Chapter 2

Artificial Neural Networks and Evolving Connectionist Systems

Classical artificial neural networks (ANN) were developed to learn from data. Evolving connectionist systems (ECOS) were further developed by the author and taken further by other researchers not only to learn in an adaptive, incremental way from data that measure evolving processes, but to extract rules and knowledge from the trained systems. Both methods were initially inspired by some principles of learning in the brain, but then they were developed mainly as machine learning and AI tools and techniques, with a wider scope of applications. Many of the architectures and learning methods of ANN and ECOS were used in the development of SNN, deep learning systems and brain-inspired AI discussed in other chapters of the book.

The chapter is organised in the following sections:

2.1. Classical Artificial Neural Networks: SOM, MLP, CNN, RNN.
2.2. Hybrid and knowledge-based ANN: Opening the "black box"
2.3. Evolving Connectionist Systems
2.4. Evolving Fuzzy Neural Networks. EFuNN
2.5. Dynamic Evolving Neuro Fuzzy Systems. DENFIS
2.6. Other ECOS methods and systems
2.7. Summary and further readings for deeper knowledge

2.1 Classical Artificial Neural Networks: SOM, MLP, CNN, RNN.

ANNs are computational models that mimic the nervous system in its main function of adaptive learning and generalization. ANNs are universal computational models. One of the most popular artificial neuron models is the McCulloch and Pitts neuron developed in 1943. It was used in early ANNs, such as Rosenblatt's Perceptron [1] and multilayer perceptron [2-5]. A simple example is given in Fig. 1.

Various types of ANN architectures and learning algorithms have been developed, e.g.:

- Self-Organising maps (SOM) and unsupervised learning algorithms [6];
- Multilayer perceptrons (MLP) and back propagation supervised learning algorithm [4,5];
- Adaptive Resonance Theory (ART) [7].
- Recurrent ANN and reinforcement learning [8].
- Convolutional and deep learning ANN [9].

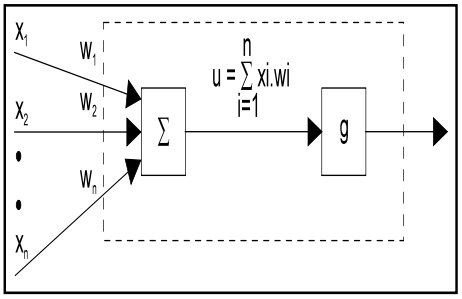


Figure 1. A diagram of a simple artificial neuron.

This section covers some of models of ANN that have influenced the development of the brain-like spiking neural networks (SNN) and brain-inspired AI techniques presented in other chapters of the book.

2.1.1 Unsupervised learning in neural networks. Self-Organising Maps (SOM)

Unsupervised learning is concerned with finding structures in the data. Techniques include:

- Clustering of data;
- Vector quantisation;

A basic technique to apply when finding structures in data is measuring distance (or similarity) between data vectors (data samples).

Measuring distance (*or similarity*) is a fundamental issue in all statistical and ANN learning methods. The following are some of the most used methods for measuring distance, illustrated on two, n-dimensional data vectors $\mathbf{x} = (x_1,x_2,\ldots,x_n)$ and $\mathbf{y} = (y_1,y_2,\ldots,y_n)$:

- Euclidean distance:

$$D(\mathbf{x},\mathbf{y}) = \sqrt{\ [\ (\sum_{i=1,..,n} (x_i - y_i)^2\ ) / n\ ]} \qquad (1)$$

- Hamming distance :

$$D(\mathbf{x},\mathbf{y}) = \ (\sum_{i=1,..,n}\ |\ x_i - y_i\ |\ ) / n\ \}\ , \qquad (2)$$

where absolute values of the difference between the two vectors are used.

- Local fuzzy normalized distance ([10]):

A local normalized fuzzy distance between two fuzzy membership vectors $\mathbf{x}_f$ and $\mathbf{y}_f$ that represent the membership degrees to which two real vector data **x** and **y** belong to pre-defined fuzzy membership functions is calculated as:

$$D(\mathbf{x_f},\mathbf{y_f}) = ||\mathbf{x}_f - \mathbf{y}_f\ ||\ /\ ||\mathbf{x}_f + \mathbf{y}_f||, \qquad (3)$$

where: $||\mathbf{x}_f - \mathbf{y}_f||$ denotes the sum of all the absolute values of a vector that is obtained after vector subtraction (or summation in case of $||\mathbf{x}_f + \mathbf{y}_f||$) of two vectors $\mathbf{x}_f$ and $\mathbf{y}_f$ of fuzzy membership values; " / " denotes division.

- Cosine distance :

$$D = 1 - SUM(\sqrt{xiyi} \Big/ \sqrt{xi^2}\sqrt{yi^2}) \quad (4)$$

- Correlation distance :

$$D = 1 - \sum_{i=1}^{n}(xi - \overline{xi})(yi - \overline{yi}) \Big/ \sum_{i=1}^{n}(xi-\overline{xi})^2 (yi-\overline{yi})^2 \quad (5)$$

where $\overline{xi}$ is the mean value of the variable x_i.

Many unsupervised learning neural network methods are based on clustering of input data. Clustering is the process of defining how data are grouped together based on similarity.

Clustering results in the following outcomes:

- Cluster centres: these are the geometrical centres of the data grouped together; they can be either pre-defined (batch- mode clustering), or not defined a priori but evolving;
- Membership values, defining for each data vector to what cluster it belongs to. This can be either a crisp value of 1 (the vector belongs to a cluster), or 0 – it does not belong to a cluster (as it is in the k-means method), or a fuzzy value between 0 and 1 showing the level of belonging – in this case the clusters may overlap (fuzzy clustering).

Self-Organizing Maps - SOMs

Here, the principles of the traditional SOMs are outlined first , and then some modifications that allow for dynamic, adaptive node creation, are presented.

Self-organizing maps belong to the vector quantisation methods where prototypes are found in a prototype (feature) space (map) of dimension l rather than in the input space of dimension d, l<d. In Kohonen's self-organizing feature map (SOM) [11-14] the new space is a topological map of 1-, 2-, 3-, or more dimensions (Fig.2).

The main principles of learning in SOM are as follows:

• Each output neuron specializes during the training procedure to react to similar input vectors from a group (cluster) of the input space. This characteristic of SOM tends to be biologically plausible as some evidences show that the brain is organised into regions which correspond to similar sensory

stimuli. A SOM is able to extract abstract information from multi-dimensional primary signals and to represent it as a location, in one-, two-, and three- etc. dimensional space.

• The neurons in the output layer are competitive ones. Lateral interaction between neighbouring neurons is introduced in such a way, that a neuron has a strong excitatory connection to itself, and less excitatory connections to its neighbouring neurons in a certain radius; beyond this area, a neuron either inhibits the activation of the other neurons by inhibitory connections, or does not influence it. One possible neighbouring rule that implements the described strategy is the so called *"Mexican hat*" rule. In general, this is "the winner-takes all" scheme, where only one neuron is the winner after an input vector was fed, and a competition between the output neurons has taken place. The fired neuron represents the class, the group (cluster), the label, or the feature to which the input vector belongs.

• SOMs transform or preserve similarity between input vectors from the input space into *topological closeness of neurons* in the output space represented as a topological map. Similar input vectors are represented by near points (neurons) in the output space. Example is given in Fig.3

The unsupervised algorithm for training a SOM, proposed by Teuvo Kohonen, is outlined in Box 1. After each input pattern is presented, the winner is established and the connection weights in its neighbourhood area N_t increase, while the connection weights outside the area are kept unchanged. α is a learning parameter. Training is done through a number of training iterations so that at each iteration the whole set of input data is propagated through the SOM and the connection weights are adjusted.

SOMs learn statistical features. The synaptic weight vectors tend to approximate the density function of the input vectors in an orderly fashion. Synaptic vectors $\boldsymbol{w}_j$ converge exponentially to centres of groups of patterns and the nodes of the output map represent to a certain degree the distribution of the input data. The weight vectors are also called *reference vectors,* or reference codebook vectors. The whole weight vector space is called a *reference codebook.*

In SOM the topology order of the prototype nodes are pre-determined and the learning process is to “drag” the ordered nodes onto the appropriate positions in the low dimensional feature map. As the original input manifold can be complicated and an inherent dimension larger than that of the feature map (usually set as 2 for visualization purpose), the dimension reduction in SOM may become inappropriate for complex data analysis tasks.

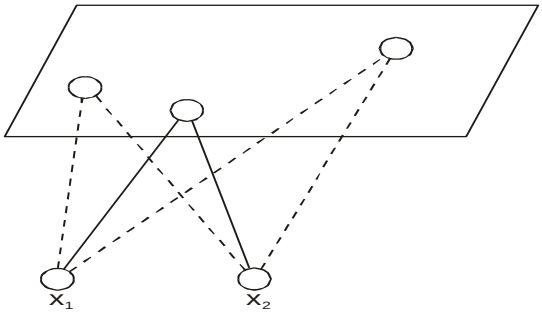


Fig. 2. Example of a simple SOM architecture of 2 input neurons and 2D output topological map (after [37])

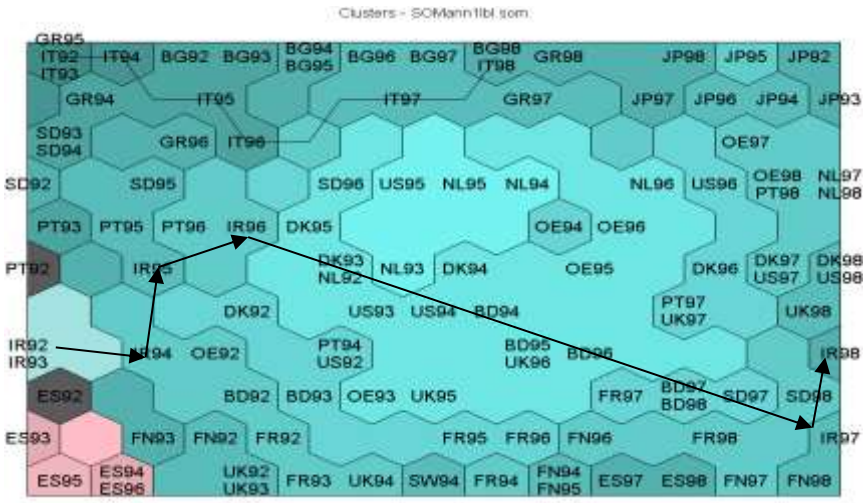


Fig.3. An example of SOM topological map trained on macroeconomic annual data of European counties. Countries with similar economic parameters are clustered together. The change in the economic development of Ireland is traced over years on the map (after [37])

Box.1 The Self-Organsing Map (SOM) training algorithm

K0. Assign small random numbers to the initial weight vectors **w**j(t=0), for every neuron j from the output map.

K1. Apply an input vector **x** at the consecutive time moment t.

K2. Calculate the distance d_j (in n-dimensional space) between x and the weight vectors **w**j(t) of each neuron j. In Euclidean space this is calculated as follows:
$d_j = sqrt((\Sigma((x_i - w_{ij})^2)))$

K3. The neuron k which is closest to x is declared the winner. It becomes a centre of a neighbourhood area Nt.

K4. Change all the weight vectors within the neighbourhood area:
$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha.(\mathbf{x} - \mathbf{w}_i(t))$, if $j \in Nt$,
$\mathbf{w}_j(t+1) = \mathbf{w}_j(t)$, if j is not from the area Nt of neighbours.

All of the steps from K1 to K4 are repeated for all the training instances. Nt and α decrease in time. The same training procedure is repeated again with the same training instances until convergence.

Some of the principles of SOM, such as topological mapping, are used and further developed in the brain-inspired SNN as discussed in chapter 6.

2.1.2. Supervised learning in ANN. Multilayer Perceptron and the Back Propagation Algorithm

Connectionist systems for supervised learning learn from pairs of data (**x,y**), where the desired output vector **y** is known for an input vector **x.**

If the model is incrementally adaptive, new data will be used to adapt the system's structure and function incrementally. If a system is trained incrementally, the generalization error of the system on the next new input vector (or vectors) from the input stream is called here *local incrementally adaptive generalization error*. The local incrementally adaptive generalization error at the moment *t,* for example, when the input vector is **x**(t), and the calculated by the system output vector is **y(**t)', is expressed as Err(t)= $||\mathbf{y}(t) - \mathbf{y}(t')||$.

The local incrementally adaptive root mean square error, and the local incrementally adaptive non-dimensional error index LNDEI(t) can be calculated at each time moment *t* as:

$$LRMSE(t)=\sqrt{(\Sigma_{i=1,2\ldots,t}(Err(i)^2)/t)}; \quad (6)$$

$$LNDEI(t)=LRMSE(t)/std(y(1): y(t)), \quad (7)$$

where: std(y(1):y(t)) is the standard deviation of the output data points from time unit 1 to time unit t.

In a general case, the global generalization root mean square error RMSE and the non-dimensional error index are evaluated on a set of *p* new (future) test examples from the problem space as follows:

$$RMSE=\sqrt{(\Sigma_{i=1,2,\ldots p}[(\mathbf{y}_i - \mathbf{y}_i')^2]/p)}; \quad (8)$$

$$NDEI=RMSE/std\ (\mathbf{y}_1: \mathbf{y}_p), \quad (9)$$

where std ($\mathbf{y}_1$: $\mathbf{y}_p$), is the standard deviation of the data from 1 to p in the test set.

After a system is evolved on a sufficiently large and representative part of the whole problem space Z, its global generalization error is expected to become satisfactorily small, similar to the off-line, batch mode learning error.

Multilayer perceptron (MLP) trained with a backpropagation algorithm (BP) use a global optimization function in both incrementally adaptive (pattern mode) training, and in a batch mode training [2, 4, 5]. The batch mode, off-line training of a MLP is a typical learning method. Figures 4 and 5 depict a typical MLP architecture and Box 2 depicts the batch mode backpropagation algorithm.

In the incremental, pattern learning mode of the backpropagation algorithm, after each training example is presented to the system and propagated through it, an error is calculated and then all connections are modified in a backward manner. This is one of the reasons for the phenomenon called catastrophic forgetting - if examples are presented only once, the model may adapt to them too much and "forget" previously learned examples, if the model is a global model. This phenomenon is illustrated on Fig.6a, where after training a MLP on a data set A it is trained on data set B and it 'forgets' a lot about data set A, etc.

In an incrementally adaptive learning mode, same or very similar examples from the past need to be presented many times again, in order for the system to properly learn new examples without forgetting them. The process of learning new examples and presenting previously used ones is called "rehearsal" training [15].

MLP can be trained in an incrementally adaptive mode, but they have limitations in this respect as they have a fixed structure and the weight optimisation is a global one if a gradient descent algorithm is used for this purpose.

A very attractive feature of the MLP is that they are universal function approximators (see [16, 17]) even though in some cases they may converge in a local minimum.

