Chapter 4
Artificial Neural Network (ANNs)

This chapter presents a review of the major forms of the Artificial Neural Networks (ANNs) (Sordo 2002). The particular topic of discussion of this chapter is how learning takes place in these models. Different ways of training the networks are examined.

- Background of the ANNs, its structure and applications
- Kohonen Self organizing maps and Hop-field networks
- Historical perspective of ANNs and its Evolution
- Applications and importance of Computational Development in the field of ANNs
- Catastrophic Forgetting
- Conclusion and summary of the relevance to the CGPDN.

1 Artificial Neural Network

The computational systems made up of interconnected neurons are termed as Artificial Neural Networks (ANNs). The properties of these neurons resemble those of the biological neurons. They can exhibit complex global behaviour which is dependent upon the interconnection of neurons, their internal parameters and their functions. These artificial neurons are bound together through different connections. The seamless transmission of signals from one neuron to another neuron takes place through these connections.

G.M. Khan, *Evolution of Artificial Neural Development*, Studies in Computational Intelligence 725, https://doi.org/10.1007/978-3-319-67466-7_4

1.1 Applications of ANN

ANNs are used in different real life applications such as function approximation, time series prediction, classification, sequence recognition, data processing, filtering, clustering, blind signal separation, compression, system identification and control, pattern recognition, medical diagnosis, financial applications, data mining, visualization and email spam filtering (Dorffner and Porenta 1994; Dorffner 1996; Sjoberg et al. 1994; Timothy 1994; Murray 1993; Ripley 1996).

1.2 History of ANN

The first generation of Artificial Neural Networks were based on the McCulloch-Pitts threshold neurons, which generated binary outputs (McCulloch and Pitts 1943). If the weighted sum of the inputs is above the threshold value, the unit was taken as 'on'; else the unit was taken as 'off'. The nature of inputs is either decimal or floating point numbers. The output of these neurons is only digital, but they have been successfully applied in artificial neural networks. The second generation Neurons utilize a continuous activation function for calculating their output. It makes them suitable for analogue input and output. Some of the frequently employed activation functions are sigmoid, and hyperbolic tangent. The second generation neurons are regarded as stronger than the first generation neurons. If the output layer of the second generation uses first generation binary units, they can be used for digital computations with few neurons in comparison to a network consisting of only the first generation units. They can also be used to approximate any analogue function which makes these networks universal for analogue computation (Maass et al. 1991). The continuous output of second generation ANN units can be interpreted in terms of a firing rate model. This value indicates the normalized firing rate of the unit in response to a particular input pattern. That is why second generation neuron models are considered a close approximation to the biological neurons and they are also more powerful than the neuron models of the first generation (DasGupta and Schnitger 1992).

The third generation of neural networks generate individual output spikes; hence they are closer to biological neurons (Ferster and Spruston 1995). The outputs can be interpreted using pulse coding mechanisms. The neurons send and receive individual pulses. The third generation networks are sometimes termed as Spiking Neural Network (SNN) (Gerstner and Kistler 2002) as explained in the next subsection. A wider range of neural coding mechanisms are entertained such as pulse coding, rate coding and mixtures of the two (Gerstner et al. 1999).

The recent experimental results have shown that the neurons of the cortex can carry out analogue computations at a very high speed. It has also been shown that the human's analysis and classification of visual inputs take place under 100 ms (Thorpe et al. 2001). At least 10 synaptic steps are required from the retina to temporal lobe, thus leaving 10 ms of processing time per neuron. This time is considered to be too

short for averaging mechanisms like rate coding for processing the information. So whenever speed is an issue, pulse coding schemes are thought to be the best (Gerstner et al. 1999; Thorpe et al. 2001).

1.3 Spiking Neural Networks (SNN)

The interaction between the biological neurons take place through a short pulse called action potential or spikes (Gerstner and Kistler 2002). Recently, researchers have shown that neuron can encode information in timing of these spikes instead of average firing frequency. The implementation of these SNN models takes place on this principal. Both in conventional ANNs and Spiking Neural Networks (SNNs), the information is usually distributed in the weight matrix. The interval between the time of spike of post synaptic neuron and pre synaptic neuron is used to adjust the weights in the SNN. The processing of a rapid temporally changing stimulus, which cannot be reproduced by having more neurons or connections, is only possible through the synaptic plasticity (Mehrtash et al. 2003).

1.4 Mode of Operation

The artificial networks can operate either in a learning (training) or testing mode. Once the learning starts, from a random set of parameters, the weights and thresholds are continuously updated until the desired solution is obtained; the parameters are then frozen and remain fixed during the testing process. During the adaptive process of learning, the weights between all the interconnected neurons are updated until an optimum point is attained. The weights of the network can be either floating point numbers or parameter dependent functions.

