1. The Echo State Network

The ESN consists basically of an input layer, a reservoir and an output layer. The weights between input layer and reservoir (here denoted as $\mathbf{W}_{in}$) as well as the recurrent reservoir weights ($\mathbf{W}$) are randomly chosen and fixed – no training is necessary. The weights from input layer and reservoir to the output layer ($\mathbf{W}_{out}$) are trained in batch mode by linear Least Squares Fitting, or in incremental mode by Recursive Least Squares (RLS)

or any other incremental linear learning algorithm, like conventional Least Mean Squares (LMS). This work will focus on the incremental mode, since we need to compare the ESN to an incremental algorithm (ESIGMN). The reservoir activation function ($f(.)$) is the hyperbolic tangent, and the output layer activation function ($g(.)$) is usually the identity, although any differentiable function could be used. An example of an ESN can be seen in figure 1.

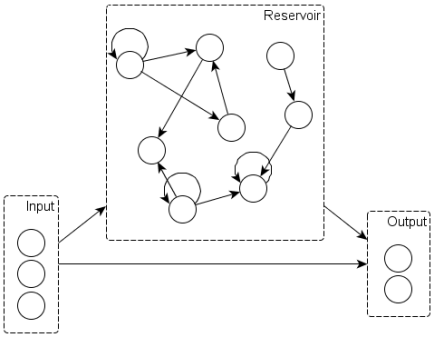


Figure 1. An example of Echo State Network with 3 input nodes, 2 output nodes and 8 reservoir nodes. All weights going to the output layer (W_{out}) must be trained, while every other weight is fixed.

In order to have a stable reservoir, however, the echo state property must be assured. It means that when a null vector is continually fed into the reservoir as its input, its state vector must decay to a null vector too, as time tends towards infinity. In practical terms, this can be ensured by rescaling the largest absolute eigenvalue $|\lambda_{max}|$ of the reservoir recurrent weight matrix $\mathbf{W}$, i.e. its spectral radius, to a value in the interval (0, 1). There is controversy about the necessity or sufficiency of this condition to ensure the echo state property [Buehner and Young 2006], but it's known to work well in practice [Jaeger 2001]. A proven and stronger sufficient condition is to rescale the largest singular value of the weight matrix to the interval (0, 1) [Jaeger 2001].

Following, a brief explanation of the ESN algorithm is shown. Given input $\mathbf{u}$ at time t, a processing step of the ESN is computed as follows:

The new state $\mathbf{x}$ of the reservoir at time t is obtained by the following equation:

$$\mathbf{x}(t) = f(\mathbf{W}_{in}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1)) \tag{1}$$

while the output $\mathbf{y}$ at time t of the ESN is given by (assuming identity activation function):

$$\mathbf{y}(t) = \mathbf{W}_{out}[\mathbf{u}(t); \mathbf{x}(t)] \tag{2}$$

where [.;.] is the vector concatenation operation. The prediction error $\mathbf{e}$ at time t is given by:

$$\mathbf{e}(t) = \mathbf{d}(t) - \mathbf{y}(t) \tag{3}$$

where $\mathbf{d}(t)$ is the target vector (teacher signal) at time t. The output weights $\mathbf{W}_{out}$ are modified, in the online case, by stochastic gradient descent as follows:

$$\Delta\mathbf{W}_{out} = \eta\mathbf{e}(t)[\mathbf{u}(t); \mathbf{x}(t)]^T \tag{4}$$

where η is a learning rate in the interval [0, 1].

3. Incremental Gaussian Mixture Network

The IGMN (Incremental Gaussian Mixture Network) algorithm [Heinen and Engel 2010] uses an incremental approximation of the EM algorithm [Dempster et al. 1977], the IGMM (Incremental Gaussian Mixture Model) [Engel and Heinen 2011]. It creates and continually adjusts probabilistic models consistent to all sequentially presented data, after each data point presentation, and without the need to store any past data points. Its learning process is aggressive, or "one-shot", meaning that only a single scan through the data is necessary to obtain a consistent model.