Some connectionist systems, that include MLP, use local objective (goal) function to optimise the structure during the learning process. In this case when a data pair (**x, y**) is presented, the system optimises its functioning always in a local vicinity of **x** from the input space X, and in the local vicinity of **y** from the output space Y [18].

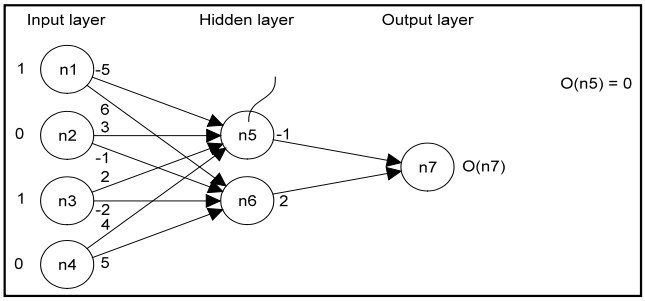


Figure 4. An example of a simple feedforward ANN (after [37]).

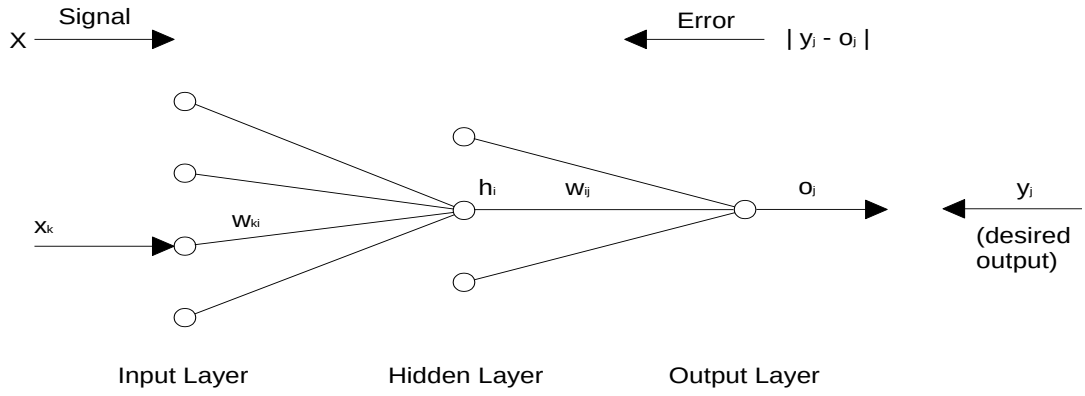


Figure 5. A schematic diagram used to explain the error backpropagation algorithm (after [37])

Box 2. The backpropagation algorithm

Forward pass:

BF1. Apply an input vector **x** and its corresponding output vector **y** (the desired output).

BF2. Propagate forward the input signals through all the neurons in all the layers and calculate the output signals.

BF3. Calculate the Err_j for every output neuron j as for example:
$Err_j = y_j - o_j$, where y_j is the jth element of the desired output vector **y**.

Backward pass:

BB1. Adjust the weights between the intermediate neurons i and output neurons j according to the calculated error:
$\Delta w_{ij}(t+1) = lrate.\ o_j(1 - o_j).\ Err_j.\ o_i + momentum.\ \Delta w_{ij}(t)$

BB2. Calculate the error Err_i for neurons i in the intermediate layer:
$Err_i = \sum Err_j.\ w_{ij}$

BB3. Propagate the error back to the neurons k of lower level:
$\Delta w_{ki}(t+1) = lrate.o_i(1 - o_i).\ Err_i.x_k + momentum.\ \Delta w_{ki}(t)$

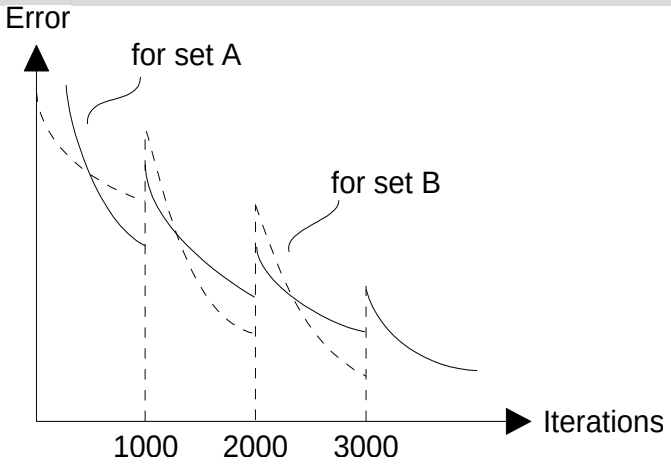


Figure 6a. Illustration of learning with catastrophic forgetting in MLP (after [37])

When a MLP is trained for too many iterations, its generalization ability (to recognize new data) may deteriorate which is known as *overfitting* (See Fig.6b)

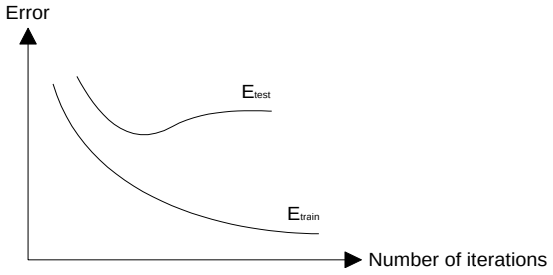


Figure 6b. Illustration of overfitting the data in MLP (after [37])

Principles of MLP and backpropagation algorithms are used and further developed for multiple layers as illustrated in Fig.7, called deep ANN.

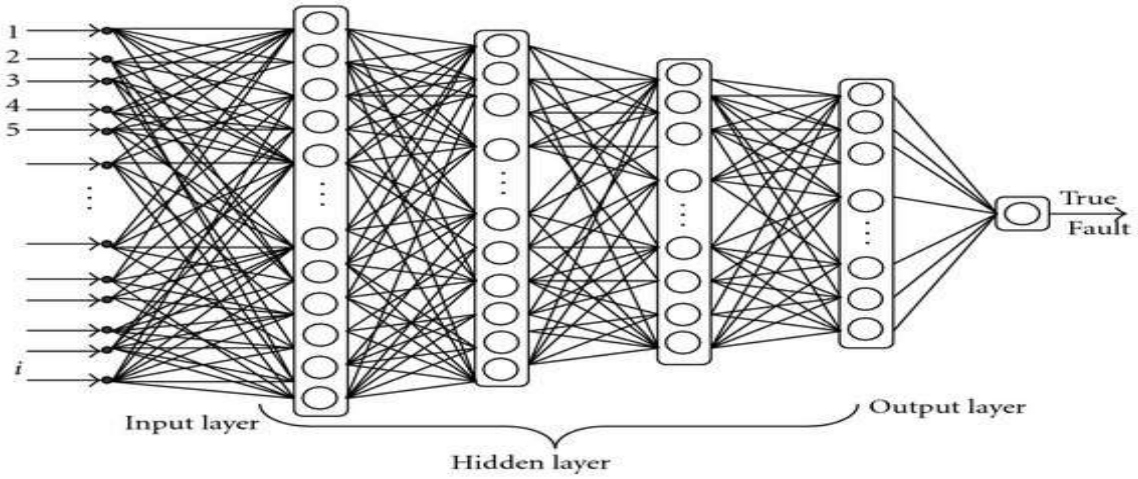


Fig.7. Multiple layers of MLP

2.1.3 Convolutional neural networks (CNN)

Fukushima proposed a biologically inspired MLP in which the first layer performs feature extraction from subspace of the image data and the other layers combine these features, similar to the visual cortex. He called these ANN Cognitron (1975) and Neocognitron (1980) [19, 20] – Fig.8. This is perhaps the first deep NN structure that was inspired by the structure and functionality of the visual cortex.

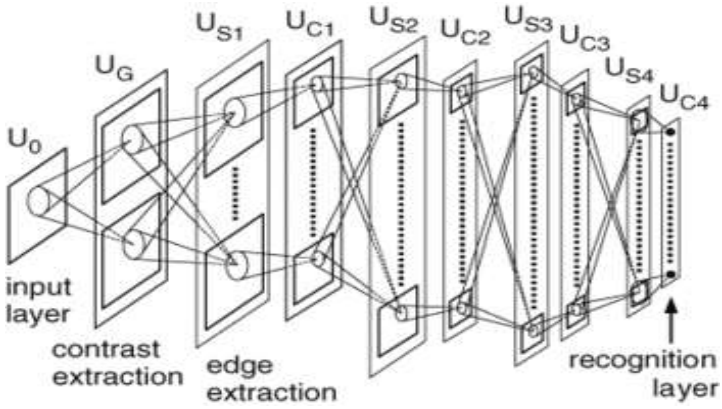


Fig.8. Fukushima's Neocognitron CNN (after [105])

The Neocognitron principles were further developed in a series of ANN called convolutional ANN (CNN) illustrated in Fig.9a and multiple layer convolutional MLP – Fig.9b.

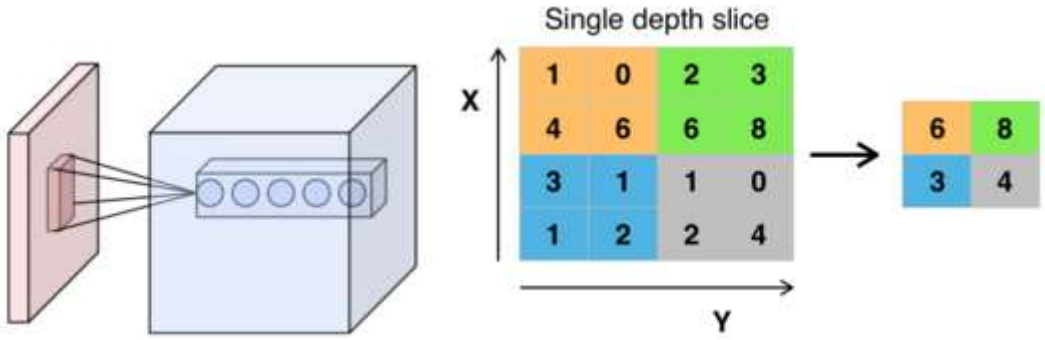


Fig.9a. Illustration of the principle of convolutional ANN (CNN) (after [105])

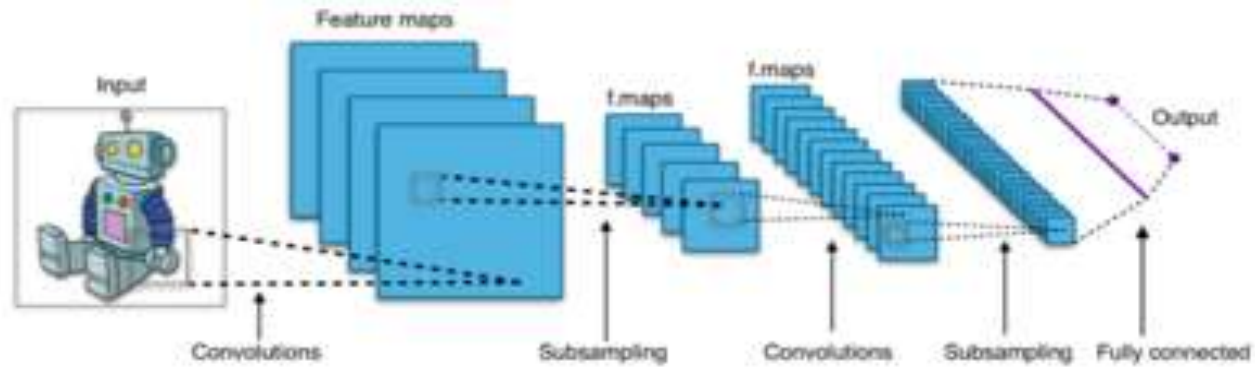


Fig.9b. Illustration of the principle of CNN and MLP with multiple layers (after [105]).

CNN and deep ANN have been successfully developed for large scale image classification (e.g. 14 mln images in the ImageNet data set), recognizing spoken words from a large corpus of data (e.g. TIMIT), structuring large repositories of data in the IBM Watson question-answering systems, playing games such as Go with human masters, etc. [110-116].

The CNN and the deep ANN are excellent tools for vector, frame- based data (e.g. image recognition), but not much for spatio-temporal data that measure evolving processes, as these models can manifest catastrophic forgetting when trained on new data incrementally. There is no *time* of asynchronous events learned in the models and they are difficult to adapt to new data and change their structures. Even though this approach allows for deep learning of vector based data across many layers of neurons, it still lacks methods for deep learning and deep knowledge representation in time-space as defined in Chapter 1.

Knowledge representation in an ANN was achieved in the knowledge–based ANN and more specially in the evolving connection systems (ECOS) as presented in the next sections. It was also achieved in evolving spiking neural networks (eSNN) in Chapter 5. Deep learning and deep knowledge representation in time-space as defined in Chapter 1 is achieved in the brain-inspired SNN as discussed in Chapter 6.

2.1.4. Recurrent and LSTM ANN

Recurrent ANN (RNN) have feedforward, feedback and later connections – Fig.10.

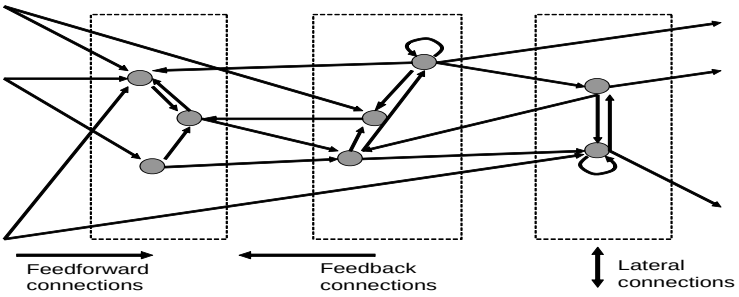


Fig.10. Recurrent ANN have feedforward, feedback and later connections (after [106]).

As a further continuation of the RNN, the so called Long-Short Term Memory (LSTM), was developed [107,108]. A LDTM RNN consists of units, each unit consisting of a cell, an input gate, an output gate and a forget gate. The cell is responsible for remembering input data over time. Each of the three gates can be thought of as an artificial neuron, as in a multi-layer (or feedforward) neural network. They compute an activation based on a weighted sum. There are connections between these gates and the cell (see [105-108] for more explanation).

2.2. Hybrid and knowledge-based ANN

Some of the ANN discussed above are considered 'black boxes' as it was difficult to interpret their internal structures and to articulate the essential knowledge learned. That led to the development of hybrid and rule based ANN that can both incorporate and extract essential information from the data and reveal new knowledge about the modelled processes.

