Cascade-Correlation Neural Networks: A Survey

Gábor Balázs
Department of Computing Science,
University of Alberta, Edmonton, Canada
gbalazs@cs.ualberta.ca

Abstract

This paper is an overview of cascade-correlation neural networks which form a specific class inside neural network function approximators. 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 survey describes the cascade-correlation architecture variants, shows important applications and points to many future research directions.

1 Introduction

Artificial neural networks are universal function approximators 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. Section 2 gives an overview about the cascade-correlation architecture. It also investigates how the ANN training can be improved by curing the previously discussed problems and presents more CCNN variants. Section 3 shows important CCNN applications. Finally section 4 concludes the survey and summarizes the mentioned future research directions.

1.1 Artificial neural networks

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

$$y = f\left(\sum_{k=1}^{n} w_k x_k\right), \tag{1}$$

equation where y is the output value transferred further by all the output connections, x_k is the kth input, w_k is the weight of the connection related to the kth input and f is the activation function which is usually the signum ($sign$) or the sigmoid (σ) function defined as

$$sign(x) = \begin{cases} 1, & \textit{if } x > 0 \\ 0, & \textit{otherwise} \end{cases}, \quad \sigma(x) = \frac{1}{1+e^{-x}}. \tag{2}$$

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.* 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 activation function, so these neurons just feed the data into the network. The ANN of Figure 1 has only connections which goes into subsequent layers. These kind of networks are called *feedforward ANNs.* A network with connections going to the same or to a previous layer is called *recurrent ANN.*

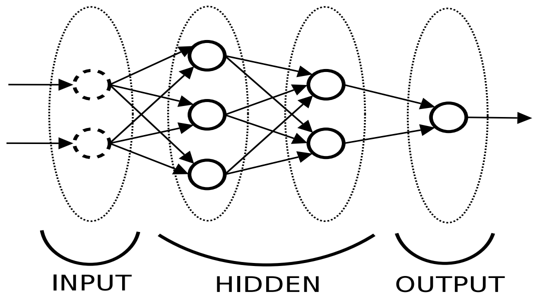

Figure 1. A 2-3-2-1 feedforward ANN

It is known that a feedforward ANN with one hidden layer is an universal function approximator [1], so it can approximate any bounded continuous function with arbitrary precision (however it needs exponentially many neurons for this). Furthermore, any feedforward 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 trained accordingly. Then the locally corrigated 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 quickprop, was published by Fahlman (1988) [2].

1.2 Problems of neural network training

ANNs are trained by supervised learning where input-output pairs (*samples*) are presented. Then the network tries to minimize the prediction error by adjusting its weights. The learning process is usually done in *epochs*, a complete training pass through the entire sample set. The most popular and widely used method is the backpropagation algorithm which has many problems leading to an inefficient, very slow learning process.

Because backpropagation is a gradient descent algorithm, it converges only to a *local minimum.* A popular way to handle this problem is to initialize the network with random weights and repeat the whole learning process with different randomizations. This approach increases the computational efforts seriously and gives no guarantees to reach the global or even a close local optimum.

The backpropagation algorithm has a *step size* (sometimes called *learning rate*) parameter which determines how much the error is corrigated by the weights in a given step. The step size can be global for the entire network or local for the individual weights. It also can be static for the whole training process or can be adaptively changed during that. Unfortunately it is far from obvious how to select the step size properly. If it is too small then the learning can become very slow. If it is too big, the learning can diverge without reaching a solution.

During learning each neuron tries to be a feature detector and so plays a useful role in the network's behavior. But because most of the neurons do not communicate with each other, they learn independently according to a common error signal computed by the difference between the network's and the desired outputs. If the error signal is strong enough, a neuron can change its target and decides to be a detector of another feature. Without synchronization the independent parts cannot quickly agree on the feature targets and so they keep changing for a long time. This fact can significantly slow down the learning convergence. It is known as the *moving target problem.*

Another important issue of neural networks is to define their *structure* which can really affect the learning capabilities of the network. One can use a standard structure as the fully connected feedforward ANN (shown on Figure 1), generate it by using an evolution algorithm (which is computationally expensive) or use the cascade-correlation architecture.

2 The cascade-correlation architecture

The cascade-correlation architecture, published by Fahlman and Lebiere (1990) [3], 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.

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 2, where the dashed connections are trained. The empty layer shows the place of the candidate neuron where it is activated to after its input weights are learned and become frozen. 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.