IGMN (and IGMM) adopts a gaussian mixture model of distribution components (known as a *cortical region*) that can be expanded to accommodate new information from an input data point, or reduced if spurious components are identified along the learning process. Each data point assimilated by the model contributes to the sequential update of the model parameters based on the maximization of the likelihood of the data. The parameters are updated through the accumulation of relevant information extracted from each data point.

Differently from IGMM, however, the IGMN is capable of supervised learning, simply by assigning any of its input vector elements as outputs (any element can be used to predict any other element). This architecture is depicted on figure 2.

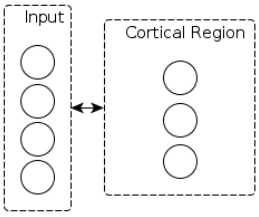


Figure 2. An example of IGMN with 4 input nodes and 3 gaussian components. Any input element can be predicted by using any other element, which means that the input vector can actually be divided into input and output elements.

3.1. Learning

The algorithm starts with no components, which are created as necessary (see subsection 3.2). Given input $\mathbf{x}$, the IGMN algorithm processing step is as follows. First, the likelihood for each component j is calculated:

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right) \tag{5}$$

where D is the input dimensionality, $\boldsymbol{\mu}_j$ the j^{th} component mean and $\mathbf{C}_j$ its covariance matrix.

After that, posterior probabilities are calculated for each component as follows:

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{q=1}^{M} p(\mathbf{x}|q)p(q)} \qquad \forall j \tag{6}$$

where M is the number of components. Now, parameters of the algorithm must be updated according to the following equations:

$$v_j(t) = v_j(t-1) + 1 \tag{7}$$

$$sp_j(t) = sp_j(t-1) + p(j|\mathbf{x}) \tag{8}$$

$$\mathbf{e}_j = \mathbf{x} - \boldsymbol{\mu}_j \tag{9}$$

$$\omega_j = \frac{p(j|\mathbf{x})}{sp_j} \tag{10}$$

$$\Delta\boldsymbol{\mu}_j = \omega_j \mathbf{e}_j \tag{11}$$

$$\boldsymbol{\mu}_j(t) = \boldsymbol{\mu}_j(t-1) + \Delta\boldsymbol{\mu}_j \tag{12}$$

$$\mathbf{C}_j(t) = \mathbf{C}_j(t-1) - \Delta\boldsymbol{\mu}_j\Delta\boldsymbol{\mu}_j^T + \omega\left[\mathbf{e}\mathbf{e}^T - \mathbf{C}_j(t-1)\right] \tag{13}$$

$$p(j) = \frac{sp_j}{\sum_{q=1}^{M} sp_q} \tag{14}$$

where sp_j and v_j are the accumulator and the age of component j, respectively, and $p(j)$ is its prior probability.

3.2. Creating New Components

In order to create new components, the cortical region must reconstruct its input **x** based on the posterior probabilities obtained in equation 6. Let **x** be a concatenation of two vectors **a**, the actual input or known part, and **b**, the target / desired value ($\mathbf{x} = [\mathbf{a}; \mathbf{b}]$). Then the reconstruction of **b** given **a** is obtained by the following equation:

$$\hat{\mathbf{b}} = \sum_{j=1}^{M} p(j|\mathbf{a})\boldsymbol{\mu}_{j,b} \tag{15}$$

in the naïve approach (only diagonal covariance matrixes are used), where $\boldsymbol{\mu}_{j,b}$ is just the target part of the jth component's mean vector. Note that it's only an average of the region's means weighted by their posterior probabilities. This approach will be called here ESIGMNn (naïve). The full multivariate version (ESIGMN) is as follows:

$$\hat{\mathbf{b}} = \sum_{j=1}^{M} p(j|\mathbf{a})(\boldsymbol{\mu}_{j,b} + \mathbf{C}_{j,ba}\mathbf{C}_{j,a}^{-1}(\mathbf{a} - \boldsymbol{\mu}_{j,a})) \tag{16}$$

