Adaptive Learning of Polynomial Networks

Genetic Programming, Backpropagation and Bayesian Methods
GENETIC AND EVOLUTIONARY COMPUTATION SERIES

Series Editors

David E. Goldberg
University of Illinois at Urbana-Champaign

John R. Koza
Stanford University

Selected titles from the Series:


For a complete listing of books in this series, go to http://www.springer.com
Adaptive Learning of Polynomial Networks
Genetic Programming, Backpropagation and Bayesian Methods

Nikolay Y. Nikolaev
University of London

Hitoshi Iba
The University of Tokyo

Springer
Contents

Preface xi

1. INTRODUCTION 1
   1.1 Inductive Learning 3
       1.1.1 Learning and Regression 4
       1.1.2 Polynomial Models 5
       1.1.3 Inductive Computation Machinery 5
   1.2 Why Polynomial Networks? 7
       1.2.1 Advantages of Polynomial Networks 8
       1.2.2 Multilayer Polynomial Networks 9
   1.3 Evolutionary Search 16
       1.3.1 STROGANOFF and its Variants 17
   1.4 Neural Network Training 21
   1.5 Bayesian Inference 22
   1.6 Statistical Model Validation 23
   1.7 Organization of the Book 23

2. INDUCTIVE GENETIC PROGRAMMING 25
   2.1 Polynomial Neural Networks (PNN) 26
       2.1.1 PNN Approaches 27
       2.1.2 Tree-structured PNN 29
   2.2 IGP Search Mechanisms 35
       2.2.1 Sampling and Control Issues 36
       2.2.2 Biological Interpretation 36
   2.3 Genetic Learning Operators 38
       2.3.1 Context-preserving Mutation 38
       2.3.2 Crossover Operator 40
Contents

5. SEARCH NAVIGATION 111
  5.1 The Reproduction Operator 112
    5.1.1 Selection Strategies 113
    5.1.2 Replacement Strategies 117
    5.1.3 Implementing Reproduction 118
  5.2 Advanced Search Control 119
    5.2.1 Macroevolutionary Search 119
    5.2.2 Memetic Search 120
    5.2.3 Search by Genetic Annealing 122
    5.2.4 Stochastic Genetic Hillclimbing 124
    5.2.5 Coevolutionary Search 125
    5.2.6 Distributed Search 128
  5.3 Performance Examination 128
    5.3.1 Fitness Evolvability 129
    5.3.2 Convergence Measures 130
    5.3.3 Diversity Measures 133
    5.3.4 Measures of Self-Organization 139
  5.4 Chapter Summary 146

6. BACKPROPAGATION TECHNIQUES 147
  6.1 Multilayer Feed-forward PNN 148
  6.2 First-Order Backpropagation 149
    6.2.1 Gradient Descent Search 150
    6.2.2 First-Order Error Derivatives 151
    6.2.3 Batch Backpropagation 157
    6.2.4 Incremental Backpropagation 158
    6.2.5 Control of the Learning Step 159
    6.2.6 Regularized Delta Rule 162
  6.3 Second-Order Backpropagation 163
    6.3.1 Second-Order Error Derivatives 164
    6.3.2 Newton's Method 169
    6.3.3 Pseudo-Newton Training 170
    6.3.4 Conjugate Gradients 170
    6.3.5 Levenberg-Marquardt Method 171
  6.4 Rational Backpropagation 172
  6.5 Network Pruning 176
    6.5.1 First-Order Network Pruning 176
    6.5.2 Second-Order Network Pruning 177
6.6 Chapter Summary 179

7. TEMPORAL BACKPROPAGATION 181

7.1 Recurrent PNN as State-Space Models 182
7.2 Backpropagation Through Time 184
  7.2.1 Derivation of BPTT Algorithms 185
  7.2.2 Real-Time BPTT Algorithm 189
  7.2.3 Epochwise BPTT Algorithm 190
7.3 Real-Time Recurrent Learning 191
7.4 Improved Dynamic Training 198
  7.4.1 Teacher Enforced Training 198
  7.4.2 Truncating in Time 199
  7.4.3 Subgrouping 199
  7.4.4 Common Temporal Training Problem 200
7.5 Second-Order Temporal BP 200
7.6 Recursive Backpropagation 204
7.7 Recurrent Network Optimization 206
  7.7.1 Regularization 207
  7.7.2 Recurrent Network Pruning 207
7.8 Chapter Summary 208

8. BAYESIAN INFERENCE TECHNIQUES 209

8.1 Bayesian Error Function 211
8.2 Bayesian Neural Network Inference 212
  8.2.1 Deriving Hyperparameters 215
  8.2.2 Local vs. Global Regularization 217
  8.2.3 Evidence Procedure for PNN Models 218
  8.2.4 Predictive Data Distribution 221
  8.2.5 Choosing a Weight Prior 221
8.3 Bayesian Network Pruning 222
8.4 Sparse Bayesian Learning 224
8.5 Recursive Bayesian Learning 229
  8.5.1 Sequential Weight Estimation 229
  8.5.2 Sequential Dynamic Hessian Estimation 230
  8.5.3 Sequential Hyperparameter Estimation 232
8.6 Monte Carlo Training 234
  8.6.1 Markov Chain Monte Carlo 235
  8.6.2 Importance Resampling 237
Contents

8.6.3 Hybrid Sampling Resampling 237
8.7 Chapter Summary 239

9. STATISTICAL MODEL DIAGNOSTICS 241
9.1 Deviations of PNN Models 242
9.2 Residual Bootstrap Sampling 243
9.3 The Bias/Variance Dilemma 244
  9.3.1 Statistical Bias and Variance 244
  9.3.2 Measuring Bias and Variance 245
9.4 Confidence Intervals 248
  9.4.1 Interval Estimation by the Delta Method 248
  9.4.2 Bootstrapping Confidence Intervals 252
9.5 Prediction Intervals 254
  9.5.1 Analytical Prediction Intervals 255
  9.5.2 Empirical Learning of Prediction Bars 256
9.6 Bayesian Intervals 262
  9.6.1 Analytical Bayesian Intervals 263
  9.6.2 Empirical Bayesian Intervals 265
9.7 Model Validation Tests 267
9.8 Chapter Summary 271

10. TIME SERIES MODELLING 273
10.1 Time Series Modelling 274
10.2 Data Preprocessing 276
10.3 PNN vs. Linear ARMA Models 277
10.4 PNN vs. Genetically Programmed Functions 279
10.5 PNN vs. Statistical Learning Networks 281
10.6 PNN vs. Neural Network Models 283
10.7 PNN vs. Kernel Models 285
10.8 Recurrent PNN vs. Recurrent Neural Networks 288
10.9 Chapter Summary 290

11. CONCLUSIONS 291

References 295

Index 313
Preface

This book provides theoretical and practical knowledge for development of algorithms that infer linear and nonlinear models. It offers a methodology for inductive learning of polynomial neural network models from data. The design of such tools contributes to better statistical data modelling when addressing tasks from various areas like system identification, chaotic time-series prediction, financial forecasting and data mining. The main claim is that the model identification process involves several equally important steps: finding the model structure, estimating the model weight parameters, and tuning these weights with respect to the adopted assumptions about the underlying data distribution. When the learning process is organized according to these steps, performed together one after the other or separately, one may expect to discover models that generalize well (that is, predict well).