In order to incorporate human knowledge into an intelligent system, an ANN module can be combined with a rule-based module in the same system. The rules can be fuzzy rules as a partial case [21]. An exemplar system is shown in Fig. 11, where, at a lower level, an ANN module predicts the next day value of a stock index and, at a higher level, a fuzzy reasoning module combines the predicted values with some macro-economic variables, using the following types of fuzzy rules [21]:

IF <the stock value predicted by the ANN module is high>
AND <the economic situation is good>
THEN <buy stock>

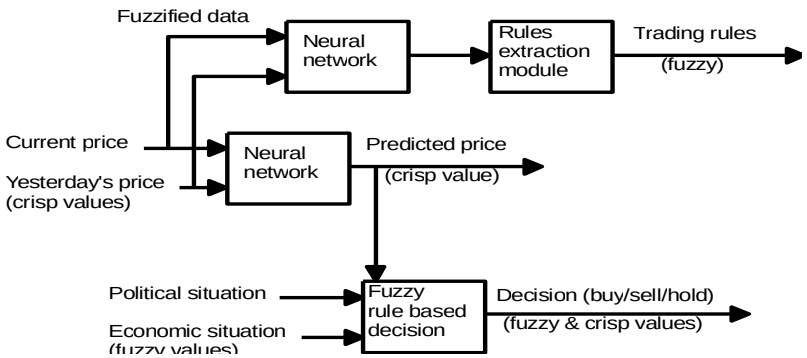


Figure 11. A hybrid ANN-fuzzy rule-based expert system for financial decision support [21].

Hybrid systems can also use crisp propositional rules, along with fuzzy rules [22]. The type of hybrid systems illustrated in from Fig. 3 are suitable to use when decision rules are available to integrate with a machine learning module that learns from incoming data.

Another group of ANN methods can be used not only to learn from data, but to extract rules from a trained ANN and/or insert rules into an ANN as initialization procedure. These are the knowledge-based neural networks (KBNN).

Types of Rules used in KBNN

Different KBNNs are designed to represent different types of rules, some of them listed below:

(1) *Simple propositional rules* (e.g., IF x1 is A AND/OR x2 is B THEN y is C, where A, B and C are constants, variables, or symbols of true/false type) (see for example, [23, 24, 25]. As a partial case, interval rules can be used, for example:

IF x1 is in the interval [x1min, x1max] AND x2 is in the interval [x2min, x2max] THEN y is in the interval [ymin, ymax], with Nr1 examples associated with this rule.

(2) *Propositional rules with certainty factors* (e.g., IF x1 is A (CF1) AND x2 is B (CF2) THEN y is C (CFc)), (see for example [26]).

(3) *Zadeh-Mamdani fuzzy rules* (e.g., IF x1 is A AND x2 is B THEN y is C, where A, B and C are fuzzy values represented by their membership functions) (see for example [27, 28]).

(4) *Takagi-Sugeno fuzzy rules* (for example, the following rule is a first order rule: IF x1 is A AND x2 is B THEN y is a.x1 + b.x2 +c, where A and B are fuzzy values and a, b and c are constants) ([29, 30]). More complex functions are possible to use in higher-order rules.

(5) *Fuzzy rules of type (3)* with degrees of importance and certainty degrees (e.g.; IF x1 is A (DI1) AND x2 is B (DI2) THEN y is C (CFc), where DI1 and DI2 represent the importance of each of the condition elements for the rule output, and the CFc represents the strength of this rule) (see [21]).

(6) Fuzzy rules that represent associations of clusters of data from the problem space (e.g., Rule j: IF [an input vector x is in the input cluster defined by its centre (x1 is Aj, to a membership degree of MD1j, AND x2 is Bj, to a membership degree of MD2j) and by its radius Rj-in] THEN [y is in the output cluster defined by its centre (y is C, to a membership degree of MDc) and by its radius Rj-out, with Nex(j) examples represented by this rule]. These are the EFuNN rules discussed in Chapter 3.

(7) *Temporal rules* (e.g., IF x1 is present at a time moment t1 (with a certainty degree and/or importance factor of DI1) AND x2 is present at a time moment t2 (with a certainty degree/importance factor DI2) THEN y is C (CFc)).

(8) *Temporal, recurrent rules* (e.g., IF x1 is A (DI1) AND x2 is B (DI2) AND y at the time moment (t-k) is C THEN y at a time moment (t+n) is D (CFc)).

(9) *Type-2 fuzzy rules*, that are fuzzy rules of the form of: IF x is $A^{\sim}$ AND y is $B^{\sim}$ THEN z is $C^{\sim}$, where $A^{\sim}$, $B^{\sim}$, and $C^{\sim}$ are type-2 fuzzy membership functions (see the extended glossary, and also the section in this chapter on type-2 ECOS).

The integration of ANN and fuzzy systems into one system attracted many researchers. The integration of fuzzy rules into a single neuron model and then into larger neural network structures, tightly coupling learning and fuzzy reasoning rules into connectionists structures, was initiated by Professor Takeshi Yamakawa and other Japanese scientists [31]. Many models of fuzzy neural networks are developed based on these principles [21, 32, 33]. Adaptive neural networks for incremental learning and rule extraction: The neuro-fuzzy systems (no more the “black box curse”). As a general case, input and/or output variables can be non-fuzzy (crisp) or fuzzy. Example of fuzzy Gaussian membership functions is shown in Fig. 12:

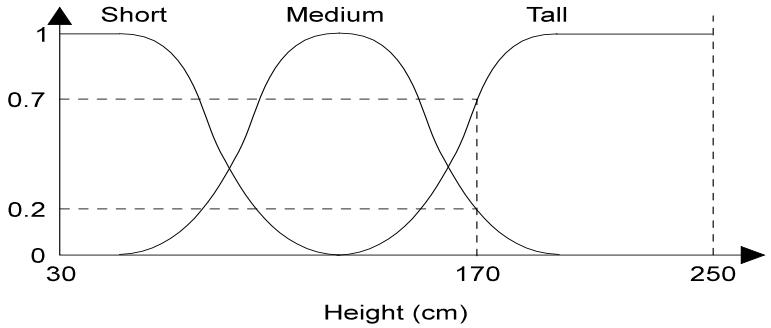


Figure. 12. Example of fuzzy Gaussian membership functions that represent a variable Height

Hybrid connections systems can incorporate fuzzy rules. They can also be used to extract fuzzy rules from already trained ANN called fuzzy neural networks as it is the case with the ECOS discussed in the next section. An example of a fuzzy rule is given below:

IF Input 1 is High and Input 2 is Low THEN Output is Very High

A typical example of hybrid ANN are ECOS, presented in the next section.

2.3. Evolving connectionist systems (ECOS)

2.3.1. Principles of ECOS

In the evolving connectionist systems (ECOS), introduced by the author, instead of training a fixed ANN through changing its connection weights, the connectionist structure and its functionality are evolving from incoming data, often in an on-line, one-pass learning mode [10, 34-36].

ECOS are modular connectionist based systems that evolve their structure and functionality in a continuous, self-organized, on-line, adaptive, interactive way from incoming information [10]. They can process both data and knowledge in a supervised and/or unsupervised way. ECOS learn local models from data through clustering of the data and associating a local output function for each cluster represented in a connectionist structure. They can learn incrementally single data items or chunks of data and also incrementally change their input features [35, 37]. Elements of ECOS have been proposed as part of the classical neural network models, such as Self-Organizing Maps, Radial Basis Functions, Fuzzy ARTMap, growing neural gas, neuro-fuzzy systems, Resource Allocation Network (for a review see [21]). Other ECOS models, along with their applications, have been reported in [38] and [39].

The principle of ECOS is based on *local learning* – neurons are allocated as centers of data clusters and the system creates local models in these clusters. Methods of fuzzy clustering, as a means to create local knowledge-based systems, were developed by Bezdek, Yager, Filev and others [40, 41].

To summarize, the following are the main principles of ECOS as stated in [10]:

(1) Fast learning from large amount of data, e.g. using “one-pass” training, starting with little prior knowledge;

(2) Adaptation in real-time and in an on-line mode where new data is accommodated as it comes based on local learning;

(3) “Open”, evolving structure, where new input variables (relevant to the task), new outputs (e.g. classes), new connections and neurons are added/evolved “on the fly”;

(4) Both data learning and knowledge representation is facilitated in a comprehensive and flexible way, e.g., supervised learning, unsupervised learning, evolving clustering, “sleep” learning, forgetting/pruning, fuzzy rule insertion and extraction;

(5) Active interaction with other ECOSs and with the environment in a multi-modal fashion;

(6) Representing both space and time in their different scales, e.g., clusters of data, short- and long-term memory, age of data, forgetting, etc.;

(7) System's self-evaluation in terms of behavior, global error and success, and related knowledge representation.

2.3.2. Evolving self-organising maps

Several methods, such as: Dynamic Topology Representing Networks [42] and Evolving Self-organizing Maps (ESOM) [43] further develop the principles of SOM. These methods allow the prototype nodes to evolve quickly in the original data space X, and at the same time acquire and keep a topology representation. The neighbourhood of the evolved nodes (neurons) is not pre-defined as it is in SOM. It is defined in an on-line mode according to the current distances between the nodes. These methods are free of the rigid topological constraints in SOM. They do not require searching for neighbourhood ranking as in the neural gas algorithm, thus improving the speed of learning.

Here, the ESOM method is explained in more detail.

Given an input vector **x**, the activation on the *i*-th node in ESOM is defined as:

$$a_i = e^{-\|x-w_i\|^2/\varepsilon^2} \tag{10}$$

Where ε is a radial. Here a_i can be regarded as a matching score for the i-th prototype vector w_i onto the current input vector x. The closer they are, the bigger the matching score is**.**

The following on-line stochastic approximation of the error minimization function is used:

$$E_{app} = \sum_{i=1,n} a_i \|x - w_i\|^2 \tag{11}$$

where n is the current number of nodes in ESOM upon arrival of the input vector **x**.

To minimize the criterion function above, weight vectors are updated by applying a gradient descent algorithm. From Eqn. (11) it follows:

$$\frac{\partial E_{app}}{\partial w_i} = a_i(w_i - x) + \|x - w_i\|^2 \partial a_i / \partial w_i \tag{12}$$

For the sake of simplicity, we assume that the change of the activation will be rather small each time when the weight vector is updated, so that a_i can be treated as a constant. This leads to the following simplified weight- updating rule:

$$\Delta w_i = \gamma a_i (x - w_i), \quad for\ i = 1,2,\dots,n \tag{13}$$

Here γ is a learning rate held as a small constant.

The likelihood of assigning the current input vector **x** onto the *i*-th prototype $\mathbf{w}_i$ is defined as:

$$Pi\ (x, w_i) = a_i \,/\, \Sigma_{k=1,2,\dots,n}\ (a_k) \tag{14}$$

Evolving the Feature Map

During on-line learning, the number of prototypes in the feature map is usually unknown. For a given data set the number of prototypes may be optimum at a certain time but later it may become inappropriate as when new samples are arriving the statistical characteristics of data may change. Hence it is highly desirable for the feature map to be dynamically adaptive to the incoming data.

The approach here is to start with a null map, and gradually allocate new prototype nodes when new data samples cannot be matched well onto existing prototypes. During learning, when old prototype nodes become inactive for a long time, they can be removed from the dynamic prototype map.

If for a new data vector **x** none of the prototype nodes are within a distance threshold, then a new node $\mathbf{w}_{new}$ is inserted representing exactly the poorly matched input vector $\mathbf{w}_{new}=\mathbf{x}$, resulting in a maximum activation of this node for **x**.

The ESOM learning algorithm is given in Box 3.

Box.3. The ESOM evolving self-organised maps learning algorithm:

Step 1: Input a new data vector **x**.

Step 2: Find a set S of prototypes that are closer to **x** than a pre-defined threshold.

Step 3: If S is null, go to step 4 (insertion), otherwise – calculate the activations a_i of all nodes from S and go to step 5 (updating).

Step 4 (insertion): Create a new node $\mathbf{w}_i$ for **x** and make a connection between this node and its two closest nodes (nearest neighbours) that will form a set S.

Step 5 (updating): Modify all prototypes in S and re-calculate the connections s(i,j) between the winning node i (or the newly created one) and all the nodes j in the set S:

$s(i,j) = a_i a_j / \max \{a_i, a_j\}$

Step 6: After a certain number of input data are presented to the system, prune the weakest connections. If isolated nodes appear, prune them as well.

Step 7: go to step 1

Visualising the Feature Map:

Sammon projection or other dynamic visualisation techniques [37] can be used to visualize the evolving nodes of an ESOM at each time of incremental, evolving learning.

2.3.3. Evolving MLP

A simple evolving MLP method is called here eMLP and presented in Figure 13 as a simplified graphical representation. An eMLP consists of three layers of neurons, the input layer, with linear transfer functions, an evolving layer, and an output layer with a simple saturated linear activation

function. It is a simplified version of the evolving fuzzy neural network (EFuNN), presented later in this chapter [44].

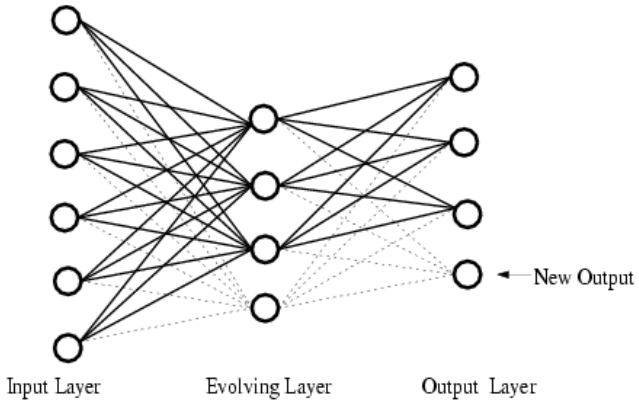


Figure 13. A schematic diagram of a simple eMLP

The evolving layer is the layer that will grow and adapt itself to the incoming data, and is the layer with which the learning algorithm is most concerned. The meaning of the incoming connections, activation and forward propagation algorithms of the evolving layer all differ from those of classical connectionist systems.

If a linear activation function is used, the activation A of an evolving layer node n is determined by equation (15)

$$A_n = 1 - D_n \qquad (15)$$

where: A_n is the activation of the node n and ; D_n is the normalised distance between the input vector and the incoming weight vector for that node.

Other activation functions, such as a radial basis function could be used. Thus, examples which exactly match the exemplar stored within the neurons incoming weights will result in an activation of 1 while examples that are entirely outside of the exemplars region of input space will result in an activation of near 0.

The preferred form learning algorithm is based on accommodating, within the evolving layer, new training examples by either modifying the connection weights of the evolving layer nodes, or by adding a new node. The algorithm employed is presented in Box 4.

Box 4. eMLP learning algorithm

1. Propagate the input vector I through the network

IF the maximum activation $a_{\max}$ is less than a coefficient called sensitivity threshold S_{thr} :

2. Add a node, ELSE

3. Evaluate the error between the calculated output vector O_c and the desired output vector O_d

4. IF the error is greater than an error threshold E_{thr} OR the desired output node is not the most highly activated,

5. Add a node, ELSE

6. Update the connections to the winning node in the evolving layer

7. Repeat the above procedure for each training vector

When a node is added, its incoming connection weight vector is set to the input vector I , and its outgoing weight vector is set to the desired output vector O_d .

