THE CONTRIBUTION TO MODELLING, DESIGNING AND IMPLEMENTING MULTIPLE NEURAL NETWORKS: AUTO-ADAPTATION OF DATA PROCESSING SYSTEM BY TREELIKE-DIVIDE-TO-SIMPLIFY APPROACH USING COMPLEXITY ESTIMATION METHODS

By

Mariusz Rybnik

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
UNIVERSITY PARIS XII
CRETEIL, FRANCE
DECEMBER 2004

UNIVERSITY PARIS XII
DEPARTMENT OF
COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**The contribution to modelling, designing and implementing Multiple Neural Networks: Auto-adaptation of Data Processing System by Treelike-Divide-to-Simplify Approach using Complexity Estimation methods**" by **Mariusz Rybnik** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: December 2004

External Examiner: ______________________
Name

Research Supervisor: ______________________
Kurosh Madani

Examing Committee: ______________________
Name

____
Name

Table of Contents

List of Figures

Index of symbols

Symbol	Signification
x	input set of features
X	space of input values
y	output of real system
$\hat{y}$	output predicted by model
D_{train}	training data provided for prediction
$f(x;\theta)$	prediction model (function of input features x and model parameters)
θ	set of estimated parameters of model
$S(\cdot)$	score (cost) function of model
C	set of possible class values
$c_1, ..., c_m$	class values
d	scalar distance (between real output y and predicted output y)
ε	Bayes error
μ	arithmetical mean
Σ	covariance matrix
ε_u	Chernoff bound
ε_b	Bhattacharyya bound
D	divergence
d_E	Euclidean distance
d_{WE}	weighted Euclidean distance
d_{MH}	Manhattan distance
d_{MI}	Minkovsky distance
d_{MH}	Mahalanobis distance

Symbol	**Signification**
d_T	Tchebyshev distance
d_C	Canberra distance
m_D	Matusita distance
d_{norm}	Normalized mean distance
J	Shannon measure of entropy
CDM	Class Discriminability Measure
$S_{H(l)}$	degree of separability in cell l
S_H	overall degree of separability
AS_H	purity
S_{NN}	overall separability of data
AS_{NN}	neighborhood separability
ASE	collective entropy
f_1	Fisher discriminant ratio
S_w	within class scatter matrix
S_b	between class scatter matrix
S_m	mixture scatter matrix
J_1, J_2, J_3	measures manipulating the scatter matrices
rd	length of class overlapping
v_o	volume of overlap region
S_{IC}	average inter-cluster distance
S_{BC}	average between-cluster distance
r_h	radius of hypersphere
O_S	overlap sum
w	Neuron weight
b	Neuron bias
v	Neuron induced field (activation potential)
$\Phi(\cdot)$	Activation function
u	Linear combiner output

Symbol	**Signification**
z^{-1}	Unit delay operator
$M(\cdot)$	Model function
$f(\cdot)$	unknown system activity function
$\hat{d}_k$	Estimation of output value of neuron k
e_k	Error of neuron k
E	Cost function (index of performance)
η	Rate of learning
EB	Energy of Boltzmann machine
$F_I(x)$	Ensembled average function
g_i	Gating weight in Mixture of Experts approach

INTRODUCTION

This work is concentrated on developing a semi-automatic data processing structure, which can reduce complexity of processing tasks by techniques similar to task decomposition. This data-processing system is called **Treelike Divide To Simplify** (**T-DTS** in short). This idea is represented in part by the name - we *divide* the task in order *to simplify* the processing. So - the most important idea of the work is connected to Task Decomposition techniques called also Divide to Conquer (see 2.1), that offer potential advantages over one-piece problem processing. Processing tasks are of very various origins. Sometimes there exists a relation between processing time and size of data that makes task decomposition interesting, because of the gain of performance, possibly in temporal or processing quality aspects, that is - to decrease processing time and to increase processing quality. We are also trying to develop a task difficulty evaluation in order to minimalize the need for user intervention in decomposition. The processing tasks on which we are concentrated in this work are model identification and classification, because the results are relatively easy to interpret and compare.

The T-DTS system can also attune itself in part to classification processing task by using a family of statistical methods called Complexity Estimation techniques. The goal is to estimate the difficulty of classification task and modify the processing algorithm so that can process the task more efficiently. The modification may include among others: 1) task decomposition up to some degree dependant on measurements (i.e. up to what degree we want to decompose the task), 2) the choice of appropriate processing structure, 3) the choice of individual processing modules for each subset of decomposed data. These concepts are quite different and complicated so we are presenting here applications only for the first case. The reason for using Complexity Estimation as diagnostic techniques is to minimalize the need for user intervention and to process data efficiently and automatically. We are linking together statistical methods (in order to well recognize the character of the task) and universal flexible data processors (Artificial Neural Networks). With such cooperation we are expecting to make these objectives a little closer.

T-DTS has modular structure. The modules are based on Artificial Neural Networks (ANN), because ANN are universal data processors and are naturally well suited for modular processing. ANN could be considered as universal data processors because they are on intersection of mathematics, statistics and informatics sciences, so we can obtain the properties from all these disciplines when necessary. T-DTS decomposes the processing task into clusters in recurrent way and process them separately in order to ease and speed up the processing, as well as in order to use less complicated processing modules. In most cases decomposition of a problem will also result in decreasing overall processing complexity for complicated tasks that are difficult to process as a whole, due to great complexity or excess processing times. The gain incorporates also simplification of structure of task, that lead to less complicated processing modules and possibly less parameters to control. Task decomposition makes also possible parallel processing (on parallel computing machine or network of machines) that result in great reduction of processing time if the splitting/combining phases are negligible comparing to the processing phase.

The thesis contains two main parts: the first presents state of the art in domains close to the work included in the thesis, while the second concerns our work and presents theoretical basis of T-DTS approach and examples of practical applications and computer T-DTS implementation. Part concerning state of the art contains three chapters. First chapter describes Artificial Neural Networks in order to give ideas about their universality and particular usefulness in modular approach. Chapter two is concerned with modular algorithms as closely connected to task decomposition techniques. Chapter three presents complexity estimation that is used in part of experiments in order to auto-organize the modular T-DTS structure and presents useful perspectives for the further development of auto-organizing structure. T-DTS concerned part is composed of two chapters, numbered consequently 4 and 5. T-DTS approach is presented in detail in chapter four, including methods and strategies for building the modular structure, decomposition of databases and finally processing and obtaining the results. Chapter five is aimed at evaluating the universality of T-DTS approach, by showing its applications to different classes of processing problems. It is split into two main sections: 5.1 that concerns model identification problems and 5.2 which presents classification problems. Finally chapter six is a presentation of T-DTS computer implementation, starting from static structure of implemented functions, user interface description, ending up with controllable parameters structure. The practical part is followed by global conclusions that summarize up essence of the work, and give further perspectives for the future development of T-DTS system. This is pursued by list of publications and annexes contenting additional information of less importance. The references to works related and used in the thesis end up the text.

PART I - STATE OF THE ART

Chapter 1

Artificial Neural Networks

1.1 Introduction

This chapter is concentrated on Artificial Neural Networks. ANN are the substance and basis of T-DTS solution. ANN are used in our approach in order to split up a problem into several sub-problems and also to process (usually modelize) the sub-problems. As problem is represented by data, so from technical point of view ANN operate on sets of data, splitting them up and processing them.

Artificial Neural Networks are especially useful in modular solutions, as they are well suited by their nature itself to modular and universal applications. *That simplifies connecting and cooperation* between modules. These properties are desired in T-DTS approach, so processing modules were based entirely on ANN. Decomposition modules could be based on any efficient clustering algorithm. We have chosen to use for decomposition ANN structures similar in activity to well-known k-means clustering algorithm, however presenting more possibilities when it comes to cluster organization. The advantage of our approach is its flexibility - it's easy to replace one part of system with another algorithm that will enlarge the possible usage of whole system.

Artificial Neural Networks (ANN) are grand and powerful data processing tools, being basically a cross between mathematics and informatics. Their most important features are distributed processing and adaptive morphology. Distributed processing is derived from decentralized and modular structure of ANN. They are connected to many other disciplines, like: neurobiology, mathematics, computer science, statistics, physics and engineering. Artificial neural networks are applied with success in many areas: modelling, classification, pattern recognition, signal processing, time series analysis and others.

Section 1.2 presents main ideas and properties concerning ANN. Section 1.3 introduces general ideas about architecture of layered networks. Section 1.4 presents learning tasks applicable with machine learning and in particular ANN. Section 1.5 will exhibit learning strategies that are most popular in the area of Artificial Neural Networks. Conclusion in section 1.6 ends this chapter.

Additional information about ANN is contained in appendixes: Appendix B is a brief description of ANN history, appendix C presents structural similarities of artificial and biological neurons and as well discuses the artificial neuron activation functions. Appendix D is a presentation of general ideas of most popular ANN learning algorithms, appendix E enumerates industrial applications of ANN, and finally appendix F is concerned with system identification learning tasks.

1.2 Main ideas

Artificial Neural Networks are inspired by biological nervous systems. Similarly to biological neural networks (i.e. human brain), ANN are composed of relatively simple elements called neurons. Neurons are connected creating networks. Networks can be very large and have complicated structure. Network function is determined widely by the connections between neurons. Neural networks work in parallel way, which is a very important feature for many computation tasks. In nature, NN perform many various tasks, beginning from simple reaction to environment change in one-cell organisms up to still little known functions of human brain. ANN also have this feature and although simple comparing to Neural Networks in the nature, they are very powerful and universal computation instrument. ANN similarly to natural NN usually learn through example. It means that first, a network of appropriate structure is built and then the network's behavior is modified to meet the expectations. The mechanism is similar to feedback known in electronics and other areas. It is depicted on figure 1.1.

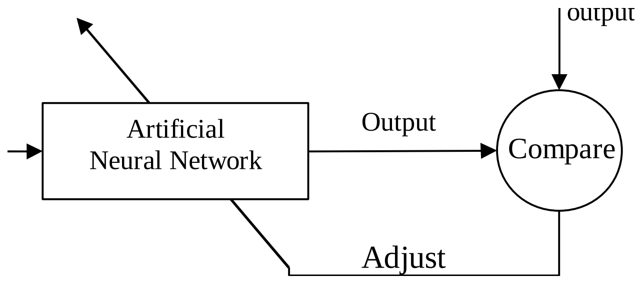


Figure 1.1: Neural network learning by example

Depending on the comparison between an output from ANN and a requested output (known as target) we adjust its connections (called connection weights) and possibly structure in a way which is expected to ameliorate its performance. This mechanism is called supervised learning, as it needs a supervisor who will compare the results with expected values and perform an action depending on it. ANN can be trained to solve problems that are difficult for human beings and for conventional computing methods. They have proven their efficiency in a number of applications, among others: pattern recognition, identification, classification, speech, vision and control systems. Two other important ways of training ANN are called unsupervised learning and direct design methods. Unsupervised learning can be used for instance to identify and localize groups of data. Certain types of linear networks as Hopfield networks are designed directly. Unsupervised learning and direct design methods are thoroughly described in subsequently sections 1.5.3. In summary there is a great variety of possible network architectures and learning methods which are used in very different areas of applications.

Artificial Neural Networks have following properties:

1. **Uniformity of analysis and design** - ANN are universal data processors, so it is possible to share theories, learning algorithms in different applications of neural networks. It is possible also to connect the different types of ANN modules. This paradigm is known as Multiple Neural Network (DU) [Arb89]. It is very important to this work.

2. **Nonlinearity** - when network includes nonlinear artificial neurons it is nonlinear. Nonlinearity is important property, when underlying physical mechanism responsible for generation of the output signal is inherently nonlinear.

3. **Input-output mapping** - ANN are capable of constructing a mapping between input and output, based on the training examples. From a statistical point of view, the process is similar to nonparametric statistical inference (model-free estimation) [HMS01] and from biological viewpoint tabula rasa

learning [GBD92].

4. **Adaptivity** - Some ANN are designed to be able to adapt their work to minor changes in operating environment. They can be used in constantly changing (non-stationary) environment where they change their synaptic weights in real time. To take a full benefit of adaptivity, the ANN should ignore spurious (short) disturbances and respond to meaningful changes in environment. The problem is referred to as stability-plasticity dilemma [Gro88].

5. **Fault tolerance** - ANN implemented in hardware form are prone to faults, like any electronic device. However, due to distributed nature of ANN, usually damage has to be extensive, before the overall response of system is impaired seriously. In general ANN exhibits gradual degradation of performance rather than catastrophic failure.

6. **VLSI implementability** - due to their distributed and parallel structure, ANN are potentially well suited for implementation using Very-Large-Scale-Integrated technology circuits [Arb89].

7. **Evidential response** - in the context of pattern classification, ANN can provide not only predicted response, but also the confidence about the decision made. This information can be used later to improve the classification performance of network by rejecting ambiguous patterns or to present the doubtful patterns to another system (for example human expert).

8. **Neurobiological analogy** - *"The design of a neural network is motivated by analogy with the brain, which is a living proof that fault tolerant parallel processing is not only physically possible but also fast and powerful."* [Hay99].

It is still useful to research the structure of brain and natural neural systems, as they have still many mysteries and solution to offer, created not by hand of a man, but by evolution.

9. **Statistical analogy** - ANN ideas are frequently known in the optimalization and statistics area [Sar94]. The use of parallel and distributed representation of data processing elements makes them however easier to integrate and connect with other ideas.

In the morphology ANN one can distinguish several concepts:

- Artificial neurons
- Structure (organization) of neurons
- Learning algorithms

The concept of artificial neuron as a heritage of biological neuron is described in detail in appendix C. Section 1.3 presents Artificial Neural Networks architectures. Section 1.4 will present learning tasks in general manner, while section 1.5 will give details on machine learning.

1.3 Artificial Neural Networks' architectures

Neural layer is a combination of several (sometimes also only one) neurons working in parallel way. In general, an ANN consists of several layers of neurons. In fact large part of functionality of ANN depends on the connections between neurons, propagation of values between them (current similar to biological neural impulses) and whole layers. Depending on the direction of information flow, ANN can be classified as follows: Feedforward networks - Information in that structure flows only forward, i.e. layer doesn't contain internal feedback paths. The simple form of such network is single-layer feedforward network. It contains input layer of source nodes that projects onto a output layer of neurons. An example of such network is depicted in the Figure 1.2.

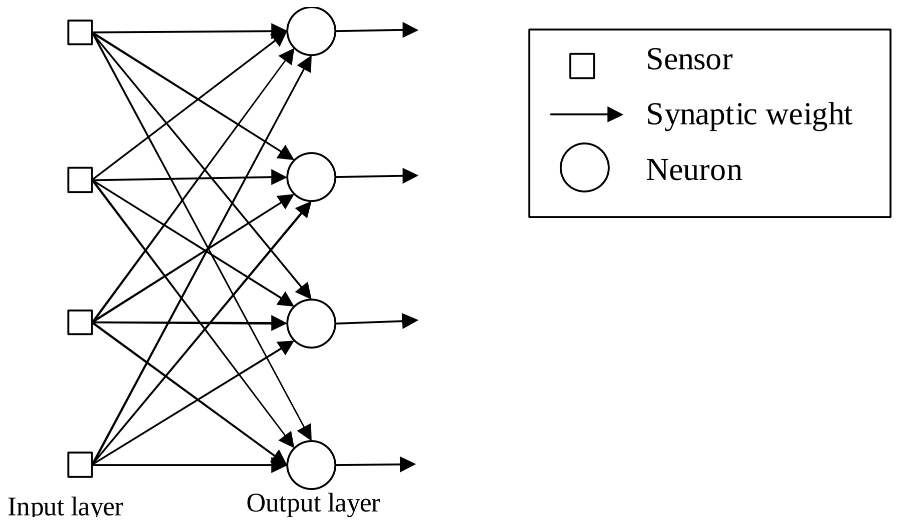


Figure 1.2: Example of single layer feedforward network

Multilayer feedforward networks - the structure consists of not only input and output layers, but also one or more hidden layers (consisting of hidden neurons also referred as hidden units). By adding hidden layers a network is enabled to extract higher-order statistics. Extra set of synaptic connections enable the network to acquire a global perspective [CS92]. The network is fully connected if each neuron from the preceding layer has a connection with all neurons forming the succeeding layer. If some connections are missing the network is partially connected. An example of fully connected ANN feed-forward structure with one hidden layer is depicted on the Figure 1.3.

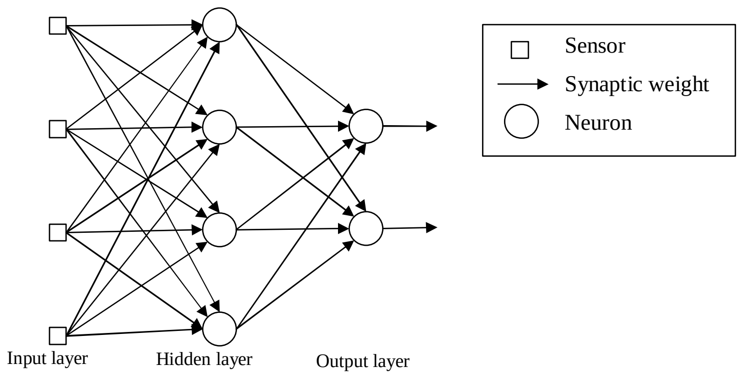


Figure 1.3: Example of multilayer feedforward network structure

Recurrent networks are networks, which contain at least one internal feedback loop. A self feedback loop is a delayed connection from a neuron to itself. Recurrent networks can contain hidden layers. They are able to show dynamical behavior (depending on the previous conditions). An example of recurrent network is depicted on the Figure 1.4

Feed-forward ANN cannot perform temporal computation like recurrent ANN, but building and training of recurrent ANN is more difficult. In the next section learning tasks will be presented to show the most popular goals of data processing techniques.

1.4 Learning tasks

Artificial Neural Networks are used in many tasks. In this chapter, learning tasks taxonomy is presented showing the main application classes. The specific applications of ANN are presented in the Annex C.

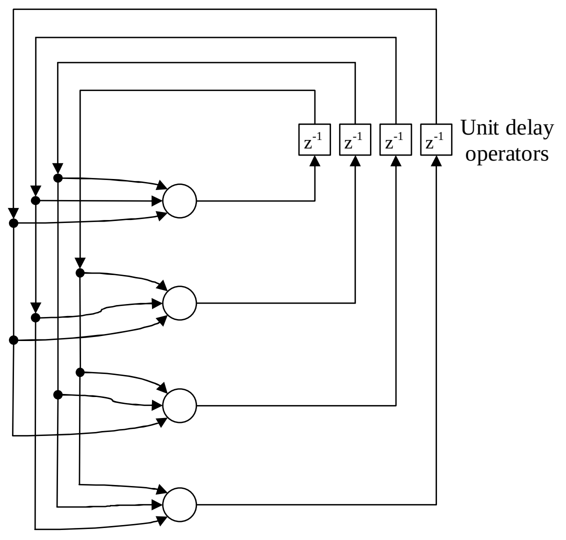


Figure 1.4: Example of recurrent network with no self-feedback loops

1.4.1 Classification

Classification is a process where a signal is assigned to one of prescribed classes (categories). Neural network first undergoes a training session, during which a network is repeatedly presented a set of input pattern together with the class to which each pattern belongs. Later a network is presented with input patterns unlearnt and ANN is able to classify (attach a class) the patterns using the knowledge extracted from the learning session.

The patterns are represented as points in multidimensional (feature) decision space. The decision space is divided into **decision regions**; each one is assigned with a class. The decision boundaries are determined during the learning process and divide the **decision regions**.

Classification system using ANN can take two forms:

- The system consists of two parts: first part (unsupervised network) is used to do feature selection (transformation from input pattern x into feature vector y). The transformation may result in dimensionality reduction (data compression), which is expected to ease the classification task. The second part (supervised network) is a classifier which maps feature vector y to class labels.

- The system is designed as a single multilayer feedforward network using a supervised learning algorithm. In this case, the task of feature extraction can be considered as performed in explicit way by hidden layer(s) of the network.

1.4.2 Pattern association

Association is a process of linking objects. Network is required to respond to a key pattern x_k with a memorized pattern y_k:

$$x_k \rightarrow y_k, \quad k = 1, 2, ..., q \tag{1.4.1}$$

Association takes two forms: autoassociation and heteroassociation. In autoassociation task, network is required to store a set of patterns and respond to a distorted version of original pattern with that particular pattern. In heteroassociation task, key pattern and memorized pattern can be completely different (for example dimensionality).These two patterns are paired (associated to each other). The associative memory is a set of such pairs.

There are two phases of operation of an associative memory:

- **Storage phase** - when network is trained by presenting the patterns;
- **Recall phase** - when network is presented with a key pattern and responds with a memorized pattern.

When associative memory responds with wrong pattern, it is said to make an **error in recall**. The number of patterns stored in memory provides a direct measure of the storage capacity of the network. In designing of associative memory, the goal is to enhance the storage capacity and efficiency and minimalize the recall errors.

1.4.3 Function approximation

The aim here is to create a model function $M(\cdot)$ which imitates the activity of unknown system. The activity of system is marked as function $f(\cdot)$, which is usually unknown and can be approximated by analyzing the input-output relation of system.

$$d = f(x)$$

d are outputs of real system $f(\cdot)$ to inputs x

The approximation can be exploited in two ways:

- **system identification**, when model $M(x)$ imitates explicitly a function $f(x)$. Both original system and model are given input values x, the responses are compared to determine error e, which is used to train the system identification model. The model $M(x)$ produces in this case estimations of output values, given real input values x.

  $$\hat{d} = M(x)$$

  The performance of system is measured by taking into account absolute difference between responses of system and model to input patterns:

  $$\|M(x) - f(x)\|$$

- **inverse system**, when model $M(x)$ imitates the function $f^{-1}(x)$, which is reverse to system behavior. The model $M(x)$ produces thus estimations of input values , given real output values y of system:

$$\hat{x} = M(y)$$

  The system's performance is measured by taking into account absolute difference between responses of model (estimated input pattern) and real input pattern (unknown to model):

$$\|M(y) - x\|$$

System identification and inverse system computation are detailed in Annex F.

1.4.4 Filtering

The filtering is a process of extracting interesting properties from a set of noisy data. Filters can be used to three basic tasks:

- **filtering** - extraction of information about a quality of interest at discrete time n by using data measured up to and including time n.

- **smoothing** - extraction of information about a quality of interest at discrete time n by using data measured before and after time n. In statistical sense, smoothing is expected to be more accurate than filtering, because it can use more data.

- **prediction** - forecasting information about a quality of interest at discrete time $n + n_o$ by using data measured up to and including time n. This task is most difficult.

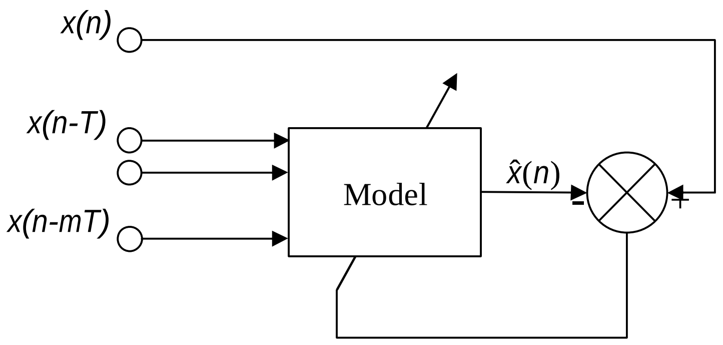


Figure 1.5: Nonlinear prediction

A spatial form of filtering is known as **beamforming**. It is used to distinguish between the spatial properties of a target signal and background noise.

Beamforming is commonly used in radar and sonar systems where the primary task is to detect and track a target of interest in the presence of noise and interfering signals.

Next section gives details on learning algorithms, which lead to accomplishment of the learning tasks.

1.5 Learning algorithms

Learning algorithm in ANN approach is an algorithm which is used to modify the weights, biases and/or structure of ANN. By these modifications, the performance of ANN is expected to be improved. One of possible morphology of learning algorithms is as follows:

- Learning with a teacher (supervised learning)
  - error correction learning

        * incremental training
        * batch training

- Learning without a teacher
    - Reinforcement learning/neurodynamic programming
    - Unsupervised learning
    - Competitive learning
- Memory based learning
- Boltzmann learning

1.5.1 Learning with a teacher

Learning with a teacher is also called supervised learning. To adjust the parameters of learning system, one uses an error signal, which is a difference between actual system response and desired (optimal) response given by teacher.

In **incremental training**, weights and biases are updated each time a new input vector and corresponding target is presented to the network; i.e. sequentially. Incremental learning can be applied to both static and dynamic networks, but is most commonly used with dynamic networks, such as adaptive filters. In **batch training**, weights and biases are updated after whole learning database (learning examples and corresponding targets) was presented; i.e. concurrently.

1.5.1.1 Error correction learning

Error of neuron k with m synapses given a signal vector $x(n)$ at time step n is equal to:

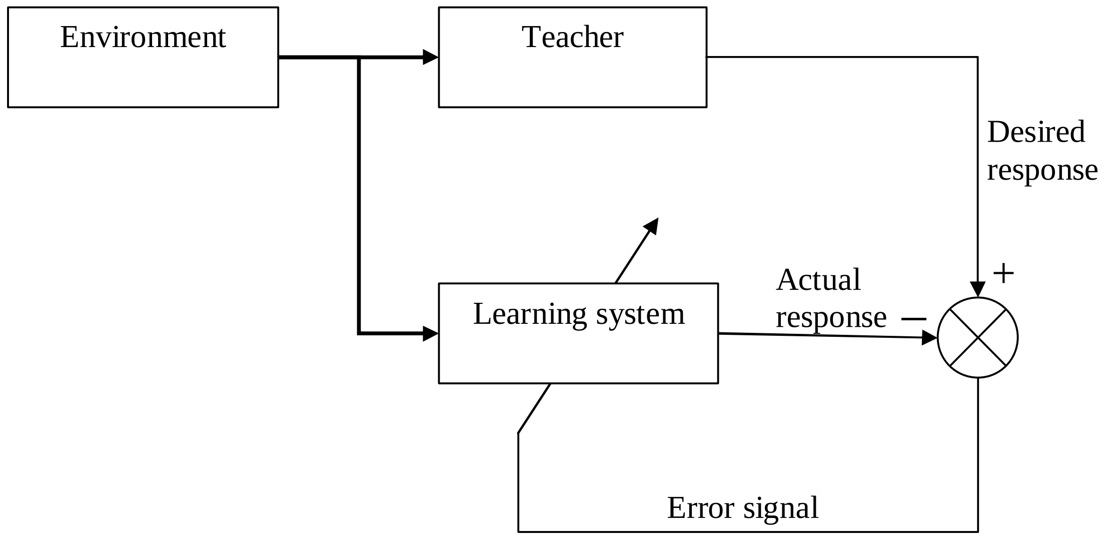


Figure 1.6: Learning with a teacher (supervised)

$$e_k(n) = d_k(n) - y_k(n) \tag{1.5.1}$$

where $y_k(n)$ denotes neuron output, $d_k(n)$ is a desired output of neuron k, $e_k(n)$ denotes error signal. The corrective mechanism is expected to minimalize this error. This is accomplished by minimalizing a cost function or index of performance:

$$E(n) = \frac{1}{2}e_k^2(n) \tag{1.5.2}$$

$E(n)$ is a instantaneous value of error energy. The corrections are continued until system is in stabilized state. One of the ways to minimalize the error energy is the Widrow-Hoff rule or delta rule. The adjustment to the j-th synaptic weight of neuron k equals:

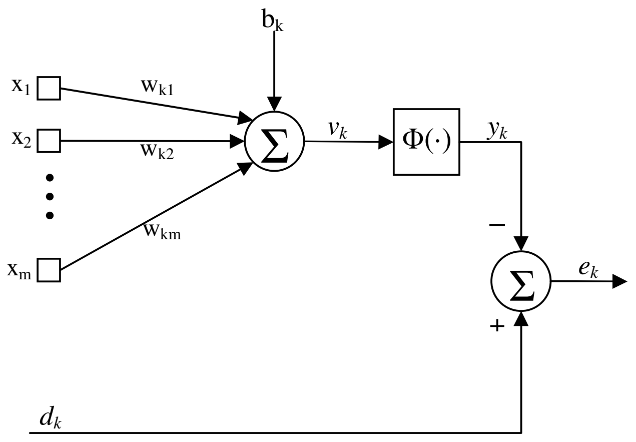


Figure 1.7: Error computation

$$\Delta w_{kj}(n) = \eta e_k(n) x_j(n) \tag{1.5.3}$$

where η is the positive constant called rate of learning, $w_{kj}(n)$ is the synaptic weight value of neuron k, when excited by element $x_j(n)$ of input vector $x(n)$ at time step n. The adjustment is proportional to the product of error signal and the input signal of the synapse. The updated value of synapse w_{kj} at time $n+1$ is computed:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \tag{1.5.4}$$

The network with new values of synapses is expected to achieve better performance. The very early approach was perceptron learning rule [Ros58]. The most popular techniques in error correction learning are based on backpropagation of gradient [Arb89], [Hay99], which is a generalization of the perceptron learning rule.

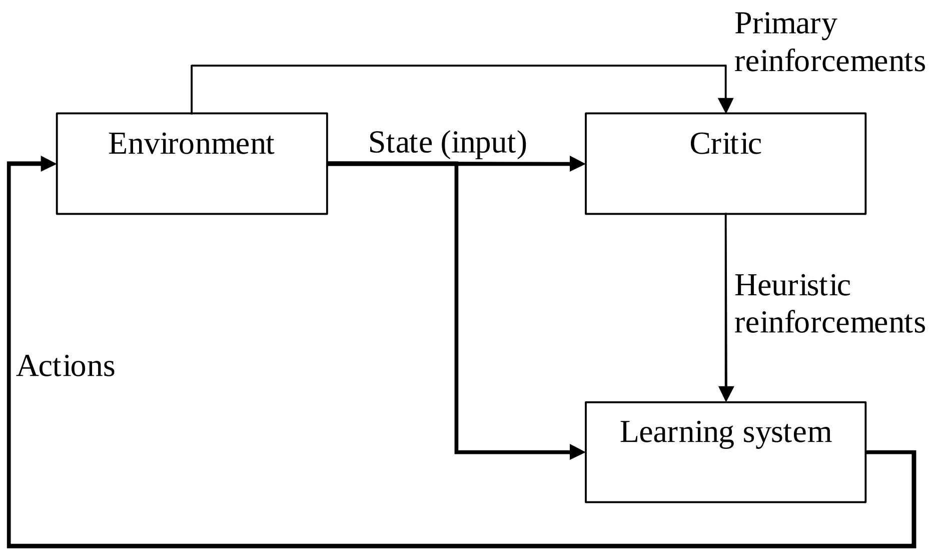


Figure 1.8: Reinforcement learning

1.5.2 Learning without a teacher

In **learning without a teacher**, there are no examples given to the system.