1.5 Learning Rules

The methods used for adjusting certain quantities responsible for the learnt information, typically weights are termed as learning rules. Supervised and unsupervised learning are the two main mechanisms of learning. When a desired output result is used to guide the update in the neural parameters, it is termed as supervised learning. While in the later mechanism, the training of the network is entirely dependent upon the input data and there is no provision of the target results for updating the network parameters which can be used to extract features from the input data (Hinton et al. 2006).

Back-propagation and evolutionary methods are the two conventional learning methods. In the back-propagation, the output and the desired results are compared

with each other, and the error is reflected backward to update the weights of ANNs accordingly. In evolutionary methods, the weights of the best performing ANN are slightly changed (either through mutation or cross over) for the production of next set of weights. In this manner, the optimum performance weights are obtained. Back propagation is also used for multilayer perceptron having input layer, hidden layers and output layer. Cost function is the predefined error function which can be calculated by comparing the output with target in back propagation. The cost function is given by

$e = f(d_i - y_i)$

Where;

d_i = The desired value

y_i = The system output

e = error

In order to reduce the error, it is fed back such that the weight of each connection is adjusted in a direction that minimizes the overall error. The process is repeated for converging the network to a state of minimum possible error.

Gradient Descent is an optimization method used for adjusting the weights in a manner which reduces the net error. The error function is differentiated with respect to the network weights. On the basis of the results of differentiation, the weights are adjusted for reducing the error. Because of this reason, back propagation is applied to networks which have differentiable active function.

The units of the intermediate layer of the feed forward neural network can be instructed through back propagation algorithm. The features of the input vector for predicting the desired output are represented by these units (Rumelhart et al. 1986). This training can be performed through the provision of information regarding the discrepancy between the actual output and the desired output of the network in order to customize the connection weights to reduce the discrepancy.

2 Types of Neural Networks

Different types of neural networks have been introduced over the years, but the most common one are feed-forward, Kohonen Self organizing maps, and Hopfield neural networks.

2.1 Feed Forward Neural Network

These networks are usually arranged in the form of layers where each layer has a number of neurons as the processing units (Sordo 2002). Signals are transferred from layer to layer through the input-output manner, where signals are processed at each layer and transferred in the forward direction. This basic architecture of a traditional ANN is called Multilayer Perceptron (MLP). Figure 1 presents a sketch of a general

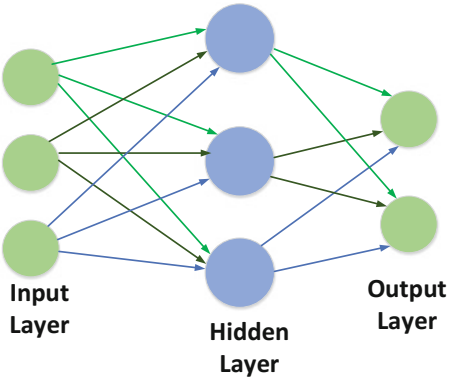

Fig. 1 Multilayer perceptron

model of Multi-layer perceptron consisting of an input layer, hidden layer and an output layer.

There are usually two layers of processing elements and a hidden layer in the MLP networks (as shown in Fig. 1), however the number of hidden layers can vary. The external signal arrives at the input layer which is then propagated by the input layer to the next hidden layer as a weighted sum. The hidden layer processes it through the activation function. The commonly used activation functions are hyperbolic tangent, the value of which ranges from −1 to 1;

$$\phi(x_i) = \tanh(x_i)$$

and sigmoid function with values range from 0 to 1;

$$\phi(x_i) = (1 + e^{-x_i})^{-1}$$

x_i is the received weighted and summed up signal from the input layer.

The job of the hidden layer is to transfer the processed signal to the neurons of another hidden layer, and if it is the last hidden layer; then it transfers the processed signal to the output. The signals reach output as the weighted sum, processed through the activation function. The output of the network is taken from the last layer.

The training of MLP networks is carried out by altering their connection weights after every processing interval. The variation in the weights is dependent upon the error between the output and the desired value. Usually this is done through back propagation. The error *(e)* in output node *j* in the *nth* data point is given by;

$$e_j(n) = d_j(n) - y_j(n)$$

where;

d = target value

y = value produced by the perceptron (Haykin 1998).

The error can be used to adjust the weights of the nodes in a manner that the energy $\mathcal{E}$ of error in the entire output is minimized as given by:

$$\mathcal{E}(n) = \frac{1}{2}\sum_j e_j^2(n).$$

2.2 *Kohonen Self Organizing Neural Networks*

The Self Organizing Maps (SOMs) which are used as computational methods for the visualization and analysis of high dimensional data were introduced by Teuvo Kohonen (Kohonen 1982, 2001). The maps are based on unsupervised competitive learning whose source of inspiration is the biological structure of the cortex. Cortex has different areas which are responsible for different human activities (motor, sensory, visual and somatosensory). Every sensory area is mapped to the corresponding area in the cerebral cortex. It is thought that the cortex contains the self-organizing computational map of the body. The sensory cortex also preserves the spatial relations between the body parts as much as possible. The same phenomenon also occurs in the motor cortex.