The incoming weights to the winning node *j* are modified according to Equation (16), while the outgoing weights from node j are modified according Equation (17)

$$W_{i,j}(t+1) = W_{i,j}(t) + \eta_1(I_i \times W_{i,j}(t)) \quad (16)$$

where:

$W_{i,j}(t)$ is the connection weight from input i to *j* at time t

$W_{i,j}(t+1)$ is the connection weight from input i to *j* at time $t+1$

η_1 is the learning rate one parameter

I_i is the i th component of the input vector I

$$W_{j,p}(t+1) = W_{j,p}(t) + \eta_2(A_j \times E_p) \quad (17)$$

where:

$W_{j,p}(t)$ is the connection weight from *j* to output p at time t

$W_{i,p}(t+1)$ is the connection weight from j to p at time $t+1$

η_2 is the learning rate two parameter

A_j is the activation of j

$$E_p = O_{d(p)} - O_{c(p)} \quad (18)$$

where: E_p is the error at p ; $O_{d(p)}$ is the desired output at p ; $O_{c(p)}$ is the calculated output at p .

The distance measure D_n in equation (15) above is preferably calculated as the normalised Hamming distance, as shown in Equation (19):

$$D_n = \frac{\sum_i^I |E_i - W_i|}{\sum_i^I |E_i + W_i|} \tag{19}$$

where: I is the number of input nodes in the eMLP, E is the input vector, W is the input to evolving layer weight matrix.

Aggregation of nodes in the evolving layer can be employed to control the size of the evolving layer during the learning process. The principle of aggregation is to merge those nodes which are spatially close to each other. Aggregation can be applied for every (or after every n) training examples. It will generally improve the generalisation capability of EMLP. The aggregation algorithm is as follows:

FOR each rule node $r_{j,} j = 1:n,$ where n is the number of nodes in the evolving layer and $W1$ is the connection weights matrix between the input and evolving layers and $W2$ is the connection weights matrix between the evolving and output layers.

- find a subset R of nodes in evolving layer for which the normalised Euclidean distances $D(W1_{rj}, W1_{ra})$ and $D(W2_{rj}, W2_{ra})$ $r_j, r_a \in R$ are below the thresholds W_{thr}.
- merge all the nodes from the subset R into a new node r_{new} and update $W1_{r_{new}}$ and $W2_{r_{new}}$ using the following formulae:

$$W1_{r_{new}} = \frac{\sum r_a \in R(W1_{r_a})}{m} \tag{20}$$

$$W2_{r_{new}} = \frac{\sum r_a \in R(W2_{r_a})}{m} \tag{21}$$

where m denotes the number of nodes in the subset R.

- delete the nodes $r_a \in R$

Node aggregation is an important regularization that is not present in ZISC. It is highly desirable in some application areas, such as speech or image recognition systems. In speech recognition, vocabulary of recognition systems needs to be customised to meet individual needs. This can be achieved by adding words to the existing recognition system or removing words from existing vocabulary.

eMLP are also suitable for on-line output space expansion because it uses local learning which tunes only the connection weights of the local node, so all the knowledge that has been captured in the nodes in the evolving layer will be local and only covering a "patch" of the input-output space. Thus, adding new class outputs or new input variables does not require re-training of the whole system on both the new and old data as it is required for traditional neural networks.

The task is to introduce an algorithm for on-line expansion and reduction of the output space in eMLP. As described above the eMLP is a three layer network with two layers of connections. Each node in the output layer represents a particular class in the problem domain. This local representation of nodes in the evolving layer enables eMLP to accommodate new classes or remove an already existing class from its output space.

In order to add a new node to the output layer, the structure of the existing eMLP first needs to be modified to encompass the new output node. This modification affects only the output layer and the connections between the output layer and the evolving layer. The graphical representation of this process is shown in Figure 3.8. The connection weights between the new output in the output layer and the evolving layer are initialised to zero. In this manner the new output node is set by default to classify all previously seen classes as negative. Once the internal structure of the eMLP is modified to accommodate the new output class, the eMLP is further trained on the new data. As a result of the training process new nodes are created in the evolving layer to represent the new class.

The process of adding new output nodes to eMLP is carried out in a supervised manner. Thus, for a given input vector, a new output node will be added only if it is indicated that the given input vector is a new class. The output expansion algorithm is as follows:

FOR every new output class:

1. Insert a new node into the output layer;
2. FOR every node in the evolving layer $r_i, i = 1:n$, where n is the number of nodes in the evolving layer, modify $W2$ the outgoing connection weights from the evolving to output layer by expanding $W2_{i,j}$ with set of zeros to reflect the zero output.

This is equivalent to allocating a part of the problem space for data that belong to new classes, without specifying where this part is in the problem space.

It is also possible to remove a class from an eMLP. It only affects the output and evolving layer of eMLP architecture:

FOR every output class o to be removed,

1. Find set of nodes S in the evolving layer which are committed to that output o
2. Modify $W1$ the incoming connection from input layer to evolving layer by deleting $S_i, i = 1:n$, where n is the number of nodes in the set S committed to output o
3. Modify $W2$ the outgoing connection weights from the evolving to output layer by deleting output node o

The above algorithm is equivalent to dis-allocating a part of the problem space which had been allocated for the removed output class. In this manner, there will be no space allocated for the deleted output class in the problem space. In other words the network is unlearning a particular output class.

The eMLP is further studied and applied in [45, 46].

2.4. Evolving fuzzy neural networks. EFuNN

Here the concept of ECOS is illustrated on two implementations: the evolving fuzzy neural network (EFuNN) [34] and the dynamic evolving neuro-fuzzy inference system (DENFIS) [36]. In ECOS, clusters of data are created based on similarity between data samples either in the input space (this is the case in some of the ECOS models, e.g., DENFIS), or in both the input and output space (this is the case, e.g., in the EFuNN models). Samples (examples) that have a distance to an existing node (cluster center, rule node) less than a certain threshold are allocated to the same cluster. Samples that do not fit into existing clusters form new clusters. Cluster centers are continuously adjusted according to new data samples, and new clusters are created incrementally. ECOS learn from data and automatically create or update a local fuzzy model/function, e.g.:

IF <data is in a fuzzy cluster C_i > THEN <the model is F_i>

where F_i can be a fuzzy value, a logistic or linear regression function or ANN model [36, 37].

Generally speaking, fuzzy neural networks are connectionist structures that can be interpreted in terms of fuzzy rules [21, 31, 32 and 47]. Fuzzy neural networks are NN, with all the NN characteristics of training, recall, adaptation, etc. while neuro- fuzzy inference systems (chapter 5) are fuzzy rule based systems and their associated fuzzy inference mechanism that are implemented as neural networks for the purpose of learning and rule optimisation. The evolving fuzzy neural network (EFuNN) presented here is of the former type, while DENFIS systems are of the latter type. Some authors do not separate the two types that makes the transition from one to the other type more flexible and also broadens the interpretation and the application of each of these systems.

EFuNNs have a five-layer structure (fig. 14). Here nodes and connections are created/connected as data examples are presented. An optional short-term memory layer can be used through a feedback connection from the rule (also called, case) node layer. The layer of feedback connections could be used if temporal relationships of input data are to be memorized structurally.

.

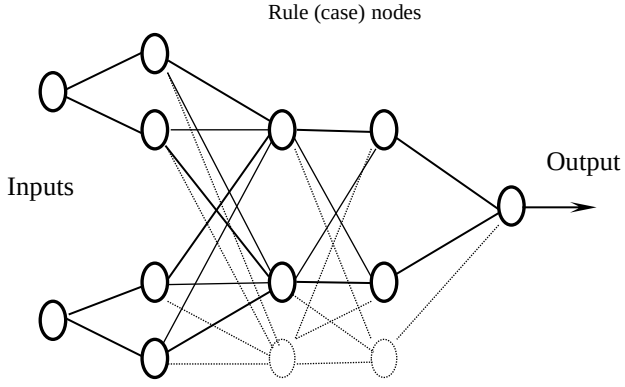


Figure 14. A simplified architecture of EFuNN

The input layer represents input variables. The second layer of nodes (fuzzy input neurons, or fuzzy inputs) represents fuzzy quantisation of each input variable space (similar to the factorizable RBF networks). For example, two fuzzy input neurons can be used to represent "small" and "large" fuzzy values. Different membership functions (MF) can be attached to these neurons.

The number and the type of MF can be dynamically modified. The task of the fuzzy input nodes is to transfer the input values into membership degrees to which they belong to the corresponding MF. The layers that represent fuzzy MF are optional, as a non-fuzzy version of EFuNN can also be evolved with only three layers of neurons and two layers of connections as it is used in chapter 6.

The third layer contains rule (case) nodes that evolve through supervised and/or unsupervised learning. The rule nodes represent prototypes (exemplars, clusters) of input-output data associations that can be graphically represented as associations of hyper-spheres from the fuzzy input and the fuzzy output spaces. Each rule node r is defined by two vectors of connection weights – W1(r) and W2(r), the latter being adjusted through supervised learning based on the output error, and the former being adjusted through unsupervised learning based on similarity measure within a local area of the problem space. A linear activation function, or a Gaussian function, is used for the neurons of this layer.

The fourth layer of neurons represents fuzzy quantization of the output variables, similar to the input fuzzy neuron representation. Here, a weighted sum input function and a saturated linear activation function is used for the neurons to calculate the membership degrees to which the output vector associated with the presented input vector belongs to each of the output MFs. The fifth layer represents the values of the output variables. Here a linear activation function is used to calculate the defuzzified values for the output variables.

A partial case of EFuNN would be a three layer network without the fuzzy input and the fuzzy output layers (e.g. eMLP, or an evolving simple RBF network). In this case a slightly modified versions of the algorithms described below are applied, mainly in terms of measuring Euclidean distance and using Gaussian activation functions.

The evolving learning in EFuNNs is based on either of the following two assumptions:

(1) No rule nodes exist prior to learning and all of them are created (generated) during the evolving process; or

(2) There is an initial set of rule nodes that are not connected to the input and output nodes and become connected through the learning (evolving) process. The latter case is more biologically plausible as most of the neurons in the human brain exist before birth, and become connected through learning, but still there are areas of the brain where new neurons are created during learning if “surprisingly” different stimuli from previously seen are presented. (See chapter 1 for biological inspirations for ECOS).

The EFuNN evolving algorithm presented next does not make a difference between these two cases.

Each rule node, e.g. r_j, represents an association between a hyper-sphere from the fuzzy input space and a hyper-sphere from the fuzzy output space (see Fig. 15), the W1(r_j) connection weights representing the co-ordinates of the centre of the sphere in the fuzzy input space, and the W2 (r_j) – the co-ordinates in the fuzzy output space. The radius of the input hyper-sphere of a rule node r_j is defined as R_j=1- S_j, where S_j is the sensitivity threshold parameter defining the minimum activation of the rule node r_j to a new input vector x from a new example (**x,y**) in order the example to be considered for association with this rule node.

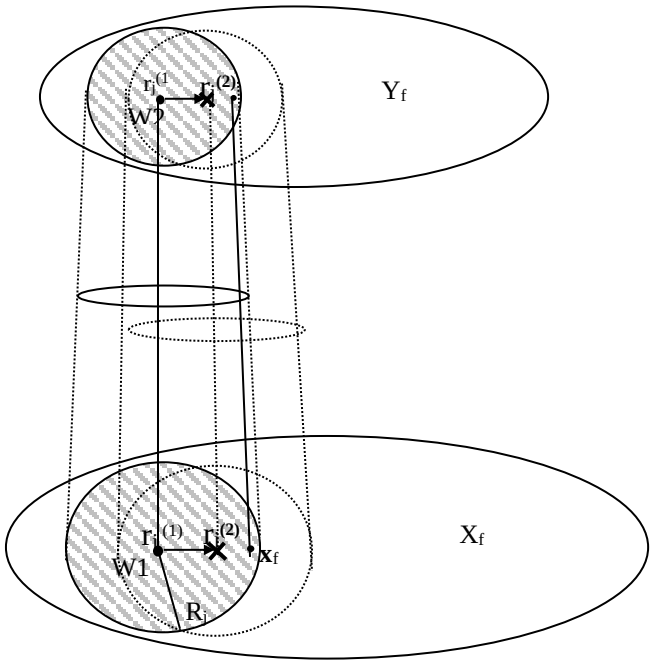


Figure 15. EFuNN maps clusters from the input space to clusters in the output problem space, where the radii of the clusters can change during learning

The pair of fuzzy input-output data vectors (**x_f,y_f**) will be allocated to the rule node r_j if **x_f** falls into the r_j input receptive field (hyper-sphere), and **y_f** falls in the r_j output reactive field hyper-sphere. This is ensured through two conditions, that a local normalised fuzzy difference between x_f and W1(r_j) is smaller than the radius R_j, and the normalised output error Err= ||y – y'|| / Nout is smaller than an error threshold E. Nout is the number of the outputs and **y**' is the produced by EFuNN output. The error parameter E sets the error tolerance of the system.

Definition. A local normalised fuzzy distance between two fuzzy membership vectors $\mathbf{d}_{1f}$ and $\mathbf{d}_{2f}$ that represent the membership degrees to which two real vector data $\mathbf{d_1}$ and $\mathbf{d_2}$ belong to pre-defined MFs, is calculated as:

$$D(d_{1f},d_{2f}) = ||d_{1f} - d_{2f}|| / ||d_{1f} + d_{2f}||, \quad (22)$$

where: ||x - y|| denotes the sum of all the absolute values of a vector that is obtained after vector subtraction (or summation in case of ||x + y||) of two vectors x and y; " / " denotes division. For example, if d_{1f}=(0,0,1,0,0,0) and d_{2f}=(0,1,0,0,0,0), than D(d_1,d_2) = (1+1)/2=1 which is the maximum value for the local normalised fuzzy difference. In EFuNNs the local normalised fuzzy distance is used to measure the distance between a new input data vector and a rule node in the local vicinity of the rule node.

In RBF networks Gaussian radial basis functions are allocated to the nodes and used as activation functions to calculate the distance between the node and the input vectors.

Through the process of associating (learning) of new data points to a rule node r_j, the centres of this node hyper-spheres adjust in the fuzzy input space depending on the distance between the new input vector and the rule node through a learning rate l_j, and in the fuzzy output space depending on the output error through the Widrow-Hoff lest mean square (LMS) delta algorithm [48]. This adjustment can be represented mathematically by the change in the connection weights of the rule node r_j from W1($r_j^{(t)}$) and W2($r_j^{(t)}$) to W1($r_j^{(t+1)}$) and W2($r_j^{(t+1)}$) respectively, employing the following vector operations:

$$W1(r_j^{(t+1)}) = W1(r_j^{(t)}) + l_j . (\mathbf{x}_f - W1(r_j^{(t)})) \quad (23)$$
$$W2(r_j^{(t+1)}) = W2(r_j^{(t)}) + l_j . (\mathbf{y}_f - A2) . A1(r_j^{(t)})$$

where: A2=f_2(W2.A1) is the activation vector of the fuzzy output neurons in the EFuNN structure when x is presented; A1($r_j^{(t)}$) =f_2 (D (W1 ($r_j^{(t)}$) , x_f)) is the activation of the rule node $r_j^{(t)}$; a simple linear function can be used for f_1 and f_2, e.g. A1($r_j^{(t)}$) = 1- D (W1 ($r_j^{(t)}$) , $\mathbf{x}_f$)); l_j is the current learning rate of the rule node r_j calculated for example as l_j = 1/ Nex(r_j), where Nex(r_j) is the number of examples currently associated with rule node r_j.

