Uninorm Based Evolving Neural Networks and its Approximation Capabilities

F. Bordignon, F. Gomide

DCA-FEEC-Unicamp, Av. Albert Einstein, 400
Campinas/SP, Brazil 13083-852

Abstract

Learning from data streams is a contemporary and challenging issue due to the increasing rate of the size and temporal availability of data, turning traditional learning methods impracticable. This work addresses a structure and introduces a learning approach to train uninorm-based hybrid neural networks using extreme learning concepts. Uninorms bring flexibility and generality to fuzzy neuron models as they can behave like triangular norms, triangular conorms, or in between by adjusting identity elements. This feature adds a form of plasticity in neural network modeling. Fuzzy C-Means is used to granulate the input space, and a scheme based on extreme learning is developed to train the batch neural network. It is proved that the network approximates continuous functions in compact domains, i.e. it is a universal approximator. Subsequently, an evolving version of the network is developed based on the described model, using online clustering methods and recursive extreme learning. It is postulated, and computational experiments endorse, that the evolving neuro fuzzy network share equal or better approximation ability in dynamic environments than its static equivalent.

Keywords: hybrid neural networks, unineurons, uninorms, evolving systems, online learning, extreme learning

1. Introduction

Fuzzy neural networks provide a framework to build hybrid models based on fuzzy set theory and neural network learning techniques. The goal is to combine the best of both worlds, mainly the ability to learn from data while generating a more transparent model. The use of t-norms and t-conorms provide an extension to the two valued and or logic operations. Exploring these extensions, fuzzy and or neurons and networks were suggested in [1].

Preprint submitted to Neurocomputing

January 9, 2013
A generalization of t-norms and s-norms called uninorms was studied in [2]. Recently it was used to construct logic neurons in [3], [4] and [5]. These works use uninorms at some level of the neuron models aiming at flexibility as a uninorm can have its identity element $e$ anywhere in the unit interval, while for t-norms $e = 1$ and for s-norms $e = 0$.

Today the amount of data produced by information systems is rising fast, increasing the need for methods to deal with large volume of high dimensional data in an online manner. With this in mind a new approach to system modeling called Evolving Systems was proposed [6]. Here, evolving means continuous gradual change and adaptability. Evolving models update their structures and parameters in an online fashion, which means that they should learn continuously from a stream of data. Successful examples of Evolving Systems have been reported in [7, 8, 9].

In this paper we address a neuron model based on uninorm called unineuron. Here, the unineuron uses uninorms at both, local synaptic processing level and global aggregation level. We emphasize a network structure that has two parts, the first corresponding to a fuzzy inference system, and the second to an aggregation neural network. The first part has fuzzy membership functions in the input layer neurons plus a subsequent layer of unineurons. A classical sigmoidal neurons layer composes the second part. A training procedure based on FCM clustering and extreme learning concept is developed. It is proved that this construction approximates continuous functions in compact domains, i.e. it is a universal approximator.

An evolving version of the described model is developed using two online clustering alternatives, namely eClustering+ and ePL, to granulate the input space and a recursive version of extreme learning to adjust the output weights. Computational experiments were conducted considering benchmark time series forecasting problems to evaluate and compare the performance of the uninorm-based hybrid neural networks. The results show that, from the point of view of approximation error, the uninorm-based network is highly competitive with alternative approaches. The evolving version share equivalent or better approximation capability in dynamic environments than its static equivalent.

After this introduction, the paper proceeds as follows. Section 2 presents the structure of the model and the static learning procedure. Next, Section 3 discuss the universal approximation capability of the model. Thereafter, the evolving version is proposed in Section 4. Computational experiments results are summarized in Section 5. Section 6 concludes the paper summarize its contributions and addressing issues for further investigation.
2. Uninorm-based Neural Network

For short, we call the class of hybrid fuzzy neural network models XUninet (eXtreme learning Uninetwork) in the remaining of this paper. We start reviewing the concept of uninorm and defining unineurons. The learning procedure that develops the structure and weights of XUninet are detailed next.

