Developing Adaptive Systems : Constructive and Pruning Algorithms

eingereichte
HAUPTSEMINAR
von

stud. Alexis Guibourgé

geb. am 28.07.1990
wohnhaft in:
Lierstraße 11A
80639 München
Tel.: 089 1791762787

Lehrstuhl für
STEUERUNGS- UND REGLUNGSTECHNIK
Technische Universität München

Univ.-Prof. Dr.-Ing./Univ. Tokio Martin Buss

Betreuer: Dipl.-Ing./Dr. Betreuer Cristian Axenie
Beginn: 01.10.2014
Abgabe: 14.01.2015

Abstract

This paper focuses on adaptive systems generating artificial neural networks. More precisely, the Constructive, the Pruning and the Hybrid algorithms will be studied. Firstly, we will go through the Constructive and the Pruning processes. For both of them, some classical examples will be presented. Then, the focus will be on some of the Hybrid algorithms existing in the literature. A huge number of them have already been developed and three main methods to build such algorithms will be exposed. Finally, in order to illustrate how to use those approaches, a case study has been imagined as well as a solution using the seen concepts.

Contents

Chapter 1

Introduction

Since the Eighties, the domain of artificial intelligence has developed neural networks (NN), following the concept of biocybernetics by trying to imitate the brain functioning. While the human brain is currently the most powerful intelligence that we know, its structure seems to be rather simple: it is only made up of neurons, the synapses connecting them with one another. Taking advantage of the processing principles of the brain, the artificial neural networks (ANN) were used in many applications.

1.1 Applications of Neural Networks

The sample tasks performed by artificial neural networks are classification and regression. This means neural networks will either be used to recognize patterns and classify them, or to approximate an unknown function. The application domain of NN is very large: they are used in economics for market prediction, in mechatronics for system identification, in signal processing for data compression, etc.

1.2 Motivating CoNN and PNN

However, the brain structure is not as simple as we suggested previously. Indeed, its complexity appears with the dynamical mechanisms it utilises during its life. The brain changes over time and adapts itself to its environment. Indeed, during the first phase of our life, the number of neurons and synapses increases reaching a peak at six months. Then, depending on their use, the latter survive or die.
Biomimicry motivates the development of the Constructive Neural Network (CoNN) and the Pruning Neural Network (PNN) [T.K]. Indeed, unlike classical neural network construction, CoNN and PNN have a much more realistic training stage, in so far as the network topology is able to change during this period. CoNN will add new connections and new neurons during this stage, as long as the performance improves, like during the first phase of our life. PNN will, contrary to CoNN, delete

connections and neurons that do not contribute enough to the performance of the network, as the brain would do in its mature phase by letting neurons die and abandoning connections. Finally, recent research has focused on hybrid algorithms that combine CoNN and PNN together. As we will see, this could be a way to take advantage of the power of both algorithms.
Another motivation of CoNN and PNN is the difficulty of predicting the right topology of a neural network for a given task. This strong argument does not promote the use of an apriori fixed network topology but rather an adaptive network learning algorithm, exploring the entire space of neural network topologies. That is exactly what CoNN and PNN do, and this approach leads to an efficient network with a minimal complexity.

1.3 Neural network formalism

1.3.1 NN modeling

In this section we will introduce all the necessary notations in order to be able to describe neural networks easily. CoNN and PNN only build feedforward neural network (FNN). This means the signal always goes from the input to the output without any loop. The figure (1.1) gives us the main notation:

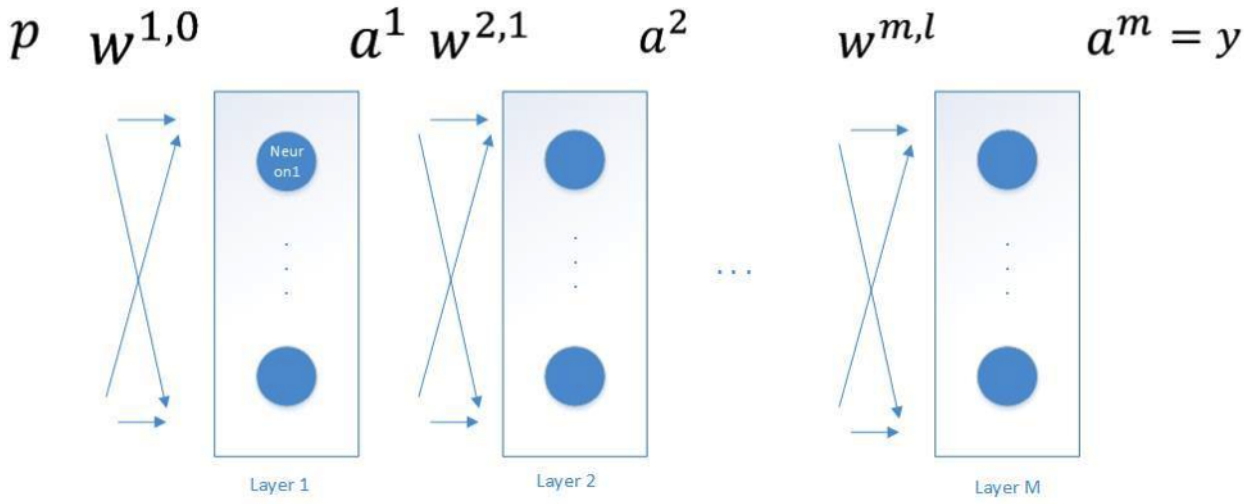

Figure 1.1: NN formal description

M : number of layers
S^m : number of neurons in the layer m
$W^{m,l}$: Weight matrix going from the l^{th} layer to the m^{th} layer
Q : dimension of the training set
$\underline{p}^q$: q^{th} input signal

$\underline{\hat{y}}^q$: q^{th} output signal
$\underline{y}^q$: desired output signal
$\underline{a}^m$: output of the layer m

1.4 Typical usage of ANN

In the literature, two main problems are considered: regression and classification. The regression problem aims to approximate an unknown function, and more generally an unknown system. To be able to solve it, we need an objective function computing the performance of the network. In most cases, this function will be the mean square error function, and we will try to minimize it. The equation (1.1) shows the formal formulation of this optimization problem according to the introduced notations.

$$\min_W E = \frac{1}{2} \sum_{p=1}^{Q} \sum_{k=1}^{S^{out}} (\underline{\hat{y}}_{k,p} - \underline{y}_{k,p})(\underline{\hat{y}}_{k,p} - \underline{y}_{k,p})^H \tag{1.1}$$

The classification problem can be seen as a particular case of the regression task. The function we want to approximate is here a mapping function, taking values depending on the class of the input, as shown in the equation (1.2)

$$\underline{p}^q \rightarrow class(\underline{p}^q) \quad \forall q \in [1..Q] \tag{1.2}$$

As previously, we can introduce the same error function or a weighted error function according to the class importance. For all the cases, the CoNN and PNN will change the structure of the network until the objective function is included in an acceptable range. This range will be set by the user and will depend on the application.
Apart from the error minimization which is related to the precision of the network, two other criteria are taken into account while designing the algorithm. These are the convergence speed and the final network size.

Chapter 2

Formalism and Applications of CoNN models

As suggested by the name, constructive neural networks are governed by a "constructivist" process. This means we start from a minimal network and we increment its complexity, that is the number of neurons and connections, as we go along the algorithm. The initial network is generally a single layer feedforward network (SLFNN).

2.1 Operation principle and issues

There are plenty of ways to construct CoNN. Of these, we will focus on three which are widely used: the pyramid / tower algorithm, the tiling algorithm and the cascade correlation algorithm. Each of these algorithms has its own way to construct NN and solves different types of tasks. The pyramid and the tower algorithms will add a fixed sized layer at each step of the process, constructing a deep neural network. Conversely, the tiling algorithm will start by increasing the size of the last layer, before adding a new one and will thus lead to shallow NN. Finally, the cascade correlation algorithm will again add a fixed size layer to the network at each iteration but will take the correlation between the last layer's output and the MSE as the objective function. The choice of the algorithm to use will depend on the problem we want to solve, as we will see later on.
While designing a NN, several issues appear. The most common one is the "moving target problem". At each step, the network topology will change and all the neurons will be retrained in a new environment. It will thus be very difficult for a neuron to find its optimal weights if its environment is not stable. Finding the optimum weight's values may thus be very demanding and it can extend significantly the training time. The most frequently used strategy to contain this issue is the so called "weight freezing". Depending on the algorithm, some weights will be frozen as we go along the process. Once frozen, the weight is not able to change any more. Consequently, we will only train the last introduced neurons, which will accelerate

the training and avoid too brutal changes of the network. Unfortunately, this method often leads to suboptimal solutions.

2.2 Main algorithms

2.2.1 Tower and Pyramid Algorithms

The tower and the pyramid algorithms are used for the particular case of patterns classification [Li10]. As said before, they will add fixed sized layers to the network until the objective function reaches a satisfying value. We will firstly see how those two algorithms work for the very simple case of two-class classification, and we will then extend our explaination to the N-class classification.

The input layer will be as large as the dimension of the sample we want to classify, the hidden layers and the output layer will be as large as the number of classes. Assuming that if a sample does not belong to the first class, it then belongs to the second class, the network will only need one neuron in each hidden layer as well as in the output layer. During the process, we will add neurons one by one and set the connections. The only difference between the tower and the pyramid algorithm lies in this setting. For the tower algorithm, each layer will be connected to the input layer and to the previous layer. For the pyramid algorithm all shortcut connections are set, that is each layer will be connected to the input layer but also to all the previous layers and not only the one just before. The figure (2.1) gives a representation of both networks. We can notice that the red connection is specific to the pyramid algorithm. Once this is done, we compute the objective function seen in the previous chapter. If it is satisfying, we stop there. If not, we train the network with a classic training algorithm (in the studied case, backpropagation is used) and we add a new layer.

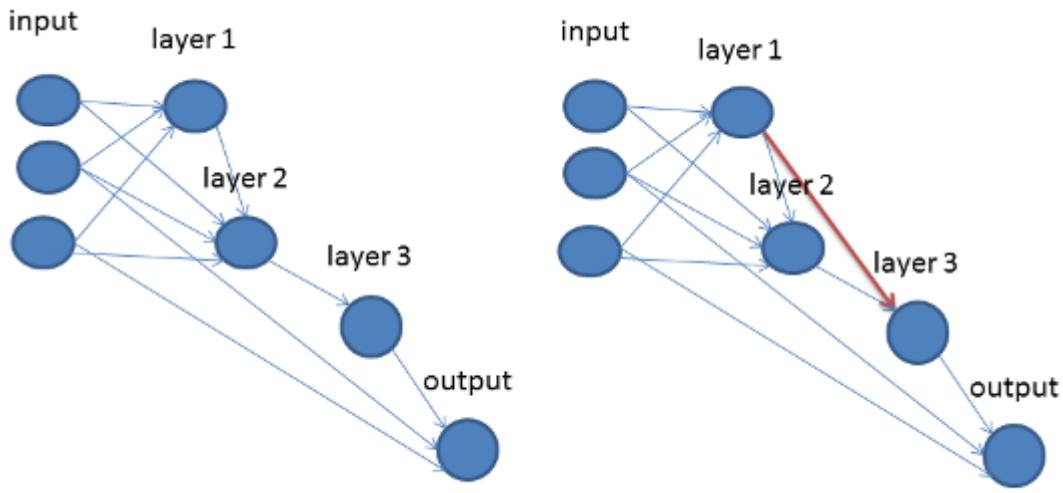

Figure 2.1: Left : Tower network Right : Pyramid network M = 3

It is proven that this algorithm converges. Indeed, adding a new neuron will always increase by at least one the number of well matched pattern. As the training set is finite, the convergence to zero error is guaranteed.
This structure can easily be extended for a M-class classification task, by adding M neurons in each new layer instead of only one. The other steps remain similar.
The main advantage of those algorithms is their simplicity. They are easy to implement and to train. Unfortunately, the topology space allowed is very limited, and we thus often come to a suboptimal network, in terms of size and speed. The Pyramid algorithm will build smaller networks on average, but its training time will be longer.

2.2.2 The Tiling Algorithm

We again consider the two-class classification task. Unlike the previous algorithm, the tiling algorithm will start by adding neurons to the last layer before adding any new layers. The motivation for proceeding like this is to create a more compact network and thus build a more efficient network in terms of speed [Jr.06]. Moreover, this algorithm is designed to handle the "moving target problem" as weights freezing is used and its convergence speed will thus be increased.
The process starts constructing a new layer by adding a single neuron to it, in the 2-class classification case. This first neuron is called the master neuron and it will be trained to give the closest signal to the desired output with help of the perceptron algorithm. If the achieved mapping is not perfect, we then add tiles to this last layer. Those tiles, called ancillary neurons, will have to make the different classes faithful. This means that they will make sure that two patterns, belonging to different classes, will have two different internal representations. To do so, we

take an arbitrary output a^L and we look for the number of classes giving this same output. We then add to the last layer as many neurons as there are classes giving this output, and we train them to separate those classes with the perceptron algorithm. As soon as the classes are faithful, we add a new output layer made up of a single new master neuron, and we repeat the described operation until the classification is satisfying. The figure (2.2) shows visually how the network looks. The squares are the master neurons.

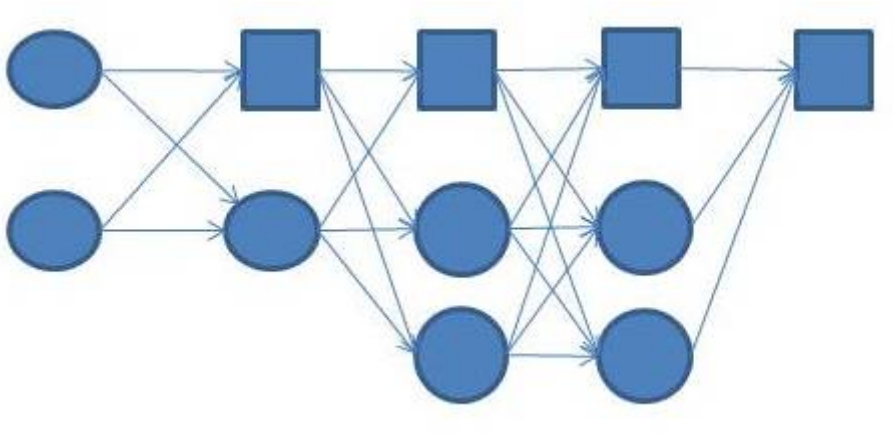

Figure 2.2: Tilling Network M = 3

Again, the following theorem ensures the convergence of this algorithm: "If the layer L-1 gives a faithful representation of the input set and the error amount of the master unit of this layer is non-zero, we can find a set of weights connecting this layer to the next master neuron, such that $e_L \leq e_{L-1} - 1$"
Important to note is that only the last layer is trained at each iteration. The other layers are frozen. This explains mainly why the tiling algorithm shows good convergence speed.

2.2.3 The Cascade Correlation Algorithm

Instead of minimizing the mean square error function as would do most of the CoNNs, the idea of the cascade correlation algorithm (CCA) is to maximize the correlation between the MSE and the output of the hidden layers [Sha10]. Neurons and layers are added one by one, and shortcut connections are used in order to do so. Another interesting feature of this NN is that it uses continuous transfer functions rather than boolean transfer functions. This algorithm is thus used to perform general regression tasks and not only classification.
The algorithm starts with a minimal network: the input layer is directly connected to the output layer. Then, one neuron will be added and will only be connected to the input layer. With help of classical training algorithms such as backpropagation we train the neurons weight to maximize the correlation between the network error

and the output of this new neuron. The equation (2.1) illustrate the corresponding optimization problem.

$$\max_{W_{L,k}} Corr = \sum_{i=1}^{Q} \sum_{j=1}^{S^{out}} (a(i) - \bar{a})(y_j(i) - \bar{y}_j) \tag{2.1}$$

where a is the output of the newly added neuron, y_i is the i^{th} output of the output layer, $\bar{y}$ and $\bar{a}$ are the average value over all the training pattern. L is the number of the layer, and k belongs to the set $\{1, ..., L-1\}$. Once the weights are found, we freeze them and set the hidden to output connection. This is done very simply: if the hidden neuron correlates positively with the error at a given output neuron, we introduce a negative weight. Otherwise, the weight will be positive.
The weight freezing process associated with the fact that MSE is not directly minimized widely reduces the moving target problem as well as the sub optimum problem. This explains the success of CAA.

2.3 Conclusion

We saw in this section three different ways of performing CoNN. As we said, one will choose the algorithm to use depending on the problem to solve. Pyramid, tower and tiling algorithms will be used to perform classification tasks. If the problem is highly nonlinear, we advise the use of the deep NN built by pyramid or tower algorithms. Conversely, if the problem is easier to solve, we will rather use shallow NN produced by the tiling algorithm in order to get a better rate. Finally, to perform more general tasks, the cascade correlation algorithm is one of the most efficient algorithms as it manages to handle the local minimum problem as well as the moving target problem. Variations in those algorithms have been implemented in order to increase their performance, and they are still studied today.
However, CoNN shows two main weaknesses. Firstly, the number of neurons needed to solve a task is often overestimated, as CoNN algorithms are only able to add neurons but not to remove them. Secondly, and this is related to the first point, CoNN algorithms face the overfitting issue. This means that the networks produced are often over trained: they show excellent performance on the training samples set, but weak performance for unknown data.

Chapter 3

Formalism and Applications of pruning network models

Broadly speaking, the pruning neural network algorithm is exactly the opposite of the constructive neural network algorithm. For a given problem, we initialize the NN with an oversized network and select the neurons and connections we want to keep. The other connections and neurons will be erased as we go along the procedure. This "selectivist" algorithm imitates the neuron death process of our brain [T.K]. This process mainly happens to neurons that are rarely active.

3.1 Operation principle

The most important step in all pruning neural network algorithms (PNN) is the research of the neurons and connections that will be pruned. The latter have to be selected so that, if erased, the quality of the network output remains acceptable. A lot of strategies have been developed to find this set of neurons and we will go through some of the most widely used algorithms. Once the set of neurons is found, the next step will generally be to erase this selection (neuron death process) or to merge them. Then, a classic training stage takes place, with help of the backpropagation algorithm for example. This procedure is then repeated over and over until the size of the network becomes satisfying or until the MSE exceeds a user defined threshold. To measure the network quality, the mean square error is again used.

3.2 Main algorithms

3.2.1 Simple algorithms

In some cases, simple algorithms can show an as good performance as more elaborate ones. This is true for the magnitude based algorithm and the randomly pruning algorithm [Cos02]

The magnitude weight algorithm simply selects the weakest connections. It then starts to erase them one by one, until the MSE exceeds a preset value. The randomly pruning algorithm itself erases connections randomly. If the MSE variation is included within an acceptable range the weight will definitely be removed, if not it will be maintained.
Surprisingly, those algorithms show good performance on small networks. Indeed, for networks with only one or two layers, small weights will have little impact on the output. In the same way, randomly pruning shows good performance on small networks as it will not be too demanding to find the set of weak weights this way. Moreover, those algorithms are much appreciated as they do not need to retrain the network at each step. However, the efficiency of the algorithms decreases as the size of the network grows. Indeed, small weights can have a large impact on complex networks and trying to randomly find the subset of weak connections in a large set of weights can become very difficult. This is why more elaborate algorithms are needed to prune larger networks.

3.2.2 Optimal Brain Damage

Unlike the magnitude based algorithm, the OBD algorithm will not judge the importance of a connection with respect to its weight value, but with respect to its saliency. The saliency of a parameter represents the change in the objective function introduced by its deletion. The optimal brain damage algorithm will then delete the set of neurons with the weakest saliency.
The key step of the OBD algorithm is thus to find an efficient way to compute the saliency of each neuron. It is not possible to compute the latter directly by deleting each neuron as this would be too demanding for large network. The method proposed by OBD to achieve our goal is to construct a local error model by approximating the error surface with its Taylor series. By doing so, and by assuming some approximations, we will be able to approximate the saliency of each neuron. The Taylor series of the error at a certain point is given by the expression (3.1)

$$\delta E = G^T \delta w + \frac{1}{2} \delta w^T . H . \delta w + O(\| \delta w \|^3) \tag{3.1}$$

E : MSE (defined at the beginning of this paper)
w : Weight vector
H : Hessian matrix of E with respect to W
G : Gradient of E with respect to W

Each weight variation is associated to an error variation, the saliency, that is computable with the previous expression. However, in most cases, it is not possible to compute H as it is very demanding. More approximations have to be done in order to get the saliency. We will thus assume that the saliency of each connection is not influenced by the deletion of other connections, we will also assume that the network

is in an optimal state at the beginning of the algorithm and that we can neglect the order bigger than 2 in the Taylor approximation of the MSE surface. This allows us to consider that the Hessian matrix is diagonal and that the gradient is equal to zero. (3.2) shows the expression of δE after simplification.

$$\delta E = \frac{1}{2} \sum_i h_{i,i} \delta w_i^2 \tag{3.2}$$

The saliency then appears immediately and is given by (3.3)

$$s_k = \frac{1}{2} |h_{k,k}| \tag{3.3}$$

Once we have the saliencies, we have a more realistic view of the impact of each weight on the MSE. The algorithm will then delete the weakest weight, in terms of saliency, until the MSE exceeds the threshold. Then, the network will again be trained with help of the backpropagation algorithm. Once an optimum is reached, the saliencies are computed one more time and the whole process starts again. This is done over and over, until the size of the network is satisfying or until it is not possible to prune anymore without leaving the MSE outside the tolerated range.
The weakness of this algorithm lays in the assumptions we make on the Hessian matrix [Yan06]. Indeed, this matrix is often highly non diagonal, especially for deep neural network. To overcome this, another algorithm has been developed, the optimal brain surgeon algorithm.

3.2.3 Optimal Brain Surgeon

This algorithm also uses the Hessian matrix to prune weights, but without directly computing the saliency. It does not assume that the Hessian matrix is diagonal, that is that a neuron deletion will have an impact on other neuron's saliencies. This means that, unfortunately, the whole hessian matrix will have to be computed. This algorithm is therefore very demanding, but it shows better performance than OBD since it prunes more appropriate weights. [Yan06]
As in the previous algorithm, we will consider that the saliency computation starts once the network has reached an optimal state. Therefore, the gradient is once again equal to zero and the error surface can be approximated by the expression (3.4).

$$\delta E = \frac{1}{2} \delta w^T . H . \delta w + O(\| \delta w \|^3) \tag{3.4}$$

At each step, the algorithm will simulate a weight erasure and compute the set of other weights that minimizes the MSE approximation. It is reasonable to do so as those optimization problem are really easy to solve once we have the Hessian matrix. Let e_q be the q^{th} canonic base vector of the weight's space. (3.5) illustrates the optimization problem.

$$\min_{\delta w} \frac{1}{2}\delta w^T.H.\delta w \qquad subject\ to \qquad e_q^T.\delta w + w_q = 0 \tag{3.5}$$

Using the Lagrangian function (3.6) and assuming that strong duality holds, we can find the solution of each optimization problem.

$$L(\lambda, \delta w) = \frac{1}{2}.\delta w^T.H.\delta w + \lambda.(e_q^T.\delta w + w_q) \tag{3.6}$$

$$\delta w = -\frac{w_q}{[H^{-1}]_{qq}} H^{-1} e_q \tag{3.7}$$

Important to note is that the q-th component of δw is deleted and that all its other component are updated. We so compute δw for all the optimization problems and choose the one that minimize δE . After that, there is no need to retrain the network with backpropagation as all the weights have already been updated in a manner that minimize the MSE. The algorithm thus computes the new hessian matrix and solves again the optimization problems. As usual, the algorithm stops when the MSE exceeds a certain value or when the network reaches a reasonable size. This algorithm is very powerful because there is no need to train the NN after each prune step. Moreover, the deleted connections are very well chosen and it is thus possible to reduce significantly the size of the network with help of this algorithm. However, like the OBD algorithm, the OBS algorithm has a low convergence speed. Indeed, a lot of computation power is required and weights are deleted one by one. The unit - optimal brain surgeon tries to overcome these problems.

3.2.4 Unit - Optimal Brain Surgeon

To increase the convergence speed of the pruning process, unit - optimal brain surgeon erases a whole neuron at each step, and not only a single weight. The principle is the same as for OBS, but instead of computing the weight saliency, we compute the neuron saliency. For a given neuron u, we thus introduce M, its unit matrix defined such that gives the output weights of the neuron u. Assuming the same hypothesis as for the OBS, the equation (3.8) will give the MSE differential.

$$\delta E(u) = \frac{1}{2} w^T M (M^T H^{-1} M)^{-1} M^T w \tag{3.8}$$

Like before, the algorithm has to solve as many optimization problems as there are neurons in the network. The weights will be updated with the solution δw leading to the smallest increase of E. The solution of each optimization problem is given by (3.9).

$$\delta w = -H^{-1} M (M^T H^{-1} M)^{-1} M^T w \tag{3.9}$$

If the maximum number of output weights is not too large in comparison to the number of neurons, one can show that the complexity of this algorithm is of same order

of magnitude as the OBS algorithm. In this case, it will thus be more appropriate to use the unit-OBS as the convergence will be faster [Yan06].

3.3 Conclusion

The key point of PNN algorithm is the choice of weights to prune. If this is done properly, the neural network size can be significantly reduced. PNNs overcome problems CoNNs face, such as overfitting and bad generalization. Moreover, PNN algorithms produce neural network with high rates, as the network structure is simplified. However, it is very difficult to initialize PNNs because of the initial network. This network has to be large enough to solve the given task, but not too large because this will increase the execution time of the algorithm.

Chapter 4

Hybrid constructive-pruning models

4.1 Issues and motivations

Hybrid constructive-pruning algorithms are precisely designed to combine the advantages of CoNNs and PNNs and to overcome the different problems they face. The main idea of hybrid algorithms is to use both CoNN and PNN algorithms, and to make them cooperate. This leads to algorithms that are able to add or remove neurons depending on the need.

In this chapter, we will go through four different ways to combine those algorithms. The cascade neural network design algorithm combines them in the simplest way: the two algorithms are juxtaposed and the NN will first be constructed and then be pruned. On the contrary, the input side weight pruning and the adaptive merging and growing algorithm totally mix the pruning and the constructive part of the algorithm in so far that at each step, a neuron will be created and another one pruned if possible. Finally, some hybrid algorithms are adaptations of CoNN and PNN algorithms. This is the case for the Hybrid M-Tiling algorithm: a pruning process has simply been added to the M-Tiling algorithm in order to increase its performance.

The main concern when designing a hybrid neural network is not to increase the training time of the process too much. Indeed, if the merging is done incorrectly, the training time can become much larger than the sum of the training time needed for a CoNN and a PNN taken separately. However, in some cases, it is possible to constrain this increase by up to only ten percent of the CoNN training time when adding a pruning step.

4.2 Main Algorithms

4.2.1 Cascade Neural Network Design Algorithm

This algorithm will start with a "constructivist" process. However, this first stage will be performed with help of pruning concepts. Indeed, we will watch the saliency of each neuron as we follow the construction. Two reasons motivate this: firstly, this will facilitate the pruning stage coming after, secondly, this constitutes an efficient way to perform weight freezing. Any weight whose saliency does not evolve along the process will be frozen. As we know, this will then accelerate the training stage. The constructive process in this network is different to the ones we have already seen. Firstly, depending on how efficient we want the network to be, we choose a certain number of layers and we initialize them by adding one neuron to it. We train this minimal network with backpropagation, and use the cell division process to make it grow. This means we select the neuron with the highest number of connections and we copy paste it, following the weight update rule given by the expression (4.1). This is one efficient way to reduce the moving target problem as the local environment then changes smoothly.

$$w^1 = (1+\beta)w \qquad w^2 = -\beta w \tag{4.1}$$

w **:** weight vector of the initial neuron
w^1 **:** weight vector of the first new neuron
w^2 **:** weight vector of the second new neuron

Once this is done, the pruning step takes place. Since we already know the saliency of each weight, we can find the least important neurons and prune them. Then, a set of arbitrary weights is unfrozen and we retrain the network. This is done over and over, until the stop criterion is reached.
This algorithm presents an interesting way of performing weight freezing in the CoNN stage, with help of tools generally used for PNN. However, it is not possible to use this algorithm on large neural networks, as it would take too much time to compute the saliency for every neurons at each iteration. This algorithm is thus appropriate for small NNs [Isl00], but other options have to be found in order to build larger networks.

4.2.2 Local weight pruning

Instead of separating the constructive and the pruning step, local weight pruning achieves those two stages at the same time. In fact, as soon as a new neuron is created, weak connections are looked for and deleted. Interestingly, is that we can adapt this procedure to almost all of the existing CoNN algorithms.

We thus choose a CoNN algorithm, for example the tower algorithm, to start the network construction. After having added a new neuron to the network, the algorithm will be trained with help of backpropagation until the MSE reaches an optimum value. Once it has been done, we again compute the saliency, but only for the weights of the newly added neuron. If, compared to the other, one saliency is way weaker, the connection corresponding to it will be pruned. This means that we will only keep the strong connections of each new neuron while going through the process.
This way of performing the pruning is rather efficient [Kho00] as only a few saliencies need to be computed at each step. The training stage duration is thus comparable to the training stage of the CoNN chosen. The leading experiment on this algorithm reported that the pruning step does not increase the final error and, in addition to that, the NN built according to this method shows better generalization capability. However, as the pruning is only done locally without considering the global environment, the solution found is only suboptimal and the achievable rate of the final network will not be increased.

4.2.3 Adaptive Merging and Growing Algorithm

Adaptive Merging and Growing Algorithm (AMGA) is one of the most faithful to nature algorithms created up to now. Indeed, it does not only use the cell division process to create new neurons, but it also chooses by itself when to make neurons divide or when to make them die according to the state of the network. If a part of the network is saturated, that is to say more training will not make it evolve, the algorithm will add neurons in this area. Conversely, if the network shows redundancy in its structure, the algorithm will make the weakest neuron die by melting them with the appropriate strong ones.
Each step of the algorithm begins with an analysis of the network's state. The latter will judge whether cell division or cell merging is needed. To do so, we consider a time dimension that corresponds to the number of backpropagation step used up to this point. As time goes by, we look for stagnating layers, those whose response does not evolve. If we find some, we compute the saliency of the neurons belonging to it and we make the most important one divide. If no cell division is needed, the algorithm looks for redundancy in the network. To do so, weak and strong neurons are first selected. The time dimension enables us to compute the age of each neuron and this value will be used to determine whether a neuron is weak or not, respectively strong or not. Indeed, an old neuron with a small saliency will be judge as weaker than a young neuron with the same saliency, as the young neuron has a higher evolution potential. Once the set of weak neurons and the set of strong neurons are found, we compute the correlation of each pair of neurons, made up of one neuron in each set. If a pair is highly correlated, the neurons belonging to it will be merged. After having performed cell division, cell merging or neither of them, we start again the backpropagation step.

We already saw the way of performing cell division in the CNNDAs' section, but let us finally look at how cell merging is done. Let's say that the correlation C of one weak neuron A and one strong neuron B has exceeded the preset threshold. Then, A and B will be merged to produce the neuron C. This neuron will be connected to all the A and Bs' input neurons and to all the A and Bs' output neurons. The weight update will be done according to the following rules. Two connections going out of the same input neuron to A and B will be replaced by one, having the average weight of the previous connections. Two connections going respectively out of A and B to the same output neuron will be replaced by one connection, having as weight the sum of the previous connections' weight.
This hybrid algorithm has a very efficient training stage [Isl08]. Indeed, as cell division and cell merging processes are used to construct and to prune the network, the moving target problem is avoided. Moreover, the pruning process is not too demanding as only the saliencies of one layer are needed for this stage. However, two main defects remain. The first one is the use of backpropagation as it is a very slow process. The second defect is the size of the parameter set. In fact, we have to configure when a neuron will be considered as weak, when it will be considered as strong, when we will consider that a layer is stagnating and when we will consider that the correlation is large enough to merge a pair of neurons. Finding the optimal set of parameters can thus be very difficult in some cases and may require many attempts.

4.2.4 Hybrid MTiling algorithm

Finally, let us see how a classic CoNN algorithm can be turned into a hybrid algorithm. Taking a CoNN algorithm as a starting point can be advantageous as it enables use of all its specificity during the design. Moreover, in some cases, it is not possible to make abstraction of the constructive stage while designing the pruning stage as some CoNNs lead to special network structure that cannot be pruned without taking precautions.
This is the case for the hybrid MTiling algorithm. As we saw, MTiling considers two types of neurons: the master neurons and the ancillary neurons [Jr.06]. To design the pruning part, we have to take into account that a master neuron cannot be erased; otherwise, we will not be able to achieve the classifying task anymore. On the contrary, there is no restriction on pruning ancillary neurons, as long as the classes stay faithful. To achieve the pruning stage, we will thus design a special saliency function. It will be constructed as follows

S(i) = 1 : if eliminating neuron i renders the layer L unfaithful
S(i) = 1 : if i is a master neuron
S(i) = 0 : otherwise

The main task of the hybrid algorithm will be to achieve the mapping between all neurons and their saliency value. To do so, the algorithm will, at each new iteration,

look for the new master neuron and set its saliency value to one. Then, it will start to look for neurons having zero saliency. Three kinds of neurons belong to this class: dead neurons, correlated neurons and redundant neurons. A neuron is said to be dead when its output is always the same. A correlated neuron is a neuron that always gives the same output as another neuron in its layer. Finally, redundant neurons are the other neurons belonging to the zero saliency class. Unfortunately, the only way to find those neurons is to prune them and see if the layer stays faithful. This hybrid algorithm presents an excellent trade-off between additional training time and performance increase [Par97]. Indeed, the computation of the saliency function only increases the training time up to 10 percent, and the pruning stage can reduce the network by up to 60 percent. Obviously, this way of generating a hybrid algorithm cannot be generalized to any CoNN, and we thus observe the same restriction as the one we had for the tiling algorithm, that is the produced NN will not be adapted to highly nonlinear problems.

4.3 Conclusion

Hybrid algorithms form an interesting way of producing NNs, as they are able to combine the advantages of CoNN and PNN algorithms but to avoid their issues. To be precise, the two main problems solved by hybrid algorithms are the overfitting problems and the difficulties of the initialization. Moreover, NNs built with the help of hybrid algorithms generally show better rates. However, performing a hybrid algorithm necessarily implies a training time increase. Depending on the user expectation, this could be a discriminatory factor for its use.

Chapter 5

A complicate grabbing task

5.1 Scenario formulation

We analyse the usage of CONN and PNN in a robot oriented scenario. More precisely, let us consider two different case studies: the "caution scenario" and the "emergency scenario". In the first one, we want the robot to handle very precious and fragile objects. This means we want the pattern recognition process to be very accurate, no matter its speed. In the second case, we would like the robot to handle urgent grabbing tasks. This could be required if some wounded person urgently needed medical devices. In this case, the pattern classification will be expected to have a high rate but the accuracy will not be as important as previously.

5.2 Constraints and Issues

We are here considering an embedded system. This means that two major constraints will apply. The first one is a memory constraint, that is the robot will not be able to keep a large set of different classification processes in memory. The second constraint affects the computation power: as the power of the robots' processor is limited, over-complex processes will not be able to be executed.
Assuming we want to achieve these tasks with the help of neural networks, two main issues arise while considering how to solve this problem. First, as the two goals we want to reach are in contradiction, it will be difficult to perform the pattern recognition with a unique network: either we will have to build as many networks as there are users' expectations, or we will have to think of an evolving network, that is a network changing its structure according to the task to achieve. As the scenario imposes memory restrictions, having more than one network is not an option. We now face the second issue: building evolving networks does not seem possible either, as the neural networks' structure is fixed after the training time and it is not possible to change it without retraining the network. Let us see how to overcome those problems with help of constructive, pruning and hybrid algorithms.

5.3 Solution design

As we know, CoNN, PNN and hybrid algorithms belong to the so called adaptive systems. Those algorithms will start their construction with an arbitrary network and will adapt it to any given task. In our scenario, using those algorithms could be a way to overcome the problem of multiple objectives. Indeed CoNN, PNN and hybrid algorithms do not request any information about the task we want to solve. If the task has a high complexity, the network will automatically have the appropriate size and structure and vice versa. However, once a neural network is trained, it cannot be changed. This implies that if the NN is designed for the "caution scenario" it will not be able to handle the "emergency scenario". This leads us to the following conclusion: we will have to reconstruct the network at the beginning of each task.
To solve the case study, we can thus imagine the following process. A set of adaptive algorithms will be implemented in the robot. This set will be made up constructive, pruning and hybrid algorithms. Moreover, an oversized neural network will be registered in the robots' processor. As a new task occurs, the robots will first analyse its type. According to this, the robot will choose the most appropriate algorithm to use in order to construct the recognition solution. If the latter has to be fast but not so precise, the robot will choose a pruning algorithm. Indeed, this algorithm constructs neural networks quickly and those are themselves quick in terms of rates. The registered oversized network will be pruned until an optimal size, preset by the user. If the recognition solution has to be very precise, the constructive algorithm will this time be used. Neurons will be added to the network until it reaches the size limitation or the required precision. However, this solution may lead to a complex NN that will not be usable by the robot processor. To prevent this, a hybrid algorithm will be used in this case, as we know hybrid algorithms give equal performance but faster and less complex networks.

5.4 Proof concept

To study the feasibility of this concept, we should mainly focus on the speed of the algorithms used. This question is rarely studied in the literature, as the research focuses on the final neural network rather than on the optimization of the algorithms that produce them [Par00]. The ability to construct algorithms, that need a training time of the same order of magnitude as the execution time of the NN, is crucial. Finding the right balance between algorithm simplicity and powerful NN is another avenue that should be explored. Indeed, the algorithm shouldn't be too complex as the computation power is limited. However, this complexity is needed to generate powerful NNs and therefore, the algorithm shouldn't be too simplified.
Assuming we find solutions to those issues, we can expect such a process to take advantage of the diversity of adaptive algorithms. The main disadvantage is that a retraining is needed at each new task. However, overcoming this greatly extends the range of achievable tasks and therefore the robot's possibilities.

Chapter 6

Conclusion

Since the Eighties, artificial neural networks have attracted a lot of researchers as they show a high potential for solving complex tasks. Mainly used for system modeling and classification, the difficult part in their construction is to choose the appropriate topology given the problem to solve. Adaptive systems exactly focus on this point. With CoNN and PNN algorithms, we can build a neuron network without any previous knowledge of the problem they will have to solve.
Two different ways of doing so exist. Either we start from a minimal network and we add neurons to it as we go along the algorithm, or we initialize the algorithm with an oversized network and we prune it by deleting less important neurons. Those two approaches lead to several difficulties sometimes hard to overcome. However, it is interesting to note that the weaknesses of one algorithm are the strengths of the other. Indeed, as the main problem of constructive neural networks is the overfitting, a strength of PNNs is their generalization ability.
Knowing this, a way has been found to significantly reduce the impact of those problems. The idea is simple: combine the two previous algorithms. This leads to hybrid algorithms. They demand more computational power but provide NNs of higher quality. Finally, depending on the project constraints, one will be able to choose the most efficient solution to the problem from those presented.

Bibliography

[Cos02] Marcelo Azevedo Costa. Constructive and pruning methods for neural network design. IEEE, 2002.

[Isl00] Md. Monirul Islam. An algorithm for automatic design of two hidden layered artificial neural networks. IEEE, 2000.

[Isl08] Md. Monirul Islam. An adaptive merging and growing algorithm for designing artificial neural networks. IEEE, 2008.

[Jr.06] J.R. Bertini Jr. Two variants of the constructive neural network tiling algorithm. IEEE, 2006.

[Kho00] K. Khorasani. Input-side training in constructive neural networks based on error scaling and pruning. IEEE, 2000.

[Li10] Zhen Li. Some classical constructive neural network and their new developments. IEEE, 2010.

[Par97] Rajesh Parekh. Pruning strategies for the mtiling constructive learning algorithm. IEEE, 1997.

[Par00] Rajesh Parekh. Constructive neural network learning algorithms for pattern classification. IEEE, 2000.

[Sha10] Sudhir Kumar Sharma. Constructive neural networks: A review. IEEE, 2010.

[T.K] T.Kowaliw. Artificial neurogenesis: An introduction and selective review. IEEE.

[Yan06] Zhu Yan. Mw-obs: An improved pruning method for topology design of neural networks. IEEE, November 2006.

License