Optimization techniques can be classified as classical optimization techniques and meta-heuristic optimization techniques. In recent decades, the ever-increasing complexity and difficulty of real-world problems resulted in the need for more reliable optimization techniques, especially meta-heuristic optimization techniques. These techniques are mostly stochastic and estimate optimal solutions for different optimization problems. Such optimization techniques supersede classical optimization techniques due to gradient-free (derivative-free) mechanisms and high local optimal avoidance capability. The optimization process finds the optimal decision variables of a function or a problem by minimizing or maximizing its objective function. Generally speaking, real-world and optimization problems have non-linear restrictions, complex, high computational time, non-convex, and wide search spaces, which make them challenging to solve.

Meta-heuristic optimization algorithms have two important search strategies: (1) exploration/diversification and(2) exploitation/intensification. Exploration is the capability to explore the search space globally. This ability is related to the avoidance of local optima and resolving local optima entrapment. On the contrary, exploitation is the capability to explore nearby promising solutions to improve their quality locally. Excellent performance of an algorithm requires a proper balance between these two strategies. All population-based algorithms use these features but with different operators and mechanisms.

Meta-heuristics techniques are classified as artificial neural networks based on the human nervous system, evolutionary algorithms, swarm intelligence algorithms and physics-based methods. Evolutionary algorithms simulate habits in natural evolution and use operators motivated by biology behaviors like crossover and mutation. A classical evolutionary algorithm is the genetic algorithm (GA), which is motivated by Darwinian evolutionary ideas. Other methods of this group include evolutionary programming, differential evolution, etc.

Swarm intelligence algorithms are another group of meta-heuristics, which simulates the behavior of animals in movement or hunting groups. The main characteristic of this group is the sharing of organism information of all animals through the optimization course. Different methods of this group include particle swarm optimization, cuckoo search optimization Algorithm, artificial bee colony optimization, symbiotic organisms search and many more.

Physics-based methods are another group of optimization algorithms. This group originates from physical laws in real-life and typically describes the communication of search solutions based on controlling rules ingrained in physical methods. The most commonly utilized algorithms in this group are simulated annealing, gravitational search algorithm and many more.

Artificial neural networks

Nonlinear models based on artificial intelligence (AI) methods have been widely implemented over the years in several fields of scientific research, and the most important among them is the ANN. ANN models were designed in the second half of the 20th century by mathematical simulation of the procedures on which the human nervous system works. Artificial neural networks (ANNs) have received considerable attention due to its powerful ability in image processing, speech recognition, natural language processing, data processing, process analysis and control, fault detection, pattern recognition etc. The ANN models performance depends largely on the quantity and quality of data, computing power, and the efficiency of algorithms. Traditional ANNs train models by iteratively tuning all the weights and biases in minimizing a loss function which is defined as the difference between model predictions and real observations. During the training process, the derivatives of the loss function are back propagated to each layer to guide parameter adjustment