where $\mathbf{C}_{j,ba}$ is the submatrix of the jth component covariance matrix associating the input and output parts of the data, $\mathbf{C}_{j,a}$ is the submatrix corresponding to the input part only and $\boldsymbol{\mu}_{j,a}$ is just the input part of the jth component's mean vector. After reconstructing the input, the reconstruction error can be obtained by:

$$\epsilon = \max_{i \in D} \left[\frac{|x_i - \hat{x}_i|}{X_{max,i} - X_{min,i}} \right] \tag{17}$$

where $X_{max,i}$ and $X_{min,i}$ are the ith column's maximum and minimum values expected for the entire dataset (just approximated values are ok, since IGMN-based algorithms don't require availability of the entire dataset beforehand). If there are no components or ϵ is greater than a manually chosen threshold ϵ_{max} (e.g., 0.1), then a new component is created and initialized as follows:

$$\boldsymbol{\mu} = \mathbf{x}; \qquad sp = 1; \qquad v = 1; \qquad p(j) = \frac{1}{\sum_{i=1}^{M} sp_i}; \qquad \mathbf{C} = \sigma_{ini}^2$$

where M already includes the new component and σ_{ini} can be obtained by:

$$\sigma_{ini} = diag(\delta[X_{max} - X_{min}]) \tag{18}$$

where δ is a manually chosen scaling factor (e.g., 0.1) and diag returns a diagonal matrix having its input vector in the main diagonal.

3.3. Removing Spurious Components

A component j is removed whenever $v_j > v_{min}$ and $sp_j < sp_{min}$, where v_{min} and sp_{min} are manually chosen (e.g., 5.0 and 3.0, respectively). In that case, also, $p(q)$ must be adjusted for all $q \in M$, $q \neq j$, using equation 14.

3.4. Recalling

In IGMN, any element can be predicted by any other element. This is done by reconstructing data from the target elements (**b**) by estimating the posterior probabilities using only the input elements, as follows:

$$p(j|\mathbf{a}) = \frac{p(\mathbf{a}|j)p(j)}{\sum_{q=1}^{M} p(\mathbf{a}|q)p(q)} \qquad \forall j \tag{19}$$

It's similar to equation 6, except that it uses the actual input vector **a** instead of the full **x** vector, with the target elements **b** removed from calculations. After that, **b** can be reconstructed using equation 15 or 16.

4. The Echo State Incremental Gaussian Mixture Network

The ESIGMN is a temporal extension of the IGMN algorithm, which is augmented with an ESN-style reservoir between its input and cortical region. The reservoir is responsible

for mapping the input space into a feature space which captures temporal dynamics of the data, as with the ESN. But instead of feeding the input and the reservoir state into a linear output layer, they are fed into an unmodified IGMN in the ESIGMN algorithm. The IGMN does its usual spatial processing, but its inputs already incorporate temporal information. This architecture can be seen in figure 3.

Therefore, all IGMN equations from the previous section apply, with a difference only in the source of the IGMN input. In ESIGMN, the input data $\mathbf{u}$ is divided into input $\mathbf{u}_a$ and output/target elements $\mathbf{u}_b$, and is processed in the following way:

$$\mathbf{s}(t) = f(\mathbf{W}_{in,r}\mathbf{u}_a(t) + \mathbf{W}\mathbf{s}(t-1)) \tag{20}$$

$$\mathbf{x}(t) = [\mathbf{s}(t); \mathbf{u}(t)] \tag{21}$$

where $\mathbf{s}$ is the reservoir state (previously known as $\mathbf{x}$ in section 2), $\mathbf{u}_a$ is the known part of the input (the actual input portion of the data), excluding the target values $\mathbf{u}_b$, and $\mathbf{W}_{in,r}$ are the input weights (without the target values) to the reservoir (equation 20 is analogous to equation 1 for the ESN). The IGMN is trained by receiving $\mathbf{x}$ (the concatenation of the full input and reservoir state) as its input. By omitting $\mathbf{u}_b$ and using equation 19 for recalling, it's possible to predict the output $\mathbf{u}_b$.