1.5.2.1 Reinforcement learning

In reinforcement learning the learning of input-output mapping is performed through continued interaction with the environment in order to minimalize a scalar index of performance.

Critic converts **primary reinforcement signal** received from an environment into a higher quality **reinforcement signal** called **heuristic reinforcement**. The system incorporates delayed reinforcement which means that system is able to observe temporal sequence of state vectors. The goal is to minimalize a **cost-to-go function** defined as an expectation of the cumulative cost of actions taken over a sequence of steps. Reinforcement learning is related to **dynamic programming** [Bel57], which provides the mathematical formalism for sequential decision making.

1.5.2.2 Hebbian learning

Hebb's postulate of learning [Heb49] says that if a neuron A repeatedly excites other neuron B, then the connection from A to B (axon) becomes more efficient. The postulate was made in neurobiological context.

The idea expanded by Changeeux and Danchin says as follows:

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of synapse are activated asynchronously, then that synapse is selectively weakened or eliminated

Hebbian synapse is characterized by four properties:

1. **time-dependent mechanism** - modification of the Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals

2. **local mechanism** - the modifications are local and are triggered by local conditions

3. **interactive mechanism** - Hebbian form of learning depend on interaction between presynaptic and postsynaptic signals two neurons.

4. **conjunctional or correlational mechanism** - temporal correlation between presynaptic and postsynaptic signals is viewed as being responsible for change.

General form of Hebbian adjustment of synapse w_{kj} of neuron k at time step n is expressed as follows:

$\Delta w_{kj}(n) = F(y_k(n)_j, x_j(n))$

where $F(\cdot, \cdot)$ is a function of presynaptic and postsynaptic signals, y_k is postsynaptic signal and x_j is presynaptic signal. Hebb's hypothesis in the simplest form (only development of synapse, no weakening) is realized by adjustment of following form:

$$\Delta w_{kj}(n) = \eta y_k(n)_j, x_j(n) \tag{1.5.5}$$

where η is the learning rate parameter. Covariance hypothesis [Sej77] expands the functionality by taking into account average values of respectively presynaptic and postsynaptic signals:

$$\Delta w_{kj} = \eta(x_j - \bar{x}_j)(y_j - \bar{y}_j) \tag{1.5.6}$$

where η is the learning rate parameter. The covariance hypothesis realizes Hebbian learning in the full form, described by Changeeux and Danchin, by making synapse stronger or weaker depending on the pre- and post synaptic activities. In particular the synaptic adjustment achieves negative value (with minimum of $-\eta(x_j - \bar{x})\bar{y}$), where synapse is weakened. When presynaptic activity x_j or postsynaptic activity y_k equal the average values respectively $\bar{x}$ and $\bar{y}$ the synapse remains unchanged.

There is a psychological evidence for the occurrence of Hebbian learning in the area of brain called hippocampus [BL73].

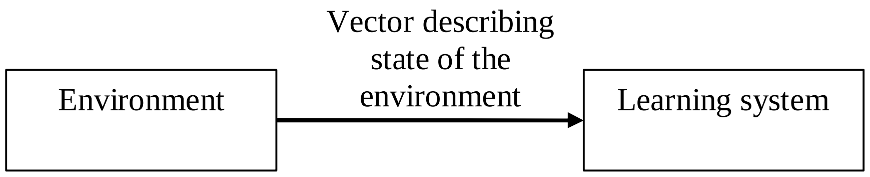


Figure 1.9: Unsupervised learning

1.5.3 Unsupervised learning

In **unsupervised** or **self-organized learning** the stress is placed on independent measure of the quality of network representation. Unsupervised learning is usually a way to form 'natural groupings' or clusters of patterns (in this interpretation referred to also as clustering).

The advantages of clustering:

- Separate processing of clustered groups can be much easier and less expensive than processing of the database as a whole. Collecting and labelling a large set of sample patterns can be very expensive. For example recorded speech is virtually free, but accurately labelling the speech can be very expensive and time consuming. By designing a basic classifier with a small set of labelled samples, and then tuning the classifier up by allowing it to run without supervision on a large, unlabelled set, much time and trouble can be saved;

- Training with large amounts of often less expensive, unlabelled data, and then using supervision to label the groupings found. This may be used for large "data mining" applications where the contents of a large database are not known before;

- Unsupervised methods can be used to find features which can be useful for categorization. There are unsupervised methods that represent a form of data-dependent ”smart pre-processing” or ”smart feature extraction.”;

- Clustering can give insight into the nature or structure of the data.

Clustering is described in details in the section 2.4 of this work - in connection with modular algorithms.

1.5.4 Competitive learning

In competitive learning the output neurons of ANN compete among themselves to become active. Only one neuron in the output layer can be active (fire) at the same time (opposite to Hebbian learning, where many output neurons can be active simultaneously). Basic elements of competitive learning rule are:

- a set of neurons which are the same except for the synaptic weights. Therefore they respond differently to a given set of input patterns
- a limit of the ”strength” of each neuron
- a mechanism which forces the competition between neurons, so only one of the group is active at a time [RZ85].

The individual neurons specialize on ensembles of similar patterns, so they became **feature detectors** of input patterns

A mechanism called **lateral inhibition** is a set of negative feedbacks connections linking excited neuron to other neurons. It means that when the neuron is excited it decreases the possibility that other neurons are excited. Lateral inhibition is also present in the neurobiological systems like eye retina, ear cochlea and pressure sensitive nerves of the skin [Arb89].

Each neuron in the structure is allocated a fixed amount of synaptic weight (all weights are positive):

$$\sum_j w_{kj} = 1 \tag{1.5.7}$$

to ensure that neuron learns by shifting the synaptic weights from inactive to active input nodes. To be a winning neuron, the neuron's **induced local field** v_k must be the largest among all neurons in the network. **Induced local field** is equal to:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k \tag{1.5.8}$$

The output of winning neuron is then set to 1, the outputs of other neurons are set to 0. The winning neuron is moved towards the input pattern x by competitive learning rule:

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron k wins the competition} \\ 0 & \text{if neuron k loses the competition} \end{cases} \tag{1.5.9}$$

where η is the learning parameter.

1.5.5 Memory-based learning

In memory-based learning, all or part of the experiences are stored in memory of correctly classified examples. When classification of test vector x_{test} is requested, the algorithm searches the memory for the similar examples of x_{test}("local neighborhood") and produces an output depending on the values of the memorized similar examples.

Memory-based algorithms have two properties:

- definition of local neighborhood of the test vector x_{test};
- learning rule applied to the local neighborhood of x_{test};

The memory based learning simplest example is the k-nearest neighbor classification [HMS01]. ANN which use memory learning are for example radial-basis function network.

1.5.6 Boltzmann learning

The **Boltzmann learning rule** [AK89] is a stochastic learning algorithm derived form statistical mechanics. A neural network designed on the basis of the Boltzmann learning rule is called a **Boltzmann machine**.

In a Boltzmann machine, the neurons constitute a structure and operate in a quasi-binary way (two possible states of neurons: 1 and -1). The machine is characterized by **energy function** EB:

$$EB = -\frac{1}{2}\sum_j \sum_k w_{kj} x_k x_j \tag{1.5.10}$$

where x_j,x_k are the states of neurons j and k, w_{kj} is the synaptic weight connecting neuron j to neuron k. None of the neurons in the machine has self-feedback. The machine operates by choosing a neuron at random and flipping the state of the neuron with probability:

$$P(x_k \rightarrow -x_k) = \frac{1}{1 + \exp(-\Delta E_B k/T)} \tag{1.5.11}$$

where E_{Bk} is the change in the energy of the machine resulting form such a flip and T is a **pseudotemperature** - parameter used to control the uncertainty in firing. The goal is to achieve thermal equilibrium - steady state of the machine.

1.5.7 Learning algorithms conclusion

In this section a number of learning algorithms for single structure have been presented; all of them are well suited for machine learning (and in particular ANN). In the next section the concept of many collaborating structures will be introduced.

1.6 ANN Conclusion

ANN present huge variety of architectures and learning algorithms. The area is developing rapidly. Consequently, they are used as powerful tools in many problems, especially problems characterized by:

- lack of physical or statistical understanding of a problem
- statistical variations in the data
- nonlinear mechanism underlying the problem

ANN have many advantages:

- uniformity of analysis and design - it's possible to share theories and learning algorithms in different applications of ANN;
- adaptive morphology - ANN can adapt to major changes in the operating environment, ignoring the minor changes;
- input-output mapping - model building based on training examples;
- universality - many areas of interest, i.e. modelling, pattern recognition, signal processing, time series analysis;
- nonlinearity - ANN are capable of solving non-linear problems;

- fault tolerance - resistance to faults in hardware implementation (due to distributed structure);
- easy VLSI implementability in most cases;
- distributed processing - ANN are capable of parallel data processing;
- evidential response - can produce decisions with confidence rates;
- neurobiological and statistical analogies - in terms of morphology and algorithms.

ANN have also limitations, mainly due to distributed and randomized way of working:

- the given solution is sometimes unpredictable;
- if ANN doesn't work as expected it's sometimes not trivial to find the way to fix that behavior;
- Some tasks are more suited to an algorithmic approach (like arithmetic operations), where ANN are sometimes less efficient than conventional computation methods and algorithms;
- For some learning algorithms and structures the examples must be selected carefully, otherwise useful time is wasted or even worse - the network might be functioning incorrectly;
- specialized ANN may not be universal - ANN once trained is fixed on some data and cannot process other task until trained again, once ANN was designed for some task its design may be found inappropriate for other tasks;

- Possibly large number of training examples and long training periods are needed for some tasks;
- Knowledge embodied in ANN is sometimes not easily accessible (not understandable) outside of the ANN (i.e. ANN decides that patient needs a surgery, but the reason is unknown as the reasoning is based on trained examples and the decision is hidden and distributed in numerous neurons weights);
- It's sometimes hard to incorporate prior knowledge into network.

We have introduced so many models in order to show the great possibilities of ANN structures in processing of different tasks. In T-DTS approach we need two general class of modules: clustering and processing. We can found ANN structures for both cases, what makes easier the cooperation between these and offer great variability of possible processing modules.

Next chapter will talk about modular algorithms - solutions that have distributed structure. Our approach is a modular structure, so these solutions especially in ANN area are very interesting to us.

Chapter 2

Modular Algorithms

Apart from explicit solution of problem by specialized "one-piece" algorithm, there exist a number of solutions, which have modular structure. In modular structure modules could have some defined and regularized structure or be more or less randomly connected, ending up at completely independent and individual modules. The modules can communicate with others: send them data or orders (to modify module itself or structure, cooperate, perform specific action, autodestruct, etc.). A modular structure when modules are composed of Neural Networks is called Multi Neural Network (MNN).

We will present here three modular paradigms that are of particular interest for us: "divide and conquer" paradigm, Committee Machines and Multi-agent approach. "Divide and conquer" paradigm is certainly a leading idea of the work presented in the thesis. Committee machines are in large part incorporation of this paradigm. Multi-agent approach is more distant cousin as the stress is put on the independence of modules, but the idea of intelligent modules (taking decisions) is close to T-DTS approach.

2.1 "Divide and conquer" algorithms

The approaches most close to presented in this work are known as "divide and conquer" algorithms. They are based on the principle "Divide et impera" (Julius Caesar). The main frame of the principle can be expressed as:

1. Break up problem into two (or more) smaller subproblems of similar structure.
2. Solve subproblems
3. Combine results to produce solution to original problem.

The ways in which the original problem is split differ as well as the algorithms of solving subproblems and combining the sub-solutions. The splitting of the problem can be done in recursive way. Very known algorithm using the paradigm is **Quicksort** [Hoa62], which splits recursively data in order to sort them in defined order. In the ANN area the most known algorithm of similar structure is Mixture of Experts. An issue could be model complexity reduction by splitting of a complex problem into a set of simpler problems: multi-modelling where a set of simple models is used to sculpt a complex behavior [GK96]. Another promising approach to reduce complexity takes advantage from hybridization [KV95]. Several ANN based approaches were suggested allowing complexity and computing time reduction. Among proposed approaches, one can note the Intelligent Hybrids Systems [KV95], Neural Network Ensemble concept [Han93], Models or experts mixture ([BS95], [SN95]), Dynamic Cell Structure architecture [LW98] and active learning approaches [FL90].

It was observed earlier that one reason for examining algorithmic paradigms was the fact that their running time could often be precisely determined. This is useful in allowing comparisons between the performances of two algorithms to be made.

For Divide-and-Conquer algorithms the running time is mainly affected by 3 criteria:

- The number of sub-instances into which a problem is split: α
- The ratio of initial problem size to sub-problem size: β
- The number of steps required to divide the initial instance and to combine sub-solutions, expressed as a function of the input size, n.

Suppose, P, is a divide-and-conquer algorithm that instantiates α sub-instances, each of size n/β. Let $Tp(n)$ denote the number of steps taken by P on instances of size n. Then:

$$\begin{gathered} Tp(n_0) = constant(recursive - base) \\ Tp(n) = \alpha \cdot Tp(n/\beta) + \gamma(n) \end{gathered} \tag{2.1.1}$$

In the case when α and β are both constant (as in all the examples we have given) there is a general method that can be used to solve such recurrence relations in order to obtain an asymptotic bound for the running time $Tp(n)$.

In general: $T(n) = \alpha T(n/\beta) + O(n^\gamma)$, (where γ is constant) has the solution:

$$T(n) = \begin{cases} O(n^\gamma), & \alpha < \beta^\gamma \\ O(n^\gamma \log n), & \alpha = \beta^\gamma \\ O(n^{\log^{-\beta}\alpha}), & \alpha > \beta^\gamma \end{cases} \tag{2.1.2}$$

2.2 Committee Machines

The **committee machines** are based on engineering principle divide and conquer. According to that rule, a complex computational task is solved by dividing it into a number of computationally simple tasks and then combining the solutions of these tasks. In supervised learning, the learning task is distributed among a number of experts which divides the space into a set of subspaces. The combination of experts is called committee machine. Committee machine fuses knowledge of experts to achieve an overall decision, which is supposedly superior to that achieved by any of the experts alone. Committee machines are universal approximators [Tre01].

The taxonomy of committee machines could be as follows:

1. Static structures
   - **a)** Ensemble averaging
   - **b)** Boosting
2. Dynamic structures
   - **a)** Mixture of experts
   - **b)** Hierarchical mixture of experts

Next several subsections will present the types of committee machines in detail.

2.2.1 Ensemble averaging

In ensemble averaging technique [Hay99], [Arb89], a number of differently trained experts (i.e. neural networks) share a common input and their outputs are combined to produce an overall output value y.

The advantage of such structure over a single expert is that the variance of the ensembled average function $F_I(x)$ is smaller than the variance of single expert $F(x)$. Simultaneously both functions have the same bias. These two facts lead to a training

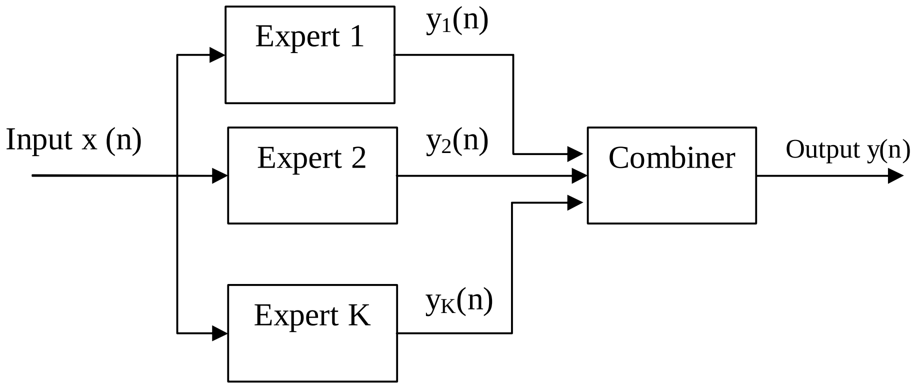


Figure 2.1: Ensemble averaging structure

strategy for reducing the overall error produced by a committee machine due to varying initial conditions [NIH97]: the experts are purposely overtrained, what results in reducing the bias at the cost of variance. The variance is subsequently reduced by averaging the experts, leaving the bias unchanged.

2.2.2 Boosting

In boosting [Sch99a] (in contrast with the ensemble averaging) the experts are trained on the data sets with entirely different distributions; it is a general method which can improve the performance of any learning algorithm. Boosting can be implemented in three different ways:

1. **Boosting by filtering** - the training examples are filtered by different versions of a weak learning algorithm with the examples either discarded or kept during training.

2. **Boosting by subsampling** - the second approach works with a sample of fixed size. The examples are resampled according to given probability distribution

during training.

3. **Boosting by reweighing** - the approach works with a fixed training sample and assumes that the weak learning algorithm can receive ”weighted” samples.

2.2.2.1 Boosting by subsampling

The algorithm that realizes **boosting by subsampling** is called **AdaBoost** algorithm [Sch99b]. It adaptively resamples the data set and uses a data set of fixed size. AdaBoost is running a given weak learner several times on slightly altered training data, and combining the hypotheses to one final hypothesis, in order to achieve higher accuracy than the weak learner's hypothesis would have. The algorithm is as follows:

- Assign every example an equal weight $1/N$
- For $t = 1, 2, , T$ Do
    - Obtain a hypothesis (classifier) $h(t)$ under $w(t)$
    - Calculate the weighted error $e(t)$ of $h(t)$
    - If e(t) ¿ 1/2, repeat for loop with different sample
    - Re-weight the correct examples with $b(t) = e(t)/(1 - e(t))$
    - Normalize $w(t + 1)$ to sum of 1
- Output a weighted sum of all the hypothesis, with each hypothesis weighted according to its accuracy on the training set.

2.2.3 Mixture of experts

Mixture of experts consists of K supervised models called expert networks and a gating network, which performs a function of mediator among expert networks. The output is a sum of experts' outputs (weighted by gating network). It is assumed that the different networks work best in different regions of the input space, in accordance with the probabilistic generative model [JJ95].

An exemplary Mixture of Experts structure is presented in figure 2.2. One can not the K experts and a gating network that filters the solutions of experts. Finally the weighted outputs are combined to produce overall structure output.

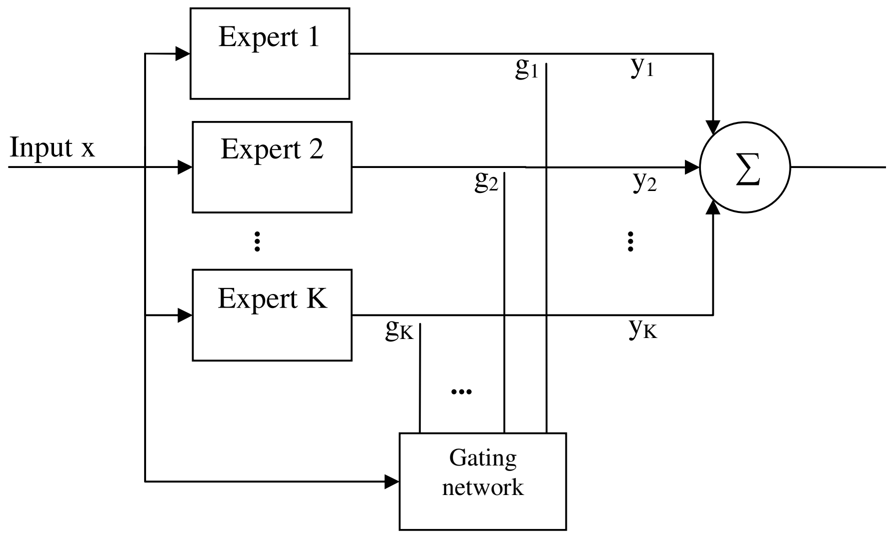


Figure 2.2: Mixture of Experts

The gating network consists of K neurons, with every neuron assigned to specific expert. The gating network structure is depicted in the figure 2.3.

The neurons in gating network are nonlinear with activation function defined by:

$$g_k = \frac{\exp(u_k)}{\sum_{j=1}^{K} \exp(u_j)} \text{where} k = 1, 2, , K \tag{2.2.1}$$

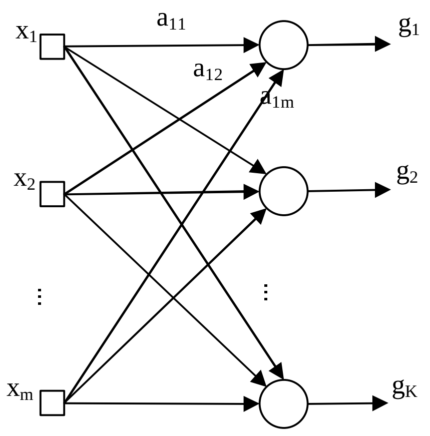


Figure 2.3: Structure of gating network

where u_k is the inner product of the input vector x and synaptic weight vector w_k for the neuron k:

$$u_k = w_k^T x, k = 1, 2, ..., K \tag{2.2.2}$$

The transfer function is a differentiable version of "winner-takes-all" operation of picking the maximum value. It is referred to as "softmax" transfer function [Bri90], described in section C.4 of appendixes. The use of "softmax" as the activation function for the gating network induces the following properties:

$$0 \leq g_k \leq 1, \text{for all } k \tag{2.2.3}$$

$$\sum_{k=1}^{K} g_k = 1 \tag{2.2.4}$$

The gating network maps the input vector into multinomial probabilities, so the different experts will be able to match the desired response [JJ95]. The output is a sum of experts' outputs and is equal to:

$$y = \sum_{i=1}^{K} g_i y_i \tag{2.2.5}$$

The mixture of experts is an **associative Gaussian mixture model**, which is a generalization of **traditional Gaussian mixture model** [TSM85], [MB88].

2.2.4 Hierarchical Mixture of experts

Hierarchical mixture of experts works similarly to ordinary mixture of experts, except that there exist multiple levels of gating networks. So the outputs of mixture of experts are gated in order to produce combined output of several mixture of expert structures. In figure 2.4 one can see two separate mixture of experts blocks (marked with dashed rectangles). The additional gating network is gating the outputs of these two blocks in order to produce the global output of the whole structure.

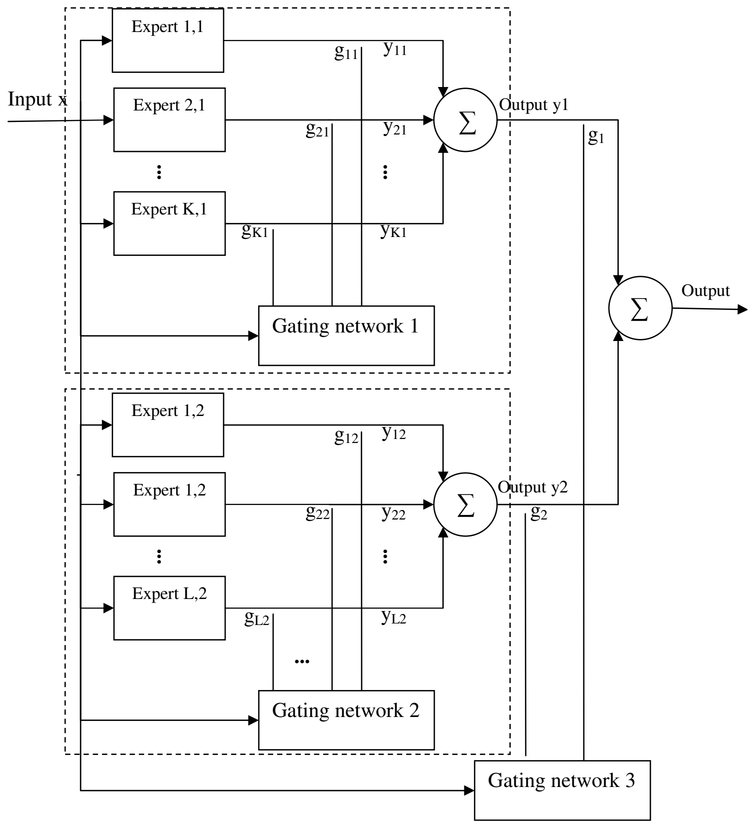


Figure 2.4: Example of hierarchical mixture of experts

2.3 Multi Agent Systems

There is a great variety of intelligent software agents and structures. The characteristics of Multi Agent Systems [Fer98] are that:

- each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint;
- there is no system global control;
- data are decentralized;
- computation is asynchronous

In Multi Agent Systems many intelligent agents interact with each other. The agents are considered to be autonomous entities, such as software programs or robots. Their interactions can be either cooperative or selfish. That is, the agents can share a common goal (e.g. an ant colony), or they can pursue their own interests (as in the free market economy).

Classification of agents by task:

- **Interface Agents** - "[C]omputer programs that employ artificial intelligence techniques in order to provide assistance to a user dealing with a particular application ...The metaphor is that of a personal assistant who is collaborating with the user in the same work environment." [Mae]
- **Information Agents** - "An information agent is an agent that has access to at least one, and potentially many information sources, and is able to collate and manipulate information obtained from these sources to answer queries posed by users and other information agents... " [WJ95a].

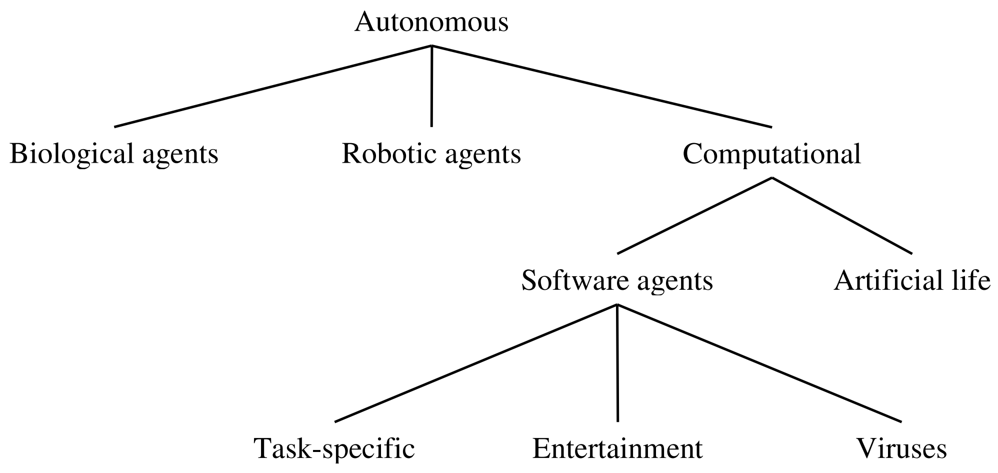


Figure 2.5: Classification of intelligent artificial agents considering origin [WJ95b]

- **Commerce Agents** - a commerce agent is an agent that provides commercial services (e.g., selling, buying and prices' advice) for a human user or for another agent. "
- **Entertainment Agents** - " ... artistically interesting, highly interactive, simulated worlds ... to give users the experience of living in (not merely watching) dramatically rich worlds that include moderately competent, emotional agents" [BLR92].

Agents can communicate, cooperate and negotiate with other agents. The basic idea behind Multi Agent systems is to build many agents with small areas of specialized knowledge and link them together to create structure which is much more powerful than the single agent itself. This is similar to T-DTS paradigm, where the Processing Unit models are specialized in small areas and conjunction of the areas covers whole problem space.

The theoretical basis for multiple agents is given in research field known as distributed artificial intelligence (DAI), which is a part of distributed problems solving. DAI is the study of distributed but centrally designed AI systems [AG92] and involves design of multiple-agents distributed system. The aim is to solve a problem or accomplish a task. The DAI decomposes the task into subtasks, each of which is processed by an agent. It is assumed also that there exist a single control structure which can influence the preferences and control the agents. The DAI infrastructure can be constructed with an architecture known as blackboard [AG92].

DAI systems contrast with multiagent systems. In multiagent systems, there is no single control structure (designer) which controls all agents. Each of these agents can work on different goals, sometimes in parallel and sometimes in contradictory. Both cooperation and competition is possible among agents [DSW97]. In multiagent systems, a complex problem is decomposed into subproblems, each of which is assigned and agent that works independently of others and is supported by a knowledge base. The agents make acquiring and interpretation of information by using deductive and inductive methods as well as computations. The resulting data is sent to coordinator who chooses one or more solutions.

2.4 Clustering

Clustering is a process of forming 'natural groupings' or clusters of patterns from database. The process is often referred to as unsupervised learning. The advantages of clustering:

- Separate processing of clustered groups can be much easier and less expensive than processing of the database as a whole. This is especially visible when the processing cost could be expressed as a magnitude of size of data.

- Training with large amounts of often less expensive, unlabelled data, and then using supervision to label the groupings found. This may be used for large "data mining" applications where the contents of a large database are not known before. Characteristics of patterns can change slowly with time. For example in automated food classification as the seasons change. If these changes can be tracked by a classifier running in an unsupervised mode, improved performance can be achieved.

- Unsupervised methods can be used to find features which can be useful for categorization. There are unsupervised methods that represent a form of data-dependent "smart pre-processing" or "smart feature extraction.

- Clustering can give insight into the nature or structure of the data. The discovery of similarities among patterns or of major departures from expected characteristics may suggest an approach to design the classifier.

2.5 Modular algorithms conclusion

Modular algorithms present advantages over single structures. By using a structure of simple networks one can obtain good results without a need for creating large and complicated networks and obtaining a well-defined structure, that is potentially much easier to control, modify and manage.

Next section will introduce complexity estimation - methods to estimate the complication of classification task. Incorporation of the techniques can provide knowledge about efficient methods for designing the problem solution, in particular modification of T-DTS structure. It can be also as a knowledge source for a system that adapts itself to problem.

Chapter 3

Complexity estimation

This chapter concerns complexity estimation methods. Complexity estimation is a supportive tool for classification. Complexity estimation is used as a knowledge harvester that provides clues about organization of processing structure. The goal of complexity estimation is to verify and measure the difficulty of classification task, yet before proper processing, in order to modify the processing parameters. The goal is to reduce the user intervention to minimum, while keeping processing efficient and accommodated to the processing task. This technique is one of possible tools that could lead finally to automate processing of computation task.

Structure of this chapter is as follows: section 3.1 introduces complexity estimation, shows its possible applications and identifies the sources of classification errors. Section 3.2 presents the basics of classification as a special case of prediction. Section 3.4 proposes taxonomy of complexity estimation methods. Section 3.5 exhibits most popular complexity estimation methods, following the taxonomy proposed in section 3.4. Conclusion in section 3.6 ends the chapter.

3.1 Complexity estimation - definition

Complexity estimation methods are intended for estimating difficulty of data in classification problem. Classification complexity estimation is used to understand the behavior of classifiers and for feature selection. It can be also used to choose or adjust classifier depending on the measured characteristics of the problem.