The book offers statisticians a shift in focus from the standard linear models toward highly nonlinear models that can be found by contemporary learning approaches. Specialists in statistical learning will read about alternative probabilistic search algorithms that discover the model architecture, and neural network training techniques that identify accurate polynomial weights. They will be pleased to find out that the discovered models can be easily interpreted, and these models assume statistical diagnosis by standard statistical means.

Covering the three fields of: evolutionary computation, neural networks and Bayesian inference, orients the book to a large audience of researchers and practitioners. Researchers in genetic programming will study how to elaborate model representations, how to make learning operators that sample the search space efficiently, how to navigate the search process through the design of objective fitness functions, and how to examine the search performance of the evolutionary system. The pos-
sibility to use reliable means for observing the search behavior of genetic programming is one of its essential advantages.

Practitioners in artificial neural networks will study how to determine automatically the network structure prior to applying the weight training algorithm. They will realize what are the steps of the global search mechanisms that allow identification of relevant network topologies from data. Searching for the optimal network structure is essential for adjusting the network to the concrete application task. Even if one knows how to make the weight training algorithm, there is a need to determine in advance the network architecture. In addition to this, the book gives various activation and basis functions for elaborating different neural network mappings, such as kernels, harmonics, Gaussians etc., which extend their descriptive power.

Specialists in Bayesian inference will read about applications of the principles of probabilistic learning to polynomial networks. The Bayesian training of polynomial networks makes them practical tools for solving difficult real-world problems from various fields. This book demonstrates that polynomial networks can be trained probabilistically not only in off-line mode, but also in recursive mode. Novel Bayesian techniques for reliable recursive training of polynomial networks are offered. Practitioners in econometrics will especially find interesting the fact that polynomial networks can be trained using sampling methods, which makes them attractive for financial forecasting.

Students will find this book useful for studying genetic programming, neural networks, and probabilistic learning. There is teaching material from the courses that we taught the last few years to our students in artificial intelligence, machine learning, evolutionary computation and neural networks. The material is self-contained and includes: definitions of the main inductive tasks, formulations of the basic approaches to addressing these tasks, introduction to the fundamentals of genetic programming, review of backpropagation training, and presentations of the basics of Bayesian learning. Undergraduate students will learn how to design and implement the basic mechanisms of a genetic programming system, including the selection scheme, and the crossover and mutation learning operators. Postgraduate students will study advanced topics such as improving the search control of genetic programming systems, and tools for examination of their search performance. In order to facilitate the understanding and easy memorization of the algorithms, they are summarized in tables at the end of each subsection.

The inspiration to write this book came after a long period of conducting research in inductive problem solving. The authors experience started with work on inductive tasks using machine learning algorithms.
The work in this area led to disappointment due to the many fundamental inabilities of the typical machine learning algorithms. First, they seem to be quite conservative, dealing mainly with inexpressive propositional concept description languages. Even advanced inductive programming systems cannot directly manipulate well-known nonlinear models from the statistical analysis. Second, they lack flexible learning operators that can efficiently sample the search space, as for each representation language they require specific learning operators. Third, they do not allow for observation of their performance, while the learning proceeds, which hinders tuning of the parameters.

Genetic programming offered interesting and challenging ideas for making innovative computational algorithms. This research showed that it is possible to devise domain independent model representations; it is possible to make general learning operators that need not be changed when changing some representation details; it is possible to organize both global and local searches for induction of good models; it is possible to navigate the search using reliable formulae from statistics and numerical optimization; and it is possible to validate the results with standard statistical tools. Above all, the results were better than those produced by machine learning algorithms. In addition to this, the genetic programming paradigm allowed incorporation of the most recent achievements in artificial neural networks and generation of well-performing models from benchmark and real-world data.

An enormous number of experiments on various data sets were conducted during the last several years by both authors. Most of the experiments were successful, which was considered convincing evidence for the potential of genetic programming. The idea of further improving the best results by neural network training and Bayesian algorithms was suggested later by several researchers who expressed scepticism that genetic programming alone could discover optimal models from data. Specialized backpropagation and Bayesian techniques for training polynomial networks were further developed and tested. It was found that on some tasks they really could achieve considerable improvements, and so were worth investigation and special attention.

The developments in this book aim to facilitate the inference of polynomial models for time-series prediction. The orientation toward time-series forecasting comes from the inherent difficulty in many scientific areas to find models that describe, sufficiently well, unknown dynamical systems that have generated the series. The interest in such natural problems is that many everyday tasks actually fall in this category. The presented successful experimental results suggest that, in general, the proposed methodology can be useful in practical inductive modelling.
Acknowledgments. The material in this book matured with our experience. The first author, Dr. Nikolaev, developed his doctoral thesis under the supervision of Professor N. Kasabov who currently works at the University of Auckland. After that he was a postdoctoral fellow in the lab of Professor D.T. Pham at University of Cardiff. During the years Dr. Nikolaev enjoyed working with his colleague and best friend Dr. Evgeny Smirnov, University of Maastricht, who inspired him to work on machine learning algorithms. He also conducted research with Dr. Vanyo Slavov from New Bulgarian University, Sofia on immune network algorithms and Dr. Lilian de Menezes from City University, London on statistical model diagnostics. Dr. Peter Tino from the University of Birmingham thought him to do Bayesian inference. He thanks especially to his colleagues from The American University in Bulgaria and Goldsmiths College for providing him with a pleasant working atmosphere. He is particularly grateful to his dear girlfriend, Dr. Snezhana Dimitrova, for her inspiration, encouragement, and tolerance while writing this book. Nikolaev is extremely grateful to his parents for their love and support. They have always helped him, shaped his thinking and made him a scientist.

The second author, Dr. Hitoshi Iba, acknowledges the pleasant research atmosphere created by his colleagues and students from the research laboratory associated with Graduate School of Frontier Sciences at the University of Tokyo. He is also grateful to his previous group and colleagues at Electro-Technical Laboratory, where he used to work for ten years. Particular thanks are due to Dr. Hirochika Inoue and Dr. Taisuke Sato for their providing precious comments and advice on numerous occasions. And last, but not least, he would like to thank his wife Yumiko and his sons and daughter Kohki, Hirono, and Hiroto, for their patience and assistance.

Both authors are grateful to Professor Alexey G. Ivakhnenko and his son Gregory Ivakhnenko for the invaluable discussions on the multilayer and harmonic GMDH algorithms. These discussions reinforced our enthusiasm to work in this area.
Chapter 1

INTRODUCTION

Polynomial neural networks (PNN) are global function models whose parameters, once learned from given data, can be used for generating predictions without the need to infer additional versions for the separate neighborhoods in the data. PNN provide opportunities to reach accuracy superior to many global models such as nonlinear functions, statistical learning networks, multilayer perceptrons, and other feed-forward neural networks on multivariate nonlinear regression, time-series forecasting and classification problems. Inferring PNN models from data is an inductive computation problem that requires development of corresponding learning algorithms.

This book develops a methodological framework for induction of multilayer PNN. It involves several steps: 1) elaboration of neural network representations of polynomials; 2) search for the proper network structure (architecture) using the evolutionary genetic programming paradigm; 3) adjustment of the polynomial coefficients (referred to further as weights) by gradient descent search using backpropagation training techniques; 4) enhancement of the network generalization potential using Bayesian inference; and 5) model validation with diagnostic methods. These five steps make a coherent and integrated methodology for identification of well-performing polynomial models. The rationale is in the tight coupling of the second, third and fourth learning steps which suggest to further adapt the evolved polynomial network and its coefficients by backpropagation and Bayesian techniques.

The difficulties in this polynomial learning methodology are what kind of representation to choose, how to organize evolutionary search with it, whether it can enable neural network training, and whether its prediction can be improved by assumptions about the underlying distribution
of the data. What this book emphasizes is that the model representation is essential for the development of inductive learning algorithms. This is the representation which allows us to employ efficient search methods, to derive neural network training algorithms, to tune the coefficient’s variance with respect to the target density, and to perform reliable analysis of the results. The key idea is to make tree-structured polynomial networks that can be flexibly tailored to the data. When equipped with neural network and Bayesian methods for weight training and pruning, these polynomial networks become PNN, and thus alternative methods for statistical data analysis.

The PNN are multilayer perceptrons of neuron-like units which produce high-order polynomials [Barron, 1988, Elder and Brown, 2000, Farlow, 1984, Gosh and Shin, 1992, Ivakhnenko, 1971, Müller and Lemke, 2000, Marmarelis and Zhao, 1997, Pao, 1989, Pham and Liu, 1995, Tenorio and Lee, 1990, Wray and Green, 1994, Zhang and Mühlenbein, 1995]. Their distinctive advantage is the ability to find optimal higher-order term weights. PNN inherit some good features from their predecessors, the multilayer perceptron networks, while often showing better accuracy of fit and forecasting. Such an important feature is that they assume gradient descent training by error backpropagation.

The multistep inductive learning of polynomials from data became possible with the recent progress in biologically inspired computation. Search for optimal tree-like topologies can be organized using genetic programming. Search in the weight space can be implemented using both backpropagation and Bayesian techniques for neural networks. Research in artificial neural networks demonstrates that they have the capacity to carry out reliable learning despite discrepancies in the data. A serious criticism to most connectionist models is that they require us to predefine their structure. There are algorithms that construct the network architectures but they do this with an inefficient topological search. A better search may be conducted using the evolutionary paradigms, such as evolution strategies, genetic algorithms, and genetic programming. They perform global exploration as well as local exploitation of the neural network shape spaces, which helps to locate good solutions. Most of the neural network strategies, however, assume that the data are fixed and produce point predictions. Such predictions are unrealistic in practical situations where it is necessary to exploit the data together with uncertainties in them. This directs the attention toward investigating Bayesian methods for proper treatment of the data with their inherent noise. Also proposed are Bayesian inference algorithms that adjust the weights along with their variances, and so enable us to make more reliable probabilistic predictions.
Introduction

This book integrates the strengths of evolutionary paradigms, artificial neural networks, and the probabilistic methods for efficient simulation of inductive computation processes. The existing multilayer polynomial networks have a flexible structure that allows adaptation of the polynomials to the data. Applying evolutionary algorithms to them helps to discover the relevant explanatory variables and their interrelationships. Multilayer polynomial networks typically use conventional methods to find the values of the weights, such as least squares fitting methods. While ordinary least squares fitting methods are optimal for linear models, it is not clear at all whether they are optimal for nonlinear models. The problem of using least squares fitting for learning the weights of composed nonlinear models is that they do not guarantee reaching sufficiently good accuracy. When nonlinear models are built, the relationships between the variables in them are complex and need sophisticated treatment. This is the rationale for applying specialized connectionist algorithms to polynomial networks.

Even equipped with such specialized weight training algorithms, the polynomial networks assume further adaptation with regard to the underlying assumptions about the noise in the data. During the weight training process, the possible noise in the data distribution has to be considered so as to tune the model better. This involves updating the belief in the amount of noise in order to adjust the belief in the uncertainty of the weights. Having inferred the degree of certainty in the network weights, probabilistic predictions can be made which are more reliable as they reflect the characteristics of the data generating function with a computable level of confidence.

1.1 Inductive Learning

The problem with inductive learning is to identify a model that optimally describes the characteristics of provided data, usually collected from observations of natural, real-world phenomena. The necessary tools for dealing with such a problem include [Vapnik, 1995, Vapnik, 1998]: a generator of example input data, a teacher that associates each input with a corresponding desired output, and a learning device that implements the model. The learning device maps the given input vectors to estimated outputs. Since the desired outputs are known, this formulation is commonly named supervised inductive learning. The objective is to find a model whose estimated output is maximally close to the desired output. The search for the parameters of the learning device that best describes the given data is organized by a machinery for inductive computation. Inductive learning is essentially a search problem guided by proper criteria for closeness with the data.
ADAPTIVE LEARNING OF POLYNOMIAL NETWORKS

The data generator draws independently random example (input) vectors $x$ from some probability density $Pr(x)$. The teacher assigns to each example a corresponding desired (target) output $y$ according to a probability distribution $Pr(y|x)$. The joint probability density function of these independent and identically distributed data is: $Pr(x, y) = Pr(x) Pr(y|x)$. Having collected a set of data $D = \{(x_n, y_n)\}_{n=1}^{N}$, the goal is to find a mapping $P(x) \rightarrow y$ as close as possible to the true unknown mapping $P^*(x)$. The mapping is parameterized by a set of weights $w = (w_1, w_2, \ldots)$, that is we have functions of the kind $P(x, w)$. Having an estimate of the error only on the available data, the intention is to achieve high generalization on modelling unseen data. The learning criterion is given by the total generalization error:

$$E = \int L(y, P(x, w)) Pr(x, y) dx dy$$

where $L(y, P(x, w))$ is a loss estimate (a measure of the training error).

The difficulty in solving this learning problem arises from the unknown probability density, but from another point of view this motivates the need to construct special inductive computation machinery. This computation machinery should be powerful enough to learn from finite data, which is known to be an ill-posed problem that requires apriori assumptions about the complexity of the learning device. The learning device is typically a function, such as a non-linear function, a polynomial, a radial-basis function, a spline function, a neural network function, etc.. This is why the learning device is simply called a function or model. Function models are suitable for addressing many inductive tasks such as density estimation, classification, regression, etc..

1.1.1 Learning and Regression

Inductive learning can be formulated as a regression task as follows: given an example set of input vectors $D = \{(x_n, y_n)\}_{n=1}^{N}$ that are measurements of explanatory variables $x_n = [x_{n1}, x_{n2}, \ldots, x_{nd}], x \in \mathcal{R}^d$, and corresponding desired values of the dependent variable $y_n \in \mathcal{R}$, the goal is to find a function $P$ that describes the mapping $y = P(x) + \varepsilon$, where $\varepsilon$ is a zero mean normally distributed noise and $P \in L_2$, which on average converges to the true unknown mapping $P^*(x)$. The linear space $L_2$ contains functions with integrable squares, i.e. is the integral $\int P^2(x) d\xi$, where $\xi$ is the space metric, it exists and it is finite [Kolmogorov and Fomin, 1999]. The function models the conditional mean of the data:

$$P(x, w) = \int y Pr(y|x) dy$$

where $Pr(y|x)$ is the probability density of the given outputs.
The learning criterion in the Euclidean space is the following quadratic empirical loss:

\[ L(y, P(x, w)) = (y - P(x, w))^2 \] (1.3)

The objective of inductive computation is to discover a model that minimizes the risk functional applied with this empirical loss, which is the error on the provided training data. Although the search is guided by the empirical loss, the expectation is that the best solution will also have low prediction risk, that is low error on unseen data.

1.1.2 Polynomial Models

The presented investigations adopt polynomials as universal function models. The rationale for choosing polynomials comes from the Stone-Weierstrass approximation theorem [Davis, 1975]. It states that polynomials are a universal approximation format with which there could be described any continuous function on a compact set. The polynomials provide sufficient expressive power for accurate data modelling; in other words they are reliable descriptive tools. Polynomial models can be built, e.g. like algebraic polynomials, orthogonal polynomials, trigonometric polynomials, rational polynomials, local basis polynomials, etc.. Such models can be represented as multilayer neural networks in order to facilitate the structure selection, the coefficient estimation, and the complexity tuning (including term pruning).

This book offers approaches to design and implementation of computational micromechanisms for inductive learning and specializes them for automated discovery of polynomial neural network models.

1.1.3 Inductive Computation Machinery

The development of inductive computation machinery for a chosen function model, like polynomials for example, involves: 1) choosing a search paradigm for model selection; 2) elaborating model representations suitable for manipulation by the operators offered by this paradigm; 3) organizing search navigation with relevant selection criteria; 4) designing parameter training algorithms for their fine-tuning to the data; 5) implementing probabilistic inference tools for enhancing the generalization performance; and 6) carrying out model validation using diagnostic methods and tools.

The Model Search Paradigm. The search paradigm serves to find the most adequate model structure for the data. When working with polynomials, the objective is to find PNN with complexity relevant to the training data in the sense of terms and order. The search paradigm should be powerful enough to perform exploration of large, distant ar-
eas on the search landscape as well as to perform exploitation of small, neighborhood areas. Among the different paradigms such as heuristic search, simulated annealing, random search and others, the genetic programming paradigm is preferred. Genetic programming offers learning operators and micromechanisms inspired from biology that have abilities to conduct guided stochastic search in large spaces. Genetic programming is a general search paradigm which operates on tree-structured model representations.

**The Model Representation.** A search paradigm is equipped with learning operators to manipulate the concrete structures that implement the adopted model representation. The genetic programming evolves tree-like model representations that are processed by evolutionary learning operators. This book develops different polynomials in the form of tree-structured networks, including algebraic network polynomials, kernel network polynomials, orthogonal network polynomials, trigonometric network polynomials, rational network polynomials, and dynamic network polynomials. The approximation characteristics of the models in evolutionary computation are traditionally referred to as their fitness.

**Search Navigation.** The model search process may be envisioned as flowing on a landscape surface built from the fitnesses of all possible models of the selected kind. The learning apparatus has to efficiently navigate the search on the landscape in order to examine it thoroughly and to locate a good solution. The search is guided by inductive principles which tell us how to define model selection criteria. One advantage of choosing polynomials is that they enable us to directly apply well-known principles for automated induction. Such inductive principles that can be built in the learning machinery are the statistical inference principle, the Bayesian inference principle, the maximum likelihood principle, the minimum description length principle, the structural risk minimization principle, and the regularization principle.

Considered for learning polynomials these principles become criteria that helps to find models that trade off between the accuracy and the generalization capacity. This is necessary because the provided training samples in practice are finite, which requires us to adapt the model well to the data. Complex models exhibit high accuracy on the training data but they are poor predictors, while simple models may tend to predict well but fail to reach a satisfactory training accuracy.

**Weight Training.** The inductive learning paradigms usually identify the model structure by search, and during the search they estimate the model parameters. The genetic programming conducts evolutionary search for the most relevant polynomial network tree-like structure from the data using the least squares fitting method for coefficient estima-
Introduction

However, there is no guarantee that only parameter estimation is sufficient to obtain a good solution with optimal coefficients. This book claims that evolved polynomial networks assume further improvement by connectionist training algorithms. Backpropagation techniques for gradient descent search are derived especially for high-order neural networks with polynomial activation functions.

**Generalization Enhancement.** Having a promising network model structure is not sufficient in most cases to achieve good generalization. The polynomial network model may not exhibit good predictive performance if its weights do not reflect the underlying assumptions about the noise in the data. There could be normally distributed noise, or heavy tail noise, or other noise distributions that affect the model performance on unseen data. If the characteristics of the output noise are not taken into account in training, the weights will not be able to well capture the characteristics of the true data generation process. Probabilistic learning algorithms are presented and specialized here for PNN training. It is shown how the learning algorithm updates the belief in the weights along with the arrival of the data with respect to the preliminary assumptions about the probability distribution of the data.

**Model Validation.** After inductive inference of a promising model structure and its parameters, this model has to be validated. It has to be measured as to what degree the model is an adequate description of the data. The polynomials can be easily validated using standard statistical diagnosis tools. In addition to this, when polynomials are made as networks they can also be tested using approaches designed for neural network and Bayesian diagnosis. Statistical diagnosis can be performed by residual sampling methods, which can be used to measure confidence and prediction intervals of the polynomial neural networks. The possibility to test polynomials with both kinds of methods increases the certainty in their approximation characteristics.

1.2 Why Polynomial Networks?

Multilayer polynomial networks are a class of power series function models constructed using hierarchical networks of first-order and second-order processing units. These are higher-order neural networks with multiplicative activation functions, sparse connectivity, and parsimonious structure. There are three conjectures that motivate the representation of polynomials as neural networks [Marmarelis, 1994]. First, polynomials can be related to multilayer perceptron networks as universal approximators, so it is worth building polynomials as neural networks in order to use the achievements in connectionist learning.
Second, the modelling of a function requires, in general, a polynomial network of smaller size than the corresponding neural network required for the same function. The polynomial networks produce more compact and accurate models than multilayer perceptrons. This expressive difference is mainly due to the restriction of the activation functions of the multilayer perceptron to a specific squashing nonlinearity, typically the sigmoidal or tangential functions. This leads to increasing the number of hidden units necessary to achieve satisfactory performance.

Third, polynomial networks can be designed with bounded as well as unbounded activation polynomials in the hidden network nodes. There is a long debate on whether or not to use bounding of the activation polynomials through some squashing functions. It is traditionally considered that unbounded activations will enable faster convergence and will also help to reach more accurate results.

A polynomial can show anomalous behavior when applied to inputs outside of its definition domain [Matthews and Moschytz, 1994]. The main reason for such a pathological performance is that polynomials are extremely sensitive to the inputs. Such problems can be avoided by allocating polynomial models onto neural network architectures using bounded activations. Bounding the activation polynomials through squashing functions filters the inputs and diminishes the unexpected deteriorating effects from the high-order terms. Using networks with bounded activations can improve the usefulness of polynomials, especially when backpropagation training is applied. The estimation of the weights by least squares techniques, however, may lose accuracy [Marmarelis and Zhao, 1997, Wray and Green, 1994].

Although bounding of the activation polynomials in multilayer PNN is not explicitly shown in the presented research, it can be applied with minor modifications of the given formulae.

1.2.1 Advantages of Polynomial Networks

Polynomial networks are attractive modelling tools from a theoretical, as well as from a practical, point of view. Theoretically, they are: 1) universal approximators with which one may approximate any continuous function on a compact set to an arbitrary precision if there are sufficiently large numbers of terms; 2) mathematically tractable since they assume manipulations, like decompositions and reformulations, which make them flexible for structural identification; 3) probabilistically tractable with specific assumptions for normal and heavy tail distributions; and 4) statistically tractable as they assume standard statistical analysis including testing of their residuals, the covariances between the variables, and the sensitivity to the data sampling.
Practically, the polynomial networks are: 1) computationally tractable, as their training often proceeds on unimodal error surfaces that enable fast and reliable convergence by well-known algorithms, such as ordinary least squares fitting; 2) open-box transparent models which are amenable to easy comprehension and understanding.

1.2.2 Multilayer Polynomial Networks

The subjects of interest here are the multilayer networks with transfer (activation) polynomials in the nodes [Barron, 1988, Elder and Pregibon, 1996, Farlow, 1984, Ivakhnenko, 1971, Müller and Lemke, 2000]. The transfer polynomials are selected according to some predefined criteria from a pool of candidate basis polynomials, and are cascaded hierarchically. The connection scheme suggests that the transfer polynomial outcomes feed-forward to their parent nodes where partial models are composed of received outcomes from the polynomials below and/or input variables. The network output is a high-order polynomial.

The polynomials are discrete analogs of the Volterra series, known as Kolmogorov-Gabor polynomials [Kolmogorov, 1957, Gabor et al., 1961]:

\[ P(x) = a_0 + \sum_{i=1}^{S} a_i x_i + \sum_{i,j}^{S} a_{ij} x_i x_j + \sum_{i,j,k}^{S} a_{ijk} x_i x_j x_k + \ldots \]  

(1.4)

where \( a_i \) are the polynomial coefficients or weights, and \( x_i, x_j, x_k \) are the components of the input vector \( x = [x_1, x_2, \ldots] \).

Popular are the multilayer polynomial networks constructed using the algorithms from the Group Method of Data Handling (GMDH) [Ivakhnenko, 1971, Madala and Ivakhnenko, 1994], which influenced research into similar approaches such as the Polynomial NETwork TRaining algorithm (PNETTR) [Barron, 1988] and the Algorithm for Synthesis of Polynomial Networks (ASPN) [Elder and Pregibon, 1996, Elder and Brown, 2000]. Being familiar with their characteristics gives ideas as to how they work and how they can be improved.

**Group Method of Data Handling.** The GMDH pioneered the development of network algorithms using polynomials as a concept description language. These are constructive algorithms that grow networks of polynomial node functions layer by layer in a bottom-up manner. In this sense, these algorithms perform hill-climbing search in the space of network structures by adding successively to the network nodes with smallest output error [Barron, 1988, Elder and Brown, 2000, Farlow, 1984, Green et al, 1988, Ivakhnenko, 1971, Madala and Ivakhnenko, 1994, Müller and Lemke, 2000, Ng and Lippmann, 1991, Pham and Liu, 1995]. As well as synthesizing the network architecture, they also find
the weights. An important advantage of GMDH is that it infers complex high-order polynomials by cascading hierarchically simple low-order polynomials. This strategy allows us to alleviate to a great degree the fundamental learning problem known as the curse of dimensionality.

Having selected polynomials, one needs to know that the number of terms increases exponentially with the number of inputs. When GMDH grows polynomial networks vertically from the lowest to the highest layer there is no combinatorial explosion of the number of nodes with the increase of the input dimension. For example, a function of maximum degree $S$ can be realized by a GMDH network with quadratic transfer polynomials using $[\log_2 S]$ hidden layers (where $[.]$ is the ceiling function). This efficacy is due to the original representation which keeps the polynomials as compact compositions of simple transfer models. The weights that accommodate the higher-order correlations between the variables appear when the model is expanded, while during processing they are virtual. Whereas the complete multinomial (1.4) of degree $S$ in $m$ variables theoretically has $(S + m)!/(S!m!)$ terms, the GMDH polynomial network is an unexpanded multinomial that practically contains only a small number of these terms. The number of terms is restricted by predefining the network width $K$ and depth $S$ parameters.

The GMDH family contains a wide spectrum of constructive network algorithms [Madala and Ivakhnenko, 1994]: multilayer, combinatorial, recursive, orthogonal, harmonic, etc.. Most popular among them is the multilayer GMDH which iteratively builds strictly layered networks with regular topology. Strictly layered means that the network consists of nodes whose distance from the inputs at any particular layer is the same. Regular means that the outputs of each layer feed directly only the next layer nodes. When the polynomial outcomes feed forward their parent nodes, partial models are composed hierarchically from the received outcomes. An example GMDH network is shown in Figure 1.1.

The complete second-order bivariate transfer (activation) polynomial is used in every network node:

$$p(x_i, x_j) = w_0 + w_1 x_i + w_2 x_j + w_3 x_i x_j + w_4 x_i^2 + w_5 x_j^2$$ (1.5)

where $x_i$ and $x_j$ are variables from the input vector $x_i \in \mathbf{x}, x_j \in \mathbf{x}, i \neq j$, whose indices range up to the preselected dimension $1 \leq i, j \leq d$.

The weights of the polynomial $p(x_i, x_j)$ (1.5) are estimated so as to minimize the sum-squared error:

$$SSE = \sum_{n=1}^{N} (y_n - p(x_{ni}, x_{nj}))^2$$ (1.6)

where the index $n$ in $x_{ni}$ and $x_{nj}$ enumerates the data.
Figure 1.1. Polynomial network of width $K = 5$ constructed by the multilayer GMDH algorithm. The best polynomial model is derived from the leftmost output $y_1$ which makes the model $P(x) = p(p(p(x_1, x_3), p(x_1, x_2)), p(p(x_1, x_2), p(x_3, x_4)))$.

The minimum of this sum-squared error function is found by ordinary least squares (OLS) fitting using the matrix equation:

$$
w = (\Phi^T \Phi)^{-1} \Phi^T y$$

where $w$ is the column vector of weights $w = [w_0, w_1, \ldots, w_m]^T$ whose size is $m + 1$, $\Phi$ is the $N \times (m + 1)$ design matrix of vectors produced by the transfer functions applied to pairs of input variables $(x_{ni}, x_{nj})$, $1 \leq n \leq N$, and $y$ is the $N \times 1$ output column vector.

The design matrix $\Phi$ is made from the data as follows:

$$
\Phi = 
\begin{bmatrix}
\phi_0(x_{1i}, x_{1j}) & \phi_1(x_{1i}, x_{1j}) & \ldots & \phi_m(x_{1i}, x_{1j}) \\
\phi_0(x_{2i}, x_{2j}) & \phi_1(x_{2i}, x_{2j}) & \ldots & \phi_m(x_{2i}, x_{2j}) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_0(x_{Ni}, x_{Nj}) & \phi_1(x_{Ni}, x_{Nj}) & \ldots & \phi_m(x_{Ni}, x_{Nj})
\end{bmatrix}
$$

where $(x_i, x_j)$ is the pair of the $i$-th and the $j$-th input variables, and $\phi_i$ are basis functions introduced for notational convenience.
This design matrix $\Phi$ can be assumed as built of row vectors:

$$\Phi(x_{ni}, x_{nj}) = [\phi_0(x_{ni}, x_{nj}), \phi_1(x_{ni}, x_{nj}), \ldots, \phi_m(x_{ni}, x_{nj})]$$  \hspace{1cm} (1.9)

or alternatively it can be represented using a short notation as follows: $\Phi = \{\phi_l(x_{ni}, x_{nj})\}$, where $1 \leq l \leq m$, $1 \leq n \leq N$, and $1 \leq i, j \leq m$.

Typically, there are considered the following six bivariate basis functions:

$$\phi_0(x_i, x_j) = 1, \phi_1(x_i, x_j) = x_i, \phi_2(x_i, x_j) = x_j, \phi_3(x_i, x_j) = x_i x_j, \phi_4(x_i, x_j) = x_i^2, \text{ and } \phi_5(x_i, x_j) = x_j^2.$$  

Using these basis functions, the transfer polynomial (1.5) can be expressed as follows:

$$p(x_i, x_j) = \phi(x_i, x_j) w$$  \hspace{1cm} (1.10)

where $\phi(x_i, x_j)$ is the $1 \times (m + 1)$ basis vector.

Since bivariate transfer polynomials are used, there are only two connections feeding each node. The weights do not correspond exactly to the connections, rather one may envision that each connection is multiplied to feed a particular transfer polynomial term. That is why the network connectivity is sparse. The number of nodes in each layer, which is the network width, is determined in advance to limit the possibility of a combinatorial explosion of terms. This is part of the greedy search strategy according to which once a node is added to a layer it remains fixed there and cannot be removed anymore. The network width is chosen less than the number of combinations from input variables. At the highest layer there are several results which are sorted, and the best polynomial is obtained from the leftmost node.

The multilayer GMDH algorithm (Table 1.1) starts with the generation of all combinations $c = d(d - 1)/2$ of input variables $(x_i, x_j)$, $x_i \in \mathbf{x}$, $x_j \in \mathbf{x}$, $\mathbf{x} = [x_1, x_2, ..., x_d]$. These combinations of variables are passed to enter the transfer polynomials $p_c^{(1)}(x_i, x_j)$, $1 \leq i, j \leq k = d$, in the first, lowest network layer. Their weights are estimated by OLS fitting (1.7), and the outputs $z_c$ from all $K$ nodes in the first layer are computed $z_c \equiv p_c^{(1)}(x_i, x_j) = \phi(x_i, x_j) w_c$. Next, these candidate nodes are ranked according to the selection criterion, and only the best $k < c$, $1 \leq k \leq K$ of them are retained in the first layer.

The outputs of the first layer activation polynomials are passed forward as pairs of intermediate variables $(z_r = p_r^{(1)}, z_s = p_s^{(1)})$ to feed the second layer nodes $p^{(2)}(z_r, z_s) = \phi(z_r, z_s) w$, $1 \leq r, s \leq K$, where the subscript $^{(2)}$ shows the layer number. All $c = K(K - 1)/2$ combinations of such variables $(z_r, z_s)$, $r \neq s$, are generated and used to estimate the transfer polynomials in the second layer, again by OLS fitting (1.7). This iterative network growing continues until the error stops to decrease.
Introduction

Table 1.1. Summary of the multilayer GMDH algorithm.

<table>
<thead>
<tr>
<th>step</th>
<th>Algorithmic sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Initialization</td>
<td>Data $\mathcal{D} = {(x_n, y_n)}<em>{n=1}^N$ as $d$-dimensional input vectors $x$. Let the current variables be $x_i \in x$, $1 \leq i \leq d$, and $k = d$. Let the network width be $K$, $K &lt; c = d(d - 1)/2$. Let the layer be $(h) = 1$, and the error is $\varepsilon = \text{MaxInt}$. Let the transfer polynomials be: $p(x_i, x_j) = \Phi(x_i, x_j)w$ or $p(x_i, x_j) = w_0 + w_1 x_i + w_2 x_j + w_3 x_i x_j + w_4 x_i^2 + w_5 x_j^2$ and also: $\Phi = {\phi_l(x</em>{ni}, x_{nj})}$, where: $1 \leq l \leq m$.</td>
</tr>
<tr>
<td>2. Network building and training</td>
<td>a) Generate all $c$ combinations $(x_i, x_j)$, $1 \leq i, j \leq k$. b) Make a polynomial $p_c^{(h)}(x_i, x_j)$ from each combination; estimate the polynomial weights $w_c$ by OLS fitting $w_c = (\Phi^T \Phi)^{-1} \Phi^T y$. c) Order the polynomials according to their $f_c^{(h)}$, and choose these $k &lt; c$, $1 \leq k \leq K$, with lower values. d) Take the lowest error from this layer: $\varepsilon^{(h+1)} = \min{f_c^{(h)}}$. e) If the lowest layer criterion is $\varepsilon^{(h+1)} &gt; \varepsilon$ then terminate; else set the overall error $\varepsilon = \varepsilon^{(h+1)}$ and continue. f) The polynomial outputs become current variables: $x_c = p_c^{(h)}$, $1 \leq c \leq k = K$, and $c = K(K - 1)/2$. g) Repeat network building and training with $(h) = (h + 1)$.</td>
</tr>
</tbody>
</table>

This GMDH framework is general as it helps to study not only many of the algorithms from the family, but it also describes the basics of the other popular multilayer polynomial network algorithms such as PNETTR [Barron, 1988] and ASPN [Elder and Pregibon, 1996, Elder and Brown, 2000]. These algorithms differ mainly in the chosen transfer polynomials and in the model selection criteria. They consider similar network growing and weight training approaches to construct high-order multivariate polynomial models (1.4).

GMDH Design Issues. There are several GMDH algorithmic decisions which seriously affect its performance. Polynomial networks are inferred when addressing inductive tasks where the goal is to learn models that generalize beyond the provided data sample. The critical requirement of the discovered models is that they should approximate the general trends in the data and should not overfit the training data. Polynomials that generalize well are accurate and simple, i.e. they have...
parsimonious structure. In order to avoid overfitting, there are several design issues that have to be investigated in order to attain good results: 1) what kind of transfer polynomials to use; 2) what model selection criteria to choose; 3) what data assimilation strategy to follow; and 4) whether complexity tuning can be applied.

The generalization performance depends on the polynomial structure and weights. When the complete quadratic bivariate polynomial is used in all nodes, the GMDH networks tend to overfit the data because the order of the polynomial rapidly increases, and its curvature attempts to pass through all data, thus trying to fit the particularities in them. The early experiments suggested to consider linear bivariate polynomials or incomplete bivariate polynomials, which help to produce more sparse models [Ivakhnenko, 1971, Madala and Ivakhnenko, 1994]. Some elaborated versions even use three-input and cubic transfer polynomials [Elder and Brown, 2000]. There are improvements by passing some inputs directly to higher layer nodes [Barron, 1988]. Section 2.1 offers a set of transfer polynomials for making flexible polynomial networks.

Using different transfer polynomials requires precise tools for analysis of the results, that is there is a need for model selection criteria. The aim is to account more precisely for the contribution to the fit of each polynomial term which is actually a weighted monomial of variables. There exist model criteria that measure not only the approximation error but also other model features like the magnitudes of the weights, the number of the weights when different transfer polynomials are used, the degree of smoothness, etc.. These additional features serve as explicit complexity penalties which help to navigate the search toward parsimonious and smooth polynomials with better forecasting potential. The current research provides various statistical functions [Moody, 1992, Akaike, 1970, Barron, 1988, Barron and Xiao, 1991, Craven and Wahba, 1979], probabilistic functions [Schwartz, 1978, MacKay, 1992a], information-theoretic functions [Mallows, 1973, Akaike, 1973, Rissanen, 1989], risk minimizing functions [Vapnik and Chervonenkis, 1971, Vapnik, 1995, Cherkassky et al., 1999], and dynamic functions [DeBoer and Hogeweg, 1989], that may serve as model selection criteria. Model selection criteria for polynomial networks are investigated in Section 4.1.

The original multilayered GMDH uses a model selection formula that relies on splitting the training set into two nonoverlapping subsets: a training subset which is used for learning of the polynomial weights, and a testing subset which is used for model selection. This data splitting strategy is reasonable when the mean squared error is taken as a model selection criterion. This reduces the risk of overfitting the training examples. However, when the data are split, not all of the information in
the training set is used for learning. There are model selection criteria available which do not require data splitting to achieve overfitting avoidance. The polynomial networks developed here identify their structure and weights from all the data.

The GMDH-type algorithms do not attempt additional reduction of the network complexity to prevent overfitting. Recent empirical investigations show that during network construction and weight training, the polynomial term structure can be reduced in order to derive parsimonious models. Several complexity tuning techniques are explored in the book, including shrinking by regularization of the weights while learning and pruning of insignificant weights. The regularization techniques not only help to make the polynomials smoother, but also help to avoid numerical instabilities in training with imprecise real data. Advanced network pruning algorithms that rely on first-order and second-order derivatives of the network error with respect to the weights, are given in Chapters 6 and 7, while Bayesian pruning is explained in Chapter 8.

**Advantages and Disadvantages of GMDH.** The multilayer polynomial networks from the GMDH family influence the contemporary artificial neural network algorithms with several advantages.

First, they offer adaptive network representations that can be tailored to the given task. They enable us to customize the polynomial network to the problem domain. The network structures are flexible and amenable to topological search. Seeking to discover the most appropriate architecture for the task, these qualities allow us to develop network search algorithms not only using hill-climbing mechanisms, but also using simulated annealing, evolutionary search, etc..

Second, they pioneered the idea of finding the weights in a single step by standard ordinary least squares fitting which eliminates the need to search for their values. This overcomes training problems due to the inability of the learning algorithm to sufficiently identify accurate network weights. The success of least squares fitting guarantees learning at least locally good weights for the transfer polynomials in the network.

Third, these polynomial networks feature relatively sparse connectivity which means that the best discovered networks can be trained fast by backpropagation techniques for artificial neural networks. This implies that if one needs globally optimal weights within the polynomial network, backpropagation can be applied to further improve the weights previously estimated by least squares fitting.

A disadvantage of the GMDH-type algorithms is that they carry out greedy hill-climbing search for the polynomial network structure, and thus only perform exploitation of small neighborhoods from the space of possible topologies. Alternative nodes are discarded early when growing
the network and they do not participate in the later stages of the learning process. These algorithms are not sophisticated enough, and even their improvements (for example, by increasing the number of transfer polynomials, and redefinition of the error criterion and complexity tuning) are not enough to adapt the network sufficiently well to the data.

The GMDH polynomial networks can be further improved in three ways: 1) by conducting global evolutionary search for the proper model architecture; 2) by better coordinating the magnitudes of the weights within the model; and 3) by improving the structure through balancing the model bias and variance with Bayesian techniques.

### 1.3 Evolutionary Search

The evolutionary computation provides approaches for doing global and local search simultaneously. The main evolutionary paradigms are: evolution strategies [Bäck, 1996, Schwefel, 1995, Bäck et al., 2000, Eiben and Smith, 2003, Yao, 1999], evolutionary programming [Fogel et al., 1966, Fogel, 1999], genetic algorithms [Holland, 1975, Goldberg, 1989], and genetic programming (GP) [Koza, 1992, Koza, 1994, Koza et al., 1999, Koza et al., 2003, Riolo and Worzel, 2003]. They conduct probabilistic population-based search which is a powerful tool for broad exploration and local exploitation of the model space. The population-based strategy is an advantage over other global search algorithms such as simulated annealing [Kirkpatrick et al., 1983] and tabu search [Glover, 1989], which works with only one hypothesis at a time, and over algorithms for local search [Atkeson et al., 1997] that perform only narrow examination of the search space. Their stochastic character is an advantage over the heuristic AI [Nilsson, 1980] and machine learning algorithms [Mitchell, 1997, Smirnov, 2001] that also search with one hypothesis.

The research into learning GMDH polynomial networks using evolutionary computation techniques has already developed: genetic algorithms which produce GMDH networks [Kargupta and Smith, 1991], GP systems which learn GMDH networks [Iba and Sato, 1992, Iba and de Garis, 1994, Iba et.al, 1996b, Nikolaev and Iba, 2001a], and evolutionary systems which discover neural trees [Zhang and Mühlenbein, 1995, Zhang et al., 1997]. The genetic algorithms that manipulate GMDH networks can be criticized for the inefficient fixed length genome representation of the polynomials. The fixed size network implementation restricts the topological network manipulations and requires specific operators to preserve the structural relationships in the model. Thus, the fixed length network representation limits the possibility to examine the network structure space well.
There are evolutionary systems that evolve sigma-pi neural trees [Zhang and Mühlenbein, 1995, Zhang et al., 1997], which compose polynomials from sigmoidal and multiplicative activation functions allocated in their nodes. These systems attempt to locate the relevant topology of the neural trees by means of genetic programming. During the population-based search for the polynomial structure, they also conduct a search for the weights by a genetic algorithm. This unfortunately leads not only to slow computations, but to inefficient weights because of the limited capacity of the genetic algorithm to perform numerical search. Another problem is that both evolutionary computation techniques depend on too many parameters which are difficult to tune. The GMDH-type polynomial networks are preferred so as to facilitate not only global structural learning, but also local weight learning.

The sparse connectivity of the GMDH networks makes them resilient and amenable to topological search. The target polynomials produced at the outputs exist within the tree-structured network topology. A single GMDH polynomial is a binary tree-like network which is suitable for tree transformations by classical algorithms. The tree-like polynomial networks have irregular topology; they are not strictly layered because higher layer nodes may be fed directly with input variables. Such tree-like polynomial networks are flexible and assume efficient processing by genetic learning operators. This enables the search engine to detect and discard insignificant terms. Learning by GP allows us to find good tree-like networks, in the sense of terms and maximal order (degree). The first GP system that initiated the research into evolutionary learning of GMDH-type polynomial networks is STROGANOFF [Iba et al,1993, Iba and de Garis, 1994, Iba et.al, 1996].

This reasoning motivates the research into genetic programming with PNN whose principles are established in Chapter 2. The emphasis is on design and implementation of various polynomials represented as tree-structured networks, including algebraic polynomials, orthogonal polynomials, trigonometric polynomials, rational polynomials, local basis polynomials, and dynamic polynomials.

1.3.1 STROGANOFF and its Variants

The research conducted by one of the authors [Iba et al,1993, Iba et.al, 1996] proposed STROGANOFF (Structured Representation On Genetic Algorithms for NON-linear Function Fitting), which integrates GP-based adaptive processes with the GMDH process. STROGANOFF was very successful in solving system identification problems and lead to the development of a whole family of related algorithms.