IGMN: An Incremental Gaussian Mixture Network that Learns Instantaneously from Data Flows

Milton Roberto Heinen[1], **Paulo Martins Engel**[1] **and Rafael C. Pinto**[1]

[1]Informatics Institute – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15064, CEP 91501-970, Porto Alegre-RS, Brazil

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

Abstract. *This works proposes IGMN (standing for Incremental Gaussian Mixture Network), a new connectionist approach for incremental concept formation and robotic tasks. It is inspired on recent theories about the brain, specially the Memory-Prediction Framework and the Constructivist Artificial Intelligence, which endows it with some unique features that are not present in most ANN models such as MLP and GRNN. Moreover, IGMN is based on strong statistical principles (Gaussian mixture models) and asymptotically converges to the optimal regression surface as more training data arrive. Through several experiments using the proposed model it is demonstrated that IGMN is also robust to overfitting, does not require fine-tunning its configuration parameters and has a very good computational performance, thus allowing its use in real time control applications. Therefore, IGMN is a very useful machine learning tool for incremental function approximation.*

1. Introduction

Artificial neural networks (ANNs) [Haykin 2008] are mathematical or computational models inspired by the structure and functional aspects of biological neural networks. They are composed by several layers of massively interconnected processing units, called artificial neurons, which can change their connection strength (i.e., the synaptic weights values) based on external or internal information that flows through the network during learning. Modern ANNs are non-linear machine learning [Mitchell 1997] tools frequently used to model complex relationships between inputs and outputs and/or to find patterns in data. In the past several neural network models have been proposed. The most well-known model is the Multi-Layer Perceptron (MLP) [Rumelhart et al. 1986], which can be used in function approximation and classification tasks. The MLP supervised learning algorithm, called Backpropagation, uses gradient descent to minimize the mean square error between the desired and actual ANN outputs.

Although in the last decades neural networks have been successfully used in several tasks, including signal processing, pattern recognition and robotics, most ANN models have some disadvantages that difficult their use in incremental function approximation and prediction tasks. The Backpropagation learning algorithm, for instance, requires several scans over all training data, which must be complete and available at the beginning of the learning process, to converge for a good solution. Moreover, after the end of the training process the synaptic weights are "frozen", i.e., the network loses its learning capabilities. These drawbacks highly contrast with the human brain learning capabilities because: (i) we don't need to perform thousands of scans over the training data to learn

something (in general we are able to learn using few examples and/or repetitions, the so called *aggressive learning*); (ii) we are always learning new concepts as new "training data" arrive, i.e., we are always improving our performance through experience; and (iii) we don't have to wait until sufficient information arrives to make a decision, i.e., we can use partial information as it becomes available. Besides being not biologically plausible, these drawbacks difficult the use of ANNs in tasks like incremental concept formation, reinforcement learning and robotics, because in this kind of application the training data are available just instantaneously to the learning system, and in general a decision must be made using the information available at the moment.

This paper proposes a new artificial neural network model, called IGMN (standing for Incremental Gaussian Mixture Network) [Heinen 2011], which is able to tackle great part of these problems described above. IGMN is based on parametric probabilistic models (Gaussian mixture models), that have nice features from the representational point of view, describing noisy environments in a very parsimonious way, with parameters that are readily understandable. In [Heinen and Engel 2010a, Heinen and Engel 2010b] a previous version of IGMN, called IPNN, was proposed. That previous version is based on the supposition of conditional independence among distinct domains, called "naive Bayes" hypothesis. The neural network model proposed in this paper, on the other hand: (i) takes into account the correlation among distinct domains, thus fully implementing a multivariate hypothesis; (ii) uses an error driven mechanism to decide if it is necessary to add a neuron to the neural network; and IGMN (iii) has a unique neural network topology in which there is no distinction between sensory and motor stimuli.

Moreover, IGMN is inspired on recent theories about the brain, specially the Memory-Prediction Framework (MPF) [Hawkins 2005] and the constructivist artificial intelligence [Drescher 1991], which endows it with some unique features that are not present in other neural network models such as: (i) it learns incrementally using a single scan over the training data; (ii) the learning process can proceed perpetually as new training data arrive; (iii) it can handle the stability-plasticity dilemma and does not suffer from catastrophic interference; (iv) the neural network topology is defined automatically and incrementally; and (v) IGMN is not sensitive to initialization conditions. The remaining of this paper is organized as follows. Section 2 presents the ANN model proposed in this paper; Section 3 describes some experiments performed to evaluate the proposed model; and Section 4 provides some final remarks about this work.

2. Incremental Gaussian Mixture Network

Figure 1 shows the general architecture of IGMN. It is composed by an association region $\mathcal{P}$ (in the top of this figure) and many cortical regions, $\mathcal{N}^{\mathcal{A}}, \mathcal{N}^{\mathcal{B}}, \ldots, \mathcal{N}^{\mathcal{S}}$. All regions have the same number of neurons, M. Initially there is a single neuron in each region (i.e., $M = 1$), but more neurons are incrementally added when necessary using an error driven mechanism. Each cortical region $\mathcal{N}^{\mathcal{K}}$ receives signals from the kth sensory/motor modality, $\mathbf{k}$ (in IGMN there is no difference between sensory and motor modalities), and hence there is a cortical region for each sensory/motor modality.