2.1. Uninorms and Unineurons

A generalization of t-norms and t-conorms called uninorm was introduced by [2]. Formally, a uninorm is a mapping \( u : [0,1] \times [0,1] \rightarrow [0,1] \) with the following properties:

1. Commutativity: \( a u b = b u a \)
2. Associativity: \( a u (b u c) = (a u b) u c \)
3. Monotonicity: if \( b \leq c \), then \( a u b \leq a u c \)
4. Identity element: \( a u e = a \), \( \forall a \in [0,1] \)

While t-norms fulfill the first three conditions with the identity element \( e = 1 \), t-conorms satisfies the first three as well but with identity element \( e = 0 \). Uninorms extends triangular norms and conorms by allowing the identity element to be a value in the unity interval.

Fuzzy neurons were introduced as basic units of fuzzy neural networks in [1]. The two basic neuron models, called or and and, are defined using t-norms and s-norms as follows:

\[
\text{or}(A, W) = S_{i=1}^n(a_i t w_i)
\]
\[
\text{and}(A, W) = T_{i=1}^n(a_i s w_i)
\]

Here \( T \) is a t-norm, \( S \) is a s-norm, \( a_i \in [0,1] \) are the inputs and \( w_i \in [0,1] \) the synaptic logic weights. Neural networks based on these neurons were successfully used in various applications such as thermal modeling of power transformers [10] and time series prediction [11]. Recent work has suggested uninorm-based neurons (unineurons) [3] to explore uninorms in modeling distinct neuron operators, a desired characteristic to add flexibility in fuzzy neural networks. Further improvements of the unineuron model were reported in [4] and [5]. The unineuron used in the XUninet is of the type \( U_U \), which consists of uninorms at synaptic processing level and at global aggregation level, as in (3).

\[
U_U = U_{i=1}^n(a_i u w_i)
\]

The uninorm realization used in this paper adopts the following construct:
\[ u(a,b) = \begin{cases} 
  e + (1 - e) \left( \frac{a-e}{1-e} \right) & \text{if } a, b \in (e, 1] \\
  e^{\left( \frac{a}{e} \cdot \frac{b-e}{e} \right)} & \text{otherwise} 
\end{cases} \tag{4} \]

The t-norm \( t \) and s-norm \( s \) are the algebraic product \( ab \) and the probabilistic sum \( a + b - ab \), respectively. The aim is to get a smoother surface because the usual min-max choice produces abrupt changes near \((a, e)\) or \((e, b)\) [2].

2.2. Network Structure

The structure of the uninetwork adopted in this paper is similar to one studied in [12]. It has a feedforward topology with two major parts: a fuzzy inference system and an aggregation neural network, assembled through three layers. The fuzzy inference system is composed by the first two layers. The input layer consists of neurons whose activation functions are membership functions of fuzzy sets that granulate the input space to form fuzzy partitions. Input space granulation is done by a clustering procedure which produces \( c_L \) cluster centers. Here we adopt Gaussian membership functions centered at \( c_l \) with dispersion (radius) \( \sigma \).

The membership degree of input \( x_i \) is computed using (5). For each dimension \( x_i \) of the \( n \)-dimensional input vector \( \mathbf{x} \) there are \( L \) fuzzy sets \( A^i_l \), \( l = 1, \ldots, L \). \( L \) corresponds to the number of equivalent fuzzy rules that assembles the system model.

\[ a_{li} = e^{-\frac{(x_i - c_{li})^2}{2\sigma^2}} \tag{5} \]

here \( l = 1, \ldots, L, i = 1, \ldots, n \) and \( c_{li} \) is the \( i \)th coordinate of the \( l \)th cluster center.

The radius \( \sigma \) is defined a priori and kept constant during the process.

The second layer contains \( U_U \) unineurons to aggregate the outputs of the input layer neurons weighted by synaptic logic weights \( w_{li} \):

\[ z_l = U_U^{li} (a_{li} w_{li}) \tag{6} \]

here \( z_l, l = 1, \ldots, L \) is the output of the \( l \)th unineuron and \( w_{li} \) are the synaptic weights. The third layer is a neural network layer with sigmoidal activation functions \( f(\cdot) \) whose outputs are:

\[ \tilde{y}_j = f \left( \sum_{l=1}^{L} r_{jl} z_l \right) \tag{7} \]

where \( m \) is the dimension of the output space, \( j = 1, \ldots, m \), and \( r_{jl} \) are the output weights connecting the \( j \)th output with the \( l \)th rule. Figure 1 shows the uninetwork structure.
2.3. Training Procedure

Most batch fuzzy neural networks rely on a clustering method to granulate the input space. The Fuzzy C-Means method is a common choice and it was also used in this paper. Cluster centers, its radii and the normalized input data for an example application are shown in Figure 2.

