Modified Cascade Correlation Neural Network and its Applications to Multidisciplinary Analysis Design and Optimization in Ship Design

Adeline Schmitz, Frederick Courouble,
Hamid Hefazi and Eric Besnard
California State University, Long Beach
USA

1. Introduction

Artificial Neural Networks (NN) basically attempt to replicate functions of the human brain by connecting neurons to each other in specific manners. In most cases, the number of neurons and connections has been limited to tens or hundreds of neurons with a similar order of magnitude of connections between them. This contrasts with the human brain which has many orders of magnitude more neurons and also many more connections between them. The work done so far in this field can therefore be categorized as precursory and the potential of this technology has yet to be fully realized. Jain & Deo (2005) present a survey of applications of NNs in ocean engineering. They have been used for predicting environmental parameters (wave heights, sea level, wind speeds, tides, etc.), forces and damage on ocean-going structures, and ship and barge motions, in various ship design applications and more. Gougoulidis (2008) presents an overview of utilization of neural networks in many other marine applications including structures, stability, propulsion and seakeeping. All of the applications that are reviewed, except one, are for predictions of characteristics of ships.
In the area of ship design, NNs have been employed for various purposes. For example, Koushan (2003) presents a hull form optimization employing NNs in which eight hull form parameters are varied in order to minimize resistance. Similarly, the use of NNs in hull shape optimization by Danõşman *et al.*(2002) and Koh *et al*. (2005) allows for a more advanced flow model (panel method instead of thin-ship flow theory) and thus leads to improved shapes. Koh (2004) present a similar approach for investigating resistance, manoeuvrability and seakeeping characteristics of a high speed hull form. In Mesbahi & Bertram (2000), and Bertram & Mesbahi (2004), NNs are used to derive functional relations between design parameters for cargo ships as an alternative to using charts. In Maisonneuve (2003), several NN application examples from the industry are discussed, showing state-of-the-art of European research in marine design: integrating real calculations (using CAD

model) and artificial calculations (using NNs as response surface methods) to perform single- or multi-objective optimizations.

In most of the applications above, a single hidden layer feed-forward type of network and back-propagation are used. Also, the number of input nodes used by the different investigators was relatively small, on the order of one to ten. Very few applications used a large number of inputs to utilize the real power of the neural networks. One exception is Danõşman *et al*. (2002) which used a single, hidden layer NN with 40 input and four output parameters. The inputs represent a series of half breadths of the aft end of a catamaran and the output parameters are related to wave resistance, wave elevation and displacement. The network used back-propagation with a training set of 300 and a validation set of 50 points. The number of hidden units used and the errors produced by the neural network are not discussed in the paper, however.

As the number of parameters to be varied increases, training the network becomes more and more challenging. It is demonstrated that feedforward NNs have universal approximation ability for a wide variety of functions classes, provided that a sufficient number of hidden units are available (Cybenko, 1989, Hornik, 1991). The approximation ability of a particular network depends on the numbers of input and output units, the number of training cases, the amount of noise in the targets, the complexity of the function to be learned, the actual architecture of the network, the type of hidden unit activation function and the training algorithm. This often leaves NN users having to determine the network size by trial and error. Also, back-propagation, the most commonly used algorithm to train single hidden layer feedforward NNs, is known to be very slow for large input spaces. Other network structures and training algorithms should be investigated.

This chapter presents an alternative NN structure based on a constructive network topology and a corresponding training algorithm suitable for large number of input/outputs to address the problems where the number of design parameters is fairly large, say up to 30 or even more.

The chapter is divided into four sections. First, the use of NNs as advanced regression models in the ship design cycle is reviewed and the choice of the particular topology (constructive network) and training algorithm (modified cascade correlation, or "MCC") of the NN is justified.

The next section describes in detail the MCC. This algorithm is an improvement from the original cascade algorithm introduced by Fahlman and Lebiere (1990). Improvements include altering the weight initialization, modifying the candidate hidden unit training, and introducing normalized inputs, "early stopping" and "ensemble averaging."

Next, the NN approach is applied to the design/optimization of an underwater hull configuration using a genetic algorithm search method. Results are compared with those obtained with a classical optimization approach in which the CFD code is directly coupled with the optimizer.

The last section presents a NN-based performance analysis and optimization of sailing configurations of an America's Cup class yacht. The objective is to maximize boat speed (objective function) by varying the sailing setups (design variables). In the approach, the experimental data from the sailing records provided by sensors is used to train an MCC neural network. The network is coupled with a genetic algorithm to determine the maximum boat speed and corresponding yacht settings at various wind speeds. Because the majority of the data is gathered in a small region of the search space corresponding to a valid set of sailing configurations, the remaining regions of the domain are not well populated and can lead to training errors for the neural network. The chapter presents an automatic method to fill the domain of investigation by adding artificial points to the database in regions without sufficient experimental data.

2. Neural Networks as Response Surface Methods in the Design Cycle

2.1 NNs in the Design Cycle

The systems engineering approach, originated and widely used in the aerospace industry, consists of decomposing a system into subsystems. For a ship, those would correspond to the hull forms definition, propulsion, structure, payload, etc. This system decomposition approach is described, for example, in Blanchard & Fabrycky (1997), and can be applied at the ship level as well as at subsystem levels in the systems architecture. For example, this systems approach may also be used to design a propulsion subsystem which will be integrated into the ship, based on requirements established at higher levels. In other words, every *component* may be looked as a system which gets integrated into a *system of systems*.

The analyses performed at each subsystem level rely, in general, on a combination of semi-analytical models, advanced numerical methods such as finite element analysis, and use of existing databases. The modern approach used in the design of such systems usually includes optimization at some level.

Neural networks may be inserted directly at all levels of the system design process and on a broad basis. Specialists who use advanced computational tools for detailed analyses are often remotely connected to the design loop. The use of NN allows for them to be indirectly integrated very early into the design cycle by generating a computational database representative of the problem at hand over the desired design space. For example, the database might consist of a few hundred CFD analyses performed for a configuration represented by tens of widely varying design parameters. This database can then be used to train –hence its name, "training set"– a neural network which is then inserted in the design loop (Fig. 1). At this point, the designer (not the analyst) can use the NN and get a solution for a variety of designs in a fraction of a second. For example, if a network has been trained for estimating the ship resistance, the latter can be obtained by the designer for any desired point in the design space with minimal computational time.

Similar uses of neural networks can be made when dealing with available large databases. Such databases may be from one or more sources, numerical and/or experimental. In this case, the database can be used directly to train the NN and the latter can also be integrated into the design loop (Fig. 1).

The result is a design approach in which the function, such as ship resistance, corresponding to a particular set of design variables either selected by the designer or by the computer ("design tool"), can be obtained instantaneously.

In practical terms, the introduction of NN allows for extraction of time-consuming or difficult operations (performing an advanced numerical analysis or extracting information from a large and evolving database) from the design loop while still keeping their influence on the outcome of the design process via the NN. The cost has thus been moved (and possibly reduced in the process) to the training set generation (if it was not already available) and to the training of the network.

The result is a NN which can estimate the function or functions over the design space it has been trained on. This ability to quickly evaluate new designs allows in turn for the use of global optimization tools such as genetic algorithms instead of having to rely on local optimization methods or exploring a restricted part of the design space.

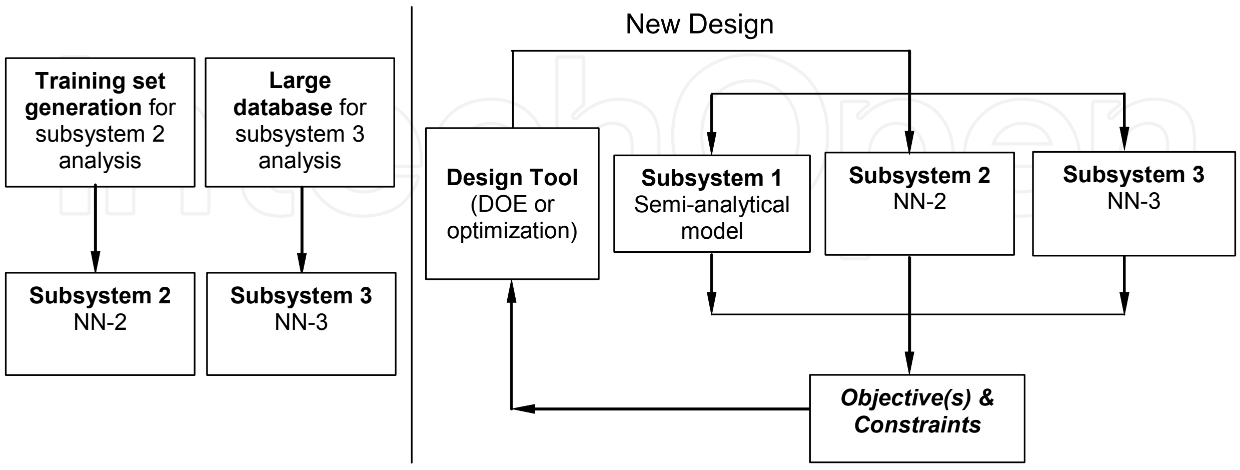
Training set generation for subsystem 2 analysis
Large database for subsystem 3 analysis
Subsystem 2 NN-2
Subsystem 3 NN-3
New Design
Design Tool (DOE or optimization)
Subsystem 1 Semi-analytical model
Subsystem 2 NN-2
Subsystem 3 NN-3
Objective(s) & Constraints

Fig. 1. System design loop utilizing NNs. The NNs are generated outside the design loop based on computationally extensive models and/or large databases.

2.2 Advantages Offered by Constructive Neural Networks

NNs have been investigated for many applications outside the field of ocean engineering that require a large number of function analyses, ranging from chemistry (Agatonovic-Kustrin *et al.*, 1998, and Takayama *et al.*, 2003) to structural analysis (Deng *et al.*, 2005). Research clearly indicates that NNs compare favourably with classical RSM (Gougoulidis, 2008, Todoroki *et al.,* 2004, Bourquin *et al.*, 1998, Gomes & Awruch 2004, Dutt *et al.*, 2004 and Lee & Hajel, 2001), in particular when the function is non-convex over the desired domain and the function may be highly nonlinear. In addition, for problems with a large number of inputs (design variables), the size of the dataset required for the classical RSM rapidly grows. Schmitz (2007) and Besnard *et al.* (2007) present a survey of the different RSM available and demonstrate the clear advantage in using NNs in this type of application.

As discussed above, for most marine applications reported to date, however, NNs employed either a fixed topology and/or back propagation for training. The former implies that one needs to have some information about the function to approximate and the latter leads to increasingly large CPU time requirements for training in the case of a large number of inputs.

Methods which use a fixed network topology involve evaluating in advance (before training) the type of network that would best suit the application (how many neurons, how many hidden-layers) to match the complexity of the NN to that of the function. This point is best illustrated by Kwok & Yeung (1997a): "*Consider a data set generated from a smooth underlying function with additive noise on the outputs. A polynomial with too few coefficients will be unable to capture the underlying function from which the data was generated, while a polynomial with too many coefficients will fit the noise in the data and again result in a poor representation of the underlying function. For an optimal number of coefficients the fitted polynomial will give the best representation of the function and also the best predictions for new data. A similar situation arises in the application of NN, where it is again necessary to match the network complexity to the problem* *being solved. Algorithms that can find an appropriate network architecture automatically are thus highly desirable*."

There are essentially two approaches for training multilayer feedforward networks for function approximation which can lead to variable networks (Kwok and Yeung 1997a). The Pruning Algorithms start with a large network, trains the network weights until an acceptable solution is found, and then uses a pruning method to remove unnecessary units or weights (units connected with very small weights). On the other hand, Constructive Algorithms start with a minimal network, and then grow additional hidden units as needed. The primary advantage of constructive algorithms versus pruning algorithms is that the NN size is automatically determined. Constructive algorithms are computationally economical compared to pruning algorithms which spend most time training on networks larger than necessary. Also, they are likely to find smaller network solutions, thus requiring less training data for good generalization. They also require a small amount of memory because they usually use a "greedy" approach where only part of the weights is trained at once, whereas the remaining part is kept constant (Schmitz 2007).

Cascade-Correlation, first introduced by Fahlman & Lebiere (1990), is one such supervised learning algorithm for NNs. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one-by-one in a cascading manner. This architecture has several advantages over other algorithms: it learns very quickly; the network determines its own size and topology; it retains the structure it has built even if the training set changes; and it requires no back-propagation of error signals through the connections of the network. In addition, for a large number of inputs (design variables), the most widely used learning algorithm, back-propagation, is known to be very slow. Cascade-Correlation does not exhibit this limitation (Fahlman & Lebiere 1990).

3. Modified Cascade Correlation Neural Networks

As mentioned above, the constructive Cascade-Correlation algorithm begins with a minimal network consisting of the input and output layers and no hidden unit (neurons), then automatically trains and adds one hidden unit at a time until the error (E_p) between the targets (f_p) of the training set and the outputs from the network ($f_{NN,p}$) reaches a desired minimal value. Thus, it self-determines the number of neurons needed as well as their connectivity or weights.

The original algorithm of Fahlman & Lebiere (1990) was geared towards pattern recognition and has been improved to make it a robust and accurate method for function approximation (Schmitz *et al.*, 2002, Schmitz, 2007). Although the definitions used here assume that the NN has a single output, i.e. that the NN represents a single scalar function such as ship resistance, the process is easily extended to networks with multiple outputs (Schmitz *et al.*, 2002). In the applications considered in this paper, multiple functions, such as ship resistance and ship displacement, are each represented by a separate network, i.e. each uses multiple single-output networks rather than a single multiple-output network. It was found during the course of the study that it was more advantageous to train multiple single-output networks than one large multiple output network in terms of error on an unseen dataset (generalization error) and computing time (since multiple "single output" networks could be trained simultaneously faster than one single "multiple output" network.

3.1 Base Algorithm as Introduced by Fahlman and Lebiere

The basic algorithm as introduced by Fahlman and Lebiere (1990) includes the following 11 steps:
Step 1: Start with the required input and output units; both layers are fully connected. The number of inputs and outputs is dictated by the problem.
Step 2: Train all connections ending at an output unit with a common learning algorithm until the squared error *Es* of the NN no longer decreases.

$$Es = \frac{1}{2}\sum_{p=1}^{Np}\left\| y_p - t_p \right\|^2 = \frac{1}{2}\sum_{p=1}^{Np}\sum_{i=1}^{m}\left[y_{i,p} - t_{i,p} \right]^2 \quad (1)$$

Here *m* is the size of the outputs (or number of outputs), *Np* is the size of the training set, $y_{i,p}$ is the ith output from NN, and $t_{i,p}$ is the corresponding target.
Step 3: Generate a candidate unit that receives trainable input connections from all of the network's external inputs and from all pre-existing hidden units (if any). The output of this candidate unit is not yet connected to the active network (output).
Step 4: Train the unit (its weight) to maximize the correlation referred to as S_C (Eq. 2). Learning takes place with an ordinary learning algorithm; training is stopped when the correlation score no longer improves. The correlation formula is given by

$$S_C = \sum_{i=1}^{m}\left| \sum_{p=1}^{P\max}\left(z_{o,p} - \overline{z_o} \right)\left(E_{i,p} - \overline{E_i} \right) \right| \quad (2)$$

Here $z_{o,p}$ is the output of the candidate hidden unit and $E_{i,p}$ is the residual error of the outputs calculated at Step 2, $E_{i,p}=y_{i,p}-t_{i,p}$. The bar above a quantity denotes the average over the training set.
Step 5: Connect the candidate unit with the outputs and freeze its input weights. The candidate unit acts now as an additional input unit.
Step 6: Train again the input-outputs connections by minimizing the squared error *Es* as defined in Step 2.
Steps 7 to *10*: Repeat Steps three to six adding one hidden unit at a time.
Step 11: Stop training when the error *E* of the net falls below a given value, ε.
Instead of a single candidate unit, it is possible to use a pool of candidate units, each with a different set of random initial weights. All receive the same input signals and see the same residual error for each training pattern. Because they do not interact with one another, or affect the active network, they can be trained simultaneously. Only the candidate whose correlation score is the best is installed. The use of a pool of candidates greatly reduces the chances that a useless unit will be permanently installed because an individual candidate got stuck during training. Fahlman and Lebiere (1990) typically show that four to eight candidate units are enough to ensure good candidates in each pool.
Steps 2 and 4 require the use of an optimization routine. Fahlman uses the so-called Quickprop algorithm. Quickprop computes the derivative of the error with respect to the weights as in standard back-propagation, but instead of simple gradient descent, Quickprop uses a second order method, related to Newton's method, to update the weights (Fahlman, 1988).