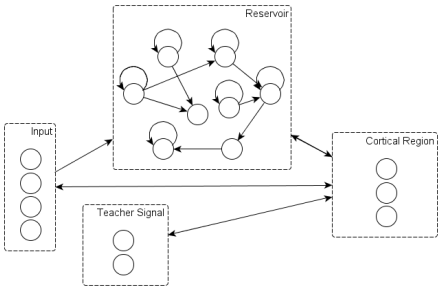


Figure 3. An example of ESIGMN with 4 inputs ($\mathbf{u}_a$), 2 targets/outputs ($\mathbf{u}_b$), 3 gaussian components and 8 reservoir neurons. Only the actual input portion of the data is used to update the reservoir, while the full input data (with target values / teacher signal) is fed into the IGMN together with the reservoir state s in order to train it. By omitting $\mathbf{u}_b$, the IGMN can be used in recalling mode to predict $\mathbf{u}_b$.

5. Experiments and Results

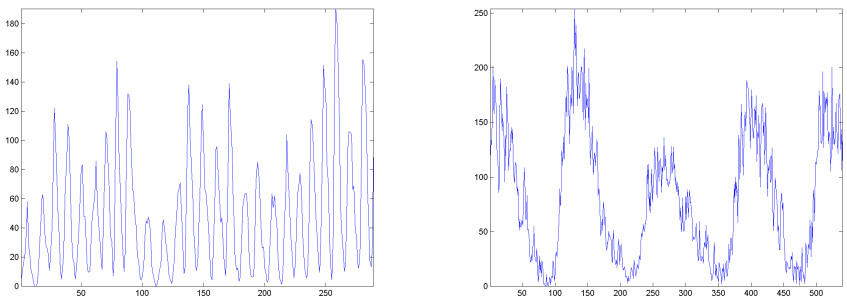
Experiments both with one-dimensional stochastic and chaotic time-series were performed. The task is to predict the scalar value $\mathbf{u}_a(t+1)$ given $\mathbf{u}_a(t)$ (the target value $\mathbf{u}_b(t)$ is $\mathbf{u}_a(t+1)$). Both the naïve and full ESIGMN were compared to the ESN (Echo State Network), Elman Network (also known as Simple Recurrent Network or SRN [Elman 1990]) and static IGMNn and IGMN (using only current input to predict the next one). The error measure used was the normalized MSE with respect to the trivial solution (always predicting the latest observation $\mathbf{u}_a(t)$ for the expected value $\mathbf{u}_a(t+1)$) and is defined as

$$NMSE_t = \frac{\frac{1}{N}\sum_{t=1}^{N} \|\mathbf{y}(t) - \mathbf{u}_a(t+1)\|^2}{\frac{1}{N}\sum_{t=1}^{N} \|\mathbf{u}_a(t) - \mathbf{u}_a(t+1)\|^2} \tag{22}$$

where N is the number of observations, $\mathbf{u}_a(t)$ is the observation at time t, $\mathbf{y}(t)$ is the predicted value at time t and $\mathbf{u}_a(t+1)$ is the desired/target value at time t. It means that solutions worse than the trivial one will have $NMSE_t$ greater than 1, while better solutions will have $NMSE_t$ smaller than 1. The runtime (in seconds) and number of epochs are also informed. The gaussian components information refers to the configuration at the end of training. The parameters of the IGMN based algorithms were the default ones suggested in section 3.1, with ϵ_{max} and δ set to 0.1. v_{min} and sp_{min} were set to 5 and 3, respectively. The ESN output layer was trained with the Conjugate gradient backpropagation with Fletcher-Reeves updates algorithm ('traincgf' in Matlab, which doesn't allow using the Levenberg-Marquardt training algorithm for recurrent networks), with early stopping and default parameters, and this same configuration was used by both layers of the Elman Network. All networks used a hidden layer with 10 neurons, and all input and output layers had size 1, since one-dimensional data was used. All reservoirs, both for ESN and ESIGMNn/ESIGMN, were scaled to a spectral radius of 0.9. All experiments were averaged over 100 runs and executed on a Intel Core 2 Quad Q8400 with 4GB RAM on Matlab 2009b without parallelization.