Extreme learning is a method developed as a way to train single hidden layer feedforward neural networks (SLFNNs). Originally introduced in [13], the method chooses the hidden layer weights randomly and estimates the weights of the output layer using the least squares algorithm. Results indicate good performance and greater ability to generalize [14]. Interestingly, the authors have shown that general nonlinear activation functions can be used by the hidden layer neurons.

The XUninet training is performed similarly assigning random values in $(0, 1)$ to weights and identity elements of the $U_{ij}$ neurons, and next the least squares
method is used to update the output layer weights matrix $R_{L \times m}$:

$$
R = (Z^T Z)^{-1} Z^T f^{-1}(Y) \tag{8}
$$

with $f^{-1}(Y) = \log(Y) - \log(1 - Y)$, $Y_{N \times m}$ being target values matrix with $N$ samples of $m$ dimension and $Z_{N \times L}$ the logic layer output matrix, which is obtained by propagating the $N$ input training samples $x$ through the network. Algorithm 1 shows the main steps of the XUninet learning.

**Algorithm 1 XUninet training**

- read and normalize the data
- run FCM to partition the input space
- randomly assign weights and identity elements
- propagate the input samples through the network
- apply least squares method

### 3. Universal Approximation

In this section we will prove that XUninet have the universal approximation ability.

**Theorem 1.** Given an XUninet model described in this paper, for $N$ arbitrary distinct samples $(x_t, y_t)$, where $x_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$, with $N$ rules ($N = L$),
membership functions $A_l$ centered at $x_t$ when $l = t$ and close to zero radii, i.e. $\sigma_l \to 0$. For any $w_{li}$ and $e$ identity elements of the uninorms randomly chosen from $(0, 1)$, with probability one, the logic layer output matrix $Z$ is invertible and $||Zr - Y|| = 0$.

**Proof.** When the membership functions $A_l$ are centered at the input samples $x_t$ when $l = t$ and have close to zero radii, the outputs $a_l$ will be at level 1 only when $l = t$. In other words, the inputs of the $l$th unineuron will be one only on $l = t$ while all the other unineurons will have 0 at the inputs. Let $z_l(1)$ be the output of the $l$th unineuron when the inputs are all 1 and $z_l(0)$ when the inputs are all 0. The logic layer output matrix $Z_{N \times N}$ will be of the following form:

$$Z = \begin{pmatrix}
z_1(1) & z_2(0) & \cdots & z_N(0) \\
z_1(0) & z_2(1) & \cdots & z_N(0) \\
\vdots & \vdots & \ddots & \vdots \\
z_1(0) & z_2(0) & \cdots & z_N(1)
\end{pmatrix}$$

The matrix $Z$ can be rewritten as $Z = B + pq^T$, where $B$ is:

$$B = \begin{pmatrix}
z_1(1) - z_1(0) \\
0 \\
z_2(1) - z_2(0) \\
0 \\
\vdots \\
z_N(1) - z_N(0)
\end{pmatrix}$$

$p$ is an $N \times 1$ column vector of ones and $q^T = [z_1(0), z_2(0), \ldots, z_N(0)]$.

Knowing that uninorms have the monotonicity property, as well as that we generated the weights in $(0, 1)$, it is true that $z_l(1) > z_l(0)$. We can then conclude that $B$ is invertible because $z_l(1) - z_l(0) > 0$ making its determinant greater than 0.