The statistical rationale behind this is that the more examples are currently associated with a rule node, the less it will "move" when a new example has to be accommodated by this rule node, i.e. the change in the rule node position is proportional to the number of already associated example to the new, single example.

When a new example is associated with a rule node r_j not only its location in the input space changes, but also its receptive field expressed as its radius Rj, and its sensitivity threshold Sj:

$$Rj^{(t+1)} = Rj^{(t)} + D(W1(r_j^{(t+1)}), W1(r_j^{(t)})), \text{ respectively} \quad (24)$$
$$Sj^{(t+1)} = Sj^{(t)} - D(W1(r_j^{(t+1)}), W1(r_j^{(t)})) \quad (25)$$

The learning process in the fuzzy input space is illustrated in Fig.16 on four data points d_1, d_2, d_3 and d_4. Fig. 16 shows how the centre $r_j^{(1)}$ of the rule node r_j adjusts (after learning each new data point) to

its new positions $r_j^{(2)}$, $r_j^{(3)}$, $r_j^{(4)}$ when one pass learning is applied. Fig.16 shows how the rule node position would move to new positions $r_j^{(2(2)}$, $r_j^{(3(2)}$, and $r_j^{(4(2)}$, if another pass of learning was applied. If the two learning rate l_1 and l_2 have zero values, once established, the centres of the rule nodes will not move.

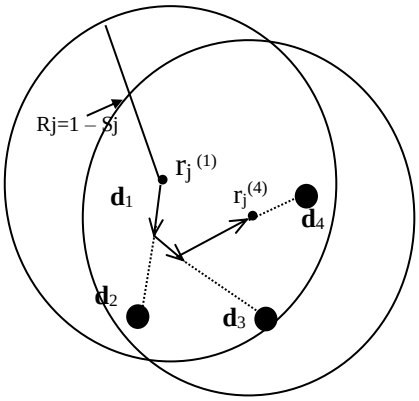


Figure 16. Adjusting the rule nodes during learning of 4 data points in EFuNN ([37])

The weight adjustment formulas (Equ. 23) define the standard EFuNN that has the first part updated in an unsupervised mode, and the second part – in a supervised mode similar to the RBF networks. But here the formulas are applied once for each example (**x,y**) in an incrementally adaptive mode, that is similar to the RAN model [49] and its modifications. The standard supervised/unsupervised learning EFuNN is denoted as EFuNN-s/u. In two other modifications of EFuNN, namely double pass learning EFuNN (EFuNN-dp), and gradient descent learning EFuNN (EFuNN-gd), slightly different update functions are used as explained in the next sub-section.

The learned temporal associations can be used to support the activation of rule nodes based on temporal pattern similarity. Here, temporal dependencies are learned through establishing structural links. These dependencies can be further investigated and enhanced through synaptic analysis (at the synaptic memory level) rather than through neuronal activation analysis (at the behavioural level). The ratio spatial similarity/ temporal-correlation can be balanced for different applications through two parameters Ss and Tc such that the activation of a rule node r for a new data example $\mathbf{d}_{new}$ is defined through the following vector operations:

$$A1(r) = \left| 1 - Ss.\, D(W1(r), \mathbf{d}_{newf}) + Tc.W3(r_{max}^{(t-1)}, r) \right|_{[0,1]} \quad (26)$$

where: $|.|_{[0,1]}$ is a bounded operation in the interval [0,1]; $D(W1(r),d_{newf})$ is the normalised local fuzzy distance value and $r_{max}^{(t-1)}$ is the winning neuron at the previous time moment. Here temporal connections can be given a higher importance in order to tolerate a higher distance. If T_c=0, then
The supervised learning in EFuNN is based on the above explained principles, so when a new data example **d**=(**x,y**) is presented, the EFuNN either creates a new rule node r_n to memorize the two input and output fuzzy vectors $W1(r_n)$= $\mathbf{x_f}$ and $W2(r_n)$= $\mathbf{y_f}$, or adjusts an existing rule node r_j.

After a certain time (when certain number of examples have been presented) some neurons and connections may be pruned or aggregated.

The supervised learning algorithms above allow for an EFuNN system to always evolve and learn when a new input-output pair of data becomes available. This is an *active learning mode*.

EFuNN Sleep Learning Rules

In another mode, *passive or sleep learning*, learning is performed when there is no input pattern presented. This may be necessary to apply after an initial learning has been performed. In this case existing connections, that store previously fed input patterns, are used as "echo" to reiterate the learning process. This type of learning may be applied in case of a short initial presentation of the data, when only small portion of data is learned in one-pass, incrementally adaptive mode, and then the training is refined through the sleep learning method when the system consolidates what it has learned before.

Sleep learning in EFuNN and in some other connectionist models is illustrated on several examples in [50].

One- pass versus multiple-passes learning

The best way to apply the above learning algorithms is to draw randomly examples from the problem space, propagate them through the EFuNN and tune the connection weights and the rule nodes, change and optimise the parameter values, etc., until the error becomes a desirably small one. In a fast learning mode, each example is presented only once to the system. If it is possible to present examples twice, or more times, the error may become smaller, but that depends on the parameter values of the EFuNN and on the statistical characteristics of the data.

The evolved EFuNN can perform *inference* when recalled on new input data. The EFuNN inference method consists of calculating the output activation value when a new input vector is applied. This is part of the EFuNN supervised learning method when only an input vector **x** is propagated through the EFuNN. If the new input vector falls in the receptive field of the winning rule node (the closest rule node to the input vector) *one-of-n* mode of inference is used that is based on the winning rule node activation (one rule inference). If the new input vector does not fall in the receptive filed of the closest to it rule node, than *m-of-n* mode is used, where *m* rule nodes (rules) are used in the EFuNN inference process, with an usual value of *m* being 3.

Different pruning rules can be applied for a successful *pruning* of unnecessary nodes and connections. One of them is given below:

> IF (Age(r_j) > OLD) AND (the total activation TA(r_j) is less than a pruning parameter Pr times Age (r_j)) THEN prune rule node r_j,

where Age(r_j) is calculated as the number of examples that have been presented to the EFuNN after r_j had been fist created; OLD is a pre-defined age limit; Pr is a pruning parameter in the range [0,1], and the total activation TA(r_j) is calculated as the number of examples for which r_j has been the correct winning node (or among the m winning nodes in the m-of-n mode of operation).

The above pruning rule requires that the fuzzy concepts of OLD, HIGH, etc. are defined in advance. As a partial case, a fixed value can be used, e.g. a node is OLD if it has existed during the evolving of a FuNN from more than p examples. The pruning rule and the way the values for the pruning parameters are defined, depend on the application task.

One of the learning algorithms for EFuNN is shown in Box 5 [37].

Box 5. EFuNN-s/u Learning Algorithm

---

1. Set initial values for the system parameters: number of membership functions; initial sensitivity threshold (default S=0.9); error threshold E; aggregation parameter Nagg – a number of consecutive examples after which an aggregation is performed (to be explained in a later section); pruning parameters OLD an Pr; a value for m (in m-of-n mode); thresholds T_1 and T_2 for rule extraction.

2. Set the first rule node to memorize the first example (**x,y**):

$W1(r_0)=\mathbf{x}_f$, and $W2(r_0)=\mathbf{y}_f$; (27)

3. Loop over presentations of input-output pairs (**x,y**)

{Evaluate the local normalized fuzzy distance D between $\mathbf{x}_f$ and the existing rule node connections W1 (formulae (22)).

Calculate the activation A1 of the rule node layer. Find the closest rule node r_k (or the closest m rule nodes in case of m-of-n mode) to the fuzzy input vector $\mathbf{x}_f$.

if $A1(r_k) < S_k$ (sensitivity threshold for the node r_k), create a new rule node for ($\mathbf{x}_f,\mathbf{y}_f$)

else

Find the activation of the fuzzy output layer A2=W2.A1 and the output error Err= || **y**- **y**'|| / Nout.

i*f* Err > E

create a new rule node to eMLPommodate the current example ($\mathbf{x}_f,\mathbf{y}_f$)

else

Update W1 (r_k) and W2(r_k) eMLPording to (23) (in case of *m-of-n* EFuNN update all the *m* rule nodes with the highest A1 activation).

Apply *aggregation* procedure of rule nodes after each group of N_{agg} examples are presented.

Update the values for the rule node r_k parameters S_k, R_k, Age(r_k), TA (r_k).

Prune rule nodes if necessary, as defined by pruning parameters.

Extract rules from the rule nodes (as explained in a later sub-section)

} End of the main loop.

At any time (phase) of the evolving (learning) process of an EFuNN fuzzy or exact rules can be *inserted* and extracted. Insertion of fuzzy rules is achieved through setting a new rule node r_j for each new rule, such that the connection weights $W1(r_j)$ and $W2(r_j)$ of the rule node represent this rule. For example, the fuzzy rule *(IF x_1 is Small and x_2 is Small THEN y is Small*) can be inserted into an EFuNN structure by setting the connections of a new rule node to the fuzzy condition nodes x_1- Small and x_2- Small and to the fuzzy output node y-Small to a value of 1 each. The rest of the connections are set to a value of zero. Similarly, an exact rule can be inserted into an EFuNN structure, e.g. *IF x_1 is 3.4 and x_2 is 6.7 THEN y is 9.5.* Here the membership degrees to which the input values x_1 = 3.4 and x_2=6.7, and the output value *y=9.5* belong to the corresponding fuzzy values are calculated and attached to the corresponding connection weights.

Each rule node r_j can be expressed as a fuzzy rule, for example:

Rule r_j: IF x_1 is Small 0.85 and x_1 is Medium 0.15 and x_2 is Small 0.7 and x_2 is Medium 0.3 (Radius of the receptive field Rj=0.1, maxRadiusj=0.75) THEN y is Small 0.2 and y is Large 0.8 (20 out of 175 examples associated with this rule),

where the numbers attached to the fuzzy labels denote the degree to which the centres of the input and the output hyper-spheres belong to the respective MF. The degrees associated to the condition elements are the connection weights from the matrix W1. Only values that are greater than a threshold T1 are left in the rules as the most important ones. The degrees associated with the conclusion part are the connection weights from W2 that are greater than a threshold of T2. An example of rules extracted from a bench-mark dynamic time series data is given in sub-section 3.5. The two thresholds T1 and T2 are used to disregard the connections from W1 and W2 that represent small and insignificant membership degrees (e.g., less than 0.1).

Rule Node Aggregation in EFuNNs

Another knowledge-based technique applied to EFuNNs is *rule node aggregation*.

For example, for the aggregation of three rule nodes r_1, r_2, and r_3 the following two aggregation rules can be used to calculate the new aggregated rule node r_{agg} W1 connections (the same formulas are used to calculate the W2 connections):

(a) as a geometrical centre of the three nodes:

$$W1(r_{agg})=(W1(r_1)+W1(r_2)+W1(r_3))/3 \quad (28)$$

(b) as a weighted statistical centre:

$$W2(r_{agg})=(W2(r_1).Nex(r_1)+W2(r_2)Nex(r_2)+W2(r_3).Nex(r_3))/Nsum \quad (29)$$

where:

$$Nex(r_{agg})= Nsum = Nex(r_1)+Nex(r_2)+Nex(r_3);$$

r_j is the rule node from the three nodes that ha a maximum distance from the new node r_{agg} and Rj is its radius of the receptive field.

The three rule nodes will aggregate only if the radius of the aggregated node receptive field is less than a pre-defined maximum radius Rmax:

$Rr_{agg} = D(W1(r_{agg}), W1(r_j)) + Rj <= Rmax$

In order for a given node r_j to aggregate with other nodes, two subsets of nodes are formed – the subset of nodes r_k that if activated to a degree of 1 will produce an output value $\mathbf{y'}(r_k)$ that is different from $\mathbf{y'}(r_j)$ in less than the error threshold E, and the subset of nodes that cause output values different from $\mathbf{y'}(r_k)$ in more than E. The W2 connections define these subsets. Than all the rule nodes from the first subset that are closer to r_j in the input space than the closest to r_j node from the second subset in terms of W1 distance, get aggregated if the radius of the new node r_{agg} is less than the pre-defined limit Rmax for a receptive field

Instead of aggregating all rule nodes that are close to a rule node r_j than the closest node from the other class, it is possible to keep the closest to the other class node from the aggregation pool out of the aggregation procedure - as a separate node – a "guard", thus preventing a possible miss-classification of new data on the bordering area between the two classes.

Through node creation and their consecutive aggregation, an EFuNN system can adjust over time to changes in the data stream and at the same time – preserve its generalisation capabilities.

Through analysis of the weights W3 of an evolved EFuNN, *temporal correlation* between time consecutive exemplars can be expressed in terms of rules and conditional probabilities, e.g.:

$$\text{IF } r_1^{(t-1)} \text{ THEN } r_2^{(t)} (0.3) \qquad (30)$$

The meaning of the above rule is that some examples that belong to the rule (prototype) r_2 follow in time examples from the rule prototype r_1 with a relative conditional probability of 0.3.

2.5. Dynamic Evolving Neuro-Fuzzy Inference Systems - DENFIS

The dynamic evolving neuro-fuzzy system, DENFIS, in its two modifications - for on-line-, and for off-line learning, use Takagi-Sugeno type of fuzzy inference method [37]. The inference used in DENFIS is performed on *m* fuzzy rules as described below:

$$\begin{cases} \text{if } x_1 \text{ is } R_{11} \text{ and } x_2 \text{ is } R_{12} \text{ and } \ldots \text{ and } x_q \text{ is } R_{1q}, \text{ then } y \text{ is } f_1(x_1, x_2, \ldots, x_q) \\ \text{if } x_1 \text{ is } R_{21} \text{ and } x_2 \text{ is } R_{22} \text{ and } \ldots \text{ and } x_q \text{ is } R_{2q}, \text{ then } y \text{ is } f_2(x_1, x_2, \ldots, x_q) \\ \\ \text{if } x_1 \text{ is } R_{m1} \text{ and } x_2 \text{ is } R_{m2} \text{ and } \ldots \text{ and } x_q \text{ is } R_{mq}, \text{ then } y \text{ is } f_m(x_1, x_2, \ldots, x_q) \end{cases} \qquad (31)$$

where "x_j is R_{ij}", $i = 1, 2, \ldots m$; $j = 1, 2, \ldots q$, are $m \times q$ fuzzy propositions that form m antecedents for m fuzzy rules respectively; x_j, $j = 1, 2, \ldots, q$, are antecedent variables defined over universes of discourse X_j, $j = 1, 2, \ldots, q$, and R_{ij}, $i = 1, 2, \ldots m$; $j = 1, 2, \ldots, q$ are fuzzy sets defined by their fuzzy membership functions μ_{Rij}: $X_j \rightarrow [0, 1]$, $i = 1, 2, \ldots m$; $j = 1, 2, \ldots, q$. In the consequent parts of the fuzzy rules, y is the consequent variable, and crisp functions f_i, $i = 1, 2, \ldots m$, are employed.

In the DENFIS on-line model, the first-order Takagi-Sugeno type fuzzy rules are employed and the linear functions in the consequence parts are created and updated through learning from data by using the linear least-square estimator (LSE)).

If the consequent functions are crisp constants, i.e. $f_i(x_1, x_2, \ldots, x_q) = C_i$, $i = 1, 2, \ldots m$, we call such system a zero-order Takagi-Sugeno type fuzzy inference system. The system is called a first-order Takagi-Sugeno type fuzzy inference system if $f_i(x_1, x_2, \ldots, x_q)$, $i = 1, 2, \ldots m$, are linear functions. If these functions are non-linear functions, it is called high-order Takagi-Sugeno fuzzy inference system. For an input vector $\boldsymbol{x}^0 = [x_1^0\ x_2^0\ \ldots\ x_q^0]$, the result of inference, y^0 (the output of the system) is the weighted average of each rule's output indicated as follows:

$$y^0 = \frac{\sum_{i=1,m} \omega_i f_i(x_1^0, x_2^0, \ldots, x_q^0)}{\sum_{i=1,m} \omega_i} \qquad (32)$$

$$\text{where: } \omega_i = \prod_{j=1}^{q} \mu_{R_{ij}}(x_j^0); i = 1, 2, \ldots, m; j = 1,2, \ldots, q. \qquad (33)$$

In the DENFIS on-line model, the first-order Takagi-Sugeno type fuzzy rules are employed and thelinear functions in the consequences can be created and updated by linear least-square estimator (LSE) on the learning data. Each of the linear functions can be expressed as follows:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_q x_q. \qquad (34)$$

For obtaining these functions a learning procedure is applied on a data set, which is composed of p data pairs $\{([x_{i1}, x_{i2}, \ldots, x_{iq}], y_i), i = 1, 2, \ldots, p\}$. The least-square estimator (LSE) of $\boldsymbol{\beta} = [\beta_0\ \beta_1\ \beta_2 \ldots \beta_q]^T$, are calculated as the coefficients $\boldsymbol{b} = [b_0\ b_1\ b_2 \ldots b_q]^T$, by applying the following formula:

$$\boldsymbol{b} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\boldsymbol{y}. \qquad (35)$$

where

$$\mathbf{A} = \begin{pmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1q} \\ 1 & x_{21} & x_{22} & \ldots & x_{2q} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \end{pmatrix}$$

$$1 \quad x_{p1} \quad x_{p2} \quad \ldots \quad x_{pq}$$

and $\boldsymbol{y} = [y_1 \; y_2 \; \ldots, y_p]^T$.

A weighted least-square estimation method is used here as follows:

$$\boldsymbol{b}_w = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1} \mathbf{A}^T \mathbf{W} \boldsymbol{y}, \tag{36}$$

where:

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \ldots & 0 \\ 0 & w_2 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & \ldots & w_p \end{pmatrix}$$

and w_j is the distance between j-th example and the corresponding cluster centre, $j = 1, 2, \ldots p$.

We can rewrite the equations(35) and (36) as follows:

$$\begin{cases} \mathbf{P} = (\mathbf{A}^T\mathbf{A})^{-1}, \\ \boldsymbol{b} = \mathbf{P}\,\mathbf{A}^T\boldsymbol{y}. \end{cases} \tag{37}$$

$$\begin{cases} \mathbf{P}_w = (\mathbf{A}^T \mathbf{W} \mathbf{A})^{-1}, \\ \boldsymbol{b}_w = \mathbf{P}_w \mathbf{A}^T \mathbf{W} \boldsymbol{y}, \end{cases} \tag{38}$$

Let the k-th row vector of matrix A defined in Equation (35) be $a_k^T = [1 \; x_{k1} \; x_{k2} \; \ldots \; x_{kq}]$ and the k-th element of $\boldsymbol{y}$ be y_k, then $\boldsymbol{b}$ can be calculated iteratively as follows:

$$\begin{cases} \boldsymbol{b}_{k+1} = \boldsymbol{b}_k + \mathbf{P}_{k+1}\,\mathbf{a}_{k+1}\,(y_{k+1} - \mathbf{a}_{k+1}{}^T\,\boldsymbol{b}_k), \\ \mathbf{P}_{k+1} = \mathbf{P}_k - \dfrac{\mathbf{P}_k\,\mathbf{a}_{k+1}\,\mathbf{a}_{k+1}{}^T\,\mathbf{P}_k}{1 + \mathbf{a}_{k+1}{}^T\,\mathbf{P}_k\,\mathbf{a}_{k+1}} \end{cases} \tag{39}$$

for k = n, n+1, ... p – *1.*

Here, the initial values of $\mathbf{P}_n$ and $\boldsymbol{b}_n$ can be calculated directly from Equation (39) with the use of the first n data pairs from the learning data set.

The Equation (42) is the formula of recursive LSE. In the DENFIS on-line model, we use a weighted recursive LSE with forgetting factor defined as the following equations:

$$\boldsymbol{b}_{k+1} = \boldsymbol{b}_k + w_{k+1}\,\mathbf{P}_{k+1}\,\mathbf{a}_{k+1}\,(y_{k+1} - \mathbf{a}_{k+1}{}^T\,\boldsymbol{b}_k),$$

$$\mathbf{P}_{k+1} = \frac{1}{\lambda}\left[\mathbf{P}_k - \frac{w_{k+1}\,\mathbf{P}_k\,\mathbf{a}_{k+1}\,\mathbf{a}_{k+1}{}^{T}\,\mathbf{P}_k}{\lambda + \mathbf{a}_{k+1}{}^{T}\,\mathbf{P}_k\,\mathbf{a}_{k+1}}\right] \qquad k = n, n+1, \ldots p-1. \qquad (40)$$

where w is the weight defined in Equation (36) and λ is a forgetting factor with a typical value between 0.8 and 1.

In the on-line DENFIS model, the rules are created and updated at the same time with the input space partitioning using on-line evolving clustering method (ECM) and Equation (34) and (40). If no rule insertion is applied, the following steps are used for the creation of the first m fuzzy rules and for the calculation of the initial values $\mathbf{P}$ and $\boldsymbol{b}$ of the functions:

(1) Take the first n_0 learning data pairs from the learning data set.

(2) Implement clustering using ECM with these n_0 data to obtaining m cluster centres.

(3) For every cluster centre $\boldsymbol{C}_i$, find p_i data points whose positions in the input space are closest to the centre, $i = 1, 2, \ldots, m$.

(4) For obtaining a fuzzy rule corresponding to a cluster centre, create the antecedents of the fuzzy rule using the position of the cluster centre and Equation (34). Using Equation (38) on p_i data pairs calculate the values of $\mathbf{P}$ and $\boldsymbol{b}$ of the consequent function. The distances between p_i data points and the cluster centre are taken as the weights in Equation (38).

In the above steps, m, n_0 and p are the parameters of the DENFIS on-line learning model, and the value of p_i should be greater than the number of input elements, q.

As new data pairs are presented to the system, new fuzzy rules may be created and some existing rules are updated. A new fuzzy rule is created if a new cluster centre is found by the ECM. The antecedent of the new fuzzy rule is formed by using Equation (36) with the position of the cluster centre as a rule node. An existing fuzzy rule is found based on the rule node that is the closest to the new rule node; the consequence function of this rule is taken as the consequence function for the new fuzzy rule. For every data pair, several existing fuzzy rules are updated by using Equation (40) if their rule nodes have distances to the data point in the input space that are not greater than $2 \times Dthr$ (the threshold value, a clustering parameter). The distances between these rule nodes and the data point in the input space are taken as the weights in Equation (40). In addition to this, one of these rules may also be updated through changing its antecedent so that, if its rule node position is changed by the ECM, the fuzzy rule will have a new antecedent calculated through Equation (34).

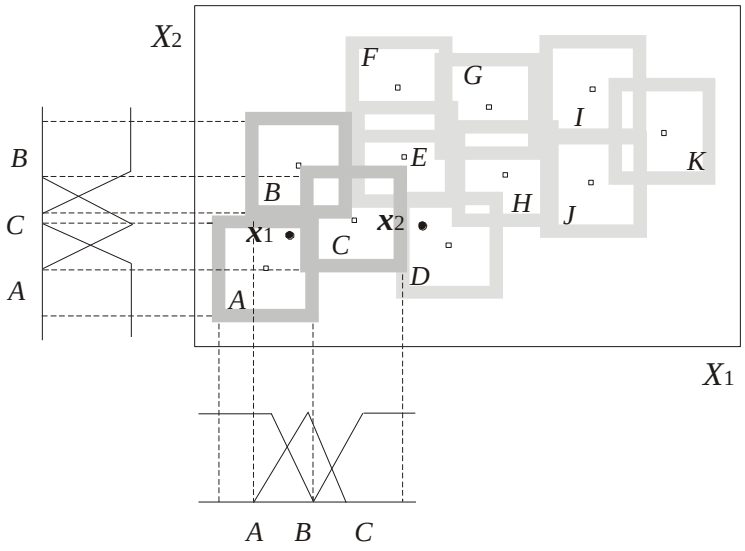
Takagi-Sugeno fuzzy inference in DENFIS

The Takagi-Sugeno fuzzy inference system utilised in DENFIS is a dynamic inference. In addition to dynamically creating and updating fuzzy rules the DENFIS on-line model has some other major differences from the other inference systems.

First, for each input vector, the DENFIS model chooses m fuzzy rules from the whole fuzzy rule set for forming a current inference system. This operation depends on the position of the current input vector in the input space. In case of two input vectors that are very close to each other, especially in the DENFIS off-line model, the inference system may have the same fuzzy rule inference group. In the DENFIS on-line model, however, even if two input vectors are exactly the same, their corresponding inference systems may be different. It is due to the reason that these two input vectors are presented to the system at different time moments and the fuzzy rules used for the first input vector might have been updated before the second input vector has arrived.

Second, depending on the position of the current input vector in the input space, the antecedents of the fuzzy rules chosen to form an inference system for this input vector may vary. An example is illustrated in Fig.18 where two different groups of fuzzy inference rules are formed depending on two input vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ respectively in a 2D input space. We can see from this example that, for instance, the region *C* has a linguistic meaning 'large', in the X_1 direction for fig. 17a group, but for the group of rules from fig. 17b it denotes a linguistic meaning of 'small' in the same direction of X1. The region *C* is defined by different membership functions respectively in each of the two groups of rules.

(a) Fuzzy rule group 1 for a DENFIS



(b) Fuzzy rule group 2 for a DENFIS

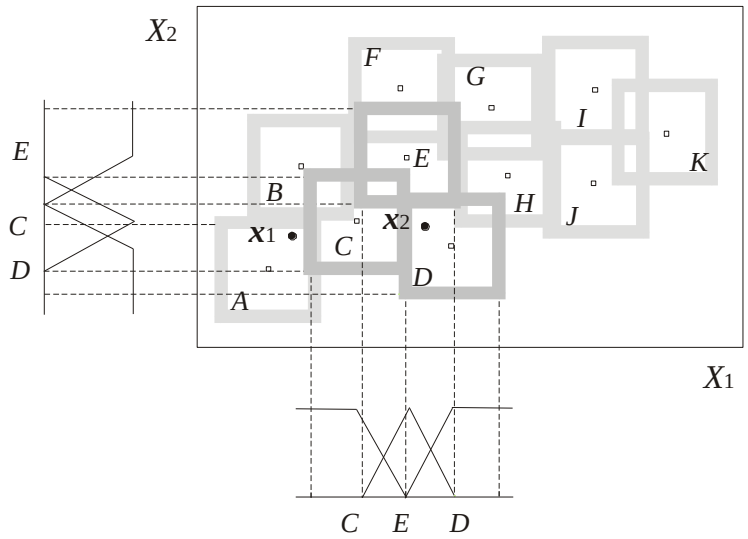


Figure. 17. Example of a fuzzy inference in DENFIS (after [37])

Example of a DENFIS application system is given in Fig. 18. Six input variables are used to train a DENFIS system for a medical decision support on a renal function GFR, where the evolved hidden nodes represent clusters of input data and the data in each of these clusters is approximated by a regression function.

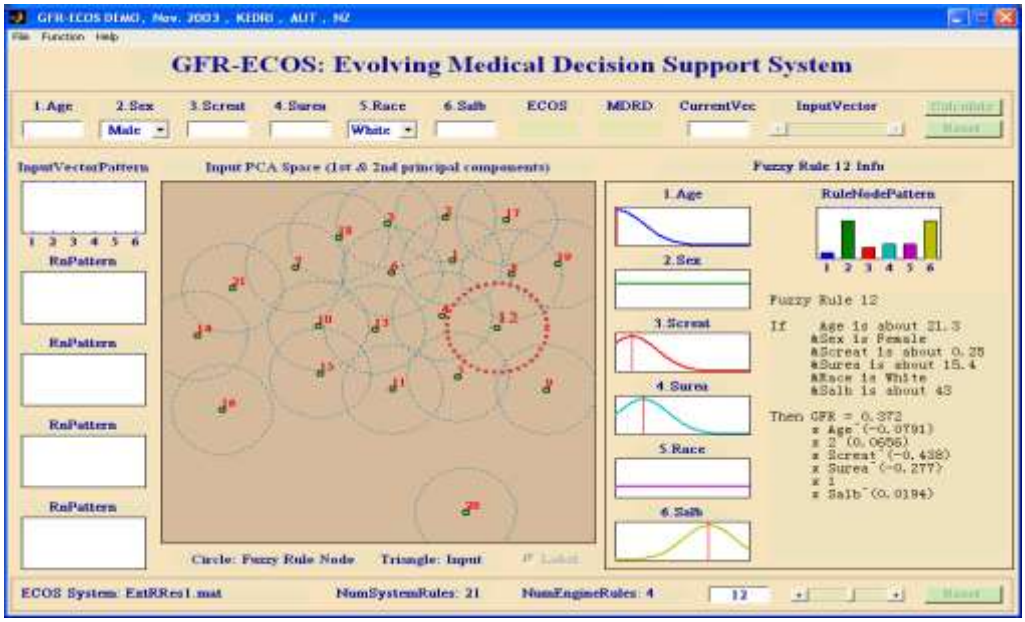


Figure 18. Example of a DENFIS application system using six input variables to train a DENFIS system for a medical decision support on a renal function GFR, where the evolved hidden nodes represent clusters of input data and the data in each of these clusters is approximated by a regression function ([37]).