3.2 Overview of Algorithm Modifications for Function Approximation

While the basic CC algorithm described in the previous section provides a good foundation for regression applications, it also has areas which can be improved. This section presents the modifications that were developed and implemented to the CC algorithm described above. Fahlman introduced this algorithm for classification tasks which typically use a large number of inputs. The network is thus well suited for the application in mind in this research, i.e. approximation of functions with a large number of variables. Practical optimization problems require a large number of design variables which define the configuration to be optimized. CC algorithm learns very quickly and uses minimal computer memory as it only trains some of the network weights while others are frozen. It has been shown to work well for regression tasks (Kwok & Yeung, 1993, 1997a and 1997b, Prechelt, 1997, Treagold & Gedeon, 1999, Lehtokangas, 1999, Lahnajärvi *et al.*, 2002). Each author, however, points out some potential downfalls of the algorithm and proposes some possible fixes. These problems are primarily:

- Maximizing Fahlman’s correlation formula trains candidate neurons to have a large activation (weight) whenever the error at their output is not equal to the average error. Cascade correlation has a tendency to overcompensate errors. (Prechelt, 1997)
- The candidate unit weight optimization might get stuck in a local maximum, and thus units which are not correlating well with the error are installed on the NN, leading to more units than necessary to reach the desired level of accuracy (deeper network) (Lehtokangas, 1999, Kwok & Yeung, 1997b, Lahnajärvi *et al.*,2002).
- Cascading units can result in a network that can exhibit very strong non-linearities, thus affecting generalization (Kwok & Yeung 1993, 1997a).

The critical issue in developing a neural network for regression tasks is generalization: how well will the network make predictions for cases that are not in the training set? Neural networks, like other nonlinear estimation methods such as kernel regression and even linear methods like polynomial regression, can suffer from either underfitting or overfitting. As stated in Sarle (2002): “*A network that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. A network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of NNs.*”

Model selection plays an important role in the generalization ability of the network. As explained above, constructive methods, like cascade correlation, are usually better methods than fixed network topologies trained with back-propagation or pruning methods because they automatically find the number of hidden units that matches the complexity of the problem. They also find smaller network solutions. Smaller networks mean less connections or weights to adjust and thus usually require smaller training sets for similar generalization ability (Kwok & Yeung, 1997b).

The generalization ability of the NN has been addressed in the Modified Cascade Correlation algorithm. Various improvements to the original CC based on some of the solutions proposed by the above-referenced works have been implemented. Extensive research has been conducted that demonstrates the clear advantages of using the MCC on a test function for dimensions varying from 2 to 30 inputs (Schmitz 2007).

In the MCC, inputs and outputs to the NN are non-dimensionalized, weights are constrained to a maximum value in order to limit strong non-linearities of the response surface, training of the input-to-hidden-unit weights and hidden-to-output weights is performed with a second order optimization method (Sequential Quadratic Programming) instead of the Quickprop algorithm introduced by the original authors. Several stopping criteria are also available to limit overfitting of the network; they all continue training slightly past the minimum validation error (error measured on the Validation Set) and the resulting network is that which has the smallest squared error on the VS, whereas the original authors stop when the error on the training set reaches a predetermined value. Also, ensemble averaging, a well known technique for reducing overfitting is available when training with the MCC. These improvements are described in more detail in the following sections.

3.2.1 Normalization of Inputs

The activation function of the hidden units is usually highly nonlinear. In this research, the activation function chosen is the sigmoid function which varies between zero and one.
During training, weights are initialized with small random values. It is commonly known that optimization algorithms will perform faster if optimization is started in an area where the objective function varies rapidly. It is thus better to ensure that the hidden units are not in their saturated portion but rather in the area of the sigmoid which is quasi-linearly varying when optimization is started. Along with using small initial weights, it was also decided to normalize the inputs to the NN. The training set minimum and maximum values are first calculated for each input *i*.

$$MinInput_i = \min_{1 \le p \le Np}\left(z_{ip}\right) \tag{3}$$

$$MaxInput_i = \max_{1 \le p \le Np}\left(z_{ip}\right) \tag{4}$$

where *Np* is the size of the training set and z_{ip} is the ith input for the pth point of the training set.
The training set is next rescaled from zero to one according to the equation below.

$$\begin{bmatrix} z_{1p} \\ \vdots \\ z_{np} \end{bmatrix} = \begin{bmatrix} \dfrac{z_{1p} - MinInput_1}{MaxInput_1 - MinInput_1} \\ \vdots \\ \dfrac{z_{np} - MinInput_n}{MaxInput_n - MinInput_n} \end{bmatrix} \quad \left(for\ p \in \{1...Np\}\right) \tag{5}$$

Also the validation and generalization sets are rescaled using the same minimum and maximum values found for the training set, so they will vary from around zero to one (but not exactly between 0 and 1).

3.2.2 Weights Initialization

In any nonlinear optimization problem, the initialization of the parameters has an important influence on the ability of the training program to converge and the speed of that convergence. The training of weights in NNs can be viewed as a nonlinear optimization problem in which the goal is to find a set of network weights that optimizes a cost function. In the MCC algorithm, there are two separate optimization problems. The first one is to maximize a correlation function to train the candidate hidden unit newly added to the network; the other is to minimize the error on the training set. Both describe a surface in the weight space. Training algorithms are simply methods used to find the minimum of this surface. The complexity of the search is governed by the nature of this surface. Error surfaces for multilayer NNs have typically many flat regions where learning is slow and long narrow "canyons" that are flat in one direction and steep in the other directions. This makes it very difficult to search the surface efficiently using gradient-based routines. In addition, the cost function is characterized by a large number of local minima with values in the vicinity of the best global minimum. The efficiency of the search method depends much on the initial weight distribution. The simplest category among the weight initialization methods is random weight initialization. It is commonly known that if all the weights of an NN are initialized with a zero, they cannot change to any other value during training if some simple training algorithms are used. Random initialization has been proposed to avoid this undesired situation and its ability to break the symmetry. Very little research has been reported on weight initialization in the literature (Lehtokangas, 1999 and Lahnajärvi *et al.*, 2002).

In the cascade correlation algorithm, there are three separate weight optimization problems to investigate:

1. The first optimization problem is the squared error minimization between input and output units before any hidden unit is added to the network. A previous study showed that the weights between the inputs and the outputs should be initialized randomly between -0.5 and +0.5 (Hefazi *et al.*, 2003).
2. The second optimization problem consists of finding the best candidate hidden unit by maximizing the correlation formula. The weight initialization consists in calculating the norm of the input vector $\mathbf{Z}_p$, $\|\mathbf{Z}_\mathrm{p}\|$, for each training set point p (also called pattern), and then initializing the weights for the new candidate unit w_j so that:
3.
$$\|\mathbf{w}\| = \sum_{j=1}^{n+h+1} w_j^{\,2} \leq 4 * \max_{p=1,\dots,Np} (\|\mathbf{Z}_\mathrm{p}\|) . \tag{6}$$

   Also during the optimization, the weights values are limited between -10 and +10, as lower values lead to very deep networks and higher values lead to severe overfitting of the data.
4. A third optimization consists of minimizing again the squared error after a new hidden unit (hth hidden unit) has been added to the network and connected to the outputs. Hefazi *et al.* (2003) showed that the weights v_{ij} found at the previous step (unit *h-1*) are already close to the optimal value with this new hidden unit added to the network. For this weight initialization problem, it is then best to use, as initial weights, the ones found at the previous iteration (weights between the inputs and previous hidden units to hidden unit *h-1*) and to initialize at zero the new weights between the inputs and previous hidden unit to hidden unit *h*. This method leads to the fastest search for the

optimum as well as the smallest overfitting. Similarly, the weights are allowed to vary only between -10 and +10 during optimization.

3.2.3 Choice of optimization routine

The weight optimization must be solved by using some optimization software. Kwok & Yeung (1993) demonstrate that the CC algorithm can always reach $Es<\varepsilon$ for a given $\varepsilon>0$ for L^2 functions, even when using a local optimizer. Fahlman (1988) uses its own optimization routine to update the weights; the Quickprop algorithm, a second order local search method related to Newton's method. In this research, a commercially available software DOT, developed by Vanderplaats (1995) was chosen. This software contains a choice of the latest state-of-the-art optimization methods. Therefore the Broydon-Fletcher-Goldfarb-Shanno (BFGS) method from DOT software was chosen for its proven efficiency and accuracy for unconstrained optimization problems. It is also a quasi-Newtonian method because it creates an approximation of the inverse of the Hessian matrix. The details of the method are explained in Vanderplaats (1995). One advantage of using this software is that the gradient of the squared error and the correlation formula can be supplied directly to DOT and, thus, considerably speed up the weight optimization.

3.2.4 Candidate Hidden Unit Training

Fahlman's original algorithm calls for randomly initializing a pool of four to eight candidate units and then maximizing all candidate units. The candidate whose correlation score is the highest is then added to the network. This is done because, as explained in 0, the weight surface has many local maxima. And since the method used for finding the best weights is a gradient search, i.e. local search, the optimization may get stuck in a local maxima and fail to find the global one. So doing several searches starting with different initial weight values increases the chance of finding the "global" optimum. One might want to use a global search method, but this becomes prohibitive in terms of computer time requirements. Another idea is to use a much larger pool, of the order of 100-500 candidates, initialized at random and then only optimizing the one whose correlation after random initialization is best. Random initialization is very fast, and increases the chance of starting the optimization with a unit close to the global optimum and only one candidate is trained using the time consuming optimization algorithm. Lehtokangas (1999) has applied this method successfully to his constructive algorithm and found it beneficial in terms of time requirements and performance of the NN. Both options are implemented in the algorithm. A study in Schmitz (2007) shows that for a number of inputs greater than 5 or 10, it is advantageous to use the method using a large pool of candidate units.

3.2.5 Stopping Criterion

When training a NN, one is usually interested in obtaining a network with optimal generalization performance. Generalization performance means small errors on examples not seen during training. As hidden units are added to the network, the error on the TS decreases, i.e. the network is able to fit the training data better. However, when looking at the error on an unseen data set, the error initially decreases but at some point during training it increases. The network starts to overfit the training data and the generalization ability of the network gets worse. This is even more pronounced when the data is noisy

(Bishop, 1995). This phenomenon is called the bias variance tradeoff; underfitting produces excessive bias in the outputs, whereas overfitting produces excessive variance. To our knowledge, Fahlman and Lebiere did not study the generalization properties of their CC network and looked only at the convergence of their network on the training data.
An easy way to find a network having the best performance on new data is to evaluate the error function using data which is independent of that used for training, i.e. on the validation set (VS) and to stop training when the error is minimum on the VS. This method is called *early stopping* or stopping the learning procedure before full convergence of the network on the training set (TS) to obtain optimal generalization properties. It is widely used in all feedforward NN architectures. Because of the stochastic nature of the CC algorithm, the minimum validation error might exhibit several local minima as hidden units are added one by one before the global minimum can be attained. This implies that one must continue training the NN past each local minimum to make sure that the global minimum has been found and then choose the network with the number of hidden units which correspond to this minimum validation error. Prechelt (1998a & 1998b) has derived several classes of stopping criteria which may be used to determine how long training should be continued to make sure that the global minimum has been found. These criteria are described in detail in Schmitz (2007). Only the criterion used in the applications presented in this chapter is described in Section 0.

3.2.6 Ensemble Averaging

Because of the stochastic nature of the process in building NNs, it is a common practice to train many different candidate networks and then to keep only the one with best performance. Each network leads to different weight values, different numbers of HUs and different errors. Usually, the one with the best performance is chosen. Performance is usually measured by how the network predicts data on an independent validation set. There are two disadvantages in this approach. First, the effort involved in training the remaining networks is wasted. Second, the generalization performance on the validation set has a random component since it is a relatively small set and so the network which had best performance on the validation set might not be the one with the best generalization, i.e. performance on the rest of the computational domain. These drawbacks can be overcome by combining the networks together by forming a committee. There are several ways of combining networks; one simple way is to take the output of the committee to be the average output of each individual network. This method, called *ensemble averaging,* appears to be a very simple way to limit the overfitting of the network. According to Bishop (1995), the error on the committee is always less than the average error calculated by averaging the error on each individual network. Also, networks trained with the CC algorithm usually show strong non-linearities in their response because the hidden units are added in cascade, making a network with many layers and one hidden unit in each layer. In the application in mind involving approximation of smooth functions, the idea of using all the networks constructed, and average them out to “smooth out” the response surface, appears promising. Tekto & Villa (1997) have done some preliminary research on ensemble averaging combined with early stopping (ESE) for NNs trained according to the cascade correlation algorithm for simple single input/single output functions. Their work shows that the technique they call ESE provides an improvement in the generalization ability of the network for those test cases. An extensive study in Schmitz (2007) has shown that ensemble averaging always improves the generalization ability of the NN and should indeed be used each time an NN is constructed with the MCC algorithm.

3.3 Equations/Mathematical Formulation/Algorithm

This section describes the mathematical formulation the modified CC algorithm for function approximation. The training algorithm was programmed in C++ language and coupled with the DOT software which uses FORTRAN language.

3.3.1 Step 1: No Hidden Units/Linear Inputs to Outputs Connection

In the first step of building the network with the cascade correlation algorithm, there are no hidden units. Inputs and outputs are fully connected, the weights, v_{ij}, determine the strength of the connection from the ith input to the jth output. These will need to be adjusted to minimize the squared error *Es*. Fig. 2 shows a schematic of the input-outputs connections without hidden units. The vertical lines sum all incoming activation. X connections correspond to weights to be trained. Square boxes represent neurons; input-output neurons have a linear activation function.

The bias term can be modelled in the equations as an additional input unit of value one and with weighted connections to the outputs. Without loss of generality, the output neurons can have a linear activation function of slope one (i.e. $\varphi(z)=z$) because there is always a linear component to a nonlinear function, thus a linear link between inputs and outputs. This greatly simplifies the equations for training the NN. The bias parameter is useful to compensate for the difference between the mean (over the training set) of the output vector and the corresponding mean of the target data.