Sherman-Morrison formula [15] gives an explicit form for the inverse of $Z$ in the way that we presented:

$$Z^{-1} = (B + pq^T)^{-1}$$

$$= B^{-1} - \frac{B^{-1}pq^TB^{-1}}{1 + q^TB^{-1}p}$$

We already showed that $B^{-1}$ exists, further on, since all elements of $B$ are greater than zero and it is a diagonal matrix, its inverse will have only positive elements. By definition $p$ have only positive elements and elements from $q$ are non-negative. Therefore $q^TB^{-1}p \geq 0$ and the denominator $1 + q^TB^{-1}p$ is always greater or equal to 1 proving that $Z$ is invertible.
Note that we generated the logic weights $w \in (0, 1)$ to avoid the degenerate cases where $e = 1$ and $w = 0$ or $e = 0$ and $w = 1$ may result in $z_n(0) = z_n(1)$. In practice this is not an issue because if you use standard uniform distribution $U(0, 1)$ samples to generate the weights, the probability of these cases occur is very close to zero. Mathematically is zero but the discretization of machines gives us a small probability.

**Theorem 2.** Given a small value $\varepsilon > 0$ and an XUninet model, for $N$ arbitrary distinct samples $(x_t, y_t)$, where $x_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$, with $L$ rules where $L \leq N$. For any $w_{il}$ and $e$ identity elements of the uninorms randomly chosen from $(0, 1)$, with probability one, $||Zr - Y|| < \varepsilon$.

**Proof.** The theorem is valid because otherwise we could use $N = L$ and get $||Zr - Y|| = 0$ as a result from Theorem 1.

### 4. Evolving Uninorm-based Network

In this section we propose an evolving version of the XUninet, called hereinafter eXUninet. The network structure is similar to the previously described, except that we have the possibility to add and remove rules as new information from the system arrives. So the number of rules and membership functions are indexed by time, as seen on Figure 3.

Two alternatives for online clustering were considered and are described in the next sections.

#### 4.1. Clustering Procedures

Most batch and online fuzzy neural networks relies on a clustering method to granulate the input space. We consider two alternative online clustering procedures to endow the model with evolving capabilities: eClustering+ [?] and ePL [16].

**4.1.1. eClustering+**

The first procedure estimates the density recursively of each data point using (11).

$$D_t(o_t) = \left( (t-1) (so_t + 1) + b_t - 2 \sum_{j=1}^{n+m} o_{tj} h_{tj} \right)^{-1}$$

(11)

with $o_t = [x^T, y^T]^T$; $D_1(o_1) = 1$; $b_t = b_{t-1} + \sum_{j=1}^{n+m} o_{(t-1)j}^2$; $b_1 = 0$;

$h_{tj} = h_{(t-1)j} + o_{(t-1)j}$; $h_{1j} = 0$; $j = 1, \ldots, n + m$; $so_t = \sum_{j=1}^{n+m} o_{tj}^2$.

Density is calculated recursively for each cluster center using (12):
Fuzzy Inference System Aggregation Neural Network

Figure 3: Evolving uninetwork structure

\[
D_t(o^*) = \frac{t - 1}{t - 1 + (t - 2) \left( \frac{1}{D_{t-1}(o^*)} - 1 \right) + \text{dist}} \tag{12}
\]

where \( \text{dist} = \sum_{j=1}^{n+m} (o_{ij} - o_{(t-1)j}) \) and \( D_1(o^*) = 1 \). Cluster centers candidates are selected if the following condition is true: Condition A: \( D_t(o_i) > \max D_t(o^*) \) or \( D_t(o_i) < \min D_t(o^*) \) where \( i^* \) are the indexes of cluster centers. To avoid overlapping and redundancy, the cluster is created only if it does not satisfy: Condition B: \( \exists \ell \in [1, L^t] : a_{i\ell} > e^{-1}, \forall i \in [1, n] \). Algorithm 2 summarizes the eClustering+ procedure.
Algorithm 2 eClustering+

read first data sample
initialize first cluster center $o^* \leftarrow o_1$;
$D_1(o_1) \leftarrow 1; b_1 \leftarrow 0; h_{1j} \leftarrow 0; D_1(o^*) \leftarrow 1;

while there are incoming data do

read input-output vector: $o_t$
update density using (11)
update clusters density using (12)