The self-organizing networks have a two layer topology (as shown in Fig. 2). The first layer is the Input layer while the second one is the Kohonen Layer. There is a node for each dimension of the input in the input layer where every input is connected to all the nodes in the Kohonen layer hence the two layers are fully connected. The node value in the Kohonen layer represents the output. The number of nodes in the Kohonen or output layer must be at least equal to the number of categories to be recognized. One neuron in the output layer has to be activated for every dimension of the input. The Kohonen layer neurons are neighboured by the grid (Kohonen and Somervuo 2002; Kaski et al. 1998; Martinetz et al. 1993). These networks are of great importance in applications, such as data clustering which occurs in speech recognition and handwriting recognition for sparsely distributed data. Lateral inhibitions are used by them which are inspired by the vision system working in biological neural systems.

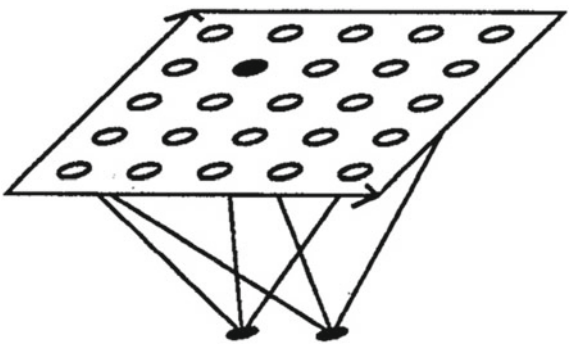

Fig. 2 Structure of Kohonen Self Organizing map, showing input neurons and the kohonen layer neurons. Input neurons are fully connected with the kohonen layer neurons, A winning neuron represented by a black dot. Taken from (Hertz et al. 1991)

Kohonen networks rely on the principal of mapping input vectors (pattern) of arbitrary dimension onto the Kohonen network in a way where the sequences closer to each other in the input space should be within close range. The training in Kohonen network begins with the fairly large sized neighbourhood of the winner. As the training proceeds, the distance reduces. The unit whose weight vector has the shortest Euclidean distance from the input sequence is the winning output unit. The neighbourhood of a unit consists of all the units which lie in its proximity on the map (not in the weight space). In the process of training, the closely distant node is selected along with its neighbour's weight; the modification for increasing the similarity with input takes place. The radius of neighbourhood decreases with the passage of time, and finally only a specific area in the network is identified for an input pattern is left. The following equation is used for updating the weights of the winning unit along with its neighbourhood.

$$w_i = w_i + \alpha(x_i - w_i)$$

where;

w_i = The weight of ith unit

x_i = The input

α = The Kohonen's rule for adjustment of weights.

In order to model the directional motion in the visual cortex, Farkas and Miikkulainen used SOMs (Farkas 1999). Their neuron model has 'leaky integrators' at synapses. It performs time-dependent summation with decay of incoming spikes. Once the dynamic threshold is exceeded, then a spike is fired. The spikes decay exponentially with time and are accumulated over a set of afferent and lateral inputs. The weighted output from leaky integrator is then applied to the spike generator. The spike generator will generate a spike only if the input threshold is exceeded. The output spike is then applied again for increasing the threshold, which makes it less likely to produce the second spike. There is an exponential decrease in the threshold with time. Every node has the receptive field of the receptors in the retina. They are weighted and integrated over time for creating a Hebbian type weight adjustment.

2.3 Hopfield Networks

A recurrent neural network is known as Hopfield Network (Hopfield 1982). Recurrent networks possess the property of bi-directional flow of information i.e. forward and backward direction. The nodes in such networks are fully connected to each other and they can function as both the input and output. The idea behind it is that the instability of states is iterated until a stable state is attained. This guarantees the convergence of the dynamics (Fig. 3).

