# A Review of Cascade Correlation Neural Network for Software Cost Estimation

Vinayak Suresh Dhole
ME Student
Department of Computer Engineering
R. C. Patel Institute of Technology
Shirpur, Maharashtra, India

Nitin N. Patil
Associate Professor,
Department of Computer Engineering
R. C. Patel Institute of Technology
Shirpur, Maharashtra, India

## ABSTRACT

Cascade-Correlation is a new architecture and supervised learning algorithm for artificial neural networks and classification techniques. Rather than adjusting the weights in a network of predefined topology, Cascade-Correlation begins with a minimal network and it automatically trains, adds new hidden units one after the other by creating a multi-layer structure. As soon as a new hidden unit has been added to the network, its input-side weights are getting fixed. After that these unit then becomes a permanent feature-identifier in the network, present for producing outputs, then cascade-correlation is behaves as more complex feature detectors. The Cascade-Correlation networks have several benefits over existing algorithms as it learns very fast. It determines its own size and topology fast. It maintains the structures which it has built even after the training set changes, and it doesn't need back-propagation of error signals through the connections of the network and its component. Cascade Correlation Neural Network (CCNN) types such as recurrent CCNN, evolving CCNN, genetic CCNN are used to predict software effort from Use Case diagrams in advance manner which helps further for software cost estimation. The use case diagrams are developed in the early stages of the software development and they are used for input. This paper is an overview of cascade-correlation neural networks in which we study different types of cascade-correlation neural network. They are based on a special architecture which autonomously adapts to the application and makes the training much more efficient than the widely used backpropagation algorithm. This review focuses on different types of CCNN and also describes the cascade-correlation architecture variants.

## General Terms

Cascade-correlation neural network, artificial neural network,

## Keywords

Neural network, cascade architecture, evolving.

## 1. INTRODUCTION

Artificial neural networks are universal function approximates which makes them suitable for the most applications. However, their training is usually cumbersome and requires proper tuning of the learning algorithm based on deep knowledge about the problem. Without this care their answer can converge very slowly resulting a never ending learning process. Furthermore, the widely used backpropagation learning algorithm requires the network structure to be provided. This structure has a serious impact on the learning capabilities, so it has to be designed properly for the application. Because of these reasons the constructive training algorithms have become appealing where the structure is adaptively built during the training process. The constructive methods have two main classes. One uses evolutionary algorithms to evolve the network structure by training and combining many networks at the same time. This approach has huge computational costs compared to backpropagation and so it is usually infeasible. The other is represented by cascade-correlation neural networks (CCNN). They have a special network architecture which autonomously adapts to the application. They also have a special training process which reduces the computational costs and cures many problems of the backpropagation algorithm at once. The rest of this section shortly covers ANNs in general and considers backpropagation training problems. This paper also investigates how the ANN training can be improved by curing the previously discussed problems and presents more CCNN variants. A Neural Network (NN) is an adaptive system that learns from examples using interconnected processing nodes. Artificial neural networks serve as general purpose mechanisms for training a machine by examples. This neural network is to be used for software cost estimation in recent ten decades [1]. Neural networks are classified as artificial intelligence because of their ability to learn and its basis in biological activities of the human brain. They are modeled after the human brain, which are perceived as highly connected network of neurons termed as nodes. It has three parts (layers): an input layer, a hidden layer and the output layer [2]. The number of input, hidden, and the output nodes is referred to as the neural network topology or the network architecture. Figure 1 below shows a network of nodes, left-hand nodes x1, x2 and x3 are the input node and constitute the input layer [3]. The input nodes represent the predictor variables. The two middle nodes z1 and z2 are the hidden nodes and constitute the hidden layer. The right-hand node (Y) is the output node and makes up the output layer. The output layer represents the target variable. The heart of the neural network algorithm involves series of mathematical operations that use the weights to compute a weighted sum of the inputs at each node [4]. For a particular example, the net input to a unit in a hidden or output layer is given by:

$$net_{ij} = \sum w_{ij} x_{ij} \quad (1)$$

Where $x_{ij}$ represents the input to node, represents the weight associated with the $i^{th}$ input to node and there are I+1 inputs to the node is the activation of unit j [5].

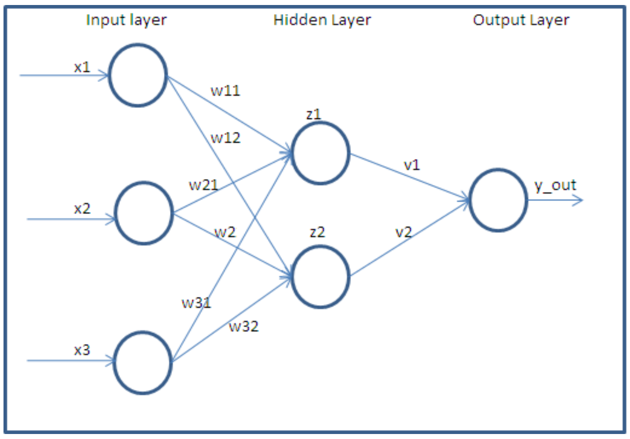

**Figure 1: Simple Neural Networks**

All Artificial neural networks (ANN) were inspired by the human brain. They contain simple processing elements (neurons) and connections among them which the data can “flow” on. Each neuron has many inputs, outputs and an activation function. The computational model of a neuron can be described by the

$$y = f\left( \sum_{k=1}^{n} \mathrm{wk} * \mathrm{xk}\right) \qquad (2)$$

equation where *y* is the output value transferred further by all the output connections, *xk* is the *k*th input, *wk* is the weight of the connection related to the *k*th input and *f* is the activation function which is usually the signum (*sign*) or the sigmoid (¾) function defined as

$$Sign(x) = 1 \text{ if } x > 0 \qquad (3)$$

A neuron can be efficiently trained by the perceptron learning algorithm to solve any linearly separable problem. However, most of the problems are not linearly separable, so an individual neuron cannot give an acceptable solution. For these problems the neurons have to be structured and connected to form a network ANNs with many neurons are usually organized into layers [6]. Each network has an input, an output and may have hidden layers. Figure 1 shows an ANN with an input, an output and two hidden layers. The input layer contains only special neurons with one input, many outputs and the identity of an activation function is with them, so these neurons were feed the data in a network [7]. The ANN of Figure 1 has only connections which go into subsequent layers. These kind of networks are called feed forward ANNs. A network with connections going to the same or to a previous layer is called recurrent ANN.

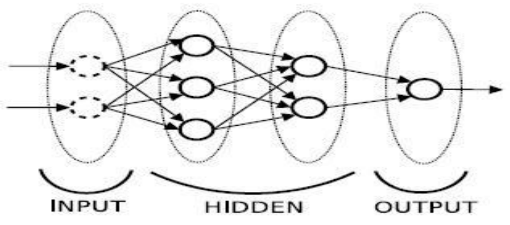

**Figure 2: Artificial Neural Network**

It is known that a feed forward ANN with one hidden layer is an universal function approximator [8], so it can approximate any bounded continuous function with arbitrary precision (however it needs exponentially many neurons for this). Furthermore, any feed forward ANN can be trained (in the supervised way) by the backpropagation algorithm. It calculates the error at the output and propagates it back to the neurons of the previous layer which can be trailed accordingly. Then the locally corrugated errors are propagated further to the previous layer and so forth until the input layer is reached. Technically this algorithm calculates the gradient of the network according to the connection weights. One of the most robust backpropagation variant, called quick prop, was published by Fahlman (1988) [9].

The cascade-correlation architecture, published by Fahlman and Lebiere (1990) [10], has two key ideas. First it grows the network on demand, so it only adds new neurons when they can help for solving the problem. Second the new neurons are added and trained one by one which can eliminate many of the problems presented in the previous section. At the beginning the learning algorithm starts with an “empty” network which has only the input and the output layers and does not have any hidden layers. Because of the absence of hidden neurons this network can be learned by a simple gradient descent algorithm applied for each output neuron individually [11]. During the learning process new neurons are added to the network one by one. Each of them is placed into a new hidden layer and connected to all the preceding input and hidden neurons. Once a neuron is finally added to the network (activated), its input connections become frozen and do not change anymore. The neuron-creation step can be divided into two parts. First a new, so called candidate neuron is connected to all the input and hidden neurons by trainable input connections, but its output is not connected to the network. Then the weights of the candidate neuron can be trained while all the other weights in the network are frozen. This state is illustrated on Figure 3, where the dashed connections are trained [12].

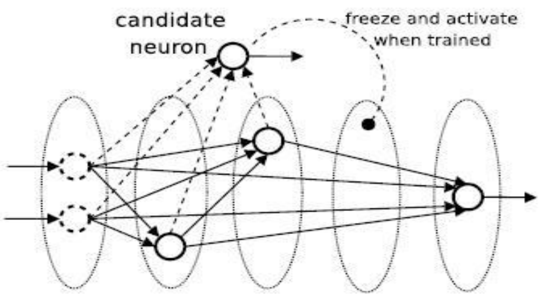

**Figure 3: The cascade-correlation architecture**

The empty layer shows the place of the candidate neuron where it is activated to after its input weights are learned and become frozen [13]. Second the candidate is connected to the output neurons (activated) and then all the output connections (all the input connections of any neuron in the output layer) are trained. The whole process is repeated until the desired network accuracy is obtained.

## 2. LITERATURE SURVEY

### 2.1 Cascade Correlation Neural Network

An important but difficult problem in neural network modeling is the selection of the appropriate number of hidden units. The *cascade correlation algorithm*, proposed by Fahlman and Lebiere, addresses this issue by recruiting new units according to the residual approximation error. The algorithm succeeds in giving structure to the network and reducing the training time necessary for a given task [14]. Figure 2 shows a network trained and structured using cascade correlation. Assume for the moment that a single output unit is required [15]. The algorithms start with zero hidden units and add one at a time according to the residual error. The output unit is trained to minimize the quadratic error. Training stops when the error has leveled off. If the average quadratic error is still greater than the desired upper bound, we must add a hidden unit and retrain the network. The motivation behind this step is to specialize on hidden unit to the detection of the residual error of the network [16]. The backpropagation algorithm is applied as usual and it will take care to compute with the sign of the argument with an absolute value operator. Another way of overcoming this drawback is to take an absolute values at all, is to train a weight at the output of the hidden unit, so that it assumes the appropriate sign (see Exercise 1). Once the hidden unit H1 has been trained, i.e., when the correlation level cannot be further improvised, the present unit is added to the network as figure shows in Figure 4 (left diagram). The weights of the hidden unit are frozen. The output unit receives information now from the input sites and from the hidden unit H1.

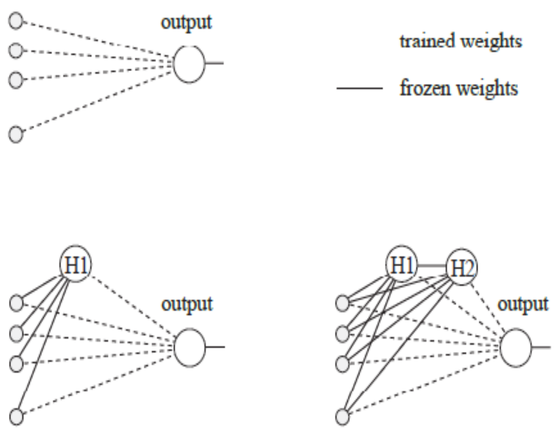

**Figure 4: The cascade correlation architecture**

All the weights of the output unit are retrained until the error levels off and we test if a new hidden unit is necessary. Any new hidden unit receives an input from the input sites and from all other previously defined hidden units. The algorithm continues adding hidden units until we are satisfied with the approximation error [17]. The advantage of the algorithm, regarding learning time is that on iteration of a single layer of weights has to be trained. Basically, we only train one sigmoidal unit in an iteration whenever they trained. The final network which we get has more structure than the usual flat feed-forward networks and, if training results correctly, we have to stop when the minimum number of necessary hidden units has been selected. To guarantee this, several hidden units can be trained at each step and the one with the highest correlation selected for inclusion in the network [18]. In the case of more than one output unit, the average of the correlation of all output units is maximized at each step. The previously discussed network structures have two important drawbacks. First of all, one of the network types has to be selected. After that, also the number of units in the hidden layer has to be chosen. Both choices are not trivial. A cascade-correlation approach solves this problem and makes a combination of the simple methods which are easy to train. The concept of original cascade-correlation algorithm [19] is to identify not only the weights, as well as the network structure of the network at the same time. This can be happen in a positive constructive way; by means of only one neuron at a time is trained and then added to the network structure. While each time only one starts with a network without any hidden unit, and then in a network hidden neurons which are needed are added, one by one, until some stopping criterion is occurred. As soon as a hidden neuron is being added to the network, its obtained weights are remains final throughout the rest of the procedure. This also means that, besides the actual input vector in a network, the output values of these current hidden units are to be used as extra inputs for any new hidden neuron in a network. At the output level, a linear combination function can be used effectively [20].

The assumption is that its correlation with the residual error will make a new neuron useful in reducing the residual error and improving the prediction of the actual target in a network. The increase values are done by computing the gradient of the network and performing some form of gradient ascent. Instead of training only one examine candidate neuron at a single time, a collection of neurons called pool is initialized with random values of weights, can be trained. At the end, the best one is selected. This increases the chance that a good candidate will be identified. As well as the best candidate is selected and added to the selected network, the output weights for the updated network can be updated automatically. Whether a linear function is used at the output level, the output weights can be obtained by simple linear regression. The concept of cascade-correlation networks can be extended to networks for learning aggregate functions. The crucial difference is that instead of the simple collection of hidden neurons, we can use units that can process bags are used effectively. These units come from the simple networks presented above. The *sym*, *hsum*, *hmax*, *hsmx* and *lrc* units can all be used as aggregation units in the hidden layer of the cascade-correlation network which to construct. For the remaining, the network and the training of network of it work in the same way as for the feedforward cascade-correlation networks are examined effectively [21]. A schema of an aggregate cascade-correlation network for 2 input vectors is shown in figure 1. With all parts of the aggregate cascade-correlation network explained, it only remains to discuss the training of the network in more detail. Each time a new unit should be added to the hidden layer to be selected and a collection of neurons called a pool of units is created of all possible types. Weights are initialized randomly. After that, all units in the pool are trained and examine for a number of iterations of same values, similar to backpropagation in a networks. This training is basically a gradient ascent, maximizing the correlation with the outputs. The computation of the gradient depends of course on the type of unit. As described, this method has the advantage that the step size is determined automatically and convergence is faster than for a finalized step size in a network. The basic concept behind it is to increase the step size and value when the sign of the gradient remains the same, and decrease the step size when the sign changes. When all units in the pool have been trained, the best one is chosen. In this case, the best unit is the one with the highest correlation. When the unit with the highest correlation has been chosen, it is admitted in the network and the obtained output weights have to be learned and examine again. As a linear activation functions in a network are used for the output, while the output weights can be identified with least squares linear regression. Ali Bou Nassif et al. proposed that Software cost estimation is a crucial element in project management. If we failed to use a proper cost estimation method might lead to project failures. If we conduct software cost estimation in the early stages of the software life cycle then it is important and this would be helpful to project [22].

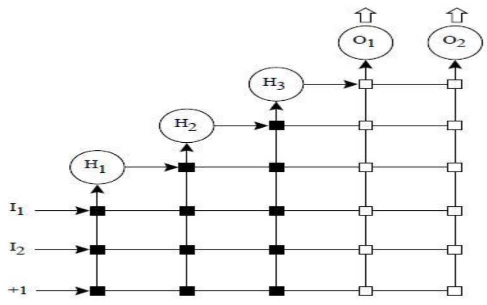

**Figure 5: Cascade-Correlation Network after the Addition of Three Hidden Units**

M. Mohammed et al. proposed that the performance and capacity of multicast routing protocols over ANN. They summarized traffic models for multicast routing protocols in ANN. They also evaluated the performance of the existing multicast protocols in ANN using similar traffic models to justify their proposal. Multicast routing protocols were categorized into tree-based, mesh based, stateless, hybrid-based and flooding protocols and proposed with a focus on how to rise above the constraints present in the previously proposed multicast protocols for better result in ANN.

M. Azzeh et al. proposed that Software effort estimation at early stage is a crucial task for project bedding and feasibility study. Since collected data at early stage of software development lifecycle is always imprecise and uncertain, it is very hard to deliver exact estimated value. In an analogy-based estimation, which is currently is the one of the popular estimation methods, is frequently used at early stage because of uncertainty associated with attribute measurement and data availability. In order to improve performance of analogy-based estimation at earl stage, using all available early data, we integrated it with Fuzzy numbers and new software project similarity measure and a new adaptation technique based on Fuzzy numbers in order to support analogy-based estimation at early stage of software development lifecycle [23]. In all data sets the empirical evaluations have shown that the proposed similarity measure and adaptation techniques method were able to remarkably improve the performance of analogy-based estimation at early stage of software development and in effort calculation. The results obtained are matches with proposed method outperforms some well know estimation techniques such as case-based reasoning and stepwise regression. It is concluded that the proposed estimation model could form a useful approach for early stage estimation especially when data is almost uncertain. Software effort estimation at early stage is a crucial task for project bedding and feasibility study. Since collected data at early stage of software development lifecycle is always imprecise and uncertain, it is very hard to deliver accurate estimate. Analogy-based estimation, which is one of the popular estimation methods, is rarely used at early stage because of uncertainty associated with attribute measurement and data availability. In order to improve performance of analogy-based estimation at early stage, using all available early data, we integrated it with ANN [24].

## 2.2 Recurrent CCNNs

When the order of samples has specific patterns, recurrent neural networks (RNN) can fit better for the problem than feedforward ones. RNNs have feedback connections going into the current or previous layers and so they can circulate data inside the network. Using these data they can simulate a short-term memory which is capable of recognizing input patterns and adapt the outputs accordingly. Unfortunately there is not any known efficient learning algorithm for general RNNs. Only a few special recurrent architecture have efficient learning methods including the recurrent cascade-correlation (RCC) architecture published by Fahlman (1991).

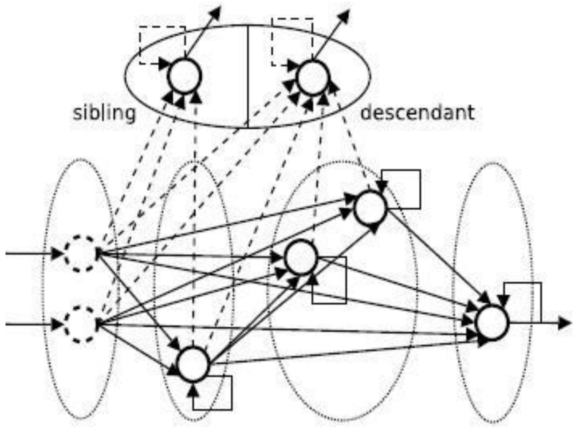

**Figure 6: Recurrent Cascade-Correlation Network**

Because the CCNN training does not change any weights of a frozen neuron, the new candidate neurons cannot have feedback connections going to activated neurons. So the only option, applied by the RCC architecture, when each neuron has a self-recurrent connection (its own output act as an input too) which can be trained during candidate learning and become frozen later [25].

## 2.3 Evolving Cascade Neural Networks

A new learning algorithm for Evolving Cascade Neural Networks (ECNNs) is described here. An ECNN starts to learn with one input node and then adding new inputs as well as new hidden neurons which is actually participates in it. The trained ECNN consist of less or min number of input along with hidden neurons along with present connections. The algorithms of ECNN were implemented successfully to classify artifacts and normal segments in clinical electroencephalograms (EEGs). The EEG segments were visually identified by EEG-viewer segments. The trained ECNN has correctly classified 96.69% of the testing segments. It is slightly better than a standard fully connected neural network. The training algorithm was applied to a real-world problem related to classification of normal segments and artifacts in the EEG recordings. The EEG segments are characterized by several noisy and unexpected irrelative features. The obtained result in the EEG recordings of two patients was visually labeled by obtained segments of EEG-viewer segments. The ECNN has been found that it learns to detect automatically classification of the EEG segments. The ECNN, trained on the EEG segments, has correctly classified 96.69% of the testing segments. A standard feed-forward network using a PCA preprocessing applied to the same datasets has provided 94.46% of correct classifications on the given records [26]. Hence, the ECNN algorithm implemented on the EEG problem efficiently that has performed slightly better than a standard neural network technique. Evolving Cascade Neural Networks (ECNNs) and a new training algorithm were capable of selecting informative features which elaborated widely in classification. The ECNN initially learns with one input node and then adding new inputs as well as new hidden neurons which is actually participates in it. The resultants ECNN were consist of less number of hidden neurons and inputs that was applied on ECNN efficiently.

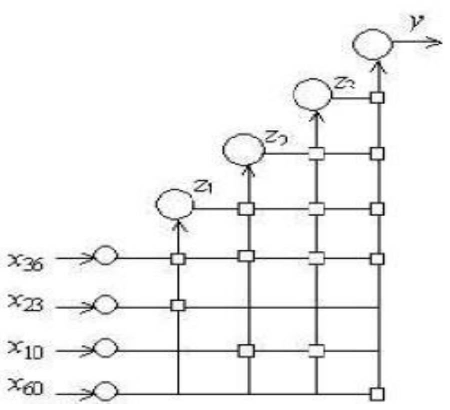

**Figure 7: The structure of an ECNN trained to recognize the EEG artifacts**

The ECNN starts to learn with one neuron, and new inputs as well as new neurons are added to the network while its performance increases. As a result, the ECNN has a near optimal complexity. The ECNN was applied for recognizing artifacts in the clinical EEGs recorded from newborns during sleep hours. The artifacts in these EEG were visually labeled by Reviewers. Some of the spectral and statistical features, calculated to present the EEGs for an automated recognition, were noisy and redundant. In our experiments with the artifact recognitions in the EEGs, the ECNNs, learnt from the data, has slightly outperformed the standard FNNs with a fixed structure. The ECNN has also outperformed the evolutionary GMDH-type as well as the standard decision tree techniques [27].

## 2.4 Genetic Cascade Neural Networks

A genetic algorithm has been applied within the Cascade-Correlation architecture to train connection weights given in a network. The performance of the Genetic Cascade-Correlation program when implemented on the two-spiral and eight-bit parity problems compares favorably with the original Cascade-Correlation program that used the Quickprop algorithm to train connection weights in a network. Though Quickprop solved both problems with less number of hidden units, the genetic algorithm required less epochs in the case of the two-spiral problem and only slightly more epochs than Quickprop in the case of the eight-bit parity problem in calculations. If the creation of compact widely used networks is expected, the genetic algorithm can be lower down to find solutions with less hidden units at the contribution of additional training epochs. The performance of the genetic algorithm is encouraging considering it uses no gradient information (thus doesnot require differentiable transfer functions) and depends entirely on generic recombination operators. The advantage of the Quickprop algorithm is that it is significantly more efficient than the genetic algorithm when gradient information is available. However, in applications such as non-linear neural control where an error gradient is sometimes not computable, genetic algorithms may provide a reasonable alternative. Future work will consist of applying the Genetic Cascade-Correlation algorithm to such an application [28]. A comparison of the genetic algorithm performance with and without the crossover operator did not demonstrate a statistically significant increase in performance when standard two-point crossover was applied. Still we can say, the earlier attempts to use genetic algorithms to train neural networks demonstrated the crossover operator significantly decreased the performance due to incompatible feature-detector mappings onto hidden units. The empirical results of this paper indicate that the application of crossover within the Cascade-Correlation architecture does not suffer from this defect which identified above. In future it may lead to alternate genetic representations within the Cascade-Correlation architecture that will enable the crossover operator to demonstrate a significant performance contribution. The results are inconclusive as to whether the genetic algorithm reduces the possibility of convergence to suboptimal local minimum solutions. Both Genetic Cascade-Correlations and Quickprop Cascade-Correlation consistently found acceptable solutions to the two-spiral and eight-bit parity problems [29].

## 3. CONCLUSION

In this paper we studied about cascade correlation neural networks which have an adaptively built architecture autonomously fitting to the target application. We covered different forms of cascade correlation networks which do not suffer from the common problems of the backpropagation algorithm. We considered many variants and showed important applications where these networks can bring significant improvements. Despite of the old age of cascade-correlation neural networks, we studied the recurrent cascade-correlation architecture. The study of evolving cascade-correlation networks were usually based on comparisons against fixed structure feed forward neural-networks trained by the backpropagation algorithm. Cascade correlation neural network also can be used in genetics for finding the optimal solution of complex problems. The evolving CCNN is another field where we can add more hidden units in CCNN with previous output as a input to the next level of evolving CCNN. The evolving CCNN can be used efficiently for complex problems. RNNs have feedback connections going into the current or previous layers and so they can circulate data inside the network. Using these data they can simulate a short-term memory which is capable of recognizing input patterns and adapt the outputs. In future all of the forms of the CCNN can be efficiently used for software cost estimation in software engineering, in operating system for avoiding deadlocks and in genetics as a basic fundamental unit for implementing genetic algorithms. In future we can use ECNN, RNN and genetic CNN can be used for software cost and effort estimation.

## 4. REFERENCES

[1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks Are Universal Approximators. Neural Networks Vol. 2, Issue 5, pages 359–366, 1989

[2] Scott E. Fahlman. Faster-Learning Variations on Back-Propagation: An Empirical Study. *Proceedings1988 Connectionist Models Summer School*, pages 38–51, 1988.

[3] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation Learning Architecture. *D. S. Touretzky (ed.), Advances in Neural Information Processing Systems 2*, pages 524–532, 1990.

[4] Jean-Philippe Thivierge, Fracois Rivest, and Thomas R. Schultz. A Dual-Phase Technique for Pruning Constructive Networks. *Proceedings of the International Joint Conference on Neural Networks*, 2003.

[5] Steffen Nissen. Large Scale Reinforcement Learning using Q-SARSA and Cascading Neural Networks. *MSc Thesis, University of Copenhagen*, 2007.

[6] Dale Schuurmans and Finnegan Southey. Metric-Based Methods for Adaptive Model Selection and Regularization. *Machine Learning, 48*, pages 51–84, 2002.

[7] Shumeet Baluja and Scott E. Fahlman. Reducing Network Depth in the Cascade-Correlation Network Architecture. *Technical Report CMU-CS-94-209*, 1994.

[8] Scott E. Fahlman. The Recurrent Cascade-Correlation Architecture. *D. S. Touretzky (ed.), Advances in Neural Information Processing Systems 3*, pages 190–196, 1991.

[9] Schetinin, V.: Polynomial neural networks for classifying EEG signals, In: *Proceedings of* NIMIA-SC2001 NATO Advanced Study Institute on Neural Networks for Instrumentation, Measurement, and Related Industrial Applications, Crema, Italy, 2001.

[10] Müller JA, Lemke F. Self-Organizing Data Mining: Extracting Knowledge from Data. Trafford Publishing, Canada, 2003

[11] Thomas R. Schultz, Francois Rivest, L´aszl´o Egri, Jean-Philippe Thivierge, and Fr´ed´eric Dandurand. Could Knowledge-based Neural Learning Be Useful in Developmental Robotics? The Case of KBCC. *International Journal of Humanoid Robotics Vol. 4, No. 2*, pages 245–279, 2007

[12] M. Azzeh, D. Neagu and P. Cowling, "Fuzzy grey relational analysis for software effort estimation," *Empirical Software Engineering,* vol. 15, pp. 60-90, 2010.

[13] M. Azzeh, D. Neagu and P. I. Cowling, "Analogy-based software effort estimation using Fuzzy numbers," *Journal of Systems and Software,* vol. 84, pp. 270-284, 2011.

[14] A. Idri, A. Zakrani and A. Zahi, "Design of Radial Basis Function Neural Networks for Software Effort Estimation," *International Journal of Computer Science Issues,* vol. 7, pp. 11-17, 2010.

[15] C. Lopez-Martin, "A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables," *Applied Soft Computing,* vol. 11, pp. 724- 732, 1, 2011.

[16] C. Lopez-Martin, "Applying a general regression neural network for predicting development effort of short-scale programs," *Neural Computing & Applications,* vol. 20, pp. 389- 401, 2011.

[17] C. Lopez-Martin, C. Isaza and A. Chavoya, "Software development effort prediction of industrial projects applying a general regression neural network," *Empirical Software Engineering,* vol. 17, pp. 1-19, 2011.

[18] W. L. Du, D. Ho and L. F. Capretz, "Improving Software Effort Estimation Using Neuro-Fuzzy Model with SEER-SEM," *Global Journal of Computer Science and Technology,* vol. 10, pp. 52-64, 2010.

[19] Y. Li, M. Xie and T. Goh, "Adaptive ridge regression system for software cost estimating on multi-collinear datasets," *Journal of Systems and Software,* vol. 83, pp. 2332-2343, 2010.

[20] G. Kousiouris, T. Cucinotta and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *Journal of Systems and Software,* vol. 84, pp. 1270-1291, 2011.

[21] X. Huang, D. Ho, J. Ren and L. F. Capretz, "Improving the COCOMO model using a neuro-fuzzy approach," *Appl. Soft Comput.,* vol. 7, no. 1, pp. 29-40, 2007.

[22] A. Mittal, K. Parkash and H. Mittal, "Software cost estimation using fuzzy logic," *SIGSOFT Softw. Eng. Notes,* vol. 35, no. 1, pp. 1-7, 2010.

[23] A. B. Nassif, D. Ho and L. F. Capretz, "Regression model for software effort estimation based on the use case point method," in *2011 International Conference on Computer and Software Modeling,* Singapore, 2011, pp. 117-121.

[24] A. B. Nassif, L. F. Capretz and D. Ho, "Estimating software effort based on use case point model using sugeno fuzzy inference system," in *23rd IEEE International Conference on Tools with Artificial Intelligence,* Florida, USA, 2011, pp. 393-398.

[25] A. B. Nassif, L. F. Capretz and D. Ho, " Software Effort Estimation in the Early Stages of the Software Life Cycle Using a Cascade Correlation Neural Network Model," in *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing,* Las Vegas, U.S.A, 2012, pp. 393-398

[26] Vitaly Schetinin, "A Learning Algorithm for Evolving Cascade Neural Networks," in TheorieLabor, Friedrich-Schiller University of Jena Ernst-Abbe-Platz 4, 07740 Jena, Germany

[27] Vitaly Schetinin, "An Evolving Cascade Neural Network Technique for Cleaning Sleep Electroencephalograms," in *Computer Science Department, University of Exeter, Exeter, EX4 4QF, UK*

[28] G´abor Bal´azs, "Cascade-Correlation Neural Networks: A Survey," in Department of Computing Science, University of Alberta, Edmonton, Canada

[29] Mitchell A. Potter, "A Genetic Cascade-Correlation Learning Algorithm," in Computer Science Department George Mason University Fairfax, VA 22030 USA