if Condition A holds then
form a new cluster center
$L^t \leftarrow L^t + 1; o^* \leftarrow o_t; D(o^*) \leftarrow 1;$
end if

if Condition B holds then
remove the cluster for which the condition holds
end if

end while

4.1.2. ePL

An alternative is to use the ePL (evolving Participatory Learning) clustering algorithm of [16]. Participatory learning assumes that learning and knowledge about a system depend on what the learning mechanism knows about the system itself. It uses a scheme with a compatibility measure $\rho$ and an arousal index $\gamma$. If a data sample with high compatibility arrives, then learning proceeds. Else, if data samples with low compatibility (low $\rho$) are input, then the system assumes new knowledge in the data and creates a new cluster center.

The compatibility of input $x^t$ with the $i$-th cluster is computed as follows:

$$\rho^t_i = 1 - \frac{||x^t - v_i||}{n}$$

The arousal index $\gamma$ is updated using:

$$\gamma_{i}^{t+1} = \gamma_{i}^{t} + \beta(1 - \rho_{i}^{t+1} - \gamma_{i}^{t})$$

The value of $\beta \in [0, 1]$ controls the rate of change of arousal: the closer $\beta$ is to one, the faster the system is to sense compatibility variations. If the value of the arousal index exceeds a threshold $\tau \in [0, 1]$, then a new cluster centered at $x^t$ is created. Otherwise, the most compatible cluster center, i.e., the one with the highest $\rho$, is updated using:

$$v_i^{t+1} = v_i^t + \alpha \rho_i^{t+1}(x^t - v_i^t)$$
Compatibility between centers are also computed to check for redundancy:

\[ \rho_{t \nu_i}^l = 1 - \sum_{j=1}^{L_t} |v_i - v_j| \]  

That is, if \( \rho_{t \nu_i}^l > \lambda \), \( \lambda \in [0, 1] \), then the cluster is excluded for being redundant. Figure 3 summarizes the ePL procedure.

**Algorithm 3 ePL clustering**

1. read first data sample
2. initialize first cluster center \( v_1^1 \leftarrow x_1^1 \); \( \gamma_1^1 \leftarrow 0 \);
3. **while** there are incoming data **do**
   1. read input vector \( x_t \)
   2. **for** \( i = 1 \) to \( L_t \) **do**
      1. update \( \rho_t^i \) using (13)
      2. update \( \gamma_t^i \) using (14)
   **end for**
   3. if \( \gamma_t^i > \tau, \forall i \in [1, L_t] \) then
      1. form a new cluster centered at \( x_t \)
   **else**
      1. Update the cluster center with highest \( \rho \) using (15)
   **end if**
   4. Compute \( \rho_{t \nu_i}^l \) using (16)
   5. **if** \( \exists \rho_{t \nu_i}^l \) such that \( \rho_{t \nu_i}^l > \lambda \) then
      1. \( v_t^i \) is redundant; remove the \( i \)-th cluster;
   **end if**
4. **end while**

4.2. Parameters Adjusting

The eXUninet training is performed using extreme learning as well. Random values in \([0, 1]\) are assigned to weights and identity elements of the \( U_U \) neurons. Next it uses the recursive least squares (RLS) algorithm with forgetting factor \( \lambda \) to update the output layer weights \( r^t = [r_{11}^t \ldots r_{jl}^t \ldots r_{mL_t}^t] \):

\[ J^t = Q^{t-1} z^t \{ \lambda + z^t Q^{t-1} (z^t)^T \}^{-1} \]  
\[ Q^t = (I_{L^t} - J^t z^t) \lambda^{-1} Q^{t-1} \]  
\[ r^t = r^{t-1} + (J^t)^T (f^{-1}(y^t) - z^t (r^{t-1})^T) \]
where \( f^{-1}(y^t) = \log(y^t) - \log(1 - y^t) \). Initialization of \( Q \) is commonly \( I_L \omega \), \( \omega = 1000 \), where \( I_L \) is the identity matrix. Algorithm 4 shows the eXUninet learning procedure.