5.1. Yearly Mean Sunspot Numbers

This dataset consists of 289 yearly (mean) observations of a stochastic time-series, which can be seen in figure 4(a). The first 200 observations were used for training, while the remaining 89 were used for testing. For this experiment, $NMSE_t$ = 1 corresponds to $NMSE$ = 0.35 (normalized MSE w.r.t. the observations mean). Results are summarized in table 1 with mean values and standard deviations between parenthesis.



(a) The yearly mean sunspot numbers time-series.

(b) The first 550 data points of the monthly sunspot numbers time-series.

Figure 4. The two flavors of sunspot time-series.

	Elman	ESN	IGMNn	IGMN	ESIGMNn	ESIGMN
$NMSE_t$	0.97 (0.04)	0.92 (0.02)	1.38 (0.00)	0.93 (0.00)	2.23 (0.63)	0.50 (0.06)
Epochs	10.97 (5.32)	5.33 (2.26)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Runtime	0.98 (0.77)	0.55 (0.45)	0.56 (0.34)	0.89 (0.40)	0.58 (0.18)	0.66 (0.28)
Gaussian Components	-	-	5.00 (0.00)	3.00 (0.00)	4.58 (1.23)	1.58 (0.57)

Table 1. Results of the mean yearly sunspot numbers experiment. Mean values outside parenthesis, standard deviations inside.

The ESIGMN algorithm achieved very good results in this experiment, well ahead of its competitors. The ESIGMNn, on the other hand, had the worst result, and also had high variability in the error measure, meaning it's highly sensitive to the reservoir random initialization (It's worth mentioning that a typical reservoir has hundreds of neurons, while only 10 were used in this work; larger reservoirs would potentially increase the chances of giving useful information to the IGMNn, thus reducing error mean and variance). The static IGMN could solve the problem by exploiting the fact that when the time-series is low, it tends to increase, and vice-versa. Its three gaussian components encoded low, medium and high values. A similar phenomenon happened in all experiments.

5.2. Monthly Sunspot Numbers

This dataset consists of 2987 monthly observations of a stochastic time-series, which can be seen in figure 4(b). The first 2000 observations were used for training, while the remaining 987 were used for testing. For this experiment, $NMSE_t$ = 1 corresponds to $NMSE$ = 0.1. Results are summarized in table 2.

	Elman	ESN	IGMNn	IGMN	ESIGMNn	ESIGMN
$NMSE_t$	1.02 (0.08)	0.99 (0.01)	3.67 (0.00)	0.99 (0.00)	2.59 (0.46)	0.88 (0.02)
Epochs	17.49 (16.79)	6.29 (3.83)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Runtime	4.74 (4.27)	1.65 (1.05)	5.19 (0.94)	7.37 (1.57)	6.38 (1.42)	6.19 (2.26)
Gaussian Components	-	-	4.00 (0.00)	4.00 (0.00)	5.41 (0.82)	2.10 (0.36)

Table 2. Results of the monthly sunspot numbers experiment.

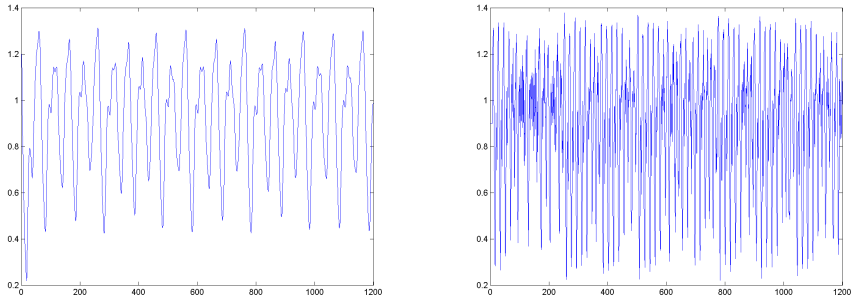
The results were very similar to the previous ones. ESIGMNn didn't do better than the trivial solution ($NMSE_t$ = 1), but the ESIGMN got the best result again.