Another important feature of IGMN is that all cortical regions $\mathcal{N}$ execute a common function, i.e., they have the same kind of neurons and use the same learning algorithm. Moreover, all cortical regions can run in parallel, which improves the performance

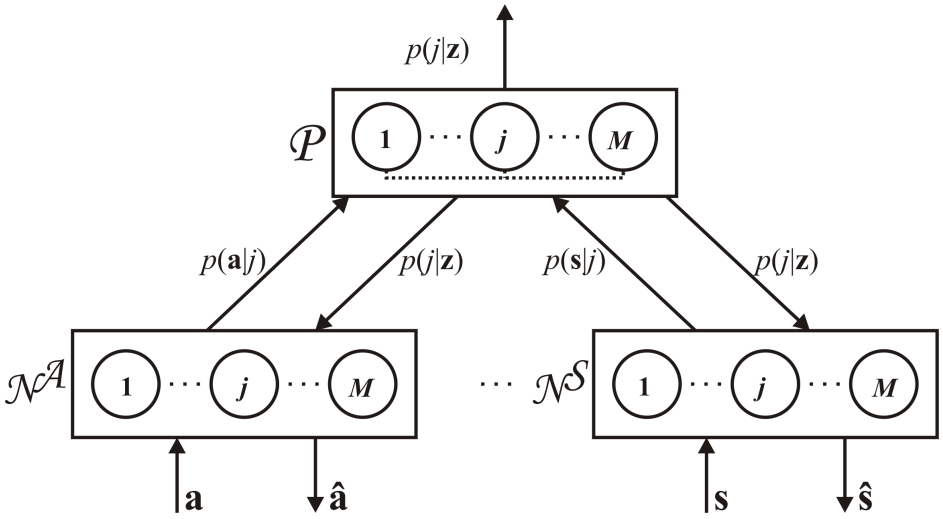


Figure 1. General architecture of the proposed neural network model

specially in parallel architectures. Each neuron j of region $\mathcal{N}^{\mathcal{K}}$ performs the following operation:

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

i.e., a multivariate Gaussian distribution, where $D^{\mathcal{K}}$ is the dimensionality of $\mathbf{k}$ (different sensory/motor modalities $\mathbf{k}$ can have different dimensions $D^{\mathcal{K}}$). Each neuron j maintains a mean vector $\boldsymbol{\mu}_j^{\mathcal{K}}$ and a covariance matrix $\mathbf{C}_j^{\mathcal{K}}$.

In IGMN the regions are not fully connected, i.e., the neuron j of $\mathcal{N}^{\mathcal{K}}$ is connected just to the jth neuron of $\mathcal{P}$, but this connection is bidirectional. It is important to notice that there are no *synaptic weights* in these connections, i.e., all IGMN parameters are stored in the neurons themselves. A bottom-up connection between $\mathcal{N}^{\mathcal{K}}$ and $\mathcal{P}$ provides the component density function $p(\mathbf{k}|j)$ to the jth neuron in $\mathcal{P}$. Therefore, a neuron j in the association region $\mathcal{P}$ is connected with the jth neuron of all cortical regions $\mathcal{N}$ via bottom-up connections and computes the a posteriori probability using the Bayes' rule:

$$p(j|\mathbf{z}) = \frac{p(\mathbf{a}|j)\, p(\mathbf{b}|j) \ldots p(\mathbf{s}|j)\; p(j)}{\sum_{q=1}^{M} p(\mathbf{a}|q)\, p(\mathbf{b}|q) \ldots p(\mathbf{s}|q)\; p(q)}, \tag{2}$$

where it is considered that the neural network has an arbitrary number, s, of cortical regions and $\mathbf{z} = \{\mathbf{a}, \mathbf{b}, \ldots, \mathbf{s}\}$. The dotted lines in Figure 1 above indicate the lateral interaction among the association units for computing the denominator in (2).

Each neuron j of the association region $\mathcal{P}$ maintains its a priori probability, $p(j)$, an accumulator of the a posteriori probabilities, sp_j, and an association matrix to store the correlations among each sensory/motor modality. If a neural network has two cortical regions, $\mathcal{N}^{\mathcal{A}}$ and $\mathcal{N}^{\mathcal{B}}$, for instance, then the association matrix $\mathbf{C}_j^{\mathcal{AB}}$ will have two dimensions and size $D^{\mathcal{A}} \times D^{\mathcal{B}}$. Note that it is not necessary to maintain $\mathbf{C}_j^{\mathcal{BA}}$ because $\mathbf{C}_j^{\mathcal{BA}} = \mathbf{C}_j^{\mathcal{AB}^T}$. The top-down connections between $\mathcal{P}$ and $\mathcal{N}^{\mathcal{K}}$, on the other hand, returns *expectations* to $\mathcal{N}^{\mathcal{K}}$ which are used to estimate $\hat{\mathbf{k}}$ when $\mathbf{k}$ is missing. This architecture

is inspired on the memory-prediction framework (MPF) [Hawkins 2005], which states that different cortical regions are not fully connected in the neocortex. Instead, they are linked to the association areas $\mathcal{P}$ through bottom-up and top-down connections, thus providing predictions and expectations, respectively, to all cortical regions $\mathcal{N}^K$. The main advantage of this strategy is to speed up IGMN and make it more suitable to real-time and critical applications, because it is much faster to invert two covariance matrices of size M than a single covariance matrix of size $2M$. Moreover, a large number of samples is required to obtain good estimates from a large covariance matrix, and therefore using this strategy IGMN becomes more aggressive, i.e., it is able to provide good estimates using few training samples. Next subsections describe the IGMN operation in details.

2.1. IGMN operation

IGMN has two operation modes, called *learning* and *recalling*. But unlike most ANN models, in IGMN these operations don't need to occur separately, i.e., the learning and recalling modes can be intercalated. In fact, even after the presentation of a single training pattern the neural network can already be used in the recalling mode (the acquired knowledge can be immediately used), and the estimates become more precise as more training data are presented. Moreover, the learning process can proceed perpetually, i.e., the neural network parameters can always be updated as new training data arrive. As described in [Heinen 2011], BP-trained MLP neural networks cannot learn perpetually because: (i) they suffer from catastrophic interference; (ii) do not have means to handle the stability plasticity dilemma; and (iii) require that the entire database be available at the beginning of the learning process [Haykin 2008].