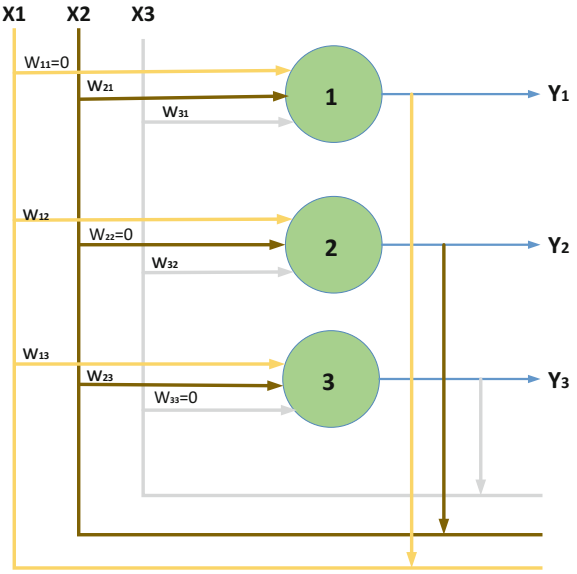

Fig. 3 Hopfield Network: Three node Hopfield network, with x_i = Input, y_i = Output, and w_{ij} = Weights attached to connections

The processing units which are used are binary threshold units. The binary threshold units only take two different values for their state. The values can be either -1 and 1, or 1 and 0. The two possible definitions for the activation y_i of unit i's are

$$y_i \leftarrow \begin{cases} 1 & \text{if } \sum_j w_{ij} x_j > \theta_i, \\ -1 & \text{otherwise.} \end{cases} \tag{1}$$

$$y_i \leftarrow \begin{cases} 1 \text{ if } \sum_j w_{ij} x_j > \theta_i, \\ 0 \quad \text{otherwise.} \end{cases} \tag{2}$$

where;

w_{ij} = weight of the connection
x_j = The state of unit j
θ_i = The threshold of unit i
There are two main restrictions in the connections of Hopfield net.

- No unit must be connected with itself
- Connections are symmetric

There is an energy function associated with every state of the network in Hopfield net given by the following equation:

$$E = -\frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j + \sum_i \theta_i \, x_i$$

where;
E = Energy of the network state.
w_{ij} = The weight of connection.
x_i = The state of unit i
x_j = The state of unit j
θ_i = The threshold of unit i

With update in the network, there is decrease in the energy till a minimum is reached. In Hopfield networks, the energy of the states is lowered in the training phase. However, the network keeps track of its energy state (Hopfield and Tank 1985). The network can approach a previous state if it is granted only a portion of that state, hence working as a content addressable memory system. We can also recover a distorted input from the trained state of the network. The input most similar to the distorted form is used as the recovery. As the memory recovery is based on the basis of similarity, therefore it is termed as the associative memory. As a result, Hopfield Networks are sometimes called associative networks as well.

Hopfield networks can be used in many optimization problems. The problem first has to be transformed into variables in a way that the desired optimization corresponds to the minimization of the respective energy function (Hopfield and Tank 1985). Hopfield networks can also be applied to the non-linear factorization problems (Husek et al. 2002).

In the next section, inclusion of artificial evolution into ANNs is described.

3 Neuro-Evolution

This section describes the Neuro-evolution (NE), which involves the use of artificial evolution with ANNs. Neuro-evolution refers to the evolution of various aspects of neural network. It is a combination of ANN and genetic algorithm, with ANN being the phenotype and genetic algorithm being the corresponding genotype. The genotype can represents the connection weights, connection type, node function, topology of ANNs or combination of any two, three or all the parameters. The genotype is evolved until desired phenotypic behaviour is achieved. Encoding is the important aspect of the NE system, since it affects the search space of the solutions (Yao 1999). Depending upon the methods, either the weights of the network or the topology or both are evolved. When a fixed topology network is used and only weights are evolved, the network solution space is constrained; it has to work in a restrictive environment not attaining any novel solution to the problem. It is not an easy job to select the proper topology of ANN for a specific problem. The Topology and Weight Evolving Neural Networks (TWEANNs) evolve both weights and network topologies. In this method, evolution is provided with flexibility for selecting the desired topology and weights for its network. So, TWEANNs genotype can encode both the topology and weight of the network. This increases the efficiency of the network, but it comes at the cost of increase in computational cost.

TWEANNs can also use both direct and indirect encoding methods of genotype. In direct encoding of genotype, every connection and node in the phenotype has to be specified in the genotype (Zhang and Muhlenbein 1993; Lee and Kim 1996; Dasgupta and McGregor 1992; Opitz and Shavlik 1997; Yao and Liu 1996; Angeline et al. 1993; Maniezzo 1994). In indirect encoding, only the rules for constructing the phenotype are specified in the genotype (Bongard and Pfeifer 2001; Gruau et al. 1996; Hornby and Pollack 2002; Mandischer 1993). The genotype doesn't specify every node and connection in the phenotype in the indirect encoding. TWEANNs which utilize the indirect encoding use a developmental approach that is akin to an artificial embrogeny (AE) (Stanley and Miikkulainen 2003) in which the small phenotypical structures act as the starting point which are developed to produce the final phenotype.