**Algorithm 4 eXUninet learning procedure**

1. read first data sample \( x^1 \)
2. set the first cluster center at \( x^1 \)
3. initialize the model structure and parameters
4. **while** there are incoming data **do**
   1. read input \( x^t \)
   2. estimate output \( \hat{y}^t \)
   3. read output \( y^t \)
   4. run one clustering step
   5. update model to match current cluster structure
   6. update model parameters using (17) - (19)
5. **end while**

The evolving version capabilities suggest that eXUninet approximates functions in dynamic environments because the same structure is used. However, a formal proof is not trivial and currently is under investigation. This is because the evolving version uses an online clustering algorithm that may add, remove, or modify clusters when either new data are input or concept drifts occur. Moreover, it is not simple to prove RLS algorithm convergence in nonstationary environments, and it also depends on the forgetting factor [17].

The static neural fuzzy model may eventually help to prove the evolving case and, generally speaking, proofs may not be too different from what was stated above. In fact, a series of computational experiments and simulations shows that the evolving network with \( L \ll N \) (as it is the case in actual, especially online and real-time applications) performs equally or even better than its batch version. This is due to its capability to continuously explore local approximations instead of trying to learn in a global manner, which usually is a much more difficult task.

A major element of adaptability is the forgetting factor of the RLS algorithm. Figure 4 shows the impact of using different values of forgetting factor (100 runs for each). The figure shows that when \( \lambda \) is closer to one, more conservative adjusting of the parameters were performed resulting on more frequent missing of the peaks, valleys and phase changes. When the value is closer to one, the model is considering a wider exponential window, i.e. it uses a greater number of past samples to estimate the current parameter value.
Figure 4: Impact of $\lambda$ in eXUninet+ evolving learning
5. Computational Experiments

Simulations were performed using a 2.27GHz dual-core personal computer and MATLAB. The forgetting factor was fixed at 0.9. The root mean squared error (RMSE) at step \( t \) is computed using:

\[
RMSE = \sqrt{\frac{1}{t} \sum_{j=1}^{t} (y_j - \hat{y}_j)^2}
\]  

The two clustering approaches are identified by eXUninet+ when eClustering+ is used, and by eXUninet-ePL when ePL is adopted. Data were normalized in the range \([0.1, 0.9]\), and the errors computed for normalized data. All the results reported here were obtained after running eXUninet and XUninet 100 times.

The best RMSE value and the average RMSE value are displayed. For the static version, the best RMSE value makes sense since we can choose the best network to be used on the application. The average error should be more important for the evolving version, since we cannot know which set of weights is better a priori. Using cross validation is appropriate for static training but makes no sense for an online method. For comparison we used the whole set for training the batch network and computed the error for the same set. The effects of overfitting and local minima are avoided as extreme learning is being used [14] and the number of rules is being set as equal to the number found by eClustering+.

5.1. Mackey-Glass Time Series

The Mackey-Glass time series is a well known benchmark whose data is generated using:

\[
\frac{dx}{dt} = \frac{Ax^{t-\tau}}{1 + (x^{t-\tau})^C} - Bx^t, \quad A, B, C > 0
\]  

Semi-periodic or chaotic behaviour depends on the parameters values chosen. Several studies adopt: \( A = 0.2 \), \( B = 0.1 \), \( C = 10 \) and \( \tau = 17 \), with a time step of 0.1 for integration [18, 7], 3200 data samples were generated and data from \( t = 200 \ldots 3200 \) were used for training and testing simultaneously. The goal is to predict the value of \( x^t \) 85 steps ahead, that is:

\[
x^{t+85} = p(x^t, x^{t-6}, x^{t-12}, x^{t-18})
\]

Results comparing the best and average RMSE values with alternative approaches reported in the literature are shown in Table 1.
Table 1: Performances for Mackey-Glass