IGMN adopts an error-driven mechanism to decide if it is necessary to add a neuron in each region for *explaining* a new data vector $\mathbf{z}^t$. This error-driven mechanism is inspired on the Constructivist IA [Drescher 1991, Chaput 2004], where the accommodation process occurs when it is necessary to change the neural network structure (i.e. to add a neuron in each region) to account for a new *experience* which is not explained for the current schemata (i.e., the current ANN structure), and the assimilation process occurs when the new experience is well explained in terms of the existing schemata [Piaget 1954]. In mathematical terms, the ANN structure is changed if the instantaneous approximation error ε is larger that a user specified threshold ε_{max}.

The following subsections describe the IGMN operation during learning and recalling. To simplify our explanation, we will consider that the neural network has just two cortical regions, $\mathcal{N}^{\mathcal{A}}$ and $\mathcal{N}^{\mathcal{B}}$, that receive the stimuli $\mathbf{a}$ and $\mathbf{b}$, respectively. It will be also considered that we are estimating $\hat{\mathbf{b}}$ from $\mathbf{a}$ in the recalling mode. But it is important to remember that: (i) IGMN can have more than two cortical regions (one for each sensory/motor stimulus $\mathbf{k}$); and (ii) after training it can be used to estimate either $\hat{\mathbf{a}}$ or $\hat{\mathbf{b}}$ (i.e., there is no difference between *inputs* and *outputs* in IGMN).

2.2. Learning mode

The learning algorithm used by IGMN is based on the incremental Gaussian mixture model learning algorithm presented in [Engel and Heinen 2010a,Engel and Heinen 2010b], but it has many modifications which adapt it to supervised tasks such as incremental function approximation and prediction. Before learning starts the neural network is empty, i.e., all regions have $M = 0$ neurons. When the first training pattern $\mathbf{z}^1 = \{\mathbf{a}^1, \mathbf{b}^1\}$ arrives,

a neuron in each region is created centered on $\mathbf{z}^1$ and the neural network parameters are initialized as follows:

$$M = 1; \quad sp_1 = 1.0; \quad p(1) = 1.0; \quad \mathbf{C}_1^{\mathcal{AB}} = \mathbf{0};$$
$$\boldsymbol{\mu}_1^{\mathcal{A}} = \mathbf{a}^1; \quad \boldsymbol{\mu}_1^{\mathcal{B}} = \mathbf{b}^1; \quad \mathbf{C}_1^{\mathcal{A}} = {\boldsymbol{\sigma}_{ini}^{\mathcal{A}}}^2\mathbf{I}; \quad \mathbf{C}_1^{\mathcal{B}} = {\boldsymbol{\sigma}_{ini}^{\mathcal{B}}}^2\mathbf{I},$$

where the subscript '1' indicates the neuron $j = 1$ in each region, M is the number of neurons in each region (all regions have the same size M), sp is an accumulator of posterior probabilities maintained in the association region $\mathcal{P}$, $\mathbf{0}$ is a zero matrix of size $D^{\mathcal{A}} \times D^{\mathcal{B}}$, $\boldsymbol{\sigma}_{ini}^{\mathcal{A}}$ and $\boldsymbol{\sigma}_{ini}^{\mathcal{B}}$ are diagonal matrices that define the initial radius of the covariance matrices (the pdf is initially circular but it changes to reflect the actual data distribution as new training data arrive) and $\mathbf{I}$ denotes the identity matrix. $\boldsymbol{\sigma}_{ini}^{\mathcal{A}}$ and $\boldsymbol{\sigma}_{ini}^{\mathcal{B}}$ are initialized using a user defined fraction δ of the overall variance (e.g., $\delta = 1/100$) of the corresponding attributes, estimated from the range of these values according to:

$$\boldsymbol{\sigma}_{ini}^{\mathcal{K}} = \delta \left[\max(\mathbf{k}) - \min(\mathbf{k})\right] , \tag{3}$$

where $[\min(\mathbf{k}), \max(\mathbf{k})]$ defines the domain of a sensory/motor modality $\mathbf{k}$ (throughout this paper the symbol k will be used to indicate any sensory/motor modality, i.e., either a or b in this case). It is important to say that it is not necessary to know the *exact* minimum and maximum values along each dimension to compute $\boldsymbol{\sigma}_{ini}^{\mathcal{K}}$, but just the approximate domain of each feature instead.

When a new training pattern $\mathbf{z}^t$ arrives, all cortical regions are activated, i.e., $p(\mathbf{k}|j)$ is computed using Equation 1 above, and the probabilities $p(\mathbf{z}^t|j)$ are sent to the association region $\mathcal{P}$, which computes the *joint* posterior probabilities $p(j|\mathbf{z}^t)$ using the Bayes' rule in (2). After this, the posterior probabilities $p(j|\mathbf{z}^t)$ are sent back to the cortical region $\mathcal{N}^{\mathcal{B}}$, which compute its estimate as follows:

$$\hat{\mathbf{b}} = \sum_{j=1}^{M} p(j|\mathbf{z}^t)[\boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{BA}}{\mathbf{C}_j^{\mathcal{A}}}^{-1}(\mathbf{a}^t - \boldsymbol{\mu}_j^{\mathcal{A}})], \tag{4}$$