ENZO (Evolver and Network optimizer) is a system which can optimize both topology and connections' weights at the same time. ENZO uses direct encoding scheme (Braun and Weisbrod 1993). The set of the possible connections is fixed as the gene corresponds to a connection in the network. ENZO scheme provide introduction of new combinations of the parental properties through merging the parent's genes which is done through the crossover with the connection specific distance coefficients. This increases the rate of the learning process by inheriting the knowledge from parents, which is termed as weight transmission. Pujol and Poli evolved weight, topology and activation functions of ANNs through genetic programming (Pujol and Poli 1997). Since pole balancing is a standard issue in the design of control systems, they tested the system for the development of a neural controller for a pole balancing problem; and obtained promising results.

Krishnan presented a method which could evolve the rules for changing the network weights, instead of the weights itself (Krishnan and Ciesielski 1994). Krishnan used an indirect encoding scheme where the gene represented a rule for changing the weights. They also applied the mutation and crossover operation of a standard genetic algorithm to genes until they obtained the desired weight adjustment function. This network was called the 2-Delta GANN (Whitley and Hanson 1989). This network performed better than the back propagation technique for the benchmark problems. For smaller problems, the back propagation technique was more effective, however according to the author; 2-Delta GANN was effective in solving problems which were known to be very difficult for BP. This technique also provided better results than other GANN which directly encoded the neural network weights in the chromosomes.

Yao explored all combinations of ANN parameters including: connections weights, architectures, learning rules and input features (Yao 1999). Yao explored evolving the neural architecture and found that evolution can find a near optimal ANN architecture automatically. Yao also evaluated direct and indirect genetic encoding scheme, concluding that direct encoding scheme is good at fine tuning and generating a compact architecture, while the indirect genetic encoding is superior for finding a particular type of ANN architecture quickly. He also explored various combinations of ANN parameter for evolution and concluded that evolving both ANN architectures and connection weights can produce better results.

Stanley presented a new type of TWEANN, the Neuro-Evolution of Augmenting Techniques (NEAT) (Stanley and Miikkulainen 2002). He identified three major challenges of TWEANNs and introduced solutions for them. His solutions include: "tracking genes with historical markings for easy crossover between different topologies", "innovative protection through speciation", and "starting from a minimal structure and making it complex with the passage of the generations". NEAT performed faster than many other neuro-evolutionary techniques. The complexity of NEAT network continue to grow during evolution. It starts with a very simple structure with no hidden neurons, and a simple feed-forward network of input and output neurons. During the course of evolution, the network continues to grow by addition of neurons to existing connections or by addition of a new connection between the unconnected neurons. NEAT doesn't involve the development of the neural network during the particular generation of evolution. It only updates its architecture from generation to generation. That is why NEAT is not a developmental model. The indirect method of NE is called the neural development, which will be discussed in the next section.

(Khan et al. 2013d; Khan and Zafari 2016) used CGP to introduce four different ways of evolving neural networks: Feed-forward CGP evolved ANN (FCGPANN) (Khan et al. 2013a), Recurrent CGP evolved ANN (RCGPANN) (Khan and Zafari 2016), Plastic CGP evolved ANN (PCGPANN) (Khan et al. 2013b), and Plastic Recurrent CGPANN (PRCGPANN).

In the first case, CGP is transformed to a feed-forward neural network by considering each node as a neuron, and providing each connection with a weight. The neurons of such a network are arranged in Cartesian format with rows and columns inspired by original CGP architecture, and later on restricted to a single row mostly giving the network an ability to create infinite graphs/topologies. Each neuron in the network can acquire connection from either a previous neuron or from the system input. Not all neurons are necessarily connected with each other or with system inputs, this provides the network with an ability to continuously evolve its complexity along with the weights. All the network parameters are represented by a string of numbers called genotype. The number of active neurons (connected from inputs to outputs), varies from generation to generation subject to the genotype selection. Output of any neuron or a system input can be a candidate for the system's output selection. The ultimate system functionality is identified by interconnecting neurons from output to input. FCGPANN was initially tested for its speed of learning, and evaluated against the previously introduced neuro-evolutionary techniques on benchmarks such as single and double Pole balancing (Khan et al. 2013d) showing superior performance in comparison to the previously introduced neuro-evolutionary techniques. FCGPANN is explored in a range of applications including: breast cancer detection, prediction of foreign currency exchange rates, load forecasting, internet multimedia traffic management, cloud resource estimation, solar irradiance prediction, wind power forecasting and arrhythmia detection (Nayab et al. 2013; Khan et al. 2013a, c; Arbab et al. 2014; Rehman et al. 2014a; Khan et al. 2014). FCGPANN outperformed all the previously introduced techniques as highlighted in the literature. The second type of CGPANN is the Recurrent CGPANN (RCGPANN). These networks are more suitable for modelling systems that are dynamic and nonlinear. This

network is a modification to one of the earliest networks, the Jordan's network (Jordan 1986). In the Jordan's network there are state inputs that are equal in number to the outputs. These inputs are fed by the outputs through unit weights. The state inputs are present only at the input layer. In RCGPANN unlike the Jordan's network the state inputs can be connected, not necessarily to the first layer but to any layer. RCGPANN was also tested initially for its speed of learning similar to FCGPANN on both single and double pole balancing for both Markovian and non-Markovian cases. Its performance relative to other neuro-evolutionary techniques was superior. RCGPANN has been successfully applied to a number of applications including: Load forecasting, foreign currency exchange rates, bandwidth management and estimation (Khan and Zafari 2016; Khan et al. 2013c; Rehman et al. 2014b; Khan et al. 2013a) performing better than the previous neuro-evolutionary techniques.