Based on this NN, the relationship between input and output is given by

$$\left(\forall p \in \{1 \ldots Np\}\right) \quad \begin{bmatrix} y_{1p} \\ \vdots \\ \vdots \\ y_{mp} \end{bmatrix} = \begin{bmatrix} v_{11} & \cdots & \cdots & v_{1,n+1} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ v_{m1} & \cdots & \cdots & v_{m,n+1} \end{bmatrix} \begin{bmatrix} z_{1p} \\ \vdots \\ z_{np} \\ +1 \end{bmatrix} \tag{7}$$

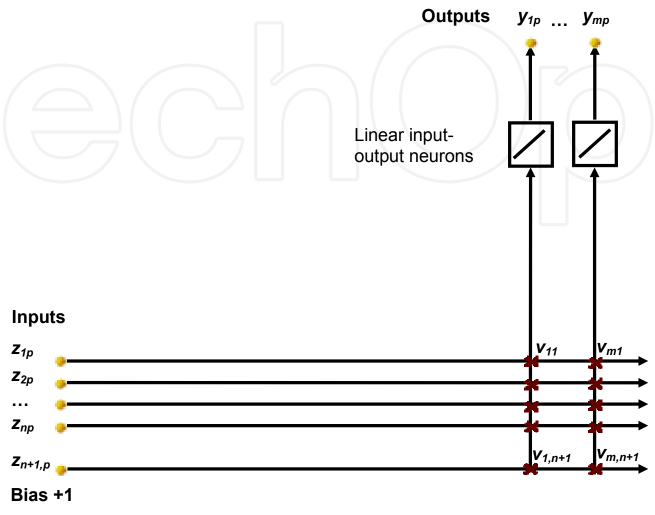

Fig. 2. Schematic of input-output connections without hidden units. The vertical lines sum all incoming activation. X connections or weights must be trained.

3.3.2 Step 2: Minimize Squared Error without Hidden Unit

The next step consists of adjusting the weights to fit the training data. The squared error between the targets and the outputs is used as the standard error measure and must be minimized. The squared error for the neural network without hidden unit is given by the following equation:

$$Es = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left\| y_{ip} - t_{ip} \right\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left[\sum_{j=1}^{n} v_{ij} z_{jp} + v_{i,n+1} - t_{ip}\right]^2 \tag{8}$$

The weights v_{ij} are initialized randomly between [-0.5, +0.5] as discussed in Section 0. The error is then minimized by adjusting the weights using the BFGS method from DOT optimization software (Vanderplaats, 1995). DOT allows the user to directly input the gradient of the function to optimize, if known, and to speed up the optimization process. The gradient of squared error with respect to the weights $\partial Es/\partial v_{kl}$ can be calculated analytically as follow:

$$\frac{\partial Es}{\partial v_{kl}} = \sum_{p=1}^{Np}\left[\sum_{j=1}^{n} v_{kj} z_{jp} + v_{k,n+1} - t_{kp}\right] z_{lp} \tag{9}$$

with $k \in \{1\ldots m\}$, $l \in \{1\ldots n+1\}$ and $j \in \{1\ldots n\}$.
It is noteworthy to point out here that a pseudo inverse method could also be used to find the minimum error since it is a linear system. However BFGS works fast on linear systems and is subsequently used to find the candidate units weights once hidden units have been added and the system is no longer linear. This approach was used here instead of calculating the pseudo inverse matrix.

3.3.3 Step 3: Adding a First Hidden Unit Connected to Inputs only

After optimizing the matrix of weights, **V**, a first hidden unit is connected to the inputs as shown in Fig. 3. Its output is noted $z_{n+2,\,p}$.

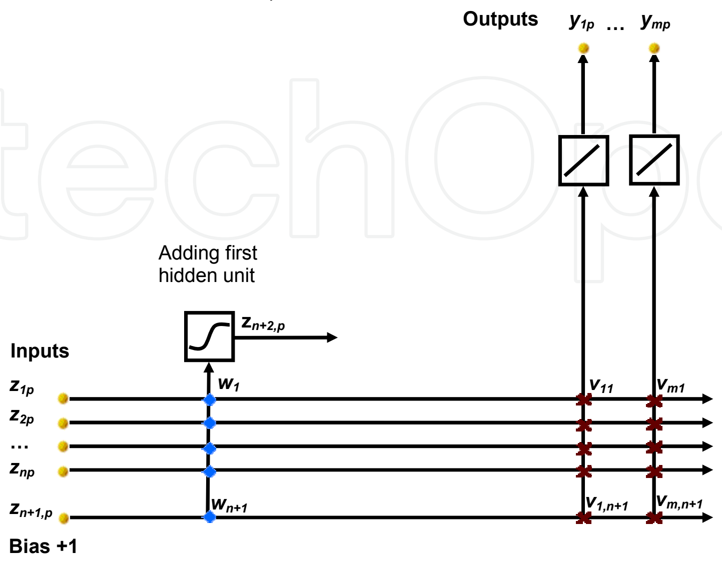

Fig. 3. Schematic of input to first hidden unit connections. Hidden unit not yet connected to outputs. Diamond connections to be trained.

The connections (weights) between inputs and the hidden unit are noted w_j in

$$z_{n+2,p} = \sigma\left(\left[w_1 \cdots w_n w_{n+1} \right] \begin{bmatrix} z_{1p} \\ \vdots \\ z_{n+1,p} \end{bmatrix} \right) = \sigma\left(\sum_{j=1}^{n+1} w_j z_{jp} \right) \tag{10}$$

where $j = 1,\ldots,n+1$ and σ is the sigmoid function. Also, to simplify the notation, the +1 of the bias is replaced by the notation $z_{n+1,\,p}$ in the equations.

3.3.4 Step 4: Maximize Correlation Formula for First Hidden Unit

The next step in the CC algorithm is to maximize the correlation for the new hidden unit installed on the network. Optimization is performed with the BFGS method from DOT (Vanderplaats, 1995).

For this weight initialization, the norm of the input vector $\mathbf{Z_p}$, $\left\| \mathbf{Z}_\mathrm{p} \right\|$, is calculated for each training set point p (also called pattern). And the weights for the new candidate unit w_j are initialized so that:

$$\sum_{j=1}^{n+1} w_j^{\,2} \le 4 * \max_{p=1,\ldots,Np} \left(\left\| \mathbf{Z}_\mathrm{p} \right\| \right) \tag{11}$$

to avoid starting the optimization in the highly saturated part of the sigmoid, and thus getting a null derivative of the correlation formula with respect to the weights w_j. In fact, a pool of candidate hidden units is generated with different random initial weights, and two options are available to train the candidates depending on the size of the pool chosen. If this number is less than 10, the algorithm is programmed so that all candidate units in the pool are trained to maximize the chosen correlation formula using the BFGS algorithm. Only the unit with the largest correlation value after training is next installed on the network. This method is the same as Fahlman's algorithm except for the use of the BFGS instead of the Quickprop (Fahlman, 1988) algorithm. If the size of the pool is greater than 10, then only the candidate unit which exhibits the largest correlation value after random initialization is trained with the BFGS method and next permanently installed on the NN. This builds the network faster since the time-consuming operation of optimizing the weights is done only once. The other option implies optimizing several candidates. For this method to work well, it is recommended to use a large pool, say 100 to 500 candidates. Only the candidate whose correlation is the highest is kept in memory. Its weights are saved in a separate matrix **WH**:

$$\mathbf{WH} = \left[w^1{}_1 \quad \ldots \quad w^1{}_{n+1} \right] \tag{12}$$

Those connections are now permanently frozen.

The following equation describes the correlation formula, denoted S_C, between the candidate unit's value and the residual output error observed at the first unit.

$$S_C = \sum_{i=1}^{m} \left| \sum_{p=1}^{Np} \left(z_{n+2,p} - \overline{z_{n+2}} \right) \left(E_{ip} - \overline{E_i} \right) \right| \tag{13}$$

where E_{ip} is the residual error $E_{ip} = y_{ip} - t_{ip}$ calculated with the outputs y_{ip} from the previous step. Strictly speaking, S_C, is actually a covariance, not a true correlation because the formula leaves out some of the normalization terms.
The gradient of S_C with respect to the w_l can again be calculated analytically and is supplied to the optimizer to speed up the process.

$$\frac{\partial S_C}{\partial w_l} = \sum_{i=1}^{m} \left[\begin{array}{l} \operatorname{sgn}\left(\sum_{p=1}^{Np} \left(z_{n+2,p} - \overline{z_{n+2}} \right) \left(E_{ip} - \overline{E_i} \right) \right) \\ \times \left(\sum_{p=1}^{Np} \left(E_{ip} - \overline{E_i} \right) \left[\frac{\partial z_{n+2,p}}{\partial w_l} - \frac{\partial \overline{z_{n+2}}}{\partial w_l} \right] \right) \end{array} \right] \tag{14}$$

where

$$\frac{\partial z_{n+2,p}}{\partial w_l} = \sigma' \left(\sum_{i=1}^{n+1} w_i z_{ip} \right) z_{lp} \tag{15}$$

$$\frac{\partial \overline{z_{n+2}}}{\partial w_l} = \frac{1}{Np} \sum_{k=1}^{Np} \frac{\partial z_{n+2,k}}{\partial w_l} \tag{16}$$

and

$$\sigma'(x) = \partial \sigma / \partial x \tag{17}$$

is the derivative of the activation function (sigmoid).

3.3.5 Step 5: Connect First Hidden Unit to Outputs

Once trained, the new hidden unit is connected to the outputs with the weights saved in matrix WH. The output $z_{n+2,p}$ is now fixed; it acts as an additional input to the NN. The NN equation can be written as:

$$\begin{bmatrix} y_{1p} \\ \vdots \\ y_{mp} \end{bmatrix} = \begin{bmatrix} v_{11} & \cdots & v_{1,n+2} \\ \vdots & & \vdots \\ v_{m1} & \cdots & v_{m,n+2} \end{bmatrix} \begin{bmatrix} z_{1p} \\ \vdots \\ z_{n+2,p} \end{bmatrix} \tag{18}$$

A schematic of the connections is represented in Fig. 4, the weights that connect the input-to-outputs weights and the first HU-to-output weights are still unknown and must be trained in Step 6.

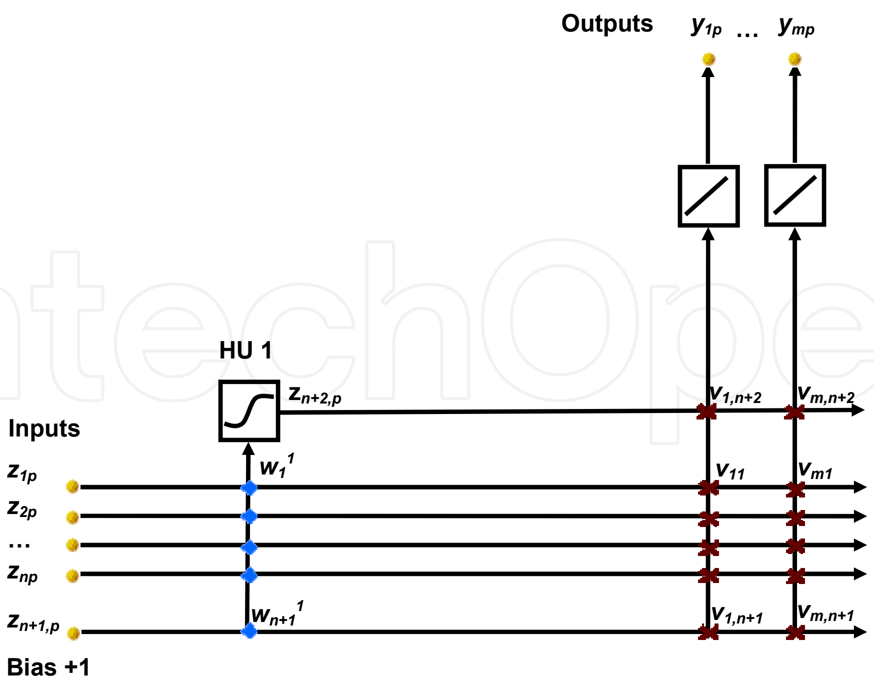

Fig. 4. Schematic of first hidden unit connected to the outputs. Diamond connections are permanently frozen. X connections must be trained again.

3.3.6 Step 6: Minimize Squared Error with first Hidden Unit

Once the new HU is installed on the network, the matrix **V** must be optimized to minimize the squared error on the TS using the BFGS algorithm from DOT.
The squared error is calculated as:

$$Es = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left\| y_{ip} - t_{ip} \right\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left[\sum_{j=1}^{n+2} v_{ij} z_{jp} - t_{ip}\right]^2 \tag{19}$$

The gradient, which is also supplied to the optimizer, is given by:

$$\frac{\partial Es}{\partial v_{kl}} = \sum_{p=1}^{Np}\left[\sum_{j=1}^{n+2} v_{kj} z_{jp} - t_{kp}\right] z_{lp} \tag{20}$$

for $k \in \{1 \ldots m\}$ and $l \in \{1 \ldots n+2\}$.
Weights are initialized by taking former weights calculated above for $j \in \{1 \ldots n+1\}$ and set to zero for $j = n+2$. Indeed, the first $n+1$ columns of the **V** matrix represent the input-to-output weights. Those have already been adjusted at Step 2 to minimize the squared error. It is therefore expected that a good initial guess for the solution with the additional HU in the network is to take the former weights calculated at Step 2 and to set to zero the weights that connect the new hidden unit to the outputs. These weights correspond to column $n+2$ in the weights matrix **V**.
After the squared error is minimized for the training set, it is next evaluated on the validation and the generalization set if they have been specified. Either the epsilon stopping

criterion or one of the early stopping criteria can be chosen. (see Section 0). The chosen criterion for stopping is checked. If it is met the program stops. If it is not met, the program continues adding hidden units one at a time.

3.3.7 Step 7: Connect h^{th} Hidden Unit to Inputs

Each time a new hidden unit is added, a link from this neuron to all the inputs (and bias) and the former hidden units is created (see Fig. 5).

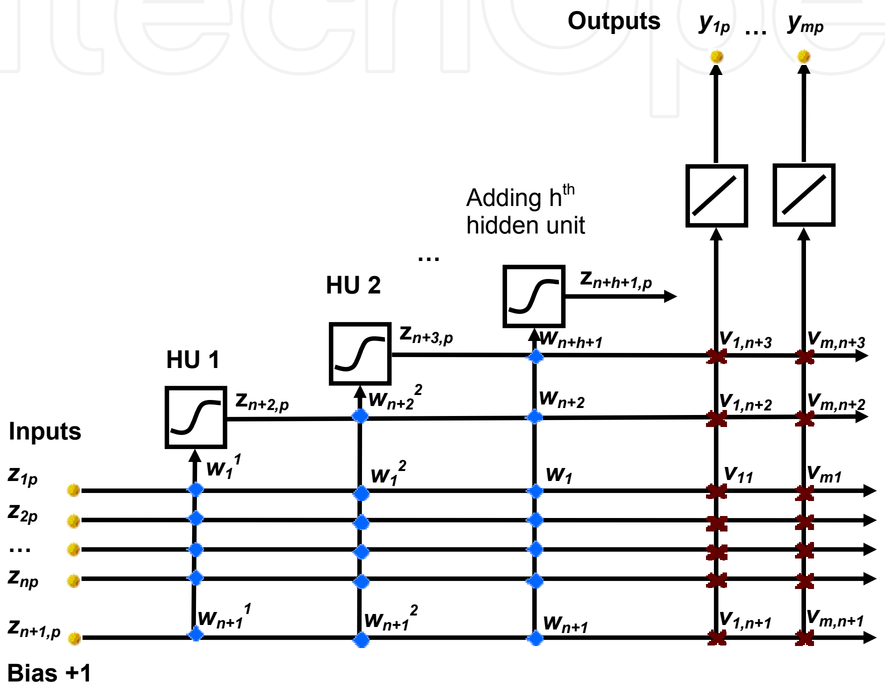

Fig. 5. Schematic after adding h^th hidden unit. Hidden unit connected to former HUs and inputs only. Diamond connections to the h^th hidden unit only must be trained (w_j weights)

The equation for the output of the h^th hidden unit added on to the NN, $z_{n+1+h,p}$, is:

$$z_{n+1+h,p} = \sigma\left(\sum_{j=1}^{n+h} w_j z_{jp}\right) \tag{21}$$

The z_{jp} are the inputs to the network for $j=1\ldots n+1$ and the outputs from the h-1 previous hidden units for $j=n+2, \ldots n+h$. Since the input-to-hidden unit weights are frozen for the h-1 previous HUs, they can be viewed as additional inputs to the network. The weights w_j from the inputs and the previous HUs to the h^th hidden unit are unknown and must be adjusted.