where $\mathbf{C}_j^{\mathcal{AB}}$ is the jth association matrix maintained in association region $\mathcal{P}$ and $\mathbf{C}_j^{\mathcal{BA}}$ is its transpose. Note that as the covariance matrix $\mathbf{C}_j^{\mathcal{B}}$ was already inverted when $\mathcal{N}^{\mathcal{B}}$ was activated in the bottom-up direction, we don't need to invert it again, i.e., we can temporarily store the corresponding inverse matrix to speed up the IGMN learning algorithm. Using the estimate $\hat{\mathbf{b}}$ the normalized approximation error ε is given by:

$$\varepsilon = \max_{i \in D^{\mathcal{B}}} \left[\frac{\|b_i^t - \hat{b}_i\|}{\max(b_i) - \min(b_i)} \right] \tag{5}$$

where $[\min(b_i), \max(b_i)]$ defines the domain of the sensory/motor feature b_i. Again $\min(b_i)$ and $\max(b_i)$ do not need to be the exact minimum and maximum values of $\mathbf{b}$ – they may be just approximations of the domain of each b_i feature (in fact $\min(b_i)$ and $\max(b_i)$ are just used to make IGMN independent from the range of the data features). If ε is larger than a user specified threshold, ε_{max}, than $\mathbf{z}^t$ is not considered as *represented* by any existing neuron in the cortical regions. In this case, a new unit j is created in each region and centered on $\mathbf{z}^t$ and all priors of the association region $\mathcal{P}$ are recomputed by:

$$p(j)^* = \frac{sp_j}{\sum_{q=1}^{M^*} sp_q}. \tag{6}$$

Otherwise (if $\mathbf{z}$ is well explained by any of the existing Gaussian units), the a posteriori probabilities $p(j|\mathbf{z}^t)$ are added to the current value of the $sp(j)$ on the association region:

$$sp_j^* = sp_j + p(j|\mathbf{z}^t), \quad \forall j, \tag{7}$$

and the priors $p(j)$ are recomputed using (6). Then $\omega_j = p(j|\mathbf{z}^t)/sp_j^*$ is sent back to all cortical regions, and the parameters of all neurons in $\mathcal{N}^{\mathcal{K}}$ are updated using the following recursive equations derived in [Engel and Heinen 2010a, Engel and Heinen 2010b]:

$$\boldsymbol{\mu}_j^{\mathcal{K}*} = \boldsymbol{\mu}_j^{\mathcal{K}} + \omega_j \left(\mathbf{z}^t - \boldsymbol{\mu}_j^{\mathcal{K}}\right) \tag{8}$$

$$\mathbf{C}_j^{\mathcal{K}*} = \mathbf{C}_j^{\mathcal{K}} - (\boldsymbol{\mu}_j^{\mathcal{K}*} - \boldsymbol{\mu}_j^{\mathcal{K}})(\boldsymbol{\mu}_j^{\mathcal{K}*} - \boldsymbol{\mu}_j^{\mathcal{K}})^T + \omega_j \left[(\mathbf{z} - \boldsymbol{\mu}_j^{\mathcal{K}*})(\mathbf{z} - \boldsymbol{\mu}_j^{\mathcal{K}*})^T - \mathbf{C}_j^{\mathcal{K}}\right], \tag{9}$$

where the superscript '$*$' refers to the new (updated) values. Finally the association matrix $\mathbf{C}_j^{\mathcal{AB}}$ is updated using the following recursive equation:

$$\mathbf{C}_j^{\mathcal{AB}*} = \mathbf{C}_j^{\mathcal{AB}} - (\boldsymbol{\mu}_j^{\mathcal{A}*} - \boldsymbol{\mu}_j^{\mathcal{A}})(\boldsymbol{\mu}_j^{\mathcal{B}*} - \boldsymbol{\mu}_j^{\mathcal{B}})^T + \omega_j \left[(\mathbf{a}^t - \boldsymbol{\mu}_j^{\mathcal{A}*})(\mathbf{b}^t - \boldsymbol{\mu}_j^{\mathcal{B}*})^T - \mathbf{C}_j^{\mathcal{AB}}\right], \tag{10}$$

which is derived using the same principles described in [Engel and Heinen 2010a, Engel and Heinen 2010b]. This equation is another important contribution of this paper, because it allows computing the covariances among distinct cortical regions incrementally, and thus estimating a missing stimulus $\mathbf{k}$ without having to maintain and invert a complete variance/covariance matrix. In fact, using (10) the complete variance/covariance matrix is broken down in separate submatrices that can be efficiently maintained.

2.3. Recalling mode

In the recalling mode, a stimulus (e.g., $\mathbf{a}$) is presented to a *partially* trained neural network (as the learning process proceeds perpetually, in IGMN we never consider that the training process is over), which computes an estimate for another stimulus (e.g., $\hat{\mathbf{b}}$). As said before, IGMN can be used to estimate either $\hat{\mathbf{a}}$ or $\hat{\mathbf{b}}$, but to simplify our explanation in this and the following sections we will consider that we are estimating $\hat{\mathbf{b}}$ from $\mathbf{a}$.

Initially the stimulus $\mathbf{a}$ is received in the cortical region $\mathcal{N}^{\mathcal{A}}$, where each neuron j computes $p(\mathbf{a}|j)$ using (1). These predictions are sent to the association region $\mathcal{P}$ through the bottom-up connections, which is activated using just $p(\mathbf{a}|j)$:

$$p(j|\mathbf{a}) = \frac{p(\mathbf{a}|j)\, p(j)}{\sum_{q=1}^{M} p(\mathbf{a}|q)\, p(q)}. \tag{11}$$

After this, $p(j|\mathbf{a})$ is sent to the cortical region $\mathcal{N}^{\mathcal{B}}$ via the top-down connections, and $\mathcal{N}^{\mathcal{B}}$ computes the estimated stimulus $\hat{\mathbf{b}}$ using Equation 4. More details about the IGMN learning and recalling algorithms are found in [Heinen 2011].