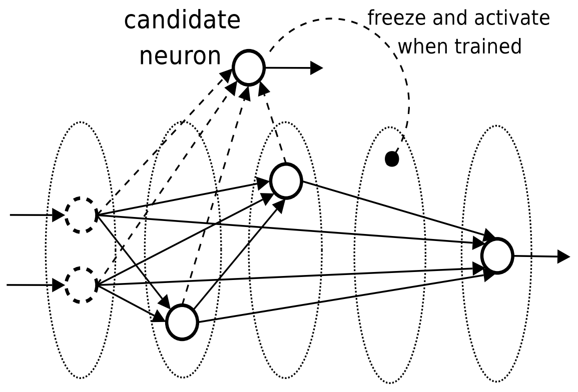

Figure 2. CCNN training

During the candidate neuron training the goal is to maximize the covariance[1] (C) between the network's and the neuron's outputs with respect to the weights of the candidate neuron. The covariance can be obtained as

$$C = \sum_{o \in O} \left| \sum_{s \in S} (y_s - \overline{y})(e_{o,s} - \overline{e_o}) \right|, \tag{3}$$

where O is the set of output neurons of the network, S is the set of training samples, y_s is the candidate's output for the sample s, $e_{o,s}$ is the error at output neuron o for the sample s and $\overline{y}$, $\overline{e_o}$ are the averaged values of y_s and $e_{o,s}$ respectively over all the samples $s \in S$. This maximization can be done by the gradient ascent algorithm.

Instead of the covariance other measures can be used for candidate selection like the correlation (by dividing the covariance with a normalization factor), the Frobenius norm of the covariance [4] or simply the squared error difference [5]. This choice should depend on the application. Some empirical results [5] concluded that the covariance is more suitable for classification while the squared error difference has better results on regression problems.

[1]The correlation in the name came by historical reasons (notes of Fahlman [3])

2.1 CCNN training efficiency

The efficiency of the training can be improved by replacing the candidate neuron by a pool of candidate neurons. They are learned simultaneously, but they act as isolated units, so they do not affect each others training. At the end of the neuron-creation step only the best is activated and the others are discarded. Using many neurons this way helps the training to avoid ending in a local minima and resulting a network more suitable for the problem than a standard feedforward ANN trained by backpropagation. The candidate neurons also can have different activation functions and so an inhomogeneous network can be built which usually has better generalization capabilities than a homogeneous one.

Because in this architecture only individual neurons are trained, the feature detection target of more neurons cannot change and so the moving target problem cannot occur. This individual training also makes the step size problem easy to handle. Because only one neuron is focused at the same time (the candidate neurons are isolated), only one step size parameter has to be tuned and so it can be chosen “bravely” making big steps towards the goal resulting a much faster convergence.

The backpropagation algorithm needs a forward and a backward sweep when computing the network output and propagating the error backward. In contrast the cascade-correlation architecture needs only a forward sweep to compute the network output and then it can use this information to train the candidate neurons. Because the candidates do not have any effect on the active network, the training time can be drastically reduced by storing and reusing the network outputs for all the training samples if enough memory is available.

During initialization or after the activation of the candidate neuron a simple gradient descent algorithm can be used to train the output connections of the network. However, Fahlman and Lebiere suggested using the quickprop algorithm even in this case. It has many built-in tuning (including a sophisticated step-size adaptation) with which it can significantly reduce the training time.

CCNNs have very expressive power to represent any training set, but they can easily overfit. So to control this problem one has to use isolated training samples as a validation set or the adaptive model selection technique proposed by Schuurmans and Southey (2002) [6].

2.2 Reducing the depth of CCNNs

Because each candidate neuron goes into a new layer, the CCNN training usually produces deep networks with slow data throughput capabilities what makes computing the network output inefficient. This depth can be drastically reduced by a simple CCNN variant called sibling/descendant cascade-correlation (SDCC) published by Baluja and Fahlman (1994) [7].

In the SDCC variant the pool of candidate neurons are divided to two parts called sibling and descendant units. The descendant units are connected to the network as in case of the regular CCNN training, but the sibling neurons are not connected to neurons of the last hidden layer, just to the previous ones. The sibling and descendant units compete each other during the learning process and only one of them is selected to be activated. If a sibling unit is chosen, it is put into the current last layer of the network. In case of a descendant neuron a new layer is created as it was done during the CCNN training.

One would expect that a sibling unit is never activated because it has fewer connections and so it cannot become such a good feature detector as a descendant neuron. However, it is not always the case. If the additional connections do not provide significant information to the descendant unit, its learning algorithm shift the related weights towards zero. If it happens, the sibling units converge much faster and can provide better results at the end of the candidate training process, so it becomes activated instead of a descendant unit.

2.3 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) [8].