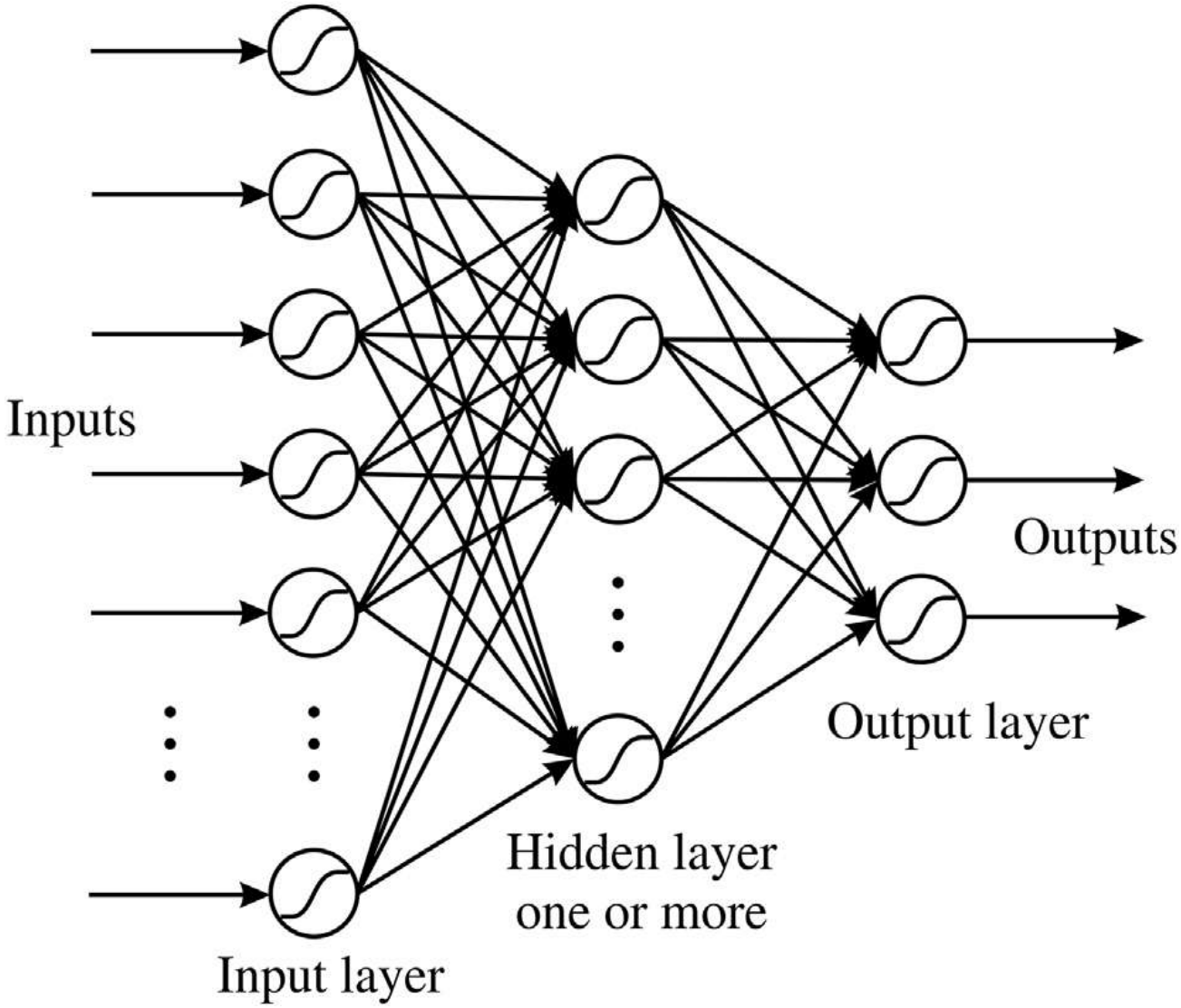

Fig. 1. Schematic diagram of an artificial neuron. From left to right: inputs ($i_1 - i_n$) are multiplied with synaptic weights ($w_1 - w_n$), the products are added (Σ), and the result is passed from a non-linear function to produce the neuron output.

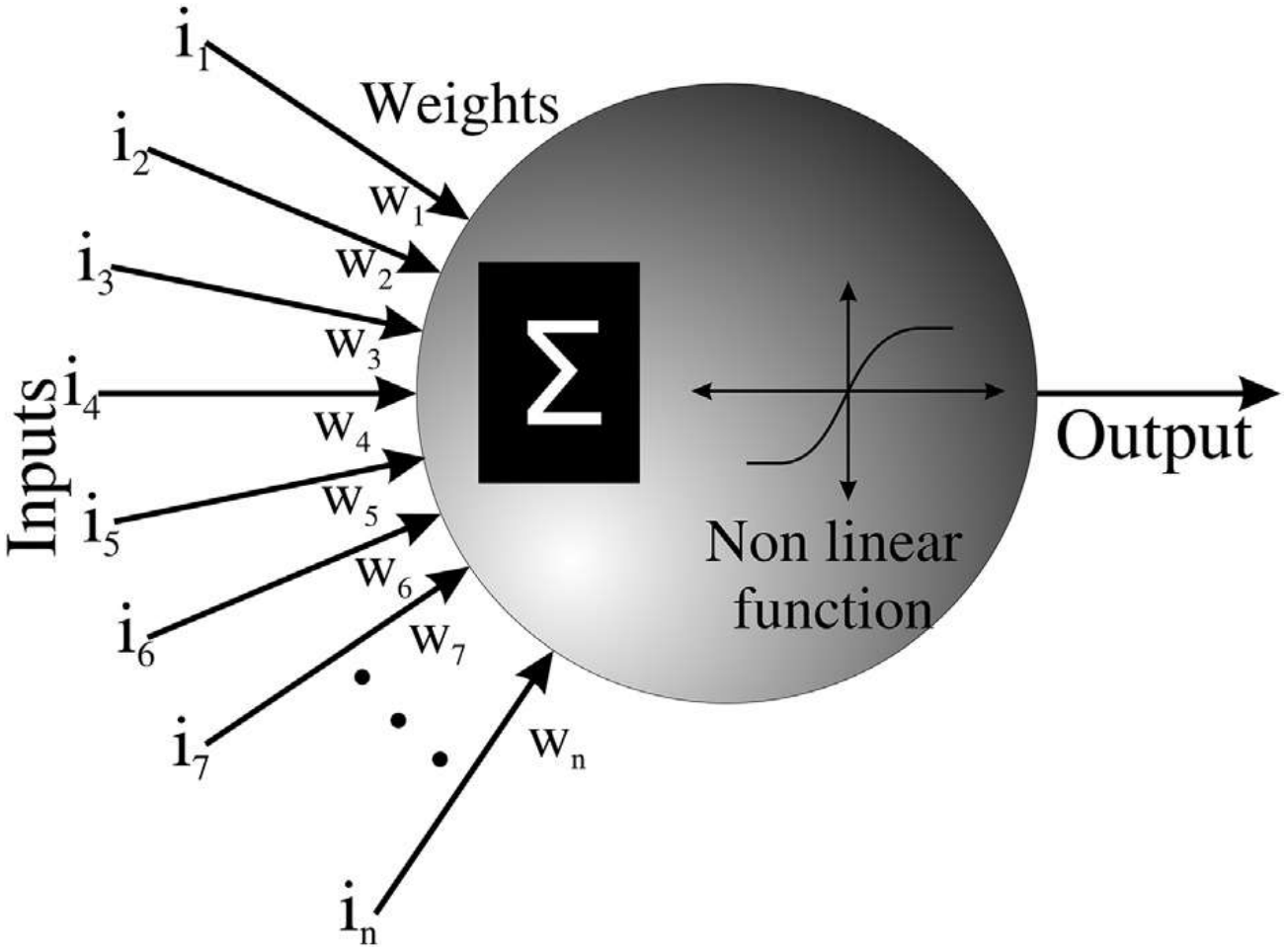

Fig. 2. Typical structure of a feed-forward multilayer neural network.

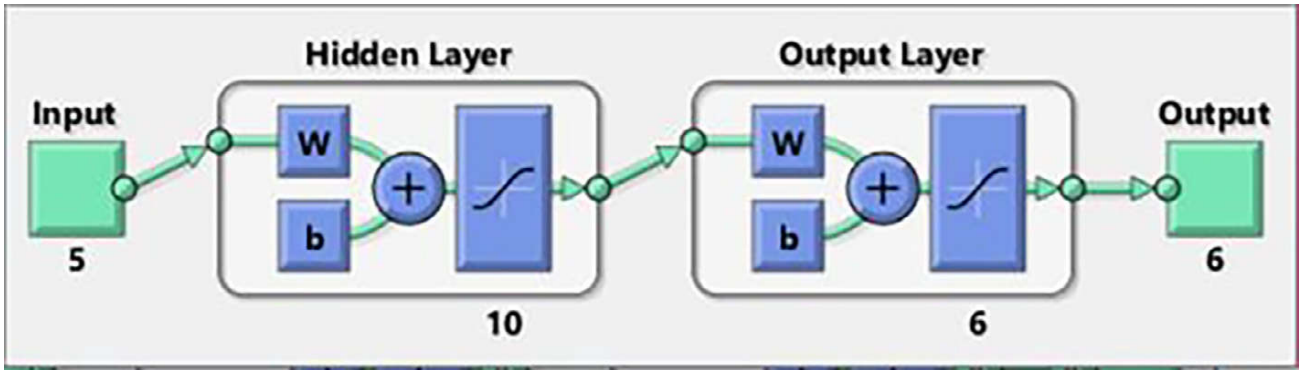

Fig. 3. Neural network architecture.

ANNs are generally categorized by feed-forward networks (Perceptron network), competitive networks (Hamming network) and recurrent networks (Hopfield network).

Multilayer perceptron neural networks or Multi-layer feed-forward (MLF) neural networks

MLF neural networks, trained with a back-propagation learning algorithm, are the most popular neural networks.

By definition, an ANN is nonlinear mathematical model inspired from the function of the human brain and considered as an information-processing system composed of many processing element arranged in several parallel layer. Feed-forward neural network architecture, network structure in which the information or signals will generate only in one direction, from input to output, is the most common neural network architecture. The number of inputs in the process is the same as the number of nodes in the feed forward neural network input layer, whereas the number of output nodes is equal to the number of process output. Training procedure of MLFNN is based on algorithm of back propagation.

Each layer is composed of several simple units called neurons interconnected with each other and each of which plays a role within the networks. Generally, there are three types of layers in ANN: (i) input, (ii) one or more hidden and (iii) and one output layers.

Each layer consists of neurons or nodes. In the input and output layer, the number of neurons is equal to the number of inputs and outputs or targeted variables.

The available information's or signal is transmitted between the layers through connections. The inputs and output layers contains the two important component of the data base: the input layer is composed of a number of neuron equal to the number of predictor and the output layer contains the unique neuron that corresponds to the dependent variable. The relationship between the input and the output is identified through the learning process. In the ANN model, the neurons are connected by a large number of parameters: the weights and biases. The ANN consists of four main components (i) the neurons, (ii) the parameters (weights and biases), (iii) the activation function for the hidden and output layers and (iv) the learning algorithm for adjusting the weights and biases. Each neuron in the hidden layer consists of two sub models: linear and nonlinear. Firstly, the neuron in the hidden layer receive the entire incoming inputs variable multiplied by its weight (w_{ij}) and adding bias terms (B_j) to the results. In the second part a nonlinear model is applied to the results by an activation function, generally the sigmoidal. The most popular ANN model is the multilayer perceptron neural network (MLPNN) and considered as universal approximators as long as sufficient training is performed. The MLPNN model is composed of three layers of processing units, with one hidden layer by using a sigmoid activation function which shows a better robustness structure. The determination of hidden neurons number is performed based on trial and error rule. The result which displays minimum error allow to set the hidden neurons number. In other word, the ideal network structure was searched by changing the number of neurons in the hidden layer till finding the optimal MLPNN topologies. The structure of the MLPNN is shown in Fig. 1. In this figure, the model is only feed forward and each neuron receives a weighted sum of outputs of all the previous neurons. According to Fig. 1, a simplified manner of mathematical formulation of the MLPNN can be presented. The input of the each neuron in the hidden layer is given as follows:

$$\mathrm{A}_j = \sum_{i=1}^{\mathrm{N}_i} w_{ij} \times x_i + \mathrm{B}_j \quad (1)$$

N_i is the total number of input neurons, w_{ij} denotes the weight characterizing the connection between the i th input neuron to the j th hidden neuron, and B_j is the bias term (or threshold) of each hidden neuron. The output of the j th hidden neuron is given by:

$$Y_j = f\left(\mathrm{A}_j\right) \qquad (2)$$

The most commonly used transfer functions for ANN are the S-shaped log-sigmoid function, Eq. (3), and the tan-sigmoid function, Eq. (4). The log-sigmoid function produces an output ranging between 0 and 1, whereas, for the tan-sigmoid function, the range is between −1 and 1.

The activation function f is represented by Eq. (3) or (4)

$$f\left(\mathrm{A}\right) = \frac{1}{1+e^{-\mathrm{A}}} \qquad (3)$$

$$f\left(\mathrm{A}\right) = \frac{e^{2\mathrm{A}}-1}{e^{2\mathrm{A}}+1} \qquad (4)$$

The output layer uses a pure linear transfer function. The neural network output is given by

$$\mathrm{O}_k = \sum_{j=1}^{\mathrm{N}_h} w_{jk} \times Y_j + \mathrm{B}_k = \sum_{j=1}^{\mathrm{N}_h} w_{jk} \times f\left(\mathrm{A}_j\right) + \mathrm{B}_k \qquad (5)$$

Where w_{jk} denotes the weight between the j th hidden neuron to the k output neuron, N_h is the total number of hidden neurons and B_k is the bias term of each output neuron.

Using the MLPNN model, the weights and biases must be optimized to obtain the best model with high generalization capacity using a training algorithm which is generally represented by the back-propagation (BP) algorithm. The network was trained using the Levenberg-Marquardt (LM) back propagation optimization method due to its fast convergence and the high performance of the network.

During the training process, the error which is calculated as the sum of squared differences between the desired and the calculated values of the output neuron, must be minimized by adjusting the weights and biases to each neuron using the optimization method.

The mean squared error (e or $\mathrm{M}SE$) for the entire network is given as:

$$e = \mathrm{M}SE = \frac{1}{2}\sum_{k=1}^{\mathrm{N}_o}\left(E_k\right)^2 = \frac{1}{2}\sum_{k=1}^{\mathrm{N}_o}\left(\mathrm{O}_k - D_k\right)^2 \qquad (6)$$

where D_{lk} is the desired value of output neuron k for a given training pattern; O_k is the calculated value of that neuron and N_o is the total number of output neurons. Each pattern corresponds to input-output pairs of the used data.

By the error function derivation of the weights and threshold of the output point, we calculate the following.

$$\frac{\partial e}{\partial w_{jk}} \text{ and } \frac{\partial e}{\partial \mathrm{B}_k}$$

In the output layer node, according to the traditional BP neural network training algorithm, the weight and threshold adjustment formula are as follows.

$$w_{jk}\left(n+1\right)=w_{jk}\left(n\right)+\eta\times\frac{\partial e}{\partial w_{jk}} \quad (7)$$

$$\mathrm{B}_k\left(n+1\right)=\mathrm{B}_k\left(n\right)+\eta\times\frac{\partial e}{\partial \mathrm{B}_k} \quad (8)$$

Similarly, in the hidden layer nodes, the weight and threshold are as follows.

$$w_{ij}\left(n+1\right)=w_{ij}\left(n\right)+\eta\times\frac{\partial e}{\partial w_{ij}} \quad (9)$$

$$\mathrm{B}_j\left(n+1\right)=\mathrm{B}_j\left(n\right)+\eta\times\frac{\partial e}{\partial \mathrm{B}_j} \quad (10)$$

η is the learning rate of the BP neural network. The adjustment algorithm for the weights and the threshold is the key feature of the training model that affects the accuracy of building electricity consumption forecasting.

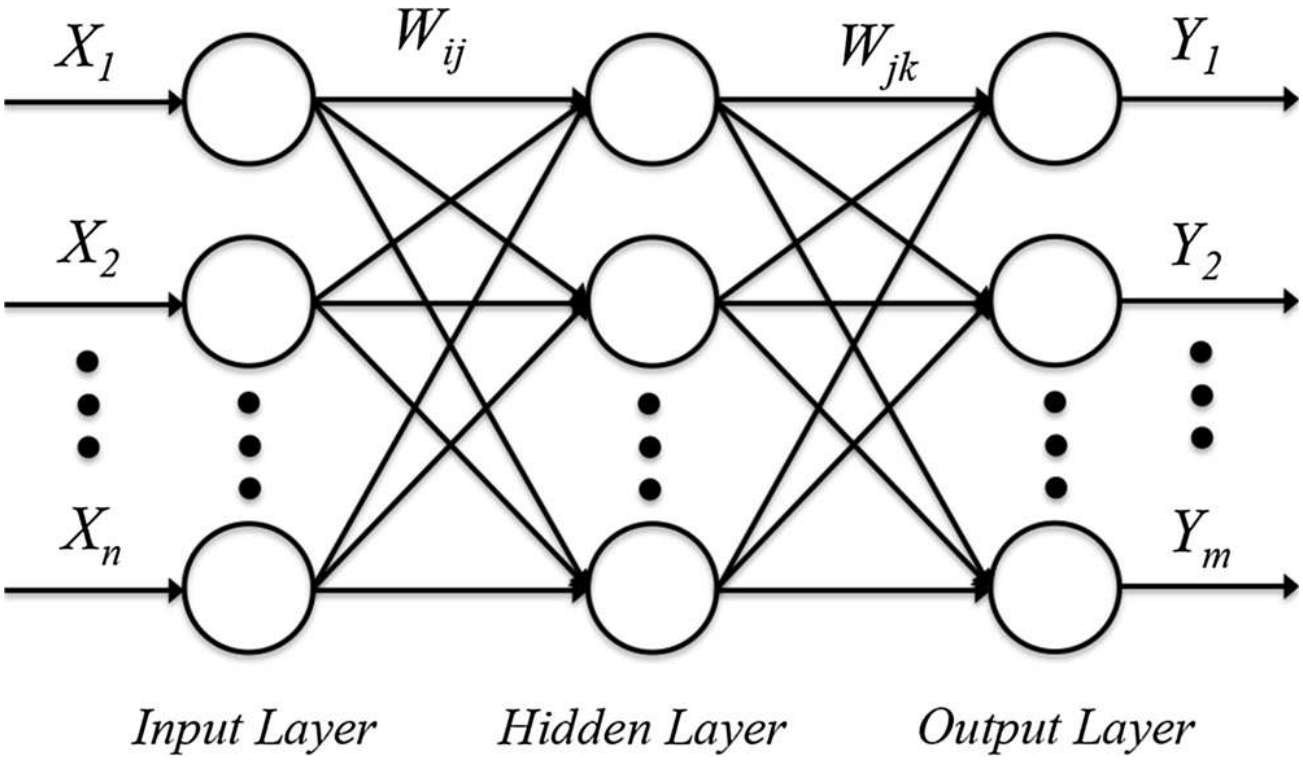

Fig. 2. Three-layer BP Neural Network Structure.

To develop an ANN model, it requires the selection of a network architecture based on the number of neurons, hidden layers, and activation function. It is an iterative process. Therefore, the effect of the increasing number of neurons and hidden layers on the model's performance is studied. Further simulations are carried out by varying these parameters in order to find the best ANN model.
The number of neurons in one hidden layer was varied from 2 to 50 to study the effect of neurons on the performance of the neural network. The logistic sigmoid function was used as the activation function. The trend shows that a large number of neurons in the hidden layer produce accurate results. The network could be trained further by adjusting hidden layers, neurons, and activation functions to improve training performance.

Hopfield network

The Hopfield model is a single layer recursive neural network, where the output of each neuron is connected to the input of every other neuron. In a Hopfield network all connective weight values are calculated initially from system data. Then as patterns or input values are applied, the network goes through a series of iterations until it stabilizes on a final output. It is important to ensure that the system will converge to a stable solution. This requires finding a bounded function (a Lyapunov or energy function) of the state variables such that all state changes result in a decrease in energy. Since the weight parameter vector is symmetric, the energy function of Hopfield neural network is defined as

$$\mathrm{E} = -\frac{1}{2}\sum_i \sum_j \mathrm{T}_{ij} V_i V_j - \sum_i \mathrm{I}_i V_i + \sum_i \theta_i V_i \qquad (1)$$

where V_i, is output value of neuron i, I_i is external input to neuron i, and θ is threshold bias.

The dynamics of the neurons is defined by

$$\frac{dU_i}{dt} = \sum_j \mathrm{T}_{ij} V_j + \mathrm{I}_i \qquad (2)$$

where U_i is the total input to neuron i and the sigmoidal function can be defined as

$$U_i(n) - U_i(n-1) = \sum_j \mathrm{T}_{ij} V_j(n) + \mathrm{I}_i \qquad (3)$$

$$V_i = g_i(\lambda U_i) = g_i\left(\frac{U_i}{U_0}\right) = \frac{1}{1+\exp\left(-\frac{U_i+\theta_i}{U_0}\right)} \qquad (4)$$

$$V_i(n+1) = g_i(\lambda U_i) = g_i\left(\frac{U_i}{U_0}\right) = \frac{1}{1+\exp\left(-\frac{U_i(n)+\theta_i}{U_0}\right)} \qquad (5)$$

U_0 is a coefficient that determines the shape of the sigmoidal function. g_i is the input-output function of the neuron i . λ is the gain parameter.

Stability is known to be guaranteed since the energy function is bounded and its increment is found to be nonpositive as

$$\frac{dE}{dt} = -\sum_i g_i'(U_i)\left(\frac{dU_i}{dt}\right)^2 \qquad (6)$$

Since $g_i(U_i)$ is a monotone increasing function each term in this sum is nonnegative. Therefore equation (6) is less than zero.

Competitive neural networks

Competitive learning models, one of the main classes of unsupervised neural networks, are widely applied to pattern recognition and data mining tasks, such as clustering and vector quantization.

Contrary to supervised training, since it does not require a desired or waited output for a given input pattern, such an unsupervised learning is obtained just presenting inputs to the network, which updates its parameters while in use. The neurons compete among themselves in the learning (training) of their synapses (weights) showing a competitive structure. The best neuron related to a given multi-valued input is activated as the winner, such that the neurons fight for training their own weights.

Essential to competitive learning is the concept of *winning neuron*, defined as the neuron whose weight vector is the closest to the current input vector. During training, the weight vectors of the winning neurons are modified incrementally in order to extract *average features* from the set of input patterns.

Neural Network Basic Structure

A neural processing unit (neuron) receives the input values and generates an output. Each output of the neuron in the input layer is connected to all the neurons in the output layer. The dimension of the input space is associated to each output neuron. The training is given by the neighboring between the patterns and the weight of the neurons. This competitive characteristic gives the neural network also the classification of "winner-takes-all" neural network. For each input presented to the competitive neural network, only one neuron is chosen to be trained. The winner neuron assumes this condition because it is the closest to the input clustering computed and grouped (pattern) from previous inputs. This approach adapts their weights in such a way to follow data distributed in groups i.e., clusters with similar features being able to identify and classify input patterns in classes using the geometric proximity.

Unsupervised Learning Mechanism

The learning mechanism detects groups by the geometric closeness of similar characteristics of input data, in which the input, x, corresponds to an output in such a way that only one neuron is active. The winner neuron, the best to be trained, is, then stepped forward to a recognized pattern and grouped according to previous inputs.

At each input data set presented to the algorithm the winner neuron approaches to the group. This neuron is labeled winner and assumes this condition because it is the closest of the pattern computed and grouped from previous inputs. The proximity of the neurons and the pattern, is determined by the Euclidean distance between the input vector, x, and the weights, w_j.

$$D = \left\| x - w_j \right\| = \left[\sum_{i=1}^{n} \left(x_i - w_{ij} \right)^2 \right]^{\frac{1}{2}} \quad (1)$$

where "j" is the number of output neurons, and "i" is the number of elements in the input vector x. This may be used with any dimensional problem and the states of the system of interest are represented as $x^k = \left[x_1^k, x_2^k, ..., x_n^k \right]$ that represent the n clusters, in such a way that k is related to the amount of elements of the input pattern. The number of clusters or classes, n, is assumed to be

previously known. The winner neuron, x_j, is the one that is closest of the vector x. Thus, the best neuron is the one who presents the smaller Euclidean norm.

$$x_j = \min_j \left\| x - w_j \right\| \qquad (2)$$

The learning mechanism is described by the similarity matching:

$$\left\| x - \hat{w}_j \right\| = \min_{1 \le i \le n} \left\{ \left\| x - \hat{w}_i \right\| \right\} \qquad (3)$$

The updating stage in which the competitive neural network teaches the winner neuron by driving it closer to the inputs is:

$$\begin{cases} \hat{w}_{j(new)} = \hat{w}_{j(old)} + \beta \left(x - \hat{w}_{j(old)} \right) \\ \hat{w}_{i(new)} = \hat{w}_{i(old)} \text{ , } i = 1, 2, ..., n \text{ , } i \neq j \end{cases} \qquad (4)$$

where the j th winner neuron is stepped forward while the loser neurons remain the same, β is the learning rate i.e., the step of training of the neural network, and $\hat{w}_j$ is the normalized vector:

$$\hat{w}_j = \frac{w_j}{\left\| w_j \right\|} \qquad (5)$$

With the input data, the neural network trains weights of the winner neuron w_i approximating it from the inputs when using Eq. (4).Geometrically, the neuron approximates to the desired cluster to be pursued.

The step size is determined according to the learning rate value and the distance (difference) of the target and the neuron (4). When it is high, the algorithm fast approximates to the target, but may never really converge to it, i.e., mostly the neuron will jump the center of the area of the target and oscillations may occur. On the other hand, if it is small the algorithm may never reach the target, i.e., the number of inputs concerning the target may not be enough when compared to the step size. An alternative is to employ an adaptive learning rate where the step size is also computed in function of the error obtained from the difference of the target and the current neuron. Another option is to determine a medium step size value.