3.3.8 Step 8: Maximize Correlation Formula for h^{th} Hidden Unit

Next, the w_j weights are adjusted to maximize the correlation formula. Again a pool of candidate units is created by initializing the weights w_j for each candidate at random and Step 4 is repeated. The candidate whose correlation is the highest is kept in memory as w_j^h. Those weights are saved on row h of the matrix **WH.** Each line of this matrix contains the weights w_j^i saved for each HU. Its dimension is increased by one row and one column each time a new HU is added.

$$\mathbf{WH} = \begin{bmatrix} w^1{}_1 & \cdots & w^1{}_{n+1} & 0 & \cdots & 0 \\ w^2{}_1 & \cdots & & w^2{}_{n+2} & \ddots & \vdots \\ \vdots & & & & \ddots & 0 \\ w^h{}_1 & \cdots & & & \cdots & w^h{}_{n+h} \end{bmatrix} \tag{22}$$

Again, the equation for the correlation can be written as:

$$S_c = \sum_{i=1}^{m} \left| \sum_{p=1}^{Np} \left(z_{o,p} - \overline{z_o} \right) \left(E_{ip} - \overline{E_i} \right) \right| \tag{23}$$

and

$$z_{o,p} = z_{n+1+h,p} \tag{24}$$

For simplicity, in the equations the output to the hth hidden unit is denoted $z_{o,p}$ instead of $z_{n+h+1,p}$. Also as before, E_{ip} is the residual error $E_{ip} = |y_{ip} - t_{ip}|$ calculated with the outputs y_{ip} for the network with h-1 hidden units.
The gradient of S_C is given by

$$\frac{\partial S_C}{\partial w_l} = \sum_{i=1}^{m} \left[\operatorname{sgn}\left(\sum_{p=1}^{Np} \left(z_{o,p} - \overline{z_o} \right)\left(E_{ip} - \overline{E_i} \right) \right) \times \left(\sum_{p=1}^{Np} \left(E_{ip} - \overline{E_i} \right) \left[\frac{\partial z_{o,p}}{\partial w_l} - \frac{\partial \overline{z_o}}{\partial w_l} \right] \right) \right] \tag{25}$$

where

$$\frac{\partial z_{o,p}}{\partial w_l} = \sigma' \left(\sum_{i=1}^{n+h+1} w_i z_{ip} \right) z_{lp} \tag{26}$$

$$\frac{\partial \overline{z_o}}{\partial w_l} = \frac{1}{Np} \sum_{k=1}^{Np} \frac{\partial z_{o,k}}{\partial w_l} \tag{27}$$

3.3.9 Step 9: Connect h^{th} Hidden Unit to Outputs

The candidate HU with the highest correlation is added on to the network and connected to the output (see Fig. 6). The matrix **V** connects the inputs and all hidden units installed to the network to the outputs. The outputs can be calculated with the following equation.

$$\begin{bmatrix} y_{1p} \\ \vdots \\ y_{mp} \end{bmatrix} = \begin{bmatrix} v_{11} & \cdots & v_{1,n+1+h} \\ \vdots & & \vdots \\ v_{m1} & \cdots & v_{m,n+1+h} \end{bmatrix} \begin{bmatrix} z_{1p} \\ \vdots \\ z_{n+1+h,p} \end{bmatrix} \tag{28}$$

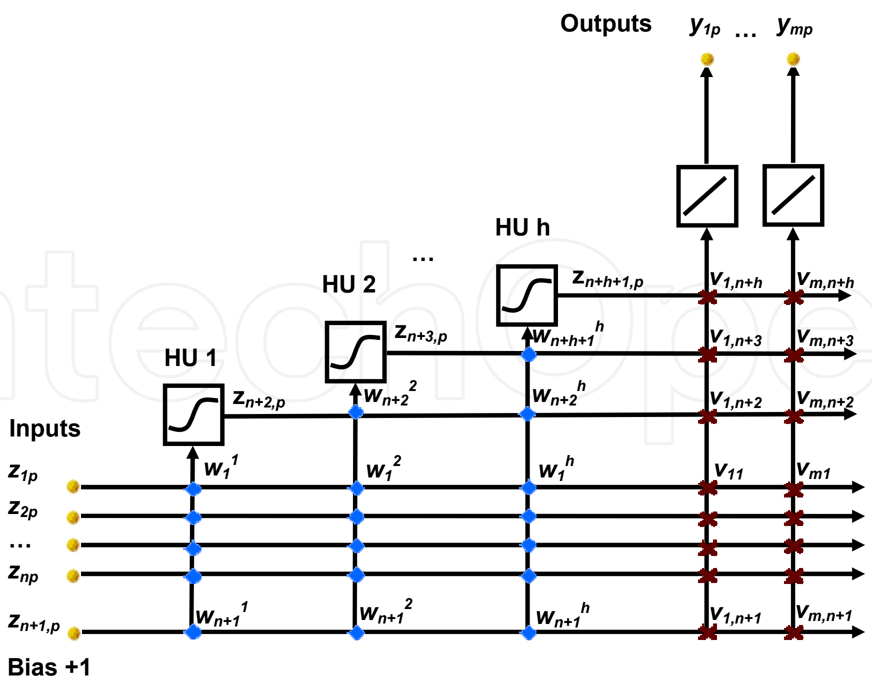

Fig. 6. Connecting hth hidden unit to the outputs. X connections must be trained again.

3.3.10 Step 10: Minimize Squared Error for h^{th} Hidden Unit

Similarly to Step 6, weights v_{ij} are initialized by taking former weights calculated for the NN with *h-1* HUs for columns $j \in \{1,\dots, n+h\}$ and set to zero for column $j = n+1+h$. The squared error is calculated over the training set and is minimized using the BFGS algorithm.
The squared error and its gradient are both supplied to the optimizer, their equations are:

$$Es = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left\| y_{ip} - t_{ip}\right\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{p=1}^{Np}\left[\sum_{j=1}^{n+1+h} v_{ij} z_{jp} - t_{ip}\right]^2 \tag{29}$$

and

$$\frac{\partial Es}{\partial v_{kl}} = \sum_{p=1}^{Np}\left[\sum_{j=1}^{n+1+h} v_{kj} z_{jp} - t_{kp}\right] z_{lp} \tag{30}$$

for $k \in \{1,\dots,m\}$ and $l \in \{1,\dots,n+1+h\}$.

3.3.11 Step 11: Stop Training when Stopping Criterion is met

Steps seven through ten are repeated and the cascade correlation algorithm stops when the stopping criterion is met. Several stopping criteria are available to the user in the modified algorithm. The first one, the epsilon stopping criterion, is used in Fahlman's original algorithm (Fahlman & Lebiere, 1990). However, this criterion does not prevent overfitting and therefore, was later replaced by the early stopping criteria as described below.
The Epsilon Stopping Criterion
The epsilon stopping criterion stops the algorithm when the square error on the training set has reached a predetermined ε value. The problem with this criterion is that the user must determine in advance which ε value to use. Also the squared error can vary significantly

from one function to another. The average value of the outputs changes the error as defined by Eq. 1. Also the number of points used in the training will change the value of the error. So, this ε value should be adjusted by the user manually every time a network needs to be trained. Also this stopping criterion does not give any information on the generalization ability of the network, i.e. how the network performs for points not in the training set. This criterion is thus replaced by the early stopping criteria described in the next subsection, which monitor the error on an unseen dataset, the validation set.

The Early Stopping Criteria

As alluded to earlier, these criteria allow to limit overfitting of the network by checking how the error decreases on the validation set. It is commonly know that as more units are added, the network is able to fit training data better since additional degrees of freedom are added. However, the error on data not used during training decreases at first but then later increases, showing signs of overfitting or overtraining. The idea of early stopping is to stop training early, before full convergence of the network on the training set, or when the error on the unseen dataset, the VS, is minimum. However in order to find the minimum of the error on the VS, one must continue training the network some time past this minimum and then stop and choose the network with the number of hidden units which correspond to that minimum error. This leads to several stopping criteria to decide how long to continue training after a minimum of the error is found.

Three classes of stopping criteria are available in the MCC. They are described in detail in Schmitz (2007) and are not repeated here. The criterion used in both applications described in this chapter is the PQ0.75 criterion. This criterion is relatively efficient and accurate in finding the true minimum error on the VS.

A few definitions are required before the PQ criterion can be derived. The squared error calculated on the TS for h hidden units added on the network will be noted $Es_{TS}(h)$ and called training set error at epoch h, or training error for short. The epoch h corresponds to a network trained with h hidden units, with h varying from 0 to the maximum number of hidden units (namely 70 in the MCC). $Es_{VS}(h)$, the validation error, is the corresponding error on the VS. Let $Es_{OPT\text{-}VS}(h)$ be the lowest validation error obtained in epochs up to h:

$$Es_{OPT-VS}(h) = \min_{h' \le h} Es_{VS}(h') \tag{31}$$

The generalization loss ($GL(h)$) at epoch h is defined as the relative increase of the validation error over the minimum so far, in percent:

$$GL(h) = 100 \cdot \left(\frac{E_{VS}(h)}{E_{OPT-VS}(h)} - 1 \right) \tag{32}$$

The training progress, denoted $P_k(h)$, measures how large the average training error is during a training strip of length k (epochs from h-k+1 to h) with respect to the minimum error during that same strip.

$$P_k(h) = 1000 \cdot \left(\frac{\sum_{h'=h-k+1}^{h} Es_{TS}(h')}{k \cdot \min\limits_{h-k+1 \le h' \le h} Es_{TS}(h')} - 1 \right) \tag{33}$$

The *PQ0.75* criterion can be defined as following: training stops when the quotient of generalization loss and progress exceeds a threshold α=0.75, such that:

$$PQ(h) = \frac{GL(h)}{P_k(h)} > 0.75\,. \tag{34}$$

Note that this criterion is only checked after every end-of-strip epoch *h*, where the length of the strip is *k* epochs. In the following study we will always assume length of strips *k*=5.
When the criterion is met, the training is stopped at some value of *h* and the resulting set of weights is the one that corresponds to the lowest validation error $Es_{OPT\text{-}VS}(h)$. So, the corresponding network usually has a number of hidden units *h'*<*h*. Note that the criterion does not ensures stopping, so a large maximum number of hidden units (h_{max} = 70) was chosen to avoid training indefinitely.

3.3.12 Resulting Single Network

Once the stopping criterion is met, the algorithm stops. If the epsilon criterion is used, the resulting network is the last trained. If one of the early stopping criteria is used, the program chooses the network with the number of hidden units corresponding to the minimum validation error. The program keeps several matrices in memory: **WH**=[$w^h{}_j$], the weights between inputs and each hidden unit saved after adding each unit is added to the NN. Also, it keeps two sets of v_{ij} weights; the one between inputs, plus all hidden units and outputs after the last hidden unit has been added to the network and the set of v_{ij} weights which correspond the minimum validation error (if needed).
The function, approximating *f* is thus given by the equations below and can be evaluated by recurrence, so that:

$$\begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} v_{11} & \cdots & v_{1,n+1+h} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ v_{m1} & \cdots & v_{m,n+1+h} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ +1 \\ u_1 \\ \vdots \\ u_h \end{bmatrix} \tag{35}$$

$$\begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ u_h \end{bmatrix} = \sigma\left(\begin{bmatrix} w^1{}_1 & \cdots & w^1{}_{n+1} & 0 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & \ddots \quad \ddots & \vdots \\ w^h{}_1 & \cdots & \cdots & \cdots & w^h{}_{n+h} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ +1 \\ u_1 \\ \vdots \\ u_h \end{bmatrix} \right) \tag{36}$$

where x_l ($l \in \{1,\dots,n\}$) are inputs to the neural network, y_i ($i \in \{1,\dots,m\}$) outputs and u_k ($k \in \{1,\dots,h\}$) intermediate states, calculated by recurrence. Also, h is the number of hidden

units corresponding to the last hidden unit added to the network, if the epsilon stopping criterion is used, or the one corresponding to the minimum validation error.
Note that the network has been trained with inputs normalized according to the TS and outputs rescaled so that the TS average is one. Therefore, the datapoint **X**, must be linearly transformed before evaluating the output of the network according to the following equation.

$$\mathbf{X} = \begin{bmatrix} x_{1p} \\ \vdots \\ x_{np} \end{bmatrix} = \begin{bmatrix} \dfrac{z_{1p} - MinInput_1}{MaxInput_1 - MinInput_1} \\ \vdots \\ \dfrac{z_{np} - MinInput_n}{MaxInput_n - MinInput_n} \end{bmatrix} \tag{37}$$

where *MinInput*$_i$ and *MaxInput*$_i$ are the minimum and maximum values obtained from Eq. 3 and 4.
When using the early stopping criterion, several networks are trained sequentially using the same TS and VS. Because the weight surfaces have many local minima and maxima and the weights are initialized at random, the networks will all be different. Only the network which leads to the smaller validation error is retained.

3.3.13 Resulting Ensemble Network

If ensemble averaging is chosen, all networks built are kept in memory and the output of the committee network is taken as the average output of each individual network. A simple average according to

$$\mathbf{Y}_{Ensemble\_NN} = \frac{1}{M}\sum_{k=1}^{M}\mathbf{Y}^{(k)} \tag{38}$$

is used to calculate the prediction ability of the ensemble. The output $\mathbf{Y}_{Ensemble\_NN}$ is average of output values of each individual network $\mathbf{Y}^{(k)}$ where $k=1,\ldots.,M$ is the index of the network belonging to the ensemble.

4. Application to Fast Ship Multi Disciplinary Design Optimization

This section describes an MDO optimization of an underwater hull configuration. The optimization is performed using both a "classical approach", in which the CFD analysis is integrated directly *inside* the optimization loop, and an "NN approach", in which the CFD analyses are used for TS generation, i.e. *outside* of the optimization loop.