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. Then the neuron's computational model changes into

$$y_t = f(r_t), \quad r_t = \sum_{k=1}^{n} w_k x_{t,k} + w_s y_{t-1}, \tag{4}$$

where y_{t-1}, y_t are the outputs at time $t-1$, t respectively, $x_{t,k}$ is the input of the kth connection at time t, w_k is the current weights of the kth feedforward connection coming from an other neuron, w_s is the weight of the self-recurrent connection and f is the activation function. During the candidate training process the following derivatives are used,

$$\begin{aligned}\frac{\partial y_t}{\partial w_i} &= f'(r_t)\left(x_{t,i} + w_s \frac{\partial y_{t-1}}{\partial w_i}\right),\\ \frac{\partial y_t}{\partial w_s} &= f'(r_t)\left(y_{t-1} + w_s \frac{\partial y_{t-1}}{\partial w_s}\right),\end{aligned} \tag{5}$$

where $1 \le i \le n$ and f' is the differential of the activation function respect to its input r. The ∂y_{t-1} derivatives and the y_{t-1} output value can be obtained from the previous learning step.

It is easy to see that the RCC architecture can be combined with the SDCC variant which is able to compress the depth of the network. Figure 3 shows such an example. The dashed connections are trained simultaneously including the self-recurrent loop as well. This structure seems to be reasonable, but there are not any empirical results published with this combination yet.

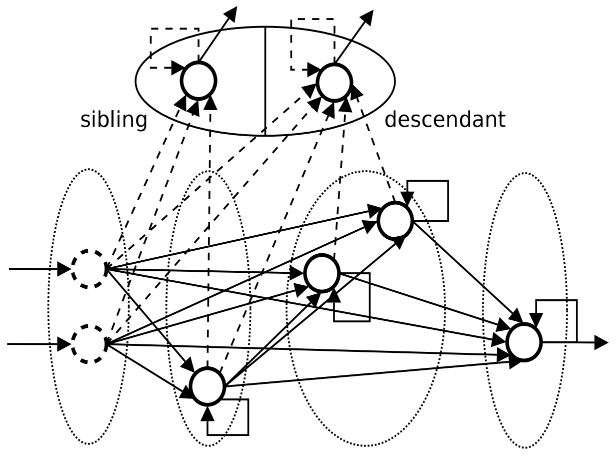

Figure 3. RCC combined with SDCC

2.4 Pruning CCNNs

The cascade-correlation architecture adaptively fits the network topology to the application, but it uses a fixed connection scheme among the neurons. This scheme may not be optimal in every case. To overcome this problem one can try to find connections having the least impact on the network output and remove them. Less connection can improve the network's generalization capabilities and the learning speed, But without care it can reduce the network's approximation power too.

Thivierge et al. [4] used the optimal brain damage (OBD) method to prune the least important connections of CCNNs. The OBD technique computes so called saliency values for each connection which measures the connection's effect on the network output. This value is computed by neglecting many terms of the error function's second order Taylor estimation. By using mean squared error (E) the computation becomes to

$$s(w) = \frac{\partial E}{\partial^2 w} \cdot \frac{w^2}{2}, \quad E = \frac{1}{2} \sum_{o \in O} (y_o - t_{s,o})^2, \quad (6)$$

where O is the set of output neurons, y_o is the output of neuron o, $t_{s,o}$ is the desired output for neuron o and sample s, and $s(w)$ is the saliency value of connection related to weight w. Then connections with less saliency values can be pruned.

In case of CCNNs the OBD technique was used both at the end of candidate training to prune input connections and at the end of the training step to prune output connections as well. They tested this algorithm on classification problems where OBD removed almost half of the connections and improved generalization capability on unseen samples.

3 Applications

In this section two important applications are shown where cascade-correlation neural networks were successfully used and compared to other neural network methods.

3.1 TD-learning with CCNNs

In online learning tasks function approximators have to interact as databases with read-write capabilities. So the training period of the approximators are not separated from their usage. One such an application is temporal-difference learning solving reinforcement learning problems [9].

In case of fixed-structure neural networks the backpropagation algorithm can be used by making one learning step when data arrives. However, this approach is infeasible for constructive networks as CCNNs. Rivest and Precup (2003) published [10] a cached training algorithm what they successfully combined with CCNNs and temporal-difference learning to solve reinforcement learning tasks. Figure 4 shows the basic idea of their algorithm. It maintains a cache of the recently read and written values and use it as an information channel between the application and the underlying neural network. When the cache is full, the algorithm trains the neural network by using the cached values (consolidation) and empties the cache. This approach can be combined with the fixed-structure networks too, but according to their results CCNNs performed sometimes better and no worse otherwise. The difficulty for CCNNs that they freeze their weights and do not change them anymore. One way to overcome this is to allow the weight change and train the whole network by backpropagation after the candidate neuron is activated. Unfortunately in this case the same problems would arise again what CCNNs tried to solve. One

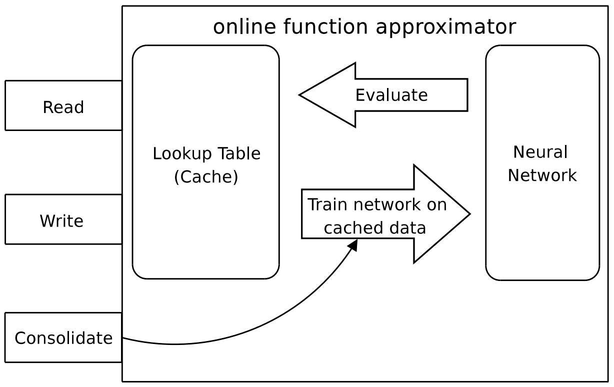

Figure 4. Cached NN training

way to handle this, proposed by Steffen Nissen [5], is to cool down the previously fixed weights during the full backpropagation step by using a small learning rate. He also proposed to keep the output connection training separately before of this full backpropagation step.

3.2 Knowledge-based and rule-based CC

In many applications huge amount of prior knowledge is available about the problem. Instead of building a completely new approximator it seems more appealing to reuse available knowledge and get a better result faster. It is the idea of knowledge-based cascade-correlation (KBCC) neural networks published by Schultz and Rivest (2001) [11].

KBCC networks use pre-trained modules as single candidate units and build them into the network if they are desirable for the solution. These modules act as nested networks, aggregate many input connections and provide as many outputs as the host cascade-correlation network has. The only requirement is the derivative of these modules has to be computable for the training process.

KBCC is tested on many toy and realistic applications including learning various geometric shapes, vowel recognition and gene splice-junction determination where it significantly outperformed the multi-task learning methods [11] and the knowledge-based artificial neural network (KBANN) algorithm [12].

One of the biggest limitations of KBCC that it can be computationally expensive if complex modules are used. Another serious problem that KBCC usually prefers to select these complex modules instead of single neuron candidates because their results can be more accurate under a single training step. It is not always beneficial and one should penalize the complexity of the candidate modules in some way which is subject of future research at the present time.

Rule-based cascade-correlation (RBCC) neural networks, published by Thivierge et al. (2004) [13], are a variant of KBCC networks. RBCC networks use rule-based prior knowledge to generate rule-based networks. They can be considered as the generalization of the KBANN framework which creates only disjunctive (OR) and conjunctive (AND) rules. RBCC networks can create any m-of-n rule where m is the number of features being selected among n ones. Using these rules RBCC can replace long conjunctive-disjunctive rule chains by a simpler m-of-n rule.

RBCC showed promising results against KBANN, regular CC and KBCC on the gene splice-junction determination problem [14], however there has not been any exhaustive comparison published yet.

4 Conclusion

In this paper we made an overview about cascade-correlation neural networks which have an adaptively built architecture autonomously fitting to the target application. We covered their training method which does 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 many questions are still open for future research. Further investigations could focus on the choice of suitable measures for candidate selection, relaxation of weight freezing, preventing easy overfitting, creating a measure for penalizing complex units used by knowledge-based networks and study further the recurrent cascade-correlation architecture. The empirical results of cascade-correlation networks were usually based on comparisons against fixed-structure feedforward neural-networks trained by the back-propagation algorithm. However, it could be beneficial having some experience how they compete against other function approximator families.

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. *Proceedings 1988 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(λ) 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] Richard S. Sutton and Andrew R. Barto. Reinforcement Learning: An Introduction. *MIT Press, Cambridge, MA*, 1998.

[10] Fracois Rivest and Doina Precup. Combining TD-learning with Cascade-correlation Networks. *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, 2003.

[11] Thomas R. Schultz and Francois Rivest. Using Knowledge to Speed Learning: A Comparison of Knowledge-based Cascade-correlation and Multi-task Learning. *Connection Science vol. 13*, pages 1–30, 2001.

[12] Thomas R. Schultz, Francois Rivest, László Egri, and Jean-Philippe Thivierge. Knowledge-based Learning with KBCC. *Proceedings in the 5th International Conference on Development and Learning*, 2006.

[13] Jean-Philippe Thivierge, Frederic Dandurand, and Thomas R. Schultz. Transferring Domain Rules in a Constructive Network: Introducing RBCC. *Proceedings in the IEEE International Conference on Neural Networks*, pages 1403–1409, 2004.

[14] Thomas R. Schultz, Francois Rivest, László Egri, Jean-Philippe Thivierge, and Frédéric 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.