Plasticity in neural networks has been the characteristic of choice when it comes to applications in dynamic systems due to its comparatively better performance (Papadrakakis et al. 1996; Sadeghi 2000; Carpenter and Grossberg 1988). The improved performance in Plastic neural networks can be attributed to the adaptability of its morphology to environmental stimuli. This is similar to the natural neural system. Plastic CGPANN has also been successfully applied to evolve a dynamic and robust computational model for efficiently predicting daily foreign currency exchange rates in advance based on past data (Khan et al. 2013b).

Plastic Recurrent Cartesian Genetic Programming Evolved Artificial Neural Network (PRCGPANN) is an online learning approach that incorporates developmental plasticity in Recurrent Neural Networks. Recurrent Neural Networks can compute random sequences of inputs due to their capability to acquire internal memory access. In a Plastic RCGPANN the output gene not only forms the system output but also plays a role in the developmental decision.

The research in artificial neural development is discussed in the next section.

4 Neural Development

The motivation behind the artificial neural networks was to replicate the computational models of the nervous system. ANN models mostly overlook the aspect that neurons present in the nervous system are part of the phenotype originated from the genotype through developmental procedure. Most of the aspects of the nervous system are determined from the information specified in the genotype (Kumar 2003). The genotype lays down the regulations for the development of the nervous system. The natural organisms have both the nervous system and genetic information stored in the nucleus of their cells (genotype).

The motive behind the development schemes is to increase the scalability of the ANNs, which is possible by having a minimum number of genes that can define the properties of the network instead of having a one to one relationship between the phenotype and genotype. These gene groups can influence several unrelated phenotypic traits with no dependency of the genotypic dimension on the phenotypic

size. For example, there is a common estimation of 30–40 thousand genes in the human genotype (45 million DNA bases out of a total 10^9) while a mature phenotype consists of 10^{14} cells (Elliot and Elliot 2001; Lodish et al. 2003).

According to Parisi and Nolfi, the neural networks should be considered along with the genotypes to be viewed in biological context, as part of a population and inherited by the offspring from parents (Parisi 1997; Parisi and Nolfi 2001). Parisi and Nolfi utilized a growing encoding scheme (Nolfi et al. 1994; Nolfi and Parisi 1995) for evolving the architecture and the connection strength of the neural networks for controlling a small mobile robot (for a similar method see (Husbands et al. 1994)). The network comprises of a 2-D space having a group of the artificial neurons distributed with growing and branching axons. The genetic code provides the instruction for growth of the axons and the branching of neurons.

A neural development model, which starts with the single cell that undergoes the process of cell division and migration, was proposed by Cangelosi (Cangelosi et al. 1994). Every cell produces two daughter cells where the new cells are separated in a 2-dimensional space. This process of cell division and migration continues until a group of neurons which are arranged in a 2D space is produced. Finally, the neurons grow their axons to produce connections among each other. This process keeps going on until a neural network is developed. The rules for the cell division and migration are present in the genotype (for a related approach see (Dalaert and Beer 1994)).

Gruau also proposed a similar method (Gruau 1994). A single cell goes through various stages of cell division and differentiation until the development of a complete neural network. Every cell is divided into two daughter cells. The old connections are strengthened along with establishing new connections. The rules related to cell division and transformation lie in the genotype. The genotype of Gruau's model is similar to the binary tree structure of GP (Koza 1992). The top node of the genotype tree is the initial cell. Every node of the genotype in the tree encodes the operation of that cell, while the sub trees specify the operation which should be applied to the two daughter cells. As a result of following the tree using instructions in these cells, the neural network is developed.

As a result of further work done by Gruau, a method which was based on the genotype-phenotype mapping that allows the repetition of phenotypical structure by re-using the same genetic information was introduced. In this case, the terminal cell or nodes point to the other trees. This encoding method can result in complex phenotypical networks from compact genotype. Gruau termed this method as an "automatic definition of neural sub-networks (ADNS)" (Gruau 1994).

For evolving the parameters which grow into artificial neurons with bio-inspired morphology, Rust and Adams used a developmental model combined with a genetic algorithm. Although Rust and Adams were able to produce morphologies of neurons, they did not apply it to substantive problems (Rust et al. 2000; Rust and Adams 1999).

For dynamic neural growth mechanism in cognitive development, Quartz and Sejnowski provided a powerful manifesto (Quartz and Sejnowski 1997). Marcus also laid emphasis on the importance of the growing neural structures by using a developmental approach. In his words "I want to build a neural networks that

grow, networks that show a good degree of self-organization even in the absence of experience" (Marcus 2001).

Jakobi developed an impressive artificial genomic regulatory network where the genes coded for proteins and the proteins either activated or suppressed the genes (Jakobi 1995). Jakobi defined the neurons, which had excitatory or inhibitory dendrites, through proteins. The individual cells divided and moved due to the interaction of the protein with the artificial genome. This resulted in the development of a multicellular system. After differentiation, every cell grew dendrites following the chemical sensitive development cones in-order to connect the cells. This resulted in a recurrent ANN capable of controlling a simulated Khepera robot for avoiding obstacles and navigation through corridors. The genotypes of every generation developed phenotypical structures, which were tested and the best one were chosen for breeding. Artificial evolutionary operations like cross over and mutation are utilized for creating offspring genotypes.

Various researches have studied the potential of Lindenmeyer Systems in developing ANNs and generative design (Lindenmeyer 1968). Boers and Kuiper adapted the L-systems for developing the architecture of the artificial neural networks (ANNs) i.e. a number of neurons and their interconnections (Boers and Kuiper 1992). A feed forward neural network was generated by evolving the rules of an L-System. They came to the conclusion that this methodology resulted in more modular neural networks, that performed better than the networks with the pre-defined structure.

Federici came up with an implicit encoding procedure for the development of the neuro-controller (Federici 2005). He also compared it with the direct scheme. He used adaptive rules relied on correlation between the post synaptic electrical activity and the local concentration of the synaptic activity and refractory chemicals.

Federici produced the neuro-controllers through two steps:

- He used a growth program in the genotype for developing the whole multi-cellular network in the form of the phenotype. This growth program inside every cell relies on local variables and is implemented by a simple recursive neural network which has a hidden layer (Similar to our use of CGP).
- During the second step, all the cells are translated into spiking neurons.

Every cell of the mature phenotype is a neuron of a spiking neuro-controller. The internal dynamics and synaptic properties of the corresponding neuron are specified by the type and metabolic concentrations of the cell. The topological properties of neurons such as its connections to the inputs, outputs and other neurons are produced by the position of the cell within organisms.

This network was implemented on a Khepera robot and the performance was tested both with direct and indirect coding schemes. Although the indirect method reached the high fitness faster, it had trouble in refining the final fitness value. Downing is in favour of a higher abstraction level in the neural development, because it avoids the complexities related with the axonal and dendritic growth. It also maintains the key aspects of the cell signalling, competition and cooperation of neural topologies in nature (Downing 2007). Downing also developed a system which he tested on a simple movement control problem called starfish. The task of the k-limbed animate

is to move as far as possible in a limited time from the starting point. This produced positive preliminary results.

The next section explains one of the major problems with ANNs known as the 'catastrophic forgetting'.

5 Catastrophic Forgetting

Catastrophic forgetting is one of the main issues with ANN. During catastrophic forgetting, the network forgets the previous task; once it is trained to do a new task. The short term memory in The Human brain can be regarded as a forgetting problem with the Biological brain, but evolution has minimized that over time (Seipone and Bullinaria 2005). The problem is more catastrophic in traditional ANNs and it is a serious limitation in such models (McCloskey and Cohen 1989; French 1999). There are many methods available for either reducing or eliminating this problem. One of the basic reasons behind the catastrophic forgetting is interference in the shared weights (McCloskey and Cohen 1989; Ratcliff 1990). There are many methods used for reducing this interference such as sharpening algorithm for reducing the hidden unit activation overlap (connection usage) and the HARM model (Sharkey and Sharkey 1995); that implements a lookup table and divides the main task into two sub-tasks (French 1999; Seipone and Bullinaria 2005). There are also certain methods which use dual additive weights where the fast weights learn new tasks and slow weights are used for long term (Hinton and Plaut 1987). A large number of the methods rely on dual model architectures which consist of two distinct networks for processing early and long term storage processing (French 1991). The inspiration behind these methods is that human brains do not suffer from catastrophic forgetting as their brains evolve two different areas i.e. hippocampal system for learning the new information, and neocortical system for slow and long term learning and problem solving.

Brain has the capability of retaining information; some of this information might degrade over time in a gradual manner. Connectionist networks which are trained with a particular set of patterns when presented with new input patterns with no correlation to the old pattern, they adapt to the new patterns and completely forget the previous patterns. Robert addresses the problem of catastrophic forgetting in connectionist networks; it's consequences by highlighting the possible reasons that cause this behaviour and possible solution to this problem. According to Robert (French 1994), the problem of catastrophic forgetting can be alleviated by having separate areas for information handling and processing; and for retention of processed information.

"Conservative Training" and "Support Vector Reversal (SVR)" are presented in (Albesano et al. 2006) as solutions to mitigate the effect of catastrophic forgetting in ANNs in the domain of automatic speech recognition. In conservative training instead of assigning a value of zero to the missing units, target uses the output of the original network as an objective. While in SVR, support vectors are used to define

the borders of the classes to keep the classification boundaries of the new network close to that of the originally trained network.

An Algorithm, elastic weight consolidation (EWC) inspired from the neurobiological model of synaptic consolidation; that is the mammalian brain is able to retain information as the excitatory synapses are strengthened. Continual learning is enabled by implementation of EWC which prevents the information of the previous task from being erased by reducing the plasticity of parameters from the previously learned task (Kirkpatric et al. 2017). Catastrophic interference can be seen in conjunction with a general dilemma coined by Grossberg (Grossberg 1980, 1982); the stability-plasticity dilemma which he published in his book in 1980. He states: "How can an organism's adaptive mechanisms be stable enough to resist environmental fluctuations which do not alter its behavioural success, but plastic enough to rapidly change in response to environmental demands that do alter its behavioural success".

Age limited learning effects are explained in the context of catastrophic forgetting by exploring the plasticity-stability dilemma in ANNs. In a parallel-distributed system, plasticity is essential for acquiring and incorporation of new information. Stability on the other hand is required to retain previously acquired knowledge. ANNs exhibit plasticity by readily adapting and learning new information at the cost of previously acquired knowledge (Mermillod et al. 2013). Human memory is emulated within a back propagation network by introducing grace degradation of information with the help of interleaved learning. Sparse encoding and activation function adjustment were also tested to assuage catastrophic interference. The results however revealed that they influenced the performance of the network but could not eradicate catastrophic forgetting in the network (Abdallah El Ali et al. 2008).

Robins' pseudo-rehearsal solution and French's activation sharpening algorithm were tested to overcome the problem of catastrophic interference with the former producing promising results. The solutions serve to reduce the catastrophic forgetting to some extent but fail a general solution to the problem (Ole-Marius et al. 2005). In Reinforcement learning (RL) problems, catastrophic forgetting can be prevented by avoiding overtraining and reasonably orthogonalising the input layer. To completely eliminate catastrophic forgetting into an RL agent, pseudo-rehearsal, a powerful continual learner can be adapted. Although, CHILD is a faster and more capable continual learner but due to its complex nature is difficult to execute (Cahill 2010).

These methods have reduced the catastrophic forgetting slightly; still the current models of ANNs cannot eliminate these problems. Although the ANN models have slightly adapted the biological neural structure, still they are not as complex as the biological neural systems. The biological neural systems can develop their own memory due to the changes in the synaptic connections, neural architecture, neurite growth, shrinkage and the variations of the chemical concentrations.

6 Conclusion

This chapter described various artificial neural networks, learning methods and their applications. Historical perspectives of evolutionary methods applied to ANNs were also elaborated. The chapter also presented a review of different methods for development of ANNs. The use of ANNs has spread to engineering and medical diagnostics. ANNs are the bio-inspired models of the brain. They have adopted some properties of biological neurons, but they are yet to match the complexity of the biological neurons.

The ANN models can perform efficiently in fixed task environment, however they seem to struggle with dynamically changing environment. As the learnt information in an ANN is encoded in the weights, retraining will cause the weights to change. This will affect the performance on the previous task. The performance of the network can also be affected; if the environmental conditions slightly change while the same task is being solved. Our network is yet to be tested on different task environments; however the weights and morphology of the network continue to develop during the task environment.

Our implemented system is inspired by the neuro-science. It also produced an artificial environment for the neurons. Our basic neuron model is based on biological study of neuron, their development and their mechanism of signal processing. The neurons can either grow more neurons or can die. They are able to produce complex neural structures based on the task requirement. We also evolved the rules for the model discussed in this book's development on the basis of neuro development techniques which were described earlier. Chalup proposed that an incremental scheme results in the development of the network in its stage of learning which would function more effectively than the artificially imposed inflexible system architecture (Chalup 2001). This argument supports the approach adopted by us.

The motivation for the model discussed is the work done on neuro-development techniques discussed in this chapter. The book evolves the rules for development of the neural architecture and their internal processing. It is evaluated on two learning environments i.e. the Wumpus world and the checkers, details are provided in Chaps. 6 and 7.

The next chapter will provide an insight into the design of the model along with biological inspiration in detail.