4.1 Design Problem Description

The problem consists of optimizing one of Pacific Marine's advanced lifting bodies. This patented underwater hull is made of two displacement bodies, called *H-bodies*, linked with a thin foil, referred to as *cross-foil*, and attached to the ship by two struts as shown in Fig. 7. The entire arrangement is referred to as the *twin H-body* configuration and can be fitted to

catamaran or pentamaran hull forms. The displacement bodies are designed to provide good sea-keeping properties at lower speeds when the hulls of the catamaran or pentamaran are partially submerged, while the cross-foil is designed to provide additional lift at higher speed when the multiple hulls are lifted out of the water in order to reduce drag.
A very similar configuration was optimized under a previous work reported by Hefazi *et al.* (2002), using the classical optimization process. Sea trials of a half-scale replica of this configuration on a 44 ft test platform were conducted to provide data for the validation and demonstrate the application of lifting body technology on a real test platform (Hefazi *et al.*, 2003). The resulting optimized geometry was integrated into the first US-built "fast ship", the HDV-100 technology demonstrator (Fig. 8).

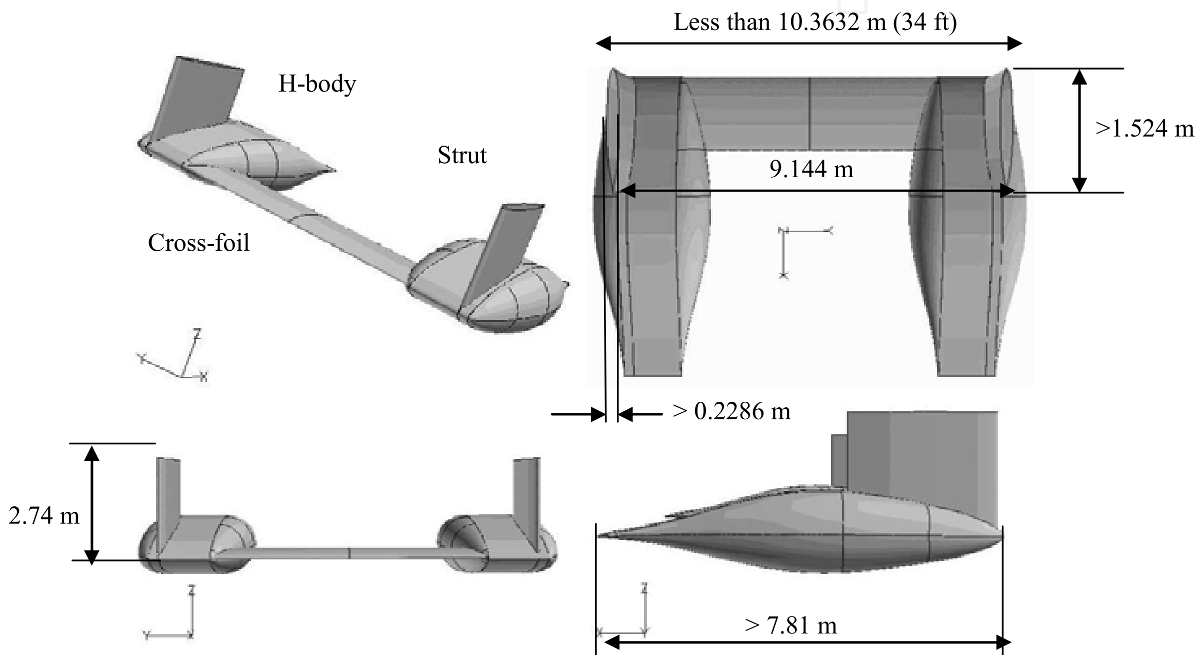

Fig. 7. Twin H-body configuration, baseline

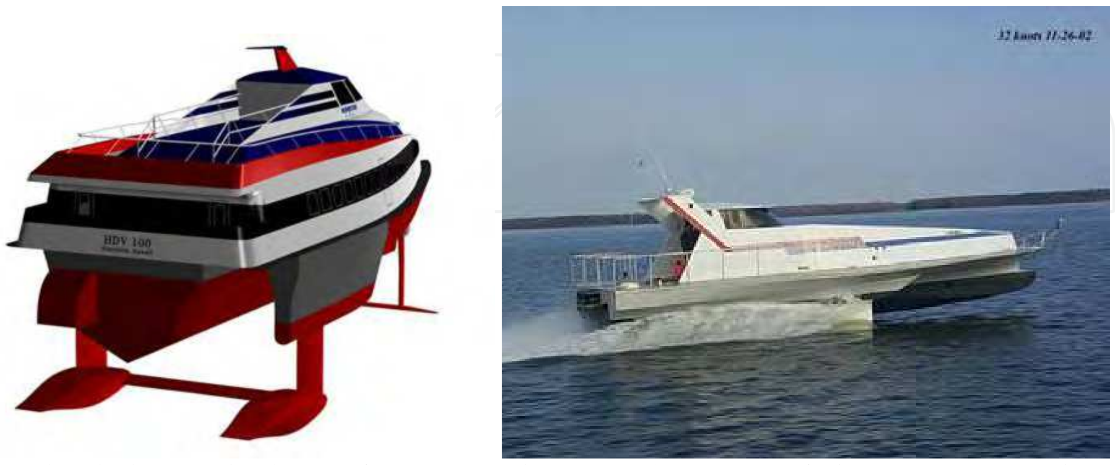

Fig. 8. Blended-wing-body configuration sketch and picture of the HDV-100 technology demonstrator (shown here at low speed)

The optimization is performed for a boat speed of 47 knots (24.2 m/s). At this velocity, the parent hull will run dry. The whole configuration must be able to generate a total lift of 80 LT, including a minimum of 16 LT by displacement effect.
The objective is to maximize range, i.e. lift-to-drag ratio, which corresponds to minimizing drag at constant lift. In addition, the configuration is to be designed such that it can operate cavitation free at 52 knots. Additionally, the operating draft is to be 2.74 m (waterline to lowest point - 9 ft) and a structural constraint is to be imposed to prevent yield of the cross-foil. Also, the maximum width of configuration shall not exceed 10.36 m (34 ft) and the strut-to-strut distance is fixed by requirements for mating to the upper hull.
From an optimization problem point of view, the objective function and design constraints can be summarized as:

- Objective: Maximize lift-to-drag (LOD*)*
- Constraints:
- Operational speed of 47 knots (i.e. Reynolds number is 211.63×10^{6} based on a reference chord of 8.69 m in 75 F Hawaii waters)
- Cavitation free at 52 knots, i.e. $C_p > -0.269$ (Hawaii water at 75 F)
- Total lift (TL) equal to 80 LT
- Displacement or buoyant lift (BL) greater that 16 LT
- 2.74 m (9 ft) operating draft
- Strut centerline to strut centerline distance is fixed at 9.14 m (30 ft) for mating with upper hull
- Overall beam length should not exceed 10.36 m (34 ft)
- Minimum strut thickness 0.2286 m (0.75 ft)
- Minimum strut chord of 1.524 m (5 ft)
- The cross foil should have the structural integrity, not to yield using a material of yield strength of 344.7379 Mpa (50 ksi), assuming solid section.
- Displacement pod length should not exceed by more than 20% the parent body (defined as optimized configuration (Hefazi, 2002)), i.e. 7.81 m.

4.2 Classical Optimization

4.2.1 Objectives, Constraints and Optimization Implementation

Nowadays, all design engineers are accustomed to designing their vehicles using some computer aided design (CAD) or solid modeling package, such as Pro-Engineer, CATIA, UniGraphics, IDEAS. In addition, the designer usually represents the configuration by a set of parameters, or design variables, which can be varied to improve the design by linking the CAD software with an appropriate analysis module. This approach is routinely used for structural design, for example, by linking the CAD software with a finite element (FE) method. Such an approach is not yet routinely used, however, in the case of hydrodynamic shape optimization, because additional challenges face the designer. Among these issues are

- The cost and accuracies associated with flow analysis using CFD
- The grid requirements for CFD methods

In order to address the issue of CFD shape optimization, one must be able to automatically vary the shape of various elements of the configuration in the CAD method, generate a mesh of sufficient quality for the CFD method, and use an efficient and accurate CFD method to obtain the hydrodynamic performance of the configuration being analyzed. The driver in the selection of the components of the CFD optimization method is the ability to link these tools together in an automated fashion, without user intervention, while ensuring that the flow analysis is both efficient and accurate for the problem at hand. For this reason, several options for each tool were considered and the following set of tools was selected:

- CAD software: *Pro-Engineer* (Parametric Technology Corporation, 2009)
- Grid generation software: *ICEM CFD* (ANSYS, Inc., 2009)
- CFD software: CSULB-developed interactive boundary layer (IBL) approach with free surface modeled by negative images (Besnard, 1998, and Hefazi *et al*., 2002)
- Optimization software: *iSIGHT* (Dassault Systèmes SIMULIA, 2009)

Pro-Engineer and *ICEM CFD* were selected because of the existence of a module which does allow for automatic data transfer between the CAD and grid generation package. The IBL approach was chosen because at high Reynolds numbers and low angles of attack, it is a very accurate and efficient approach. In addition, because of the large Froude number for the case at hand, the free surface can be modelled with negative images. Finally, the numerical optimizer *iSIGHT* offers an easy to use platform for the optimization and/or design of experiments. The method, controlled by *iSIGHT*, integrates the different software packages with several scripts, which, once set up, performs all tasks automatically, without user intervention. This automatic setup is critical in the optimization process. *iSIGHT* controls the process and calls the various scripts;

- Define and generate the new geometry (*Pro-Engineer*);
- Check for any constraint violation (from output of *Pro-Engineer*);
- Generates a mesh suitable for the CFD method (*ICEM CFD*);
- Executes the CFD method (IBL code); and
- Extracts the data needed by *iSIGHT* (objective function and constraint values) and calculate the constrained objective function for the next iteration.

The process implemented for the Twin-H body optimization is shown in Fig. 9. The configuration is represented by a total of 28 design variables which control the size and shape of each component, which, once assembled, describe the entire configuration. The first step involves generating a geometrically feasible configuration from the selected set of design variables. Simple geometrical parameters, such as width, length, chord, etc., are used to characterize the elements. Foil cross-sections are defined by their mean camber line and thickness distributions. Camber line is parameterized by the classical NACA two-parameter set. The thickness distribution, y_{th}, is represented by a 6-deg. polynomial, with an additional term, a_0, for controlling the leading edge radius:

$$y_{th}(x) = a_0\sqrt{x} + \sum_{i=1}^{6} a_i x^i, \quad 0 \le x \le 1 \tag{39}$$

This configuration is automatically generated by the CAD-based solid modeler, *Pro-Engineer*, based on the 28 design variables using scripts. This process involves two parts:

- Airfoil shape definitions ("shape.in" files)
- Configuration parameterization (".ptr" files)

Several independent scripts using the different "shape.in" files are used to regenerate updated *Pro-Engineer* files (".ptr" files). Then, *Pro-Engineer* automatically updates the geometry based on the revised data.

In the second step, constraints which may be determined from the newly generated solid model, such as volume, structural constraint, etc., are evaluated. In third step, a suitable mesh is automatically generated using *ICEM CFD*. This mesh is then used along with the CFD input data file to execute the CFD code. The CFD tool used here makes use of the high Froude and Reynolds number approximations by employing a viscous-inviscid approach. The inviscid flow is solved by a higher order panel method with the free surface effects modeled by negative images and the viscous flow is solved using an inverse boundary layer approach which can treat large regions of flow separation. Viscous and inviscid methods are coupled using the blowing velocity/displacement concept and leads to accurate pressure, lift and drag predictions at minimal costs for this type of configuration and flow conditions (see, e.g., Hefazi *et al.* 2002). Hence, only a surface mesh is needed. The use of a Reynolds averaged Navier-Stokes (RANS) method would require the use of a volume mesh which could also be implemented with the tools used here (*ICEM CFD*).
The last step involves the use of a constrained objective function, f_c. Because a global optimization method is used (genetic algorithm), the objective function and constraints are integrated into a single "constrained objective function" which is to be minimized. The constrained objective function is such that, when at least one constraint is strongly violated, f_c is set close to f_{max}. f_{max} is typically on the order of one and corresponds to a normalized value of the maximum objective function. The normalization value is given by the user and is typically chosen at the higher values of expected f over the search space. For example, with an optimization where L/D is the objective function on the order of 10-15, a normalization value of 10 to 20 can be chosen. Choosing 10 would mean that f_c might reach 1.5.
Two positive parameters ε_1 and ε_2 are now defined. If $\forall i,\ \ g_i \le -\varepsilon_1$, then the constrained objective function is f. If $\exists\ i\ \mid\ g_i \ge \varepsilon_2$, then the penalized cost function gets close to f_{max} depending on how the constraints are violated. In other words, ε_1 decides when constraints become active, and ε_2 when they become prohibitive.
The generalized constraint G is defined as

$$G(x) = \frac{1}{n_{con}} \left(\sum_{g_i \ge -\varepsilon_1} g_i(x) + \varepsilon_1 \right) \tag{40}$$

and the constrained objective function becomes

$$f_c(x) = \left[1 - \varphi(y)\right].f(x) + \varphi(y).f_{\max}.\left(1 - \frac{e^{-y}}{2}\right) \tag{41}$$

where

$$\varphi(x) = \begin{cases} 0 & \text{if } x \le 0 \\ e^{-1/x^2} & \text{if } x > 0 \end{cases} \tag{42}$$

and

$$y = \frac{10G(x)}{(\varepsilon_1 + \varepsilon_2)} \tag{43}$$

For the NN based optimization, the generation of the training and validation sets is performed using the same approach, except that instead of using iSIGHT setup to run an optimization (genetic algorithms in the present case), it is designed to run Latin Hypercube samplings of the desired TS, VS and GS sizes.

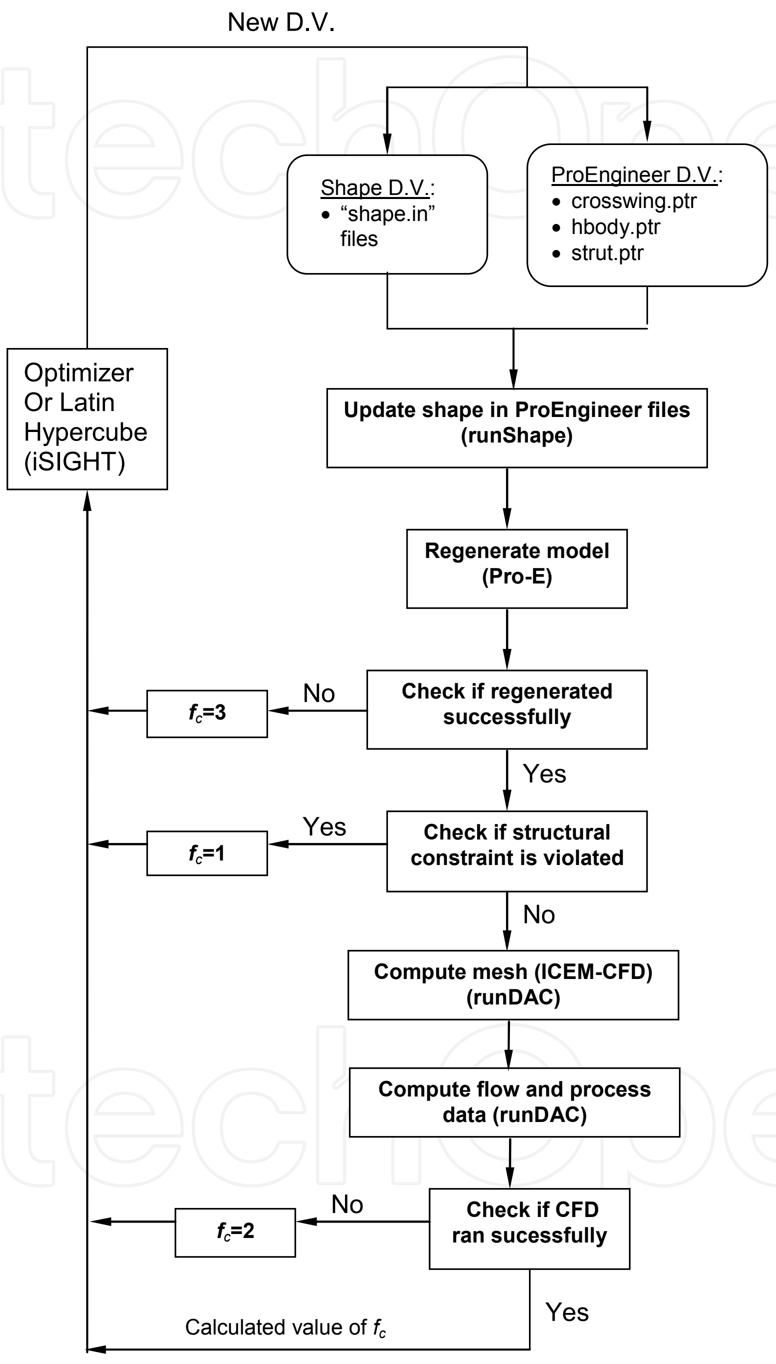
New D.V.
Shape D.V.:
• "shape.in" files
ProEngineer D.V.:
• crosswing.ptr
• hbody.ptr
• strut.ptr
Optimizer Or Latin Hypercube (iSIGHT)
Update shape in ProEngineer files (runShape)
Regenerate model (Pro-E)
Check if regenerated successfully
No
f_c=3
Yes
Check if structural constraint is violated
Yes
f_c=1
No
Compute mesh (ICEM-CFD) (runDAC)
Compute flow and process data (runDAC)
Check if CFD ran sucessfully
No
f_c=2
Yes
Calculated value of f_c

Fig. 9. Optimization with the classical approach. This same loop is used for training/validation set generation but with Latin Hypercube sampling instead of optimization.