<table>
<thead>
<tr>
<th>Model</th>
<th>Ref.</th>
<th>Final # of Rules</th>
<th>RMSE</th>
<th>AVG. RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTS</td>
<td>[18]</td>
<td>24</td>
<td>0.0779</td>
<td>0.0779</td>
</tr>
<tr>
<td>DENFIS</td>
<td>[7]</td>
<td>25</td>
<td>0.0730</td>
<td>0.0730</td>
</tr>
<tr>
<td>FBeM</td>
<td>[9]</td>
<td>26</td>
<td>0.0968</td>
<td>0.0968</td>
</tr>
<tr>
<td>eXUninet+</td>
<td>-</td>
<td>27</td>
<td>0.0433</td>
<td>0.0567</td>
</tr>
<tr>
<td>eXUninet-ePL</td>
<td>-</td>
<td>27</td>
<td>0.0507</td>
<td>0.0599</td>
</tr>
<tr>
<td>XUninet</td>
<td>-</td>
<td>27</td>
<td>0.1142</td>
<td>0.1310</td>
</tr>
</tbody>
</table>

For this case the eXUninet AVG. RMSE was better than the others with 99% confidence. Training took an average of 6.48s for the entire dataset, an average of 2.16ms per sample. XUninet training took an average of 5.15s and performed worse due to its non dynamic behaviour. Figure 5 depicts the results of eXUninet and XUninet for the first 500 data samples.

The methods addressed in this section utilize parameters to tune the final number of rules achieved. Although in some cases one can get lower errors or better clusters structures by optimizing these parameters, the goal was to demonstrate that the uninetworks does provide competitive results with considerable lower training time and effort. Extensive tests considering parameters variations still need to be done to statistically evaluate its performance. As eXUninet+ does not make use of parameters for clustering, when possible, parameters of the other methods were adjusted to match approximately the same number of rules. This is also the case for the next example.
5.2. **Box-Jenkins Gas Furnace**

Box-Jenkins gas furnace [19] also is a well known benchmark identification problem. The furnace input is the oxygen-methane gas flow rate $x$ and the output is the concentration of carbon dioxide emitted $y$. Positive correlation between the variables indicates the use of a predictor of the following form:

$$y_t = p(y_{t-1}, x_{t-4})$$

(23)

Table 2 shows the results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Ref.</th>
<th>Final # of Rules</th>
<th>RMSE</th>
<th>AVG. RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>eTS</td>
<td>[18]</td>
<td>12</td>
<td>0.0796</td>
<td>0.0796</td>
</tr>
<tr>
<td>DENFIS</td>
<td>[7]</td>
<td>12</td>
<td>0.0190</td>
<td>0.0190</td>
</tr>
<tr>
<td>FBeM</td>
<td>[9]</td>
<td>3</td>
<td>0.0421</td>
<td>0.0421</td>
</tr>
<tr>
<td>eXUninet+</td>
<td>-</td>
<td>11</td>
<td>0.0559</td>
<td>0.0698</td>
</tr>
<tr>
<td>eXUninet-ePL</td>
<td>-</td>
<td>12</td>
<td>0.0512</td>
<td>0.0619</td>
</tr>
<tr>
<td>XUninet</td>
<td>-</td>
<td>12</td>
<td>0.0327</td>
<td>0.0474</td>
</tr>
</tbody>
</table>

In this case, eXUninet+ on average, performed better than the eTS with 99% confidence, but slightly worse than the remaining methods. Training took 294ms, 1ms per sample. The best XUninet was significantly better than eTS and FBeM. The static version of the uninetwork performed better than the evolving one, presumably, because this application is simpler, so that the potential of an evolving method is not necessary in this case. Figure 6 depicts the results of eXUninet-ePL and XUninet.

6. **Conclusion**

This paper has suggested learning approaches for hybrid fuzzy neural networks based on uninorms. Computational results show that the fuzzy neural network is competitive with alternative evolving methods. The learning approach is simple, fast, accurate, and suitable for online and real time applications.

A static version of the fuzzy neural network was shown to be a universal approximator, and computational experiments suggest that the eXUninet may have this property once the structure used is the same. A more general proof of approximation capability of the evolving version needs to consider the time-varying nature of the clustering procedure, and the nature of RLS algorithm with forgetting factor and time-varying parameters.
Future work shall formalize the various techniques used to construct the eX-Uninet. Detailed statistical analysis is also required to verify if the evolving learning approach is statistically superior in static and dynamic environments. One can also analyse the use of various types of uninorms realizations at the global and local levels of neurons in the fuzzy layer of the eXUninet, as well as extensions for nullnorms-based neural fuzzy networks.

References