2.6. Other ECOS methods and systems

A special development of ECOS is *transductive reasoning and personalized modelling*. Instead of building a set of local models (i.e., prototypes) to cover the whole problem space and then use these models to classify/predict any new input vector, in transductive modelling for every new input vector a new model is created based on selected nearest neighbor vectors from the available data. Such ECOS models are the neuro-fuzzy inference system (NFI) and the transductive weighted NFI (TWNFI) [51]. In TWNFI, for every new input vector the neighborhood of closets data vectors is optimized using both the distance between the new vector and the neighboring ones and the weighted importance of the input variables, so that the error of the model is minimized in the neighborhood area [52].

In addition to the already presented methods of ECOS, such as eMLP, eSOM, EFuNN, DENFIS, following is a short summary list of other methods, systems and applications that use some of the principles of ECOS:

- Evolving Self-Organized Maps (ESOM) [53];
- Evolving Clustering Method (ECM) [54];
- Incremental feature learning in ECOS [55];
- On-line ECOS optimization [56];

- Assessment of EFuNN accuracy for pattern recognition using data with different statistical distributions [57];
- Recursive clustering based on a Gustafson–Kessel algorithm [58];
- Using a map-based encoding to evolve plastic neural networks [59];
- Evolving Takagi–Sugeno fuzzy model based on switching to neighboring models [60];
- A soft computing based approach for modeling of chaotic time series [61];
- Uninorm based evolving neural networks and approximation capabilities [62];
- Global, local and personalised modelling and profile discovery in Bioinformatics: An integrated approach [63];
- FLEXFIS: a robust incremental learning approach for evolving Takagi–Sugeno fuzzy models [64];
- Evolving fuzzy classifiers using different model architectures [65];
- RSPOP: Rough Set–Based Pseudo Outer-Product Fuzzy Rule Identification Algorithm [66];
- SOFMLS: online self-organizing fuzzy modified least-squares network [67];
- On-Line Sequential Extreme Learning Machine [68];
- Finding features for real-time premature ventricular contraction detection using a fuzzy neural network system [69];
- Evolving fuzzy rule-based classifiers [70];
- A novel generic Hebbian ordering-based fuzzy rule base reduction approach to Mamdani neuro-fuzzy system [71];
- Implementation of fuzzy cognitive maps based on fuzzy neural network and application in prediction of time series [72];
- Backpropagation to train an evolving radial basis function neural network [73];
- Smooth transition autoregressive models and fuzzy rule-based systems: Functional equivalence and consequences [74];
- Development of an adaptive neuro-fuzzy classifier using linguistic hedges [75];
- A meta-cognitive sequential learning algorithm for neuro-fuzzy inference system [76];
- Meta-cognitive RBF network and its projection based learning algorithm for classification problems [77];
- SaFIN: A self-adaptive fuzzy inference network [78];
- A sequential learning algorithm for meta-cognitive neuro-fuzzy inference system for classification problems [79];
- Architecture for development of adaptive on-line prediction models [80];
- Clustering and co-evolution to construct neural network ensembles: An experimental study [81];
- Algorithms for real-time clustering and generation of rules from data [82];
- SAKM: Self-adaptive kernel machine - A kernel-based algorithm for online clustering [83];

- A BCM theory of meta-plasticity for online self-reorganizing fuzzy-associative learning [84];
- Evolutionary strategies and genetic algorithms for dynamic parameter optimization of evolving fuzzy neural networks [85];
- Incremental leaning and model selection for radial basis function network through sleep learning [86];
- Interval-based evolving modeling [87];
- Evolving granular classification neural networks [88];
- Stability analysis for an online evolving neuro-fuzzy recurrent network [89];
- A TSK fuzzy inference algorithm for online identification [90];
- Design of experiments in neuro-fuzzy systems [91];
- EFuNNs ensembles construction using a clustering method and a co-evolutionary genetic algorithm [92];
- eT2FIS: An evolving type-2 neural fuzzy inference system [93];
- Designing radial basis function networks for classification using differential evolution [94];
- A meta-cognitive neuro-fuzzy inference system (McFIS) for sequential classification problems [95];
- An evolving fuzzy neural network based on the mapping of similarities [96];
- Incremental learning by heterogeneous bagging ensemble [97];
- Fuzzy associative conjuncted maps network [98];
- EFuNN ensembles construction using CONE with multi-objective GA [99];
- Risk analysis and discovery of evolving economic clusters in Europe [100];
- Adaptive time series prediction for financial applications [101];
- Adaptive speech recognition [102];
- and others [37].

2.7. Chapter summary and further readings for deeper knowledge

The chapter presents foundations of artificial neural networks (ANN) and evolving connectionist systems. ECOS can not only be trained on data measuring evolving processes, but they can facilitate rule and knowledge extraction for a better understanding of these processes. Twenty four Centuries after Aristoteles, now the process of rule extraction and knowledge discovery from data can be automated. And not only that. The rules can be further adapted by incrementally training ECOS to accommodate new data and information about the problem in hand. These rules will no more be considered fixed and true for ever, but evolving as well.

Additional material to some of the sections can be found as follows:

- Neuro-fuzzy systems [21];

- ECOS [37, 109];
- Fuzzy systems [104];
- Neural networks [104];
- ECOS development system NeuCom (www.theneucom.com).

The ANN and ECOS methods and systems presented above use predominantly the McCulloch and Pitts model of a neuron (Fig. 1). They have been efficiently used for wide range of applications as some of them listed above. Many of the principles of ANN and ECOS presented in this chapter have been further developed and used for the creation of SNN and evolving spiking neural networks (eSNN) correspondingly (Chapters 4 and 5) and for brain-inspired SNN (chapter 6). Overview of the development of ECOS can be found in [109].

While the hybrid ANN and ECOS 'opened the black box" and provided means for rule and knowledge representation, these rules are "flat" rules, extracted from vector-based data. The ECOS methods were further developed into evolving SNN (eSNN) in Chapter 5. Instead of scalars as it is in ECOS, eSNN use information representation as spikes, the learning is based on times of spikes and fuzzy rules can be extracted.

Chapter 3 discusses how the brain learns from data as deep learning and how deep knowledge is represented. This is taken as inspiration for deep learning and deep knowledge representation in brain-inspired SNN, presented in chapter 6 and used in other chapters.

Acknowledgements

Some of the material in this chapter was first published by the author in [21], [33-37]. For the introduction of the principles of ECOS and of the ECOS models such as EFuNN and DENFIS, I have been inspired by the earlier work and by my discussions with pioneers, such as S.Amari, T.Kohonen, W.Freeman, L.Zadeh, T.Yamakawa, J.Taylor. I acknowledge the contribution of several of my co-authors of the early publications on ECOS methods, especially Qun Song, Mike Watts, Da Deng, Mathias Futschik, all PhD students and postdoctoral fellows of mine at the University of Otago.

REFERENCES

[1] F. Rosenblatt (1958), The Perceptron: A probabilistic model for information storage and organization in the brain, Psychology Review, 65, 1958, 386-402.

[2] S. Amari (1967), A Theory of Adaptive Pattern Classifiers. IEEE Transactions on Electronic Computers, **3** (vol.EC-16), 1967, pp. 299-307.

[3] S. Amari (1990), Mathematical Foundations of Neurocomputing. Proceedings of the IEEE, **9** (vol.78), 1990, pp. 1443-1463.

[4] D. Rumelhart and J. McLelland, J (1986), (eds) Parallel and Distributed Processing, MIT Press, 1986.

[5] P. Werbos (1990), Backpropagation through time, Proc. of the IEEE, 87:10, 1990.

[6] T. Kohonen (1992), Self-Organising Maps, Springer.

[7] G. A. Carpenter and S. Grossberg (2002), Adaptive Resonance Theory, MIT Press, 1998.

[8] R. S. Sutton and A. G. Barto (1998), Reinforcement Learning, MIT Press, 1998.

[9] C. M. Bishop (2006), Pattern recognition and machine learning. Springer, 2006.

[10] N. Kasabov (1998), Evolving Fuzzy Neural Networks – Algorithms, Applications and Biological Motivation. In: Yamakawa, T. and Matsumoto, G. (Eds.). Methodologies for the Conception, Design and Application of Soft Computing, World Scientific, 1998, pp 271 – 274.

[11] T. Kohonen (1977), Associative memory - a system-theoretical approach. Berlin, Heidelberg, New York: Springer 1977, 1978

[12] T. Kohonen (1982), Self-Organized Formation of Topologically Correct Feature Maps, Springer, Biological Cybernetics, Vol. 43, No.1, 1982, pp. 59-69.

[13] T. Kohonen (1990), The Self-Organizing Map, IEEE, Vol. 78, No. 9, 1990, pp. 1464-1480.

[14] T. Kohonen (1997), Self-Organizing Map, Springer, Verlag, New York, 1997, ISBN: 3-540-62017-6.

[15] A. V. Robins (1996), Consolidation in neural networks and the sleeping brain, Connection Science, Vol. 8, 1996, pp. 259–275

[16] G. Cybenko (1988), Continuous Valued Neural Networks with Two Hidden Layers are Sufficient, Technical Report, Department of Computer Science, Tufts University, 1988.

[17] S. Funahashi, C. J. Bruce and P. S. Goldman-Rakic (1989), Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex, Journal of Neurophysiol, Vol. 61, 1989, pp. 331–349.

[18] Y. Saad (2000), Further Analysis of Minimum Residual Iteration, Numerical Linear Algebra with Applications, Vol. 7, No. 2, 2000, pp. 67-93.

[19] K. Fukushima (1975), Cognitron: A Self-Organizing Multilayered Neural Network, Springer, Biological Cybernetics, Vol. 20, No. 3-4, 1975, pp. 121-136.

[20] K. Fukushima (1980), Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, Springer, Biological Cybernetics, Vol. 36, 1980, pp. 193-202.

[21] N. Kasabov (1996), Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, MIT Press, Cambridge, Massachusetts, USA, 1996.

[22] N. Kasabov and S. I. Shishkov (1993), A Connectionist Production System with Partial Match and its Use for Approximate Reasoning. Connection Science, Vol. 5, No. 3-4, 1993, pp. 275-305.

[23] M. Feigenbaum (1989), Artificial Intelligence, A knowledge-based approach, PWS-Kent, Boston.

[24] S. Gallant (1993), Neural Network Learning and Expert Systems, MIT Press, Bradford, Cambridge, MA.

[25] J. Hendler and L. Dickens (1991), Integrating Neural Network and Expert Reasoning: An Example. In: L. Steels and B. Smith (Eds), Proceeding of AISB Conference, Springer Verlag, New York, pp. 109-116.

[26] L. Fu (1989), Integration of Neural Heuristic into Knowledge-based Inference, Connection Science, Vol. 1, No. 3, 1989.

[27] L. A. Zadeh (1965), Fuzzy Sets, Information and Control, Vol. 8, 1988, pp. 338-353.

[28] E. Mamdani (1977), Application on Fuzzy Logic to Approximate Reasoning using Heuristic Synthesis, IEEE, Transaction on Computers, Vol. 26, No. 12, 1977, pp. 1182-1191.

[29] T. Takagi and M. Sugeno (1985), Fuzzy Identification of Systems and its Applications to Modelling and Control, IEEE, Transaction on Systems Man and Cybernetics, Vol. 15, 1985, pp. 116-132.

[30] R. Jang (1993), ANFIS: Adaptive Network Based Fuzzy Inference System, IEEE, Transaction on Systems Man and Cybernetics, Vol. 23, No. 3, 1993, pp. 665-685.

[31] T. Yamakawa, E. Uchino, T. Miki, and H. Kusanagi (1992), A Neo Fuzzy Neuron and Its Application to System Identification and Prediction of the System Behavior. Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks, Japan, July 1992, pp. 477-483.

[32] T. Furuhashi, T. Hasegawa, S. Horikawa and Y. Uchikawa (1993), An Adaptive Fuzzy Controller Using Fuzzy Neural Networks. Proc. of 5th Int'l Fuzzy System Association World Congress, Korea, July 1993, pp.769-772.

[33] N. Kasabov, J. S. Kim, M. J. Watts and A. R. Gray (1997), FuNN/2 – A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition. Information Sciences – Applications, 3-4 (vol.101), 1997, pp. 155-175.

[34] N. Kasabov (2001), Evolving Fuzzy Neural Networks for On-Line Supervised/ Unsupervised, Knowledge–Based Learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, **6** (vol.31), 2001, pp. 902-918.

[35] N. Kasabov (2002), Evolving Connectionist Systems: Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines. Springer-Verlag, London, England, 2002.

[36] N. Kasabov and Q. Song (2002), DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and Its Application for Time-Series Prediction. IEEE Transactions on Fuzzy Systems, 2 (vol.10), 2002, pp. 144-154.

[37] N. Kasabov (2007), Evolving Connectionist Systems: The Knowledge Engineering Approach (2nd ed.), Springer-Verlag, London, England, 2007.

[38] M. E. Futschik and N. Kasabov (2002), Fuzzy Clustering in Gene Expression Data Analysis. Proc. of the IEEE Int'l Conf. on Fuzzy Systems, USA, 2002, pp. 414-419.

[39] M. J. Watts (2009), A Decade of Kasabov's Evolving Connectionist Systems: A Review. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, **3** (vol.39), 2009, pp. 253-269.

[40] J. C. Bezdek (1987), Analysis of Fuzzy Information, CRC Press, Boca Raton, Florida, USA, 1987.

[41] R. R. Yager and D. P. Filev (1994), Generation of Fuzzy Rules by Mountain Clustering. Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology, **3** (vol.2), 1994, pp. 209-219.

[42] J. Si, S. Lin and M. A. Vuong (2000), Dynamic Topology Representing Networks, Neural Network, Vol. 13, pp. 617-627.

[43] D. Deng and N. Kasabov (2000), ESOM: An Algorithm to Evolve Self-Organizing Maps from Online Data Streams, In: Proceedings of IJCNN's 2000, Vol. VI, Como, Italy, 2000, pp. 3-8.

[44] N. Kasabov (2001), Ensembles of EFuNNs: An Architecture for a Multi-Module Classifire, In Proceedings of the International Conference on Fuzzy Systems, Australia, Vol. 3, 2001, pp. 1573-1576.

[45] M. Watts and N. Kasabov (2002), Evolutionary Computation for the Optimization of Evolving Connectionist Systems, In Proceedings of WCCI'2002 (World Congress of Computational Intelligence), Hawaii, May, 2002, IEEE Press, Washington, DC.

[46] M. J. Watts (2006), Nominal-Scale Evolving Connectionist Systems. In Proceedings of IEEE International Joint Conference on Neural Networks, Vancouver, July 16-21, 2006, IEEE Press, Washington, DC, pp. 4057-4061.

[47] C. T. Lin, and C. S. G. Lee (1996), Neuro Fuzzy Systems, Prentice-Hall, Upper Saddle River, NJ.

[48] B. Widrow and M. E. Hoff (1960), Adaptive Switching Circuits, IPE WESCON Convention Rec., Vol. 4, pp. 96-104.

[49] J. Platt (1991), A Resource Allocating Network for Function interpolation, Neural Computation, Vol. 3, 1991, pp. 213-225.

[50] K. Yamauchi and J. Hayami (2006), Sleep Learning-An Incremental Learning System Inspired by Sleep behavior, In Proceeding IEEE International Conference on Fuzzy Systems, Vancouver, July, 16-21, IEEE Press, Piscataway, NJ, pp. 6295-6302.

[51] Q. Song, and N. Kasabov (2006), TWNFI – A Transductive Neuro-Fuzzy Inference System with Weighted Data Normalization for Personalized Modelling. Neural Networks, **10** (vol.19), 2006, pp. 1591-1596.

[52] N. Kasabov and Y. Hu (2010), Integrated Optimisation Method for Personalised Modelling and Case Study Applications. Int'l Journal of Functional Informatics and Personalised Medicine, **3** (vol.3), 2010, pp. 236-256.

[53] D. Deng and N. Kasabov (2003), On-Line Pattern Analysis by Evolving Self-Organizing Maps. Neurocomputing, (vol.51), 2003, pp. 87-103.

[54] Q. Song and N. Kasabov (2005), NFI: A Neuro-Fuzzy Inference Method for Transductive Reasoning, IEEE Trans. on Fuzzy Systems, 13, 6, 2005, 799-808.

[55] S. Ozawa, S. Too and S. Abe (2005), S. Pang and N. Kasabov, Incremental Learning of Feature Space and Classifier for Online Face Recognition, Neural Networks, August, 2005, 575-584.

[56] Z. Chan and N. Kasabov (2004), Evolutionary computation for on-line and off-line parameter tuning of evolving fuzzy neural networks, Int. J. of Computational Intelligence and Applications, Imperial College Press, 4, 3, 2004, 309-319.

[57] R. M. de Moraes (2012), Assessment of EFuNN Accuracy for Pattern Recognition Using Data with Different Statistical Distributions. Proc. of the 2nd Brazilian Congress on Fuzzy Systems, Brazil, November 2012, pp. 672-685.

[58] D. Dovžan and I. Škrjanc (2011), Recursive Clustering Based on a Gustafson–Kessel Algorithm. Evolving Systems, 1 (vol.2), 2011, pp. 15-24.

[59] P. Tonelli and J. B. Mouret (2011), Using a Map-Based Encoding to Evolve Plastic Neural Networks. Proc. of the IEEE Workshop on Evolving and Adaptive Intelligent Systems, France, April 2011, pp. 9-16.

[60] A. Kalhor, B. N. Araabi and C. Lucas (2013), Evolving Takagi–Sugeno Fuzzy Model Based on Switching to Neighboring Models. Applied Soft Computing, **2** (vol.13), 2013, pp. 939-946.

[61] J. Vajpai and J. B. Arun (2006), A Soft Computing Based Approach for Modeling of Chaotic Time Series. Proc. of the 13th Int'l Conf. on Neural Information Processing, China, October 2006, pp. 505-512.

[62] F. Bordignon and F. Gomide (2014), Uninorm Based Evolving Neural Networks and Approximation Capabilities. Neurocomputing, (vol.127), 2014, pp. 13-20.

[63] N. Kasabov (2007), Global, local and personalised modelling and profile discovery in Bioinformatics: An integrated approach, Pattern Recognition Letters, 28, 6, 2007, 673-685.

[64] E. D. Lughofer (2008), FLEXFIS: A Robust Incremental Learning Approach for Evolving Takagi–Sugeno Fuzzy Models. IEEE Transactions on Fuzzy Systems, **6** (vol.16), 2008, pp. 1393-1410.

[65] P. P. Angelov, E. D. Lughofer and X. Zhou (2008), X. Evolving fuzzy classifiers using different model architectures. Fuzzy Sets and Systems, **23** (vol.159), 2008, pp. 3160-3182.

[66] K. K. Ang and C. Quek (2005), RSPOP: rough set-based pseudo outer-product fuzzy rule identification algorithm. Neural Computation, **1** (vol.17), 2005, pp. 205-243.

[67] J. de Jesús Rubio (2009), SOFMLS: Online Self-Organizing Fuzzy Modified Least-Squares Network. IEEE Transactions on Fuzzy Systems, **6** (vol.17), 2009, pp. 1296-1309.

[68] G. B. Huang, N. Y. Liang and H. J. Rong (2005), On-Line Sequential Extreme Learning Machine. Proc. of the IASTED Int'l Conf. on Computational Intelligence, Canada, July 2005, pp. 232-237.

[69] J. S. Lim (2009), Finding Features for Real-Time Premature Ventricular Contraction Detection Using a Fuzzy Neural Network System. IEEE Transactions on Neural Networks, **3** (vol.20), 2009, pp. 522-527.

[70] P. P. Angelov, X. Zhou and F. Klawonn (2007), Evolving Fuzzy Rule-based Classifiers. Proc. of the IEEE Symposium on Computational Intelligence in Image and Signal Processing, USA, April 2007, pp. 220-225.

[71] F. Liu, C. Quek and G. S. Ng (2007), A Novel Generic Hebbian Ordering-Based Fuzzy Rule Base Reduction Approach to Mamdani Neuro-Fuzzy System. Neural Computation, **6** (vol.19), 2007, pp. 1656-1680.

[72] H. Song, C. Miao, W. Roel, Z. Shen and F. Catthoor (2010), Implementation of Fuzzy Cognitive Maps Based on Fuzzy Neural Network and Application in Prediction of Time Series. IEEE Transactions on Fuzzy Systems, **2** (vol.18), 2010, pp. 233-250.

[73] J. de Jesús Rubio, D. M. Vázquez and J. Pacheco (2010), Backpropagation to Train an Evolving Radial Basis Function Neural Network. Evolving Systems, **3** (vol.1), 2010, pp. 173-180.

[74] J. L. Aznarte, J. M. Benítez and J. L. Castro (2007), Smooth Transition Autoregressive Models and Fuzzy Rule-Based Systems: Functional Equivalence and Consequences. Fuzzy Sets and Systems, **24** (vol.158), 2007, pp. 2734-2745.

[75] B. Cetisli (2010), Development of an Adaptive Neuro-Fuzzy Classifier Using Linguistic Hedges. Expert Systems with Applications, **8** (vol.37), 2010, pp. 6093-6101.

[76] K. Subramanian and S. Suresh (2012), A Meta-Cognitive Sequential Learning Algorithm for Neuro-Fuzzy Inference System. Applied Soft Computing, **11** (vol.12), 2012, pp. 3603-3614.

[77] G. S. Babu and S. Suresh (2013), Meta-Cognitive RBF Network and Its Projection Based Learning Algorithm for Classification Problems. Applied Soft Computing, **1** (vol.13), 2013, pp. 654-666.

[78] S. W. Tung, C. Quek and C. Guan (2011), SaFIN: A Self-Adaptive Fuzzy Inference Network. IEEE Transactions on Neural Networks, **12** (vol.22), 2011, pp. 1928-1940.

[79] S. Suresh and K. Subramanian (2011), A Sequential Learning Algorithm for Meta-Cognitive Neuro-Fuzzy Inference System for Classification Problems. Proc. of the Int'l Joint Conf. on Neural Networks, USA, August 2011, pp. 2507-2512.

[80] P. Kadlec and B. Gabrys (2009), Architecture for Development of Adaptive On-Line Prediction Models. Memetic Computing, **4** (vol.1), 2009, pp. 241-269.

[81] F. L. Minku and T. B. Ludemir (2008), Clustering and Co-Evolution to Construct Neural Network Ensembles: An Experimental Study. Neural Networks, **9** (vol.21), 2008, pp. 1363-1379.

[82] D. P. Filev and P. P. Angelov (2007), Algorithms for Real-time Clustering and Generation of Rules from Data. In: Valente di Oliveira, J., and Pedrycz, W. (Eds.). Advances in Fuzzy Clustering and its Applications, John Wiley & Sons, Chichester, UK, 2007.

[83] H. Amadou Boubacar, S. Lecoeuche, and S. Maouche (2008), SAKM: Self-Adaptive Kernel Machine: A Kernel-Based Algorithm for Online Clustering. Neural Networks, **9** (vol.21), 2008, pp. 1287-1301.

[84] J. Tan and C. Quek (2010), A BCM Theory of Meta-Plasticity for Online Self-Reorganizing Fuzzy-Associative Learning. IEEE Transactions on Neural Networks, **6** (vol.21), 2010, pp. 985-1003.

[85] F. L. Minku and T. B. Ludermir (2005), Evolutionary Strategies and Genetic Algorithms for Dynamic Parameter Optimization of Evolving Fuzzy Neural Networks. Proc. of the IEEE Congress on Evolutionary Computation, Scotland, 2005, pp. 1951-1958.

[86] K. Yamauchi and J. Hayami (2007), Incremental Leaning and Model Selection for Radial Basis Function Network through Sleep. IEICE Transactions on Information and Systems, **4** (vol.e90-d), 2007, pp. 722-735.

[87] D. F. Leite, P. Costa and F. Gomide (2009), Interval-Based Evolving Modeling. Proc. of the IEEE Workshop on Evolving and Self-Developing Intelligent Systems, USA, March 2009, pp. 1-8.

[88] D. F. Leite, P. Costa, and F. Gomide (2013), Evolving Granular Neural Networks From Fuzzy Data Streams. Neural Networks, (vol.38), 2013, pp. 1-16.

[89] J. de Jesús Rubio (2010), Stability Analysis for an Online Evolving Neuro-Fuzzy Recurrent Network. In: Angelov, P. P., Filev, D. P., Kasabov, N. K. (Eds.). Evolving Intelligent Systems: Methodology and Applications, John Wiley & Sons, Hoboken, New Jersey, USA, 2010.

[90] K. Kim, E. J. Whang, C. W. Park, E. Kim and M. Park (2005), A TSK Fuzzy Inference Algorithm for Online Identification. Proc. of the 2nd Int'l Conf. on Fuzzy Systems and Knowledge Discovery, China, August 2005, pp. 179-188.

[91] C. Zanchettin, L. L. Minku and T. B. Ludermir (2010), Design of Experiments in Neuro-Fuzzy Systems. International Journal of Computational Intelligence and Applications, **2** (vol.9), 2010, pp. 137-152.

[92] F. L. Minku and T. B. Ludermir (2006), EFuNNs Ensembles Construction Using a Clustering Method and a Coevolutionary Genetic Algorithm. Proc. of the IEEE Congress on Evolutionary Computation, Canada, July 2006, pp. 1399-1406.

[93] S. W. Tung, C. Quek and C. Guan (2013), eT2FIS: An Evolving Type-2 Neural Fuzzy Inference System. Information Sciences, (vol.220), 2013, pp. 124-148.

[94] B. O'Hara, J. Perera and A. Brabazon (2006), Designing Radial Basis Function Networks for Classification Using Differential Evolution. Proc. of the Int'l Joint Conf. on Neural Networks, Canada, July 2006, pp. 2932-2937.

[95] K. Subramanian, S. Sundaram and N. Sundararajan (2013), A Metacognitive Neuro-Fuzzy Inference System (McFIS) for Sequential Classification Problems. IEEE Transactions on Fuzzy Systems, **6** (vol.21), 2013, pp. 1080-1095.

[96] J. A. M. Hernández, F. G. Castañeda and J. A. M. Cadenas (2009), An Evolving Fuzzy Neural Network Based on the Mapping of Similarities. IEEE Transactions on Fuzzy Systems, **6** (vol.17), 2009, pp. 1379-1396.

[97] Q. L. Zhao, Y. H. Jiang and M. Xu (2010), Incremental Learning by Heterogeneous Bagging Ensemble. Proc. of the Int'l Conf. on Advanced Data Mining and Applications, China, November 2010, pp. 1-12.

[98] H. Goh, J. H. Lim and C. Quek (2009), Fuzzy Associative Conjuncted Maps Network. IEEE Transactions on Neural Networks, **8** (vol.20), 2009, pp. 1302-1319.

[99] F. L. Minku and T. B. Ludermir (2006), EFuNN Ensembles Construction Using CONE with Multi-objective GA. Proc. of the 9th Brazilian Symposium on Neural Networks, Brazil, October 2006, pp. 48-53.

[100] N. Kasabov (2006), Adaptation and Interaction in Dynamical Systems: Modelling and Rule Discovery Through Evolving Connectionist Systems, Applied Soft Computing, 6, 3, 2006, 307-322.

[101] H. Widiputra, R. Pears, N. Kasabov (2011), Dynamic Interaction Network versus Localized Trends Model for Multiple Time-Series Prediction, Cybernetics and Systems, 42, 2, 2011, 100-123.

[102] A. Ghobakhlou, M. Watts and N. Kasabov (2003), Adaptive speech recognition with evolving connectionist systems, Information Sciences, 156, 2003, 71-83.

[103] N.Kasabov (ed) Springer Handbook of Bio-/Neuroinformatics, Springer, 2014.

[104] J.Kacprzyk (ed) Springer Handbook of Computational Intelligence, Springer, 2015

[105] Wikipedia, http://www.wikipedia.org.

[106] L.Benuskova, N.Kasabov, Computational neurogenetic modelling, Springer, 2007.

[107] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276.

[108] Felix A. Gers; Jürgen Schmidhuber; Fred Cummins (2000). "Learning to Forget: Continual Prediction with LSTM". Neural Computation. 12 (10): 2451–2471.

[109] N. Kasabov (2015) Evolving connectionist systems: From neuro-fuzzy-, to spiking – and neurogenetic, in: Kacprzyk and Pedrycz (eds) Springer Handbook of Computational Intelligence, Springer, 771-782.

[110] J. Schmidhuber (2015), Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[111] Y. LeCun, Y. Bengio and G. Hinton (2015), Deep learning. Nature, 521(7553), 436, doi:10.1038/nature14539

[112] A. Krizhevsky, L. Sutskever and G. E. Hinton (2012), Image net classification with deep convolutional neural networks. Proc. Advances in Neural Information Processing Systems, pp. 1097-1105.

[113] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, W. Murdock, E. Nyberg, J. Prager and N. Schlaefer (2010), Building Watson: An overview of the DeepQA project. AI magazine, 31(3), 59-79, doi.org/10.2109/aimag.v31i3.2303

[114] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, S. A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath and B. Kingsbury (2012), Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), 82-97, doi:10.1109/MSP.2012.2205597

[115] L. Sutskever, O. Vinyals and Q. V. Le (2014), Sequence to sequence learning with neural networks. In Advances in neural information processing systems, 3104-3112.

[116] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche and S. Dieleman (2021), Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489, doi:10.1038/nature21961