4.2.2 Classical Optimization Results

A genetic algorithm optimization was run over the 28-dimension design space for 5000 iterations. The outcome was a configuration with an L/D of 12.92, which represents a 26 percent improvement in L/D over the baseline design. A comparison of the performance of the baseline and optimum configurations is shown in Table 1. Note that the number of iterations used is small for a global search problem with 28 design variables, but it was limited to 5000 because of time requirements. One iteration takes about 10 min. to run on an Origin 3200 server. 5000 runs correspond to over a month of CPU time.

	Baseline	Optimum	Objective
Buoyant lift	18.4 LT	16.1 LT	> 16 LT
Total lift	80.2 LT	81.6 LT	= 80 LT
Cpmin	-0.263	-0.250	> - 0.269
LOD	10.23	12.92	Maximum

Table 1. Performance of optimized vs. baseline configuration (28 design variables)

4.3 Neural Network Optimization Approach

The use of the neural network (NN) approach encompasses several steps:

- Generation of the training set (TS) & validation set (VS)
- NN training to obtain a NN "evaluator(s)"
- Optimization with the NN evaluator(s)

The first two steps are explained in detail in the next subsections. The third step is essentially the same as the classical optimization with the CFD code replaced by the neural network, and thus is not repeated here.

The optimization approach specific to the twin H-body optimization problem calls for the use of five single output neural networks as shown in Fig. 10, one for the objective function and the others for the constraints: lift-to-drag ratio (LOD), minimum pressure (Cpmin), dynamic lift (DL), buoyant lift (BL) and maximum stress value (Struc). Alternatively, a single NN with five outputs could have been used, but typically, the resulting network is much more complex than five individual networks (has more weights). Hence, it takes more time to generate, i.e. train, and usually needs a larger training set than five single output networks to generalize well. Also the five networks can be trained in parallel on a multiprocessor machine, reducing the training time even more.

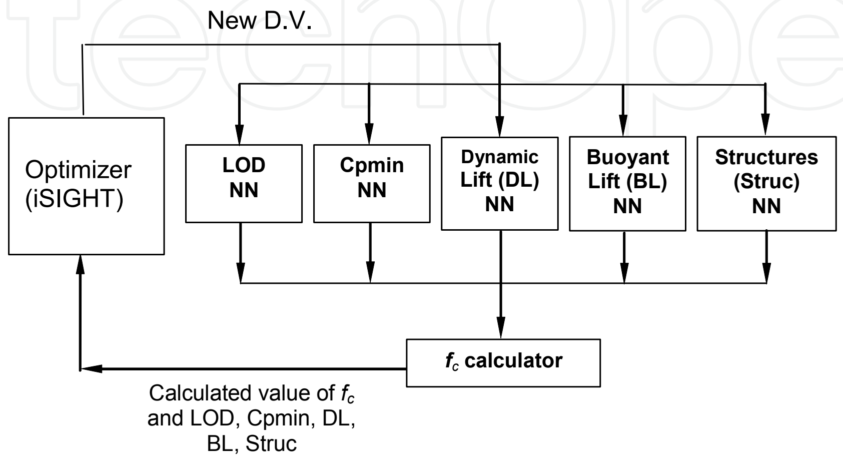

Fig. 10. Optimization process with NN approach

Also, although the optimization is performed on the constrained objective function, f_c, it was not represented directly by a single NN because it would have required a very good training set definition in the small regions of the design space where the design was feasible. Instead, using five networks allows for accurate representation of each function over the design space and thus improved predictions for f_c.

4.3.1 Training and Validation Sets

For the present analysis, a validation set (VS) of 300 points was generated with the iSIGHT setup of Fig. 9 and using the same 28 design variables as for the classical approach, but using a Latin Hypercube instead of an optimization algorithm. Two training sets (TS) were also generated using the same process, one with 1000 points, the second with 2000 points. A third set with 5000 points was also generated to be used as the Generalization Set (GS). Although a practical application may not involve the generation of multiple TS and certainly not a GS, this was done here to analyze the effect of TS size on the result quality.
For each size of Latin Hypercube sampling in the design space, approximately 20 to 25 percent of the points were geometrically unfeasible (i.e. the model could not be successfully reconstructed) and had to be removed from the training sets. Nevertheless, the VS, TS and GS will be referred to as 300-VS, 1000-TS, 2000-TS and 5000-GS subsequently. The exact sizes for each set are respectively, 228, 808, 1587, and 3852.

4.3.2 Neural Network Training

Each neural network (NN) was trained with 28 inputs (or design variables) and one output, i.e. one for each function: buoyant lift (BL), Cpmin, dynamic lift (DL), lift-to-drag ratio (LOD), structural constraint (Struc) and total lift (TL). For each function, 10 NNs were built and the one that had the best validation error was chosen. Typically, training each network takes from about an hour (for 1000-TS) to a day for the more complete data sets (5000-GS). Unlike CFD methods which are demanding in computing power (I/O, memory), training demands relatively little other than CPU time and thus, all training can be done in parallel on the same server without interfering with other ongoing computations. This feature is yet another advantage compared to training a single multiple-output network.
The results for the different TS did not vary much from one to the other and are presented here for the 1000-TS. Also, in order to evaluate the quality of the training and compare the error on the 300 points VS. The GS of 5000 points was evaluated on the trained NN. Table 2 shows the average errors and standard deviations over the TS, VS and GS. Errors and standard deviations are adequate for the problem at hand. For example, an average error of 0.04 is expected for the displacement (BL) which has a value on the order of 18 and the corresponding standard deviation is also 0.04.

	Typical	$\overline{E}_{TS}$	$\overline{E}_{VS}$	$\overline{E}_{GS}$	$\text{std}(E)_{TS}$	$\text{std}(E)_{VS}$	$\text{std}(E)_{GS}$
BL (LT)	18	0.0275	0.0381	0.0419	0.0223	0.0314	0.0427
Cpmin	-0.269	0.0055	0.0065	0.0068	0.0066	0.0071	0.0080
DL (LT)	60	0.1607	0.1847	0.1855	0.2002	0.1420	0.1874
LOD	12	0.2270	0.2602	0.2646	0.1775	0.2074	0.2054
Struc (MPa)	300	4.05	5.3	5.3	3.7	5.2	5.1

Table 2. Average errors and standard deviations on TS, VS and GS for a 1000-pt. TS and a 5000-pt. GS for the 28-design variable Twin H-body optimization

Also, average errors and standard deviations on the VS and GS are in excellent agreement, thus validating the VS approach despite the number of points used for the design space of 28 dimensions.

4.3.3 Neural Network Optimization Results

The neural networks generated in the form of five executables for Cpmin, LOD, DL, BL and Struc using the different sizes of training sets (1000-TS and 2000-TS) were integrated in iSIGHT as shown in Fig. 10. A genetic algorithm (GA) optimization was used with an initial population of 50 and 35000 iterations were run based on the constrained objective function defined in Section 0.

The best 100 runs resulting from the optimization with the five NNs (best f_c) were then run using the Pro-Engineer model and the CFD code to compute the objective function and constraints and compare them with those of the NN near the optimum. Also, a few results from the optimization (i.e. as determined by the NN) with a slightly higher LOD than that of the best f_c but with constraints closer to their limit (therefore resulting in a lower f_c) were chosen and also run through the CFD package. A summary of the results is shown in Table 3. For each size of training set, the table shows:

- The best f_c as determined by the optimizer (i.e. after 35000 GA iterations using the NN)
- The best LOD based on the NN with minimal constraint violations: corresponds to some hand-picked results from the optimization showing a slightly better LOD but with constraints close to the acceptable limits resulting in a higher f_c
- The best f_c based on CFD results from the 100 best NN points (as determined by the GA)
- The best LOD based on CFD results from the 100 best NN points (as determined by the GA)

In each case, buoyant lift (BL), total lift (TL), lift-to-drag ratio (LOD) and minimum pressure (Cpmin) values computed by the NN and the CFD method are shown. The stress value (Struc) is not shown because it exhibited little variations between points and because the constraint was not violated. Also, the dynamic lift (DL) can be directly calculated from total and buoyant lift values (the optimization actually uses DL and BL to determine TL, but the latter is presented because it corresponds to a primitive twin H-body requirement). It should also be noted that the constraints are not implemented as step functions but rather very steep functions which do vary near the constraint border. For example, the primitive requirement calls for a displacement or buoyant lift greater than 16 LT, but as implemented here, a value close to 15 is acceptable. For this reason, f_c may vary in a counter-intuitive fashion, thus rendering the analysis of its values difficult. It is therefore not shown.

The differences between using the NN and the direct CFD computation are within acceptable limits. They are very close in many instances. Also, differences between the values resulting from different selection processes for a particular TS are rather small. Finally, while one observes that the LOD is over-predicted by the NN, the differences between points are about the same for a given TS. Regardless of which TS is chosen, however, LOD is greatly improved from the 12.9 result obtained with the direct CFD method.

TS	Results	Output from NN				Same DV but with CFD			
		BL	TL	LOD	Cpmin	BL	TL	LOD	Cpmin
1000	Best f_c optimized from NN	15.23	80.59	14.39	-0.265	15.24	81.59	13.76	-0.266
	Best LOD optimized from NN	14.75	79.94	14.44	-0.267	14.78	81.04	13.85	-0.268
	Best f_c (using CFD) out of 100 best runs	15.49	80.81	14.30	-0.264	15.51	81.67	13.70	-0.266
	Best LOD (using CFD) out of 100 best	15.25	80.60	14.38	-0.265	15.25	81.64	13.78	-0.265
2000	Best f_c optimized from NN	15.20	80.67	14.49	-0.265	15.13	80.86	13.70	-0.271
	Best LOD Optimized from NN	15.07	80.66	14.51	-0.265	15.01	80.78	13.81	-0.269
	Best f_c (using CFD) out of 100 best runs	15.30	80.74	14.42	-0.263	15.23	80.96	13.71	-0.265
	Best LOD (using CFD) out of 100 best	15.17	80.59	14.47	-0.265	15.11	80.75	13.79	-0.270

Table 3. Results from NN optimization and comparison with CFD

4.4 Comparison between Classical and NN-based Methods

Depending on which design is selected, L/D (LOD) ranges from 13.70 to 13.85, which is a definite improvement from the classical method which lead to 12.92. This improvement is due to the ability to increase the exploration of the design space within the time available in a given design project. For the classical optimization, the genetic algorithm was allowed to run for 5000 iterations, which corresponds to approximately one month of constant calculations on an Origin 3200 server. Because of the use of the CFD tool inside the design loop, CPU time available limited the design space exploration and did not lead to a true optimum. On the other hand, a much larger number of iterations could be performed with the NN approach leading to a greater L/D improvement. In this case, most of the CPU time is taken by the training set generation, with all other computations (training and GA iterations) representing a small fraction of the total CPU time. With as low as 1000 points generated to approximate the various functions over a design space with 28 design variables, an improvement of about 34 percent in L/D is achieved with the NN approach compared with the original baseline twin H-body, this at one fourth the cost needed to get a 26 percent improvement when using the classical approach (with iterations limited because of CPU time constraints).

The results also point to a few improvements which would need to be implemented to address non-differentiable functions (to improve Cpmin predictions, for example), the selection of constrained objective function, and the selection of mathematical optimization method. The latter comment is particularly pertinent for problems in which the optimizer (GA here) would focus its attention in a region of the design space where one (or more) function (objective or constraint) is not approximated as well as might be desired, thus potentially leading to unusable optimization results. One could benefit from having an optimization method which explores several regions of the design space, as illustrated in Lin and Wu (2002).

Results do show, however, that the method can provide its user with a valuable tool for improving designs within a limited time frame and possibly at a lower cost than using conventional analysis tools integrated in the optimization loop. In this latter case, similarly to the twin H-body configuration optimization presented here, the cost of the analyses limits the number of iterations which can be performed in a reasonable time leading likely to sub-optimal solutions. On the other hand, with the NN approach, a large number of designs can be investigated quickly –almost instantaneously– once a training set has been made available and the NN has been trained.

5. Application to an America's Cup Class Yacht Analysis and Design

In this section, we apply the NN approach to a case where one wishes to use large experimental datasets for detailed analysis and optimization of the performance of a system, here an America's Cup class yacht. This case presents unique challenges associated with the fact that some data is not usable and that the data is usually not uniformly distributed over the design space. The objective here is to use an experimental database obtained with a yacht in at-sea trials to determine the fastest upwind speed the boat can have under prevalent wind conditions from the corresponding boat settings (including keel, rudder, sail, etc.).

5.1 Parameters

During trials, America's Cup teams record their boat performance with very high accuracy to create a true dynamic picture of the boat response under various sailing conditions. This sailing data file recorded by the Wave Technology Processor (WTP) is used here as our experimental database. In our example, each sailing "point" is a vector with a total of 40 parameters. Since the objective here is to optimize the upwind performance of the boat, the projected speed over the course is to be maximized:

$$VMG = Vs \cdot \cos(TWA) \tag{44}$$

where *Vs* is the measured boat speed, *VMG* is the Velocity Made Good, and *TWA* is the true wind angle. This *VMG* becomes the objective function, and will be the output of the NN.
Of the remaining 39 parameters, eight independent variables have been selected for the training based on their accuracy and their major influence on boat speed. Other dependent variables, parameters with calibration problems or with marginal impact on boat speed, have been discarded. The eight remaining independent variables to be used for the NN training are (with their variable name written in parenthesis):

- Heel Angle (Heel)
- Leeway Angle (Leeway)
- True Wind Angle (TWA)
- True Wind Speed (TWS)
- Trim Tab Angle (Trim Tab)
- Rudder Angle (Rudder)
- Forestay Tension (Forestay)
- Main Traveler Position (Main Traveler)

The sailing database represents several hours of sailing and about 50 percent of that sailing time is the upwind direction, which is our targeted condition. This type of dataset based on experiments requires using a process to identify the data corresponding to the desired conditions (here upwind) among the entire database and to discard the downwind sailing and all transitional moments like tacking, rounding marks.