The main drawback of IGMN is that it is necessary to invert the covariance matrices $\mathbf{C}_j^{\mathcal{K}}$ for each training sample $(\mathbf{a}, \mathbf{b}, \ldots, \mathbf{k})$, and this requires $D^{\log_2 7}$ operations using the Strassen algorithm [Strassen 1969]. A way to reduce the computational complexity is to separate the data features in more sensory/motor areas, thus dividing a big covariance matrix in many matrices of smaller size.

3. Experiments

This section describes several experiments to evaluate the performance of IGMN in function approximation and prediction tasks. The first set of experiments, described in Subsection 3.1 presents some experiments in which IGMN is used to identify a nonlinear plant originally proposed in [Narendra and Parthasarathy 1990]. These experiments are also used to assess the sensitivity of the proposed model to its configuration parameters. The second set of experiments, described in Section 3.2 evaluates the performance of IGMN using a more complex, three-dimensional "Mexican hat" function. The computer platform used in all experiments is a Dell Optiplex 755, equipped with an Intel(R) Core(TM)2 Duo CPU 2.33GHz processor, 64 bits architecture, 1.95GB of RAM memory, GPU Intel and operating system Ubuntu Linux 10.04 LTS of 64 bits. Other experiments, some of them related to incremental robotic tasks, can be found in [Heinen 2011].

3.1. Estimating the outputs of a nonlinear plant

The first experiment consists in identifying a nonlinear plant originally proposed by [Narendra and Parthasarathy 1990] for the control of nonlinear dynamical systems using MLP neural networks. This plant is assumed to be of the form:

$$y_p(k+1) = f[y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)], \tag{12}$$

where $y_p(k+1)$ is the next time sample of the plant output, $y_p(k)$ is the current output, $y_p(k-1)$ and $y_p(k-2)$ are delayed time samples of the output, $u(k)$ is the current input, $u(k-1)$ is the previous input, and the unknown function $f(\cdot)$ has the form:

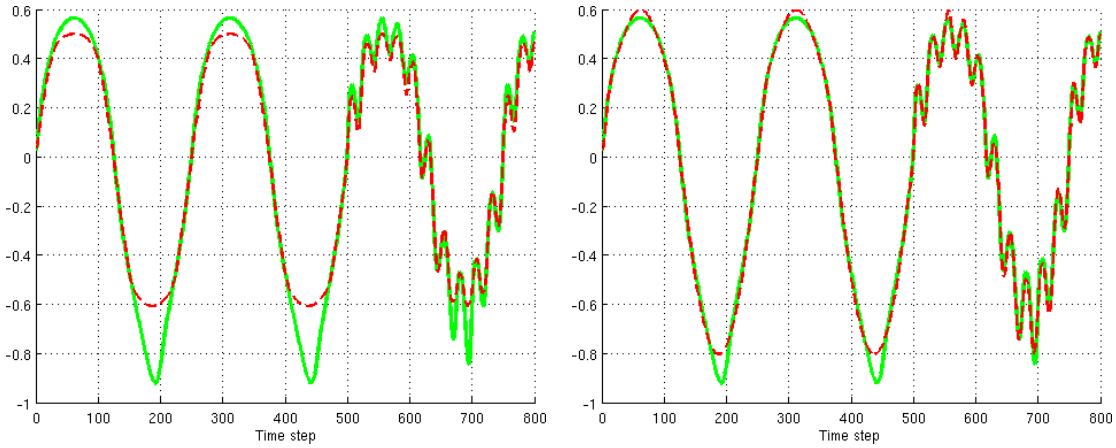
$$f[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5] = \frac{\mathbf{x}_1\mathbf{x}_2\mathbf{x}_3\mathbf{x}_5(\mathbf{x}_3 - 1) + \mathbf{x}_4}{1 + \mathbf{x}_2^2 + \mathbf{x}_3^2}. \tag{13}$$

In [Narendra and Parthasarathy 1990] this plant was identified using a MLP neural network composed by five neurons in the input region, 20 neurons in the first hidden region, 10 neurons on the second hidden region and a single neuron in the output region. This neural network was trained using the standard backpropagation algorithm for 100,000 steps using a random input signal uniformly distributed in the interval $[-1, 1]$ and a step size of $\eta = 0.25$. During the identification procedure (i.e., the learning process) a series-parallel model was used, in which the output of the plant values (i.e., the *desired* answers) were used in the delay units $y_p(k-1)$ and $y_p(k-2)$. After identification the performance of the model was assessed using a parallel model, in which the actual neural network outputs are fed into the delay units. The identified plant (i.e., the trained neural network) was tested using the following function as input:

$$u(k) = \begin{cases} \sin(2\pi k/250) & \text{if } k \leq 500 \\ 0.8\sin(2\pi k/250) + 0.2\sin(2\pi k/25) & \text{if } k > 500 \end{cases} \tag{14}$$

In [Specht 1991] a GRNN network is used to approximate the function $f(\cdot)$ above. In that experiment the identification procedure was carried out for 1000 time steps using a random input signal uniformly distributed in the interval $[-1, 1]$, the spreading parameter was set to $\sigma = 0.315$ and 1000 pattern units were used to represent this plant (one for each random input signal used to identify the plant). Unfortunately in [Specht 1991] the

approximation error computed using the testing function (14) is not informed, and thus we reproduced that experiment 10 times using the original conditions (a standard GRNN network with $\sigma = 0.315$) and different random input signals in each replication. Figure 2(a) shows the best identification performed by GRNN. The averaged NRMS error computed over 10 replications was 0.053267, and 0.213 seconds were necessary to perform each replication.