Bayes error [Irv65] is a theoretical probability of classification error, resulting from the ambiguity of classification task. It is considered as a target classification rate - no classification algorithm is expected to achieve better results on the unseen data. It is also the optimal complexity estimator. It is explained in details in section 3.3. There is a great impact of Bayes error on the most of classification complexity techniques. Bayes error is however difficult to calculate directly for three main reasons:

- conditional density estimation $p(c_k|x)$ is an ill-posed problem [Dev87],
- class probabilities $p(x)$ are needed,
- difficulty of numerical integration increases with dimensionality.

Many approaches aims at estimating Bayes error in indirect way, i.e. propose a measure which is a lower or higher bound of Bayes error but easier to calculate than direct estimation. Correlation to Bayes error is a desirable property for other measures.

To understand why classifiers are not successful, it is important to identify sources of classification problem errors. Relevant effects of geometrical complexity are described by Basu [Ho01]:

1. **Class ambiguity** - some problems are known to have nonzero Bayes error [HB97], it means that samples from two different classes may have identical

feature values. (Bayes error sets a lower bound on achievable error rate.) According to Ho [Ho00], this can happen regardless of class boundary shape and feature space dimension. Some problems may be intrinsically ambiguous; other problems may be ambiguous only due to poor features selection. The ambiguity is intrinsic if the given feature set is complete for reconstruction of the patterns. Otherwise redefining the features can remove the ambiguity.

2. **Curvature of boundary / imperfectly modelled boundary complexity** - some problems have complicated optimal decision boundary. In other words, some are much more difficult than others. Classifying algorithm should have enough capacity to cope with the problem, otherwise boundary is imperfectly modelled and classification error will occur (avoidable by reference to Bayes error). This effect is independent of class ambiguity, sampling density and feature space dimensionality

3. **Non-representative sample size and feature space dimensionality** - sometimes training set does not represent whole classification problem well. Classification algorithm considers a problem simple when in fact it is complex but insufficiently represented. This easily happens in high dimensional space where the class boundary can vary with a larger degree of freedom. The representativeness of training set is a subject of study in many theoretical and practical considerations, like Vapnik's statistical learning theory [Vap98], Kleinberg's arguments on M-representativeness [Kle96], Berlind's hierarchy of indicernibility [Ber94], Raundy's and Jain's work [RJ91] and many others [FK71], [KD82], [Tou74].

Before presenting specific classification complexity methods, classification and its complexity related properties will be discussed.

3.2 Classification as an aspect of Predictive Modelling

Predictive modelling aim is to predict unknown value of variable when provided with known values of other variables. Predictive modelling can be thought of as a learning of mapping from an **input set of features** x to **estimated scalar output** y. To learn, the mapping **model** is provided with **training data** D_{train} (consisting of pairs of measurements: vector x(i) with corresponding **true output value target** y). The goal of predictive modelling is to **estimate** (using training data Dtrain) **a function representing the model** $\hat{y} = M(x;\theta)$ that generates estimated values of output $\hat{y}$, which are expected to be close to true value of output y. That function is given an input vector $x(i)$ and a set of estimated parameters θ for a model M. There are two kinds of task in predictive modelling:

- **Classification** - values y of output set Y are categorical
- **Regression** - values y of output set Y are real

To build a predictive model one have to choose three elements:

1. **a model** M
2. **a score function** S for determining how well the model fits the original problem (how far are the real values from the results estimated by model f)

3. **optimalization strategy**, also known in literature as training algorithm, the aim of it is to minimize the score function S as function of θ

The choice of model is difficult, as huge number of solutions exists in literature. The advantages of simple models are that they are easier to interpret and usually more stable. They can be just too simple for complicated problems and don't achieve satisfactory results. Relatively new idea is a combination of multiple simple models in some way (i.e. Multiple Neural Networks [Arb89]), which can possibly lead to satisfactory results without overcomplicating.

Score function (also known as error function) is a function of difference from observed real output values y and predicted output values y:

$$S(\theta) = \sum_{D_{train}} d(y(i), \hat{y}(i)) = \sum_{D_{train}} d(y(i), f(x(i); \theta)) \tag{3.2.1}$$

where sum is taken over the tuples $(x(i), y(i))$ in the training data set and d defines a distance between real output y and predicted output $\hat{y}$ for the same input vector $x(i)$. Distance function $d(\cdot)$ is usually defined as squared distance (Euclidean) for real-valued outputs y or an indicator function for categorical values of y. Function $d(\cdot)$ has scalar values.

Most known distance functions for real-valued outputs, between feature vectors i and j where dim is the total number of dimensions (features), are:

- Euclidean distance:

$$d_E(i, j) = \left(\sum_{k=1}^{dim} (x_k(i) - x_k(j))^2 \right)^{\frac{1}{2}} \tag{3.2.2}$$

- Weighted Euclidean distance:

$$d_{WE}(i,j) = \left(\sum_{k=1}^{dim} u_k (x_k(i) - x_k(j))^2 \right)^{\frac{1}{2}} \tag{3.2.3}$$

where vector u_k defines the importance weights for features.

- Manhattan (Hamming) distance:

$$d_{MH}(i,j) = \sum_{k=1}^{dim} |x_k(i) - x_k(j)| \tag{3.2.4}$$

- Minkovsky distance:w

$$d_{MI}(i,j) = \left(\sum_{k=1}^{dim} (x_k(i) - x_k(j))^{\lambda} \right)^{\frac{1}{\lambda}} \tag{3.2.5}$$

Minkovsky distance is a generalization of Euclidean and Manhattan distance. Minkovsky distance when $\lambda = 2$ equals Euclidean distance and Minkovsky distance with $\lambda = 1$ equals Manhattan distance.

- Mahalanobis distance:

$$d_{MH}(i,j) = ((x(i) - x(j))^T \Sigma^{-1} (x(i) - x(j))^{\frac{1}{2}} \tag{3.2.6}$$

$x(i)$ and $x(j)$ are the feature vectors, T represents the transpose, Σ is the $p \times p$ covariance matrix. The purpose of using Σ^{-1} is to standardize the data relative to covariance matrix. The formula can be used also as a data complexity criterion as described in section 3.5.1.5.

- Tchebyshev distance:

$$d_T(i,j) = \max_{i=1,2,\ldots,p} |x_k(i) - x_k(j)| \tag{3.2.7}$$

Tchebyshev distance is a generalization of Minkovsky distance when λ approaches infinity.

- Canberra distance:

$$d_C(i,j) = \sum_{k=1}^{p} \frac{|x_k(i) - x_k(j)|}{|x_k(i) + x_k(j)|} \tag{3.2.8}$$

Optimalization strategy (commonly known in Neural Network literature as training algorithm) depends amongst others on type of predictive modelling task (classification or regression), specificity of task, chosen model M and score function $S(\cdot)$ used.

In classification we wish to learn a mapping (function) from feature vectors x to categorical values from space C (class labels). Class labels take values in the set $c_1, , c_m$. The most popular approach to deal with classification problems is to use discriminative classification, which is described in next section.

3.2.1 Discriminative classification

In a **discriminative classification** approach, a classification model $f(x;\theta)$ (prediction model for classification task is known as **classifier**) takes input vector x and assigns to it a value from the set of classes $c_1, , c_m$.

Decision region for class c_i (where $i \in [1, m]$) is union of all regions in the input feature space where output takes value of c_i. In other words, decision region for c_i is a set of all input values for whose output is c_i. Complement of this decision region is the decision region for all others classes.

Decision boundaries or **decision hyperplanes** separate the decision regions. Mathematical form of decision boundaries can be straight lines or planes (linear boundaries), curves boundaries such as polynomials and other (**Boundary Methods** are described in detail in section 3.5.2.3 on page 71).

In most real classification problems, the classes are not perfectly separable in feature space. The situation when relatively very close input vectors x are from different classes is referred to as **classes overlapping**. It leads to another way of seeing classification problem - we can seek a function $f(x;\theta)$ which maximalizes some measure of separation between classes. Such functions are called **discriminant functions**. The earliest formal approach to classification, **Fisher's linear discriminant** [Fis00], was based on this idea. It sought the optimal linear combination of the variables in input space. The goal was to maximally discriminate two classes.

3.3 Probabilistic models for classification

Let $p(c_k)$ be the probability that object comes from class c_k. Then, if classes are exclusive and exhaustive we have:

$$\sum_k p(c_k) = 1 \tag{3.3.1}$$

Often $p(c_k)$ are referred to as **prior probabilities** since they represent the class probabilities before observing the vector x.

Objects from class c_k are assumed to have measurement vectors x distributed according to **probability distribution** (**density function**) $p(x|c_k, \theta_k)$, where θ_k are unknown distribution parameters. For example, the estimated distribution may be multivariate Gaussian (normal), and the parameters θ_k may represent mean and variance of that distribution.

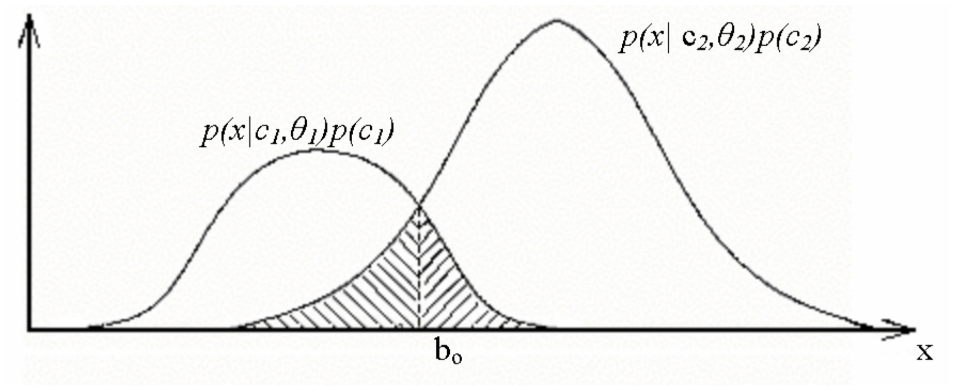


Figure 3.1: Two Gaussian distributions and Bayes error

Once the distributions have been estimated, we can yield the posterior probabilities:

$$p(c_k|x) = \frac{p(x|c_k,\theta)p(c_k)}{\sum_{i=1}^{m} p(x|c_k.\theta)p(c_k)}, \qquad \text{for } 1 \leq k \leq m \tag{3.3.2}$$

If one knew the true posterior probabilities (instead of estimating them), then it would be possible to make optimal classifications.

Figure 3.1 presents two Gaussian distributions (cases from two classes: c_1 and c_2) in one-dimensional space:

The values of x axis represent values of feature X, and the values on the y axis represent **posterior probabilities** of classes (in other words distributions, scaled using **prior probabilities**). There is an area, where the distributions overlap each other. In the overlapping area, classification errors cannot be avoided, as cases from both classes have the same feature values (the situation is referred to as **class ambiguity**). The inevitable classification error occurs only in the dashed area. To classify in the optimal way, one should take most probable class along the X space. Overlap means that there is a non-zero probability that the data comes from other class than the locally optimal. **Bayes error** of classifier $f(x;\theta)$ is the minimal probability of misclassification in whole input space X:

$$\varepsilon = \int (1 - \max_k p(c_k|x))p(x)\,dx \tag{3.3.3}$$

That value corresponds to dashed area. This is the minimum error rate. It is a lower-bound on the best possible classifier for the problem. No classifier is expected to achieve lower error on unseen data due to problem ambiguity.

Bayes optimal classifier is a classifier which error rate is equal to Bayes error.

Bayes optimal classifier is done as:

$$x \to c_k : p(c_k|x) = \max_k p(c_k|x) \tag{3.3.4}$$

Usually in real problems, we don't know however conditional probabilities $p(c_k|x)$. In the example above, Bayes optimal classifier is a classifier which separates the classes using a threshold value b_o marked in the figure 3.1, assigns cases with feature value less than b_o class label c_1 and with values greater or equal b_o to class label c_2.

Methods to compute Bayes error:

- **Analytical** - difficult to calculate, depends on distribution parameters,
- **Experimental** - estimate class densities basing on non-parametric methods.

Next section describes the various ways to build classifiers depending on the theory presented above.

3.3.1 Approaches to build classifiers

In order to build classifiers many techniques can be used. One can identify three main approaches:

1. **The discriminative approach** - find direct mapping from inputs x to one of m class label $c_1, ...c_m$. No attempt to model class conditional or posterior class probabilities. Examples: perceptron, Multi Layered Perceptron, Support Vector Machines [Arb89], [Hay99].

2. **The regression approach** - The posterior class probabilities $p(c_k|x)$ are modelled explicitly, and for prediction, the maximum of these probabilities weighted by cost function is chosen. Examples: logistic regression [CSS00].

3. **The class-conditional approach** - The class-conditionals $p(x|c_k, \theta_k)$ (where θ_k represent distribution parameters) are modelled explicitly along with estimates of $p(c_k)$ and both are used to compute $p(c_k|x)$. Also known as generative approach and classifiers as "Bayesian" classifiers.

3.3.2 Classification methods

The classification methods may cover the whole input space with one model or use submodels, where each submodel covers small area of space, and at last the union of the submodels covers the whole input space. Depending on that feature one can categorize classification methods:

1. **Global approximation** - there is only one model which covers all data feature space:

a) **Fisher discriminant function** - the method is based on the search of the linear mapping from the input variable to a new variable which separates best the classes;

b) **Perceptron** - perceptron is the simplest type of neural network and it is typically used for classification, it simply gains weighted sum of inputs and may produce only two values as output (1 or 0 usually). It can effectively solve any two-class classification problem where the classes are **linearly separable** (i.e. there exists at least one plane that is able to separate the classes; when projected in space).

c) **Logistic Discriminant Analysis (LDA)** - a statistical technique capable of discriminating individuals within a diseased group against normal representatives. It also enables classification of various diseases within a group of patients.

d) **Linear discriminants** - the method is based on the search of the linear combination of the variables which separates best the classes,

e) **Nearest neighbor methods** - to classify a new object one examines k nearest neighbors of object and assigns the object to the class that has the majority of points amongst these k neighbors,

f) **Feed-forward Neural Networks** - perceptron, Multi Layered Perceptron,

g) **Support Vector Machines** - they are long cousin of perceptron. Perceptron searches linear hyperplane that perfectly separates the data from different classes, SVM work in similar way, extending the feature space to include transformations of the raw variables and searching linear decision surface in the enhanced space. This conforms to nonlinear decision surface in the raw measurement space.

h) **The Naive Bayes model** - a simple probabilistic classification method that assumes (naively) that all model feature are independent.

i) **Bayesian model averaging** - classification method that averages over many different competing models, thus diminishing the uncertainty inherent in the model selection process.

j) **Projection pursuit methods** - they consist of linear combinations of non-linear transformations of the raw variables. They are data-driven.

k) **Probabilistic Neural Networks** - model is built depending on the learning examples presented to network.

2. **Local approximation** - model consists of many smaller models (sub-models). Each sub-model covers a part of data feature space:

   a) **Mixture models** - approximate each class-conditional distribution by a mixture of simpler distributions (for example multivariate Gaussian distributions),

   b) **Radial Basis Functions (RBF)** - ANN approach similar to mixture models, they combine many models of local influence to produce a mixture model,

   c) **Mixture of experts** - ANN approach parallel to mixture models.

   d) **Tree models** - basic principle is to partition in a recursive manner the space spanned by the input variables to maximalize some score of data separability.

Artificial Neural Networks are frequently used with success to solve classification problems. Most popular ANN used in this purpose are: perceptron, multilayered feed-forward neural networks, (especially Multi-Layered Perceptron), mixture of experts, radial basis function, projection pursuits, Support Vector Machines. Most of these structures are described in Annex B. There are methods to evaluate the efficiency of classification and compare them to others. Two most known methods are described in the next section.

3.3.3 Evaluating and comparing classifiers

Leaving-one-out method - from known data of n vectors, we create n training sets by leaving one vector each time for generalization (testing set). Thus we train the model n times, and verify obtained classification rates on left vector, which is unseen by the model (not included in learning). By using this method one can use whole available data for learning and classify whole set, keeping generalization free of data known from learning.

Bootstrap methods - model the relation between unknown true distributions and the sample by analyzing the relation between the sample and subsample of the same size drawn from the sample.

3.3.4 Feature reduction

Not all variables measured are always necessary to build a model. In fact overwhelming number of variables may lead to worse models. It is not obvious which variables are non relevant and can be deleted. General strategies to reduce the number of variables without much loss of important information are:

- **Variable selection** - the idea is to select a sub-database of variables of the original set of features. There is a combinatorially large search space of variable sub-databases which may be considered.

- **Variable Transformations** - transform the original measurements by some linear or non-linear functions in order to diminish the number of variables and thus simplify further processing. Examples:

  - **Principal Component Analysis (PCA)** [TB97] - modification of directions in space in order to achieve maximum variance;
  - **Projection pursuit** [Hub85] - search for interesting linear projections;
  - **Factor analysis** [Gor83] - linear transformation of data in search for interesting properties;
  - **Independent component analysis** [Com91] - extracting maximally independent components form data.

3.4 Taxonomy of complexity estimation methods

Significant part of complexity estimation methods are related to Bayes error estimation. Due to the difficulties with direct Bayes error computation, some approaches try to estimate Bayes error in indirect way, i.e. propose a measure which is a lower or higher bound of it but easier to calculate than direct estimation. Another method is to use non-parametric methods to estimate Bayes error.

Correlation to Bayes error is a desirable property for other measures. Some of them take advantage from space partitioning. Thus, one can divide the measures into four classes:

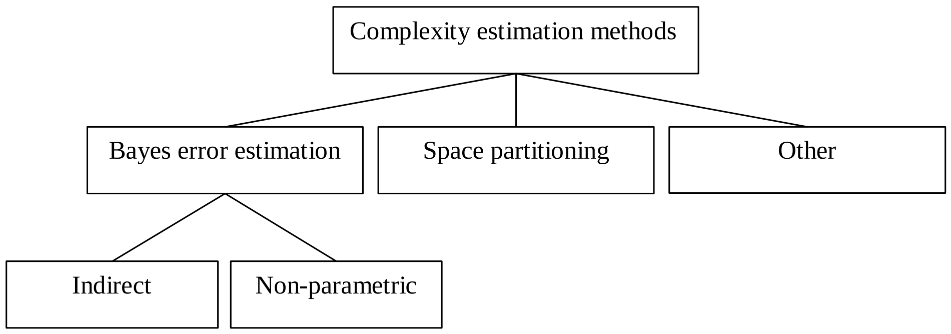


Figure 3.2: Proposed taxonomy of complexity estimation methods

1. **Indirect Bayes error estimation** (probabilistic distance measures) - parametric estimates of Bayes error;
2. **Non-parametric Bayes error estimation** - methods which are proven to estimate Bayes error but in non-parametric way;
3. **Methods based on space partitioning** - they are connected to some space partitioning algorithms;
4. **Other methods** - remaining ideas that are very different.

3.5 Presentation of selected methods

Complexity estimation is not a very popular subject. Regarding that fact, this section present in detail classification complexity estimation methods, categorized as stated in previous section. Section 3.5.1 presents indirect Bayes error methods, section 3.5.2 exhibits non-parametric Bayes error estimation methods, section 3.5.3 familiarizes with measures related to space partitioning and section 3.5.4 stages other methods. Section 3.5.5 talks briefly about a possibility of creating ensembles of estimators, which correctly used can inhibit advantages over solo methods.

3.5.1 Indirect Bayes error estimation

To avoid the difficulties related to direct estimation of the Bayes error, a popular approach is to estimate a measure directly related to the Bayes error, but easier to calculate. Usually one assumes that the data distribution is normal (Gaussian).

Statistical methods grounded in the estimation of probability distributions are most frequently used. The drawback of these is that they assume data normality. A number of limitations have been documented in literature [Vap98]:

- construction of model could be time consuming;
- model verification could be difficult;
- as data dimensionality increases, a much larger number of samples is needed to estimate accurately class conditional probabilities;
- if sample does not sufficiently represent the problem, the probability distribution function cannot be reliably approximated and Bayes error cannot be accurately estimated;
- with a large number of classes present, estimating a priori probabilities is quite difficult. This can be only partially overcome by assuming equal class probabilities [Fuk90], [HB02].
- we normally do not know the density form (distribution function);
- most distributions in practice are multimodal, while models are unimodal;

- approximating a multimodal distributions as a product of univariate distributions do not work well in practice.

Following subsections will present complexity measures based on indirect Bayes error estimation.

3.5.1.1 Normalized mean distance

Normalized mean distance is a very simple complexity measure for Gaussian unimodal distribution. It raises when the distributions are distant (measured by distance between means) and not overlapping.

$$d_{norm} = \frac{|\mu_1 - \mu_2|}{\sigma_1 + \sigma_2} \tag{3.5.1}$$

The main drawback of that estimator is that it is inadequate (as a measure of separability) when both classes have the same mean values.

3.5.1.2 Chernoff bound

The Bayes error for the two class case can be expressed as:

$$\varepsilon = \int \min_i [P(c_k)p(x|c_k)]\ dx \tag{3.5.2}$$

Through modifications, we can obtain a **Chernoff bound** ε_u, which is an upper bound on ε for the two class case:

$$\varepsilon_u = P(c_1)^s P(c_2)^{1-s} \int p(x|c_1)^s p(x|c_2)^{1-s}\, dx \qquad \text{,for } 0 \leq s \leq 1 \tag{3.5.3}$$

The tightness of bound varies with s.

3.5.1.3 Bhattacharyya bound

The **Bhattacharyya bound** is a special case of Chernoff bound for $s = \frac{1}{2}$. Empirical evidence indicates that optimal value for Chernoff bound is close to $\frac{1}{2}$ when the majority of separation comes from the difference in class means. Under a Gaussian assumption, the expression of the Bhattacharyya bound is:

$$\varepsilon_b = \sqrt{P(c_1)P(c_2)}e^{-\mu(\frac{1}{2})} \tag{3.5.4}$$

where:

$$\mu\left(\frac{1}{2}\right) = \frac{1}{8}\left(\mu_2 - \mu_1\right)^T \left(\frac{\Sigma_1 + \Sigma_2}{2}\right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2}\ln\frac{\left|\frac{\Sigma_1+\Sigma_2}{2}\right|}{\sqrt{|\Sigma_1||\Sigma_2|}} \tag{3.5.5}$$

and μ_i and Σ_i are respectively the means and covariances of classes $i \in \{1, 2\}$.

3.5.1.4 Divergence

Measure of **divergence** [Lin91] (separability) is related to verisimilitude ratio. Verisimilitude ratio L_{12} between two classes c_1 and c_2 is defined as:

$$L_{12}(X) = \frac{P(X|c_1)}{P(X|c_2)} \tag{3.5.6}$$

Divergence is defined as function of logarithm of verisimilitude ratio:

$$D_{12} = E\left[L'_{12}(X)|c_1\right] + E\left[L'_{21}(X)|c_2\right] \tag{3.5.7}$$

and after transformations:

$$D_{12} = \int_x (P(X|c_1) - P(X|c_2)) \ln\frac{P(X|c_1)}{P(X|c_@)}\, dx \tag{3.5.8}$$

The measure of divergence has the following properties:

1. $D_{12} \geq 0$;

2. Symmetry: $D_{12} = D_{21}$

3. If probabilities distributions $P(X|c_1)$ and $P(X|c_2)$ are equal then divergence D_{12} equals 0. In particular $D_{11} = 0$.

4. If variables $X^1, X^2, , X^d$ are statistically independent, i.e.

$$P(X|c_1) = \prod_{k=1}^{d} P(X^k|c_1), \tag{3.5.9}$$

then

$$D_{12}(X^1, X^2, ...X^d, X^d) = \sum_{k=1}^{d} D_{12}(X^k) \tag{3.5.10}$$

5. As consequence of properties 1 and 4:

$$D_{12}(X^1, X^2, ...X^d, X^{d+1}) > D12(X^1, X^2, ...X^d) \tag{3.5.11}$$

That property is important because it means that divergence will increase with addition of new variables.

Calculation of the divergence is significantly simplified when distributions of variables are normal. In that case divergence equals:

$$D_{12} = \frac{1}{2} tr[(\Sigma_1 - \Sigma_2)(\Sigma_1^{-1} - \Sigma_2^{-1})] + \frac{1}{2} tr[(\Sigma_1^{-1} - \Sigma_2^{-1})(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T] \tag{3.5.12}$$

where tr signifies trace of a matrix, μ_1 and mu_2 class means, Σ_1 and Σ_2 class covariance matrices.

Divergence measures the degree of separability between two classes. Therefore in order to evaluate multi class case one should count an average of all two-element combination of classes. Computational cost of divergence is significant.

Transformed divergence is defined as:

$$D_{12}^T = 2[1 - e^{-\frac{D_{12}}{8}}] \tag{3.5.13}$$

Transformed divergence takes values from range $[0, 2]$ and increases with class separability.

3.5.1.5 Mahalanobis distance

Mahalanobis distance [TKM87] is defined as follows:

$$M_D = (\mu_2 - \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1) \tag{3.5.14}$$

M_D is the Mahalanobis distance between two classes. The classes' means are μ_1 and mu_2 and Σ is the covariance matrix. Mahalanobis distance is used in statistics to measure the similarity of two data distributions. It is sensitive to distribution of points in both samples. The Mahalanobis distance is measured in units of standard deviation, so it is possible to assign statistical probabilities (that the data comes from the same class) to the specific measure values. Mahalanobis distance greater than 3 is considered as a signal that data are not homogenous (does not come from the same distribution).

Similar practice is applied in complexity estimation: high values of Mahalanobis measure indicate that classes in discrimination problem are well separated.

3.5.1.6 Jeffries-Matusita distance

Jeffries-Matusita [Mat67] distance between classes c_1 and c_2 is defined as:

$$JM_D = \int_x \left(\sqrt{p(X|c_2)} - \sqrt{p(X|c_1)} \right)^2 dx \tag{3.5.15}$$

If c_1 and c_2 distributions are normal then Jeffries-Matusita distance reduces to:

$$JM_D = 2(1 - e^{-\alpha}) \tag{3.5.16}$$

where

$$\alpha = \frac{1}{8}(\mu_2 - \mu_1)^T \left(\frac{\Sigma_2 + \Sigma_1}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} log_e \left(\frac{\det \Sigma}{\det \Sigma_1 - \det \Sigma_2} \right) \tag{3.5.17}$$

Matusita distance is bounded within range $[0, 2]$ where high values signify high separation between classes c_1 and c_2.

3.5.1.7 Entropy measures

Examining the dependence between the data vector x and the class c gives a measure of how much information these two distributions contain about each other. The proof is as follows - for the extreme case when x and c are independent, $p(c|x)$ and $p(c)$ are equivalent. It means that data provides no information about which class it should belong to. Therefore the data is worthless for classification. That logic leads to information theoretic measures for class separability. The procedure is to observe x and compute the *a posteriori* probabilities $p(c_i|x)$ to determine the amount of class information contained in the observation. One of the most popular measures is Shannon's measure:

$$J = -\int \sum_{i=1}^{L} p(c_i|x) \log_2[p(c_i|x)] p(x)\, dx \tag{3.5.18}$$

L is the number of classes. Unlike the Chernoff and Bhattacharyya distances, this equation is not limited to the two classes but neither is directly related to Bayes error. The probability densities are needed, that implies that entropy based measures suffer from difficulties of estimating Bayes error directly. It is known however that the variances associated with entropy measures of class separability are normally lower than those associated with direct Bayes error estimation.

3.5.2 Non-Parametric Bayes error estimation and bounds

Non-parametric Bayes error estimation methods make no assumptions about the specific distributions involved. They use some intuitive methods and then prove the relation to Bayes error. Non-parametric techniques do not suffer from problems with parametric techniques.

3.5.2.1 Error of the classifier itself

This is the most intuitive measure. However it varies much depending on the type of classifier used and, as such, it is not very reliable unless one uses many classification methods and averages the results. The last solution is certainly not computationally efficient.

3.5.2.2 k-Nearest Neighbors, (k-NN)

K - Nearest Neighbors [CH67] technique relays on the concept of setting a local region $\Gamma(x)$ around each sample x and examining the ratio of the number of samples enclosed k to the total number of samples N, normalized with respect to region volume v:

$$\hat{p}(x) = \frac{k(x)}{vN} \tag{3.5.19}$$

K-NN technique fixes the number of samples enclosed by the local region (k becomes constant). The density estimation equation for k-NN becomes:

$$\hat{p}(x) = \frac{k-1}{v(x)N} \tag{3.5.20}$$

where $p(x)$ represent probability of specific class appearance and $v(x)$ represent local region volume.

K-NN is used to estimate Bayes error by either providing an asymptotic bound or through direct estimation. Asymptotic bounds are derived by an application of the voting k-NN procedure. For the two class case, where e_{kNN} represents the k-NN estimate of ε, these estimates form a bound on ε by:

$$\frac{1}{2}\varepsilon \leq \varepsilon_{2NN} \leq \varepsilon_{4NN} \leq ... \leq \varepsilon \leq ... \leq \varepsilon_{3NN} \leq \varepsilon_{1NN} \leq 2\varepsilon \tag{3.5.21}$$

As $N \to \infty$ asymptotically the estimations are closer to ε. For the L class case, as $N \to \infty$, we obtain the following bound:

$$\varepsilon_{NN} \leq 2\varepsilon - \frac{L}{L-1}\varepsilon^2 \tag{3.5.22}$$

The k-NN estimate of Bayes error can exhibit significant biases and variances when N is finite. K-NN can directly estimate ε by first estimating the expected value of the risk that the biases and variances will influence the estimation. The formula for this estimate is:

$$\hat{\varepsilon} = \frac{1}{N}\sum_{i=1}^{N}\left[1 - \max_i\left(\frac{v_i(x)}{\sum_{j=1}^{L} v_j(x)}\right)\right] \tag{3.5.23}$$

This estimate is not known to bound Bayes error but it is known to have a smaller variance than the estimate found through the voting k-NN procedure.

K-NN estimation is computationally complex.

3.5.2.3 Parzen Estimation

Parzen estimation techniques relay on the same concept as k-NN: setting a local region $\Gamma(x)$ around each sample x and examining the ratio of the samples enclosed k, to the total number of samples N, normalized with respect to region volume v:

$$\hat{p}(x) = \frac{k}{vN} \tag{3.5.24}$$

The difference according to k-NN is that Parzen fixes the volume of local region v. Then the density estimation equation becomes:

$$\hat{p}(x) = \frac{k(x)}{vN} \tag{3.5.25}$$

$p(x)$ represents density and $k(x)$ represents number of samples enclosed in volume.

Estimating the Bayes error using the Parzen estimate is done by forming the log likelihood ratio functions based upon the Parzen density estimates and then using resubstitution and leave-one-out methodologies to find an optimistic and pessimistic value for error estimate. Parzen estimates are however not known to bound the Bayes error.

Parzen estimation is computationally complex.

3.5.2.4 Boundary methods

The boundary methods are described in the work of Pierson [Pie98]. Data from each class is enclosed within a boundary of specified shape according to some criteria. The boundaries can be generated using general shapes like: ellipses, convex hulls, splines and others. The boundary method often uses ellipsoidal boundaries for Gaussian data, since it is a natural representation of those. The boundaries may be made compact by excluding outlying observations. Since most decision boundaries pass through overlap regions a count of these samples may give a measure related to misclassification rate. Collapsing boundaries iteratively in a structured manner and counting the measure again lead to a series of decreasing values related to misclassification error. The rate of overlap region decay provides information about the separability of classes. Pierson discuses in his work a way in which the process from two classes in two dimensions can be expanded to higher dimension with more classes. Pierson has demonstrated that the measure of separability called the **Overlap Sum** is directly related to Bayes error with a much more simple computational complexity. It does not require any exact knowledge of the *a posteriori* distributions. Overlap Sum is the arithmetical mean of overlapped points with respect to progressive collapsing iterations:

$$O_S(mt_0) = \frac{1}{N}\sum_{k=1}^{m}(kt_0)\Delta s(kt_0) \qquad (3.5.26)$$

where t_o is the step size, m is the maximum number of iteration (collapsing boundaries), N is the number of data points in whole dataset and $\Delta s(kt_0)$ is the number of points in the differential overlap.

In his partial PhD work entitled "Using boundary methods for estimating class separability" [Pie98] Pierson writes that: "BMs (Boundary Methods) provide a measure of class separability, the overlap sum (OS), which is strongly correlated with Bayes error and easily computed. These properties suggest BMs can be used as an alternative to traditional Bayes error estimation techniques." The advantage of using Overlap Sum over estimating Bayes error is that Overlap Sum doesn't require knowledge about a posteriori distributions. Overlap Sum is correlated to Bayes error, which is a measure of class separability known to be optimal. Pierson shows theoretically and empirically in his work that there is a direct relationship between BM and Bayes error. Furthermore, he shows an advantage of Overlap Sum complexity measure over k-NN due to computational complexity.

3.5.3 Measures related to space partitioning

Measures related to space partitioning are connected to space partitioning algorithms. Space partitioning algorithms divide the feature space into sub-spaces. That allow to obtain some advantages, like information about the distribution of class instances in the sub-spaces. Then the local information is globalized in some manner to obtain information about the whole database, not only the parts of it.

3.5.3.1 Class Discriminability Measures

Class Discriminability Measures (CDM) [KNM96] are based on the idea of inhomogeneous buckets. The idea here is to divide the feature space into a number of hypercuboids. Each of those hypercuboids is called a "box". The dividing stops when any of following criteria is fulfilled:

- box contains data from only one class;
- box is non-homogenous but linearly separable;

- number of samples in a box is lower that $N^{0.375}$, where N is the total number of samples in dataset.

If the stopping criteria are not satisfied, the box is partitioned into two boxes along the axis that has the highest range in terms of samples, as a point of division using among others median of the data.

Final result will be a number of boxes which can be:

- homogenous terminal boxes (HTB)
- non-linearly separable terminal boxes (NLSTB)
- non-homogenous non-linearly separable terminal boxes (NNLSTB)

In order to measure complexity, CDM uses only Not Linearly Separable Terminal Boxes, as, according to author, only these contribute to Bayes error. That is however not true, because Bayes error of the set of boxes can be greater than the sum of Bayes errors of the boxes - partitioning (and in fact nothing) cannot by itself diminish the Bayes error of the whole dataset, however it can help classifiers in approaching the Bayes error optimum. So given enough partitions we arrive to have only homogenous terminal boxes, so the Bayes error is supposed to be non-existent, that is not true.

The formula for CDM is:

$$CDM = \frac{1}{N}\sum_{i=1}^{M}\{k(i) - \max[k(j|i)]\} \tag{3.5.27}$$

where $k(i)$ is the total number of samples in the i-th NNLSTB, k(j—i) is the number of samples from class $j$in the i-th NNLSTB and N is the total number of samples. For task that lead to only non-homogenous but linearly separable boxes, this measure equals zero.

3.5.3.2 Purity

Purity measure [Sin02a] is developed by Singh and it is presented with connection to his idea based on feature space partitioning called PRISM (Pattern Recognition using Information Slicing Method). PRISM divides the space into cells within defined resolution B. Then for each cell probability of class i in cell l is:

$$P_{il} = \frac{n_{il}}{\sum_{j=1}^{K_l} n_{il}} \tag{3.5.28}$$

where n_{jl} is the number of points of class j in cell l, n_{il} is the number of points of class i in cell l and K_l is the total number of classes. Degree of separability in cell l is given by:

$$S_{H(l)} = \sqrt{\left(\frac{k}{k-1}\right) \sum_{i=1}^{k} \left(p_{il} - \frac{1}{k}\right)^2} \tag{3.5.29}$$

These values are averaged for all classes, obtaining overall degree of separability:

$$S_H = \sum_{i=1}^{H_{total}} S_{H(l)} \frac{N^l}{N} \tag{3.5.30}$$

where N_l signifies the number of points in the l-th cell, and N signifies total number of points.

If this value was computed at resolution B then it is weighted by factor $w = \frac{1}{2^B}$ for $B = 0, 1, ..., 31)$. Considering the curve (SH versus normalized resolution) as a closed polygon with vertices (x_i, y_i), the area under the curve called purity for a total of n vertices is given as:

$$AS_H = \frac{1}{2}\left(\sum_{i=1}^{n-1}(x_i y_{i+1} - y_i x_{i+1})\right) \tag{3.5.31}$$

The x axis is scaled to achieve values bounded within range $[0,1]$. After the weighing process maximum possible value is 0.702, thus the value is rescaled once again to be between $[0,1]$ range.

The main drawback of purity measure is that if in a given cell, the number of points from each class is equal, then the purity measure is zero despite that in fact the distribution may be linearly separable. Purity measure does not depend on the distribution of data in space of single cell, but the distribution of data into the cells is obviously associated with data distribution.

3.5.3.3 Neighborhood Separability

Neighborhood Separability [Sin02a] measure is developed by Singh. Similarly to purity, it also depends on the PRISM partitioning results. In each cell, up to k nearest neighbors are found. Then one measure a proportion p_k of nearest neighbors that come from the same class to total number of nearest neighbors. For each number of neighbors k, $1 <= k <= \lambda_{il}$ calculate the area under the curve that plots p_k against k as ϕ_j. Then compute the average proportion for cell H_l as:

$$p_l = \frac{1}{N^l}\sum_{j=1}^{N_l}\phi_j \tag{3.5.32}$$

Overall separability of data is given as:

$$S_{NN} = \sum_{l=1}^{H_{total}} p_l \frac{N^l}{N} \tag{3.5.33}$$

One compute the measure for each resolution $B = (0, 1, ..., 31)$. Finally, the area under the curve versus resolution gives the measure of **neighborhood separability** for a given data set. The value is rescaled as $AS_{NN} = \frac{S_{NN}}{0.702}$ to be in range $[0, 1]$.

3.5.3.4 Collective entropy

Collective entropy [SG02], [Sin02b] measures degree of uncertainty. High values of entropy represent disordered systems. The measure is connected to data partitioning algorithm called PRISM. Calculate the entropy measure for each cell H_l:

$$E_l = \sum_{i=1}^{K_l} (-p_{il} \cdot \log(p_{il})) \tag{3.5.34}$$

Estimate overall entropy of data as weighted by the number of data in each cell:

$$E = \sum_{l=1}^{H_{total}} E_l \cdot \frac{N^l}{N} \tag{3.5.35}$$

Collective entropy for data at given partition resolution is defined as:

$$E_C = 1 - E \tag{3.5.36}$$

This is to keep consistency with other measures: maximal value of 1 signifies complete certainty and minimum value of 0 uncertainty and disorder.

Collective entropy is measured at multiple partition resolutions $B = (0, 31)$ and scaled by factor $w = \frac{1}{2^{(B)}}$ to promote lower resolution. Area under the curve of Collective Entropy versus resolution gives a measure of uncertainty for a given data set. That measure should be scaled as $AS_E = \frac{S_E}{0.702}$ to keep the values in $[0, 1]$ range.

3.5.4 Other Measures

The measures described here are difficult to classify as they are very different in idea and it's difficult to distinguish common properties.

3.5.4.1 Correlation-based approach

Correlation-based approach [RF98] is described by Rahman and Fairhust. In their work, databases are ranked by the complexity of images within them. The degree of similarity in database is measured as the correlation between a given image and the rest images in database. It indicates how homogenous the database is. For separable data, the correlation between data of different classes should be low.

3.5.4.2 Fisher discriminant ratio

Fisher discriminant ratio [Fis00] originates from Linear Discriminant Analysis (LDA). The idea of linear discriminant approach is to seek a linear combination of the variables which separates two classes in best way. The Fisher discriminant ratio is given as:

$$f_l = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \tag{3.5.37}$$

where $mu_1, \mu_2, \sigma_1, \sigma_2$ are the means and variances of two classes respectively. The measure is calculated in each dimension separately and afterwards the maximum of the values is taken. It takes values from $[0, +\infty]$; high value signifies high class separability. To use it for multi class problem it is necessary however to compute Fisher discriminant ratios for each two-element combination of classes and later average the values.

Important feature of the measurement is that it is strongly related to the structure of data. The main drawback is that it acts more like a detector of linearly separable classes than complexity measure. The advantage is very low computational complexity.

3.5.4.3 Interclass distance measures (scatter matrices)

The interclass distance measures [Fuk90] are founded upon the idea that class separability increases as class means separate and class covariances become tighter. We define: **Within-class scatter matrix:**

$$S_W = \sum_{i=1}^{L} P(\omega_i)\Sigma_i \tag{3.5.38}$$

Between-class scatter matrix:

$$S_b = \sum_{i=1}^{L} P(\omega_i)(\mu_i - \mu_0)(\mu_i - \mu_0)^T \tag{3.5.39}$$

Mixture (total) scatter matrix:

$$S_m = S_w + S_b \tag{3.5.40}$$

where μ_i are class means, $P(c_i)$ are the class probabilities, Σ_i are class covariance matrices, and $\mu_0 = \sum_{i=1}^{L} P(\omega_i) m u_i$ is the mean of all classes.

Many intuitive measures of class separability come from manipulating these matrices which are formulated to capture the separation of class means and class covariance compactness. Some of the popular measures are:

$$J_1 = tr(S_2^{-1}S_1), \quad J_2 = \ln|S_2^{-1}S_1| \quad J_3 = \frac{tr(S_1)}{tr(S_2)} \tag{3.5.41}$$

where S_1, S_2 are a tuple from among S_b, S_w, S_m, and tr signifies trace of a matrix.

Frequently many of these combinations and criteria result in the same optimal features.

3.5.4.4 Volume of the overlap region

We can find volume of the overlap region [HB98] by finding the lengths of overlapping of two classes' combination across all dimensions. The lengths are then divided by overall range of values in the dimension (normalized):

$$r_d = \frac{d_0}{d_{max} - d_{min}} \tag{3.5.42}$$

where d_o represents length of overlapping region, d_{max} and d_{min} represent consequently maximum and minimum feature values in specified dimension

Resulting ratios are multiplied across all dimensions dim to achieve volume of overlapping ratio for the 2-class case (normalized with respect to feature space)

$$v_0 = \prod_{i=1}^{dim} r_d \tag{3.5.43}$$

It should be noted that the value is zero as long as there is at least one dimension in which the classes don't overlap. In order to expand the measure to multi class case, one should take the average of the values computed for all two elements classes' combinations.

3.5.4.5 Feature efficiency

The feature efficiency [HB98] measure describes the efficiency of features in differentiating the classes. It doesn't however contribute to the joint effect of features. If there is an overlap in feature values for two classes we consider the classes ambiguous in that region (along that dimension). If the problem is linearly separable then there exists at least one dimension in which the classes don't overlap each other. For other problems that are globally ambiguous one may progressively remove the ambiguity by separating only the points that lie outside of overlapping region in each chosen dimension. The individual feature efficiency may be defined as the fraction of the remaining points separated by that feature.

3.5.4.6 Minimum Spanning Tree (MST)

The method is based on use of the **Minimum Spanning Tree** (MST)I [FR79]. One constructs a MST that connects all data points to their nearest neighbors regardless of class labels. The number of data points connected to an opposite class by an edge of MST is then computed. The fraction of remaining points to the size of dataset is used as a measure of complexity. The flaw of the approach is that it is disrupted by a small margin narrower than the distance between points of the same class.

3.5.4.7 Inter-intra cluster distance

The average inter-cluster distance is computed by considering all data in both clusters (classes):

$$S_{IC} = \frac{l_1 S_{wi} + l_2 S_{wj}}{l_1 + l_2} \tag{3.5.44}$$

where l_1 and l_2 are the numbers of samples in the two clusters and S_{wi}, S_{wj} represent consequently inter-cluster distance of cluster i and inter-cluster distance of cluster j.

Between-cluster distance S_b is computed in analogical way, by considering the distance between the point pairs across clusters.

The ratio S_w/S_b shows how separate are the two classes. The measure is bounded within range $[0; +\infty]$. A low value indicates that the two datasets are separable. A measure of multi-class problem is done by averaging the values of (S_w/S_b) across all class combination and all features.

3.5.4.8 Space covered by epsilon neighborhoods

The main idea of **space covered by epsilon neighborhoods** is to enclose each class in largest hypersphere possible excluding other classes. The number and size of that hyperspheres define the classification complexity of the problem [Ho00]. The algorithm for the space by epsilon neighborhoods is as follows:

1. compute the starting radius of hypersphere value r_h,

2. for each data point we grow the size of the hypersphere in increments of r_h, and terminate the process when increasing the hypersphere leads to inclusion of samples from other classes.

Average proportional size in terms of number of members in each hypersphere divided by the total number of data points is used as a measure of complexity.

3.5.5 Ensemble of estimators

The idea here is to combine several methods, for example: use weighted average of them. It is possible that a single measure of complexity may be not suitable for practical applications; instead a hierarchy of estimators may be more appropriate [Mad90]. The computation of several methods at once is potentially more difficult than one, but using several simple methods could be faster than one complex method.

3.6 Complexity Estimation Conclusion

Classification complexity estimation methods present great variability. The methods which are derived from Bayes error are most reliable in terms of performance, as they are theoretically stated. The most obvious drawback is that they have to do assumptions about a priori probability distributions. The methods which are designed on experimental (empirical) basis are very various and their performance is difficult to predict. The advantage of the last is that they frequently base uniquely on experimental data and do not need probability density estimates of distributions, as they obtain that in experimental way during the processing. Not every method is suitable to estimate a complexity of multi-class classification problem - some are designed only to two-class problems, and as such they need special procedures to accommodate them to multi-class problem (like counting the average of complexities of all two-class combinations) what usually result in combinatorial increase in computational complexity.

Technique	Relation to Bayes error	Computational complexity	Advantages	Disadvantages
Chernoff bound	direct			Need pdf estimates, 2 class only
Bhattacharyya bound	direct			Need pdf estimates, 2 class only
Divergence		high		2 class only
Mahalanobis distance				2 class only
Matusita distance				2 class only
Entropy measures	not equivalent		not limited class number	
Classifier error	potential	depends on the classifier used		Depends on the characteristics of classifier used - not reliable
k-Nearest neighbors	direct	high	pdf estimates not needed	
Parzen estimation	not known	high	pdf estimates not needed	
Boundary methods	yes	medium	pdf estimates not needed	
Class Discriminability Measures		low		
Purity	no	high		
neighborhood separability		high		
Collective entropy	no	high		
Correlation based approach				
Fisher discriminant ratio		very low		2 class only
Interclass distance measures (scatter matrices)	not equivalent			
Volume of the overlap region		low		
Feature efficiency				
Minimum Spanning Tree				
Inter-intra cluster distance		high		2 class only
Space covered by epsilon neighborhoods				
Ensemble of estimators	variable	probably high	potentially more reliable than single method	

PART II - T-DTS

Chapter 4

Treelike Divide-To-Simplify paradigm

This chapter has to present in detail the new Treelike-Divide To Simplify approach, define its structure, and describe the types of modules that are used in the structure. T-DTS is based on modular treelike decomposition structure, that is used for task decomposition. There are two types of modules: Decomposition Unit (DU) and Processing Unit (PU). It will present also in detail procedures and algorithms that are used for the creation, execution and modification of the modules. It will discuss also advantages and disadvantages of T-DTS approach and compare it with other approaches.

In a very large number of cases, dealing with real world dilemmas and applications (system identification, industrial processes and manufacturing regulation and optimization, decision, pattern recognition, systems and plants safety, etc.), information is available as data stored in files (databases etc.). Especially in industrial areas, efficient processing of data is the chief condition to solve problems. Efficiency concerns not only performance in terms of correct processing, but also temporal aspect of calculations. In the most of those cases, processing efficiency is closely related to several issues among which are:

- **Data nature**: the properties of data; includes complexity, quality and representativeness:
  - **Data complexity**, related to nonlinearity, may affect the processing efficiency.
  - **Data Quality** (noisy data, etc.): may influence processing success and expected results quality.
  - **Representativeness**: concerning scarcity of pertinent data, could affect processing achievement.
- **Processing technique related issues:** including model choice, processing complexity and intrinsic processing delay.

The choice or availability of appropriated theoretical model describing the behaviour related to the processed data is of major importance. Processing technique and algorithm complexity (designing, precision, etc) shapes the processing effectiveness. Intrinsic processing delay or processing time is related to the processing technique's implementation (software or hardware related issues).

One of the key points on which one can act is the complexity reduction. It concerns not only the problem solution level but also appears at processing procedure level. The constraints relative to the nature of data to be processed, difficult dilemma related to the choice of appropriated processing techniques and allied parameters make complexity reduction a key point on both data and processing levels. T-DTS (Treelike Divide To Simplify) paradigm which is a lead motive of this thesis is able to reduce complexity on both data and processing levels. The main idea of the T-DTS is based on the notion "Divide et impera" (Julius Caesar), transformed here to "Divide To Simplify" (DTS).

The main idea is to split a complex problem into many easier subproblems. T-DTS have a modular structure that allows parallel processing and simplification of the problem. The purpose is based on the use of a small set of specialized mapping Neural Networks, called Neural Network Models (Processing Unit), supervised by a Decomposition Unit (DU). Decomposition Unit could be a prototype based neural network, Markovian decision process, etc. The modules responsible for processing in the structure are Artificial Neural Networks (models). The T-DTS paradigm allows us to build a tree structure. At the node's level, the input space is decomposed into a set of subspaces of smaller sizes. At the leaf's level the aim is to learn the relations between inputs and outputs relatives to one of sub-spaces, obtained from splitting.

The organization of this chapter is as follows: next section 4.1 is a general description of T-DTS. Sections 4.2 to 4.7 concentrate on specific aspects of T-DTS algorithm: 4.2 - building of decomposition structure, 4.3 - decomposition of learning database, 4.4 - training of Neural Network models, 4.5 - decomposition of generalization database, 4.6 - using trained Processing Unit models on data, 4.7 - combining obtained results. Section 4.8 is a conclusion of this chapter, so it discuses properties of T-DTS, pointing out the advantages and the disadvantages of the paradigm.

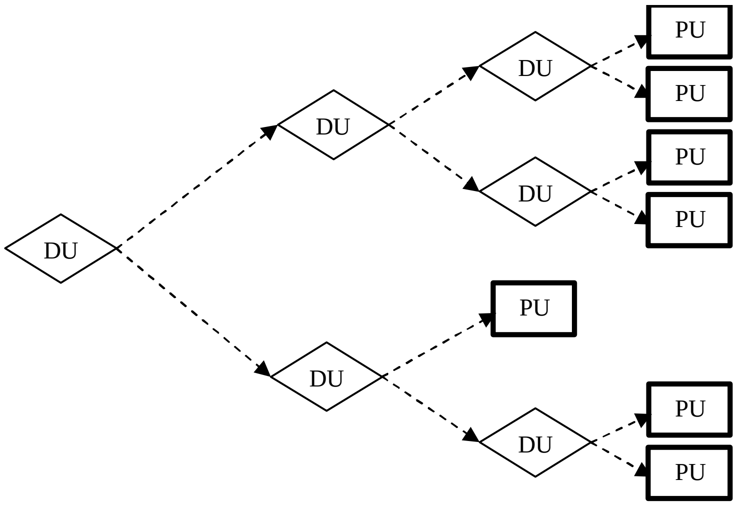


Figure 4.1: T-DTS decomposition architecture

4.1 General description of T-DTS

This section will present the general overview of T-DTS algorithm. Particular procedures will be explained in detail in the consecutive sections of the chapter.

T-DTS (Treelike Divide To Simplify) is a data driven Neural Networks based Multiple Processing (multiple model) structure. The idea is based on problem decomposition paradigm. It can reduce complexity on both data and processing chain levels [MCR03b]. T-DTS constructs a treelike architecture, where nodes are Decomposition Units (DU) and leaves correspond to Processing Units (PU), as presented in figure 4.1. The structure is used to decompose the learning database into sub-databases and can be used to assign the generalization database to models. Processing Units process specific segments of feature space, and are trained using learning sub-databases. The Processing Units are trained in supervised mode.

T-DTS algorithm consist of two main phases, first is connected to structure building and model training, while the second is equal to using the trained structure on generalization data. The detailed algorithm is as follows:

1. **Learning phase**
    - Building of decomposition structure using training data,
    - decomposition of training dataset,
    - training of Processing Unit using learning data sub-databases.
2. **Operation phase**
    - Vector to model assignment of generalization dataset,
    - using corresponding Processing Unit models on data sub-databases,
    - combining the results obtained from Processing Unit models.

The T-DTS compounds of two main operation modes. The first phase is the learning phase, when T-DTS system decomposes the input data and provides processing substructures and tools for decomposed sets of data. The second phase is the operation phase (usage of the system to process unlearned data). There could be also a preprocessing phase at the beginning, which arranges (prepares) data to be processed.

Preprocessing phase could include several steps (conventional or neural stages). Preprocessing is expected to ease the processing of data. During preprocessing, several operations such as data normalizing, data scaling, data dimensionality reduction could be performed. Preprocessing could also include other kind of operations, as removing outliers or Principal Component Analysis [TB97] to enhance input data quality, eliminate redundancy in data, etc.

The learning phase is an important phase during which T-DTS performs key operations: building the decomposition tree, splitting the learning database into many sub-databases, constructing (dynamically) a treelike structure of Decomposition Unit (DU) and building a set of submodels (Processing Unit) at leaf level of the treelike structure (corresponding to each sub-database).

The splitting (during the learning phase) could lead to two general cases. The first one corresponds to the situation where the splitting process doesn't modify the feature space dimension. That means that the initial problem's is decomposed into M sub-problems. Instead of building one complex model T-DTS builds M easier models describing behaviour in each related sub-problem. That correspond to the situation where the splitting process divides the initial feature space into M feature spaces with smaller dimensions (that doesn't means that the obtained feature sub-spaces will be orthogonal). So, in this case, the activation of appropriated Processing Unit will not depend on the complete input vector but on some partial input vector $\phi \in \mathbb{R}^{\phi}$ with $\phi \in \mathbb{R}^{\phi} \subseteq \Psi$.

Let i be an n-dimensional input pattern vector, where $\psi_i \in \Psi^{n_\Psi}$. Let $F_k(\cdot) : \mathbb{R}^{n_\Psi} \rightarrow \mathbb{R}^{n_Y}$ be the k-th Processing Unit's transfer function. $Y_k(i) \in \mathbb{R}^{n_Y}$ will be then the k-th ($k \in 1, 2, ..M$) model's output vector of dimension n_y. Let $S(\psi, \tau, \xi)$ where $B = 0, 1$ be the Decomposition Unit's (DU) output, called also Scheduling Function, which depends on $\psi(i)$, but which may also depend on some parameters τ and/or conditions ξ. τ_k represents some particular values of parameter τ and ξ_k denotes some particular value of condition ξ , respectively, obtained from learning phase process for the k-th sub-dataset.

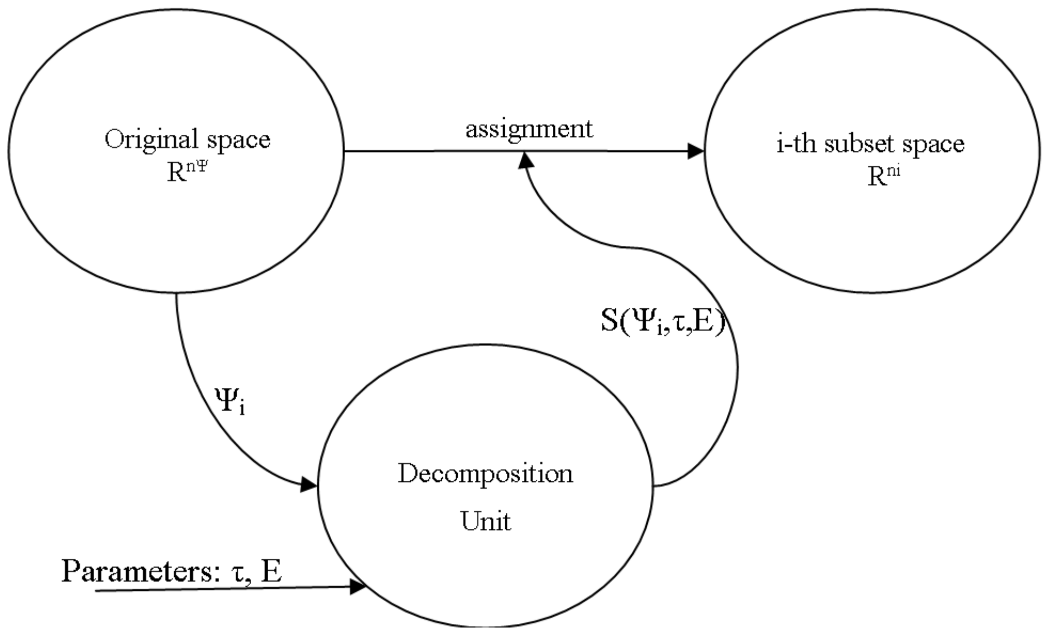


Figure 4.2: Decomposition Unit activity

$$S(\psi,\tau,\xi) = (s_1 ... s_k, ... s_M)^T \text{with} \left[\begin{array}{ll} s_k = 1 & \text{if} \quad \tau = \tau_k \text{ and } \xi = \xi_k \\ s_k = 0 & \text{else} \end{array} \right. \tag{4.1.1}$$

The scheduling vector $S(\psi_i, \tau_k, \xi_k)$ will activate (select) the k-th Processing Unit, and so the processing of an unlearned input data conform to parameter τ_k and condition ξ_k will be given by the output of the selected Processing Unit:

$$Y(\psi_i) = Y_k(i) = F_k(\psi_i) \tag{4.1.2}$$

The output is then gathered from Processing Unit models by the same multiplexer-like structure, which joins together the output values form individual modules to achieve output for the whole database.

4.2 Building of decomposition structure

For each sub-database k, T-DTS constructs a neural based model describing the relations between inputs and outputs. The decomposition structure in T-DTS approach is a tree. The creation of *decomposition tree* is data-driven. It means that the decision to-split-or-not and how-to-split is made depending on the properties of the current data sub-database (data sub-database which had arrived to some position at decomposition tree). For each database the decision to-split-or-not should be made. After positive to-split-or-not decision a Decomposition Unit (DU) is created which divides the achieved data and distributes the resulting sub-databases creating children in the tree. If the decision is negative the decomposition of this data sub-database (and tree branch) is over and a Processing Unit should be built for the sub-database. The type of the tree child depends though on the result of decision made for the current sub-database (and in some cases also on other parameters, as described in section 4.2.1). The tree is built beginning from the root which achieves the complete learning database. The process results in a tree which has DUs at nodes and Processing Unit models in tree leaves.

Figure 4.3 shows decomposition tree structure (in case of binary tree) and its recurrent construction in time, while question marks mean decomposition decisions.

Complexity indicators could be used in our approach in order to reach one of the following ways:

- Global decomposition control - estimator which evaluates the difficulty of classification of the whole dataset and chooses decomposition strategy and parameters before any decomposition has started,

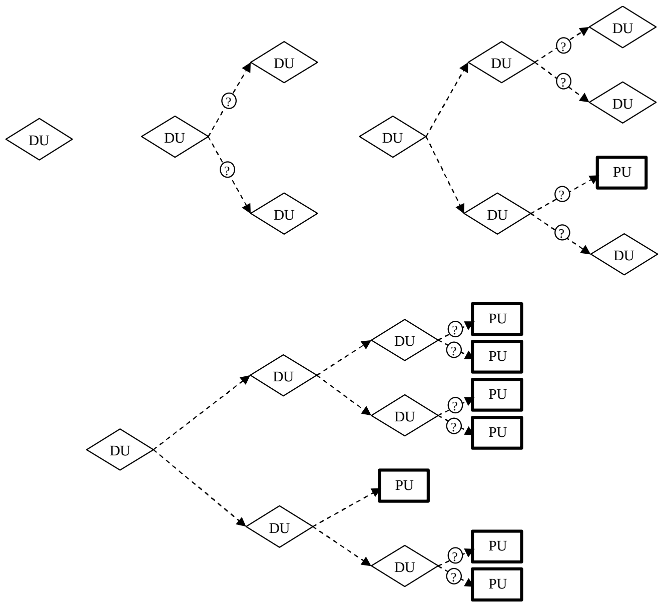


Figure 4.3: T-DTS decomposition tree creation in time

- Local decomposition control - estimator which evaluates the difficulty of classification of the current sub-database during decomposition of dataset, in particular:
  - Estimator which evaluates the difficulty of classification of the current sub-database during decomposition of dataset, to produce decomposition decision (if to divide the current sub-database or not);
  - Estimator which can be used to determine type of used classifier or its properties and parameters.
- Mixed approach - use of many techniques mentioned above at once, for example: usage of Global decomposition control to determine the parameters of local decomposition control.

The local and global decomposition control may seem similar, but they are in fact very different. In general case, estimation of whole dataset complexity is much more difficult and prone to faults (occurs only once, large size of data) than estimation of sub-database (occurs multiple times, data size is reduced; especially on lower levels of decomposition tree, data complexity is reduced). Estimation of current sub-database's complexity can be done by less advanced techniques as it is relatively small part of system comparing to the global decomposition control that affects directly all calculations. One should mention also that estimation of sub-database complexity occurs for each sub-database dividing decision thus computational complexity of the algorithm should rather be small. Thus it doesn't require advanced complexity estimation methods. Considering these features, the second option - estimation during decomposition - has been chosen in our experiments in order to achieve auto-adaptation feature of system.

The decomposition decision can be based on many techniques; we have used two approaches: **local decomposition control based on standard deviation threshold** and **local decomposition control based on complexity estimation**.

4.2.1 Decomposition Unit

The purpose of Decomposition Unit is to divide the database into several sub-databases. This task is referred in the literature as clustering. To accomplish this task a plenty of methods are known. We are using unsupervised competitive Neural Networks and in particular Kohonen Self-Organizing Maps. These methods are based on prototype, that represent the centre of cluster (cluster = group of vectors). In out approach cluster is referred to as sub-database.

4.2.2 Local decomposition control based on standard deviation estimation threshold

Local decomposition control, in particular an approach called by us AVStd threshold, is based on the Standard Deviation measurements. It allows achieving regular decomposition in terms of vectors distribution in problem space dissections. It is unsupervised in term that doesn't need the targets (output values, i.e. classes), but only their features (coordinates in feature space). It could be applied to any input-output mapping problem, and the decomposition amount can be controlled by operator by changing the AVStd threshold.

The whole set of data is normalized at the beginning to achieve standard deviation equal to one. (Additional advantage of normalization is that some distance-based algorithms treat variables in "democratic" way) Then the recurrent decomposition starts. The threshold for deciding about current set decomposition is based on standard deviation of set. Specifically it could be:

- maximum value of standard deviations (by dimension);
- average value of standard deviations (by dimension);
- minimum value of standard deviations (by dimension), etc.

The standard deviations of sub-databases will decrease gradually as the sub-databases will be more and more divided. The standard deviation was equal to one in original (not-divided) database, so the standard deviations of sub-databases will be monotonically dropping form 1 to 0 (0 means that there is only one vector in sub-database or all vectors in sub-database are equal). When the standard deviation of currently analyzed sub-database will be lower than globally defined threshold value,

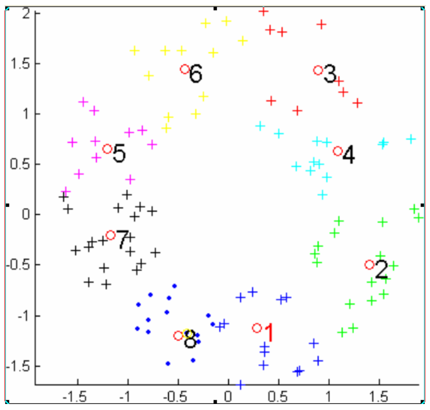


Figure 4.4: Decomposition using AVStd threshold

then the decomposition of the sub-database is abandoned. The decomposition at its extreme leads to as many sub-databases as the number of data vector in the original database is. This is of course not interesting for us, as the goal is somewhere between - we want to achieve sub-problems that are less difficult than original database, but still large enough to be efficiently processed by leaves - processing modules.

The decomposition obtained in such a way takes into account in non-explicit way the area occupied by sub-database and the dispersion of data in sub-database. With use of competitive ANN as decomposition algorithms it allows to achieve regular decomposition, as could be seen in figure 4.4.

The splitting process starts by evaluating the average of standard deviation of the learning database. If the obtained standard deviation is greater that the AvStd, then a distance based competitive ANN (for example Kohonen SOM) divides the learning database into sub-databases. These operations are repeated until the standard deviation relative to each created sub-database doesn't exceed the AvStd value.

4.2.3 Local decomposition control based on classification Complexity Estimation technique (Fisher discriminant ratio)

Local decomposition control with classification complexity estimation could be applied only to classification problem, because classification complexity methods could be applied only in classification. The decomposition degree can be controlled by operator or also measured by classification complexity estimation and controlled automatically. The automatic statement of decomposition parameters was however not included in our experiments. Local decomposition control is supervised in term that it needs both feature values and class labels to measure the classification complexity.

T-DTS could be used as an auto-adapting system, i.e. system which structure adapts to difficulty of data [RCM03a]. Thus it needs to measure the amount of necessary decomposition. The goal here is to adjust the decomposition according to the complication of task. This is done by using classification complexity estimation methods. Then it can achieve desired quasi-constant performance (in terms of misclassifications at cost of increased computational effort) with datasets of various difficulties, see section 5.2.3 for details.

T-DTS in classification-like application can incorporate classification complexity estimation indicators (described thoroughly in chapter 2). Then the complexity estimation take place to determine if actual set of data should be decomposed using DU, or rather a Processing Unit can be created directly (assuming that Processing Unit can process it efficiently). The bloc diagram of T-DTS with incorporated complexity estimation is depicted in Figure 4.5.

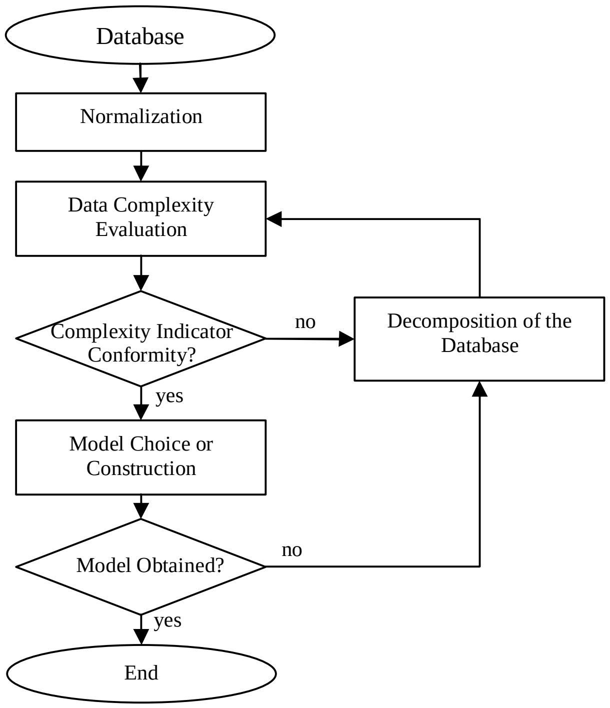

Database
Normalization
Data Complexity Evaluation
Complexity Indicator Conformity?
no
Decomposition of the Database
yes
Model Choice or Construction
Model Obtained?
no
yes
End

Figure 4.5: Bloc diagram of T-DTS

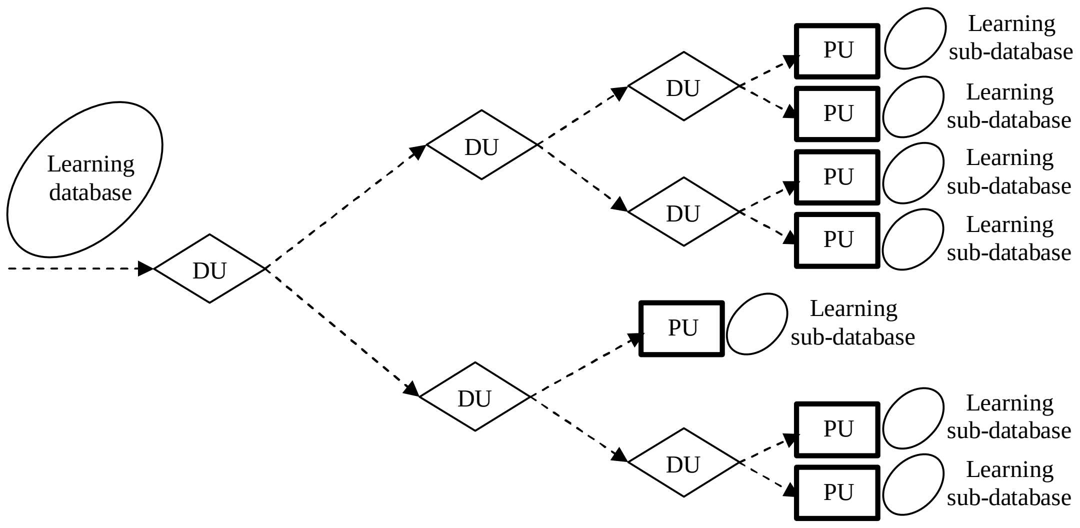


Figure 4.6: Decomposition of learning database using ”following the decomposition tree” strategy

4.3 Decomposition of learning database

The learning database is split into M learning sub-databases by DUs during building of the decomposition tree. The learning database decomposition is should be equivalent to ”following the decomposition tree” decomposition strategy (section 4.5.1). The resulting learning sub-databases could be used for Processing Unit learning. Each subdatabase has then Processing Unit attached. The Processing Unit models are trained using the corresponding learning sub-database.

4.4 Training of Neural Network Models

Training of Processing Unit models is performed using standard supervised training techniques, possibly most appropriate for the learning task required. In this work only Neural Networks are used, however there should be no difficulty to use other modelization techniques.

Processing Unit is provided with a data sub-database and target data. It is expected to model the input/output mapping underlying the subspace as reflected by the data sub-database provided. The trained model is used later in processing of the data patterns assigned to the Processing Unit by assignment rules, as specified in the previous section. Processing Unit models used in this work are: LVQ, MLP, LN, Perceptrons, GRNN, and Probabilistic Networks.

4.5 Decomposition of generalization database - rules of pattern assignment to models

When the generalization database is to be processed each vector from the database should be assigned to a model. In other words there should be defined a function that for each generalization vector returns processing model that should process this vector. This is done by using assigniment rules. The rules of assignment are expected to determine which model will process the given input pattern in generalization phase, given learning sub-databases. In particular the learning sub-databases can be represented by prototypes and characterized by other techniques and attributes. Each k-th sub-database has Processing Unit assigned, which was built to process the k-th sub-database.

Pattern assignment can be performed in two general manners:

- following the decomposition tree (based on the decomposition tree): as decision tree and making decisions at each branch node, starting from the root

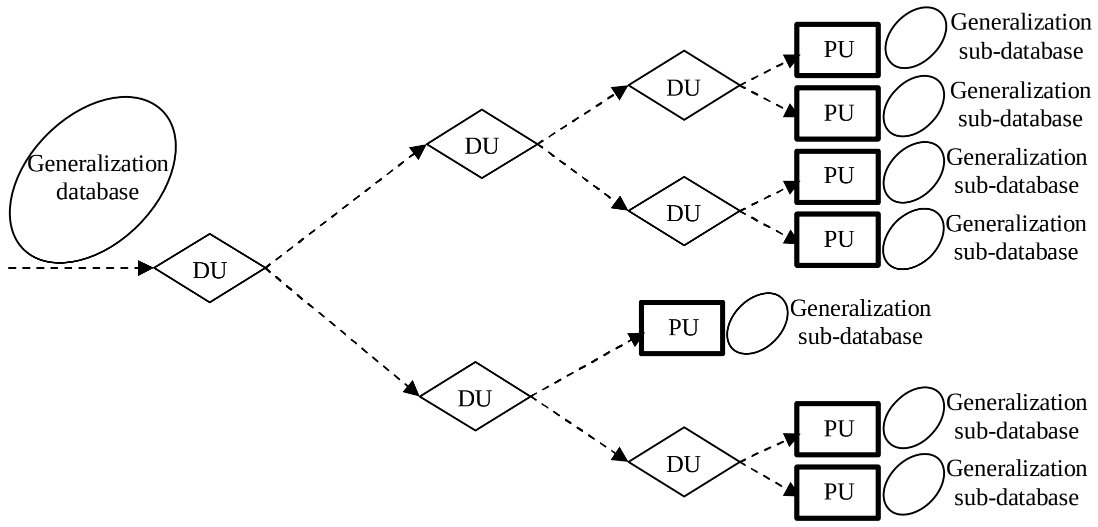

Generalization database
DU
DU
DU
DU
DU
DU
PU
PU
PU
PU
PU
PU
PU
Generalization sub-database
Generalization sub-database
Generalization sub-database
Generalization sub-database
Generalization sub-database
Generalization sub-database
Generalization sub-database

Figure 4.7: Decomposition of generalization database using "following the decomposition tree" strategy

- decomposition based on learning sub-databases: using the properties of learning sub-database to determine the similarity between pattern and learning sub-database. In particular the prototypes of learning sub-databases are represented by learning sub-databases prototypes, and the similarity could be measured between the prototype and pattern (called here prototypes based assignment). Then some criterion is used to choose the most appropriate prototype for each vector or use more than one and combine somehow the results.

4.5.1 Following the decomposition tree

Following the decomposition tree seems to be in line with decomposition tree creation technique. Following the tree requires many decisions for each data vector.

4.5.2 Prototypes based assignment

Using of prototypes based assignment seems to be more autonomous in a way that it's independent of decomposition tree creation technique. Prototypes based assignment requires only one decision per data vector. Decision used in prototypes based assignment could be based on similarity criterion. Similarity criterion could be commonly used distance measures (described in section 2.2). Next two sections present two Prototypes based assignment rules.

4.5.2.1 Prototypes similarity assignment rule

In this case, splitting process dividing the initial complex problem into M reduced sub-problems is based on similarity criterion. The activation of an appropriate Processing Unit will be issued from similarity measure between an unlearned input vector i and the k-th cluster prototype representative (Wk). As previously, it is supposed that the initial feature space has been decomposed into M clusters by a competitive network decomposition tree.

The similarity criterion could be based on prototypes of learning sub-databases. A prototype is a vector representing the set of vectors, obtained in some way. We use usually a prototype obtained from Competitive Network during building of decomposition tree. Then the similarity criterion is expressed as a distance between data vector and prototype. The most natural distance function is the one used during decomposition phase, but there is plenty of distance functions available, some of them are described in section 2.2.

Properties of similarity assignment:

- **universality** - can use any measure of similarity

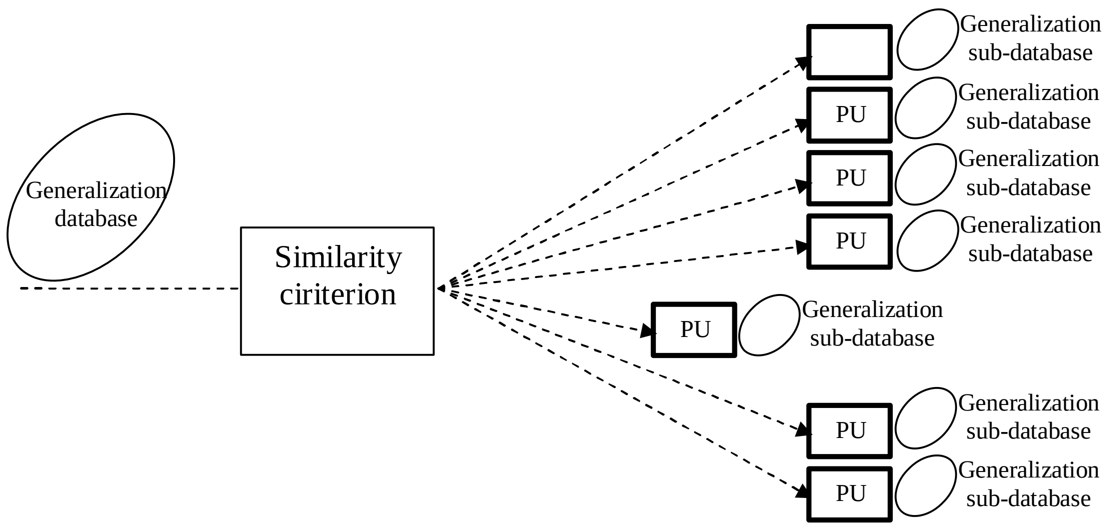


Figure 4.8: Decomposition of generalization database using "similarity matching" strategy

- deterministic or fuzzy assignment

Maximum similarity (minimum distance) deterministic assignment The Scheduling vector (DU output) will be conform to relation 4.1.1, with $sk(\psi_i, W_k)$ given by:

$$s_k(\psi_i, W_k) = \left[\begin{array}{ll} 1 & \text{if } \psi_i - W_k = \min(\psi_i, W_k) \\ 0 & \text{else} \end{array} \right. \qquad (4.5.1)$$

So to be specific: the closest model (minimum distance) is chosen to process pattern.

Fusion similarity assignment

This approach is a generalization of maximum similarity assignment. Scheduling vector contains here values for range $< 0, 1 >$, with sum of all elements equal to 1. The feature vector is directed not only to one Processing Unit like in maximum similarity assignment, but to many of them at certain degree.

$$s_k(\psi_i, W_k) = \frac{|\psi_i - W_k|)}{\sum_j |\psi_i - W_j|)} \tag{4.5.2}$$

The output for i-th feature vector is then weighted sum of outputs of all Processing Unit models.

$$o(\psi_i) = \sum_j o(\psi_i, NNM_j) s(\psi_i, NNM_j) \tag{4.5.3}$$

In this way a feature vector is processed by all Processing Unit models, but the influence of Processing Unit output on the output is proportional to the similarity between feature vector and learning sub-database corresponding to the Processing Unit. That technique could be improved by considering only output of specified number of closest Processing Unit models (when only one Processing Unit is considered this approach is equal to maximum similarity assignment). The number of closest Processing Unit models could be determined by That significantly reduces the number of calculations and allows the Processing Unit models to be local.

4.5.2.2 Probabilistic assignment rule

In this case, the activation of an appropriate Processing Unit is given in terms of probability of activation the k-th NMM among M neural network based models. If the k-th Processing Unit has been obtained with respect to the k-th learning sub-database, then the probability of activation of the corresponding Processing Unit, Pk((t)), could be expressed by relation (3), where k( ) is some Gaussian approximation of k-th learning sub-database density.

$$P_k(\psi) = \frac{\rho_k(\psi)}{\sum_{k=1}^{M} \rho_k(\psi)} \tag{4.5.4}$$

with

$$\rho_k(\psi) = \exp\left[-\frac{(\psi - \mu_k)^T(\psi - \mu_k)}{\rho_k^2}\right. \tag{4.5.5}$$

where μ_k represents the average prototype's centre and ρ_k denoting the learned prototypes standard deviation. The approach has following properties:

- $P_k(\psi(t))$ and $k()$ are reverse proportional to the distance between the input vector and sub-database prototype;
- $P_k(\psi(t))$ and $k()$ are reverse proportional to the learned prototypes standard deviation ρ_k;
- The exponential function strongly awards the prototypes that are close to μ_k and punishes the prototypes that are far from μ_k ;
- Sum of probabilities is equal to 1.

Probabilistic assignment The Scheduling vector (DU output) could be determined randomly with probabilities of assignment to Processing Units defined above.

Maximum probability assignment The Scheduling vector could be expressed according to relation 4.1.1 , where $s_k(\psi_i, P_k)$ is given by:

$$s_k(\psi_i, P_k) = \left[\begin{array}{ll} 1 & \text{if } P_k = \max(P_l) \\ 0 & \text{else} \end{array}\right. \tag{4.5.6}$$

with $l \in \{1, ..., M\}$.

Then the assignment is deterministic and the model associated with sub-database most close to input pattern vector wins. The assignment could be unfair when two or more models have similar values of probabilities for given input pattern.

4.6 Using trained models (Processing Units) on data

Processing Unit models used in our approach can be of any origin. In fact they could be also not based on Artificial Neural Networks at all. The structure used depend on the type of learning task, we use:

- for classification - MLP, LVQ, Probabilistic Networks, RBF, Linear Networks;
- for regression - MLP, RBF;
- for model identification - MLP.

Processing Unit models are created and trained (section 4.4) in the learning phase of T-DTS algorithm, using learning sub-databases assigned by decomposition structure (section 4.3). In the generalization phase, they are provided with generalization vectors assigned to them by pattern assignment rules (section 4.5). The vectors form generalization sub-databases that are processed by Processing Unit models. Each Processing Unit produce some set of approximated output vectors.

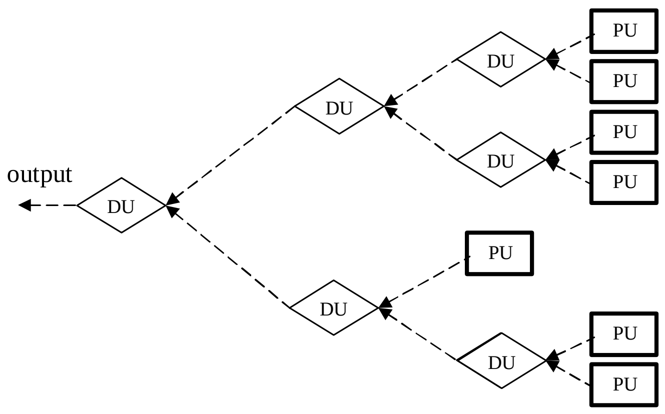


Figure 4.9: Dataflow of T-DTS: gathering of the results from individual Processing Units

4.7 Combining the results

The sets of output vectors are combined to produce the output set for the whole generalization database. Figure 4.9 represents the dataflow of output values, when DU units compose a binary tree.

On the figure 4.9 one can notice a tree structure of DU nodes and Processing Unit leaves. A set of neural network based models (trained from sub-databases) is available at leaf level and model the system behavior region-by-region in the problem's feature space. Each incoming input vector is assigned to specific Processing Unit leaf by pattern assignment rules. Then the processed data is gathered together from Processing Unit models by the DUs (in reverse direction).

4.8 Conclusion - Discussion of T-DTS properties

T-DTS (Treelike Divide To Simplify) is a multiple model (in particular Multiple Neural Networks) processing structure. It is able to reduce complexity on both data and processing chain levels [MCR03b]. T-DTS constructs a treelike evolutionary architecture of models, where nodes (DU) are decision units and leaves correspond to Neural Network - based Models (Processing Unit). That results in splitting the learning database into set of sub-databases. For each sub-database a separate model is built. Advantages:

- allows simplification of the problem - using many simple local models;
- allows parallel processing - after decomposition, the sub-databases can be processed independently and joined together after processing;
- task decomposition is useful in cases when information about system is distributed locally and the models used are limited in strength in terms of computational difficulty or processing (modelization) power
- modular structure gives universality: it allows using of specialized processing structures as well as replacing Decomposition Units with another clustering techniques,
- classification complexity estimation and other statistical techniques may influence on the parameters to automate the processing i.e. decrease the need for user intervention

Disadvantages:

- if the problem doesn't require simplification (problem is solved efficiently with single model) then Task Decomposition may decrease the time performance, as learning or executing of some problems divided into subproblems is slower than learning or executing of not split problem; especially if using sequential processing (in opposition to parallel processing),

- some problems may be naturally suited to solve by one-piece model,
- too much decomposition leads to very small learning sub-databases, that may lose generalization properties, in extreme case leading to problem solution based only on distance to learning examples, so equal to nearest-neighbor classification method.

Chapter 5

Applications of T-DTS System

This chapter will present applications of T-DTS paradigm to some problem examples. It has to answer the question: if D-DTS approach is universal or not? The answer is yet not simple. T-DTS was validated in number of applications, including classification, system identification and pattern recognition. It is impossible however to imagine and test all the cases in which such an approach could be used. The examples show however that T-DTS is able to process efficiently various tasks: model identification (section 5.1) and classification (section 5.2). This chapter will try to present the applications of T-DTS to some of the most popular classes of computing problems.

Two signal identification problems will be presented. The first is rather simple academic signal identification problem, where T-DTS were supposed to model a dynamic non-linear system, basing on Auto Regressive Moving Average *ARMAX(6,6)*-like data (six last outputs and six last inputs are given in order to provision of system response). It is presented in section 5.1.1. Second signal identification problem concerns real industrial signal identification problem - drilling rubber (section 5.1.2).

Several non-linear parameters influence the process of manufacturing rubber and the goal of T-DTS is is to model the complicated non-linear process in order to identify the global process.

Three classification problems will be presented. First of them, presented in section 5.2.1, is simple academic classification problem of classifying two spiral decision regions (section 5.2.1). This not linearly separable problem is similar to generalized Exclusive Or (XOR) - the goal is to classify properly two interlocking data clusters of spiralled shape. Second application is pattern recognition problem of recognizing separated printed letters (section(5.2.2). T-DTS uses here support from data extraction algorithm in order to encode the data. The third and last application is aimed at showing the usage of complexity estimation techniques in order to automatically classify problems of various difficulties, when keeping the system aware of problem difficulty and responding with equivalent amount of resources (section 5.2.3). The goal of this application is to automatically increase the processing effort for more complicated problems, when simultaneously keeping it low for simple problems.

5.1 Model identification

5.1.1 Model identification - linear academic problem

To evaluate T-DTS in system identification task, the T-DTS paradigm has been applied to identify a dynamic non linear system [MCR02]. This system is described by the following equations:

$$O_n = 0.18/O_{n-1} + 0.3/O_{n-2} + 0.6/i_n^3 + 0.18/i_n^2 - 0.2/i_n \tag{5.1.1}$$

where o_n represent the system output and i_n the system input at time step n . In the learning phase, the signal $i^l(t)$ is used as system input, in the generalization phase signal $i^g(t)$ is used as system input.

$$i^l(t) = 0.7\sin\left(\frac{2\pi t}{300}\right) + 0.3\sin\left(\frac{2\pi t}{30}\right) \tag{5.1.2}$$

$$i^g(t) = 0.7\sin\left(\frac{2\pi t}{300}\right) \tag{5.1.3}$$

As decomposition engine (DU) a competitive network of 2 neurons is used. As Neural Network Models, multilayer perceptrons were used (1 hidden layer with 4 neurons) with Levenberg-Marquadt learning rule (a variant of back-propagation learning). MLP input patterns are constituted as an Auto Regressive Moving Average ARMAX(6,6) vector.

The T-DTS algorithm builds a tree structure as represented in figure 5.1. This tree is constituted by 3 DU at the node level and 4 Neural Network Models at the leaf level. The initial problem space is split in two sub-databases 1a and 1b. The algorithm decides to end decomposition for sub-database 1a and decides to decompose the second subspace, 1b, in to two sub-spaces 2a and 2b. Then it decides to end decomposition for the subspace 2a (Processing Unit2) and split the subspace 2b into sub-databases 3a and 3b. The sub-databases 3a and 3b are not decomposed any further. That ends decomposition tree building phase. Then the models 1, 2, 3 and 4 are created and trained using the learning sub-databases that have arrived to their location in the tree. That ends T-DTS learning phase.

Figure 5.2 represents patterns that have been used to train these 4 Processing Units.

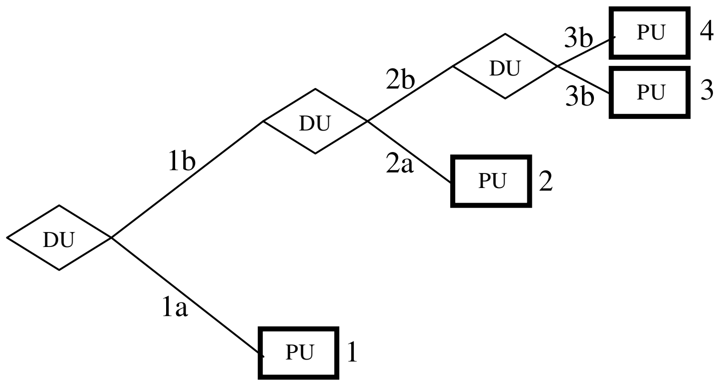


Figure 5.1: Decomposition tree for the model identification experiment

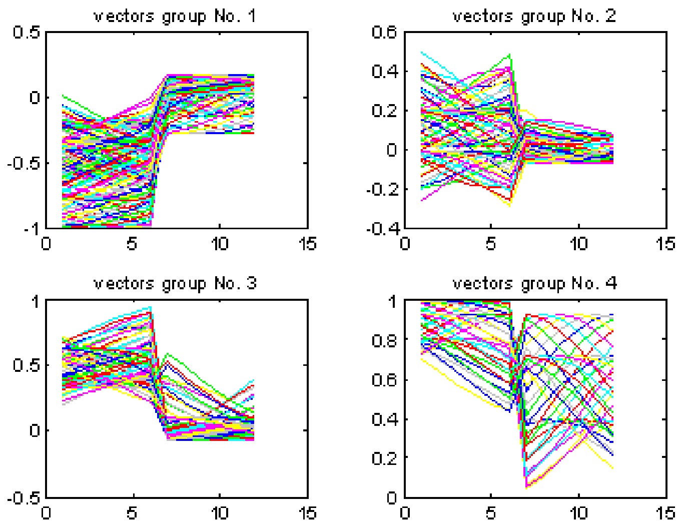


Figure 5.2: Aggregations of patterns created by decomposition

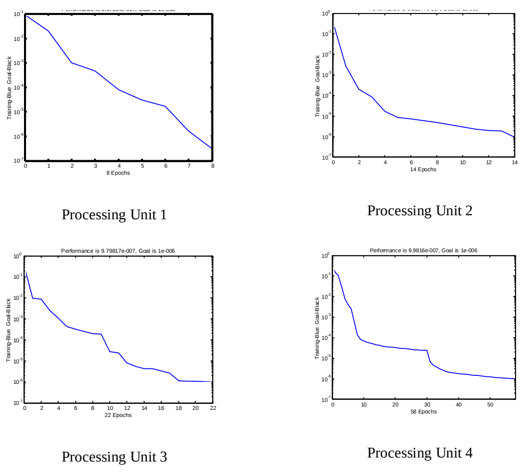


Figure 5.3: Evolution of learning error for models (LM learning)

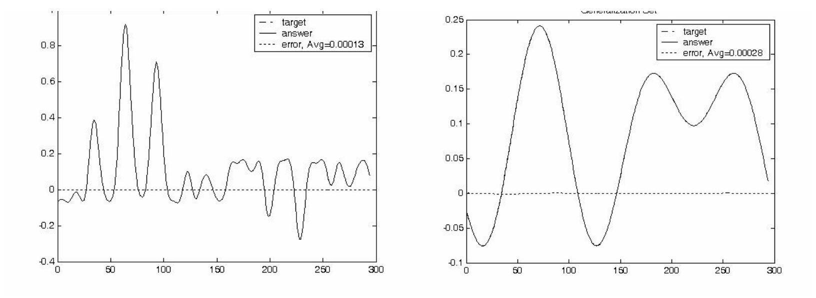


Figure 5.4: Learning signals (left) and generalization signals (right)

Figure 5.3 represent the mean square error evolution in the training step for the 4 Processing Units.

Figure 5.4 represents the T-DTS model output in the learning and generalization steps.

The decision of splitting a subspace is taken basing on the similarity of examples. The data is split recursively by an unsupervised neural network, competitive network in this case. The splitting process has discriminated four learning databases that

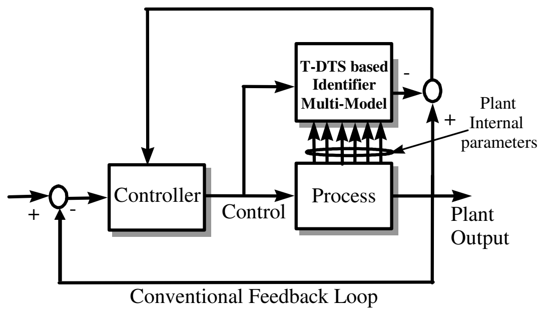

T-DTS based Identifier Multi-Model
-
+
Plant Internal parameters
+
-
Controller
Control
Process
Plant Output
Conventional Feedback Loop

Figure 5.5: Implemented industrial processing loop using T-DTS identifier

contain similar patterns (as can be verified by examining the figure 5.2). The reduction of number of data processed by one Processing Units while keeping the examples similar inside one learning database makes processing easier and faster: few epochs or recursion are needed to reach a 10^{-6} mean square error. The difference between the system output and the T-DTS estimated output, in the learning and generalization step, is very low, as seen in the figure 5.4. This assures us that a faithful model was built, proving the efficiency of T-DTS in system identification task.

5.1.2 Model identification - drilling rubber problem

This is presented in [MCR03a]. This section presents application of T-DTS paradigm to real industrial problem. The problem is a non-linear process identification, in industrial process control problem. The process is a drilling rubber process used in plastic manufacturing industry. Several non-linear parameters influence the manufacturing process. To perform an efficient control of the manufacturing quality (process quality), one should identify the global process.

Similar approach, as described in the previous section 5.1.1, has been implemented. Input patterns have M-ARMAX shape (Multi inputs ARMAX model). Figure 5.6

shows some of process signals shapes. Figures present the main parameters that are involved in the drilling rubber process. Other parameters, that are less significant, have been omitted.

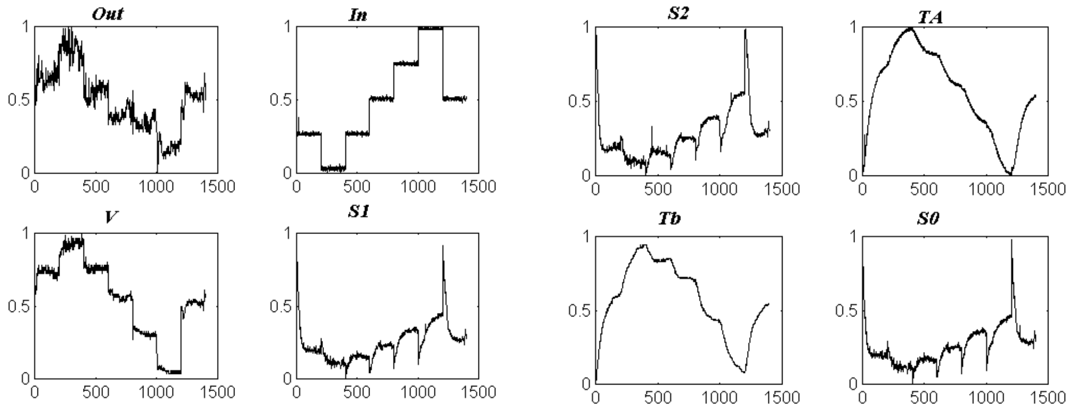
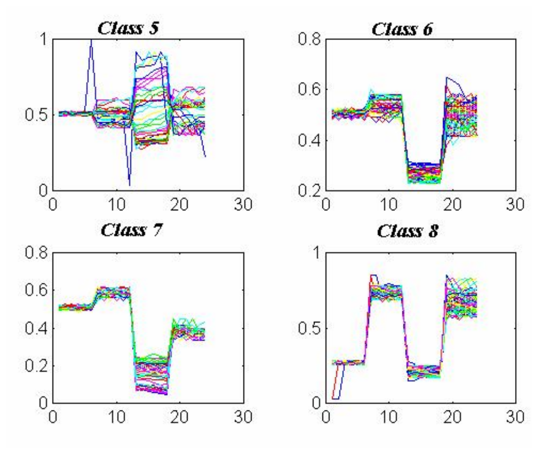


Figure 5.6: Example process input order, output (metric properties of produced profiles) and some of process parameters (confidential) shapes.

Kohonen SOM based Decomposition Unit (DU) uses a $4x3$ grid leading to 12 feature sub-spaces. Consequently, 12 Neural Network based Models (Processing Unit) have been generated and trained (from learning database). Figure 5.7 shows examples of database splitting after T-DTS learning phase, giving four among twelve obtained sub-databases. It shows also, the learning phase validation presenting the learned process output identification. Figure 5.8 shows system output in the generalization phase. One can conclude that estimated output is in accord with the measured one.

T-DTS were used here to identify complicated industrial process with many input features. The original data was split into several sub-databases, allowing faster processing and simplification of modelling task. The models then were built for each sub-database resulting in good estimation properties of resulting system.



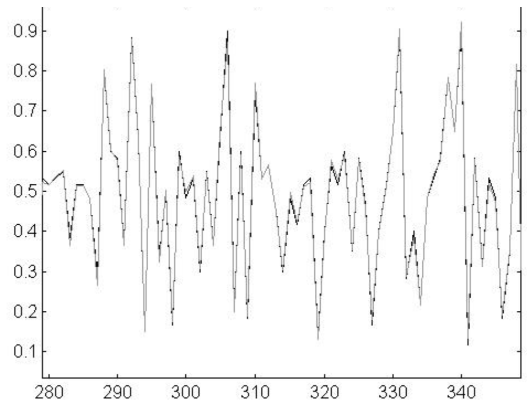


Figure 5.7: Examples of database splitting after T-DTS learning phase: four amongst twelve obtained sub-databases (left). Learned process output identification (right)

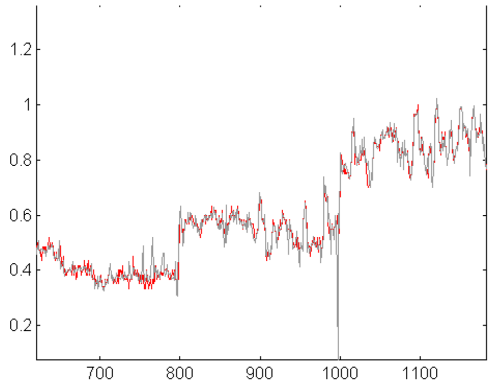


Figure 5.8: Identification of an unlearned sequence of drilling rubber plant's output in the generalization phase

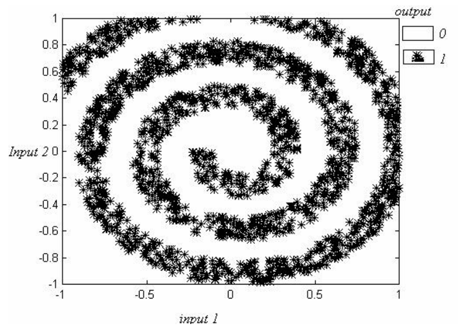


Figure 5.9: Example of Two Spiral problem's database with 1000 patterns to be classified

5.2 Classification

5.2.1 Classification - two spirals problem

Two-spiral problem [CMR02] is an academic classification problem, which is often used as a benchmark. This problem, used for performances comparison, especially in classification problems, is similar to a generalized Exclusive Or. The data is composed of two classes, where decision boundary is a spiral; as presented in figure 5.9.

Construction of neural tree to treat the problem consists of decomposing the input space into a set of subspaces (using Decomposition Units), then performing classification in each subspace by a specialized neural unit (Processing Unit) at leaf's level. The database used for evaluation and validation includes 1000 patterns. The learning was performed on half of the database, while the other half was used for generalization.

T-DTS use here local decomposition evaluation based on decomposition parameter AVStd (as described in 4.2.2).The T-DTS structure on which the validation has been performed includes two kinds of Decomposition Unit's (DU): Competitive Network

(CN) and Self Organization Map (SOM) [Koh82]. Different configurations, concerning number of neurons (for CN) and network's topology (different topologies for SOM as: 2x2, 3x2, 3x3, 4x4 or 5x5), have been implemented. In the case where Kohonen maps have a grid 2x1 topology, T-DTS builds a binary decision tree. The implemented splitting criterion corresponds to the similarity matching based on AVStd, which defines the standard deviation maximum value (in each dimension) in a given sub-database. Concerning Neural Networks based Models (processing units) several possibilities have been implemented: LVQ (Learning Vector Quantization), LN (Linear Neuron), RBF (Radial Basis Functions) and MLP (Multi-Layers Perceptron).

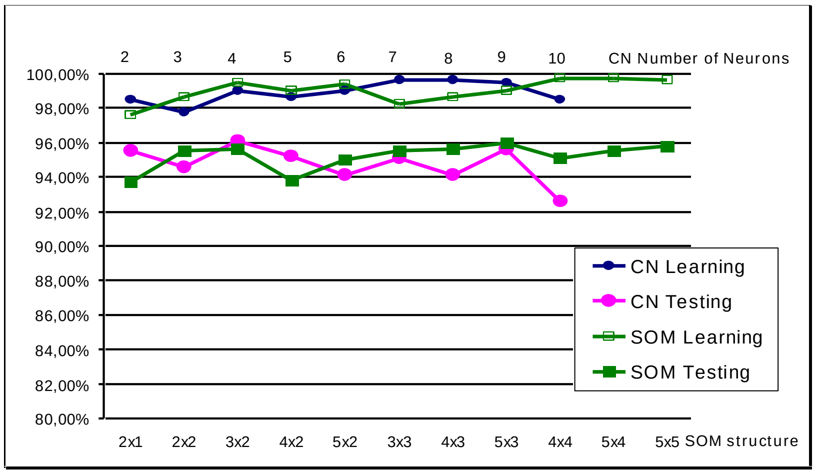


Figure 5.10: Classification Rate as a function of number of neurons (for competitive DU) and as a function of topology (for Kohonen DU)

Figure 5.10 gives a comparative study, expressed as "Correct Classification Rate" as a function of number of neurons (for competitive DU) and as a function of considered topology (for Kohonen DU). Figure 5.11 gives learning and generalization performances, expressed in terms of Classification Rates, according to the maximum standard deviation threshold value's evolution. Table 5.1 compares the processing time

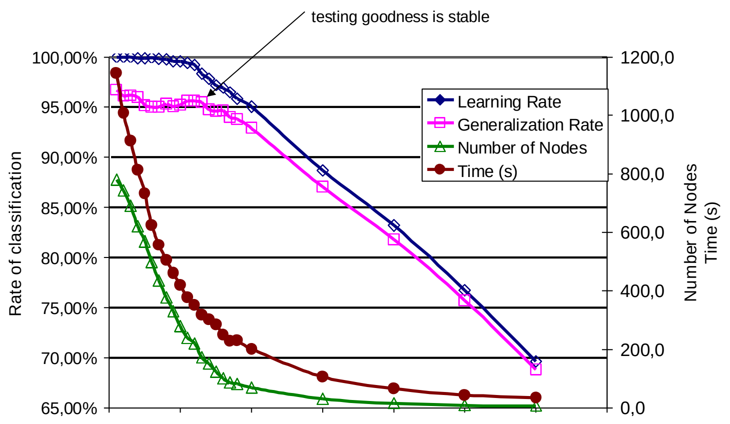


Figure 5.11: Learning and generalization evolution rates vs. AVStd value, when LVQ models are used

for the case of competitive Decomposition based T-DTS and for different Processing Unit structures. For the presented case, AVStd=0.12 leads to 107 sub-databases in the case of a Competitive-like ANN based splitting.

Table 5.1 Processing time for both DU construction and Processing Unit learning and generalization.

Splitting (107 nodes)	Processing Unit Type	Learning Database	Testing Database
159.88 s	LVQ	261.51 s	6.33 s
159.88 s	LN	4.77 s	4.84 s
159.88 s	RBF	37.67 s	5.84 s
159.88 s	MLP	110.91 s	8.46 s

By dividing the initial database into several sub-databases and by dedicated processing of each of those data sub-databases, the proposed ANN based data driven Multiple Model generator (T-DTS) reduces the initial problem's complexity at several levels, especially at processing and modelling ones. De facto, dividing the initial problem into several sub-problems with reduced sizes, on the one hand, simplifies both the learning complexity and duration (learning of relations between inputs and outputs), on the other hand reduces the processing procedure's or unit's complexity. Finally, it decreases globally implementation and parameters optimization constraints.

Results obtained in this benchmark show efficiency of such multiple model structure to enhance processing capability by reducing complexity on both processing and data levels.

In the next section the T-DTS driven by complexity estimation technique will be presented, as well as its advantages over the static decomposition parameter.

5.2.2 Classification - Pattern recognition

In order to use T-DTS in pattern recognition problem [RCM+03b], a data extraction scheme can be used. A feature extraction method, called View-Based approach

[Sae00], [SAEE02], [STA03], [Sae03] is used for feature extracting and coding of printed letters. Encoded data are fed to T-DTS. That entirely composes an original pattern recognition system. T-DTS, embedding problem complexity estimation, decomposes the complex problem into several less complex ones, and builds a set of models for the resulting sub-problems. The method has been applied with success to recognize separated printed letters of various shapes [RCM+03b].

View-Based Approach

The View-Based approach [STA02], [STA03] is based on fact, that for correct character or image recognition one usually need only information about its silhouette or contour. The idea is based on projection method developed and modified by Saeed and Tabedzki [STA02]. The approach is used here with printed letters, although the method has been modified and successfully applied also to handwritten letters [Sae03].

This method involves examining four "views" of a single character. On that ground characteristic vector is allocated, which describes that character. The view is a set of pixels belonging to the contour of a character and having extreme values of one of its coordinates. One can distinguish four views - top, down, left and right (5.12). For example, top view of a letter is a set of points having maximal y coordinate for a given x coordinate.

Next, characteristic points are marked out on the surface of each view, which describes shape of that view. The method of choosing these points and number of them may vary. In experiments seven uniformly distributed points were selected for every view (Fig. 5.13).

The next step is calculation of y coordinates for points on top $(y_{T1}, ..., y_{T7})$ and

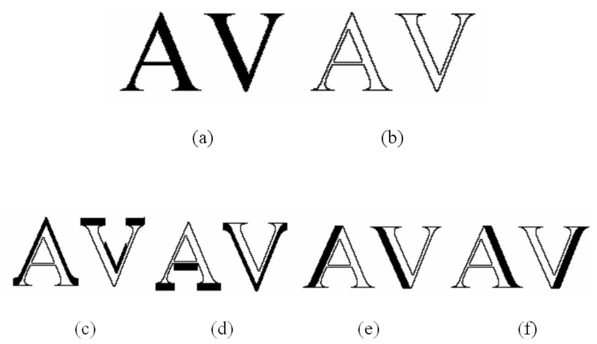


Figure 5.12: Concept of views. (a) Example letters, (b) letters' contour, (c) top view, (d) down view, (e) left view, (f) right view

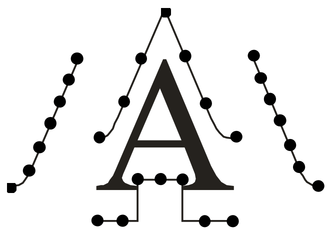


Figure 5.13: Choosing characteristic points for four views

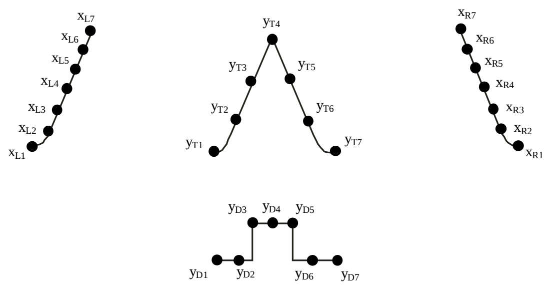
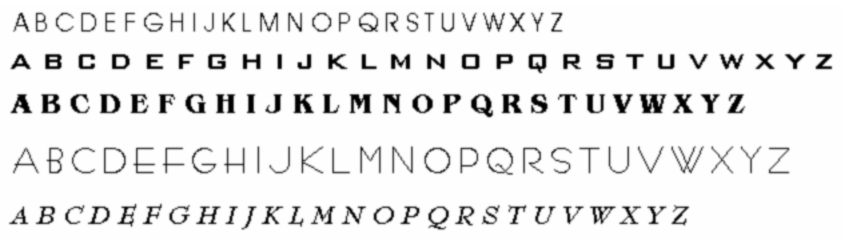


Figure 5.14: Receiving coordinates of characteristic points



ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Figure 5.15: Database sample

down $(y_{D1}, ..., y_{D7})$ views, and x coordinates for points on left $(x_{L1}, ..., x_{L7})$ and right $(x_{R1}, ..., x_{R7})$ views (5.14). These values are normalized, so they are in range $< 0, 1 >$.

Then from these 28 obtained values the characteristic vector is created:

$< y_{T1}, ..., y_{T7}, y_{D1}, ..., y_{D7}, x_{L1}, ..., x_{L7}, x_{R1}, ..., x_{R7} >$,

which describes given letter, and is a base for further analysis and classification.

To validate the approach, a database of letters were used. It consisted of 26 capital letters of Latin alphabet written in 66 different styles (fonts). That gives a total of 1716 cases from 26 classes. The data is stored as images without noise. The data varies significantly within-class by shape, slope, size (thus image resolution), some fonts are bold and others have sheriff lines. That increases the difficulty of classification task. A sample of the database is shown in the figure 5.15.

The images have been encoded using View-Based approach. The views of each letter have been sampled and normalized. After that process, letter image is represented as vector of 28 features. This is the data format fetched to T-DTS.

T-DTS has been learnt on randomly picked 50% of the total data, called Learning Set. For decomposition decision, the modified Fisher discriminant ratio was used. As decomposition algorithm, the Competitive Network composed of two neurons was used, which means splitting a set in two sub-databases each time, what leads to binary tree structure. One of T-DTS decomposition trees obtained during the experiments is depicted in the figure 5.16.

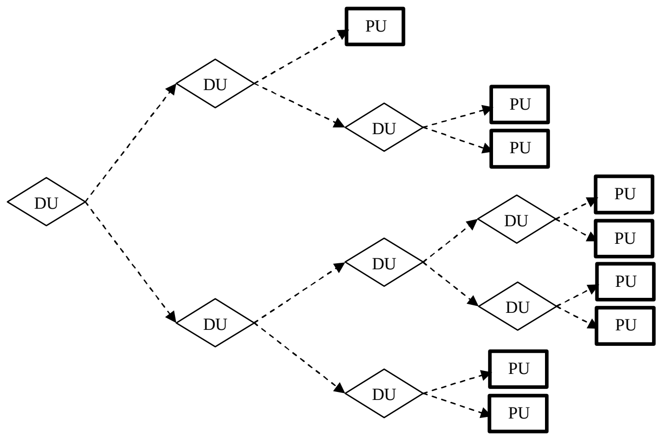


Figure 5.16: T-DTS decomposition tree obtained during experiments

After decomposition, a set of models were created, using Probabilistic Neural Networks [Was93] as classifiers. For each Processing Unit, a Probabilistic Neural Network is built, which allowed us to achieve approximately 90% of correct classification total and 80% of correct classification on the unseen data.

The misclassifications have arisen probably from class ambiguity - similarity of samples from different classes. This error is unavoidable on the data processing level and is also referred as Bayes error (see chapter 3.3 for more information). On the feature extraction level it is sometimes possible to avoid that effect by unambiguous feature encoding. Another argument for that assumption is occurrence of misclassifications between distant (by intuition) classes. This is probably caused by great variability of the letters in general. It implies that Bayes error is significant [Ho00], [Pie98], [Sin03]. Considering this, a way to increase the performance of system is to decrease class ambiguity by using preprocessing techniques like thinning and slope correction.

The T-DTS with View-based data extraction is a hybrid pattern recognition technique. The performance of the hybrid intelligent system has been validated on the varied non-homogenous database of patterns, what resulted in successfully recognized 80% of separated printed capital letters of various shapes.

Further work in the area will be concentrated on preprocessing of the images. Especially thinning and slope correction [CL96] should be useful in decreasing within-class variability, which should result in better classification performance. Concerning classification performance, there are two directions: one is enhancement of View-Based feature extraction algorithm, second T-DTS capabilities improvement.

5.2.3 Classification with complexity estimation

In the example that follows T-DTS is studied as an auto-adapting system, i.e. system which structure adapts to difficulty of data [RCM03a]. The goal here is to adjust the decomposition according to the complication of task. This is done by usingclassification complexity estimation methods.

The goal is to examine the adapting of the system to data of various difficulties.

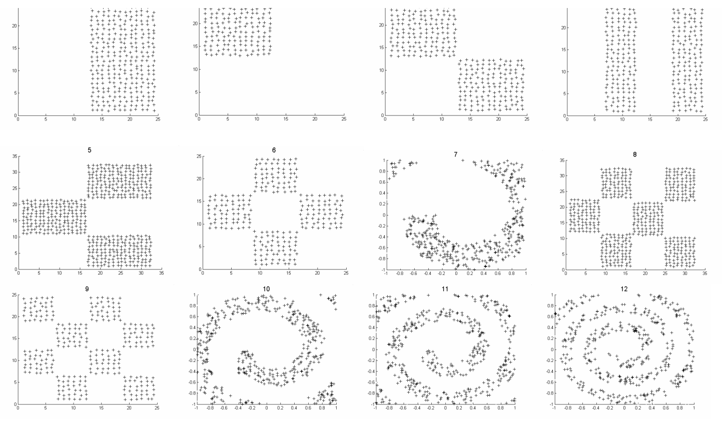


Figure 5.17: Sequence of datasets of increasing complexity

Thus a sequence of simple datasets of intuitively increasing complexity has been created. They are presented in the figure 5.17. The sequence contains both linearly separable datasets and non-linear separable datasets (spirals of different shape). For linearly separable datasets the complexity increases with fragmentation of clusters. For non-linearly separable data the complexity increases as the clusters are twisted more and more around the centre. The idea here was to make the data obviously different in complexity and observe the behaviour of the T-DTS (driven by complexity estimator). The behaviour could be verified by comparing with intuitive complexities sequence.

T-DTS were used in two cases to validate the assumptions about the increasing difficulty of the benchmark problems. In first case, the decomposition tree is composed of quasi-static structure - T-DTS generates here approximately the same tree structure and number of Processing Units. AVStd measure was used here as decomposition criterion to achieve similar degree of decomposition. In the second case, a classification complexity estimator was used in order to accommodate the

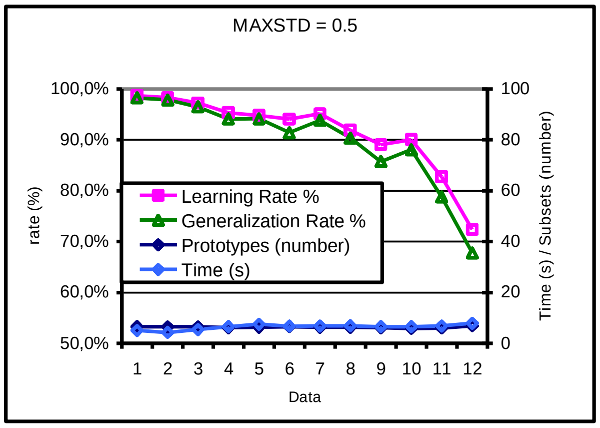


Figure 5.18: Statistics of computational effort for T-DTS without adaptation

decomposition structure to case difficulty.

On the figure 5.18 there are results for static decomposition structure. classification rates drop significantly for more complicated datasets (going to right). Processing times are approximately the same for each dataset. This is due to fact that the decomposition tree structures are practically identical and they are too simple for more difficult datasets.

When T-DTS adapts to classification complexity, Fisher discriminant ratio were used as decomposition threshold. Fisher decomposition ratio measures the separability of different class clusters. The results are depicted on figure 5.19.

One can notice in figure 5.19, that classification rates for learning phase are alike, and for generalization rates there is only small dropping tendency. Number of prototypes (related to processing time) significantly increases for more complex datasets. This fact is in contrast with the experiments with static structure presented on figure 5.18. Thus T-DTS structure with complexity estimation adapts to difficulty of dataset, creating tree structure of size proportional to data complication.

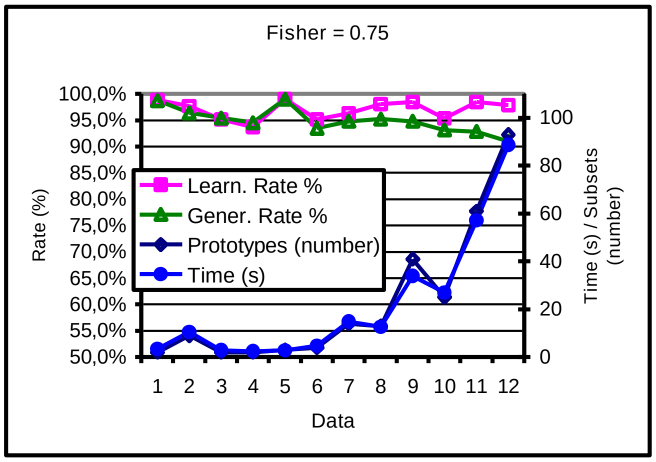


Figure 5.19: Statistics of computational effort for T-DTS adapting to problem difficulty

5.3 Applications - Conclusion

This chapter has presented T-DTS in different classes of applications: system identification, classification and Pattern Recognition. The example applications shown here present good performance in terms of processing quality in all presented tasks.

Generally speaking, solutions based on modular structure are universal, as it is easy to replace modules and thus modify part of system. This is also the case for T-DTS, where not only processing modules can be replaced this way, as well as data decomposition (clustering) modules and knowledge acquisition modules (complexity estimators and other techniques that are used to optimize the structure and parameters). This proves universality of T-DTS approach up to some point.

The application of T-DTS presented in this chapter are of various origin. First we present academic classification problem to show the benefices of decomposition when the problem overwhelms processing model used. This shows the flexibility of our approach, where processing is limited only by the nature of Processing Units.

The limitation is surely data decomposition issue, because not all problems are well suited for decomposition. Task decomposition insures the local processing the may be unsuitable for problems that need global processing. Existence of processing algorithms that from their nature already contain local processing (RBF) also makes Task Decomposition less desirable. Even for these problems T-DTS still offer benefices resulting from simplification of local models and breaking up the problem weight from one model into several. This could be valuable even for such local approximation Processing Units.

Next chapter is a conclusion of all work contained in the thesis.

CONCLUSIONS

Main idea of T-DTS is to take advantage from distributed processing and task simplification. Another goal is to decrease the amount of user intervention in specifying processing parameters, by using statistical techniques of Complexity Estimation. When dealing with classification, identification or regression task T-DTS decomposes the problem using Decomposition Units and thus creates decomposition tree. The decomposition tree is used to decompose data into sub-databases. Neural Network Models are then used to process resulting data sub-databases. The processing phase can be done using distributed parallel processing.

The efficiency of T-DTS design was shown in academic classification problem of Two-Spirals classification. In the next example classification complexity method was incorporated into T-DTS to aid the decomposition, what allows the system to be user-independent and adapt to classification task difficulty automatically. System identification task was studied with two examples. The first was academic system identification problem, the second real complex industrial problem. In both cases T-DTS decomposed the data to reduce processing complexity and build a set of models to identify the partial signals of data sub-databases. In both cases T-DTS was proven to build faithful models. Finally T-DTS was linked together with feature extraction technique (View-Based Approach) to build a pattern classification system. That combination allowed achieving 80% recognition rate of non-homogenous and rather difficult letters set. In order to summarize: T-DTS were tested in classification, Pattern Recognition and system identification area. Solutions based on modular structure are generally universal, as it is easy to replace modules and thus modify some part of system. T-DTS processing modules can be replaced this way, as well as data decomposition modules and knowledge acquisition modules. This proves universality of T-DTS approach to some point.

One may criticize the T-DTS approach as unsuitable for the tasks that are solved well with classical methods (of any kind). There is however always a possibility of incorporating such classical method as a processing tool in T-DTS and allow task decomposition that in such case may prove itself useful by decreasing the processing time with parallel processing. It is also frequent and almost natural that a solution individually designed for a specific problem will outperform solution that is universal and can serve many purposes (not just this particular problem). The power of universal solution lies though in widespread area of applications, so one doesn't need to design a new tool that is individually suited for the problem but modify and use a universal tool that is already created and ready to go.

T-DTS is implemented in Matlab 6.1 language with addition use of GUIDE environment to create user interface. Matlab environment gives easy access to library of useful functions. GUIDE environment although rather simple comparing to other GUI environments is sufficient to build a system of academic use.

The user-independent and automated processing of data is a very distant goal. By linking together statistical methods and universal data processors as Artificial Neural Networks one can expect to make that objective a little closer. T-DTS was intended to be a small step in that direction. The further development of the system could be in two main directions. The first is development of task processing tools library in order to process efficiently many tasks of different nature. The second is connected to identify nature of processing task in order to choose most appropriate tool from tools library.

APPENDIXES

Appendix A

Computer T-DTS Implementation

This appendix will present details of T-DTS paradigm computer implementation.

T-DTS is currently implemented in Matlab 6.1 language, using GUIDE user interface. Figure A.1 gives overall view of main application window.

One can see groups of controls related to input parameters, decomposition criterion, classification complexity, experiment run and output presentation.

A.1 Structure of functions call of main frame

Figure A.2 presents dynamic structure of most important T-DTS functions.

The functions will be consequently described.

Graphical user Interface (GUI) **function** This function creates a window (presented in figure A.2), which presents graphically most used parameters and allows fast modifications and experiment conduction with presentation of results. It's encoded in file named *aaa.mat*. The code here was implemented with support of

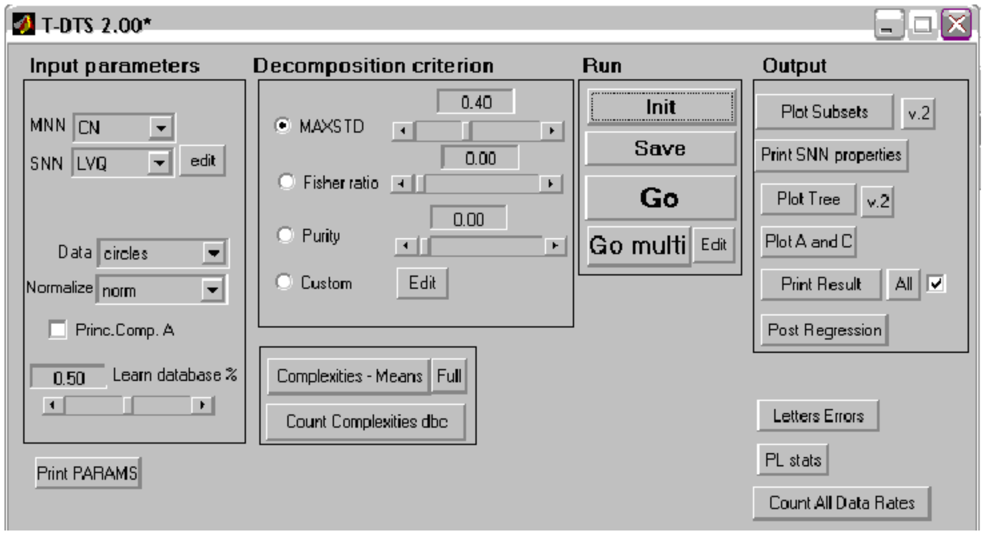


Figure A.1: T-DTS implementation

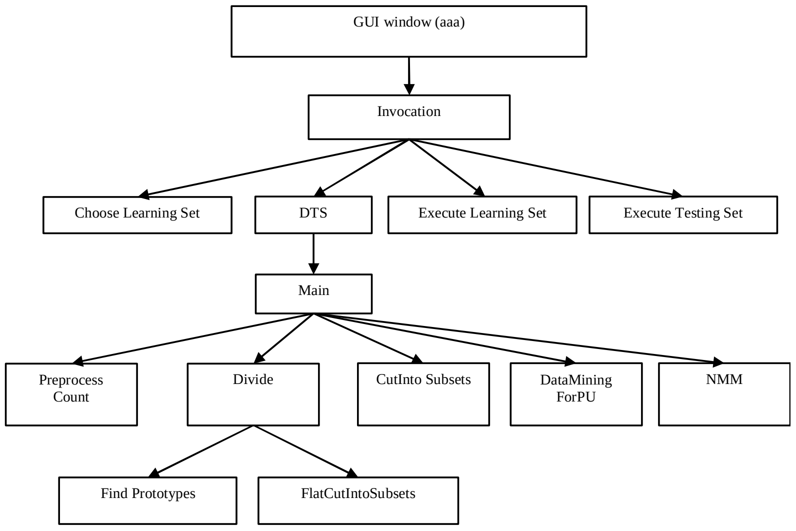


Figure A.2: Static functions structure

GUIDE (Graphical User Interface Development Environment) incorporated in Matlab. Button callback functions in the file call functions related to T-DTS experiment execution as well as presentations of data and results.

***Invocation* function** The function is a wrapper for any T-DTS experiment, loads the data, chooses the learning set (if necessary), calls T-DTS structure building and learning, finally obtains the results for learning and generalization set.

***Choose Learning Set* function** This function is responsible for random dividing the dataset fed to system into learning set and generalization set. It could be based on the learning database percentage ratio parameter called Percentage.

***DTS* function** The function invokes all task connected with. It calls Main which creates and performs learning of T-DTS structure. ***Execute Learning Set* function** That function uses T-DTS processing structure created by DTS function to process learning data. ***Execute Testing Set* function** That function uses T-DTS processing structure created by DTS function to process generalization data.

***Main* function** It creates and performs learning of T-DTS structure. That includes pre-processing (pre-process count), recurrent dividing of the sub-databases by DU (divide), final dividing of the learning set according to dividing prototypes structure created by divide, choosing of the Processing Unit(Data mining For Processing Unit) and creation of Processing Unit models for each prototype (sub-database) of the learning set.

***Preprocess Count* function** This function pre-processes the incoming data, what can include normalization and reduction of input's dimensionality by Principal Component Analysis (PCA).

Divide **function** The function performs dividing of given dataset into sub-databases. It is called recurrently. It works by finding prototypes by using Decomposition Units, which perform unsupervised learning and dividing the set into sub-databases.

Cut Into Subsets **function** The function finally divides the learning set according to dividing prototypes structure created by divide.

Data Mining For Processing Unit **function** The function chooses the structure and parameters of Processing Unit to process the specific data sub-database.

Processing Unit **function** The function performs learning of Processing Unit for the sub-database data.

Find Prototype **function** This function uses Decomposition Unit with unsupervised competitive neural network structures to find prototypes for given sub-database of data.

Flat Cut Into Subsets **function** This function uses prototypes to divide given set of data into sub-databases.

A.2 User interface

As marked in the figure there are five groups of controls. There will be subsequently explicated.

T-DTS input parameters These concern T-DTS algorithm parameters as well as database preprocessing.

DU choosing combo box allows selection of DU decomposition algorithm. One can choose either Competitive Network (CN) or Kohonen Self Organizing Map (SOM).

Some detailed parameters (like structure of the Map or number of neurons for competitive network are available only on programming level.

Using Processing Unit choosing combo box one can specify the type of ANN processing Model at leaf level of the decomposition tree. The possible choices are: Learning Vector Quantization (LVQ), Linear Neuron (LN), Radial Basis Function (RBF), Multi-Layer Perceptron with Levenberg-Marquadt learning (MLP_LM), Probabilistic Neural Network (PNN), Generalized Regression Neural Network (GRNN) [Arb89].

By using data combo box, one can choose one of databases to process or choose empty one and load it manually from Matlab command line (using symbol '*').

'Normalize' combo box allows choosing if and what type of normalization the system will apply to database. Available types are: 'norm' - Normalization to achieve Gaussian distribution in each data dimension separately; 'ones' - normalization to compartment ¡-1,1¿; and finally 'none' when no data normalization is required.

By checking the box marked as 'Princ. Comp. A' one can order Principal Component Analysis on the data before feeding it to further processing. The process is expected to eliminate redundancy in data and reduce the number of dimensions.

Slider and edit box marked 'Percent. to learn' allows to choose the amount of database available to learning algorithm.

Finally 'Print params' button displays all learning parameters in the Matlab command window.

Decomposition criterion controls The controls in the box allow choosing and setting value for the decomposition criterion. 'AVStd' parameter is a decomposition parameter described in chapter 4.2.2. 'Fisher ratio' is a threshold based on Fisher discriminant ratio (classification complexity estimator described widely in chapter 3.5.4.2). Purity parameter is a threshold based on purity classification complexity estimator described in 3.5.3.2.

Output presentation methods These controls are designed to present the results obtained by T-DTS.

'Plot sub-databases' shows first two dimensions of data and its decomposition by colour marking the data sub-databases. The decomposition is marked separately for the learning and generalization data. The button 'v.2' on the right draws the same without colour using different symbols.

Button 'Print Processing Unit properties' outputs in Matlab command window structure and parameters of Processing Unit models created during learning.

'Plot tree' shows first two dimensions of data, draws the data and shows the process of decomposition tree creation in time. The button 'v.2' on the right draws the same with alternate colours.

'Plot A and C' is used only in signal processing task and shows the original desired signal in comparison with signal outputted by the system.

'Print result' and 'All' allows outputting consequently last and all experiment results obtained by T-DTS.

'Post regression' button shows regression analysis of output signal and desired signal dependence, to verify the performance.

Classification Complexity computation for a set of databases This buttons allow computation of classification complexity for a set of databases.

'Complexities -means' shows computed complexities for a set of databases.

'Count complexities dbc' performs the set of T-DTS experiments on structure containing a set of databases in order to compare the results of multiple databases for the same T-DTS configuration (contained in cell array variable dbc).

Launching buttons Launching buttons allow initialization ('Init'), saving the T-DTS configuration data ('Save') and starting of experiment ('Go'). To perform repeated experiments one can check the box marked as 'Multiple Times'. Unless the mentioned box is checked system will make only a single experiment.

A.3 T-DTS Controlling Parameters structure

T-DTS parameters are stored in PARAMS structure available at command line level and stored in file paramsfile.mat. Most important parameters are available visually from GUI level as controls (as described in A.2). Here follow all the parameters with explanation:

PARAMS =

Compm: [1x1 struct]

Method: 'AVStd'

Purity: 0.2000

Fisher: 10

AVStd: 0.2000

Preprocess: [1x1 struct]

Normalization: 'ones'

PrinComp: [1x1 struct]

Percentage: 0.3000

nPar: [1x1 struct]

Processing Unit: 'LVQ'

DU: 'CN'

Dataname: 'spirals'

GoAll: 0

Clustering: 'independent'

IntOut: 1

ConsPro: 0

AVStd: 0.4000

Percentage: 0.5000

Showdisplay: 0

MaxClusters: 1000

DistanceFunction: 'dist'

Display: [1x1 struct]

PlotControlPar: 1

DecU: 'normal'

Compm - holds decomposition complexity method currently selected and thresholds for all complexity methods available.

Pre-process - contains preprocessing related parameters: selected normalization method, Principal Component Analysis related parameters, default percentage of database taken for learning, normalization shift and magnifier are stored in *nPar* structure.

Processing Unit- type of ANN Method selected to process (model) all leaf level sub-databases. DU - specifies type of Decomposition Unit selected to perform decomposition of dataset

Dataname - contains name of frequently used at file which will be load to do experiment. If that field contains '*' no file will be load and the data will be taken from command line workspace instead.

GoAll - the Boolean value will specify if the experiments will be performed by all Processing Units available to compare the efficiency of modelling.

Clustering - defines clustering strategy - independent means that final clustering is done with all prototypes taken together independently of tree structure after dividing, while 'treelike' means that final clustering is done simultaneously with dividing.

IntOut - parameter specifying the output format of processing task, for regression Processing Units it's sometimes necessary to convert floating point numbers to integer ones.

ConsPro - the parameter specifies if the prototypes are to be taken from previous experiments. That allows to speed up processing by skipping the decomposition phase and doing only modelling phase.

AVStd, Percentage - ancient and obsolete parameters, moved to branches of the structure.

Showdisplay - the parameter allows skipping some displaying of computation in case one doesn't want to watch them, it speeds the processing.

MaxClusters - specifies maximum number of clusters allowed to system, it's a safety valve if clustering algorithm went out of control.

DistanceFunction - specifies distance function used during clustering phase by DUs, possible values are: 'dist' - Euclidean distance, "streetdist' - Manhattan (street) distance. Another distance functions can be added by editing the code in CutIntoSubdatabases.

Display - parameters related with display, PlotControlPar - parameter used in PlotControl function to decide if the sub-databases point representation should be coloured or should differ with shape.

DecU - parameter allowing restructuring the decomposition tree, possible values: 'Normal', or 'restructure'.

Appendix B

History of ANN

This appendix will present in short the history of Artificial Neural Networks.

The field of ANN has a history of about 60 years. However ANN have found solid application only in past 20 years. This field is still developing rapidly. Due to their relative short age and rapid development the terminology of ANN area is not entirely clear like other fields i.e. optimization and control systems, where the terminology, design procedures and mathematical basics are known and applied for many years.

First approach in modern era was a paper of McCulloch and Pitts in 1943, which describes a logical calculus of neural networks that unifies neurophysiology and mathematical logic. They described a network composed of all-or-none neurons and have proved that such structure can, in principle, compute any computable function.

In 1949 Hebb published his book "The organization of behavior", where physiological learning rule of synaptic modification was presented. Hebb states that the brain connectivity is continually changing, as organism learns different function tasks, and neural assemblies are created by such changes. Hebb introduces a postulate of learning, which declared that the effectiveness of variable synapse between two neurons is increased by the repeated activation (charges or data flow) across that synapse. The paper of Rochester, Holland, Haibt and Duda from 1956 [RHHD56] describes a computer simulation which tests with success Hebb's postulate of learning.

In 1950 Taylor initiated work on the associative memory. This was followed by the introduction of learning matrix by Steinbuch in 1961; this matrix consists of a planar network of switches interposed between arrays or "sensory" receptors and "motor" effectors. In 1972 Anderson, Kohonen and Nakano independently introduced the idea of a correlation matrix memory based on the outer product learning rule.

In 1954 Gabor proposed an idea of nonlinear adaptive filters. He has built such a machine, where learning were accomplished by feeding samples from stochastic process into machine, together with the target function, that the machine was expected to produce.

Very important approach to the pattern recognition area was the work of Rosenblatt on the supervised learning method called perceptron. In 1960 Widrow and Hoff introduced least mean square (LMS) algorithm and used it to formulate Adaline (adaptive linear element). It was expanded in 1962 by Widrow and his students, who introduced Madaline (multiple-adaline), which is one of the earliest trainable layered neural networks with multiple trainable elements.

In the 70'tees, von der Malsburg as first demonstrated the idea of self-organizing maps using competitive networks. In 1976 von der Malsburg and Willshaw published a paper on the formulation of self-organizing maps, motivated by topologically ordered maps in the brain. The work was continued with success by Kohonen in 1982, who introduce one or two-dimensional lattice structure.

Grossberg building on his previous work on the self-organizing networks established a new principle of self-organization known as adaptive resonance theory (ART). The theory involves a bottom-up recognition layer and a top-down generative layer. If the input pattern and learned feedback pattern match, a dynamical state called "adaptive resonance" (amplification and prolongation of neural activity) takes place. This formulates the principle of forward/backward projections.

In 1975, Little and Shaw described a probabilistic model of a neuron and used the model to create a theory of short-term memory.

In 1982, Hopfield basing on the idea of an energy function has formulated a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections. He created a connection between such recurrent networks called Hopfield networks and an Ising model used in statistical physics.

In 1983, Kirkpatrick, Gelatt and Vecchi described a new procedure called simulated annealing, useful for solving combinatorial optimization problems. Simulated annealing is rooted in statistical mechanics. It is based on a relatively simple technique used first in computer simulation by Metropolis et al. (1953). The idea was used later in the work of Ackley, Hinton and Seynowski (1985) in the development of a stochastic machine called Boltzmann machine, which was the first successful realization of multilayered neural network. The Boltzmann machine was later used for subsequent development of sigmoid belief networks by Neal.

In 1986 Rumelhart, Hinton and Williams reported a development of a backpropagation algorithm. In fact, the Backpropagation algorithm was discovered independently in two other places about the same time (Parker, 1985; LeCun, 1985). After the discovery of the back-propagation algorithm in the mid-1980s, it turned out that the algorithm was described earlier by Werbos in his PhD thesis at Harvard University in August 1974.

In 1988 Broomhead and Lowe described a design procedure of layered feedforward networks using Radial Basis Functions (RBF). The idea of RBF is based on method of potential functions, originally proposed by Bashkirov, Braverman and Muchnik in 1964 and developed by Aizerman, Braverman, and Rozonoer [ABR64].

In 1989 a book of Mead entitled ”Analog VLSI and Neural Systems” provided a mix of concepts from neurobiology and VLSI technology. Most significant ideas contained there are silicon retina and silicon cochlea.

In 1990 Vapnik and co-workers introduced a powerful class of supervised learning networks called Support Vector Machines (SVM). SVM can be used to solve pattern recognition, regression and density estimation problems. This method is based on the results in theory of learning with finite samples sizes. SVM in natural way incorporated an idea of Vapnik-Chervonenkis (VC) dimension in their design. The VC dimension provides a measure for the capacity of neural network to learn from a set of examples.

Appendix C

Biological neuron and its artificial models

Brain is composed of about 10 billion (10^{10}) neurons. The biological neurons are inter-connected cells, processing and transmitting information in parallel way. A simplified model of biological neuron is presented in figure C.1.

"A neuron's dendritic tree is connected to a thousand neighboring neurons. When one of those neurons fires, a positive or negative charge is received by one of the dendrites. The strengths of all the received charges are added together through the processes of spatial and temporal summation. Spatial summation occurs when several weak signals are converted into a single large one, while temporal summation converts a rapid series of weak pulses from one source into one large signal. The aggregate input is then passed to the soma (cell body). The soma and the enclosed nucleus don't play a significant role in the processing of incoming and outgoing data. Their primary function is to perform the continuous maintenance required to keep the neuron functional. The part of the soma that does concern itself with the signal

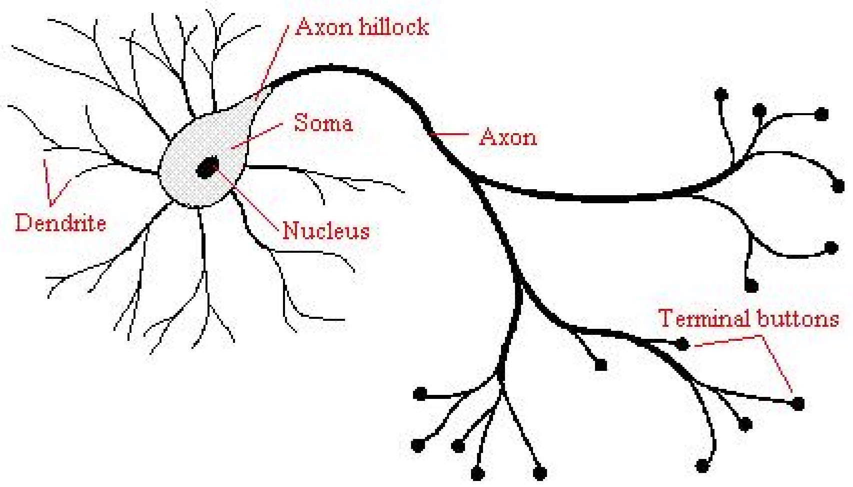


Figure C.1: Biological neuron

is the axon hillock. If the aggregate input is greater than the axon hillock's threshold value, then the neuron fires, and an output signal is transmitted down the axon. The strength of the output is constant, regardless of whether the input was just above the threshold, or a hundred times as great. The output strength is unaffected by the many divisions in the axon; it reaches each terminal button with the same intensity it had at the axon hillock. This uniformity is critical in an analogue device such as a brain, where small errors can snowball, and where error correction is more difficult than in a digital system. Each terminal button is connected to other neurons across a small gap called a synapse [left]. The physical and neurochemical characteristics of each synapse determine the strength and polarity of the new input signal. This is where the brain is the most flexible, and the most vulnerable. Changing the constitution of various neuro-transmitter chemicals can increase or decrease the amount of stimulation that the firing axon imparts on the neighboring dendrite. Altering the neurotransmitters can also change whether the stimulation is excitatory or inhibitory." [CS92]

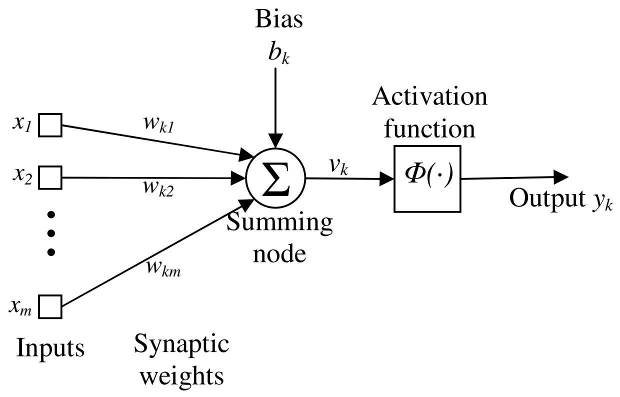


Figure C.2: Artificial neuron example

Artificial neuron inherits many biological neuron properties. There exist many artificial neuron models, the properties in common are distributed processing and high connectivity of elements. In this section the most popular and general neuron model will be presented. General artificial neuron model is depicted on the figure C.2.

This model is similar to biological neuron: The inputs x (electrical charges in biological neuron) are weighted by synaptic weights w_{kj} (properties of the dendrites) and summed creating linear combiner output u_k.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{C.0.1}$$

Summing node (soma) adds linear combiner input and bias bk what results in induced local field v_k (also known as activation potential).

$$v_k = u_k + b_k \tag{C.0.2}$$

Induced local field is modified by transfer function $f(\cdot)$ producing output y_k and propagated through output (axon) to other neurons.

$$y_k = \varphi(v_k) = \varphi(u_k + b_k) = \varphi(\sum_{i=1}^{m} w_{kj} x_j + b) \tag{C.0.3}$$

The model is a greatly simplified version of the mechanism created by nature, although it shows great capabilities and universality. Artificial neuron output value a is a function (called transfer function) of weighted input values $p_1..p_k$ and bias b. In the biological neuron one can find also output values called axons and inputs called dendrons. In fact, transfer function in biological neuron is much more complicated than in artificial one. Biological neural networks show great complexity and functionality not even approximately met by their artificial descendant. Most known transfer functions used with the model are:

- **Piecewise-linear function**

  $$\varphi(v) = \begin{cases} 1, & v \geq \frac{1}{2} \\ v, & \frac{1}{2} > v > -\frac{1}{2} \\ -1, & v \leq -\frac{1}{2} \end{cases}$$

  This transfer function can be seen as approximation of non-linear amplifier. The amplification factor in the linear region is here equal to 1. By modification of the amplification one can achieve many different functions in particular:

  - **signum function**, when linear region length is reduced to 0 by setting amplifier factor to plus infinity;
  - **threshold function**, similar shape to signum function, just with different values;

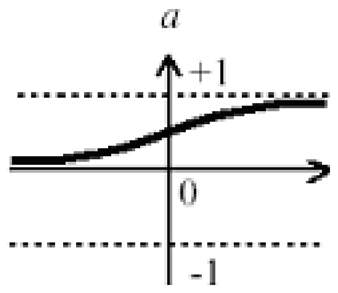


Figure C.3: Sigmoid (logistic) transfer function

– **linear combiner function**,when linear region is maintained in the whole area.

- **Sigmoid function**

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{C.0.4}$$

This is the most common transfer function, which exhibits a balance between linear and nonlinear behavior. By modifying the parameter a one can achieve different slopes of the function. Sigmoid function is differentiable, which is an important feature, as it allows achieving efficient learning algorithm optimizations used in the learning session.

- **hyperbolic tangent function**

$$\varphi(v) = \tanh(v) \tag{C.0.5}$$

Hyperbolic tangent function is similar to sigmoid function, but it is antisymmetric and its values are ranged in $[-1, 1]$, which sometimes shows analytical

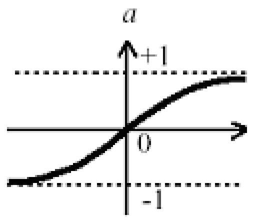


Figure C.4: Hyperbolic tangent transfer function

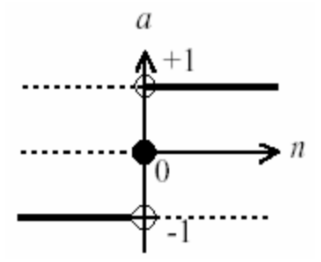


Figure C.5: Signum transfer function

benefits. Next section will show how the artificial neurons are linked together to create more powerful structures.

C.1 Signum function

$$\varphi(v) = \begin{cases} -1 & v < 0 \\ 0 & v = 0 \\ 1 & v > 0 \end{cases} \tag{C.1.1}$$

It is sometimes desirable to have output values from range [-1, 1]. The signum function is an odd function (antisymmetric).

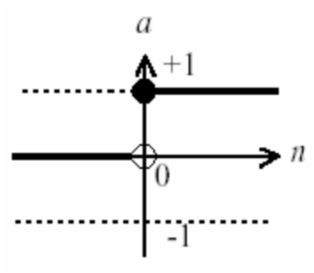


Figure C.6: Threshold transfer function

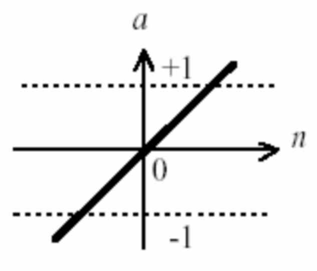


Figure C.7: Linear combiner transfer function

C.2 Threshold (Heaviside) function

$$\varphi(v) = \begin{cases} 0 & v < 0 \\ 1 & v \geq 0 \end{cases} \tag{C.2.1}$$

Threshold function is also referred in literature as linear threshold gate. Neuron with threshold transfer function is known also as McCulloch-Pitts model [Roj96]. Behavior of this model is described as all-or-none because it produces either 0 or 1. Slight modification of this function is the signum transfer function.

C.3 Linear combiner function

$$\varphi(v) = v \tag{C.3.1}$$

C.4 Softmax transfer function

Softmax transfer function is a a normalized exponential. Softmax transfer function for a input vector a_i gives the output vector y_k given by:

$$y_k = \frac{\exp[a_k]}{\sum_{i=1}^{K} \exp[a_i]} \tag{C.4.1}$$

Softmax transfer function has following properties:

$$0 \leq y_i \leq 1, \text{for all} i \tag{C.4.2}$$

$$\sum_{i=1}^{K} y_i = 1 \tag{C.4.3}$$

Appendix D

Artificial Neural Networks' structures

In this appendix some of the most popular neural network architectures are presented.

D.1 Single-layer perceptron

Perceptron is one of the simplest, oldest and most known ANN structure. [HDB1996] Rosenblatt created many variations of the perceptron [Ros61]. One of the simplest was a single layer network which weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector.

Perceptron neuron uses a threshold transfer function:

$$y = \varphi(\sum_{i=1}^{m} w_i x_i + b) = \varphi(v) \quad \varphi(v) = \begin{cases} 0 & v < 0 \\ 1 & v \geq 0 \end{cases} \tag{D.1.1}$$

Perceptron output is either 0 or 1, as it contains threshold transfer function $\varphi(\cdot)$

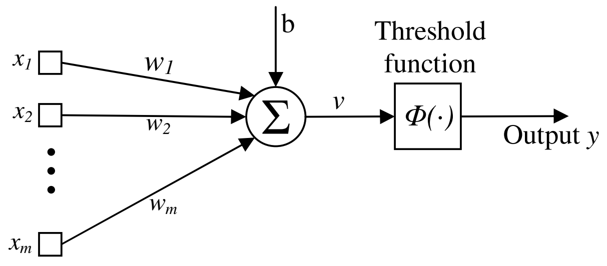


Figure D.1: Perceptron neuron

(see section C). Thus, its activity can be seen as transformation from k-dimensional surface of inputs into output values $< 0; 1 >$. Perceptron boundary is the so-called hyperplane. Perceptron hyperplane can be seen here as a plane in k-dimensional space, which classifies all the points in space as zeros or ones. Simple example of hyperplane in two-dimensional space is depicted in the figure D.2.

Position and orientation of classifying surface depends on the perceptron input weights and bias. It is changed by training technique called the perceptron learning rule:

$$\begin{aligned} W' &= W + ex^T \\ b' &= b + e \\ e &= t - y \end{aligned} \tag{D.1.2}$$

where W is the perceptron weights matrix, x is a single input vector, b is the perceptron bias value and e is the error of classification for specified input vector (difference between desired target value t and real output value y).

Single perceptron neuron use is limited to so-called linearly separable problems.

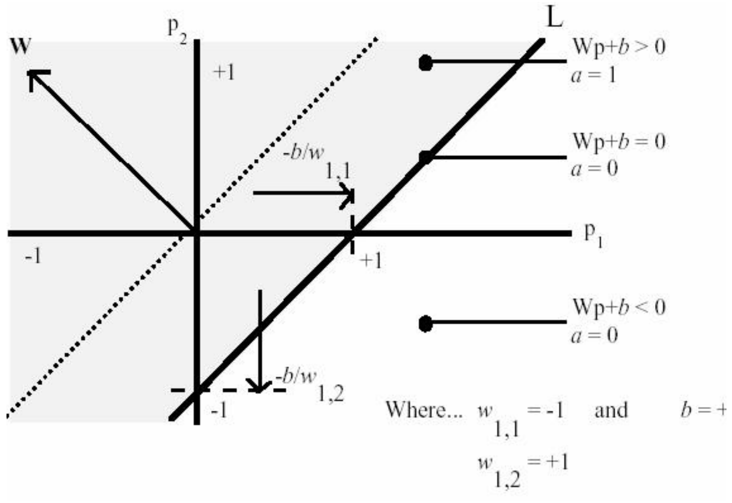


Figure D.2: Perceptron hyperplane

It means that it should be possible to separates the input values originating from different classes by hyperplane. In the other case, perceptron will not be able to classify them correctly. Another limitation of perceptron use is that it can produce only two values: zero and one. These limitations can be overcome by using layer of perceptron neurons. For example two perceptrons can take values $(0;0), (0;1), (1;0), (1;1)$, which in fact is equivalent to four classes. However single layer of perceptron neurons is still able to solve only linearly separable problems. To overcome this limitation a multiple layers of perceptron neurons are needed.

D.2 Linear networks

Linear networks are similar to perceptron, but instead of hard-limit transfer function they use linear transfer function. That allows their outputs to take any value, while Perceptron output is limited to values 0, 1. However used as classifiers they can model

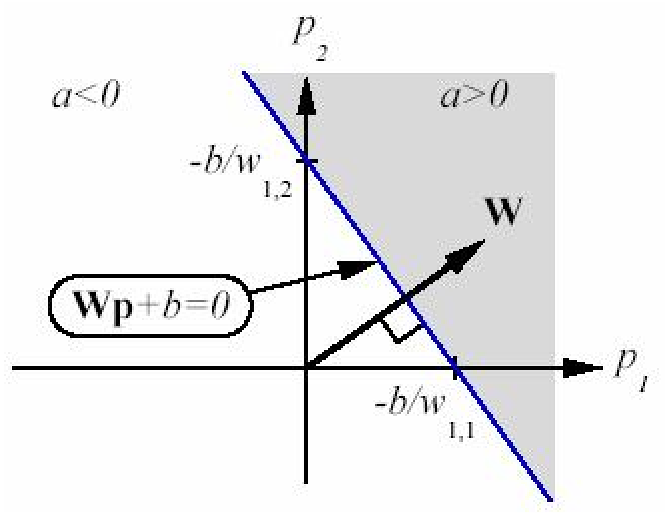


Figure D.3: Linear network example

only linearly separable problems (like perceptrons). They can be however used for regression i.e. to produce continuous output from compartment $< -\infty; +\infty >$.

Example of linear network hyperplane in two-dimensional space is given in the D.3.

Line corresponds to network output equal to zero, upper right grey area represents values greater than zero and lower left white area represent values lower than zero. If presented in three-dimensional space then linear network values form a surface.

Very popular algorithm used to train linear network is called the **Least Mean Square Error (LMS)**. It is known well in optimalization area. Learning rule is provided with a set of k examples of desired network behavior:

$$p_1, t_1, p_2, t_2, ..., p_k, t_k \qquad (D.2.1)$$

Here p_i is an input to network, t_i is corresponding target output. The error is calculated as the difference between real output and target output. The goal is to minimize average squared sum of errors (called also $L2$ metric):

$$m_{se} = \frac{1}{k}\sum_{i=1}^{k} e_i^2 = \frac{1}{k}\sum_{i=1}^{k}(t_i - a_i)^2 \tag{D.2.2}$$

a_i is the current response of the network to input vector x_i. The LMS algorithm adjusts the weights and biases of the linear network so as to minimize this mean square error function.

The mean square error is a quadratic function, thus it will have either one global minimum, weak minimum or no minimum, depending on the characteristics of the input vectors.

D.3 Multi-Layer Perceptron

Multi-Layer Perceptron is one of the most popular structures for general applications like pattern recognition. MLP are multilayered feedforward networks, which consist of input layer (sensors), one or more hidden layers and an output layer.

MLP has three distinctive characteristics:

1. Neurons in the network have nonlinear transition functions. The important property of the function is that the nonlinearity is smooth (the function is differentiable everywhere). Commonly used function is sigmoid nonlinearity defined by the logistic function:

$$\varphi(v) = \frac{1}{1 + \exp(-v_j)} \tag{D.3.1}$$

where v_j is the induced local field (weighted sum of all synaptic outputs plus the bias) of neuron j. The nonlinearity is important because otherwise the

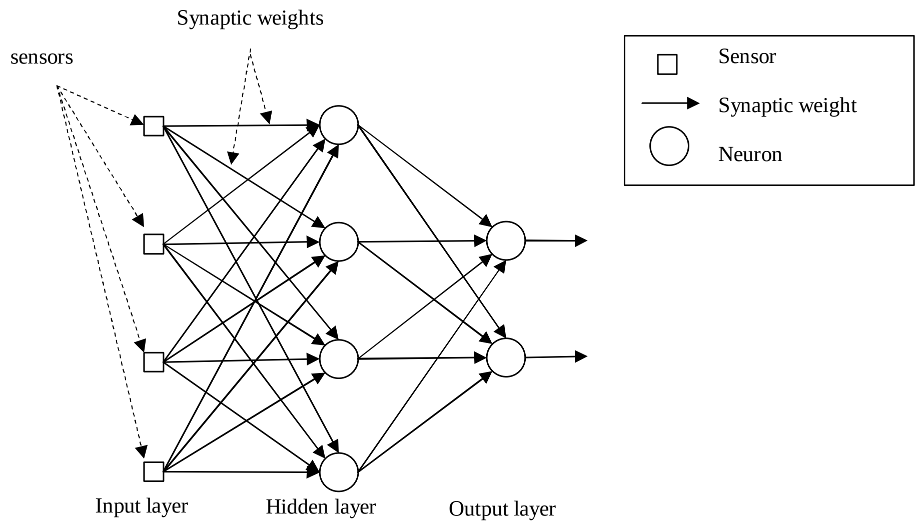


Figure D.4: Example of MLP network

functionality of the network could be reduced to that of a single-layer perceptron. The use of logistic transfer function is biologically motivated, because it attempts to account for the refractory phase of real neurons.

2. The network contains one or more hidden layers. These hidden layers enable the network to extract progressively more meaningful properties of input patterns. 3. High degree of connectivity between the neurons

3. Error back-propagation is a standard algorithm for the training of multilayer perceptrons. This algorithm is based on the error correction learning rule. It may be seen as a generalization of least-mean-square (LMS) algorithm for the special case of single linear neuron.

Back-propagation learning consists of two phases. In the first phase (forward pass), an activity pattern is applied to the network and the network produces response. In the backward pass the synaptic weights are adjusted according to the error-correction rule. This is based on the error signal which is a difference between actual response of network, and desired response. The error signal is propagated backward through the network.

Backpropagation algorithm with sequential updating of weights composes of following steps:

1. Initialization - pick the synaptic weights and thresholds

2. Sequential presentation of training examples - n-th presentation to the network with an epoch of training examples. For each example in the set perform steps 3 and 4.

3. Forward computation - the weights remain unaltered while an example is propagated through the network and error signal e(n) is computed:

   $$e(n) = t(n) - o(n) \tag{D.3.2}$$

   where $t(n)$ is the desired output and $o(n)$ is the real output of network for current example.

4. Backward computation Adjust the synaptic weights of the network according to the generalized delta rule ??? where $w_{ji}(n)$ represent weight connecting neuron i to neuron j at time step n; $w_{ji}(n+1)$ represent the same weight in time step $n{+}1$ (after back-propagation learning) and $\Delta w_{ji}(n)$ represent weight correction.

Weight correction equals ???, where η is the learning parameter, $\delta j(n)$ is local gradient and $y_i(n)$ is the input signal of neuron j.

Local gradients of the network $\delta j(n)$ are computed depending on the location of neurons:

- **for the output layer neurons**:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{D.3.3}$$

  , where $e_j(n)$ is the error signal for the output neuron j, φ' is the derivative of neuron transfer function and $v_j(n)$ is the induced field of neuron j.

- **for the hidden neurons**:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{D.3.4}$$

  , where $v_j(n)$ is the induced field of neuron j at time step n,

Finally the adjustment of the weight connecting neuron i to neuron j equals:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n) \tag{D.3.5}$$

where $w_{ji}^{(l)}(n)$ is the synaptic weight of the neuron j in layer l that is feed from neuron i in layer $l-1$; $y_i^{(l-1)}(n)$ is the output signal of neuron i in the previous layer $l-1$ at iteration n; $\delta_j^{(l)}(n)$ is the local gradient j in layer l and η is the learning rate parameter. Additional technique to ameliorate the performance is adding the momentum:

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha w_{ji}^{(l)}(n-1) + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n) \tag{D.3.6}$$

where α is the momentum constant

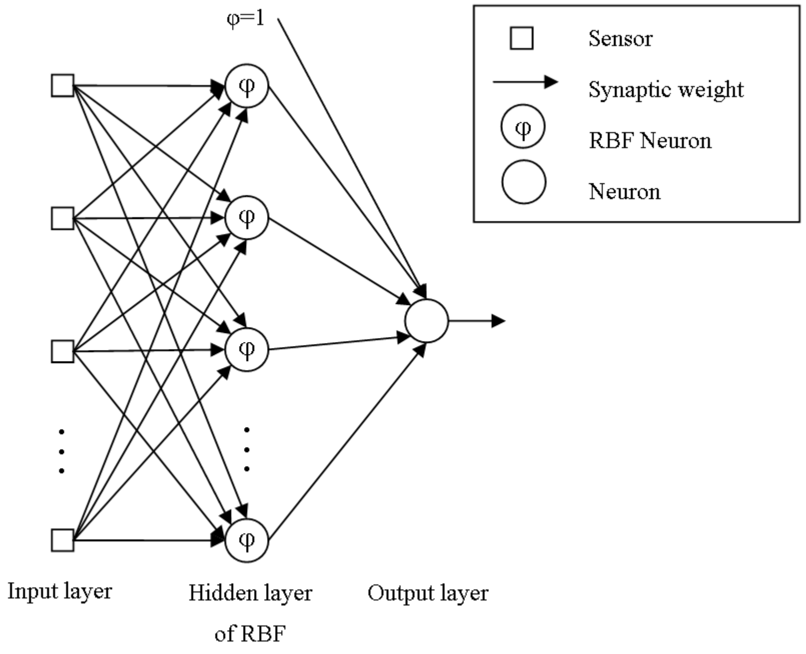


Figure D.5: Example of RBF network structure

D.4 Radial-Basis Function Networks

Structure of Radial-Basis Function (RBF) network is unusual because it has only one hidden layer and its hidden neurons are completely different from that of its output units. Theory in that field is linked closely with radial basic functions theory, which is one of the main fields of study in numerical analysis [Sin92]. Their outputs layer with linear weights is on the other hand closely linked with literature on linear adaptive filters [Hay99].

RBF networks [Pow88] have form of a function F:

$$F(x) = \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|) \tag{D.4.1}$$

where $\varphi(\cdot)$ is a set of N nonlinear functions known as radial-basis functions., and

$||\cdot||$ denotes a distance which is usually Euclidean. The known points x_i are the centers of the radial-basis functions (points in the input space). Weighting matrix w determines the influence of radial-basis functions on the network output. Radial basis function is a function of distance mentioned above. It can take many forms, for example:

$$\varphi(x) = e^{-\|x-x_i\|^2} \tag{D.4.2}$$

where x denotes input pattern and x_i denotes center of the radial basis function.

The number of radial basis functions (in network called RBF hidden neurons) is usually smaller than the number of available training examples (generalized RBF network). In the regularization RBF network, it is however equal to number of learning examples. Interesting property of RBF is that it constructs local approximations of nonlinear input-output mappings; it is in opposite to MLP which constructs global approximations.

Learning strategies for RBF networks:

1. fixed centers selected at random - radial-basis function are fixed and their centers are chosen randomly. Then the only parameters that would need to be learned are the linear weights matrix of the output layer. The drawback is that the method may need a large training set to achieve satisfactory performance

2. self-organized selection of centers - the methods compounds of two steps, first estimating appropriate location for the RBF centers, and then estimating the linear weights for the output layer by supervised learning. For the first step, one need a clustering algorithm, example of which can be *k-means algorithm*, [DH73] which is a special case of self-organizing map - neural network technique

described in section D.5. Simple method to estimate the output weights in the second phase is to use Least Mean Squares algorithm

3. supervised selection of centers - all free parameters of the network and the centers of RBF undergo a supervised learning process. Such a process can be an error correction learning using a gradient-descent procedure, which is a generalization of the LMS algorithm

4. strict interpolation with regularization [Yee88] - the method uses elements of the regularization theory and the kernel regression estimation The input-output mapping function of a Gaussian RBF network is similar to that realized by mixture of experts (2.2.3).

D.5 Competitive networks and Self Organizing Maps

Output neurons during competitive learning compete among themselves to be activated (fired). Only one neuron or one neuron per group is active. A neuron which wins the competition is called a winning neuron of winner-takes-all neuron. One of the methods of inducing the competition between neurons is to establish negative feedback path between them [Ros58].

In a self-organizing map, the neurons are placed at the nodes of a lattice (usually one- or two-dimensional). The winning neurons become selectively tuned to various input patterns during the competitive learning and finally they form a topographic map of the input patterns in which the spatial locations of the neurons in the lattice indicate intrinsic statistical features contained in the input patterns.

First model of self-organizing map was proposed by Willshaw and von der Malsburg. It consisted of two two-dimensional lattices of neurons, one representing presynaptic (input) layer and second postsynaptic (output) layer. The two lattices are interconnected by synapses of Hebbian type.

The second model of self-organizing map was introduced by Kohonen [Koh82]. The model provides a topological mapping that optimally places a fixed number of vectors into a higher dimensional place, therefore makes data compression easier. It consists of one lattice of neurons, which are fully connected to the source nodes in input layer. The model may be able to dimensional reduction on the input (compression). The Kohonen model belongs to the class of vector encoding algorithms.

D.6 Support Vector Machines

Support Vector Machine (SVM) is a linear machine. The technique can be used to pattern classification and nonlinear regression.

SVM works as an approximate implementation of the method of structural risk minimalization. It depends on the fact that error rate on test data is bounded by the sum of the training-error rate and a term that depends on the Vapnik-Chervonenkis (VC) dimension.

The base notion for the construction of SVM is the inner-product kernel between ”support vector” xi and the vector x drawn from the input space. The support vectors consist of small sub-database of the training data extracted by the algorithm. Depending on how the inner-product is generated, one can construct different learning machines, in particular:

- Polynomial learning machines
- Radial-basis function networks
- Two-layer perceptrons (i.e. with a single hidden layer)

For each of these feedforward networks, we may use support vector learning to implement the learning process using a given set of training data, automatically determining the required number of hidden units. The support vector machines learning algorithm has wide applicability and can be used with many network structures.

Appendix E

Applications of ANN

This appendix will present some of the successful applications of Artificial Neural Networks,. Applications below are listed in Defence Advanced Research Projects Agency [DARP] (DARPA) Neural Network Study.

Aerospace High performance aircraft autopilot, flight path simulation, aircraft control systems, autopilot enhancements, aircraft component simulation, aircraft component fault detection.[SD01], [AC98], [DAM00].

Automotive Automobile automatic guidance system, warranty activity analysis [JCC99]. Banking Check and other document reading, credit application evaluation [Rud95].

Defence Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification [BC91].

Electronics Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modelling

[FGMM99].

Entertainment Animation, special effects, market forecasting [GC98].

Financial Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction [TT92].

Insurance Policy application evaluation, product optimization [ADW01].

Manufacturing Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modelling of chemical process system [AR01].

Medical: Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, and emergency room test advisement [LLFM98], [BD03].

Oil and Gas Exploration [VM96], [YDA02].

Robotics Trajectory control, forklift robot, manipulator controllers, vision systems [LT99].

Speech Speech recognition, speech compression, vowel classification, text to speech synthesis [OL03].

Securities Market analysis, automatic bond rating , stock trading advisory systems [BvSP^{+}02].

Telecommunications Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems

[Ans94].

Transportation Truck brake diagnosis systems, vehicle scheduling, routing systems [SC95].

Appendix F

System identification

This appendix will present in detail system identification learning task in it's three variants: model identification, inverse system identification and plant control.

F.1 Model identification

Both original system and model are given input values x_i, the responses are compared to determine error e_i, which is used to train the Processing Unit. $d = f(x)$ - d are outputs of real system $f(\cdot)$ to inputs x $\hat{y} = M(x)$ - y are outputs of model $M(\cdot)$ to inputs x

F.2 Inverse system identification

Original system is provided with input data x_i, its output is given to inverse model which produces values y_i expected to be close to original input x$_i$. $d = f(x)$, where d are outputs of real system $f(\cdot)$ to inputs x $\hat{y} = M(x)$, where y are outputs of model

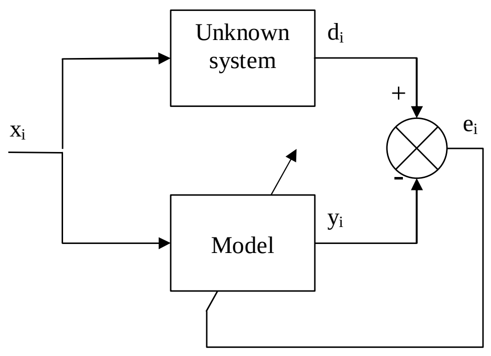


Figure F.1: System identification

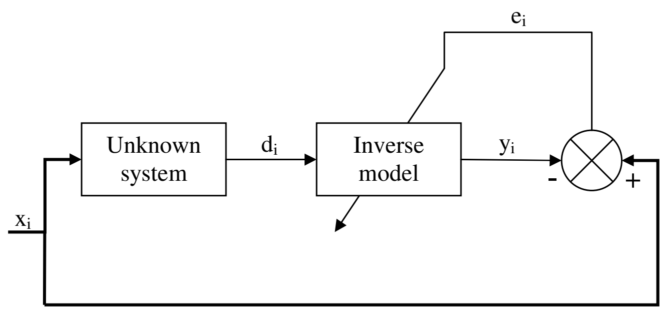


Figure F.2: Inverse system modelling

$M(\cdot)$ to original system outputs d

The error e_i is determined by comparing y_i and x_i, and used to tune the inverse model. The inverse model may be described as inverse function of $f(\cdot)$:

F.3 Plant control

This system identification task concerns control of a plant (important process, usually in real time). The objective of the controller (ANN) is to supply appropriate inputs

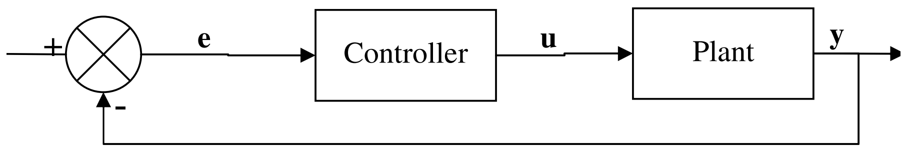


Figure F.3: Feedback control system

to the plant to make its outputs y track the reference signal d. In order to train the controller (adjust free ANN parameters) it is provided with error signal e from comparison between reference signal d and plant output y. The controller has to invert the plant's input-output behavior.

Bibliography

[ABR64] M. A. Aizerman, E. M. Braverman, and L. I. Rozono. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[AC98] S. Agarwal and S. Chaudhuri. Determination of aircraft orientation for a vision-based system using artificial neural networks. *Journal of Mathematical Imaging and Vision*, 8, Issue 3:255 – 269, May 1998.

[ADW01] A. Agarwal, J. T. Davis, and T. Ward. Supporting ordinal four-state classification decisions using neural networks. *Information Technology and Management archive*, 2:5 – 26, 2001.

[AG92] N. Avouris and L. Gasser. *Distributed Artificial Intelligence: Theory and Practics.* Kluwer Academic Press, 1992.

[AK89] E. H. L. Aarts and J. H. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, 1989.

[Ans94] N. Ansari. *Neural Networks in Telecommunications.* Kluwer Academic Publishers, 1994.

[AR01] T. Alifantis and S. Robinson. Manufacturing applications: Using simulation and neural networks to develop a scheduling advisor. *Winter*

Simulation Conference archive, Proceedings of the 33nd conference on Winter simulation, pages 954 – 958, 2001.

[Arb89] Michael A. Arbib. *The Metaphorical Brain.* New York: Wiley, 2 edition, 1989.

[BC91] R. H. Baran and James P. Coughlin. A neural network for target classification using passive sonar. Analysis of Neural Net Applications Conference Proceedings(2):188 – 198, 1991.

[BD03] A-S. Bellanger-Dujardin. *Contribution a l'etude de structures neuronales pour la classification de signatures : application au diagnostic de pannes des systemes industriels et a l'aide au diagnostic medical.* Ph.d. thesis, Paris XII Val De Marne - Creteil, 2003.

[Bel57] R. Bellman. *Dynamic Programming.* NJ: Princenton University Press, 1957.

[Ber94] R. Berlind. *An alternative method of stochastic discrimination with applications to Pattern Recognition.* Doctoral dissertation, Department of Mathematics, SUNY at Buffalo, 1994.

[BL73] T. V. Bliis and P. Lomo. Long lasting potential in the dentate area of anaesthetized rabbit following simulation of the perforatant path. *J. Physiol*, 232:331–356, 1973.

[BLR92] J. Bates, A. Bryan Loyall, and Scott W. Reilly. Integrating reactivity, goals and emotion in a broad agent. Technical report cmu-cs-92-142, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1992.

[Bri90] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing: Algorithms, architectures and applications*, 1990.

[BS95] J. Bruske and G. Sommer. Dynamic cell structure. *Advances in Neural Information Processing Systems*, 7:497–504, 1995.

[BvSP$^+$02] M. Botha, R. von Solms, K. Perry, E. Loubser, and G. Yamoyany. Research papers: information security and risk management: The utilization of artificial intelligence in a hybrid intrusion detection system. In South Africa Port Elizabeth, editor, *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 149 – 155, 2002. ISBN:1-58113-596-3.

[CH67] T. M. Cover and P. E. Hart. Nearest neighbour pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[CL96] J. Covie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.

[CMR02] A. Chebira, K. Madani, and M. Rybnik. La structure neuronale arborescente dts divide to simplify. *Colloque CNRS Neurosciences et Sciences pour Ingnieur (CNRS-NSI 2002)*, pages 16–18, Septembre 2002.

[Com91] Comon. Independent component analysis. In *Proc. Int. Workshop on Higher-Order Stat.*, pages 111–120, Chamrousse, France, 1991.

[CS92] P. S. Churchland and T. J. Sejnowski. *The computational Brain.* Cambridge, MA: MIT Press, 1992.

[CSS00] M. Collins, R. E. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. *Computational Learing Theory*, pages 158–169, 2000.

[DAM00] N. Durand, J-M. Alliot, and F. Mdioni. Neural nets trained by genetic algorithms for collision avoidance. *Applied Intelligence archive*, 13(3):205 – 213, November-December 2000.

[Dev87] L. Devroye. *A course in Density Estimation*. Birkhauser, Boston, MA, 1987.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[DSW97] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1997.

[Fer98] J. Ferber. *Multi-Agent Systems: Towards a Collective Intelligence*. Reading, MA: Addison-Wesley, 1998.

[FGMM99] A. Fanni, A. Giua, M. Marchesi, and A. Montisci. A neural network diagnosis approach for analog circuits. *Applied Intelligence archive*, 11 Issue 2(2), 1999.

[Fis00] A. Fisher. *The mathematical theory of probabilities*. John Wiley, 2000.

[FK71] K. Fukunaga and D. L. Kessel. Estimation of statistical error. *IEEE Transactions on Computers*, pages 1521–1527, December 1971.

[FL90] S. E. Fahlman and C. Lebiere. The cascaded-correlation learning architecture. *Advances in Neural Information Processing Systems*, 2:524–534, 1990.

[FR79] J. H. Friedman and L. C. Rafsky. Multi-variate generalizations of the wald-wolfowitz and smirnov two sample tests. *The Annals of Statistics*, 7(issue 4):697–717, 1979.

[Fuk90] K. Fukunaga. *Introduction to statistical pattern recognition.* Academic Press, new York, 2nd edition, 1990.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias-variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[GC98] S. Grand and D. Cliff. Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems archive*, 1 , Issue 1:39 – 57, 1998.

[GK96] S. Goonatilake and S. Khebbal. Intelligent hybrid systems: Issues, classification and future directions. *Intelligent Hybrid Systems*, pages 1–20, 1996.

[Gor83] R. L. Gorsuch. *Factor analysis.* Hillsdale, NJ: Erlbaum, 2 edition, 1983.

[Gro88] S. Grossberg. *Neural Networks and Natural intelligence.* Cambridge, MA: MIT Press, 1988.

[Han93] A. Hannibal. Vlsi building block for neural networks with on chip back learning. *Neurocoputing*, 5:25–37, 1993.

[Hay99] S. Haykin. *Neural Networks - a Comprehensive foundation.* Prentice Hall Int., 1999.

[HB97] T. K. Ho and H. S. Baird. Large scale simulation studies in pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[HB98] T. K. Ho and H. S. Baird. Pattern classification with compact distribution maps. *Computer Vision and Image Understanding*, 70(1):101–110, 1998.

[HB02] T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Transactions on pattern Analysis and Machine Intelligence*, 24, issue 3:289–300, March 2002.

[Heb49] D. O. Hebb. *The organization of behaviour: A Neuropsychological Theory*. New York: Wiley, 1949.

[HMS01] D. Hand, H. Mannila, and P. Smyth. *Principles of data mining*. Massachusetts Institute of Technology, 2001.

[Ho00] T. K. Ho. Complexity of classification problems and comparative advantages of combined classifiers. *Lecture Notes in Computer Science*, 2000.

[Ho01] T. K. Ho. Data complexity analysis for classifier combination. In *the 2nd International Workshop on Multiple Classifier Systems*, pages 53–67, Cambridge, UK, July 2001.

[Hoa62] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[Hub85] P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[Irv65] J. G. Irving. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. M. I.T. Press, 1965.

[JCC99] I. Jou, C. Chang, and H. Chen. A hybrid neuro-fuzzy system for adaptive vehicle separation control. *Journal of VLSI Signal Processing Systems archive*, 21, Issue 1:15 – 29, 1999.

[JJ95] Jordan and Jacobs. *???* ???, 1995.

[KD82] J. Kittler and P. A. Devijver. Statistical properties of error estimators in performance assessment of recognition systems. *IEEE Transactions on PAMI*, 4(2):215–220, March 1982.

[Kle96] E. Kleinberg. An overtraining-resistant stochastic modelling method for pattern recognition. *Annals of statistics*, 4(6):2319–2349, December 1996.

[KNM96] A. Kohn, L. G. Nakano, and V. Mani. A class discriminability measure based on feature space partitioning. *Pattern Recognition*, 29(5):873–887, 1996.

[Koh82] T. Kohonen. Self organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[KV95] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems*, 7:231–238, 1995.

[Lin91] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[LLFM98] S-C. B. Lo, J-S. J. Lin, M. T. Freedman, and S. K. Mun. Application of artificial neural networks to medical image pattern recognition: Detection of clustered microcalcifications on mammograms and lung cancer on chest radiographs. *Journal of VLSI Signal Processing Systems archive*, 18, Issue 3:263 – 274, April 1998.

[LT99] S-T. Lin and S-J. Tzeng. Neural network force control for industrial robots. *Journal of Intelligent and Robotic Systems archive*, 24 , Issue 3:253 – 268, March 1999.

[LW98] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In *Connectionist Models Summer School*, pages 52–59, 1998.

[Mad90] J. Maddox. Complicated measure of complexity. *Nature*, 344:705, April 1990.

[Mae] P. Maes. Social interface agents: Acquiring competence by learning from users and other agents. *Software Agents - Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*, pages 71–78.

[Mat67] K. Matusita. On the notion of anity of several distributions and some of its applications. *Annals Inst. Statistical Mathematics*, 19:181–192, 1967.

[MB88] G. J. McLachlan and K. E. Basford. *Mixture Models: Interference and Applications to Clustering.* New York: Marcel Dekker, 1988.

[MCR02] K. Madani, A. Chebira, and M. Rybnik. Multi-neural networks approach reducing complexity on both modelling and processing chain levels: application to classification and systems identification. *IAR International Workshop on Intelligent Control and Diagnosis (IAR/ICD 2002)*, November 2002.

[MCR03a] K. Madani, A. Chebira, and M. Rybnik. Data driven multiple neural network models generator based on a tree-like scheduler. *Lecture Notes in Computer Science: Computational Methods in Neural Modelling*, pages 382–389, 2003.

[MCR03b] K. Madani, A. Chebira, and M. Rybnik. A neural network based evolutionary treelike multi-models generator reducing complexity on both data and processing levels. *International Multi-conference on Computational Engineering in Systems Applications CESA 2003*, July 2003.

[NIH97] U. Naftaly, N. Intrator, and D. Horn. Optimal ensebmble averaging of neural networks. *Network*, 8:283–296, 1997.

[OL03] M. Ogihara and O. Li. Clustering: A comparative study on content-based music genre classification. In Canada Toronto, editor, *Annual ACM Conference on Research and Development in Information Retrieval archive, Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 282 – 289, 2003. ISBN:1-58113-646-3.

[Pie98] W. E. Pierson. *Using boundary methods for estimating class separability.* Phd thesis, Department of Electrical Engineering, Ohio State University, 1998.

[Pow88] M. J. D. Powell. Radial basis function approximations to polynominals. *Numerical Analysis 1987 Proceedings*, pages 223–241, 1988.

[RCM03a] M. Rybnik, A. Chebira, and K. Madani. Auto-adaptive neural network tree structure based on complexity estimator. *Lecture Notes in Computer Science: Computational Methods in Neural Modelling*, pages 558–565, 2003. Edited by: Jose Mira, Jose R. Alvarez - Springer Verlag Berlin Heidelberg.

[RCM$^+$03b] M. Rybnik, A. Chebira, K. Madani, K. Saeed, M. Tabedzki, and M. Adamski. Hybrid neural based information processing approach combining a view based feature extractor and a tree like intelligent classifier. *International Conference on Computer Information Systems and Industrial Management Applications 2003 - CISIM 2003*, June 2003. Elk, Poland.

[RF98] A. F. R. Rahman and M. Fairhurst. Measuring classification complexity of image databases : a novel approach. *Proceedings of International Conference on Image Analysis and Processing*, pages 893–897, 1998.

[RHHD56] N. Rochester, J. H. Holland, L. H. Haibt, and W. L. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on information Theory*, IT-2:80–93, 1956.

[RJ91] S. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition: recommendation for practictioners. *IEEE Transactions on PAMI*, pages 252–264, 1991.

[Roj96] R. Rojas. *Neural Networks, A Systematic Introduction.* Springer, Berlin, 1996.

[Ros58] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[Ros61] F. Rosenblatt. *Principles of Neurodynamics.* Washington D.C.:Spartan Press, 1961.

[Rud95] G. Rudorfer. Early bankruptcy detection using neural networks. *International Conference on APL archive Proceedings of the international conference on Applied programming languages*, pages 171 – 178, 1995.

[RZ85] D. E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[Sae00] K. Saeed. A projection approach for arabic handwritten characters recognition. *New Trends and Approaches in Computational Intelligence*, pages 106–111, 2000. Sinck J. Vack ed., Physica Verlag Heidelberg, New York.

[Sae03] K. Saeed. Object classification and recognition using toeplitz matrices. *Artificial Intelligence and Security in Computing Systems*, pages 167–172, 2003. Kluwer Academic Publishers, Nowell, Massachutsetts, USA, J. Soldek, L. Drobiazgowicz (Ed.).

[Sar94] W. S. Sarle. Neural networks and statistical methods. *Proceedings of the Nineteenth Anuual SAS Users Group International Conference*, April 1994.

[SC95] R. H. Smith and D. C. Chin. Evaluation of an adaptive traffic control technique with underlying system changes. In *27th conference on Winter simulation, Winter Simulation Conference archive*, pages 1124 – 1130, Arlington, Virginia, United States, 1995. ISBN:0-7803-3018-8.

[Sch99a] R. E. Schapire. A brief introduction to boosting. *IJCAI*, pages 1401–1406, 1999.

[Sch99b] Robert Schapire. Theoretical views of boosting and applications. *Tenth International Conference on Algorithmic Learning Theory*, pages 13–25, 1999.

[SD01] Simon and L. Donald. A hybrid neural network-genetic algorithm technique for aircraft engine performance diagnostics. *AIAA-2001-3763*, 2001. (NASA/TM-2001-211088.

[Sej77] T. J. Sejnowski. Strong covariance with nonlinearly interacting neurons. *Journal of Mathematical Biology*, 4:303–321, 1977.

[SG02] S. Singh and A. P. Galton. Pattern recognition using information slicing model (prism). *Proc. 15th International Conference on Pattern Recognition (ICPR2002)*, August 2002. Quebec.

[Sin92] S. P. Singh. *Approximation Theory, Spline Functions and Applications.* Kluwer, Dortrecht, The Netherlands, 1992.

[Sin02a] S. Singh. Estimating classification complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence submission*, 2002.

[Sin02b] S. Singh. Prism, cells and hypercuboids. *Pattern Analysis and Applications*, 5, 2002.

[Sin03] S. Singh. Multi-resolution estimates of classification complexity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003.

[SN95] K. K. Sung and P. Niyogi. Active learning for function approximation. *Advances in Neural Information Processing Systems*, 7:593–600, 1995. The MIT Press, Ed by G. Tesauro.

[STA02] K. Saeed, M. Tabedzki, and M. Adamski. A new approach for object-feature extract and recognition. *9th International Conference ACS (Advanced Computer Systems)*, pages 389–397, October 2002. Miedzyzdroje 2002, Poland,.

[STA03] K. Saeed, M. Tabedzki, and M. Adamski. A view-based approach for object recognition. *Conradi Research Review*, 2, Issue 1:85–95, 2003. Finland.

[TB97] M. Tipping and C. Bishop. Probabilistic principal component analysis. Technical report, Neural Computing Research Group, Aston University, 1997.

[TKM87] T. Takeshita, F. Kimura, and Y. Miyake. On the estimation error of mahalanobis distance. *Trans. IEICE*, pages 567–573, 1987.

[Tou74] G. T. Toussaint. Bibliography on estimation of misclassification. *IEEE Transactions on information Theory*, pages 472–479, July 1974.

[Tre01] Volker Tresp. *Handbook for Neural Network Signal Processing*, chapter Committee Machines. CRC Press, 2001.

[TSM85] D. M. Titterington, A. F. Smith, and V. E. Makov. *Statistical Analysis of Finite Mixture Distributions*. New York: Wiley, 1985.

[TT92] Robert R. Trippi and Efraim Turban. *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*. McGraw-Hill, Inc. New York, NY, USA, 1992.

[Vap98] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[VM96] M. A. Vukelic and E. N. Miranda. Neural networks in petroleum engineering. *International Journal of Neural Systems*, 7, 187, 1996.

[Was93] P. D. Wasserman. *Advanced Methods in Neural Computing*. New York: Van Nostrand Reinhold, 1993.

[WJ95a] M. Wooldridge and N. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10:2:115–152, 1995.

[WJ95b] Michael J. Wooldridge and Nicholas R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In Michael J. Wooldridge and Nicholas R. Jennings, editors, *Workshop on Agent Theories, Architectures & Languages (ECAI'94)*, volume 890 of *Lecture Notes in Artificial Intelligence*, pages 1–22, Amsterdam, The Netherlands, January 1995. Springer-Verlag.

[YDA02] S. Yilmaz, C. Demircioglu, and S. Akin. Application of artificial neural networks to optimum bit selection. *Computers & Geosciences archive*, 28, Issue 2:261 – 269, March 2002.

[Yee88] P. V. Yee. *Regularized Radial Basis Function Networks: Theory and Applications to Probability Estimation, Classification and Time Series Prediction.* Ph.d. thesis, McMaster university, Hamilton, Ontario, 1988.