5.3. Mackey-Glass (τ=17)

This dataset consists of 1201 observations of a chaotic time-series defined by the equation

$$\frac{dx}{dt} = 0.2\frac{x_{t-\tau}}{1 + x_{t-\tau}^n} - 0.1x_t \tag{23}$$

with $\tau = 17$, which can be seen in figure 5(a). The first 1000 observations were used for training, while the remaining 201 were used for testing. For this experiment, $NMSE_t$ = 1 corresponds to $NMSE$ = 0.0196. Results are summarized in table 3.



(a) The Mackey-Glass (τ=17) time-series. (b) The Mackey-Glass (τ=30) time-series.

Figure 5. The two flavors of Mackey-Glass time-series.

	Elman	ESN	IGMNn	IGMN	ESIGMNn	ESIGMN
$NMSE_t$	1.06 (0.19)	1.93 (0.18)	8.66 (0.00)	0.99 (0.00)	27.68 (15.63)	0.13 (0.06)
Epochs	49.81 (31.30)	7.42 (4.60)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Runtime	7.79 (5.37)	1.01 (0.84)	2.76 (0.62)	2.45 (0.73)	3.00 (1.07)	1.78 (0.81)
Gaussian Components	-	-	4.00 (0.00)	1.00 (0.00)	4.01 (2.63)	1.04 (0.20)

Table 3. Results of the Mackey-Glass (τ = 17) experiment.

It's interesting to note that the ESIGMN could achieve the best result (by a large margin) in this experiment with only a single gaussian component, meaning that it found a linear solution over input and reservoir state space. An ESIGMN with just one component is almost equivalent to an ESN trained in batch mode, but the online ESN is inefficient with stochastic gradient descent because of the large eigenvalue spread of the reservoir states [Lukoševičius and Jaeger 2009]. The ESIGMN seems to avoid such problem.

5.4. Mackey-Glass (τ=30)

This dataset with 1500 observations is generated by the same equation 23 but with $\tau = 30$, and can be seen in figure 5(b). The first 1000 observations were used for training, while the remaining 500 were used for testing. For this experiment, $NMSE_t$ = 1 corresponds to $NMSE$ = 0.359. Results are summarized in table 4.

	Elman	ESN	IGMNn	IGMN	ESIGMNn	ESIGMN
$NMSE_t$	0.92 (0.02)	0.94 (0.01)	1.10 (0.00)	0.91 (0.00)	1.67 (0.53)	0.39 (0.08)
Epochs	29.10 (15.25)	6.39 (4.21)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Runtime	4.61 (2.69)	0.87 (0.82)	2.83 (0.79)	4.72 (1.37)	3.38 (0.62)	3.56 (1.80)
Gaussian Components	-	-	5.00 (0.00)	6.00 (0.00)	5.41 (1.10)	4.06 (1.20)

Table 4. Results of the Mackey-Glass (τ = 30) experiment.

Again, the ESIGMN got the best results for this experiment, well ahead of its competitors. More components were needed to solve this time-series in relation to the previous one. This can be seen as a mixture of locally weighted linear experts, which enables the ESIGMN to achieve better performance than the ESN.

6. Conclusions

This work presented the ESIGMN, an one-shot, incremental and spatio-temporal learning algorithm, which has both ESN and IGMN advantages, like:

- Spatio-temporal pattern processing without the need to configure delay lines (all experiments used a reservoir with the same size)
- Online, incremental, one-shot learning
- Few non-critical parameters to be tuned manually (the same parameters were kept for all experiments)

Four experiments were executed, with 2 stochastic and 2 chaotic time-series. ESIGMN got the best results for all of them (excelling on the chaotic ones), showing that the IGMN is indeed a good replacement for the linear output layer of the ESN and also that the introduction of a reservoir increases the IGMN temporal capacity drastically (while being largely insensitive to the random nature of the reservoir). The ESN and IGMN algorithms alone yielded similar reasonable results, but together in the ESIGMN they complemented each other producing large performance gains. On the other hand, ESIGMNn got the worst results for all of them, indicating that the naïve version is not appropriate for time series prediction, at least with default parameters (experiments not shown in this work due to space restrictions show that lower ϵ_{max}, δ and sp_{min} values can give better results for the naïve version, better than the trivial solution). No parameter tuning or reservoir adaptation were done, meaning that there is room for a lot of improvement. Any improvements to reservoirs can be directly applied to ESIGMN, like Intrinsic Plasticity [Schrauwen et al. 2008], Orthogonal Reservoirs [White et al. 2004] or Critical ESN [Hajnal and Lőrincz 2006]. Also, any future improvements to the IGMM or IGMN algorithms will apply directly to ESIGMN.

The worst execution time for ESIGMN was of $\sim$5ms for each 12-dimensional data point (1 input + 10 reservoir state values + 1 target value), running on Matlab without any parallelization. This paves the way to real-time learning and prediction of sequences, which can be very useful in robotics, games, reinforcement learning, embedded systems and monitoring tools.

In future works, these other IGMN temporal extensions will be explored: Adding tapped delay lines [Kangas 1991]; Using differentiator-integrator neurons [Moser 2004]; Using recurrent connections in various ways [Pinto 2010].

References

Buehner, M. and Young, P. (2006). A tighter bound for the echo state property. *Neural Networks, IEEE Transactions on*, 17(3):820–824.

Dempster, A., Laird, N., Rubin, D., et al. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Dominey, P. F. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological Cybernetics*, 73(3):265–74.

Elman, J. (1990). Finding structure in time* 1. *Cognitive science*, 14(2):179–211.

Engel, P. and Heinen, M. (2011). Incremental learning of multivariate gaussian mixture models. *Advances in Artificial Intelligence–SBIA 2010*, pages 82–91.

Hajnal, M. and Lőrincz, A. (2006). Critical echo state networks. *Artificial Neural Networks–ICANN 2006*, pages 658–667.

Hayes, M. (1996). 9.4: Recursive Least Squares",". *Statistical Digital Signal Processing and Modeling*.

Heinen, M. (2011). *A Connectionist Approach for Incremental Function Approximation and On-line Tasks*. PhD thesis, Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação.

Heinen, M. and Engel, P. (2010). An Incremental Probabilistic Neural Network for Regression and Reinforcement Learning Tasks. *Artificial Neural Networks–ICANN 2010*, pages 170–179.

Jaeger, H. (2001). The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. Technical report, Technical Report GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. (2003). Adaptive nonlinear system identification with echo state networks. *Advances in Neural Information Processing Systems*, 15:593–600.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78.

Jaeger, H., Lukosevicius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352.

Kangas, J. (1991). Phoneme recognition using time-dependent versions of self-organizing maps. In *icassp*, pages 101–104. IEEE.

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

Moser, L. (2004). Modelo de um neurônio diferenciador-integrador para representação temporal em arquiteturas conexionistas. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação.

Natschläger, T., Maass, W., and Markram, H. (2002). The "liquid computer": A novel strategy for real-time computing on time series. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1):39–43.

Pinto, R. (2010). Um Estudo de Redes Neurais Não-Supervisionadas Temporais. Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação.

Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J., and Stroobandt, D. (2008). Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7-9):1159–1171.

Steil, J. J. (2004). Backpropagation-decorrelation: online recurrent learning with o(n) complexity,. In *IJCNN*.

Verstraeten, D., Schrauwen, B., and Stroobandt, D. (2006). Reservoir-based techniques for speech recognition. In *Proceedings of the world conference on computational intelligence*, pages 1050–1053.

White, O., Lee, D., and Sompolinsky, H. (2004). Short-term memory in orthogonal neural networks. *Physical review letters*, 92(14):148102.

Widrow, B. (1966). Adaptive filters I: fundamentals (TR 6764-6).