(a) Identification using GRNN

(b) Identification using IGMN

Figure 2. Identification of a nonlinear plant using 1000 training samples

To compare the performance of IGMN against the ANN models described above, we have repeated this experiment using the same conditions described above, i.e.:

- A series-parallel model for training and a parallel model for testing;
- The identification procedure was carried out for 1000 time steps using a random input signal uniformly distributed in the interval $[-1, 1]$;
- After learning the identified model was tested using inputs given by Equation 14;
- The whole experiment was repeated 10 times using different random input signals in each replication.

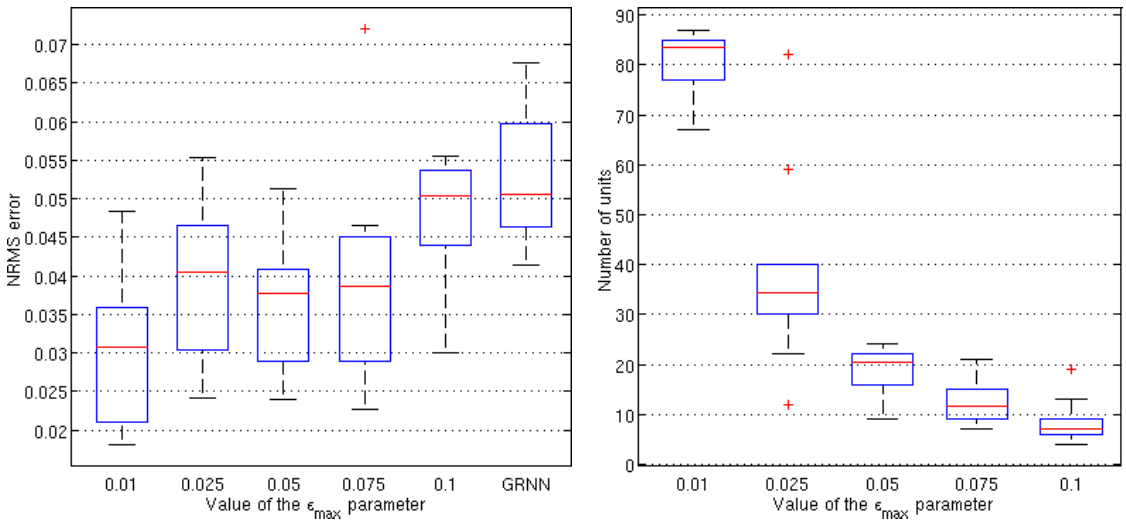
Figure 2(b) shows the identification performed by IGMN using default parameters, i.e.,$\varepsilon_{max} = 0.05$, $\delta = 0.01$ and $\Omega = 6$. Observing Figure 2(b) we can notice that the identification performed by IGMN is superior than that performed by GRNN (Figure 2(a)). The average NRMS error computed over 10 replications was 0.037, 19 neurons were added by IGMN during learning and 0.089 seconds were spent in each replication.

Table 1. Assessing the sensitivity of IGMN to the ε_{max} parameter

ε_{max}	0.1	0.075	0.050	0.025	0.010
Error	0.0474	0.0399	0.0370	0.0394	0.0304
M	8.30	12.60	19.10	38.10	80.40
Time	0.055s	0.089s	0.095s	0.125s	0.285s

To assess the sensitivity of IGMN to the ε_{max} parameter, this experiment was repeated using $\delta = 0.01$ and varying the value of the ε_{max} in the interval $[0.01, 0.1]$. Table 1 shows the results obtained in this experiment (averaged over 10 replications), and

Figures 3(a) and 3(b) show the boxplot graphs of the NRMS error and the number of units added during learning, respectively. It can be noticed in these figures that $\varepsilon_{max} = 0.05$ is a good choice, because it allows a good compromise between the model complexity (number of neurons) and the approximation level (NRMS error).



(a) NRMS error (b) Number of neurons

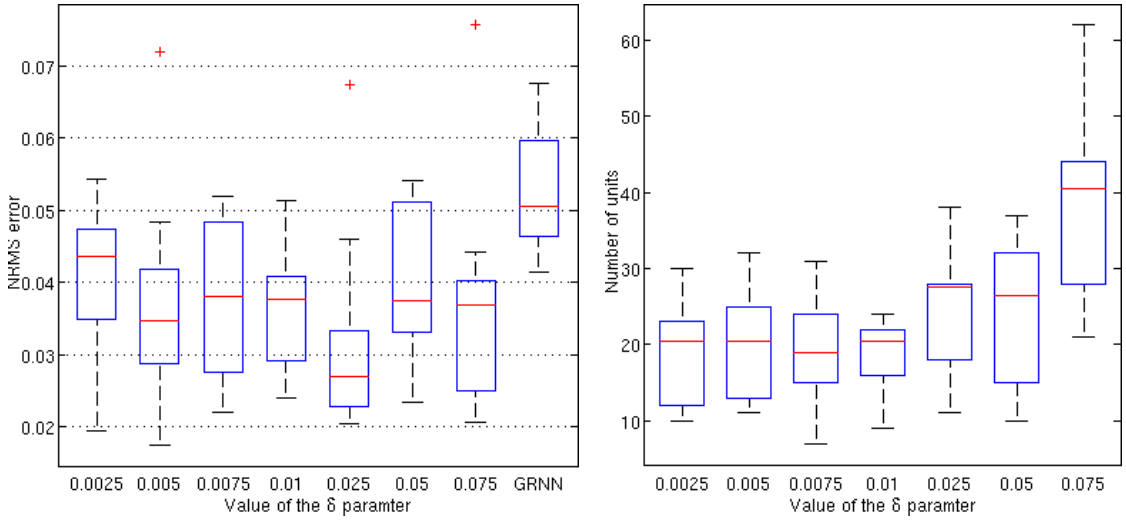
Figure 3. Assessing the sensitivity of IGMN to the ε_{max} parameter

The next experiment aims to assess the sensitivity of IGMN to δ using this plant, where ε_{max} was kept fixed at 0.05 and δ was varied in the interval $[0.0025, 0.075]$. Table 2 shows the average results obtained in this experiment, and Figures 4(a) and 4(b) show the boxplot graphs of the NRMS error and the number of units added during learning, respectively. It can be noticed that the IGMN performance is practically the same using any value of δ in the interval $[0.005, 0.05]$, although using $\delta = 0.025$ the NRMS error is slightly lower. Therefore we can notice that just extreme values of δ degrade the IGMN performance and/or require many Gaussian units.

Table 2. Assessing the sensitivity of IGMN to the δ parameter

δ	0.0025	0.005	0.0075	0.01	0.025	0.05	0.075
Error	0.042	0.037	0.037	0.037	0.032	0.039	0.037
M	18.60	20.20	19.20	19.10	24.70	24.40	38.80
Time	0.09s	0.09s	0.09s	0.09s	0.11s	0.10s	0.12s

In [Specht 1991] it is also pointed out that results similar to those presented in Figure 2(a) can be obtained using just 100 training samples. To validate this affirmation, we repeated this experiment using $N = 100$ training samples randomly chosen in the interval $[-1, 1]$. Using GRNN the NRMS error averaged over 10 replications was 0.090904, and using IGMN it was just 0.067239. Moreover, using IGMN the average number of neurons added during learning was just 3.6, whilst GRNN had used 100 pattern units in each replication.



(a) NRMS error (b) Number of neurons

Figure 4. Assessing the sensitivity of IGMN to the δ parameter

3.2. Approximating the "Mexican hat" function

In the previous experiments we have used a training data set composed by just two data features, a and b. The next experiment uses a three-dimensional data set composed by $N = 5000$ data samples given by:

$$b = \sin(a_1)/a_1 \sin(a_2)/a_2, \tag{15}$$

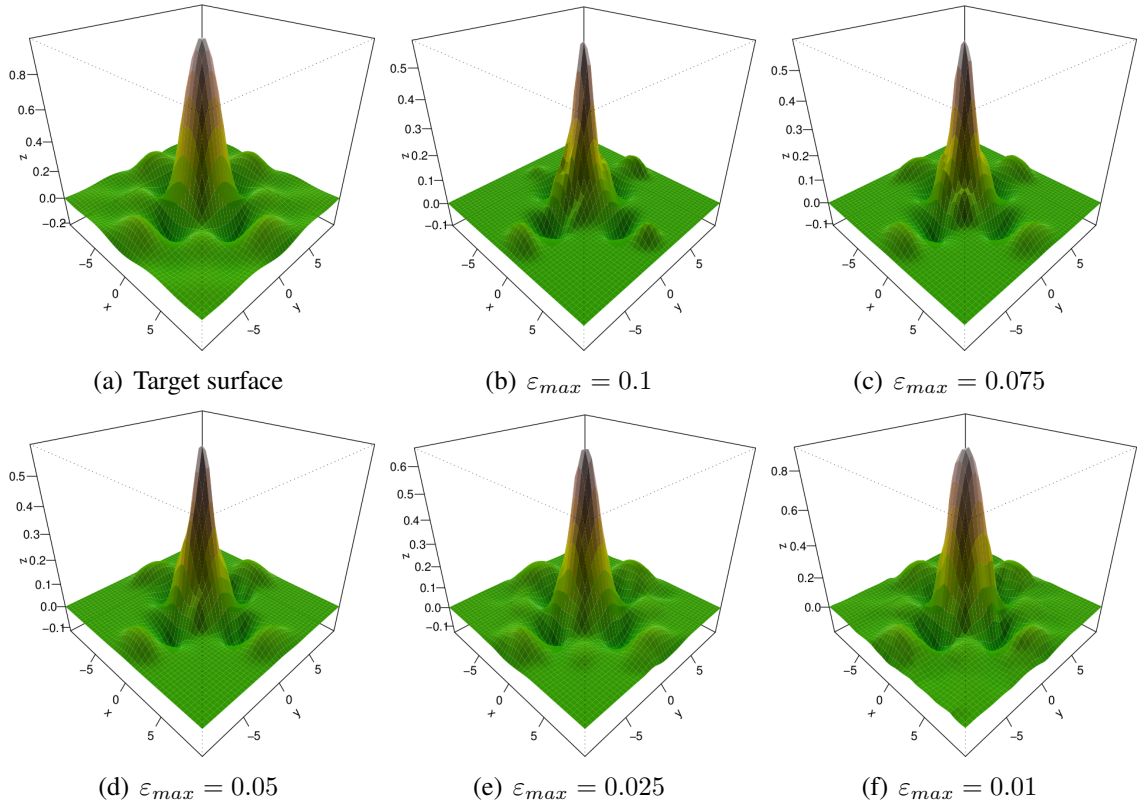
were a_1 and a_2 are randomly chosen in the interval $[-10, 10]$. Figure 5(a) shows the target surface, also known as "Mexican hat". The IGMN net used to learn this data set has two cortical regions: $\mathcal{N}^{\mathcal{A}}$, composed by two data features (a_1 and a_2), and $\mathcal{N}^{\mathcal{B}}$, composed by a single feature (b). This data set was randomly divided into two subsets, called training and testing data sets, each of them composed by 2500 data samples. Moreover, the training data was presented to IGMN in a random order (i.e., the data set was shuffled).

To assess the sensitivity of IGMN to ε_{max} over this data set we have repeated this experiment using different configurations of ε_{max}, and the results obtained in this experiment are shown in Table 3. The first row shows the configuration of the ε_{max} parameter. The following rows show, respectively, the NRMS error, number of neurons added during learning (M) and the learning time. The δ parameter was kept fixed at 0.01, but the results are practically the same using δ in the interval $0.005 \leq \delta \leq 0.05$.

Table 3. Approximating the "Mexican hat" function using IGMN

ε_{max}	0.1	0.075	0.05	0.025	0.01
NRMS	0.071302	0.067699	0.065824	0.051815	0.014288
M	19	25	37	83	184
Time	0.088s	0.104s	0.188s	0.656s	2.292s

Table 3 shows that the NRMS error is reduced as the ε_{max} parameter is decreased, whilst the number of neurons added during learning is increased. Using $\varepsilon_{max} = 0.01$,



(a) Target surface (b) $\varepsilon_{max} = 0.1$ (c) $\varepsilon_{max} = 0.075$

(d) $\varepsilon_{max} = 0.05$ (e) $\varepsilon_{max} = 0.025$ (f) $\varepsilon_{max} = 0.01$

Figure 5. Approximating the "Mexican hat" function using IGMN

for instance, the NRMS error is just 0.014288, but 184 neurons are added to each region during learning. If $\varepsilon_{max} \approx 0$ then the approximation error will be almost zero (remember that this data set is noise-free), but a huge number of neurons will be added during learning. Just for illustration purposes, Figures 5(b) to 5(f) show the surface approximated by IGMN using each ε_{max} setting. We can notice in these figures that using just 21 neurons (Figure 5(b)) the proposed model is able to learn the general structure of the target surface, and using 184 neurons (Figure 5(f)) the approximated surface is quite good.

Using a MLP neural network trained with the Levenberg-Marquardt (LM) algorithm and 20 hidden neurons, the average NRMS error computed over the testing data set was 0.015265 (we needed to repeat this experiment 10 times due to the random initialization of the synaptic weights in a MLP network), and the time required for learning was 20.633 seconds. However, using MLP we had to perform an exhaustive search to find out the best number of hidden neurons. Using GRNN with $\sigma = 1.0$, on the other hand, the NRMS error was 0.017395 and the learning time was 0.189 seconds. As said before, the number of neurons added by GRNN is equal to the number of pattern neurons, i.e., 2500 pattern neurons in this case.

4. Conclusion

This paper has presented IGMN [Heinen 2011], a new connectionist approach for incremental function approximation and prediction. IGMN is inspired on recent theories about the brain, specially the Memory-Prediction Framework (MPF) [Hawkins 2005] and the constructivist artificial intelligence [Drescher 1991], which endows it with some unique

features that are not present in other ANN models such as MLP and GRNN. Moreover, IGMN is based on strong statistical principles and asymptotically converges to the optimal regression surface as more training data arrive. The performed experiments demonstrated that IGMN is a very useful machine learning tool for incremental function approximation.

Acknowledgment

The authors acknowledge the support granted by CNPq.

References

Chaput, H. H. (2004). *The Constructivist Learning Architecture: A Model of Cognitive Development for Robust Autonomous Robots*. PhD thesis, Univ. Texas, Austin, TX.

Drescher, G. L. (1991). *Made-up Minds: A Constructivist Approach to Artificial Intelligence*. The MIT Press, Cambridge, MA.

Engel, P. M. and Heinen, M. R. (2010a). Concept formation using incremental Gaussian mixture models. In *Proc. 15th Iberoamerican Congr. Pattern Recognition (CIARP)*, LNCS, São Paulo, SP, Brazil. Springer-Verlag.

Engel, P. M. and Heinen, M. R. (2010b). Incremental learning of multivariate Gaussian mixture models. In *Proc. 20th Brazilian Symposium on AI (SBIA)*, volume 6404 of *LNCS*, pages 82–91, São Bernardo do Campo, SP, Brazil. Springer-Verlag.

Hawkins, J. (2005). *On Intelligence*. Owl Books, New York, NY.

Haykin, S. (2008). *Neural Networks and Learning Machines*. Prentice-Hall, Upper Saddle River, NJ, 3 edition.

Heinen, M. R. (2011). *A Connectionist Approach for Incremental Function Approximation and On-line Tasks*. Ph.D. thesis, Informatics Institute – Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil.

Heinen, M. R. and Engel, P. M. (2010a). An incremental probabilistic neural network for regression and reinforcement learning tasks. In *Proc. 20th Int. Conf. Artificial Neural Networks (ICANN 2010)*, LNCS, Thessaloniki, Greece. Springer-Verlag.

Heinen, M. R. and Engel, P. M. (2010b). IPNN: An incremental probabilistic neural network for function approximation and regression tasks. In *Proc. 11th Brazilian Neural Networks Symposium (SBRN)*, pages 39–44, São Bernardo do Campo, SP, Brazil.

Mitchell, T. (1997). *Machine Learning*. McGrall-Hill, New York.

Narendra, K. S. and Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks*, 1:4–27.

Piaget, J. (1954). *The construction of Reality in the Child*. Basic Books, New York, NY.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*. The MIT Press, Cambridge, MA.

Specht, D. F. (1991). A general regression neural network. *IEEE Trans. Neural Networks*, 2(6):568–576.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(3):354–356.