5.2 Data filtering

To filter or process the large WTP database several software packages such as Microsoft Access (M.A.) are available. M.A. is able to manipulate large series of data, with a good Microsoft Excel interface to generate plots from the output dataset for verification purposes. The queries technique for M.A. simplifies the experimental dataset filtering process in manipulating with a logical operator the variables and generates a new dataset without altering the original database. Once the queries (point selection criteria) are written, the system is automated to produce a new dataset in an instant from any other experimental database of similar type. M.A. is used first to discard all transitional and non-upwind sailing points. In a second step, since the upwind sailing data is a mix of port and starboard sailing with design variables being either positives or negatives, M.A. is used to convert the points to a single condition, here starboard sailing.

Seven constraints capable of removing and altering the incorrect data are used:

1. $0 \leq$ absolute [Leeway] ≤ 2
2. $0 \leq$ absolute [Trim tab] ≤ 9
3. $0 \leq$ absolute [Rudder] ≤ 12
4. if [AWA] > 0 then [Main Traveler]
5. if [AWA] < 0 then [Main Traveler] x (-1)
6. 16 < absolute [AWA] < 40
7. [Boatspeed] / [Vs_target] ≥ 0.6

where AWA corresponds to the apparent wind angle and Vs_target is the expected boat speed at such conditions.

In our example, the number of points provided by the team was 21,200. The automated filtering system removed about 40% of points from the raw sailing database.

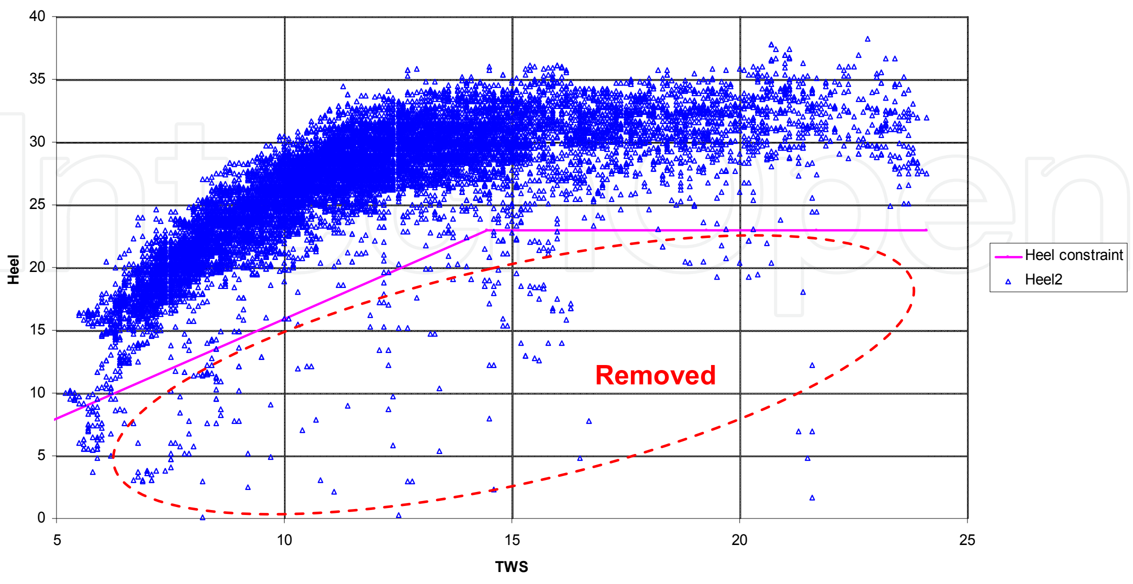

Fig 11. Filtered database shown in terms of Heel vs. TWS resulting from imposing 7 constraints

Fig 11 shows a sample point distribution (here Heel vs TWS) over the design space; the filtered points (shown in blue) are mainly gathered in a dense area but a few points are still "unrealistic" by their location in the plot, away from the main cloud area.
Based on experience, a sailing limit base line (in pink) has been drawn and all records located below the pink limit line are non-valid, non-representative sailing points. They represent 1 to 2 % of initially filtered points and can be easily removed "manually" (non-automated approach).
Similarly, two other verification plots (TWA vs TWS and Forestay vs TWS) were made. The "unrealistic" point percentage is similar to that above, with only about 1% of them needing to be removed manually. This low percentage validates our constraints and filtering procedure. The resulting database has 6,386 points.

5.3 Filling of the design/analysis space

In this analysis, the wind range selected is from 10 to 15 knots. As illustrated in Fig. 12, the experimental data, shown here in terms of Heel and Forestay settings from 10 to 11 knots, is primarily dependent on the TWS values with most data points located in a cluster or "cloud" and areas around the cloud with few or no points. The objective is to have the NN trained to provide the boat performance over a range of operating condition. Since the location of these clusters is not a priori known, it is necessary to have an approach where the NN gives adequate results in the "empty regions" of the space. This is accomplished by filling in the remainder of the space with other points. The technique consists of generating artificial points in areas of low point density (i.e. "fill-in" the space) but with their objective function value equal to some fixed "unattractive" value. Therefore, any NN function evaluation away from the "cloud" area(s) will generate a lower objective function value (if the goal is maximization of the said function, such as boat speed) in the vicinity of the points added to the dataset; preventing the optimizer to search for solutions in that area.

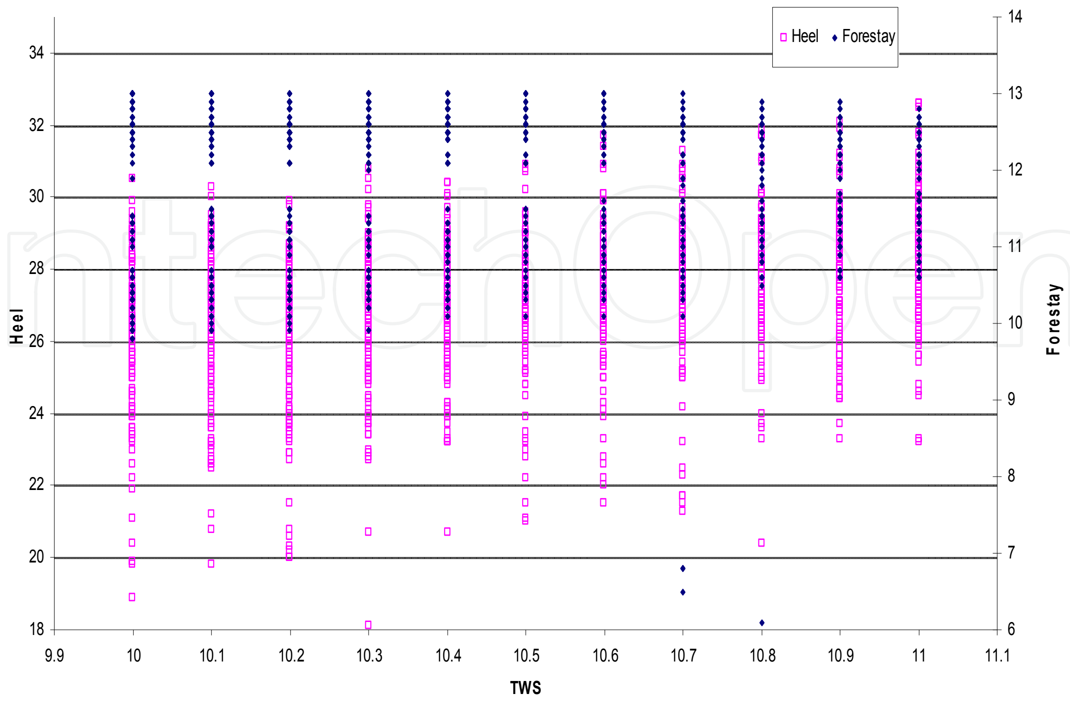

Fig. 12. Heel and Forestay settings variation with TWS between 10 and 11 knots.

The location and the quantity of these added points is the crucial element, as they need to surround the border zone of high density points without perturbing the well-defined zones of the domain, or clouds.
A data point enclosing a number "*n*" of variables is interpreted as a vector of dimension *n*. In a first step, the additional points are generated randomly over the *entire* domain. In a second step, those that end up *within* a cloud are removed. The selection (decision to keep) each of the randomly generated points is based on geometric considerations. The selection criterion is based on the largest of the distance between closest points of the experimental database, δ. In other words, if the experimental dataset is $\left(\vec{x}_i\right)_{1\le i\le N}$ with $\vec{x}_i$ a vector of dimension *n*, each vector or point has a closest neighbour and the distance between that point and its closest neighbour defined in Euclidian space is δ_i. Therefore, δ can be defined as:

$$\delta = \underset{1\le i\le N}{MAX}\left(\delta_i\right) = \underset{1\le i\le N}{MAX}\left(\underset{1\le j\le N}{MIN}\left\|\vec{x}_i - \vec{x}_j\right\|\right) \tag{45}$$

Specifically, a randomly added point $\vec{y}_i$ will be discarded if there exists a point $\vec{x}_k$ such that the distance between these two is less than *k*.δ, where *k* is a constant to be determined as discussed in the next section.
The result of this process is an experimental database which has been filled in regions away from the "cloud" and usable for generating the NN. The two questions which arise are first, what is the best value to give to *k*, and second, what value to give to the NN output, or objective function since we use the NN for optimization, at that additional point. Courouble *et al*. (2008) present a detailed analysis which answers these questions and only a summary of the results is presented here.
Since the NN is used for optimization (maximization of upwind boat speed and determination of the corresponding settings), ideally, the result obtained with the NN trained using the database including the automated filling process should be the same as that when an experienced yacht designer would restrict the sailing parameters to "appropriate ranges." This latter case is used as a *reference case* for comparison with the results obtained from the automated filling process.

5.3.1 Reference case

To preset the correct search domain to be used in the optimization, one approach is to first represent the point locations graphically like in Fig 11 and then visually define the design space boundaries by selecting, for each variable, upper and lower values, values which may vary as functions of key parameters, such as TWS. This method of pre-restraining the domain is subjective as it relies on the reader's ability to evaluate the boundary values, labor-intensive as it requires one plot for each design variable, and inadequate for complex cases where no single variable can be used to establish such bounds.
In the case of the yacht, TWS plays a key role in the actual boat speed so that by focusing on 1-knot increments in TWS, it becomes possible to define upper and lower value for the eight design variables by plotting seven similar two-dimensional graphs as functions of TWS, as in Fig.4. This approach enables us to define a reference case against which we can compare with the automated filling process where such parameter restrictions are not imposed.

5.3.2 Automated space filling

The first step is to convert the sailing database independent variables to non-dimensional values so that the database is contained in a unit hypercube of dimension *n*, with *n* = 8 in our example. Second, we generate a random set of non-dimensional points/vectors in the same hypercube. The randomly generated points are added throughout the design space and those "too close" to a valid point of the database are removed following the approach described above; only the points populating the voids are kept.

Courouble *et al.* (2008) show that 1,000 random points over the targeted space provide good accuracy. For the current dataset, the largest of the minimum distances expressed in terms of non-dimensional vectors is 0.698; for convenience we will choose δ=0.7 as the distance criterion. Therefore, the randomly generated points are kept only if they are at least at a distance of *k* x 0.7 from their neighbors. For example, for *k* = 1, about 40% of the 1000 random points are rejected with the criterion of 0.70.

5.4 Process overview

Fig. 13 presents an overview of the automated process, starting with the database containing the filtered sailing data.

Since the validation set (VS) is used for stopping the training and our interest is to train the function over the sailing telemetry, the VS will be composed of valid sailing data points only. In our case, the VS is about 300 points and all remaining points are used for the training set (TS) to which points are added following the approach described above (except in the *reference case*). The TS and VS are then used to train the NN, so that VMG can be determined instantaneously from given sailing parameters and wind conditions.

The global optimization method used here us a Genetic Algorithm (GA). GA is a search and optimization method based on the process of biological evolution, in that they involve a search from a population of solutions and not from a single point. Each iteration of a GA involves a competitive selection that penalizes poor solutions. The solutions with high fitness are recombined with others to produce members of the next generation. Reproduction and mutations are used to generate new solutions which are biased towards regions of the space for which good solutions have already been seen. The strength of the GA is that they perform well in spaces where there may be multiple local optima. The typical drawback of GA is the requirement for a rather large number of function evaluations, a requirement easily met here with the use of the NN since objective functions can be evaluated instantaneously.

The overall dataset is primarily based on the TWS going from 10 to 15 knots; consistent with sailing practice. In order to establish well defined *reference cases* to be used for evaluation of the automated approach (see above), the analysis is split in one-knot increments, starting at 10 knots. In the general case, however, such splitting would not be necessary since the NN would be capable of representing the dependency of the boat speed on TWS (as long as TWS is an independent parameter). A summary of the results is presented in the next section.

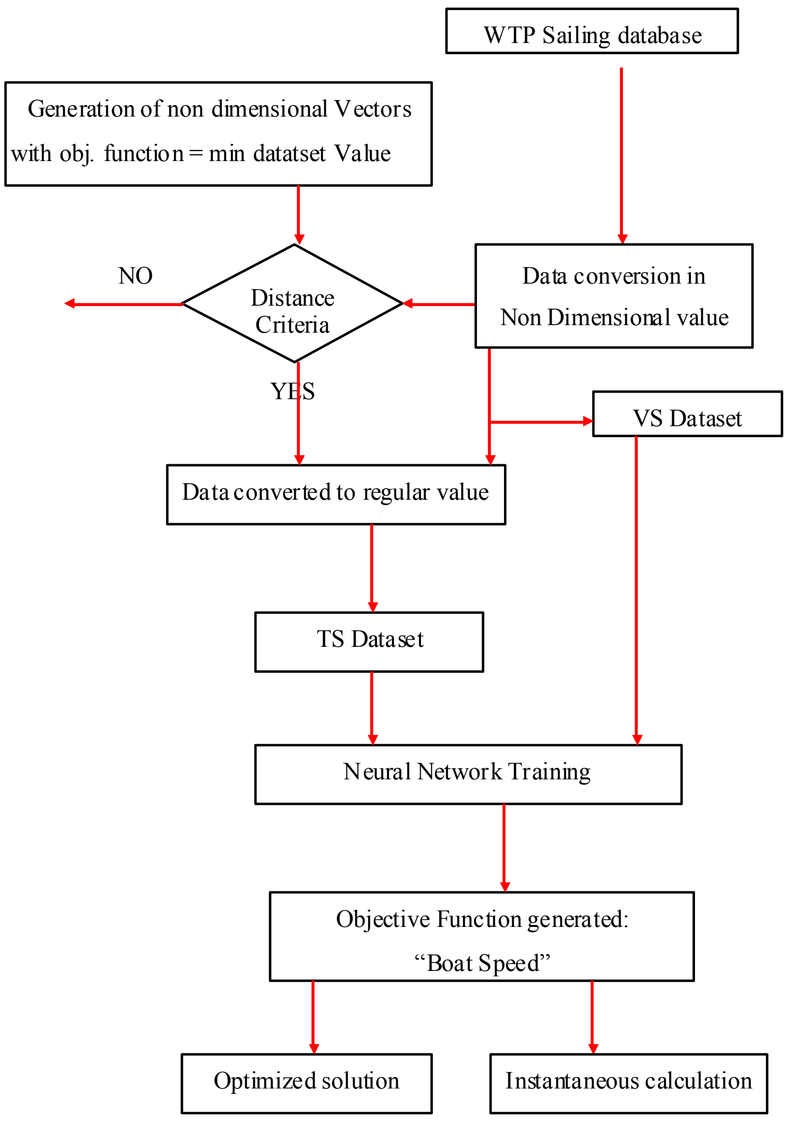
WTP Sailing database
Generation of non dimensional Vectors with obj. function = min datatset Value
NO
Distance Criteria
Data conversion in Non Dimensional value
YES
VS Dataset
Data converted to regular value
TS Dataset
Neural Network Training
Objective Function generated: "Boat Speed"
Optimized solution
Instantaneous calculation

Fig. 13. Automated process starting with the WTP sailing database and leading to a NN capable of instantaneously calculating boat speed, or which can be used to generate optimum sailing setups under varying conditions.

5.5 Results and Discussion

5.5.1 Effects of Optimum Distance Criterion and Number of Added Points

All optimizations shown here are performed with a population size of 5000 with a stopping criterion of 100 generations. One optimum is obtained in about eight minutes on a regular personal computer, corresponding to half a million operations of the trained function.
As noted above, the results obtained from the automated database processing are compared with a *reference case* where the search space has been greatly restricted to fit the available data. In this section, all added points are assigned a boat speed of 0. Table 4 presents the effect of the number of added points (before those too close to sailing data clusters are removed automatically) and distance criterion in comparison with the optimum values found in the reference case. With no added points, the optimum solution of run 1 is off by about 12% when comparing the value with the reference case. This large error is due to

"waves" in the network near the edges of data clusters because of lack of data outside these clusters and because some of the best sailing conditions are actually near some of these edges of data clusters.

Increasing gradually the number of points from one hundred to one thousand, the difference drops to 2.1%. For a margin within 2% accuracy, 1000 points added over the six thousand sailing data points, means a minimum of 16% of added points is required the whole dataset. Theoretically in training a NN there is no limitation in terms of size, so adding more points would then not be a problem. Therefore, to improve the non-valid domain coverage, a higher percentage of added points (such as up to 20% of the total original dataset size) appears to be a safe margin to start with.

The second element to investigate is the distance criterion. Our initial distance was based on *k* = 1 so as to insert additional points not too close to the valid sailing points. Results for two additional datasets are shown in Table 4, both with one thousand additional points but differing by their distance criterion (*k* coefficient). In the first dataset, the distance criterion is reduced to 75% (*k*=0.75); in the second dataset, the distance is reduced by 50% (*k*=0.5). For *k*=0.5, the dataset is within 1% of the target solution; *k*=0.75 shows a difference of 3.5%, which is still acceptable. This solution, however, is marginal, with parameters like heel angle or forestay tension being lower than typical values. This discrepancy is likely due to the fact that an objective function, boat speed, of zero was set of additional points which also creates "waves" in the NN near the edges of clusters. The value to use for these additional points is discussed below.

Case	# of added points	*k*	Error %*
1	0	1	12.5
2	100	1	7.2
3	500	1	5.5
4	1000	1	2.1
5	1000	0.75	3.5
6	1000	0.5	0.7

Table 4. Comparison of optimum obtained with the automated database processing with that obtained with the reference case over a more restrictive design space. TWS: 10-11 Knots**.**

5.5.2 Effect of Boat Speed for Added Points and Impact on Minimum Distance Criterion

To prevent the optimizer from searching in the low point density area, we assigned boat speed of zero for all the added points. Populating non-valid areas of the domain by non-sailing data points with speed value set to zero is radical but efficient, especially if the domain to be covered is very large for the amount of added points available. But as we get closer to the sailing data points, we risk that locally the function value be altered. Here, the sailing dataset shows boat speed values ranging from 7.1 to 11.3 knots, and inserting a point with zero value creates a radical damping of 170 % in the function value and may remove potential attractive solutions in the optimization process or create waves near the edges of sailing data clusters. To prevent that phenomenon near the edges, the function value for the added points is set to be equal to the overall minimum boat speed of the dataset (i.e. 7.1

* compared with reference case

knots) instead of 0.0 as set in earlier analyses. From that statement we created two new datasets with similar number of points added (1000) and *k*=1, but with function values equal to the lower boat speed value (7.1). The solutions when compared with the reference case show differences within 1.2 %, which is even better than the 3% obtained earlier.
In regard to the settings (eight sailing variables), there is also more consistency between the two solutions. On a sailboat many different set ups can provide nearly the same boat speed, as long as they are coherent. In this case, although the TWS are not exactly the same the setting numbers are globally in the same range which is an important factor to emphasize, meaning the solutions have been optimized for both domains in a similar zone. Courouble *et al*. (2008) compare these solutions in more detail and show that although the solutions are within few percent, a closer look at the optimum design variables for heel, forestay, rudder and trim tab show some differences between the two solutions, but similar trends, suggesting that several combinations of sailing parameters may be used for optimum boat speed. The results demonstrate that the method offers excellent potential for identifying the areas of interests for further investigation. Although not performed in the present study, the use of a multi-island genetic algorithm would allow a user to explore such areas systematically.

6. References

Agatonovic-Kustrin, S.; Zecevic, M.; Zivanovic, L.; and Tucker, I.G. (1998) "Application of Neural Networks for Response Surface Modeling in HPLC Optimization," *Analytica Chimica Acta,* Vol. 364, pp. 265-273.

ANSYS, Inc. (2009) "ANSYS ICEM CFD" [Online] available at <http://www.ansys.com/products/icemcfd.asp>, accessed 01 June 2009.

Bertram, V.; Mesbahi, E. (2004) "Estimating Resistance and Power for fast Monohulls Employing Artificial Neural Nets," 4th Int. Conf. High-Performance Marine Vehicles (HIPER), Rome.

Besnard, E.; Schmitz, A.; Kaups, K.; Tzong, G.; Hefazi, H.; Chen, H.H.; Kural, O.; and Cebeci, T (1998) "Hydrofoil Design and Optimization for Fast Ships," *Proceedings of the 1998 ASME International Congress and Exhibition*, Anaheim, CA.

Besnard, E.; Schmitz, A.; Hefazi, H.; and Shinde, R. (2007) "Constructive Neural Networks and their Application to Ship Multi-disciplinary Design Optimization," *Journal of Ship Research*, Vol. 51, No. 4, pp. 297-312.

Blanchard, B. and Fabrycky, W. (1997) *Systems Engineering and Analysis*, 3rd Ed., Prentice Hall.

Bishop, C. (1995) *Neural Networks for Pattern Recognition*, Oxford University Press.

Bourquin, J.; Schmidli, H.; van Hoogevest, P.; and Leuenberger, H. (1998) "Advantages of Artificial Neural Networks (ANNs) as Alternative Modeling Technique for Data Sets Showing Non-linear Relationships Using Data from a Galenical Study on a Solid Dosage Form," *European Journal of Pharmaceutical Sciences*, Vol. 7, pp. 5-16.

Cybenko, G. (1989) "Approximation by Superpositions of a Sigmoidal Function, " *Mathematics of Control, Signal and Systems*, Vol. 2, Issue 4, pp. 303-314

Courouble, F.; Besnard, E.; and Schmitz, A. (2008) “Application of Constructive Neural Networks to America’s Cup Racing Yacht Performance Optimization,” Paper presented at the MDY'08, 3rd Symposium on Yacht Design and Production, Madrid Spain.

Danõşman, D. B.; Mesbahi, E.; Atlar, M. and Goren, O. (2002) "A New Hull Form Optimization Technique for Minimum Wave Resistance," 10th International Maritime Association Mediterranean Congress (IMAM), Crete

Dassault Systèmes SIMULIA (2009). "iSIGHT – Integrate, Automate, and Optimize your Manual Design Processes," [Online] available at < http://www.simulia.com/products/isight.html >, accessed 01 June 2009.

Dutt, J. R.; Dutta, P. K.; and Banerjee, R. (2004) "Optimization of Culture Parameters for Extracellular Protease Production from a Newly Isolated *Pseudomonas* sp. using Response Surface and Artificial Neural Network Models," *Process Biochemistry,* Vol 39, pp. 2193–2198.

Fahlman, S.E. (1988) "Faster-Learning Variations on Back-Propagation Learning: An Empirical Study," in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

Fahlman, S. E. and Lebiere, C. (1990) "The Cascade-Correlation Learning Architecture, " *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

Gomes, H. M.; and Awruch, A. M. (2004) "Comparison of Response Surface and Neural Network with other Methods for Structural Reliability Analysis," *Structural Safety,* Vol. 26, pp. 49-67.

Gougoulidis, G. (2008), "The utilization of Artificial Neural Networks in Marine Applications: An Overview". Journal of Naval Engineering, Vol. 120, No.3, pp 19-26, 2008.

Hefazi, H *et al*. (2002) "CFD Design Tool Development and Validation, CCDoTT FY00 Task 2.8, " Center for the Commercial Deployment of Transportation Technologies, Long Beach, CA., [Online] available at <ftp://www.foundation.csulb.edu/CCDoTT/Deliverables/2000/Task%202.8/task2.8_1.pdf>, accessed 01 March 2009.

Hefazi, H *et al*. (2003) "Computer Software Product End Item, Deliverable 2, Optimization Tool Development Based on Neural Networks Computer DCI-MCCR-80700 report," Center for the Commercial Deployment of Transportation Technologies, Long Beach, CA. [Online] available at <ftp://www.foundation.csulb.edu/CCDoTT/Deliverables/2002/task%202.20/task%202.20_opttools%20FY%2002.pdf>, accessed 01 March 2009.

Hornik, K. (1991) "Approximation Capabilities of Multilayer Feedforward Networks, "*Neural Networks*, Vol. 4, Issue 2, pp. 251-257.

Jain, P.; and Deo, M. C. (2005) "Neural Networks in Ocean Engineering," *Ships And Offshore Structures (SAOS 2006),* Vol. 1, Issue 1, pp. 25-35.

Koushan, K. (2003) "Automatic Hull Form Optimization Towards Lower Resistance and Wash using Artificial Intelligence," FAST 2003 Conference, Ischia, Italy.

Koh, L.; Janson, C-E, Altar, M.; Larsson, L.; Mesbahi, E.; and Abt, C. (2005) "Novel Design and Hydrodynamic Optimization of a High Speed Hull Form," 5th International Conference on High Performance Marine Vessels, Shanghai.

Kwok, T. Y. and Yeung, D. Y. (1993) "Theorical Analysis of Constructive Neural Networks," *Technical Report HKUST-CS-93-12*, Hong Kong University of Science and Technology.

Kwok, T. Y. and Yeung, D. Y. (1997a) "Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems," *IEEE Transactions on Neural Networks*, Vol. 8, Issue 3, pp. 630-645.

Kwok, T. Y. and Yeung, D. Y. (1997b) "Objective Function for Training New Hidden Units in Constructive Neural Networks," *IEEE transactions on Neural Networks*, Vol. 8, Issue 5, pp 1131-1148.

Lahnajärvi J.J.T. *et al.*(2002) "Evaluation of Constructive Neural Networks with Cascaded Architectures," *Neurocomputing*, Vol. 48, pp 573-607.

Lee, J.; and Hajel, P. (2001) "Application of Classifier Systems in Improving Response Surface based Approximations for Design Optimization," *Computers and Structures*, Vol. 79, pp. 333-344.

Lehtokangas, M. (1999) "Fast Initialization for Cascade-Correlation Learning," *IEEE Transactions on Neural Networks*, Vol. 10, nº 2.

Lin, C.Y.; and Wu, W.H. (2002) "Niche Identification Techniques in Multimodal Genetic Search with Sharing Scheme," *Advances in Engineering Software,* Vol. 22, pp. 779-791.

Maisonneuve, J.J. (2003) "Chapter 7: Applications Examples from Industry," in *Optimistic - Optimization in Marine Design*, 2nd edition, Birk, L., and Harries, S. (Editors), Mensch & Buch Verlag.

Mesbahi, E.; Bertram, V. (2000) "Empirical Design Formulae Using Artificial Neural Nets," COMPIT'2000, Potsdam.

Parametric Technology Corporation (2009) "PTC : Pro/ENGINEER," [Online] available at <http://www.ptc.com/appserver/mkt/products/home.jsp?k=403>, accessed 01 June 2009.

Prechelt, L. (1997) "Investigation of the CasCor Family of Learning Algorithms," *Neural Networks*, Vol. 10, No. 5, pp 885-896.

Prechelt, L. (1998a) "Automatic Early Stopping Using Cross-Validation: Quantifying the Criteria," *Neural Networks*, Vol. 11, Number 4, 761-767

Prechelt, (1998b) "Early Stopping : But When?," in *Neural networks : Tricks of the Trade,* Lecture Notes In Computer Science, Vol. 1524, pp 55-69, Orr, G.B., and Mueller, K.-R., eds.

Sarle, W.S., ed. (2002) "Neural Network FAQ, Usenet newsgroup comp.ai.neural-nets," [Online] available at <ftp://ftp.sas.com/pub/neural/FAQ.html>, accessed 01 June 2009.

Schmitz, A. (2007) *Constructive Neural Networks for Function Approximation and their Application to CFD Shape Optimization*, Ph.D Thesis, Claremont Graduate University.

Tekto, I.; Villa, A. (1997) "An Enhancement of Generalization Ability in Cascade Correlation Algorithm by Avoidance of Overfitting/Overtraining Problem", Neural Porcessing Letters, Vol. 6, pp 43-50

Takayama, K.; Fujikawa, M.; Obata, Y.; and Morishita, M. (2003) "Neural Network based Optimization of Drug Formulations," *Advanced Drug Delivery Reviews*, Vol. 55, pp. 1217–1231.

Todoroki, A.; and Ishikawa, T. (2004) "Design of Experiments for Stacking Sequence Optimizations with Genetic Algorithm using Response Surface Approximation," *Composite Structures,* Vol. 64, pp. 349-357.

Treagold, .K.; Gedeon,T.D. (1999) "Exploring Constructive Cascade Networks," *IEEE Transactions on Neural Networks*, Vol. 10, No. 6.

Vanderplaats, Muira & Associates Inc. (1995), *DOT Users Manual, Version 4.20*, VMA Engineering.

Machine Learning
Edited by Yagang Zhang

ISBN 978-953-307-033-9
Hard cover, 438 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Machine learning techniques have the potential of alleviating the complexity of knowledge acquisition. This book presents today's state and development tendencies of machine learning. It is a multi-author book. Taking into account the large amount of knowledge about machine learning and practice presented in the book, it is divided into three major parts: Introduction, Machine Learning Theory and Applications. Part I focuses on the introduction to machine learning. The author also attempts to promote a new design of thinking machines and development philosophy. Considering the growing complexity and serious difficulties of information processing in machine learning, in Part II of the book, the theoretical foundations of machine learning are considered, and they mainly include self-organizing maps (SOMs), clustering, artificial neural networks, nonlinear control, fuzzy system and knowledge-based system (KBS). Part III contains selected applications of various machine learning approaches, from flight delays, network intrusion, immune system, ship design to CT and RNA target prediction. The book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners.