SOFT COMPUTING AND COMPLEX SYSTEMS

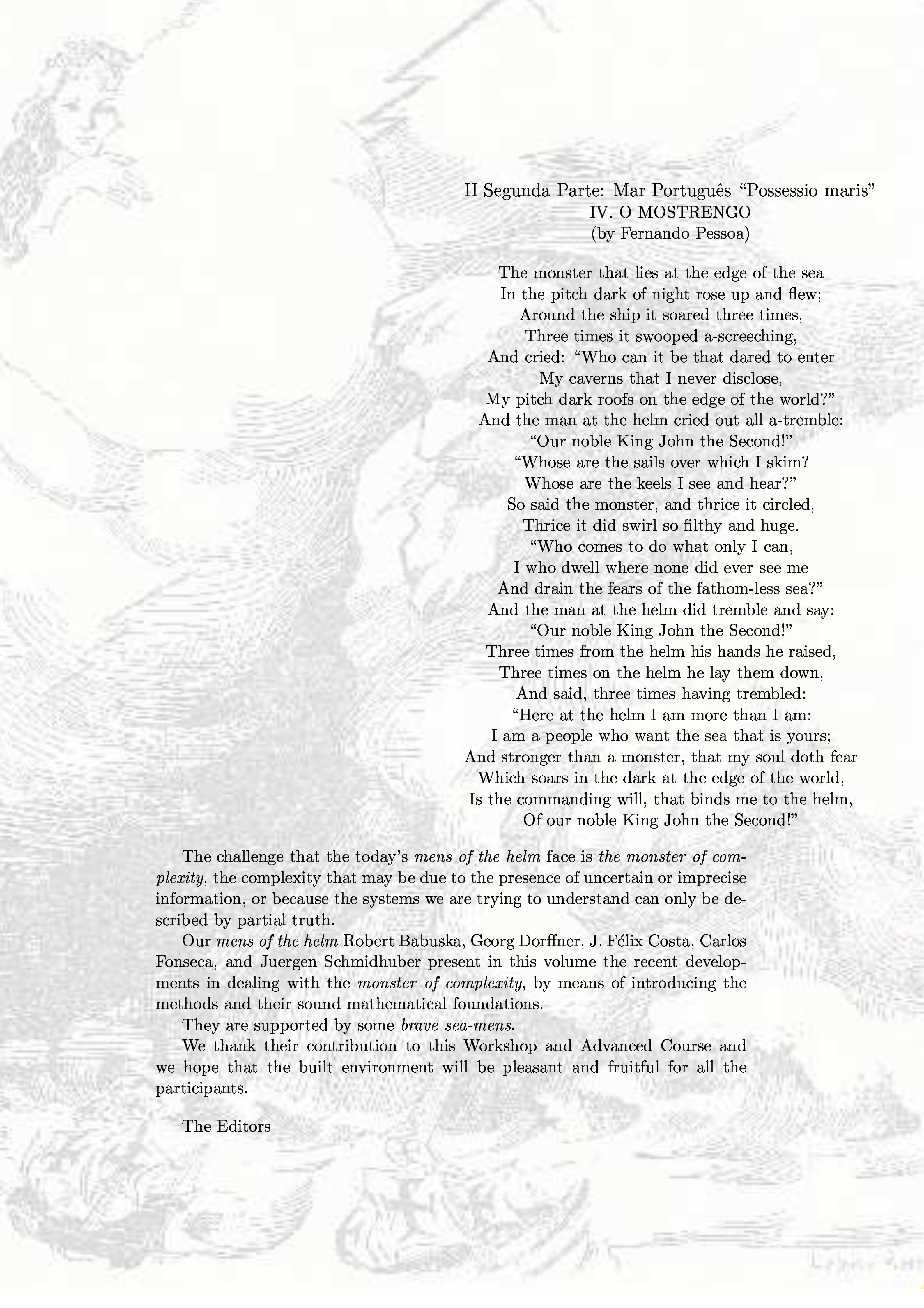
Coimbra, June 23-27, 2003

P. Quaresma, A. Dourado, E. Costa, J. Félix Costa

21

CENTRO INTERNACIONAL de MATEMÁTICA

Contents



II Segunda Parte: Mar Português "Possessio maris"
IV. O MOSTRENGO
(by Fernando Pessoa)

The monster that lies at the edge of the sea
In the pitch dark of night rose up and flew;
Around the ship it soared three times,
Three times it swooped a-screeching,
And cried: "Who can it be that dared to enter
My caverns that I never disclose,
My pitch dark roofs on the edge of the world?"
And the man at the helm cried out all a-tremble:
"Our noble King John the Second!"
"Whose are the sails over which I skim?
Whose are the keels I see and hear?"
So said the monster, and thrice it circled,
Thrice it did swirl so filthy and huge.
"Who comes to do what only I can,
I who dwell where none did ever see me
And drain the fears of the fathom-less sea?"
And the man at the helm did tremble and say:
"Our noble King John the Second!"
Three times from the helm his hands he raised,
Three times on the helm he lay them down,
And said, three times having trembled:
"Here at the helm I am more than I am:
I am a people who want the sea that is yours;
And stronger than a monster, that my soul doth fear
Which soars in the dark at the edge of the world,
Is the commanding will, that binds me to the helm,
Of our noble King John the Second!"

The challenge that the today's *mens of the helm* face is *the monster of complexity*, the complexity that may be due to the presence of uncertain or imprecise information, or because the systems we are trying to understand can only be described by partial truth.

Our *mens of the helm* Robert Babuska, Georg Dorffner, J. Félix Costa, Carlos Fonseca, and Juergen Schmidhuber present in this volume the recent developments in dealing with the *monster of complexity*, by means of introducing the methods and their sound mathematical foundations.

They are supported by some *brave sea-mens.*

We thank their contribution to this Workshop and Advanced Course and we hope that the built environment will be pleasant and fruitful for all the participants.

The Editors

Part I
Lectures

Neuro-Fuzzy Methods for Modeling and Identification

Robert Babuška
Delft University of Technology,Faculty of Information Technology and Systems
Control Systems Engineering Group, P.O.Box 5031, 2600 GA Delft, The Netherlands
e-mail: R.Babuska@dcsc.tudelft.nl

Abstract

Most processes in industry are characterized by nonlinear and time-varying behavior. Nonlinear system identification is becoming an important tool which can be used to improve control performance and achieve robust fault-tolerant behavior. Among the different nonlinear identification techniques, methods based on neuro-fuzzy models are gradually becoming established not only in the academia but also in industrial applications. Neuro-fuzzy modeling can be regarded as a gray-box technique on the boundary between neural networks and qualitative fuzzy models. The tools for building neuro-fuzzy models are based on combinations of algorithms from the fields of neural networks, pattern recognition and regression analysis. In this paper, an overview of neuro-fuzzy modeling methods for nonlinear system identification is given, with an emphasis on the tradeoff between accuracy and interpretability.

Keywords: neuro-fuzzy systems, nonlinear identification, fault-detection and diagnosis, ANFIS network, Takagi–Sugeno fuzzy system.

1 Introduction

The design of control systems is currently driven by a large number of requirements posed by increasing competition, environmental requirements, energy and material costs and the demand for robust, fault-tolerant systems. These considerations introduce extra needs for effective process modeling techniques. Many systems are not amenable to conventional modeling approaches due to the lack of precise, formal knowledge about the system, due to strongly nonlinear behavior, high degree of uncertainty, or time-varying characteristics.

Neuro-fuzzy modeling has been recognized as a powerful tool which can facilitate the effective development of models by combining information from different sources, such as empirical models, heuristics and data. Neuro-fuzzy models describe systems by means of fuzzy *if-then* rules, such as '*If x is small then y is large*' represented in a network structure, to which learning algorithms known from the area of artificial neural networks can be applied. Thanks to this structure, neuro-fuzzy models are to a certain degree transparent to interpretation and analysis, i.e., can be better used to explain solutions to users than completely black-box models such as neural networks.

[1]Based on: R. Babuška. Neuro-fuzzy methods for modeling and identification. In A. Abraham, L.C. Jain, and J. Kacprzyk, editors, *Recent Advances in Intelligent Paradigms and Applications*, pages 161–186. Copyright Springer-Verlag, Heidelberg, 2002. Reprinted with permission.

2 Fuzzy Systems and Neural Networks

Both neural networks and fuzzy systems are motivated by imitating human reasoning processes. In fuzzy systems, relationships are represented explicitly in the form of if–then rules. In neural networks, the relations are not explicitly given, but are ‘coded’ in the network and its parameters. In contrast to knowledge-based techniques, no explicit knowledge is needed for the application of neural nets.

Neuro–fuzzy systems combine the semantic transparency of rule-based fuzzy systems with the learning capability of neural networks. This section gives the background on nonlinear input–output modeling, fuzzy systems and neural nets, which is essential for understanding the rest of this paper.

2.1 Nonlinear System Identification

A wide class of nonlinear dynamic systems with an input u and an output y can be described in discrete time by the NARX (nonlinear autoregressive with exogenous input) input–output model:

$$\begin{aligned} &y(k+1) = f\left(\mathbf{x}(k)\right) \\ &\quad \text{with} \quad \mathbf{x}(k) = \left[y(k)\ \ldots\ y(k-n_y+1)\ u(k)\ \ldots\ u(k-n_u+1)\right]^T \end{aligned} \tag{1}$$

where $y(k+1)$ denotes the output predicted at the future time instant $k+1$ and $\mathbf{x}(k)$ is the *regressor* vector, consisting of a finite number of past inputs and outputs. The dynamic order of the system is represented by the number of lags n_u and n_y. Although for simplicity stated with a scalar input and output, the NARX model can also be used for multivariable systems. In that case, however, the number of regressors usually becomes large and one may prefer the nonlinear state-space description:

$$\begin{aligned} \boldsymbol{\xi}(k+1) &= g(\boldsymbol{\xi}(k), \mathbf{u}(k)) \\ \mathbf{y}(k) &= h(\boldsymbol{\xi}(k)) \end{aligned} \tag{2}$$

The task of nonlinear system identification is to infer the unknown function f in (1) or the functions g and h in (2) from available data sequences $\{(u(k), y(k)) \mid k = 1, 2, \ldots, N\}$.

In black-box modeling, these functions are approximated by some general function approximators such as neural networks, neuro-fuzzy systems, splines, interpolated look-up tables, etc. If the aim of modeling is only to obtain an accurate predictor for y, there is not much difference between these models, as they all can approximate smooth nonlinear systems arbitrarily well. Often, however, besides accurate predictions, one wants to have a model that can be used to learn something about the underlying system and analyze its properties. From this point of view, fuzzy and neuro-fuzzy systems are more transparent than most other black-box techniques.

2.2 Fuzzy Models

A mathematical model which in some way uses fuzzy sets is called a *fuzzy model*. In system identification, rule-based fuzzy models are usually applied. In these models, the relationships between variables are represented by means of if–then rules with imprecise (ambiguous) predicates, such as:

If *heating is high* then *temperature increase is fast*.

This rule defines in a rather qualitative way the relationship between the heating and the temperature in a room, for instance. To make such a model operational, the meaning of the terms ‘high’ and ‘fast’ must be defined more precisely. This is done by using fuzzy sets, i.e., sets where the membership is changing gradually rather than in an abrupt way. Fuzzy sets are defined through their membership functions which map the elements of the considered universe to the unit interval $[0, 1]$. The extreme values 0 and 1 denote complete membership and non-membership, respectively, while a degree between 0 and 1 means partial membership in the fuzzy set. Depending on the structure of the if–then rules, two main types of fuzzy models can be distinguished: the Mamdani (or linguistic) model and the Takagi–Sugeno model.

2.2.1 Mamdani Model.

In this model, the antecedent (if-part of the rule) and the consequent (then-part of the rule) are fuzzy propositions:

$$\mathcal{R}_i\text{: If } x \text{ is } A_i \text{ then } y \text{ is } B_i, \qquad i = 1, 2, \ldots, K\,. \tag{3}$$

Here A_i and B_i are the antecedent and consequent linguistic terms (such as 'small', 'large', etc.), represented by fuzzy sets, and K is the number of rules in the model. The linguistic fuzzy model is useful for representing qualitative knowledge, illustrated in the following example.

Example 1 Consider a qualitative description of the relationship between the oxygen supply to a gas burner (x) and its heating power (y):

$\mathcal{R}_1$: If O_2 flow rate is *Low* then heating power is *Low*.
$\mathcal{R}_2$: If O_2 flow rate is *OK* then heating power is *High*.
$\mathcal{R}_3$: If O_2 flow rate is *High* then heating power is *Low*.

The meaning of the linguistic terms $\{$*Low*, *OK*, *High*$\}$ and $\{$*Low*, *High*$\}$ is defined by membership functions such as the ones depicted in Fig. 1. Membership functions can be defined by the model developer based on prior knowledge or by using data (in this example, the membership functions and their domains are selected quite arbitrarily).

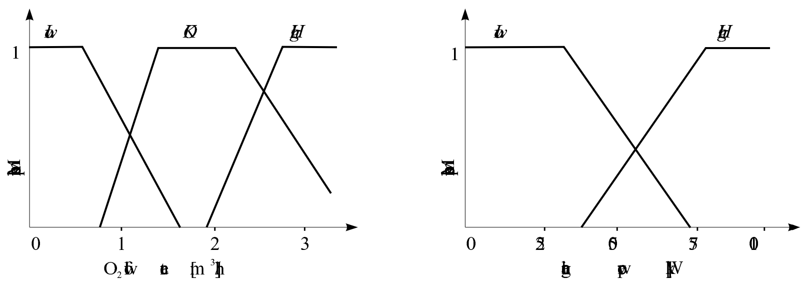


Figure 1: Membership functions for the Mamdani model.

The meaning of the linguistic terms is, of course, not universally given. In this example, the definition of the fuzzy set OK, for instance, may depend on the flow-rate of the fuel gas, the type of burner, etc. When input–output data of the system under study are available, the membership functions can be constructed or adjusted automatically, as discussed later on. Note, however, that the qualitative relationship given by the rules is usually expected to be valid for a range of conditions.

□

2.2.2 Takagi–Sugeno Model.

The Mamdani model is typically used in knowledge-based (expert) systems. In data-driven identification, the model due to Takagi and Sugeno has become popular. In this model, the antecedent is defined in the same way as above, while the consequent is an affine linear function of the input variables:

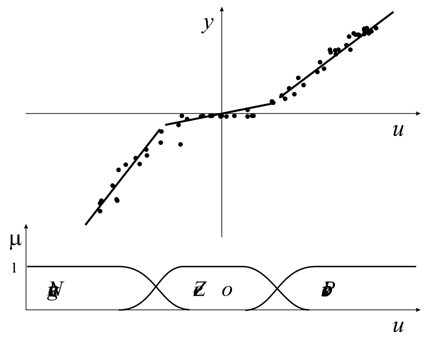
$$\mathcal{R}_i\colon\ \text{If } \mathbf{x} \text{ is } A_i \text{ then } y_i = \mathbf{a}_i^T\mathbf{x} + b_i, \qquad i = 1, 2, \ldots, K, \tag{4}$$

where $\mathbf{a}_i$ is the consequent parameter vector and b_i is a scalar offset. This model combines a linguistic description with standard functional regression: the antecedents describe fuzzy regions in the input space in which the consequent functions are valid. The output y is computed by taking the weighted average of the individual rules' contributions:

$$y = \frac{\sum_{i=1}^{K}\beta_i(\mathbf{x})y_i}{\sum_{i=1}^{K}\beta_i(\mathbf{x})} = \frac{\sum_{i=1}^{K}\beta_i(\mathbf{x})(\mathbf{a}_i^T\mathbf{x} + b_i)}{\sum_{i=1}^{K}\beta_i(\mathbf{x})} \tag{5}$$

where $\beta_i(\mathbf{x})$ is the *degree of fulfillment* of the ith rule. For the rule (4), $\beta_i(\mathbf{x}) = \mu_{A_i}(\mathbf{x})$, but it can also be a more complicated expression, as shown later on. The antecedent fuzzy sets are usually defined to describe distinct, partly overlapping regions in the input space. The parameters $\mathbf{a}_i$ are then (approximate) local linear models of the considered nonlinear system. The TS model can thus be regarded as a smooth piece-wise linear approximation of a nonlinear function or a parameter-scheduling model. Note that the antecedent and consequent variables may be different.

Example 2 Consider a static characteristic of an actuator with a dead-zone and a non-symmetrical response for positive and negative inputs. Such a system can conveniently be represented by a TS model with three rules each covering a subset of the operating domain that can be approximated by a local linear model, see Fig. 2. The corresponding rules are given in the right part of the figure.



$\mathcal{R}_1$: If u is *Negative* then $y_1 = a_1u - b_1$

$\mathcal{R}_2$: If u is *Zero* then $y_2 = a_2u - b_2$

$\mathcal{R}_3$: If u is *Positive* then $y_3 = a_3u - b_3$

$$y = \frac{\mu_{Neg}(u)y_1 + \mu_{Zero}(u)y_2 + \mu_{Pos}(u)y_3}{\mu_{Neg}(u) + \mu_{Zero}(u) + \mu_{Pos}(u)}$$

Figure 2: A Takagi–Sugeno fuzzy model as a piece-wise linear approximation of a nonlinear system.

□

As the consequent parameters are first-order polynomials in the input variables, model (4) is in the literature also called the *first-order TS model.* This is in order to distinguish it from the zero-order TS model whose consequents are simply constants (zero-order polynomials):

$$\mathcal{R}_i\colon \text{ If } \mathbf{x} \text{ is } A_i \text{ then } y_i = b_i, \qquad i = 1, 2, \ldots, K\,. \tag{6}$$

For this model, the input–output equation (5) reduces to:

$$y = \frac{\sum\limits_{i=1}^{K} \beta_i(\mathbf{x}) b_i}{\sum\limits_{i=1}^{K} \beta_i(\mathbf{x})} \tag{7}$$

The above model can also be obtained as a special case of the Mamdani system (3) in which the consequent fuzzy sets degenerate to singletons (real numbers):

$$\mu_{B_i}(y) = \begin{cases} 1, & \text{if } y = b_i, \\ 0, & \text{otherwise}\,. \end{cases} \tag{8}$$

2.2.3 Fuzzy Logic Operators.

In fuzzy systems with multiple inputs, the antecedent proposition is usually represented as a combination of terms with univariate membership functions, by using logic operators ‘and’ (conjunction), ‘or’ (disjunction) and ‘not’ (complement). In fuzzy set theory, several families of operators have been introduced for these logical connectives. Table 1 shows the two most common ones.

Table 1: Commonly used functions for fuzzy logic operators.

	A and B	A or B	not A
Zadeh	$\min(\mu_A, \mu_B)$	$\max(\mu_A, \mu_B)$	$1 - \mu_A$
probabilistic	$\mu_A \cdot \mu_B$	$\mu_A + \mu_B - \mu_A \cdot \mu_B$	$1 - \mu_A$

As an example, consider the commonly used *conjunctive form* of the antecedent, which is given by:

$$\mathcal{R}_i : \text{ If } x_1 \text{ is } A_{i1} \text{ and } x_2 \text{ is } A_{i2} \text{ and } \ldots \text{ and } x_p \text{ is } A_{ip} \text{ then } y_i = \mathbf{a}_i^T \mathbf{x} + b_i \tag{9}$$

with the degree of fulfillment

$$\beta_i(\mathbf{x}) = \min\left(\mu_{A_{i1}}(x_1), \mu_{A_{i2}}(x_2), \ldots, \mu_{A_{ip}}(x_p)\right)$$

or

$$\beta_i(\mathbf{x}) = \mu_{A_{i1}}(x_1) \cdot \mu_{A_{i2}}(x_2) \cdots \mu_{A_{ip}}(x_p)$$

for the minimum and product conjunction operators, respectively. The complete set of rules (9) divide the input domain into a lattice of overlapping axis-parallel hyperboxes. Each of these hyperboxes is a Cartesian product intersection of the corresponding univariate fuzzy sets.

2.2.4 Dynamic Fuzzy Models.

In the modeling of dynamic systems, fuzzy models are used to parameterize of the nonlinear functions f in (1) or g and h in (2). Consider, for instance, the TS NARX model:

$$\begin{aligned}\mathcal{R}_i:\ \text{If } \mathbf{x}(k) \text{ is } A_i \text{ then } y_i(k+1) &= \sum_{j=1}^{n_y} a_{ij}y(k-j+1)\\ &+\sum_{j=1}^{n_u} b_{ij}u(k-j+1)+c_i\end{aligned}$$

where the antecedent regressor $\mathbf{x}(k)$ is generally given by (1), but it may of course contain only some of the past inputs and outputs or even other variables than u and y. Similarly, state-space models can be represented in the TS framework by:

$$\mathcal{R}_i:\ \text{If } \boldsymbol{\xi}(k) \text{ is } A_i \text{ and } \mathbf{u}(k) \text{ is } B_i \text{ then } \begin{cases} \boldsymbol{\xi}_i(k+1) &= \boldsymbol{\Phi}_i\boldsymbol{\xi}(k)+\boldsymbol{\Gamma}_i\mathbf{u}(k)+\mathbf{a}_i\\ \mathbf{y}_i(k) &= \mathbf{C}_i\boldsymbol{\xi}(k)+\mathbf{c}_i\end{cases}$$

An advantage of the state-space modeling approach is that the structure of the model can easily be related to the physical structure of the real system, and, consequently, the model parameters are physically relevant. This is not necessarily the case with input-output models. In addition, the dimension of the regression problem in state-space modeling is often smaller than with input–output models.

2.3 Artificial Neural Networks

Artificial neural nets (ANNs), originally inspired by the functionality of biological neural networks can learn complex functional relations by generalizing from a limited amount of training data. Neural nets can thus serve as black-box models of nonlinear, multivariable static and dynamic systems and can be trained by using input–output data observed on the system. The most common ANNs consist of several layers of simple processing elements called neurons, interconnections among them and weights assigned to these interconnections. The information relevant to the input–output mapping of the net is stored in the weights.

2.3.1 Multi-Layer Neural Network.

A feedforward multi-layer neural network (MNN) has one input layer, one output layer and an number of hidden layers between them. For illustration purposes, consider a MNN with one hidden layer (Fig. 3).

The input-layer neurons do not perform any computations, they merely distribute the inputs x_i to the weights w_{ij}^h of the hidden layer. In the neurons of the hidden layer, first the weighted sum of the inputs is computed

$$z_j=\sum_{i=1}^{p} w_{ij}^h x_i = (\mathbf{w}_j^h)^T\mathbf{x},\quad j=1,2,\dots,m\,. \tag{10}$$

It is then passed through a nonlinear *activation* function, such as the tangent hyperbolic:

$$v_j=\frac{1-\exp(-2z_j)}{1+\exp(-2z_j)},\quad j=1,2,\dots,m\,. \tag{11}$$

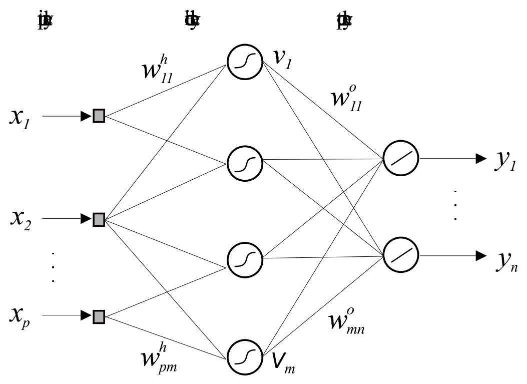


Figure 3: A feedforward neural network with one hidden layer.

Other typical activation functions are the threshold function (hard limiter) and the sigmoidal function. The neurons in the output layer are linear, i.e., only compute the weighted sum of their inputs:

$$y_l = \sum_{j=1}^{h} w^o_{jl} v_j = (\mathbf{w}^o_l)^T \mathbf{x}, \quad l = 1, 2, \ldots, n. \tag{12}$$

Training is the adaptation of weights in a multi-layer network such that the error between the desired output and the network output is minimized. Two steps are distinguished in this procedure:

1. *Feedforward computation.* From the network inputs x_i, the outputs of the first hidden layer are first computed. Then using these values as inputs to the second hidden layer, the outputs of this layer are computed, etc. Finally, the output of the network is obtained.

2. *Weight adaptation.* The output of the network is compared to the desired output. The difference of these two values, the error, is then used to adjust the weights first in the output layer, then in the layer before, etc., in order to decrease the error (gradient-descent optimization). This backward computation is called error backpropagation [1, 2].

A network with one hidden layer is sufficient for most approximation tasks. More layers can give a better fit, but the training takes longer. Choosing the right number of neurons in the hidden layer is essential for a good result. Too few neurons give a poor fit, while too many neurons result in overtraining of the net (poor generalization to unseen data). A compromise is usually sought by trial and error methods.

2.3.2 Dynamic Neural Networks.

A dynamic network can be realized by using a static feedforward network combined with an external feedback connection. The output of the network is fed back to its input through delay operators z^{-1}. This is in fact a realization of the NARX model (1). Fig. 4 shows an example of a first-order system $y(k+1) = f_{\rm nn}(y(k), u(k))$.

Another possibility is to use recurrent networks in which neurons are arranged in one or more layers and feedback is introduced either internally in the neurons, to other neurons in the same layer, or to neurons in preceding layers. Examples of these networks are the Elman network (Fig. 5) or the Hopfield network.

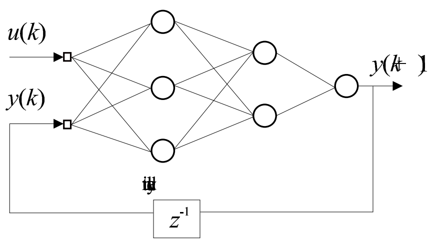


Figure 4: A feedforward neural network representing a dynamic NARX model.

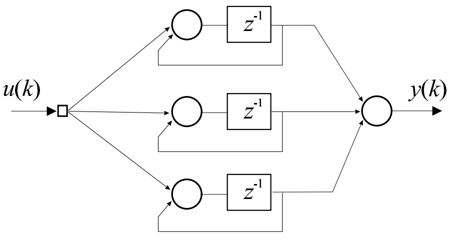


Figure 5: Dynamic network with internal feedback connections (Elman network).

2.3.3 Error Backpropagation.

Consider for simplicity a MNN with one output. A set of N input-output data pairs $\{(\mathbf{x}_k, y_k^*) \mid k = 1, 2, \ldots, N\}$ is available. We represent this set as a matrix $\mathbf{X} \in \mathbb{R}^{N \times p}$, having the input vectors $\mathbf{x}_k$ in its rows, and a column vector $\mathbf{y}^* \in \mathbb{R}^N$, containing the desired outputs y_k^*:

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T, \quad \mathbf{y}^* = [y_1^*, \ldots, y_N^*]^T. \tag{13}$$

The difference between the desired output y^* and the output of the network y is called the error. This error is used to adjust the weights in the net via the minimization of the following cost function:

$$J = \frac{1}{2} \sum_{k=1}^{N} e_k^2 \quad \text{with} \quad e_k = y_k^* - y_k\,.$$

Note that the network's output y is nonlinear in the weights $\mathbf{w}$ (for notational convenience, all the weights are lumped in a single vector $\mathbf{w}$). The training of a MNN is thus a *nonlinear optimization problem* to which various methods can be applied:

- Error backpropagation (first-order gradient).
- Newton, Levenberg-Marquardt methods (second-order gradient).
- Genetic algorithms and many others techniques

First-order gradient methods are based on the following general update rule for the weights:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \alpha(n)\nabla J(\mathbf{w}(n)), \tag{14}$$

where $\mathbf{w}(n)$ is the weight vector at iteration n, $\alpha(n)$ is a (variable) learning rate (a parameter) and $\nabla J(\mathbf{w})$ is the Jacobian of the network

$$\nabla J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial w_1}, \frac{\partial J(\mathbf{w})}{\partial w_2}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_M}\right]^T .$$

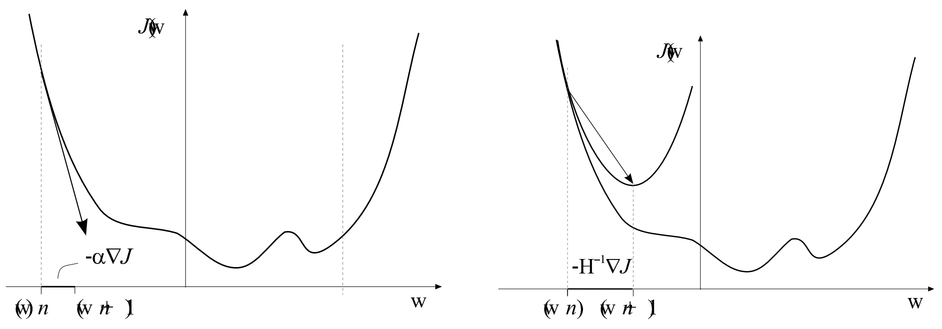
The nonlinear optimization problem is thus solved by using the first term of its Taylor series expansion (the gradient). Second-order gradient methods make use of the second term as well:

$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + \nabla J(\mathbf{w}_0)^T(\mathbf{w} - \mathbf{w}_0) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T \mathbf{H}(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0),$$

where $\mathbf{H}(\mathbf{w}_0)$ is the Hessian. After a few steps of derivations, the update rule for the weights appears to be:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}^{-1}(\mathbf{w}(n))\nabla J(\mathbf{w}(n)) \tag{15}$$

The essential difference between (14) and (15) is the size and the direction of the gradient-descent step, see Fig. 6. Second order methods are usually more effective than first-order ones. Here, we will, however, present the *error backpropagation*, (first-order gradient method) which is easier to grasp. The step toward understanding second-order methods is then quite straightforward.



(a) first-order gradient *(b) second-order gradient*

Figure 6: First-order and second-order gradient-descent optimization.

Let us derive the update laws for the sample-by-sample case, which can be used both for on-line and off-line learning. For simplicity, the sample index k is dropped out of the formulas. We start with the output-layer weights, for which the Jacobian is:

$$\frac{\partial J}{\partial w_j^o} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial w_j^o} = -v_j e, \quad j = 1, 2, \ldots m, \tag{16}$$

and from (14), the update law for the output weights follows:

$$w_j^o(n+1) = w_j^o(n) + \alpha(n) v_j e\,. \tag{17}$$

For the hidden-layer weights, we have the Jacobian:

$$\frac{\partial J}{\partial w_{ij}^h} = \frac{\partial J}{\partial v_j} \cdot \frac{\partial v_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}^h} \tag{18}$$

with the partial derivatives:

$$\frac{\partial J}{\partial v_j} = -w_j^o e, \quad \frac{\partial v_j}{\partial z_j} = \sigma_j'(z_j), \quad \frac{\partial z_j}{\partial w_{ij}^h} = x_i, \tag{19}$$

and from (14), the following update law for the hidden-layer weights is obtained:

$$w_{ij}^h(n+1) = w_{ij}^h(n) + \alpha(n) x_i \sigma_j'(z_j) e w_j^o \,. \tag{20}$$

From this equation, one can see that the error is propagated from the output layer to the hidden layer, which gave rise to the name ‘backpropagation’.

The presentation of the entire data set sample-by-sample is called an *epoch*. Usually, several learning epochs must be applied in order to achieve a good fit. From a computational point of view, it is more effective to present the data set as the whole batch. The backpropagation learning formulas are then applied to vectors of data rather than the individual samples.

2.3.4 Radial Basis Function Network.

The radial basis function (RBF) network is a two-layer network with an architecture depicted in Fig. 7. This network is represented by the following function:

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) \tag{21}$$

where the usual choice for the basis functions $\phi_i(\mathbf{x})$ is the Gaussian function:

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2}\right)$$

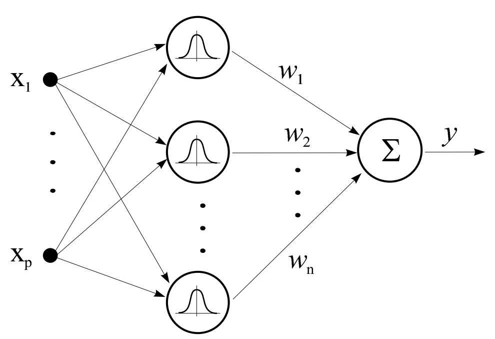


Figure 7: Radial basis function network.

Note that adjustable weights are only present in the output layer. The connections from the input layer to the hidden layer are fixed to unit weights. The free parameters of RBF nets are the output weights w_i and the parameters of the basis functions (centers $\mathbf{c}_i$ and radii σ_i). Since the network’s output

(21) is linear in the weights w_i, these weights can be estimated by least-squares methods. For each data point $\mathbf{x}_k$, first the outputs of the neurons are computed:

$$v_{ki} = \phi_i(\mathbf{x}_k)$$

and put in the matrix $\mathbf{V} = [v_{ki}]$. Introducing the weight vector $\mathbf{w} = [w_1, \ldots, w_n]$, we can write the following matrix equation for the whole data set:

$$\mathbf{y}^* = \mathbf{V}\mathbf{w} \,.$$

The least-square estimate of the weights $\mathbf{w}$ that minimize the network error $\mathbf{e} = \mathbf{y}^* - \mathbf{y}$ is:

$$\mathbf{w} = \left[\mathbf{V}^T\mathbf{V}\right]^{-1}\mathbf{V}^T\mathbf{y}^* \,. \tag{22}$$

The adaptation of the RBF parameters $\mathbf{c}_i$ and σ_i is a nonlinear optimization problem that can be solved by the gradient-descent techniques described above.

3 Neuro-Fuzzy Modeling

At the computational level, a fuzzy system can be seen as a layered structure (network), similar to artificial neural networks of the RBF type [3]. In order to optimize parameters in a fuzzy system, gradient-descent training algorithms known from the area of neural networks can be employed. Hence, this approach is usually referred to as neuro-fuzzy modeling [4, 5, 6].

Consider first a simple example of a zero-order TS fuzzy model with the following two rules:

If x_1 is A_{11} and x_2 is A_{21} then $y = b_1$
If x_1 is A_{12} and x_2 is A_{22} then $y = b_2$

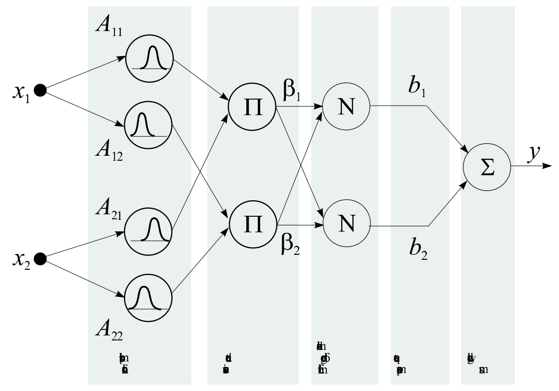


Figure 8: An example of a zero-order TS fuzzy model with two rules represented as a neuro-fuzzy network.

Fig. 8 shows a network representation of these two rules. The nodes in the first layer compute the membership degree of the inputs in the antecedent fuzzy sets. The product nodes Π in the second layer represent the antecedent connective (here the 'and' operator). The normalization node N and the summation node Σ realize the fuzzy-mean operator (5). This system is called ANFIS – Adaptive Neuro-Fuzzy Inference System [5]. Typically, smooth antecedent membership functions are used, such as the Gaussian functions:

$$\mu_{A_{ij}}(x_j; c_{ij}, \sigma_{ij}) = \exp\left(-\frac{(x_j - c_{ij})^2}{2\sigma_{ij}^2}\right). \tag{23}$$

The input–output equation of a general zero-order TS model with the conjunctive form antecedent is:

$$y = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) b_i \quad \text{with} \quad \gamma_i(\mathbf{x}) = \frac{\prod_{j=1}^{p} \exp\left(-\frac{(x_j - c_{ij})^2}{2\sigma_{ij}^2}\right)}{\sum_{i=1}^{K}\prod_{j=1}^{p} \exp\left(-\frac{(x_j - c_{ij})^2}{2\sigma_{ij}^2}\right)} \tag{24}$$

The first-order TS fuzzy model can be represented in a similar fashion. Consider again the example with two rules:

If x_1 is A_{11} and x_2 is A_{21} then $y_1 = a_{11}x_1 + a_{12}x_2 + b_1$
If x_1 is A_{12} and x_2 is A_{22} then $y_2 = a_{21}x_1 + a_{22}x_2 + b_2$

for which the corresponding network is given in Fig. 9.

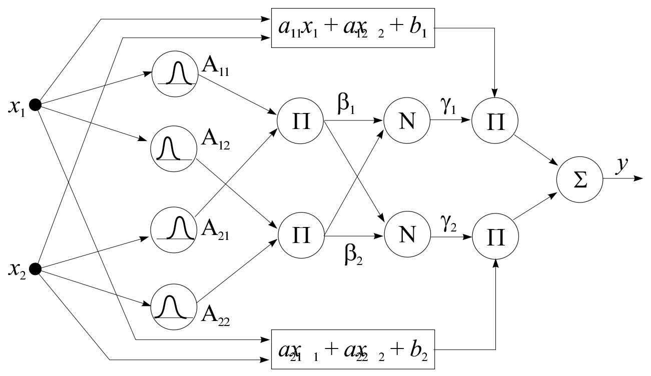


Figure 9: An example of a first-order TS fuzzy model with two rules represented as a neuro-fuzzy network called ANFIS.

The input–output equation of a first-order TS model is:

$$y = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) \left(\mathbf{a}_i^T \mathbf{x} + b_i\right) \tag{25}$$

with $\gamma_i(\mathbf{x})$ given by (24).

3.1 Constructing Neuro-Fuzzy Networks

Both prior knowledge and process data can be used to construct neuro-fuzzy systems. Prior knowledge can be of a rather approximate nature (qualitative, heuristics). Two main approaches to the integration of knowledge and data can be distinguished:

1. Expert knowledge is formulated as a collection of if–then rules. In this way, an initial model is created. The parameters of this model (the membership functions, consequent parameters) are then fine-tuned by using process data.

2. Fuzzy rules (including the associated parameters) are constructed from scratch by using numerical data. In this case, the advantage of using a neuro-fuzzy model is the possibility to interpret the obtained result (which is not possible with truly black-box structures like neural networks). An expert can confront the information stored in the rule base with his own knowledge, can modify the rules, or supply additional ones to extend the validity of the model, etc.

The above techniques can, of course, be combined, depending on the problem at hand.

3.2 Structure and Parameters

The two basic steps in system identification are *structure identification* and *parameter estimation*. The choice of the model's structure is very important, as it determines the flexibility of the model in the approximation of (unknown) systems. A model with a rich structure can approximate more complicated functions, but, at the same time, will have worse generalization properties. Good generalization means that a model fitted to one data set will also perform well on another data set from the same process. In neuro-fuzzy models, the structure selection process involves the following main choices:

- *Selection of input variables.* This involves not only the physical inputs $\mathbf{u}$ but also the dynamic regressors, defined by the input and output lags, n_y and n_u respectively. Prior knowledge, insight in the process behavior and the purpose of the modeling exercise are the typical sources of information for the choice of an initial set of possible inputs. Automatic data-driven selection can then be used to compare different structures in terms of some specified performance criteria.

- *Number and type of membership functions, number of rules.* These two structural parameters are mutually related (for more membership functions more rules must be defined) and determine the level of detail, called the granularity, of the model. The purpose of modeling and the amount of available information (knowledge and data) will determine this choice. Automated, methods can be used to add or remove membership functions and rules.

3.3 Gradient-Based Learning

It is quite straightforward to derive the gradient-descent learning rule for the b_i c_{ij} and σ_{ij} parameters. The procedure is identical to the derivation of the backpropagation formulas (16) through (20). Consider the zero-order ANFIS model, given by rules (6). For the consequent parameters b_i, we have the Jacobian:

$$\frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial b_i} = -\gamma_i e, \quad i = 1, 2, \ldots K, \tag{26}$$

and the update law:

$$b_i(n+1) = b_i(n) + \alpha(n)\gamma_i e, \quad i = 1, 2, \ldots K\,. \tag{27}$$

For the centers and spreads of the Gaussian membership functions (23) we apply the chain rule for differentiation and after some algebra, the following update formulas are obtained:

$$c_{ij}(n+1) = c_{ij}(n) + 2\alpha(n)\gamma_i e[b_i - y]\frac{x_j - c_{ij}}{\sigma_{ij}^2}, \quad i = 1,2,\ldots K, \quad j = 1,2,\ldots,p. \tag{28}$$

and

$$\sigma_{ij}(n+1) = \sigma_{ij}(n) + 2\alpha(n)\gamma_i e[b_i - y]\frac{(x_j - c_{ij})^2}{\sigma_{ij}^3}, \quad i = 1,2,\ldots K, \quad j = 1,2,\ldots,p. \tag{29}$$

The parameter-update equations for the first-order ANFIS model can be derived in a similar fashion.

3.4 Hybrid Learning Techniques

We have already noticed that the output-layer parameters in RBF networks can be estimated by linear least-squares (LS) techniques (22). As LS methods are more effective than the gradient-based update rule (27), hybrid methods are often applied that combine one-shot least-squares estimation of the consequent parameters with iterative gradient-based optimization of the membership functions [7].

In terms of error minimization, the choice of a particular least-squares estimation method is not crucial. If, however, the consequent parameters are to be interpreted as local models, for instance, great care must be taken in the choice of the estimation method. The problem is that the ANFIS models, especially the first-order one, tend to be over-parameterized for most approximation problems. This may lead to numerical problems, over-fitting and meaningless parameter estimates. The following example demonstrates this problem.

Example 3 Assume we wish to approximate a second-order polynomial $y = f_s(u) = 3u^2 - 5u + 6$ by a first-order ANFIS model. First we choose two points t_1 and t_2 and define initial triangular membership functions for $t_1 \le u < t_2$:

$$\mu_{A_1}(u) = \frac{u - t_1}{t_2 - t_1}, \quad \mu_{A_2} = 1 - \mu_{A_1} \tag{30}$$

The model consists of two rules:

$$\mathcal{R}_i\colon \text{ If } u \text{ is } A_i \text{ then } y_i = a_i u + b_i, \qquad i = 1,2$$

By substituting the membership functions (30) into (5), the output of the TS model is obtained (after some elementary algebra):

$$y = \frac{a_1 - a_2}{t_2 - t_1}u^2 + \frac{t_2 a_2 - t_1 a_1 + b_1 - b_2}{t_2 - t_1}u + \frac{t_2 b_2 - t_1 b_1}{t_2 - t_1}$$

As this is a second order polynomial in u, our model can perfectly represent the given nonlinear system. However, it has four free parameters (a_1, a_2, b_1 and b_2) while three are sufficient to fit the polynomial – it is thus over-parameterized. This is a very simple example, but the essence of the over-parameterization problem remains the same when approximating complex unknown systems.

□

To circumvent over-parameterization, the basic least-squares criterion can be combined with additional criteria for local fit, or with constraints on the parameter values. In the following, different techniques are discussed.

3.4.1 Global Least-Squares Estimation.

The global least-squares estimation method yields parameters that minimize the following prediction error criterion:

$$\boldsymbol{\theta} = \arg\min \sum_{k=1}^{N} \left(y_k^* - \sum_{i=1}^{K} \gamma_i(\mathbf{x}_k) \left[\mathbf{x}_k^T 1\right] \boldsymbol{\theta}_i \right)^2 .$$

where $\boldsymbol{\theta}^T = [\boldsymbol{\theta}_1^T \dots \boldsymbol{\theta}_K^T]$ is the concatenation of all the individual rules' parameter vectors. For the data matrices (13), this criterion can be rewritten in a matrix form:

$$\boldsymbol{\theta} = \arg\min \; (\mathbf{y}^* - \boldsymbol{\Lambda}\boldsymbol{\theta})^T (\mathbf{y}^* - \boldsymbol{\Lambda}\boldsymbol{\theta}) \tag{31}$$

with $\boldsymbol{\Lambda} = [\Gamma_1 \boldsymbol{\varphi} \dots \Gamma_K \boldsymbol{\varphi}]$ where $\boldsymbol{\varphi} = [\mathbf{X}\ \mathbf{1}]$ and $\boldsymbol{\Gamma}_i = \text{diag}\ (\gamma_i(\mathbf{x}_1) \dots \gamma_i(\mathbf{x}_N))$, i.e., a diagonal matrix having $\gamma_i(\mathbf{x}_k)$ as its kth diagonal element. The optimal solution of (31) is then directly obtained by using matrix pseudo-inverse:

$$\boldsymbol{\theta} = \left(\boldsymbol{\Lambda}^T \boldsymbol{\Lambda}\right)^{-1} \boldsymbol{\Lambda}^T \mathbf{y}^* . \tag{32}$$

3.4.2 Local Least-Squares Estimation.

While the global solution gives the minimal prediction error, it may bias the estimates of the consequents as parameters of local models. If locally relevant model parameters are required, a weighted least-squares approach applied per rule should be used. This is done by minimizing a set of K weighted local LS criteria:

$$\boldsymbol{\theta}_i = \arg\min \; (\mathbf{y}^* - \boldsymbol{\varphi}\boldsymbol{\theta}_i)^T \boldsymbol{\Gamma}_i (\mathbf{y}^* - \boldsymbol{\varphi}\boldsymbol{\theta}_i) , \quad i = 1, 2, \dots, K \tag{33}$$

for which the solutions are

$$\boldsymbol{\theta}_i = \left(\boldsymbol{\varphi}^T \boldsymbol{\Gamma}_i \boldsymbol{\varphi}\right)^{-1} \boldsymbol{\varphi}^T \boldsymbol{\Gamma}_i \mathbf{y}^*, \quad i = 1, 2, \dots, K .$$

In this case, the consequent parameters of the individual rules are estimated independently of each other, and therefore the result is not influenced by the interactions of the rules. At the same time, however, a larger prediction error is obtained than with global least squares.

Example 4 The application of local and global estimation to the TS model from Example 3 results in the consequent models given in Fig. 10. Note that the consequents estimated by local least squares describe properly the local behavior of the function, but do not give a good fit. For global least squares, the opposite holds – a perfect fit is obtained, but the consequents are not relevant for the local behavior of the system.

$\square$

When interpreting ANFIS models obtained from data, one has to be aware of the trade-offs between local and global estimation. Constrained and multicriteria optimization can also be applied to restrict the freedom in the parameters.

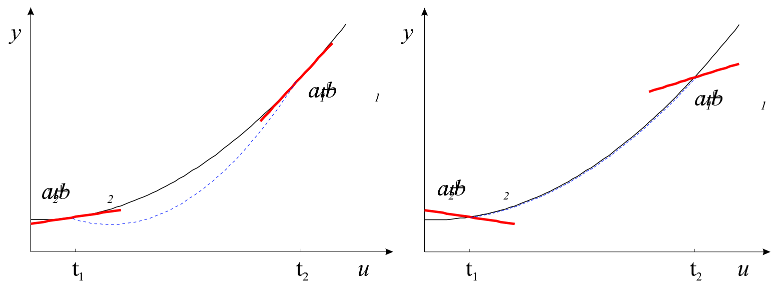


Figure 10: Results of local (left) and global (right) estimation of the consequent parameters. The dashed line is the output of the model.

3.4.3 Constrained Estimation.

Knowledge about the dynamic system such as its stability, minimal or maximal static gain, or its settling time can be translated into convex constraints on the consequent parameters (see Fig. 11). By using input-output data, optimal parameter values are then found by means of quadratic programming, instead of least squares. There are two types of constraints, global and local. Local constraints represent detail knowledge pertaining to each specific rule, while global constraints apply to the entire model and should thus refer to some global system properties such as the overall stability. To see this, realize that the affine TS model (4) can be regarded as one quasi-linear system

$$y = \left(\sum_{i=1}^{K} \gamma_i(\mathbf{x}) \mathbf{a}_i^T \right) \mathbf{x} + \sum_{i=1}^{K} \gamma_i(\mathbf{x}) b_i = \mathbf{a}^T(\mathbf{x})\mathbf{x} + b(\mathbf{x}) \,. \tag{34}$$

with input-dependent ‘parameters’ $\mathbf{a}(\mathbf{x})$, $b(\mathbf{x})$ which are convex linear combinations of the individual consequent parameters $\mathbf{a}_i$ and b_i, i.e.:

$$\mathbf{a}(\mathbf{x}) = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) \mathbf{a}_i, \quad b(\mathbf{x}) = \sum_{i=1}^{K} \gamma_i(\mathbf{x}) b_i \,. \tag{35}$$

This property allows us to define global convex constraints for the entire model. Besides, it also facilitates the analysis of TS models in the framework of polytopic systems (linear differential inclusions). Methods have been developed to design controllers with desired closed loop characteristics and to analyze their stability [8].

3.4.4 Multi-Objective Optimization.

Another possibility is to regularize the estimation by penalizing undesired local behavior of the model. This can dramatically improve the robustness of the construction algorithm, eventually leading to more relevant (interpretable) parameter estimates. One way is to minimize the weighted sum of the global and local identification criteria (31) and (33):

$$\boldsymbol{\theta} = \arg\min \left\{ (\mathbf{y}^* - \boldsymbol{\Lambda}\boldsymbol{\theta})^T (\mathbf{y}^* - \boldsymbol{\Lambda}\boldsymbol{\theta}) + \sum_{i=1}^{K} \delta_i \left(\mathbf{y}^* - \boldsymbol{\varphi}\boldsymbol{\theta}_i\right)^T \boldsymbol{\Gamma}_i \left(\mathbf{y}^* - \boldsymbol{\varphi}\boldsymbol{\theta}_i\right) \right\} .$$

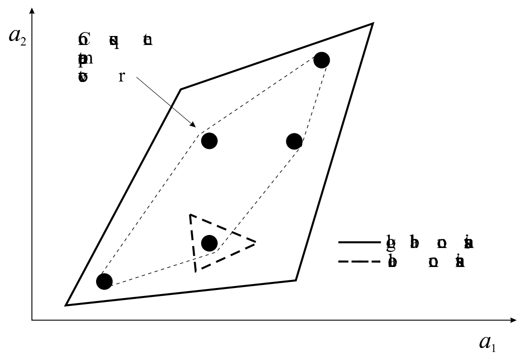


Figure 11: Convex constraints on the consequent parameters.

The weighting parameters $\delta_i \geq 0$ parameterize the set of Pareto-optimal solutions of the underlying multi-objective optimization problem and thus determine the tradeoff between the possibly conflicting objectives of global model accuracy and local interpretability of the parameters.

3.5 Initialization of Antecedent Membership Functions

For a successful application of gradient-descent learning to the membership function parameters, good initialization is important. Several initialization methods are briefly reviewed in this section.

3.5.1 Template-Based Membership Functions.

With this method, the domains of the antecedent variables are *a priori* partitioned by a number of membership functions. These are usually evenly spaced and shaped. The rule base is then established to cover all the combinations of the antecedent terms. A severe drawback of this approach is that the number of rules in the model grows exponentially. Furthermore, if no knowledge is available as to which variables cause the nonlinearity of the system, all the antecedent variables are usually partitioned uniformly. However, the complexity of the system's behavior is typically not uniform. Some operating regions can be well approximated by a local linear model, while other regions require a rather fine partitioning. In order to obtain an efficient representation with as few rules as possible, the membership functions must be placed such that they capture the non-uniform behavior of the system.

3.5.2 Discrete Search Methods.

Iterative tree-search algorithms can be applied to decompose the antecedent space into hyper-rectangles by axis-orthogonal splits. In each iteration, the region with the worst local error measure is divided into two halves (or other portions). Splits in all dimensions of the input are tested and the one with the highest performance improvement is chosen. This successive partitioning stops when a specified error goal is met or when the desired number of rules is reached. The first four steps of such an algorithm are illustrated in Fig. 12. An advantage of this approach is its effectiveness for high-dimensional data and the transparency of the obtained partition. A drawback is that the tree building procedure is sub-optimal (greedy) and hence the number of rules obtained can be quite large [9].

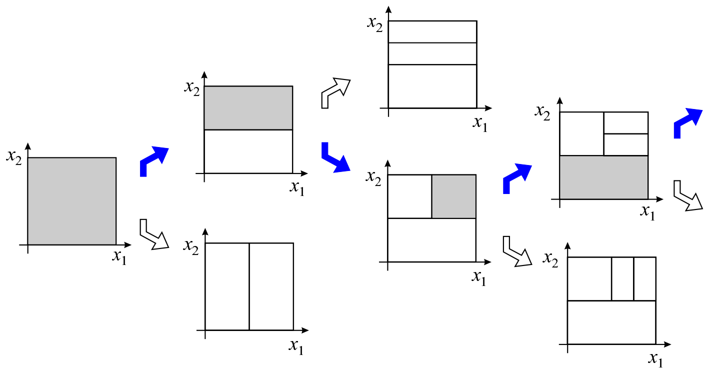


Figure 12: Antecedent space decomposition by a heuristic search algorithm. The dark areas represent rules with the worst local fit in the given step.

3.5.3 Fuzzy Clustering.

Construction methods based on fuzzy clustering originate from data analysis and pattern recognition, where the concept of fuzzy membership is used to represent the degree to which a given data object is similar to some prototypical object. The degree of similarity can be calculated by using a suitable distance measure. Based on the similarity, data vectors are clustered such that the data within a cluster are as similar as possible, and data from different clusters are as dissimilar as possible.

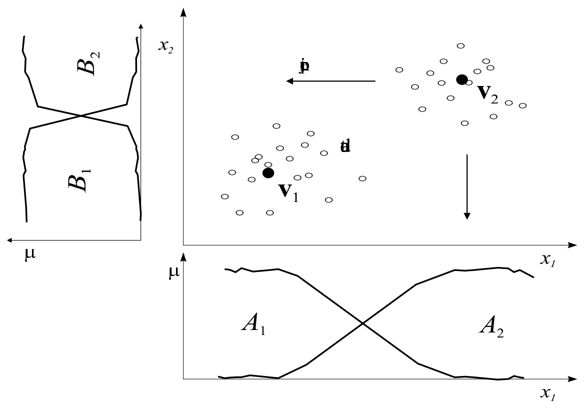


Figure 13: Identification of membership functions through fuzzy clustering.

Fig. 13 gives an example of two clusters in $\mathbb{R}^2$ with prototypes $\mathbf{v}_1$ and $\mathbf{v}_2$. The partitioning of the data is expressed in the *fuzzy partition matrix* $\mathbf{U} = [\mu_{ij}]$ whose elements are the membership degrees of the data vectors $\mathbf{x}_k$ in the fuzzy clusters with prototypes $\mathbf{v}_j$. The antecedent membership functions are then extracted by projecting the clusters onto the individual variables. For the initialization of first-order ANFIS models, the prototypes can be defined as linear subspaces or the clusters are ellipsoids with adaptively determined shape. The number of clusters in the data can either be determined *a priori* or sought automatically by using cluster validity measures and merging techniques [10].

4 Simulation Examples

In this section, two simulation examples are given to illustrate several important issues related to the training of neuro-fuzzy systems. The first example is a simple fitting problem of a univariate static function. It demonstrates the typical construction procedure of a neuro-fuzzy model. Numerical results show that an improvement in performance is achieved at the expense of obtaining if-then rules that are not completely relevant as local descriptions of the system.

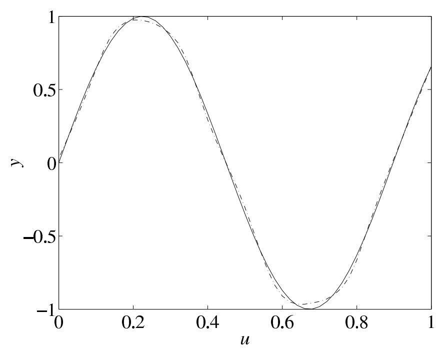
The second example, the modeling of a nonlinear dynamic system, illustrates that the performance of a neuro-fuzzy model does not necessarily improve after training. This is due to overfitting which in the case of dynamic systems can easily occur when the data only sparsely cover the domains.

4.1 Static Function

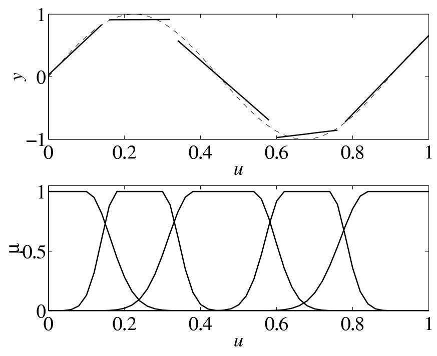
Let us approximate a univariate function $y = \sin(u)$ by the ANFIS model with linear consequent functions. We choose the number of rules to be five and construct an initial model by clustering the data $U \times Y$, using a methodology based on the Gustafson-Kessel algorithm [10]. The following rules are obtained:

$$\begin{array}{ll}
\text{If } u \text{ is } A_1 \text{ then} & y = 5.721u + 0.030 \\
\text{If } u \text{ is } A_2 \text{ then} & y = 0.035u + 0.904 \\
\text{If } u \text{ is } A_3 \text{ then} & y = -5.302u + 2.380 \\
\text{If } u \text{ is } A_4 \text{ then} & y = 0.734u - 1.413 \\
\text{If } u \text{ is } A_5 \text{ then} & y = 6.283u - 5.623
\end{array}$$

The fit of the function with this initial model is shown in Fig. 14a. The membership functions and the corresponding local models are given in Fig. 14b. The membership functions are denoted from left to right by A_1 through A_5.



(a) The data (solid) and the model (dashed-dotted).



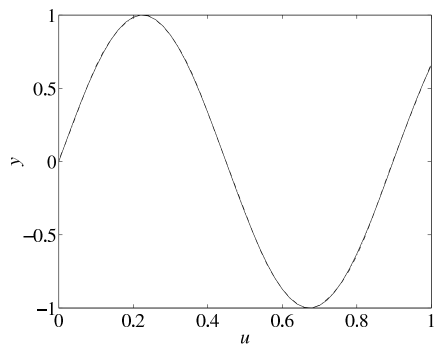
(b) Local models (top) and membership functions (bottom).

Figure 14: Approximation by the initial TS fuzzy model. The root mean squared error is RMS= 0.0258.

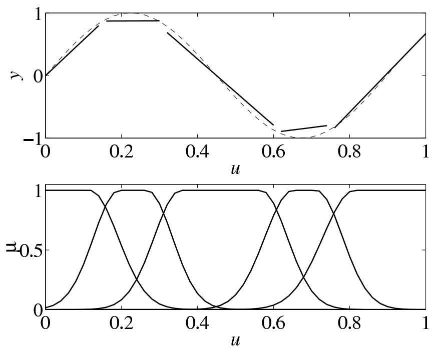
Note that this initial model can easily be interpreted in terms of the local behavior (the rule consequents) and it is already reasonably accurate (the root mean squared error is RMS= 0.0258). However, by using the ANFIS method, the model parameters can be fine-tuned and the approximation accuracy can be significantly improved. A model obtained after 100 learning epochs of hybrid learning using the `anfis` function of the MATLAB Fuzzy Logic Toolbox [7] is described by the following fuzzy rules:

$$\begin{array}{ll}
\text{If } u \text{ is } A'_1 \text{ then } & y = 5.275u + 0.065 \\
\text{If } u \text{ is } A'_2 \text{ then } & y = 0.442u + 0.899 \\
\text{If } u \text{ is } A'_3 \text{ then } & y = -3.206u + 1.405 \\
\text{If } u \text{ is } A'_4 \text{ then } & y = 0.977u - 1.693 \\
\text{If } u \text{ is } A'_5 \text{ then } & y = 5.062u - 4.388
\end{array}$$

The performance has improved to that degree that no approximation error is visible (Fig. 15a). The root mean squared error is now RMS= 0.0011, which is about 23 times better than the initial model. The membership functions have only been slightly modified, but the change in the local models is more apparent, see Fig. 15b.



(a) The data (solid) and the model (dashed-dotted).



(b) Local models (top) and membership functions (bottom).

Figure 15: Approximation by the TS fuzzy model fine-tuned by ANFIS. The root mean squared error is RMS= 0.0011.

A closer look at the consequents, both graphically and in terms of the numerical values, reveals that after learning, the local models are much further from the true local description of the function. To quantify this, we can compute the difference between the consequent parameters of the fuzzy models, denoted by $\boldsymbol{\theta}$, and the 'true' local estimates $\boldsymbol{\theta}_0$, computed by least squares for the data in $\mathrm{core}(A_i)$. For the initial fuzzy model, we have $||\boldsymbol{\theta} - \boldsymbol{\theta}_0|| = 1.81$, while for the ANFIS model we have $||\boldsymbol{\theta} - \boldsymbol{\theta}_0|| = 5.30$. The rules of the fine-tuned neuro-fuzzy model are thus less accurate in describing the system locally. This contradiction between local and global approximation accuracy is inherent to TS fuzzy systems with linear consequents [11] and thus also to the ANFIS network. Great care must be exercised when one attempts to interpret rules in trained neuro-fuzzy models.

4.2 pH Neutralization Process

A neutralization tank with three influent streams (acid, buffer and base) and one effluent stream is considered. The identification and validation data sets are obtained by simulating the model by Hall and Seborg [12] for random changes of the influent base stream flow rate Q. The influent buffer stream and the influent acid stream are kept constant. The output is the pH in the tank. The identification data set, containing $N = 499$ samples with the sampling time of 15 s, is shown in Fig. 16. This data set was obtained from [13].

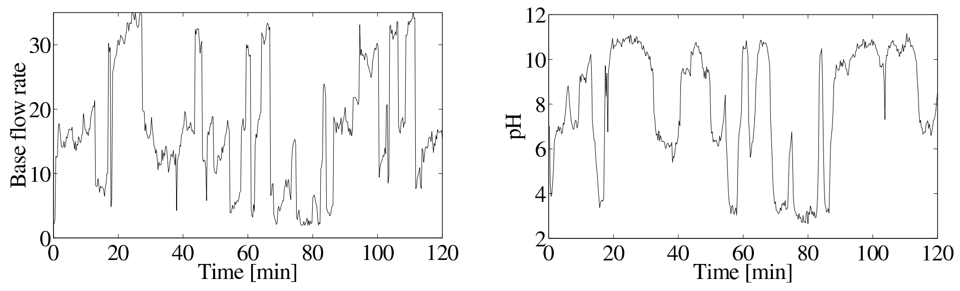


Figure 16: Identification data.

The process is approximated as a first-order discrete-time NARX model:

$$\text{pH}(k+1) = f(\text{pH}(k), Q(k)),$$

where k denotes the sampling instant, and f is an unknown relationship approximated by a neuro-fuzzy model. Based on prior knowledge about the process, it was decided to include only $Q(k)$ in the antecedent (it is known that the main source of nonlinearity is the titration curve, which is the steady-state characteristic relating Q to pH). The number of membership functions (and thus also rules) was set to three. The initial membership functions were evenly spread over the domain, as shown in the left panel of Fig. 17.

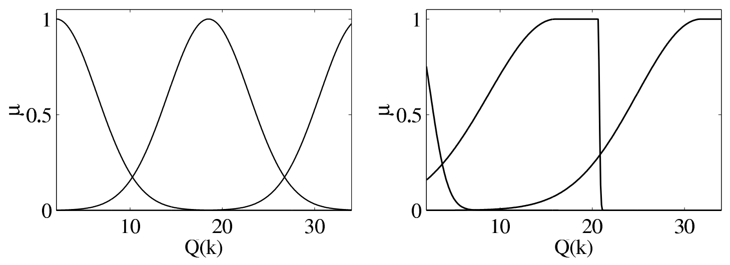


Figure 17: Membership functions before (left) and after training (right). The membership functions are denoted from left to right by 'Low', 'Medium' and 'High'.

The initial rule base, with the consequent estimated by weighted local least squares (33), is given by:

If $Q(k)$ is *Low* then $\text{pH}(k+1) = 0.83\text{pH}(k) + 0.09Q(k) + 0.03$
If $Q(k)$ is *Medium* then $\text{pH}(k+1) = 0.83\text{pH}(k) + 0.09Q(k) + 0.10$
If $Q(k)$ is *High* then $\text{pH}(k+1) = 0.46\text{pH}(k) + 0.02Q(k) + 5.09$

After 1000 epochs of hybrid learning using the ANFIS function of the MATLAB Fuzzy Logic Toolbox [7], the following rule base has been obtained:

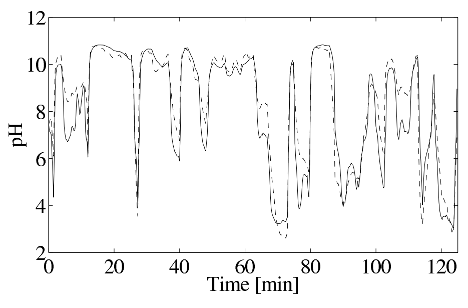
$$\begin{array}{lll}
\text{If } Q(k) \text{ is } \textit{Low}' & \text{then} & \mathrm{pH}(k+1) = 0.37\mathrm{pH}(k) - 0.05Q(k) + 2.14 \\
\text{If } Q(k) \text{ is } \textit{Medium}' & \text{then} & \mathrm{pH}(k+1) = 0.91\mathrm{pH}(k) + 0.06Q(k) - 0.23 \\
\text{If } Q(k) \text{ is } \textit{High}' & \text{then} & \mathrm{pH}(k+1) = 0.40\mathrm{pH}(k) + 0.03Q(k) + 5.63
\end{array}$$

Note that the consequent model in the first rule has a negative coefficient for $Q(k)$. As this is a physically impossible value, not interpretation can be given to these parameters and this trained model has become a complete black-box. Also notice in Fig. 17 that the membership functions were adjusted in a very peculiar way by the gradient-descent optimization method.

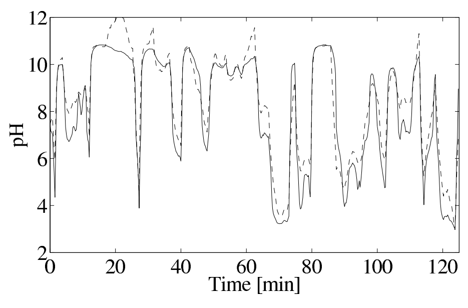
Table 2: Comparison of RMS before and after training.

	before training	after training
training data set	0.90	0.82
validation data set	0.81	0.89

Table 2 shows that while the numerical performance in terms of the root-mean-square error improved for the training data, it has become worse for the validation data. This is a typical example of overtraining. This can also be observed in Fig. 18 where the predictions generated by the model are less accurate after training than before. Clearly, this kind of behavior is difficult to predict for a new problem at hand. The importance proper model validation can thus hardly be overemphasized.



(a) initial model



(b) after training

Figure 18: Performance of the initial and trained model on the validation data set (solid – data, dashed – model).

5 Concluding Remarks

Neuro-fuzzy modeling is a flexible framework in which different paradigms can be combined, providing, on the one hand, a transparent interface with the designer and, on the other hand, a tool for accurate nonlinear modeling and control. The rule-based character of neuro-fuzzy models allows for the analysis and interpretation of the result. Conventional methods for numerical validation can be complemented by human expertise, which often involves heuristic knowledge and intuition.

A drawback of neuro-fuzzy modeling is that the current techniques for constructing and tuning fuzzy models are rather complex, and their use requires specific skills and knowledge. In this sense, it will probably never become a 'one-button', fully automated identification technique. Neuro-fuzzy modeling should rather be seen as an interactive method, facilitating the active participation of the user in a computer-assisted modeling session. This holds, to a certain degree, also for other, more established methods. Modeling of complex systems will always remain an interactive approach.

Further Reading. More details on the different methods and tools can be found in references [4, 6, 14], among others. A large number of works are being regularly published in fuzzy systems oriented journals (IEEE Transactions on Fuzzy Systems, Fuzzy Sets and Systems) and also IEEE Transactions on Systems Man and Cybernetics.

Software. Various tools were developed for MATLAB.$^{\text{TM}}$ Examples are the Fuzzy Logic Toolbox for MATLAB (http://www.mathworks.com/products/fuzzylogic) and the Fuzzy Modeling and Identification Toolbox developed by the author of this chapter (http://Lcewww.et.tudelft.nl/~babuska). These tools were used to generate the solutions of the examples in this chapter.

References

[1] P.J. Werbos. *Beyond regression: new tools for prediction and analysis in the behavior sciences*. PhD Thesis, Harvard University, Committee on Applied Mathematics, 1974.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[3] J.-S.R. Jang and C.-T. Sun. Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 4(1):156–159, January 1993.

[4] M. Brown and C. Harris. *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall, New York, 1994.

[5] J.-S.R. Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transactions on Systems, Man & Cybernetics*, 23(3):665–685, May 1993.

[6] J.-S.R. Jang, C.-T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing; a Computational Approach to Learning and Machine Intelligence*. Prentice-Hall, Upper Saddle River, 1997.

[7] The Mathworks. Fuzzy logic toolbox for use with MATLAB. User's guide, version 2, The Mathworks, Inc., Natick, MA, USA, June 2001.

[8] K. Tanaka, T. Ikeda, and H.O. Wang. Fuzzy regulators and fuzzy observers: relaxed stability conditions and LMI-based designs. *IEEE Transactions on Fuzzy Systems*, 6(2):250–265, February 1998.

[9] O. Nelles, A. Fink, R. Babuška, and M. Setnes. Comparison of two construction algorithms for Takagi–Sugeno fuzzy models. *International Journal of Applied Mathematics and Computer Science*, 10(4):835–855, 2000.

[10] R. Babuška. *Fuzzy Modeling for Control*. Kluwer Academic Publishers, Boston, USA, 1998.

[11] J. Abonyi and R. Babuška. Local and global identification and interpretation of parameters in Takagi–Sugeno fuzzy models. In *Proceedings 9th IEEE International Conference on Fuzzy Systems*, pages 835–840, San Antonio, USA, May 2000.

[12] R.C. Hall and D.E. Seborg. Modelling and self-tuning control of a multivariable pH neutralization process. Part I: Modelling and multiloop control. In *Proceedings American Control Conference*, volume 2, pages 1822–1827, Pittsburgh, U.S.A., 1989.

[13] T.A. Johansen. *Operating Regime Based Process Modelling and Identification*. PhD dissertation, The Norwegian Institute of Technology – University of Trondheim, Trondheim, Norway, November 1994.

[14] H. Hellendoorn and D. Driankov, editors. *Fuzzy Model Identification: Selected Approaches*. Springer, Berlin, Germany, 1997.

Neural Computation and Applications in Time Series and Signal Processing

Georg Dorffner
Dept. of Medical Cybernetics and Artificial Intelligence
University of Vienna
and
Austrian Research Institute for Artificial Intelligence

Abstract

This paper presents a brief overview on models from neural computation and their applicability to problems in time series and signal processing. Much focus is put on pointing out the relationships between neural networks and more traditional methods for time series analysis. For more details on some of the advanced models, the reader is refered to the bibliography.

1 Neural computation

Neural computation in pattern recognition refers to an array of models and methods originating in the first attempts to formalise information processing in the brain. A typical *neural network*, depicted in figure 1, is of the form

$$x_j^{out} = \sum_{l=1}^{k} v_{lj} f(\sum_{i=1}^{n} w_{il} x_i^{in}) \quad (1)$$

or

$$x_j^{out} = \sum_{l=1}^{k} v_{lj} f(\sqrt{\sum_{i=1}^{n} (w_{il} - x_i^{in})^2}) \quad (2)$$

where x_i^{in} and x_j^{out} stand for input and output values, respectively, and v_{lj} and w_{il} stand for the so-called *weights*, or degrees of freedom, of the models. Equation 1 corresponds to the well-known *multilayer perceptron* (MLP), when f is a so-called *sigmoid function*, such as $f(x) = \frac{1}{1-e^{-x}}$ or $f(x) = \tanh(x)$. Equation 2 corresponds to the *rdial basis function network* (RBFN), with f, for instance, being the Gaussian function $f(x) = \exp(-x^2)$.

The main strength of this type of neural network is that it can approximate any arbitrary nonlinear function $\vec{x}^{out} = F(\vec{x}^{in})$, provided the number

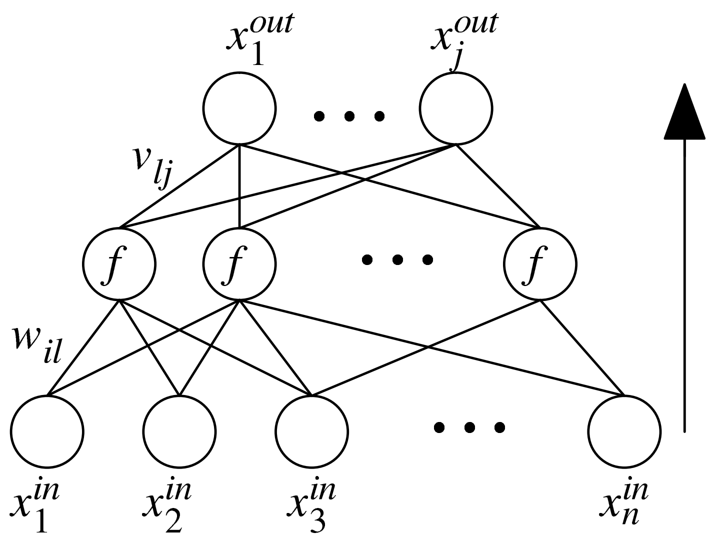


Figure 1: A generic multilayer neural network for function approximation

k of so-called *hidden units* is large enough ([Hornik et al. 1989]). Thus they are called *universal function approximators.* Approximation is done by a weighted superposition of simple nonlinear functions (the sigmoid or the Gaussian). [Bishop 1995] calls this a *semi-parametric* estimation of a function, since on one hand the function F is parameterised through the weights, but within the capacity of a given network little has to be assumed about the shape of the function (similar to non-parametric estimates).

When it comes to pattern recognition, neural networks have little to do with the brain, despite their original motivation (the weighted sum in equation 1 and the sigmoid were introduced to roughly mimic the potential accumulation and firing behavior of biological neurons). Instead, they are advanced methods for nonlinear exploratory and inductive statistics. *Learning* in neural networks (also called training) must be seen in the same realm. Weights are usually derived during model estimation in a *maximum likelihood* framework, using data from a so-called training set. In its simplest form (see below), maximizing likelihood amounts to minimizing the *summed squared error* $E = \sum_{i=1}^{N}(x_i^{\text{target}} - x_i^{\text{out}})^2$, where x^{target} is given by the training samples. Two types of applications are usually distinguished: *regression* – i.e. estimating continuous output values – and *classification.* For more details on neural networks for pattern recognition, see [Bishop 1995].

Neural computation nowadays encompasses a much wider array of methods than the original neural networks. They include *support vector machines* (for which the lower part of the network is kept fixed through the choice of proper *kernel functions*), *independent component analysis* for source separation, *Gaussian* and other *mixture models*, and many types of unsupervised learning methods. What they all have in common is that

- there is a strong focus on nonlinearity
- complexity is approximated by superpositions of simpler building blocks
- thus, focus is on semi-parametric methods

In this paper, we restrict ourselves to universal approximators (such as the MLP or RBFN), since they are mostly used in time series and signal processing. An important extension, which is of interest with respect to time series, are so-called *recurrent* neural networks. They are characterized by the introduction of feedback connections from hidden or output units to the input layer (see figure 7 and 8 below).

2 Time Series Processing

Time series processing is the field of pattern recognition and analysis of time-varying data. A typical time series problem is given by a vector of observable measurements at consecutive points in time t:

$$\vec{x}_t, t = 0, 1, ... \tag{3}$$

If $\vec{x}$ is indeed a vector, i.e. more than one variable is observed, one speaks of a *multivariate time series*: if it is a scalar, the time series is called *univariate*. The representation of a time-varying set of variables in equation 3 makes several important assumptions:

- Time is discrete, meaning that the observables are measured only at discrete points in time. This is different from so-called *continuous time* models. In signal processing terms one could speak of "sampling" the original process observable into the variable vector $\vec{x}_t$.
- Points in time for measurement are equi-distant, meaning that there is a constant time interval between points of measurement.

We restrict our discussion in this paper to this kind of discrete-time processes, since they are the most common models for practical applications and are most amenable to analysis with common methods from neural computation.

The length of the time interval depends on the type of application. Typical examples of time series are

- sampled acoustic or biosignals (time interval usually milli- or nanoseconds)

- measurments of oxygen saturation in intensive care (typically seconds)
- measurements of process parameters in industry (typically seconds or minutes)
- temperature measurements in meteorology (typically hours)
- stock or option prices in the financial markets (typically days)
- econometric measures like inflation rate (typically weeks)
- the number of sunspots (years)

This overview of possible applications highlights that there is no principle difference between what is called *signal processing* and what is called *time series processing*. Formally the methods are the same or very similar, only the time interval (short for signals, longer for time series) and the focus of the application usually differs. In signal processing, typical problems are

- Filtering of the signal, i.e. changing its general characteristics
- Source separation, i.e. considering the signal as a mixture of unknown sources and dividing it into them

Typical problems in the domain of time series processing are

- Forecasting, i.e. estimating the future development of the time series
- Noise modeling, i.e. estimating the stochastic variability of the time series

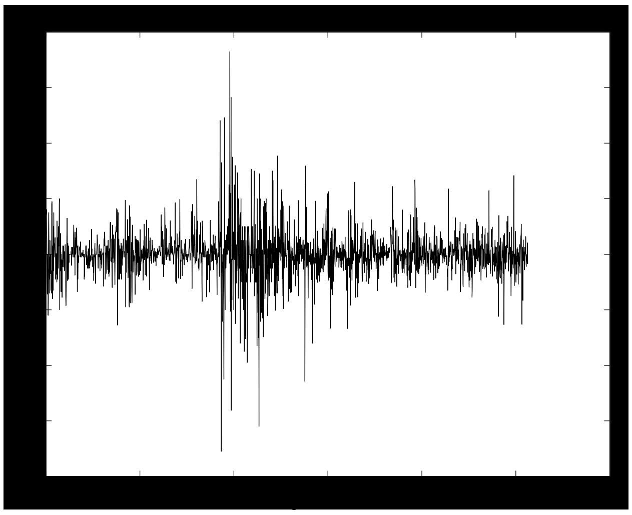
Problems common to both domains are

- Pattern recognition, i.e. recognising typical wave forms or subsequences
- Modeling of the underlying process, i.e. finding a mathematical model that describes the generating process underlying the observable variables.

In this paper, the focus will be on modeling and forecasting.

Figure 2 depicts two typical time series, which will be used as examples in this paper – one from finance, namely returns from the daily Austrian stock exchange index ATX, and one from astronomy, namely the annual number of observed sunspots since the 18th century (this is a well-known benchmark time series).

The former is a typical case of a rather noisy time series. It is derived from differencing the original time series of daily index values by calculating



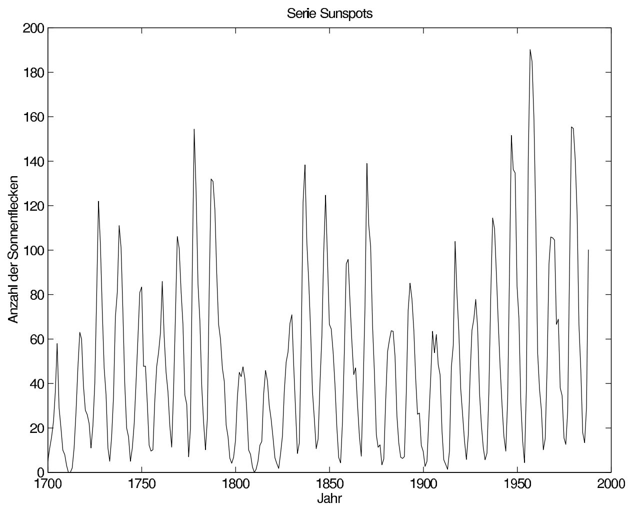


Figure 2: Two typical time series: Returns from the Austrian stock exchange index ATX (left), and the annual number of subspots since 1770 (right).

$$r_t = x_t - x_{t-1} \tag{4}$$

Differencing is a proper preprocessing method for many applications in order to remove *trends* and some *instationarities*. In financial applications this has the direct interpretation as "returns", i.e. potential wins or losses one would get when trading the commodity.

The latter is a typical case of a more structured time series. What can be observed are distinct so-called *seasonalities*, i.e. recurring periodic patterns in the time series. For optimal modeling, such seasonalities should also be removed, e.g. by the following type of differencing:

$$y_t = x_t - x_{t-s} \tag{5}$$

where s is the time interval between seasonal peaks.

3 Forecasting as modeling

In this section we will see that forecasting time series is akin to finding a model for the generating process. We will restrict ourselves to univariate time series, keeping in mind that all models can easily be extended to the mulitvariate case.

3.1 Autoregressive modeling and feedforward networks

Forecasting can be interpreted as making optimal use of past information to predict the future. Of course, in real-world applications most of the time we must assume stochasticity, i.e. forecasting can only lead to an estimate in terms of an *expected value* or *expectation*, from which actual observations will differ due to unpredictable influences, modeled as a noise process.

One of the most common assumptions in time series forecasting is based on taking past observations as the sole past information available. The expected value is assumed to a be a function of a fixed, and limited, number of past observations. The noise process is assumed to be additive. This leads to the following expression:

$$x_t = F(x_{t-1}, x_{t-2}, \ldots, x_{t-p}) + \epsilon_t \tag{6}$$

For convenience we denote $X_{t,p} = (x_{t-1}, x_{t-2}, \ldots, x_{t-p})$. ϵ_t is usually refered to as a *random shock.* This type of model is generally called an *autoregressive* (or AR) model, since it amounts to a general regression of the observable variable x_t over its own past values. p is called the *order* of the model and corresponds to the number of past observations used in the regression. The best forecast after model estimation is to output the expected value $\mu_t = F(X_{t,p})$.

Using a limited number of past observations corresponds to the common *Markov assumption*, which states that all we have to know to predict the future is the present state of the system, in this case given by the vector of p past observations. In other words, we assume that we do not have to know the entire evolution of the time series in order to predict.

It is obvious that equations like 6 can be seen both as a model for forecasting (i.e. the best prediction is, after an appropriate estimation of the parameters describing f, to forecast the expected value $\mu_t = \hat{x}_t = F(X_{t,p})$) and as a generative model describing the underlying process. Given proper starting values, equation 6 can be used to generate time series values with the same characteristics as the original observations, drawing ϵ_t from the assumed distribution.

The most commonly used version in time series processing literature is the class of *linear* AR models, i.e. where $F(x)$ is a linear function of the following type:

$$x_t = \sum_{i=1}^{p} a_i x_{t-i} + \epsilon_t \tag{7}$$

The noise process behind ϵ_t is usually assumed to be *identically independently distributed* (i.i.d.), typically following a Gaussian distribution with zero mean and a given variance σ^2, i.e. $\epsilon \sim N(0, \sigma^2)$.

The simplest form of a linear AR process is one of order 1 with $a_1 = 1$ and $\epsilon \sim N(0,1)$:

$$x_t = x_{t-i} + \epsilon_t \tag{8}$$

This process is called *random walk*. The expected value, and therefore the best possible prediction, for x_t is the previous value x_{t-1}. In other words, the process is characterised by the fact that at each time step an i.i.d. disturbance ϵ_t is added to the observed variable. If a given time series follows a random walk, the best possible prediction is trivial and leads to no new information. This process is especially important for domains like financial time series processing, since it corresponds to the hypothesis of an efficient financial market. The hypothesis says that at each time step (e.g. each day) all available information that could be used to beat the market (i.e. to profit from a non-trivial prediction) has already been absorbed by the market mechanisms, rendering such profitable forecasting impossible. But also in other domains it is important to keep the random walk in mind before applying any more complex prediction method, and always benchmark such a model against it.

Naturally, more complex time series processes exist, also in the financial markets, and thus its is now worthwhile to explore how neural computation can enhance the common linear AR models. From equation 6 the main contribution from neural computation in this context becomes clear: any universal approximator can potentially be used to model an arbitrary nonlinear function $F(x)$. An example is shown in figure 3. A multilayer perceptron, as an example, can be used to model an arbitrary *nonlinear autoregressive process*. The way the past p observations x_{t-i} are used is usually called *time window*, or sometimes a special form of *time delay*, in neural network literature.

3.2 Complex noise models

Nonlinearity is not the only sensible extension to the classical linear AR model in equation 7. Recently, research has focused on modeling more complex noise processes than the ususal Gaussian distribution with constant variance σ^2 ([Schittenkopf et al. 2000, Neuneier et al. 1994, Husmeier 1999]).

[Bishop 1995] has demonstrated that the minimization of the summed squared error in regression (and, therefore, in autoregression), assuming linear output activation functions, corresponds to a maximum likelihood estimation assuming constant Gaussian noise $N(0, \sigma^2)$. After estimation ("learning"), σ^2 corresponds to the normalised residual quadratic error. This is illustrated in figure 4. An autoregressive process of order 1 can be visualised by plotting all past observations x_{t-1} against the present observations x_t, to be forecast. The noise process determines how the actual observations

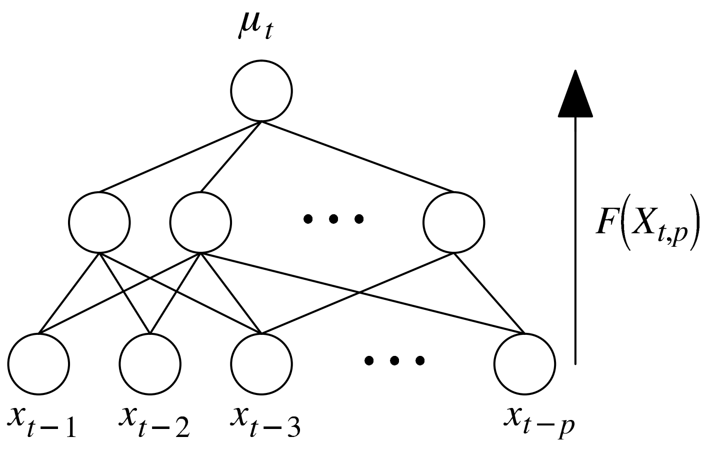


Figure 3: A feedforward neural network as a nonlinear autoregressive model.

are distributed around the expected value, given by the, potentially nonlinear, function $F(x_{t-1})$.

The illustration in figure 4 makes clear that the noise around the expected value does not have to be constant, nor does it have to be Gaussian. Instead, it can be seen as also being dependent on the input, i.e. the past observations. In formal terms

$$\epsilon \sim D(\vec{\theta}), \tag{9}$$

where

$$\vec{\theta} = g(X_{t,p}) \tag{10}$$

D is an arbitrary parameterised distribution with parameters $\vec{\theta}$ and is called the *conditional distribution*, since it depends on the past. One could also speak of a time-dependent noise distribution, since it is permitted to change at every time step.

Estimation of the models is straight-forward if one sticks to the maximum likelihood framework suggested above. The model likelihood becomes

$$L = \prod_{i=1}^{N} d(\vec{\theta} = g(X_{t,p}^{(i)})) \tag{11}$$

where N is the number of time series samples in the training set, and d is the probability density function corresponding to D.

The simplest extension to the standard AR case is sticking to the Gaussian distribution but keeping the variance σ^2 time-dependent. The likelihood in this case is

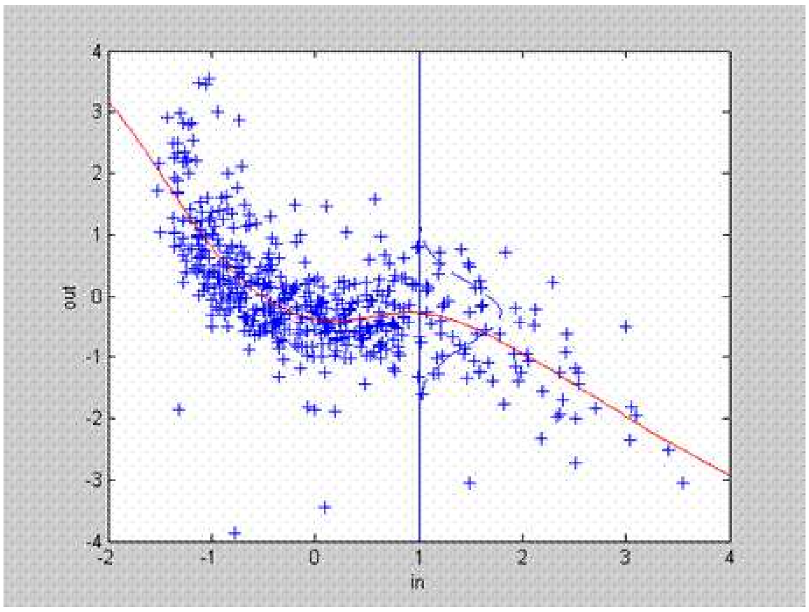


Figure 4: Plotting x_t over x_{t-1} can visualize the role of the noise process in autoregression. For any particular input x_{t-1}, the density function of the noise process ϵ_t describes the distribution of actual observations corresponding to that input. This makes clear that the noise does not have to constant, i.e. independent from x_{t-1}, or even Gaussian.

$$L = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2(X_{t,p})}} e^{-\frac{(x^{(i)}-\mu(X_{t,p}))^2}{2\sigma^2(X_{t,p})}} \tag{12}$$

In neural networks terms this means an additional output unit corresponding to the estimate of σ^2, as depicted in figure 5. In time series literature this case is known as a *heteroskedastic* time series, i.e. a time series with time-dependent variance of its noise process. This is of particular interest in financial time series analysis, since it corresponds to the case of a time-dependent *volatility* of a commodity. Looking at the ATX return series, one sees that this is apparently the case. At different points in time the variance of returns appears to be of different quantity, which can be modeled by a heteroskedastic process. When building a model for option pricing, this dependency is of particular interest. But this is not the only application where heterosketasticity plays an important role. The fact that knowing about the variance of the noise process permits the estimation of a reliable confidence interval in which actual observations are expected to lie highlights the great potential of using such extended models. The noise process in this case is no longer i.i.d. – the distributions guiding the random shocks are still independent, but no longer identical (although still all Gaussian).

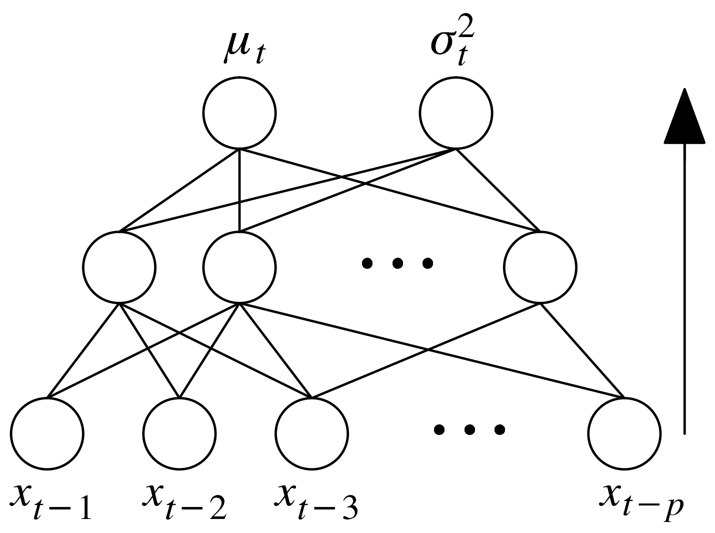


Figure 5: A simple extension to the feedforward neural network to account for heteroskedasticity. An additional output for σ_t^2 is added to account for its potential nonlinear dependence on $X_{t,p}$

What has been said about the general (nonlinear) neural network case is, of course, also true for the linear case. It is worth noting that a particular type of linear[1] case to model heteroskedastic time series is the well-known *autoregressive conditional hetereoskedasticity* (ARCH) model [Engle 1982], frequently used in finance. The ARCH model is defined as

$$\sigma_t^2 = \sum_{i=1}^{p} a_i r_{t-i}^2 \tag{13}$$

where r_{t-i}^2 are past returns of the time series (i.e. the values of the differenced original time series) and σ_t^2 is the variance of a Gaussian distribution. In this case, μ_t, i.e. the expected value of the time series r_t, is assumed to be 0 (or modeled separately using an AR model, replacing r_t by the residuals of that process). Comparing equation 13 to 12 one sees that the neural network case is a nonlinear generalisation of the ARCH model.

Another extension is to choose a parametric probability density function other than the Gaussian, in order to model specific characteristics of the conditional distribution of time series values. In financial time series analysis, it is known that conditional distributions (as well as unconditional ones) can have a higher kurtosis than a Gaussian. For these purposes, the use of a student t-distribution is rather commen.

[1]Strictly speaking, the ARCH model is quadratic in r_{t-i} but linear in r_{t-i}^2

In the spirit of neural computation, an even more general model appears appropriate in those cases where conditional distributions are unknown but expected to be non-Gaussian. Any arbitrary density function can be approximated by a *mixture of Gaussians*. Consequently, [Bishop 1994] has suggested the so-called *mixture density network* (MDN) to model arbitrary noise distributions in regression. Several authors ([Schittenkopf et al. 2000, Neuneier et al. 1994, Miazhynskaia et al. 2003]) have demonstrated the viability to use mixture density networks for arbitrary conditional return distributions in financial time series analysis.

The definition of a mixture density network is straight-forward given equation 11, if one inserts the following density function:

$$d(\vec{\mu}, \vec{\sigma^2}, \vec{\pi}) = \sum_{i=1}^{k} \frac{\pi_i(X_{t,p})}{\sqrt{2\pi\sigma_i^2(X_{t,p})}} e^{-\frac{(x-\mu_i(X_{t,p}))^2}{2\sigma^2(X_{t,p})}} \qquad (14)$$

Introducing arbitrary nonlinearity, this amounts to a neural network with $3k$ outputs, each one corresponding to one of the parameters of the mixture, in a straight-forward extension of the network in figure 5. Since there is no reason to assume that the three sets of paramaters – namely the centers μ_i, the widths σ_i^2 and the weights, or priors, π_i – have related nonlinear dependencies on the past, the choice of three different networks is often more appropriate. Learning (model estimation), as was the case for the simpler cases above, amounts to maximising the corresponding likelihood function (minimising the negative log likelihood, respectively). The corresponding error, or loss, function no longer corresponds to a simple squared error, however.

The advantage of using mixture density networks in forecasting is that now arbitrary conditional distributions can be modeled in a semi-parametric way. By replacing the multilayer network with a single-layer perceptron, linear versions can be obtained as well. [Miazhynskaia et al. 2003] has shown that this leads to more reliable confidence intervals for the estimated forecasts, which in their case can be used in risk analysis.

One remark, however, is in place. Mixture models – similar to the approximation of nonlinear functions in a neural network – usually suffer from an *identifiability* problem. In other words, parameters resulting from estimations cannot be interpreted by assigning a meaning to them. Similarly, it can no longer be guaranteed that it is really "pure" noise that is modeled. This is exemplified in figure 6, depicting an MDN estimate for the sunspot time series[2]. An autoregressive mixture density network of order 1 is used to model the distributions x_t (plotted against x_{t-1}). It is obvious that an AR model of order 1 is insufficient to model the structure behind the time series.

[2]Note that the original, although normalised, time series is used, without removing seasonalities

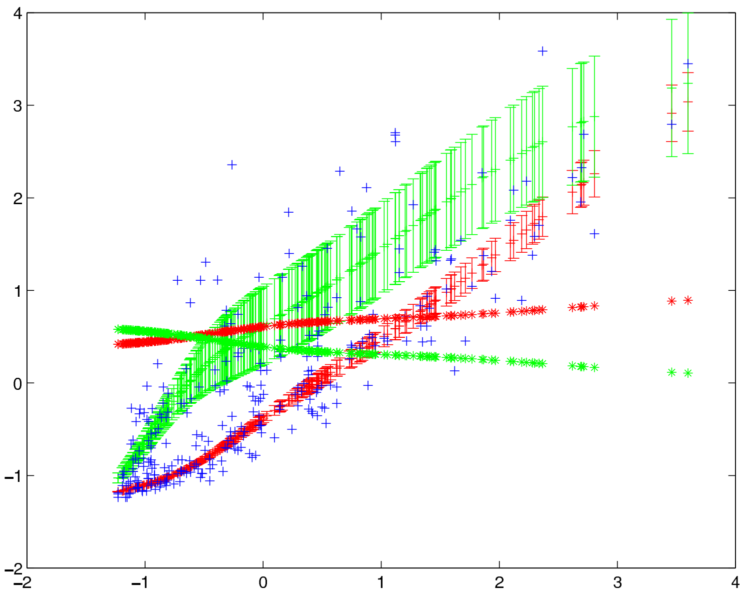


Figure 6: The resulting estimations from training a mixture density network with the subspot data. Two lines of error bars are drawn depicting the centers and widths of the two Gaussians of the mixture, dependent on the input x_{t-1}. Especially in the middle range, the resulting mixture density is bimodal reflecting the fact that in an AR(1) view, for each input the next value is about as likely to be higher as it is to be lower. The two additional lines depict the priors π_i for each Gaussian in the mixture.

Given a time series value around 0, the probability of the series to go up is about equal to it going down. For reliable forecasts of the expected value, an order of at least 2 would be necessary. The resulting model estimate in figure 6 describes this as a bimodal distribution for each x_{t-1} around the value 0. This apparently complex noise process thus captures some of the structure in the data, which would more appropriately be captured by a second-order AR process. Choosing too general a model, therefore, no longer permits the strict distinction between structure in the data and noise. Therefore such models should be used with care.

3.3 Moving average models and recurrent networks

Another common class of models uses a different kind of past information for forecasting. Instead of past time series values, the expectation for x_t is assumed to depend on past random shocks ϵ_{t-i}. In the linear case this amounts to:

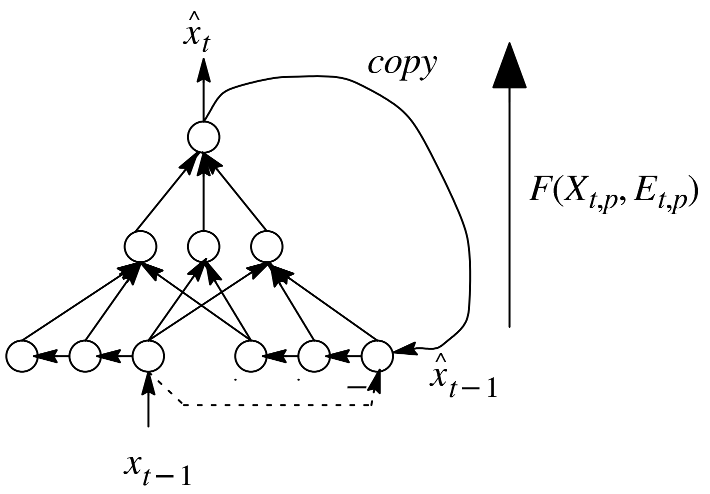


Figure 7: A recurrent Jordan network as an instantiation of a nonlinear moving average model.

$$x_t = -\sum_{i=0}^{q} b_i \epsilon_{t-i} \tag{15}$$

For convenience, we denote $E_{t,q} = (\epsilon_{t-1}, \epsilon_{t-2}, \ldots \epsilon_{t-q})$. By letting the index run from 0 to q, the current random shock is included in this model ($b_0 = 1$). This type of model is called a *moving average* (MA) process of order q. It has been shown that any finite AR process corresponds to an MA process of infinite order, and vice versa. Still, a finite MA process, or even the combination of AR and MA processes (so-called ARMA models) can be viable models for a given time series.

In analogy to above, neural networks can be used to generalise linear MA process to arbitrary nonlinear versions. The question is, however, what input to use to the network, if past random shocks are not really known. [Connor et al. 1992] and [Dorffner 1996] have demonstrated that using past estimates, i.e. past outputs of the network, as inputs, in the limit of the model converging to the true model, amounts to being equivalent to using past random shocks as inputs. If the network outputs the correct expected value for x_t, $\hat{x}_t = \mu_t$, then $\epsilon_{t-1} = x_{t-1} - \hat{x}_{t-1}$. In other words, $\hat{x}_{t-1}$ implicitly contains the information about the past random shock and can thus be used as an input to the network. Using past estimates amounts to a recurrent connection of the network's output to the input (see figure 7), i.e. a recurrent network usually called a *Jordan network* (see [Dorffner 1996] for the theoretical equivalence to an MA process in the limit).

We can therefore conclude: Recurrent Jordan networks are instantiations of nonlinear moving average processes. Little is known, however, about convergence properties of this type of model (i.e. whether and how quickly they converge toward the true model, which is a prerequisite for being an MA process).

In this context, it is worth looking at another common time series model and its nonlinear generalisation. For heteroskedastic time series, a property known as *volatility clustering* is often observed. This means that a high variance (high volatility) is often followed by several time steps of high variance. For instance, in the financial markets large shocks tend to prevail for some time. To model this property, the GARCH model (generalised autoregressive conditional heteroskedasticity) was introduced ([Bollerslev 1986]):

$$\sigma_t^2 = \sum_{i=1}^{p} a_i r_{t-i}^2 + \sum_{i=1}^{p} b_i \sigma_{t-i}^2 \tag{16}$$

Couched in neural networks terms, as the nonlinear ARCH above, this again means using past estimates (this time of σ^2) as input to the network. Therefore, a nonlinear GARCH model can be obtained by introducing the *recurrent mixture density network* [Schittenkopf et al. 2000, Tino et al. 2001].

3.4 State space models and recurrent networks

Another common method for time series processing are so-called *state space models* [Chatfield 1989]. Here the assumption is that the current state (in terms of the Markov assumption) is not given by the past observations directly but is hidden. The observations thus depend on a state vector $\vec{s}$:

$$x_t = \mathbf{C}\vec{s}_t + \epsilon_t \tag{17}$$

where $\mathbf{C}$ is a transformation matrix. The time-dependent state vector is usually modeled by a (multivariate) linear AR(1) process:

$$\vec{s}_t = \mathbf{A}\vec{s}_{t-1} + \mathbf{B}\vec{\eta}_t \tag{18}$$

where $\mathbf{A}$ and $\mathbf{B}$ are matrices, and $\vec{\eta}_t$ is a vectorial noise process, similar to ϵ_t above.

If we further assume that the states are also dependent on the past time series observations (an assumption, which is common, for instance, in signal processing – see [Ho et al. 1991]), and neglect the additional noise term $\mathbf{B}\vec{\eta}_t$:

$$\vec{s}_t = \mathbf{A}\vec{s}_{t-1} + \mathbf{D}X_{t,p} \tag{19}$$

then we basically obtain an equation describing a recurrent neural network type, known as *Elman network* (after [Elman 1990]), depicted in figure 8. The Elman network is an MLP with an additional input layer, called the *state layer*, receiving as feedback a copy of the activations from the hidden layer at the previous time step. If we use this network type for forecasting,

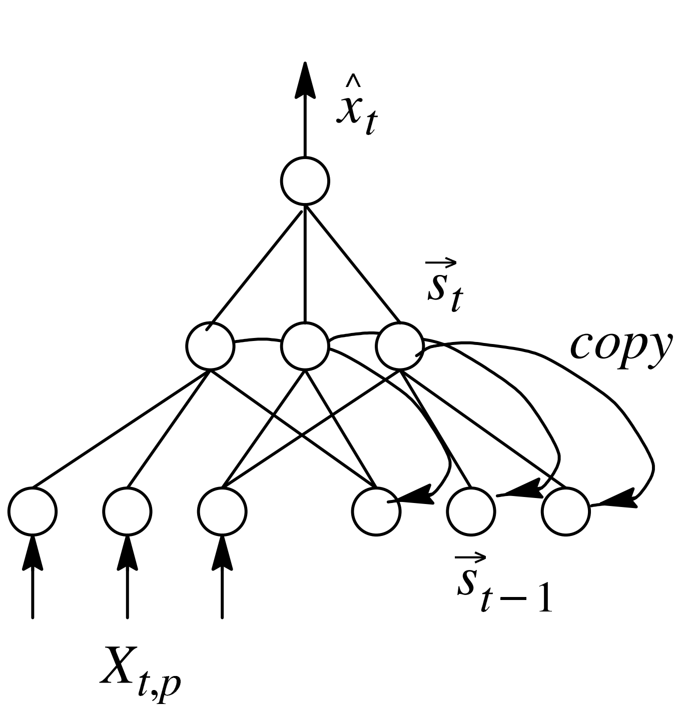


Figure 8: The Elman recurrent network as an instantiation of the state-space model.

and equate the activation vector of the hidden layer with $\vec{s}$, the only difference to equation 19 is the fact that in an MLP a sigmoid activation function is applied to the input of each hidden unit:

$$\vec{s}_t = f(\mathbf{A}\vec{s}_{t-1} + \mathbf{D}X_{t,p}) \tag{20}$$

where f is a sigmoid function. In other words, the transformation is not linear but the application of a *logistic regressor* to the input vectors. This leads to a restriction of the state vectors to vectors within a unit cube, with non-linear distortions towards the edges of the cube. Note, however, that this is a very restricted non-linear transformation function and does *not* represent the general form of non-linear state space models (see below).

Like above, the strong relationship to classical time series processing can be exploited to introduce "new" learning algorithms. For instance, in [Williams 1992] the *Kalman algorithm*, developed for the original state space model (Kalman filter) is applied to general recurrent neural networks.

As hinted upon above, a general non-linear version of the state space model is conceivable, as well. By replacing the linear transformation in equations 17 and 18 by an arbitrary non-linear function, one obtains

$$\vec{x}_t = F_1(\vec{s}_t) + \vec{e}_t \tag{21}$$

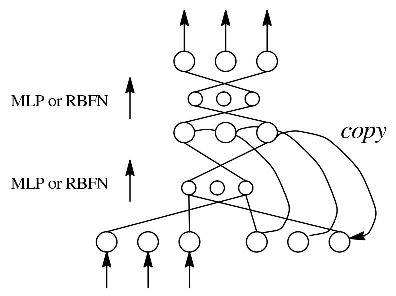


Figure 9: An extension of the "Elman" network as realization of a non-linear state-space model

$$\vec{s}_t = F_2(\vec{s}_{t-1}) \tag{22}$$

Like in the previous sections on non-linear ARMA models, these non-linear functions F_1 and F_2 could be modeled by an MLP or RBFN. The resulting network is depicted in figure 9. An example of the application of such a network is [Kamiho et al. 1993].

It has frequently been noted in literature that the state vector $\vec{s}$ apparently implicitly contains information of the *entire* past of the time series. Therefore, recurrent networks seemingly are not limited to a fixed time horizon, as are ARMA models. In practice, however, this is not true. The influence of past information vanishes exponentially, leading to a model that actually only takes recent information into account. A similar observation applies to model estimation. The gradient in minimizing the negative log likelihood, which must be "propagated back" via the recurrent connections, also vanishes exponentially ([Bengio et al. 1994]), rendering training difficult in many cases. Thus there is evidence that in practical applications, recurrent networks can have a disadvantage against feedforward networks, although potentially equally powerful (see, for instance, [Hallas & Dorffner 1998]).

4 Discrete valued and symbolic time series

So far, we have discussed time series with continous values x_t. In many applications, however, observations can only take one of a small finite set of values. Examples are binary measurements or very coarsely quantised signal values. A special case are *symbolic time series*, where values do not have an order (or the order is neglected) and can be considered as symbols from a given finite alphabet, i.e. $x_t \in \{s_i\}$, where s_i are arbitrary symbols. Examples are letters in a text, amino acids in a gene string, or continuous time series that have been quantised into a small number of intervals (e.g. 'up' and 'down' for stock price returns).

Forecasting can again be viewed as estimating expected values based on past information. The main difference is that probability densities are replaced by discrete probabilities for the symbols in the alphabet. The equivalent to an AR model would be looking at the string of past p symbols and finding estimates for each symbol to follow the string. This conditional probability distribution can be denoted as $P(x_t|x_{t-1}x_{t-2}\dots x_{t-p})$, where the condition part denotes the concatenation of the past p symbols.

This type of model is called a *Markov chain* (or Markov model) of order p. The probabilities can easily be estimated as the empirical probabilities (frequencies) that each symbol in the alphabet follows the particular given string. However, this type of model easily runs into problems for higher orders, since long substrings can be rather rare in the entire time series, rendering the estimation of empirical probabilities impossible.

Therefore, several authors ([Ron et al. 1996, Tino & Dorffner 2001]) have proposed so-called *variable-length Markov models* which only consider contexts (i.e. substrings) that occur frequently enough in the time series. The approach by [Tino & Dorffner 2001], the so-called *fractal prediction machine* (FPM), not only has a very intuitive geometric interpretation, but also bears an important similarity to another class of recurrent neural network. Basically, each potential substring is mapped onto a point in a $\log_2(n)$-dimensional space (where n is the size of the alphabet), following a simple mapping borrowed from iterated function systems ([Barnsley 1988]). This is illustrated in figure 10. When this is done for every substring of length p, the resulting distribution of points has the following interesting property: Strings that have a large common suffix (i.e. have a large common ending substring) are mapped onto points that are close together in space. Therefore, clusters of points correspond to substrings that frequently occur in the time series. Using a simple clustering algorithm, this can be used for identifying substrings, for which a reliable probability estimate for subsequent symbols can be found.

The result can be viewed as a stochastic automaton, each state of which forms an equivalence class of substrings. For each such state, probabilities can be estimated by calculating the relative frequencies of symbols that follow. This view leads to an analogy to state space models and recurrent neural networks, as was exemplified above for continuous-valued time series. As shown in [Tino & Dorffner 1998], the mapping of the FPM corresponds to the recurrent part of a second-order neural network, while the feedforward part would take the role of estimating probabilities for each state.

5 Signal filtering

As stated above, there is no principled formal distinction between what is usually termed *time series processing* and *signal processing*. The main differ-

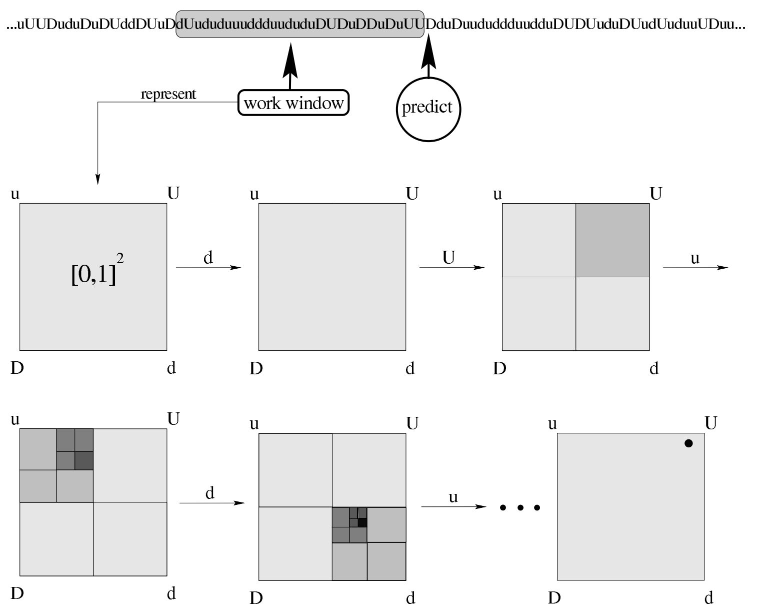

...uUUDuduDuDUddDUuDdUududuuuddduududuDUDuDDuDuUUDduDuududdduudduDUDUuduDUudUuduuUDuu...
represent
work window
predict
u
U
[0,1]²
D
d
u
U
d
U
u
d
u
•••

Figure 10: The basic mechanism of a fractal prediction machine to map substrings of symbolic sequences onto points in space: Each symbol in the alphabet is assigned a corner in the $\log_2(n)$-dimensional space for an alphabet of size n. For a given work window, all symbols in the resulting substring are processed in order, starting with the most recent one. As in an iterative function system, an affine mapping of the entire space onto one of the n subspaces ("corners") is performed, one particular mapping for each symbol. By tracing the center point of the space through all the mappings, the end point reached corresponds to the entire substring. The interesting property is that substrings with long common suffixes are mapped to points that are close in space.

ence lies in how problems are phrased and not in the chosen methodology. A common problem for signal processing is finding appropriate *filters* to describe or generate signals.

Digital filters can be depicted in a way that is pratically identical to autoregressive and moving average models. The former case, for instance, corresponds to the so-called *finite impulse response filters*. Therefore, there is a strong correspondance between forecasting and filtering models.

Autoregressive models can also be shown to be equivalent to the well known *spectral analysis* of signals. In other words, parameters of a linear AR model of a given order can be used as signal descriptors in a similar way as a spectrum derived from Fourier analysis. This gives rise to applications in recognizing typical signal patterns (waveforms) or to classify signals. A typical example is the classification of electroencephalographic (EEG) recordings of a person during sleep into one of several sleep stages (see, e.g., [Sykacek et al. 2002]).

Another important function of AR filtering is that of *denoising*. According to the formulation in equation 6, the model divides the signal into signal content (the predictable component) and a (usually white) noise process. The residuals after model estimation thus corresponds to noise that can be removed from the original signal.

From what we have seen in time series processing, it becomes clear that neural networks lend themselves for nonlinear extensions of classical linear filters ([Haykin 1986]). However, they should be seen as means to an end (i.e. more optimal denoising) but due to the identifiability problem mentioned above they cannot be used in the same way to describe signal characteristics in a parametric manner.

6 Practical considerations

We have seen that neural networks are powerful models that can be used in various time series and signal processing applications. Straight-forward extensions of simple mathematical principles have lead to advanced models desribing time series and signals in an intricate way (e.g. with respect to the noise process). Power in modeling, however, always comes with a price that has to be paid through extra care and sound validation techniques, without which neural networks are easily mis-applied.

In particular,

- Semiparametric nonlinear techniques need a sensible model selection and validation strategy. In general pattern recognition, usually resampling strategies such as n-fold cross-validation are applied to this avail – meaning that multiple runs with different training and independent validation sets must be performed. In time series processing it can be shown that, in order to be truly independent, validation sets should always be observations that occur *after* the training set. This leads to a sliding window technique that should be applied when validating process models in a maximum likelihood framework (see, for instance, [Schittenkopf et al. 2000]).

- Models with a large number of degrees of freedom (e.g. weights in a neural network) need large number of training samples. Stationarity of the time series becomes an important issue here. In other words, training sets for time series processing can often not be extended to arbitrary size, since the main characteristics of the time series or signal might change. Thus there often is an inherent limit to the complexity of the models that can reliably be estimated.

- It is not a priori clear for a given time series whether nonlinearity indeed plays a large role. In the financial markets, for instance, there is growing evidence that arbitrary nonlinearity in a model cannot significantly improve forecasting performance. Therefore, any complex neural network model should always be carefully validated against its linear or otherwise more parametric counterparts.

- Identifiability, as mentioned several times in this paper, might not be a problem for many engineering solutions (e.g. finding good forecasts). But for many applications (e.g. filtering) it does play a role and often leaves neural networks useless (or at least, difficult to deal with), despite their potential power in modeling.

7 Summary and conclusions

The purpose of this paper was to give a short overview of the potential of neural computation methods in modeling time-varying data. Much emphasis was put on showing that neural networks are embedded in more traditional time series theory and have the potential to provide powerful alternatives and extensions, especially with respect to nonlinearity. Time series and signal processing, however, is a field with a long traditon that has not waited for neural computation to provide viable models. Linear ARMA models, Markov chains, linear filters, etc. are rather powerful in themselves and must therefore always be considered before blindly applying a neural network.

Many important topics have not been addressed, such as deterministic chaos in nonlinear processes, the relationship between recurrent networks and stochastic automata, etc. Also, a large part of neural computation, from support vector machines, wavelet networks to independent component analysis could not be dealt with. Nevertheless, the hope is that the reader could get a glimpse of the fascinating potentials of advanced models to desribe time-varying data, which is prevalent in a great many of practical applications.

References

[Barnsley 1988] Barnsley, M.F.: *Fractals everywhere*, Academic Press, New York, 1988.

[Bengio et al. 1994] Bengio Y., Simard P., Frasconi P.: Learning long-term dependencies with gradient descent is difficult, *IEEE Trans. Neural Networks* **5(2)**, 157-166, 1994.

[Bishop 1994] Bishop, C. M.: Mixture density networks, Neural Computing Research Group Report: NCRG/94/004, Aston University, Birmingham, 1994.

[Bishop 1995] Bishop C.: *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, 1995.

[Bollerslev 1986] Bollerslev, T.: A generalized autoregressive conditional heteroskedasticity, *Journal of Econometrics*, **31** (1986), 307-327.

[Chatfield 1989] Chatfield C.: *The Analysis of Time Series – An Introduction*, Chapman and Hall, London, 4th edition, 1989.

[Connor et al. 1992] Connor J., Atlas L.E., Martin D.R.: Recurrent Networks and NARMA Modeling, in Moody J.E., et al.(eds.): *Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo, CA, pp.301-308, 1992.

[Dorffner 1996] Dorffner G.: Neural networks for time series processing, Neural Network World, 6(4)447-468, 1996.

[Elman 1990] Elman J.L.: Finding Structure in Time, *Cognitive Science* **14(2)**, 179-212, 1990.

[Engle 1982] Engle, R. F.: Autoregressive conditional heteroskedasticity with estimates of the variance of U.K. inflation, *Econometrica*, **50** (1982), 987-1008.

[Hallas & Dorffner 1998] Hallas M., Dorffner G.: A Comparative Study on Feedforward and Recurrent Neural Networks in Time Series Prediction Using Gradient Descent Learning, in Trappl R. (ed.), Cybernetics and Systems '98 - Proc. of 14th European Meeting on Cybernetics and Systems Research, Austrian Society for Cybernetic Studies, Vienna, pp.644-647, 1998.

[Haykin 1986] Haykin S.: Adaptive Filter Theory, Prentice-Hall, London/New York/Englewood Cliffs, NJ, 1986.

[Ho et al. 1991] Ho T.T., Ho S.T., Bialasiewicz J.T., Wall E.T.: Stochastic Neural Adaptive Control Using State Space Innovations Model, in *International Joint Conference on Neural Networks*, IEEE, pp.2356-2361, 1991.

[Hornik et al. 1989] Hornik K., Stinchcombe M., White H.: Multi-layer Feedforward Networks are Universal Approximators, *Neural Networks* **2**, 359-366, 1989.

[Husmeier 1999] Husmeier D.: Neural Networks for Conditional Probability Estimation, Springer, Berlin/Heidelberg/New York/Tokyo, 1999.

[Kamiho et al. 1993] Kamijo K., Tanigawa T.: Stock Price Pattern Recognition: A Recurrent Neural Network Approach, in Trippi R.R. & Turban E.(eds.): *Neural Networks in Finance and Investing*, Probus, Chicago, pp.357-370, 1993.

[Miazhynskaia et al. 2003] Miazhynskaia T., Dorffner G., Dockner E.: Risk Management Application of the Recurrent Mixture Density Network Models, to appear in: *Proceedings of ICONIP/ICANN 2003*, Springer Verlag, 2003.

[Neuneier et al. 1994] Neuneier R., Finnoff W., Hergert F., Ormoneit D.: Estimation of conditional densities: a comparison of neural network approaches", in: Marinaro M., and P. G. Morasso (eds.) *ICANN 94 - Proceedings of the International Conference on Artificial Neural Networks*, Berlin: Springer, 689-692.

[Ron et al. 1996] Ron, D., Singer, Y., Tishby, N.: The power of amnesia. *Machine Learning* 25, 1996.

[Schittenkopf et al. 2000] Schittenkopf C., Dorffner G., Dockner E.J.: Forecasting time-dependent conditional densities: A semi-nonparametric neural network approach, Journal of Forecasting, 19, 355-374, 2000.

[Sykacek et al. 2002] Sykacek P., Dorffner G., Rappelsberger P., Zeitlhofer J.: Improving biosignal processing through modeling uncertainty: Bayes vs. non-Bayes in sleep staging, Applied Artificial Intelligence, 16(5)395-421, 2002.

[Tino & Dorffner 1998] Tino P., Dorffner G.: Recurrent Neural Networks with Iterated Function Systems Dynamics, in NC'98, Proceedings of the ICSC/IFAC Symposium on Neural Computation, Vienna, Austria., pp.526-532, 1998.

[Tino & Dorffner 2001] Tino P., Dorffner G.: Predicting the future of discrete sequences from fractal representations of the past, Machine Learning, 45(2)187-217, 2001.

[Tino et al. 2001] Tino P., Schittenkopf C., Dorffner G.: Financial volatility trading using recurrent neural networks, IEEE Transactions on Neural Networks, 12(4)865-874, 2001.

[Williams 1992] Williams R.J.: Training Recurrent Networks Using the Extended Kalman Filter, in *International Joint Conference on Neural Networks*, Baltimore, IEEE, pp.241-246, 1992.

The complexity of real recursive functions[*]

Manuel Lameiras Campagnolo

D.M./I.S.A., Universidade Técnica de Lisboa,
Tapada da Ajuda, 1349-017, Lisboa, Portugal;
C.L.C./D.M./I.S.T., Universidade Técnica de Lisboa,
Av. Rovisco Pais, 1049-001, Lisboa, Portugal
`mlc@math.isa.utl.pt`

Abstract. Recursion theory on the reals, the analog counterpart of recursive function theory, is an approach to continuous-time computation inspired in the models of Classical Physics. In recursion theory on the reals, the discrete operations of standard recursion theory are replaced by operations on continuous functions, such as composition and various forms of differential equations as indefinite integrals, linear differential equations, and more general Cauchy problems. We define classes of real recursive functions, in a manner similar to the standard recursion theory, and we study their complexity. We consider, namely, the structural and the computational complexity of those classes. As a result, we prove both upper and lower bounds for several classes of real recursive functions, which lie inside the primitive recursive functions and, therefore, can be characterized in terms of standard computational complexity.

Key words: Continuous-time computation, differential equations, recursion theory, computational complexity.

1 Introduction

Recursive function theory provides the standard notion of computable function [Cut80,Odi89]. Moreover, many time and space complexity classes have recursive characterizations [Clo99]. As far as we know, Moore [Moo96] was the first to extend recursion theory to real valued functions. We will explore this and show that all main concepts in recursion theory like basic functions, operators, function algebras, or functionals, are indeed extendable in a natural way to real valued functions. In this paper, we define recursive classes of real valued functions analogously to the classical approach in recursion theory and we study the complexity of those classes. In recursion theory over the reals, the operations typically include composition of functions, and solutions of several forms of differential equations. On one hand, we look at the structural properties of various algebras of real functions, i.e., we explore intrinsic properties of classes of real recursive functions such as closure under iteration, bounded sums or bounded products. We investigate links between analytical and computational properties of real recursive functions. For instance, we show that a departure from analyticity to C^∞ gives closure under iteration, a fundamental property of discrete functions. On the other hand, we use standard computational complexity theory to establish upper and lower bounds on those algebras. We establish connections between subclasses of real recursive functions, which range from the functions computable in linear space to the primitive recursive functions, and subclasses of the recursive functions closed under various forms of integration. We consider, in particular, indefinite integrals, linear differential equations, and more general Cauchy problems. Finally, we describe some directions of work that suggest that the theory of real recursive functions might be fruitful in addressing open problems in computational complexity.

[*] A similar version of this paper was presented at UMC'02 [Cam02]. This paper describes join work with Félix Costa (I.S.T./Universidade Técnica de Lisboa) and Cris Moore (Santa Fe Institute and University of New Mexico).

2 Recursive functions over $\mathbb{R}$

Moore [Moo96] proposed a theory of recursive functions on the reals, which is defined in analogy with classical recursion theory. A function algebra

$$[B_1, B_2, ...; O_1, O_2, ...],$$

which we also call a computational class, is the smallest set containing basic functions $\{B_1, B_2, ...\}$ and closed under certain operations $\{O_1, O_2, ...\}$, which take one or more functions in the class and create new ones.

Although function algebras have been defined in the context of recursion theory on the integers, they are equally suitable to define classes of real valued recursive functions. As a matter of fact, if the basic functions in a function algebra are real functions and the operators map real functions into real functions, then the function algebra is a set of real functions. Furthermore, if the basic functions have a certain property (e.g. continuity or differentiability) which is preserved by the operators, then every function in the class will have that same property on its domain of definition. In recursion theory on the reals we consider operations such as the following.

COMP (Composition). Given functions $f_1, \ldots, f_p$ of arity n and g of arity p, then define h such that $h(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), ..., f_p(\boldsymbol{x}))$.

$\int$ (S-integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n+1+m$, if there is a unique set of functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I - S, \end{aligned} \tag{1}$$

on an interval I containing 0, where $S \subset I$ is a countable set of isolated points, and $\boldsymbol{h}$ is continuous for all $y \in I$, then $h = h_1$ is defined.

μ (Zero-finding). Given f of arity $n+1$, then define h such that

$$h(\boldsymbol{x}) = \mu_y f(\boldsymbol{x}, y) \overset{\text{def}}{=} \begin{cases} y^- = \sup\{y \in \mathbb{R}_0^- \, : \, f(\boldsymbol{x}, y) = 0\}, \text{ if } -y^- \le y^+ \\ y^+ = \inf\{y \in \mathbb{R}_0^+ \, : \, f(\boldsymbol{x}, y) = 0\}, \text{ if } -y^- > y^+ \end{cases}$$

whenever it is well-defined.

To match the definition in [Moo96], derivatives of functions can have singularities (we denote the set of singularities by S). The definition above allows the derivative of h to be undefined on the singularities, as long as the solution is unique and continuous on the whole domain. To illustrate the definition of the operator $\int$, let's look at the following example.

Example 1. ($\sqrt{\cdot}$) Suppose that the constant 1 and the function $g(y, z) = 1/2z$ are defined. Then, the solution of

$$\partial_y h = \frac{1}{2h} \quad \text{and} \quad h(0) = 1. \tag{2}$$

is defined on $I = [-1, 1]$. Indeed, the function $h(y) = \sqrt{y+1}$ is the unique solution of Equation (2) on $[-1, 1]$. The set of singularities is $S = \{-1\}$, so $\partial_y h(y)$ doesn't have to be defined on $y = -1$.

Clearly, the operations above are intended as a continuous analog of operators in classical recursion theory, replacing primitive recursion and zero-finding on $\mathbb{N}$ with S-integration and zero-finding on $\mathbb{R}$. Composition is a suitable operation for real valued functions and it is therefore unchanged. Then, the class of real recursive functions is defined in [Moo96] as:

Definition 1. *The real recursive functions are* $[0, 1, U; \text{COMP}, \int, \mu]$, where 0 and 1 are simply constant functions, and U denotes the set of projections $U_i^n(x_1, \ldots, x_n) = x_i$. We also define real recursive constants as:

Definition 2. *A constant a is said to be computable if there is an unary real recursive function f such that $f(0) = a$.*

Then, if a constant a is computable, then one can also define, with composition and zero, a constant unary function g as $g(x) = f(0(x)) = a$, for all x. As we will see below, some irrational constants like e or π are real recursive, and therefore we can define a function whose value is precisely e or π. This is in contrast to the definition of real numbers computable by Turing machines, where an irrational number is said to be computable if there is a sequence of rationals that converge to it effectively.

If μ is not used at all we get M_0, the "primitive real recursive functions", i.e., $[0, 1, -1, U; \text{COMP}, \int]$. These include the differentially algebraic functions, as well as constants such as e and π. However, M_0 also includes functions with discontinuous derivatives like $|x| = \sqrt{x^2}$.

To prevent discontinuous derivatives, and to make our model more physically realistic, we may require that functions defined by integration only be defined on the largest interval containing 0 on which their derivatives are continuous. This corresponds to the physical requirement of bounded energy in an analog device. We define this in a manner similar to S-integration, but with the additional requirement of the continuity of the derivative:

$\bar{\text{I}}$ (SC^1-integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n+1+m$, if there is a unique set of functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I - S, \end{aligned} \tag{3}$$

on an interval I containing 0, where $S \subset I$ is a countable set of isolated points, and $\boldsymbol{h}$ and $\partial_y \boldsymbol{h}$ are both continuous for all $y \in I$, then $h = h_1$ is defined.

Then, restricting $\int$ to $\bar{\text{I}}$, we define the class $[0, 1-1, U; \text{COMP}, \bar{\text{I}}]$. It is clear that all functions in $[0, 1-1, U; \text{COMP}, \bar{\text{I}}]$ are continuously differentiable on their domains. (A question that arises naturally is if they are of class C^∞.) Therefore, $f(y) = \sqrt{y+1}$ mentioned in Example 1 cannot be defined on the interval $[-1, 1]$ in $\overline{\mathcal{D}}$ anymore, since its derivative is not continuous on that interval.

Example 2. (θ_∞) In $\overline{\mathcal{D}}$ we can define a non-analytic function θ_∞ such that $\theta_\infty(t) = \exp(-1/t)$, when $t > 0$, and $\theta_\infty(t) = 0$, when $t \leq 0$. First consider the unique solution of the initial condition problem

$$z' = \frac{1}{(t+1)^2} z \quad \text{and} \quad z(0) = \exp(-1) \tag{4}$$

with a singularity at $t = -1$. This is $z(t) = 0$ if $t \leq -1$, and $z(t) = \exp(-\frac{1}{t+1})$ if $z > -1$. Then $\theta_\infty(t) = z(t-1)$. The function θ_∞ can be though as a C^∞ version of the Heaviside function θ, defined by $\theta(x) = 0$ when $x < 0$ and $\theta(x) = 1$ when $x \geq 0$.

We can restrict the integration operation even more, if we don't allow singularities for the derivatives in the domain of existence of the solution. Formally, we say that a functions is defined by proper integration if it is defined with the following operator:

I (Proper integration). Given functions $f_1, \ldots, f_m$ of arity n, and $g_1, \ldots, g_m$ of arity $n+1+m$, if there is a unique set of continuous functions $h_1, \ldots, h_m$, such that

$$\begin{aligned} \boldsymbol{h}(\boldsymbol{x}, 0) &= \boldsymbol{f}(\boldsymbol{x}), \\ \partial_y \boldsymbol{h}(\boldsymbol{x}, y) &= \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y)), \qquad \forall y \in I, \end{aligned} \tag{5}$$

on an interval I containing 0, then $h = h_1$ is defined.

This proper form of integration preserves analyticity [Arn96]. Moreover, if the functions $f_1, \dots, f_m$ and $g_1, \dots, g_m$ are of class C^k, then h is also of class C^k on its domain of existence (cf. [Har82, 5.4.1]). Since constants and projections are analytic and composition and integration preserve analyticity, then:

Proposition 1. *All functions in* $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ *are analytic on their domains.*

Similarly, one proves that functions of one variable in $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$ are precisely the differentially algebraic functions [Moo96,GC]. This means that the Gamma function, for instance, is not in the class $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$. Next, we give some examples of functions that do belong to that class.

Proposition 2. *The functions* $+$, $-$, $\times$, $\exp$, $\exp^{[m]}$, *defined as* $\exp^{[0]}(x) = 1$ *and* $\exp^{[n+1]}(x) = \exp(\exp^{[n]}(x))$ *for any integer* n, $\sin$, $\cos$, $1/x$, $\log$, *and* $\arctan$ *belong to* $[0, 1, -1, U; \mathrm{COMP}, \mathrm{I}]$.

To further explore the theory of real recursive functions, we restrict the integration operator to solving time-varying linear differential equations, i.e.,

LI *Linear integration.* Given $f_1, \dots, f_m$ of arity n and $g_{11}, \dots, g_{mm}$ of arity $n+1$, then define the function $h = h_1$ of arity $n+1$, where $\boldsymbol{h} = (h_1, \dots, h_m)$ satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{f}(\boldsymbol{x})$ and $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y)\boldsymbol{h}(\boldsymbol{x}, y)$.

As in classical recursion theory, we define new classes by restricting some operations but adding to the class certain basic functions which are needed for technical reasons. A typical example is the integer function called cut-off subtraction, defined by $x \dot{-} y = x - y$ if $x \geq y$, and $x \dot{-} y = 0$ otherwise. In some real recursive classes we include, instead, a basic function we denote by θ_k and is defined by $\theta_k(x) = 0$ if $x \leq 0$, and $\theta_k(x) = x^k$ if $x > 0$. Clearly, θ_0 is an extension to the reals of the Heaviside function, and $\theta_1(x)$ is an extension to the reals of $x \dot{-} 0$. In general, θ_k is of class C^{k-1}.

For example, we explore the class $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ for some fixed k. Since, unlike solving more general differential equations, linear integration can only produce total functions, then:

Proposition 3. *For any integer* $k > 0$, *if* $f \in [0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$, *then* f *is defined everywhere and belongs to class* C^{k-1}.

We will also consider an even more restricted form of integration, which is just the indefinite integral. Formally, this is defined by:

INT *Indefinite integral.* Given $f_1, \dots, f_m$ of arity n and $g_1, \dots, g_m$ of arity $n+1$, then define the function $h = h_1$ of arity $n+1$, where $\boldsymbol{h} = (h_1, \dots, h_m)$ satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{f}(\boldsymbol{x})$ and $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y)$.

3 Structural complexity

In this section, we ask questions about *intrinsic* properties of classes of real recursive functions such as closure under certain operations. We will see that some intriguing connections exist among closure properties and analytical properties of the classes we consider.

Closure under iteration is a basic operation in recursion theory. If a function f is computable, so is $F(x, t) = f^{[t]}(x)$, the t'th iterate of f on x. We ask whether these analog classes are closed under iteration, in the sense that if f is in the class, then so is some $F(x, t)$ that equals $f^{[t]}(x)$ when t is restricted to the natural numbers.[1]

Proposition 4. $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ *is closed under iteration.*

[1] In [CMC00] we answer this question for Shannon's General Purpose Analog Computer. For connections between real recursion theory and Shannon's model see [GC].

Proof. (Sketch) Let's denote $[0,1,-1,U;\mathrm{COMP},\bar{\mathrm{I}}]$ bu $\overline{\mathcal{D}}$. Given f, we can define in $\overline{\mathcal{D}}$ the differential equation

$$\begin{aligned}\big(\theta_\infty(\cos\pi t)+\theta_\infty(-\cos\pi t)\big)\,\partial_t y_1 &= -(y_1-f(y_2))\,\theta_\infty(\sin 2\pi t)\\ \big(\theta_\infty(\sin\pi t)+\theta_\infty(-\sin\pi t)\big)\,\partial_t y_2 &= -(y_2-y_1)\,\theta_\infty(-\sin 2\pi t)\end{aligned} \tag{6}$$

with initial condition $y_1(x,0)=y_2(x,0)=x$, where θ_∞ is the function defined in Example 2. We claim that the solution satisfies $y_1(x,t)=f^{[t]}(x)$, for all integer $t\geq 0$. On the interval $[0,\frac{1}{2}]$, $y_2'(x,t)=0$ because $\theta_\infty(-\sin 2\pi t)=0$. Therefore, y_2 remains constant with value x, and $f(y_2)=f(x)$. The solution for y_1 on $[0,\frac{1}{2}]$ is then given by

$$\exp\big(\tfrac{1}{\sin(2\pi t)}-\tfrac{1}{\cos(\pi t)}\big)\,y_1' = -(y_1-f(x)),$$

which we rewrite as $\epsilon\, y_1' = -(y_1-f(x))$. Note that $\epsilon\to 0^+$ when $t\to 1/2$. Integrating the equation above we obtain

$$y_1-f(x)=\exp(-\frac{1}{\epsilon}t),$$

where the right hand side goes to 0 when t approaches $1/2$. Therefore, $y_1(x,1/2)=f(x)$. A similar argument for y_2 on $[\frac{1}{2},1]$ shows that $y_2(x,1)=y_1(x,1)=f(x)$, and so on for y_1 and y_2 on subsequent intervals. The set of singularities of Equation (6) is $\{n/2, n\in\mathbb{N}\}$. □

However, if we replace SC^1-integration by proper integration, which preserves analyticity, then the resulting class is no longer closed under iteration. More precisely,

Proposition 5. $[0,1,-1,U;\mathrm{COMP},\mathrm{I}]$ *is not closed under iteration.*

Proof. (Sketch) We denote $[0,1,-1,U;\mathrm{COMP},\mathrm{I}]$ by $\mathcal{D}$. Let's suppose that $\mathcal{D}$ is closed under iteration. Since $\exp\in\mathcal{D}$, then there is a function F in $\mathcal{D}$ such that $F(x,t)=\exp^{[t]}(x)$ for all $t\in\mathbb{N}$ and all $x\in\mathbb{R}$. Therefore, F has a finite description in $\mathcal{D}$ with a certain fixed number of uses of the I operation. However, it is known that functions of one variable in $\mathcal{D}$ are differentially algebraic [Moo96], that is, they satisfy a polynomial differential equation of finite order. So, for any fixed t, F is differentially algebraic in x. But, from a result of Babakhanian [Bab73], we know that $\exp^{[t]}$ satisfies no non-trivial polynomial differential equation of order less than t. This means that the number of integrations that are necessary to define $\exp^{[t]}$ has to grow with t, which creates a contradiction. □

Since $[0,1,-1,U;\mathrm{COMP},\bar{\mathrm{I}}]$ contains non-analytic functions while all functions in $[0,1,-1,U;\mathrm{COMP},\mathrm{I}]$ are analytic, one could ask if there is a connexion between those two structural properties of real recursive classes. We believe that closure under iteration and analyticity are related in the following sense:

Conjecture 1. Any non trivial real recursive class which is closed under iteration must contain non-analytic functions.

As a matter of fact, even if it is known that the transition function of a Turing machine can be encapsulated in an analytic function [KM99,Moo98], no analytic form of an iteration function is known.

Next we consider restricted operations as bounded sums and bounded products and we ask which real recursive classes are closed under those operations. We say that an analog class is closed under bounded sums (resp. products) if given any f in the class, there is some g also in the class that equals $\sum_{n<t} f(x,n)$ (resp. $\prod_{n<t} f(x,n)$) when t is restricted to the natural numbers.

Let's see how to define bounded sums in a real recursive class. Not surprisingly, we find that this is related to indefinite integrals. We first define a step function F which matches f on the integers, and whose values are constant on the interval $[j,j+1/2]$ for integer j. F can be defined as $F(t)=f(s(t))$, where s is a continuous step function that matches the identity over the integers. This can be defined with the indefinite integral $s(0)=0$ and $s'(x)=c_k\theta_k(-\sin 2\pi x)$.[2] Then $s(t)=j$, and $F(t)=f(s(t))=f(j)$, whenever $t\in[j,j+1/2]$ for integer j. The bounded sum of f is then given by g, such that $g(0)=0$ and $g'(t)=c_kF(t)\,\theta_k(\sin 2\pi t)$. Then $g(t)=\sum_{z<n} f(z)$ whenever $t\in[n-1,n-1/2]$. So, we can define bounded sums with the constant π, a periodic function like sin, θ_k, and the operation of indefinite integrals. More precisely,

[2] The constant c_k is a rational or a rational multiplied by π.

Proposition 6. *For all $k \in \mathbb{N}$, $[0, 1, -1, \pi, \theta_k, \sin, U; \mathrm{COMP}, \mathrm{INT}]$ is closed under bounded sums. Moreover, any real recursive class which is closed under composition and indefinite integrals and contains the functions $0, 1, -1, \pi, \theta_k, \sin, U$ is closed under bounded sums.*

If a class is closed under bounded products and it contains, for instance, the identity function, then it has to contain functions that grow faster than polynomials. For instance, the class $[0, 1, -1, \pi, \theta_k, \sin, U; \mathrm{COMP}, \mathrm{INT}]$ cannot be closed under bounded products. What can we say if the analog class is closed under linear integration, instead of just indefinite integrals? We conjecture that the answer is still negative, since we believe that the simulation of bounded products would have to rely on a technique similar to Proposition 4 using synchronized clock functions, although we have no proof of this.

Let's then consider the following weaker property. We say that a class is closed under bounded products in a *weak sense* if, given any f in the class which has integer values for integer arguments (i.e., f is an extension to the reals of some $\tilde{f} : \mathbb{R}\times\mathbb{N} \to \mathbb{N}$), there is a g in the class such that $g(x, t) = \prod_{n<t} f(x, n)$ when t is restricted to the natural numbers. Then, in the presence of some appropriate non-analytic function like θ_k, proper integration and even linear integration are sufficient to simulate bounded products. In particular,

Proposition 7. *For all $k \in \mathbb{N}$, $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ is closed under bounded products in a weak sense.*

Proof. (Sketch) Let f be a function on $\mathbb{N}$ and g be a the function on $\mathbb{N}$ defined from f by bounded product. We show that if f has an extension to the reals in $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ then g does also. First, set $g_n = \prod_{j<n} f_j$. We can approximate the iteration $g_{j+1} = g_j f_j$ using synchronized clock functions as in proof of Proposition 4. However, since we only allow linear integration, the simulated functions cannot coincide exactly with the bounded product. Nevertheless, we can define a sufficiently close approximation because f and g have bounded growth (we can show that any function in $[0, 1, -1, \pi, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ is bounded by the iterated exponential $\exp^{[m]}$ for some m).

Let's define a two-component function $\boldsymbol{y}(\tau, t)$ where $y_1(\tau, 0) = y_2(\tau, 0) = 1$,

$$\begin{aligned} \partial_t y_1 &= (y_2 F(t) - y_1)\, c_k \theta_k(\sin 2\pi t)\, \beta(\tau) \\ \partial_t y_2 &= (y_1 - y_2)\, c_k \theta_k(-\sin 2\pi t)\, \beta(\tau) \end{aligned} \tag{7}$$

$\beta(\tau)$ is an increasing function of τ, F is defined as $f \circ s$ as in the proof of Proposition 6. We can show that if β grows fast enough (roughly as fast as $\exp^{[m]}$), then by setting $\tau = n$ we can make the approximation error $|y_1(n, n) - g_n|$ as small as we like. Since g has integer values, the accumulated error on $[0, n]$ resulting from this approximation can be removed with a suitable continuous step function that matches the identity over integers. Note that the Equation (7) is linear in y_1 and y_2.

We illustrate this construction in Figure 1. We approximate the bounded product of the identity function, i.e. the factorial $(n - 1)! = \prod_{j<n} j$. We numerically integrated Equation (7) using a standard package.

We can also show that a class is closed under the iteration of extensions to the reals of integer valued functions, as long as it is closed under proper integration, and it contains the non-analytic function θ_k or θ_∞. We call this property closure under iteration in a weak sense. For instance, it can be shown, using a technique similar to [Bra95], that

Proposition 8. $[0, 1, -1, \theta_\infty, U; \mathrm{COMP}, \mathrm{I}]$ *is closed under iteration in a weak sense.*

4 Computational complexity

In this section we explore connections among real recursive classes and standard recursive classes. Since we are interested in classes below the primitive recursive functions, we can characterize them in terms of standard space or time complexity, and consider the Turing machine as the underlying computational model. This approach differs from others, namely BSS-machines [BSS89] or information-based complexity

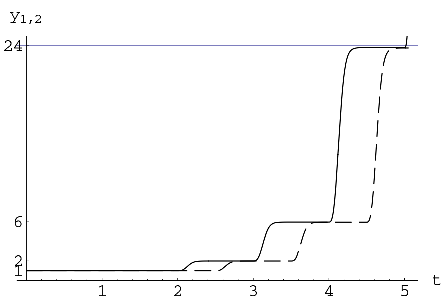


Fig. 1. A numerical integration of Equation (7), where f is a $\mathcal{L}$ function such that $f(0) = 1$ and $f(x) = x$ for $x \geq 1$. Here, $k = 2$. We obtain an approximation of an extension to the reals of the factorial function. In this example, where we chose a small $\tau < 4$, the approximation is just sufficient to remove the error with ϕ and obtain exactly $\prod_{n<5} n = 4! = \phi(y_1(5))$.

[TW98], since it focus on *effective* computability and complexity. There are two main reasons to this. First, the Turing machine model allows us to represent the concept of Cauchy sequences and, therefore, supports a very natural theory of computable analysis. Second, we aim to use the theory of real recursive functions to address problems in standard computational complexity. This would be difficult to achieve with an intrinsically analog theory like the BSS-machines over $\mathbb{R}$.

To compare the computational complexity of real recursive classes and standard recursive classes we have to set some conventions. On one hand, we follow a straightforward approach to associate a class of integer functions to a real recursive class. We simply consider the *discretization* of a real recursive class, i.e., the subset of functions with integer values for integer arguments. More precisely,

Definition 3. *Given a real recursive class* $\mathcal{C}$, $\mathcal{F}_{\mathbb{N}}(\mathcal{C}) = \{f : \mathbb{N}^n \to \mathbb{N}$ *s.t.* f *has an extension to the reals in* $\mathcal{C}\}$.

If $\mathcal{F}_{\mathbb{N}}(\mathcal{C})$ contains a certain complexity class $\mathcal{C}'$, this means that $\mathcal{C}$ has at least the computational power of $\mathcal{C}'$, i.e., we can consider $\mathcal{C}'$ as a lower bound for $\mathcal{C}$.

On the other hand, we consider the computational complexity of real functions. We use the notion of [Ko91], which is equivalent to the one proposed by Grzegorczyk [Grz55], and whose underlying computational model is the function-oracle Turing machine. Intuitively, the time (resp. space) complexity of f is the number of moves (resp. the amount of tape) required by a function-oracle Turing machine to approximate the value of $f(x)$ within an error bound 2^{-n}, as a function of the input x and the precision of the approximation n.

Let's briefly recall what a function-oracle Turing machine is (we give an informal description: details can be found in [HU79,Ko91]). For any x in the domain of f, the oracle is a computable sequence ϕ such that for all $n \in \mathbb{N}$, $|\phi(n) - x| < 2^{-n}$. The machine is a Turing machine equiped with an additional query tape, and two additional states. When the machine enters in the query state, it replaces the current string s in the query tape by the string $\phi(s)$, where ϕ is the oracle, moves the head to the first cell of the query tape, and switches to the answer state. This is done in one step of the computation. We say that the time (resp. space) complexity of f on its domain is bounded by a function b if there is a function-oracle Turing machine which, for any x in the domain of f and an oracle ϕ that converges to x, computes an approximation of $f(x)$ with precision 2^{-n} in a number of steps (resp. amount of tape) bounded by $b(x, n)$. Then, for space complexity we define:

Definition 4. *Given a set of functions* S, $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S) = \{f : \mathbb{R}^n \to \mathbb{R}$ *s.t. the space complexity of* f *is bounded by some function in* $S\}$.

Therefore, if a real recursive class $\mathcal{C}$ is contained in $\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S)$, then S can be considered a space complexity upper bound for $\mathcal{C}$.

Suppose that a function f can be successively approximated within an error 2^{-n} in a certain amount of space. Then, if f is integer, it just has to be approximated to an error less than $1/2$ to know its value exactly. Therefore, if a real recursive class $\mathcal{C}$ is computable in space bounded in $\mathcal{S}$, then the discretization of $\mathcal{C}$ is also computable in space bounded in S. Formally,

Proposition 9. *Let $\mathcal{C}$ be a real recursive class. If $\mathcal{C} \subset \mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(S)$, then $\mathcal{F}_{\mathbb{N}}(\mathcal{C}) \subset \mathcal{F}\mathrm{SPACE}(S)$.*

Given the two conventions established in Definition 3 and Definition 4, we will show upper and lower bounds on some real recursive classes. We can use the closure properties described in the last section to compare discretizations of real recursive classes with standard recursive classes. For instance, since $[0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}]$ contains extensions of the zero function, successor, projections, and the cut-off function, and is closed under and composition and iteration, then we have the following upper bound for the primitive recursive functions (see [CMC00]):

Proposition 10. $\mathcal{PR} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, U; \mathrm{COMP}, \bar{\mathrm{I}}])$.

Note that the same inductive proof works for $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{I}]$. Therefore, $\mathcal{PR} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{I}])$.

The elementary functions $\mathcal{E}$, which are closed under bounded sums and products, are a well-known class in recursion theory. All elementary functions are computable in elementary time or space, i.e., in time or space bounded by a tower of exponentials. As a matter of fact, the elementary functions are the smallest known class closed under time or space complexity [Odi00]. From Propositions 6 and 7 it follows that

Proposition 11. *For all $k \geq 0$, $\mathcal{E} \subset \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}])$.*

In addition, all functions in $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}]$ are computable in elementary space (or time) [CMC02]. Formally,

Proposition 12. *For all $k > 1$, $[0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}] \subset \mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(\mathcal{E})$.*

Combining this with Proposition 9 and Proposition 11, we conclude that:

Proposition 13. *For all $k > 1$, $\mathcal{E} = \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, U; \mathrm{COMP}, \mathrm{LI}])$.*

which gives an analog characterization of the elementary functions. It is interesting that linear integration alone gives extensions to the reals of all elementary functions, since these are all the functions that can be computed by any practically conceivable digital device. Notice that the above results on $\mathcal{E}$ can be generalized to the levels $\mathcal{E} = \mathcal{E}^3$, $\mathcal{E}^4$, ... of the Grzegorczyk hierarchy if we include in our model a non linear differential operator that generates total functions [CMC02].

In recursion theory, several forms of bounded recursion are widely used, namely to obtain characterization of low complexity classes [Clo99]. In bounded recursion, an *a priori* bound is imposed on the function to be defined with the recursion scheme. Similarly, we can consider the following operator on real functions:

BI (Bounded integration). Given functions $f_1, \ldots, f_m$ of arity n, $g_1, \ldots, g_m$ of arity $n + 1 + m$, and b of arity $n + 1$, if $(h_1, \ldots, h_m)$ is the unique function that satisfies the equations $\boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{f}(\boldsymbol{x})$, $\partial_y \boldsymbol{h}(\boldsymbol{x}, y) = \boldsymbol{g}(\boldsymbol{x}, y, \boldsymbol{h}(\boldsymbol{x}, y))$, and $||\boldsymbol{h}(\boldsymbol{x}, y)|| \leq b(\boldsymbol{x}, y)$ on $\mathbb{R}^{n+1}$, then $h = h_1$ of arity $n + 1$ is defined.

Let's consider the class $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$.[3] All its functions are defined everywhere since this is true for the basic functions and its operators preserve that property. The *a priori* bound on the integration operation strongly restricts this class. All functions in the class $[0, 1, -1, \theta_k, \times, U; \mathrm{COMP}, \mathrm{BI}]$ and its derivatives (for $k > 1$) are bounded by polynomials. Moreover, all functions computable in linear space have extensions in that class:

[3] Given an appropriate bound, the binary product $h(x, y) = xy$ could be easily defined with bounded integration: $h(x, 0) = 0$, and $\partial_y h(x, y) = U_1^2(x, y) = x$. However, no other basic function grows as fast as the binary product, so this needs to be included explicitly in the class.

Proposition 14. *For all $k \geq 0$, $\mathcal{F}$LINSPACE $\subset \mathcal{F}_{\mathbb{N}}([0,1,-1,\theta_k,\times,U;\mathrm{COMP},\mathrm{BI}])$.*

Proof. (Sketch) Let's denote $[0,1,-1,\theta_k,\times,U;\mathrm{COMP},\mathrm{BI}]$ by $\mathcal{B}_0$. Ritchie [Rit63] proved that the set of integer functions computable in linear space is the function algebra $[0,S,U,\times;\mathrm{COMP},\mathrm{BREC}]$, usually denoted by $\mathcal{E}^2$, where BREC is bounded recursion. It is easy to verify that $\mathcal{B}_0$ contains extensions to the reals of zero, successor, projections, and binary product. Since $\mathcal{B}_0$ is closed under composition, we just have to verify that $\mathcal{B}_0$ is closed under bounded recursion in a weak sense. But since all functions in $\mathcal{B}_0$ have polynomials bounds, then this can be done with techniques similar to [Bra95] using bounded integration instead of integration. Details can be found in [Cam01]. □

The Ritchie hierarchy [Rit63] is one of the first attempts to classify recursive functions in terms of computational complexity. The Ritchie classes, which range from $\mathcal{F}$LINSPACE to the elementary functions, are the sets of functions computable in space bounded by a tower of exponentials of fixed height. Next we describe a hierarchy of real recursive classes where the first level is $[0,1,-1,\theta_k,\times,U;\mathrm{COMP},\mathrm{BI}]$ (see above), and the n-th level is defined by allowing n nested applications of the linear integration operator. In each level of the hierarchy, indefinite integrals are freely used. As in the Ritchie hierarchy, composition is restricted. In [Rit63], the arguments of each recursive function are of two possible types: free and multiplicative. If f is multiplicative in the argument x, then f grows at most polynomially with x. The restricted form of composition forbids composition on two free arguments. For instance, if 2^x+y is free in x and multiplicative in y, then the composition $z=2^x+y$ with $x(t)=2^t$, which is free in t, is not allowed while the composition $z=2^x+y$ with $y(t)=2^t$ is. We denote this restricted composition by RCOMP and define the following hierarchy of real recursive classes (see [Cam01] for details):

Definition 5. *(The hierarchy $\mathcal{S}_n$) For all $n \geq 0$, $\mathcal{S}_n = [\mathcal{B}_0;\mathrm{RCOMP},\mathrm{INT},n\cdot\mathrm{LI}]$, where $\mathcal{B}_0=[0,1,-1,\theta_k,\times,U;\mathrm{COMP},\mathrm{BI}]$ for any fixed integer $k>2$, and where the notation $n\cdot\mathrm{LI}$ means that the operator* LI *can be nested up to n times.*

A few remarks are in order. First, all the arguments of a function h defined with linear integration, from any functions f,g of appropriate arities, are free. For instance, we are not allowed to compose the exponential function with itself, since its argument is free. Second, since solutions of linear differential equations $y'(t)=g(t)y(t)$ are always bounded by an exponential in g and t, and at most n nested applications of linear integration are allowed, then all functions in $\mathcal{S}_n$ have bounds of the form $\exp^{[n]}(p(x))$, where p is a polynomial. Even if the composition $\exp(\exp(x))$ is not permitted, towers of exponentials $\exp^{[n]}=\exp\circ\ldots\circ\exp$ can be defined in $\mathcal{S}_n$:

Example 3. ($\exp^{[n]}\circ p\in\mathcal{S}_n$). Let $u_i(\boldsymbol{x},y)=\exp^{[i]}(p(\boldsymbol{x},y))$ for $i=1,\ldots,n$, where p is a polynomial. Then, the functions u_i are defined by the set of linear differential equations

$$\partial_y u_1 = u_1\cdot\partial_y p \qquad \ldots \qquad \partial_y u_n = u_n\cdot u_{n-1}\cdots u_1\cdot\partial_y p$$

with appropriate initial conditions. Thus u_n can be defined with up to n nested applications of LI and, therefore, $\exp^{[n]}\circ p\in\mathcal{S}_n$.

Next we relate the $\mathcal{S}_n$ hierarchy to the exponential space hierarchy (details of the proofs can be found in [Cam01]). Consider the following set of bounding functions:

$$2^{[n]}=\{b_k:\mathbb{N}\to\mathbb{N}\text{ s.t. }k>0, b_k(m)=2^{[n]}(km)\text{ for all }m\}.$$

On one hand, $\mathcal{S}_n$ has the following upper bound:

Proposition 15. *For all $n\geq 0$, $\mathcal{S}_n\subset\mathcal{F}_{\mathbb{R}}\mathrm{SPACE}(2^{[n+1]})$.*

Proof. (Sketch) All functions in $\mathcal{S}_n$, and its first and second derivatives, are bounded by $2^{[n]}\circ p$, where p is some polynomial. This follows from the fact that all basic functions in $\mathcal{S}_n$ have such property (this is why we restrict k in the Definition 5) and the operators of $\mathcal{S}_n$ preserve it. Then, using numerical techniques we show how to approximate a function defined by composition or bounded integration in $\mathcal{S}_0 = \mathcal{B}_0$. Given the bounds on the functions in $\mathcal{S}_0$ and their first derivative, composition can be computed in a straightforward manner, without increasing the space bounds. The major difficulty has to do with integration. We have to use an exponential amount of space to achieve a sufficiently good approximation. In fact, the standard techniques for numerical integration (Euler's method) require a number of steps which is exponential in the bounds on the derivatives of the functions we want to approximate [Hen62]. Since the bounds for functions in $\mathcal{S}_0$ are polynomial, the required number of steps N in the numerical integration is exponential. Thus all functions in $\mathcal{S}_0$ can be approximated in exponential space. Finally, we follow the same approach for other levels of the $\mathcal{S}_n$ hierarchy, where restricted composition replaces composition, and linear integration replaces bounded integration. □

On the other hand, all functions computable in space bounded by $2^{[n-1]}$ have extensions in $\mathcal{S}_n$. Formally,

Proposition 16. *For all* $n \geq 1$, $\mathcal{F}\text{SPACE}(2^{[n-1]}) \subset \mathcal{F}_{\mathbb{N}}(\mathcal{S}_n)$.

Proof. (Sketch) As in [Rit63], we show that $\mathcal{F}\text{SPACE}(2^{[n-1]})$ has a recursive definition, using restricted composition and a restricted form of bounded recursion. The following step is to define this restricted form of bounded recursion with bounded sums. Let's suppose that $f \in \mathcal{F}\text{SPACE}(2^{[n-1]})$ is defined by bounded recursion. Then, we can encode the finite sequence $\{f(1), \ldots, f(n)\}$ as an integer (using for instance prime factorization), and replace bounded recursion by a bounded quantification over those encodings.[4] We use the fact that bounded quantifiers can be defined with bounded sums and cut-off subtraction. However, the bound on the encoding of the sequence $\{f(1), \ldots, f(n)\}$ is exponential on the bound on f. Therefore, we need an additional level of exponentials to replace bounded recursion by bounded sums. Finally, we know from Proposition 6 that $\mathcal{S}_n$ is closed under bounded sums, and contains cut-off subtraction as well. □

Unfortunately, we were not able to eliminate bounded integration from the definition of $\mathcal{B}_0$, neither were we able to show that $\mathcal{F}\text{SPACE}(2^{[n]})$ is precisely $\mathcal{F}_{\mathbb{N}}(\mathcal{S}_n)$. We believe those issues are related with the open problem:

$$\mathcal{L}^2 \overset{?}{=} \mathcal{E}^2,$$

where $\mathcal{L}^2 = [0, S, U, \dot{-}; \text{COMP}, \text{BSUM}]$ is defined with bounded sums and is known as Skolem's lower elementary functions.[5] We consider instead the following problem:

$$\mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, +, U; \text{COMP}, \text{INT}]) \overset{?}{=} \mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, \times, U; \text{COMP}, \text{BI}]).$$

At first sight, it seems that the equality above is false, since bounded integration is more general than indefinite integrals. However, the problem only concerns the discretizations of the analog classes. One could try to use results of the theory of differential equations to show directly that bounded integration is reducible, up to a certain error, to a finite sequence of indefinite integrals. It is known that solutions of general differential equations, $y'(t) = f(t, y)$ and $y(0) = y_0$, can be uniformly approximated by sequences of integrals, given some broad conditions that guarantee existence and uniqueness [Arn96,Har82]. However, that result, which is based on Picard's successive approximations, requires a sequence of integrals whose length increases with t. Since all functions in $[0, 1, -1, \theta_k, \times, U; \text{COMP}, \text{BI}]$ and its derivatives are polynomially bounded, it might be possible to find a finite approximation for bounded integration, which would be sufficient to approximate functions which range on the integers. Notice that the standard numerical techniques (Euler's method) to approximate the solution of $y'(t) = f(t, y)$ and $y(0) = y_0$ require a number of approximation steps which are exponential in the bounds on the derivative, while Picard's method only needs a polynomially long sequence of indefinite integrals, if the bounds on the derivatives are polynomial.

If the equality above is true, and if $\mathcal{F}_{\mathbb{N}}([0, 1, -1, \theta_k, +, U; \text{COMP}, \text{INT}]) \subset \mathcal{L}^2$, then we would obtain a chain of inclusions that would show that $\mathcal{L}^2 = \mathcal{E}^2$. These remarks above establish a connection between the theory of real recursive functions and computational complexity that would be interesting to explore.

[4] We follow a known technique in recursion theory (see [Ros84]).

[5] Recall that $\mathcal{E}^2 = [0, S, U, \times; \text{COMP}, \text{BREC}]$ and is precisely $\mathcal{F}\text{LINSPACE}$. Notice that $\mathcal{L}^2 \subset \mathcal{E}^2$.

5 Final remarks

We described some results on real recursive functions and we listed some open problems and directions for further research. We believe that recursion theory over the reals is not only an interesting area of research by itself, but it is also related to other areas such as computational complexity, numerical analysis or dynamical systems.

We mentioned possible links to computational complexity in the last section. It would be interesting to look at real recursive classes related to low time complexity classes. For instance, it is unlikely that the class in Proposition 6 is contained in $\mathcal{F}_{\mathbb{R}}\text{TIME}(P)$, where P is the set of polynomials, since if $\mathcal{F}_{\mathbb{R}}\text{TIME}(P)$ is closed under INT, then $\#P = \mathcal{F}\text{PTIME}$ [Ko91]. Therefore, schemes of integration other than the ones we described in this paper have to be explored to find analogues to $\mathcal{F}$PTIME or other low time complexity classes.

We would like to clarify the connections between real recursive functions and dynamical systems. It is known that the unary functions in $[0, 1, -1, U; \text{COMP}, \text{I}]$ are precisely the solutions of equations $\boldsymbol{y}' = p(\boldsymbol{y}, x)$, where p is a polynomial [GC]. We conjecture that $[0, 1, -1, U; \text{COMP}, \text{LI}]$ corresponds to the family of dynamical systems $\boldsymbol{y}' = f(\boldsymbol{y}, x)$, where each f_i is linear and depends at most on $x, y_1, \ldots, y_i$. Given such canonical representations of classes of real recursive functions, one could investigate their dynamical properties.

Acknowledgements. I thank Félix Costa for inviting me to participate in the Advanced School and Workshop on Soft Computing and Complex Systems. I also thank Cristian Calude, Michael Dinneen and Ferdinand Peper for inviting me to UMC'02, where I first presented the contents of this paper, and Amílcar Sernadas, Daniel Graça, Yuzuru Sato for their valuable suggestions. This work was supported by FCT via the Laboratório de Modelos e Arquitecturas Computacionais, grant PRAXIS XXI/BD/18304/98, and more recently the Center for Logic and Computation within the project ConTComp. I also thank the Instituto Superior de Agronomia, the Fundação Luso-Americana para o Desenvolvimento and the Santa Fe Institute for their support.

References

[Arn96] V. I. Arnold. *Equations Différentielles Ordinaires.* Editions Mir, 5 ème edition, 1996.

[Bab73] A. Babakhanian. Exponentials in differentially algebraic extension fields. *Duke Math. J.*, 40:455–458, 1973.

[Bra95] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995.

[BSS89] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21:1–46, 1989.

[Cam01] M.L. Campagnolo. *Computational complexity of real recursive functions and analog circuits.* PhD thesis, Instituto Superior Técnico, 2001.

[Cam02] M.L. Campagnolo. The complexity of real recursive functions. In C.S. Calude, M.J. Dinneen, and F. Peper, editors, *Unconventional Models of Computation, UMC'02*, number 2509 in LNCS, pages 1–14. Springer-Verlag, 2002.

[Clo99] P. Clote. Computational models and function algebras. In E.R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. Elsevier, 1999.

[CMC00] M.L. Campagnolo, C. Moore, and J.F. Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642–660, 2000.

[CMC02] M.L. Campagnolo, C. Moore, and J.F. Costa. An analog characterization of the Grzegorczyk hierarchy. *Journal of Complexity*, 4(18):977–1000, 2002.

[Cut80] N. J. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.

[GC] D. S. Graça and J. F. Costa. Analog computers and recursive functions over the reals. To appear in the *Journal of Complexity.*

[Grz55] A. Grzegorczyk. Computable functionals. *Fund. Math.*, 42:168–202, 1955.

[Har82] P. Hartman. *Ordinary Differential Equations.* Birkhauser, 2nd edition, 1982.

[Hen62] P. Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* Wiley, New York, 1962.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addisson-Wesley, 1979.

[KM99] P. Koiran and C. Moore. Closed-form analytic maps in one or two dimensions can simulate Turing machines. *Theoretical Computer Science*, 210:217–223, 1999.

[Ko91] K.-I. Ko. *Complexity Theory of Real Functions.* Birkhauser, 1991.

[Moo96] C. Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162:23–44, 1996.

[Moo98] C. Moore. Finite-dimensional analog computers: flows, maps, and recurrent neural networks. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computation*, DMTCS. Springer-Verlag, 1998.

[Odi89] P. Odifreddi. *Classical Recursion Theory.* Elsevier, 1989.

[Odi00] P. Odifreddi. *Classical Recursion Theory II.* Elsevier, 2000.

[Rit63] R.W. Ritchie. Classes of predictably computable functions. *Transactions Amer. Math. Soc.*, 106:139–173, 1963.

[Ros84] H.E. Rose. *Subrecursion: Functions and Hierarchies.* Clarendon Press, 1984.

[TW98] J. Traub and A.G. Werschulz. *Complexity and Information.* Cambridge University Press, 1998.

Evolutionary Multi-Criterion Optimisation

Carlos M. Fonseca
CSI – Centro de Sistemas Inteligentes
Faculdade de Ciências e Tecnologia, Universidade do Algarve
Campus de Gambelas, 8005-139 FARO, Portugal
e-mail: cmfonsec@ualg.pt

Abstract

Evolutionary algorithms, a broad class of optimisation algorithms inspired in the process of natural evolution, are introduced, and an artificial model of evolution is given which encompasses most established evolutionary algorithm variants. This model is then reinterpreted in the light of multiobjective optimisation, and a link to decision analysis is established.

1 Introduction

Nature has been a major source of inspiration and metaphors for scientific and technical development. It is not difficult to identify the links between the ear and the microphone, the eye and the camera, the bat and the sonar system, the brain and artificial neural networks, and so forth. Similarly, the process of natural evolution has inspired a growing amount of research in artificial systems, only made possible by the increasing availability of modern computing power.

Evolutionary optimisation is a term used to describe a broad class of optimisation algorithms inspired in the process of natural evolution. Such evolutionary algorithms (EAs) have been applied with various degrees of success to many difficult optimisation problems in engineering and operations research, among several other areas.

Many practical optimisation problems involve, not a single objective, but a number of possibly competing objectives, or criteria. Although different objectives may usually be combined by means of aggregating functions to produce a single cost measure, it is not always easy, or even appropriate, to define such a function. In the absence of information about the relative importance of individual objectives, the optimisation problem may still be approached, but will generally admit no unique solution. Rather, a number of optimal solutions may exist, in the sense that each such solution may be improved with respect to some criteria only at the expense of degradation in other criteria.

In this paper, evolutionary optimisation is introduced in the more general context of evolutionary processes, after reviewing some relevant problem solving concepts. Then, an artificial model of evolution is given which encompasses most established EA variants. Finally, that model is reinterpreted so as to accommodate multiple criteria, and a link to decision analysis is established.

2 Problem solving

Evolutionary algorithms may be defined as a broad class of computational methods inspired in the process of natural evolution, which are aimed at solving difficult problems. Before considering EAs in more detail, it is worth reviewing some concepts related to problem solving.

2.1 What is a problem?

Definition 1 (Abstract problem) *An* abstract problem *Q is a binary relation on a set I of problem* instances *and a set S of problem* solutions *[1].*

Considering the Travelling Salesman Problem (TSP) as an example, an *instance* consists of a set of cities and of the distances between them. A *solution*, on the other hand, consists of a sequence of cities, which describes the order in which they should be visited. Note that this view of a problem is very general, and that one is often interested in more restricted classes of problems.

Definition 2 (Decision problem) *An abstract problem is called a* decision problem *if it has a* yes/no *solution, i.e. $S = \{0,1\}$ [1].*

Definition 3 (Optimisation problem) *An abstract problem is called an* optimisation problem *if it consists of finding* minimal *or* maximal *elements [2] of a set S under a given* preorder $\preceq$.

Returning to the previous example, the TSP is an optimisation problem, as it consists of finding a tour of minimum length. Here, the preorder $\preceq$ on S, the set of all possible tours, may be defined by referring to the tour length as a cost function f of each city sequence $x \in S$:

$$\forall x_1, x_2 \in S,\ x_1 \preceq x_2 \Leftrightarrow f(x_1) \leq f(x_2)$$

Note also that, given a TSP, the problem of whether or not a tour exists which is shorter than a given length (or than a given tour) is a decision problem. One may attempt to solve such a decision problem by considering a candidate solution to the original optimisation problem, $x \in S$, and *verifying* whether its cost $f(x)$ is indeed less than or equal to the given bound. If so, the decision problem is know to have an affirmative solution. Otherwise, it remains unsolved.

In general, optimisation problems can be recast as decision problems in the way just described.

2.2 Encodings

The difficulty of a problem is inherently related to the time needed to solve it, regardless of its type. One important issue in discussing problem difficulty is that, to be solved on a computer, an abstract problem instance must be represented in some way.

Definition 4 (Encoding) *An* encoding *of a set S of abstract objects is a mapping e from S to the set of binary strings [1].*

Definition 5 (Concrete problem) *A problem is called a* concrete problem *if its instance set I is the set of binary strings [1].*

It is important to note that the time taken by a computer to solve a concrete problem my depend heavily on the underlying encoding. Thus, complexity theory restricts itself to concrete problems, and concrete decision problems in particular. Although complexity theory will not be discussed further here, one should realise that if an optimisation problem can be solved quickly, then so can any associated decision problem. Equivalently, if a decision problem is hard to solve, the related optimisation problem will also be hard.

As it will be discussed later, encodings may play a very important role in evolutionary optimisation.

2.3 Evolutionary algorithms as approximation algorithms

When a given optimisation problem cannot be solved exactly in an acceptable amount of time, it may still be possible to find approximate solutions, e.g. by verifying given candidate solutions against the best solution known to date. Indeed, EAs act as optimisers by

1. generating candidate solutions
2. evaluating them
3. using the information thus gained to generate new, possibly better, candidate solutions

Together with methods such as *Tabu Search*, *Simulated Annealing*, *Stochastic Local Search*, and *Ant Colony Optimisation*, among others, EAs integrate a class of approximation techniques which has become known as *Metaheuristics*.

3 Evolutionary processes

The process of natural evolution has inspired a growing amount of research in artificial systems. The resulting class of computational methods which simulate various aspects of natural evolution became known as Evolutionary Algorithms, having attracted interest from biology, chemistry, economics, engineering and mathematics. The area emerged in the late 1960s in Europe [3, 4] and the US [5, 6], motivated by a desire to advance the state of the art in optimisation, adaptation, and machine learning. It became popular in the 1990s due, to a great extent, to the publication of Goldberg's book [7], and has continued to grow since then.

As an optimisation process, evolution has many interesting features. In particular, individuals are selected based on how well they function, and not based on the mechanisms which account for their functionality. Thus, they tolerate a limited understanding of how existing solutions may be improved, allowing problems previously considered intractable to be approached.

Neo-Darwinism is currently the most widely accepted paradigm of natural evolutionary [8]. It is based on the four essential processes of reproduction, mutation, competition and selection. Reproduction consists of individuals being capable of generating offspring similar to themselves (either sexually or asexually), whereas mutation accounts for replication errors during reproduction. Competition arises as the number of individuals in a population grows in a resource-limited environment, and selection consists of only certain individuals, through competition, actually being able to reproduce. Since offspring tend to be similar to their parents, selection effectively shapes populations as they evolve.

3.1 Population

In the neo-Darwinian paradigm, individuals can be understood as a duality. The genetic programme, or *genotype*, consists of an *encoded* representation of the individual at the chromosome level. Individual traits, on the other hand, are expressed by executing, or decoding, the genotype. Expressed traits are also called the *phenotype*, and usually vary as a complex non-linear function of the genotype, and of its interaction with the environment. In particular, there are usually no one-gene one-trait relationships. A single gene may simultaneously affect many phenotypic traits (*pleiotropy*) and a single phenotypic trait may be affected by the interaction of several *genes* (*polygeny*).

In artificial evolutionary systems, however, the encodings used are usually simple and concise, and seldom implement pleiotropy. Polygeny usually occurs, though, especially due to the lower cardinality of the alphabets tend to be used to encode the genotype.

3.2 Selection

Individuals are selected based on their expressed traits. Selection determines the survival of the best individuals and, consequently, their opportunity to generate offspring. Individuals which produce more offspring are considered *fitter* than others. Indeed, in natural systems, fitness is *expressed*: individuals are fit *because* they generate offspring. On the contrary, in artificial evolutionary systems, fitness is usually *assigned* to individuals based on some criterion, e.g., the cost function which defines an optimisation problem. This is perhaps one of the fundamental differences between natural evolution and evolutionary optimisation. The reproductive advantage of the best individual in a population with respect to the population's average is known as *selective pressure*.

Selection may be implemented in several ways. In *generational* selection, individuals reproduce all at the same time, and offspring replace the whole of the parent population, never competing with it. This is akin to the reproductive cycle of some insects, for example, where parents die before offspring are born. In an alternative model of selection, parents may be selected to reproduce at any time, and the offspring they generate are inserted in the parent population and forced to compete with it (*incremental* selection).

Another aspect of selection is whether it is *stochastic*, as it is common in natural systems, or *deterministic*, as in animal breeding, for example. One interesting property of (stochastic) selection, which can be observed in Nature as well as on the computer, is known as *genetic drift*, and consists of the tendency finite populations exhibit to evolve towards a single type of solution even when equivalent alternatives exist [9]. On the computer, genetic drift may be controlled in some circumstances by implementing *niche induction* mechanisms [10] such as *fitness sharing* and *crowding*.

3.3 Heredity

Evolution does not arise out of selection alone. One fundamental aspect of evolution is that individual replication is not perfect, i.e., individuals are not exactly like their parents but differ from them to a certain extent. As reproduction occurs at the genotypic level, the basic assumptions are that:

1. Offspring are similar to their parents at the genotypic level (heredity).

2. Good individuals have similar genotypes.

On the computer, as in Nature, heredity may assume several forms:

Mutation Random alteration of only small parts of individual genotypes

Recombination Production of offspring from the genotypic material of two (or possibly more) parents

Learning Incorporation of knowledge acquired at higher levels into the an individual's representation. This may be seen as *Lamarckian* evolution, *memetic* evolution, or even as "genetic engineering".

Despite the fact that there can be no evolution without variation, variation must be controlled. The basic idea is that selection must be able to recover from any deleterious variation. The amount of variation beyond which evolution no longer occurs is known as the error threshold [11].

```
Evolutionary Algorithm:
  1. Generate initial population (genotypes)
  2. While not happy do
       (a) decode individuals (phenotypes)
       (b) evaluate individuals (cost)
       (c) assign reproductive ability (fitness)
       (d) select individuals based on fitness
       (e) produce offspring (genotypes)
       (f) insert offspring into the population
```

Figure 1: An artificial model of evolution.

3.4 Viability

Finally, it is not sufficient either for good individuals to produce many offspring, as their offspring must be fit as well for evolution to successfully occur. Whether or not this is the case depends on the optimisation problem itself, on the genotypic encoding and on the variation mechanisms which manipulate it. Electing a good combination of encoding and variation operators for a given problem continues to be perhaps the greatest challenge in evolutionary optimiser design.

4 An artificial model of evolution

The various concepts introduced above can now be combined to produce a general artificial model of evolution, as depicted in Figure 1.

In an evolutionary optimisation setting, the initial population is typically drawn at random from a suitable encoding of the solution set S. Encodings commonly found in the literature include binary strings, n-ary strings, permutations, graphs (and especially trees), and combinations of these, depending on the problem considered.

Individual genotypes are then decoded to yield candidate solutions in (a subset of) the solution space S. Encodings are usually such that each genotype typically decodes into a unique phenotype, although it is possible to consider encodings where this is not always the case [12].

4.1 Evaluation and fitness assignment

Once candidate solutions in S have been obtained, individuals are evaluated based on the preorder which defines the optimisation problem or on a cost function, if one is given. Provided that the preorder $\preceq$ is such that all individuals are comparable (i.e., $\forall a, b \in S$, $a \preceq b \vee b \preceq a$), evaluation may consist of no more than sorting the population and noting individual ranks.[13]). Alternatively, the cost function may be evaluated at each individual. This is the usual single-objective optimisation case.

Evaluated solutions are then assigned a fitness value. Fitness may be assigned based on rank (*ranking* [13]) or as a function of cost function values (*scaling*). Scaling is the more traditional approach. Raw fitness is calculated as a monotonic function of the cost, offset by a certain amount, and then linearly scaled. The first difficulty arises at this stage: whilst scaling aims to preserve the relative performance between different individuals, both the initial transformation and the subsequent offsetting can significantly affect the fitness ultimately assigned to each individual.

With scaling, an individual much stronger than all the others may be assigned a very large fitness and, through selection, rapidly dominate the population. Conversely, the advantage of the best individual over the rest of the population will be minimal if most individuals perform more or less equally well, and the search will degenerate into an aimless walk.

Ranking addresses these difficulties by eliminating any sensitivity to the scale in which the problem is formulated. Since the best individual in the population is always assigned the same fitness, would-be "super" individuals can never reproduce excessively. Similarly, when all individuals perform almost equally well, the best individual is still unequivocally preferred to the rest (but this may be inappropriate if the objective function is contaminated with noise).

Rank-based fitness assignment is characterised by the choice of rank-to-fitness mapping, which is usually chosen to be linear or exponential. For a population of size N, ranking the best individual zero and the worst $N-1$, and representing rank by r and fitness by $\phi(r)$, these mappings can be written as follows:

Linear

$$\phi(r) = s - (s-1) \cdot \frac{2r}{N-1},$$

where s, $1 < s \leq 2$, is the fitness desired for the best individual. The upper bound on s arises because fitness must be non-negative for all individuals, while maintaining $\sum_{i=0}^{N-1} \phi(i) = N$.

Exponential

$$\phi(r) = \rho^r \cdot s,$$

where $s > 1$ is the fitness desired for the best individual, and ρ is such that $\sum_{i=0}^{N-1} \rho^i = N/s$. Since there is no upper-bound on s, the exponential mapping is somewhat more flexible than the linear.

For $1 < s \leq 2$, the main difference between linear and exponential rank-based fitness assignment is that the exponential mapping does not penalise the worst individuals as much, at the expense of assigning middle individuals fitness slightly less than average. As a consequence, exponential assignment generally contributes to a more diverse search.

4.2 Selection

A number of parents are selected from the population through a sampling mechanism, which may be deterministic or stochastic. A well-established sampling procedure is known as Stochastic Universal Sampling [14], and may be visualised as the result of spinning a roulette wheel with slots proportional in width to the fitness of the individuals in the population, and with multiple, equally spaced pointers (Figure 2). Once it stops, the number of pointers over each sector must be an integer, either immediately above or immediately below the corresponding desired number of offspring, guaranteeing minimum deviations from the desired fitness value. The replicates obtained in this way should be shuffled before the algorithm proceeds with recombination.

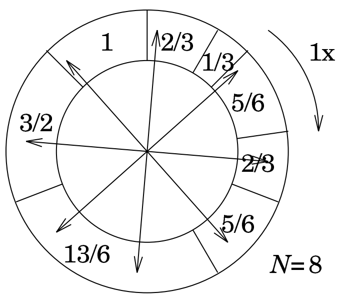


Figure 2: Stochastic Universal Sampling

4.3 Recombination and mutation

Offspring are produced from the parents selected, by manipulating them at the genotypic level. Parents may be recombined and/or mutated to generate offspring. A typical recombination operator for binary and other string chromosomes is single-point crossover, whereby two individuals exchange a portion (right or left) of their chromosomes to produce offspring, as illustrated in Figure 3. The crossover point is selected at random. Other recombination operators commonly used with binary strings are:

Double-point crossover Two crossover points are selected instead of one [15].

Uniform crossover Each bit is exchanged independently, with a given probability [16].

Shuffle crossover The chromosomes are shuffled before single-point crossover is applied, and consequently deshuffled [17].

Reduced-surrogate crossover The non-identical bits in the chromosomes are first identified, and one of the above crossover types applied to the smaller string thus defined [15]. This has the effect of guaranteeing the production of offspring different from their parents.

As for bit mutation (see Figure 4), it is most commonly implemented by independently flipping each bit in the chromosome with a given probability.

4.4 Reinsertion

Finally, the offspring produced are inserted in the population, replacing:

- random members of the parental population,
- the oldest members of the parental population,
- their own parents, or
- the least fit members of the parental population.

Actual reinsertion may occur

- unconditionally,

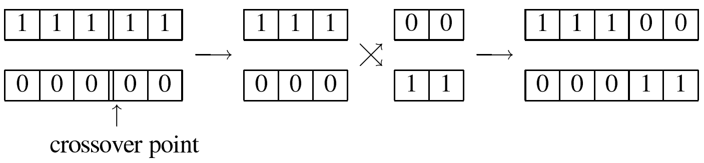


Figure 3: Single point crossover

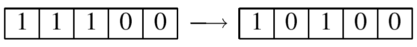


Figure 4: Bit mutation

- only if the offspring are fitter than the individuals they are to replace, or
- probabilistically, depending on whether or not the offspring are stronger than the individuals they are to replace.

Note that, by denying some individuals the possibility of reproducing further, reinsertion has ultimately the same effect as selection. The overall *selective pressure* imposed on the population is not only determined by the fitness assignment strategy, but is also affected by when and how reinsertion is performed. In particular, always replacing the least fit individuals in the population strongly increases the *effective*, as opposed to *assigned*, fitness differential between stronger and weaker individuals in the population. This is because, in addition to being less likely to be selected, weaker individuals tend to die earlier, thus participating in less selection trials than stronger ones. Reinsertion strategies which guarantee the preservation of the best individual are known as *elitist*.

5 Multiobjective optimisation

Practical problems are often characterised by several non-commensurable and often competing measures of performance, or objectives. The multiobjective optimisation problem may be stated as the problem of simultaneously minimising the n components f_i, $i = 1, \dots, n$, of a vector function $\mathbf{f}(x)$, with $x \in S$, where

$$\mathbf{f}(x) = (f_1(x), \dots, f_n(x)).$$

The problem usually has no unique, perfect (or Utopian) solution, but may admit a set of non-dominated, alternative solutions, known as the Pareto-optimal set [18]. Assuming a minimisation problem, dominance is defined as follows:

Definition 6 (Pareto dominance) *A real vector* $\mathbf{u} = (u_1, \dots, u_n)$ *is said to dominate* $\mathbf{v} = (v_1, \dots, v_n)$ *if and only if* $\mathbf{u}$ *is partially less than* $\mathbf{v}$ *(*$\mathbf{u} \;{}_{p}\!< \mathbf{v}$*), i.e.,*

$$\forall i \in \{1, \dots, n\}\ ,\ u_i \leq v_i \quad \wedge \quad \exists i \in \{1, \dots, n\} : \ u_i < v_i \,.$$

Definition 7 (Pareto optimality) *A solution* $x_{\mathrm{u}} \in S$ *is said to be Pareto-optimal if and only if there is no* $x_{\mathrm{v}} \in S$ *for which* $\mathbf{v} = \mathbf{f}(x_{\mathrm{v}}) = (v_1, \dots, v_n)$ dominates $\mathbf{u} = \mathbf{f}(x_{\mathrm{u}}) = (u_1, \dots, u_n)$.

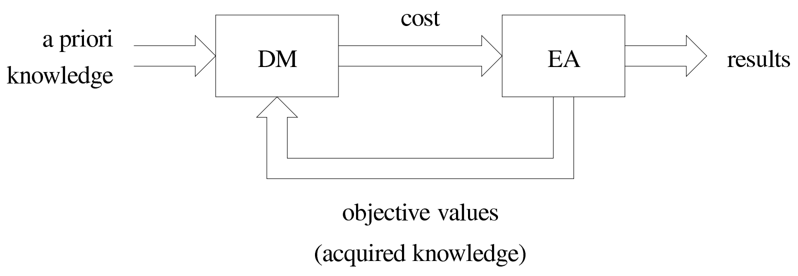


Figure 5: A general multiobjective evolutionary optimiser.

Pareto-optimal solutions are also called efficient, non-dominated, and non-inferior solutions. The corresponding objective vectors are simply called non-dominated. The set of all non-dominated vectors is known as the non-dominated set, or the trade-off surface, of the problem.

Alternatively, the multiobjective optimisation problem may be defined by specifying the preorder in Definition 3 as:

$$\forall x_1, x_2 \in S, \quad x_1 \preceq x_2 \Leftrightarrow f_i(x_1) \leq f_i(x_2)\ \forall i \in \{1, \ldots, n\}.$$

This greatly simplifies establishing a formulation of multiobjective evolutionary algorithms.

5.1 Multiobjective evolutionary algorithms

Definition 3 is general enough that it accommodates both single and multiobjective optimisation problems. On the other hand, the artificial model of evolution presented in Figure 1 was based simply on this definition. It is therefore clear that the main difference between a single-objective and a multiobjective evolutionary algorithm must lie in the individual evaluation step. In particular, some elements of S may now be incomparable and assigning a cost value to each candidate solution becomes a *decision analysis* problem.

A general multiobjective evolutionary optimiser may also be seen as the result of the interaction between between a Decision Maker (DM) and an Evolutionary Algorithm (see Figure 5). The EA generates a new set of candidate solutions according to the cost assigned to the current set of candidates by the DM. New candidate solutions, as they are evaluated provide new trade-off information which the DM can use to refine the current preferences. The EA sees the effect of any changes in the decision process, which may or may not result from taking recently acquired information into account, as an environmental change. The DM block represents any cost assignment strategy, which may range from that of an intelligent Decision Maker to a simple aggregating function approach.

Aggregating function approaches to multiobjective evolutionary optimisation, although useful and very common in the literature, do convert multiobjective optimisation problems into a single-objective problems, raising no particular issues as far as the EA formulation is concerned. A number of alternative approaches, known as population-based approaches [19], typically assign different objectives to different subsets of the population, so as to promote the emergence of good compromise solutions. Schaffer's pioneering work on *Vector Evaluated Genetic Algorithms* [20] falls in this category. A third class of approaches is based directly on the definition of Pareto-dominance, and includes most modern evolutionary multiobjective optimisers.

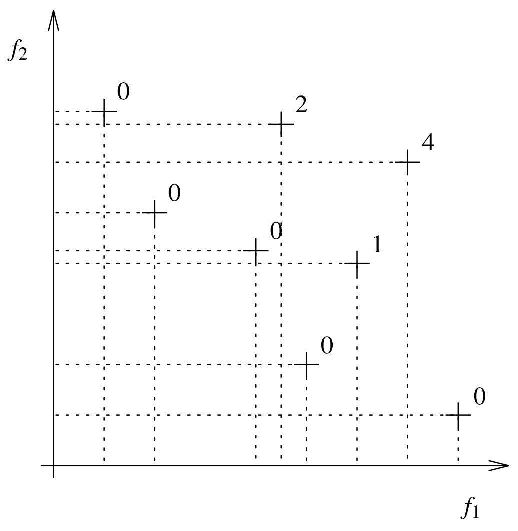


Figure 6: Pareto ranking

5.2 Pareto-based approaches

In the absence of information concerning the relative importance of the objectives, an individual can only be said to perform better than another if it dominates it. Therefore, non-dominated individuals should be assigned the same cost [7], e.g., zero. Deciding about the cost of dominated individuals is a more subjective matter. One alternative consists of assigning individuals a cost proportional to how many other individuals in the population dominate them (Figure 6), which also guarantees that non-dominated individuals are treated as desired. This is essentially the Pareto-ranking scheme proposed in [21].

Another popular Pareto-ranking scheme [7], also known as non-dominated sorting [22], consists of removing the non-dominated individuals (still ranked zero, for ease of comparison) from contention, finding the non-dominated individuals in the remaining population and assigning them rank 1, and so forth, until the whole population is ranked.

Both approaches guarantee that non-dominated individuals are all ranked best, and that all individuals are assigned better ranks than those individuals they dominate. However, the first ranking scheme does appear to be easier to interpret and analyse mathematically [23].

5.3 Incorporating preference information

When goal and/or priority information is available for the objectives, it may become possible to discriminate between some non-dominated solutions. For example, if degradation in objective components which meet their goals does not go beyond the goal boundaries, and results in the improvement of objective components which do not yet satisfy the corresponding goals, then it should be accepted. Similarly, in a dual priority setup [23], it is only important to improve on high priority objectives (i.e., constraints) until the corresponding goals are met, after which improvement should be sought for the remaining objectives. These considerations have been formalised in terms of a transitive relational operator (*preferability*), based on Pareto-dominance, but which selectively excludes objectives according to their priority and to whether or not they meet their goals.

For simplicity, only one level of priority will be considered here. The full, multiple priority version of the preferability operator is described in detail in [23]. Consider two objective vectors **u** and **v** and a goal vector **g**. Also, let the smile $\overset{u}{\smile}$ and the frown $\overset{u}{\frown}$ denote the components of **u** which meet their goals and those which do not, respectively. Assuming minimisation, one can write

$$\mathbf{u}^{\overset{u}{\smile}} \le \mathbf{g}^{\overset{u}{\smile}} \quad \wedge \quad \mathbf{u}^{\overset{u}{\frown}} > \mathbf{g}^{\overset{u}{\frown}},$$

where the inequalities apply componentwise. This is equivalent to

$$\forall i \in \overset{\mathbf{u}}{\smile},\ u_i \le g_i \quad \wedge \quad \forall i \in \overset{\mathbf{u}}{\frown},\ u_i > g_i$$

where u_i and g_i represent the components of **u** and **g**, respectively. Then, **u** is said to be preferable to **v** given **g** if and only if

$$(\mathbf{u}^{\overset{u}{\frown}}\ \mathrm{p}{<}\ \mathbf{v}^{\overset{u}{\frown}}) \vee \left\{ (\mathbf{u}^{\overset{u}{\frown}} = \mathbf{v}^{\overset{u}{\frown}}) \wedge \left[(\mathbf{v}^{\overset{u}{\smile}} \not\le \mathbf{g}^{\overset{u}{\smile}}) \vee (\mathbf{u}^{\overset{u}{\smile}}\ \mathrm{p}{<}\ \mathbf{v}^{\overset{u}{\smile}}) \right] \right\}$$

where **a** p< **b** denotes **a** dominates **b**. In other words, **u** will be preferable to **v** if and only if one of the following is true:

1. The violating components of **u** dominate the corresponding components of **v**.
2. The violating components of **u** are equal to the corresponding components of **v**, but **v** violates at least another goal.
3. The violating components of **u** are equal to the corresponding components of **v**, but **u** dominates **v** as a whole.

Like Pareto-dominance, this relation can be used to rank the individuals in a population by one of the methods described above.

6 Concluding remarks

In this paper, evolutionary optimisation was introduced, and an artificial model of evolution was given which encompasses most established EA variants. That model was then reinterpreted so as to accommodate multiple criteria optimisation problems. In the same light, it was shown how existing preferences may be combined with the notion of Pareto dominance by defining an alternative relation.

Much more could be said about evolutionary multi-criterion optimisation. Modern evolutionary multi-criterion optimisers have introduced additional mechanisms in the evolutionary process, including niche induction techniques, for maintaining diversity, and solution archiving, for preserving good solutions in the population. As a result, it has become increasingly less clear which algorithm works best in general, and increasing attention is being paid to experimental methodology for studying the performance of multiobjective optimisers.

In the mean time, existing evolutionary multi-criterion optimisers have been used in a wide range of industrial applications, making this one of the most promising research areas in evolutionary computing. The reader is referred to [24] for a comprehensive text book on Evolutionary Multiobjective Optimisation, and to [25, 26] for recent developments.

References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[2] P. Taylor, *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

[3] I. Rechenberg, *Evolutionsstrategie, Optimierung technischer Systeme nach Prinzipen der biologischen Evolution*. Stuttgart: frommann-holzboog, 1973. In German.

[4] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in *Genetic Algorithms: Proceedings of the Fourth International Conference* (R. K. Belew and L. B. Booker, eds.), pp. 2–9, San Mateo, California: Morgan Kaufmann, 1991.

[5] D. B. Fogel, *System Identification Through Simulated Evolution: A machine learning approach to modelling*. Needham, Massachusetts: Ginn Press, 1991.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.

[8] D. B. Fogel, "Principles of evolutionary processes," in *Evolutionary Computation 1: Basic Algorithms and Operators* (T. Baeck, D. Fogel, and T. Michalewicz, eds.), ch. 4, pp. 23–26, Institute of Physics Publishing, 2000.

[9] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in Grefenstette [27], pp. 41–49.

[10] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in Schaffer [28], pp. 42–50.

[11] G. Ochoa, I. Harvey, and H. Buxton, "Error thresholds and their relation to optimal mutation rates," in *European Conference on Artificial Life*, pp. 54–63, 1999.

[12] R. K. Belew, "Evolution, learning and culture: Computational metaphors for adaptive algorithms," *Complex Systems*, vol. 4, pp. 11–49, 1990.

[13] J. E. Baker, "Adaptive selection methods for genetic algorithms," in Grefenstette [29], pp. 101–111.

[14] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in Grefenstette [27], pp. 14–21.

[15] L. Booker, "Improving search in genetic algorithms," in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), Research Notes in Artificial Intelligence, ch. 5, pp. 61–73, London: Pitman, 1987.

[16] G. Syswerda, "Uniform crossover in genetic algorithms," in Schaffer [28], pp. 2–9.

[17] R. A. Caruana, L. J. Eshelman, and J. D. Schaffer, "Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (N. S. Sridharan, ed.), pp. 750–755, Morgan Kaufmann, 1989.

[18] A. Ben-Tal, “Characterization of Pareto and lexicographic optimal solutions,” in *Multiple Criteria Decision Making Theory and Application* (G. Fandel and T. Gal, eds.), vol. 177 of *Lecture Notes in Economics and Mathematical Systems*, pp. 1–11, Berlin: Springer-Verlag, 1980.

[19] C. M. Fonseca and P. J. Fleming, “An overview of evolutionary algorithms in multiobjective optimization,” *Evolutionary Computation*, vol. 3, Spring 1995.

[20] J. D. Schaffer, “Multiple objective optimization with vector evaluated genetic algorithms,” in Grefenstette [29], pp. 93–100.

[21] C. M. Fonseca and P. J. Fleming, “Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization,” in *Genetic Algorithms: Proceedings of the Fifth International Conference* (S. Forrest, ed.), pp. 416–423, San Mateo, CA: Morgan Kaufmann, 1993.

[22] N. Srinivas and K. Deb, “Multiobjective optimization using nondominated sorting in genetic algorithms,” *Evolutionary Computation*, vol. 2, Fall 1994. To appear.

[23] C. M. Fonseca and P. J. Fleming, “Multiobjective optimization and multiple constraint handling with evolutionary algorithms I: A unified formulation,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 1, pp. 26–37, 1995.

[24] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.

[25] E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, eds., *Evolutionary Multi-Criterion Optimization, First International Conference*, vol. 1993 of *Lecture Notes in Computer Science*. Springer, 2001.

[26] C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, eds., *Evolutionary Multi-Criterion Optimization, Second International Conference*, vol. 2632 of *Lecture Notes in Computer Science*. Springer, 2003.

[27] J. J. Grefenstette, ed., *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum, 1987.

[28] J. D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, 1989.

[29] J. J. Grefenstette, ed., *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, 1985.

Towards Solving the Grand Problem of AI[⋆]

Jürgen Schmidhuber
juergen@idsia.ch - http://www.idsia.ch/~juergen

IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland

Abstract. The grand problem of artificial intelligence (AI) as we understand it is to build a learning rational agent that optimally uses its limited computational (and other) resources to maximize expected reward in arbitrary, possibly unknown, real world environments. This problem is not solved yet, mainly due to the very issue of resource limitation. Still, the new millennium has brought very general, optimal algorithms for prediction, reinforcement learning and decision making in absence of resource limitations, and also practically feasible, optimal algorithms for search, incremental problem solving, and inductive inference based on Occam's razor. Here we review them, and point out what's missing.

1 Introduction

Remarkably, there is a theoretically *optimal* way of making predictions based on observations, rooted in the early work of Solomonoff and Kolmogorov [59, 27]. The approach reflects basic principles of Occam's razor: simple explanations of data are preferable to complex ones.

The theory of inductive inference quantifies what simplicity really means. Given certain very broad computability assumptions, it provides techniques for making optimally reliable statements about future events, given the past.

Once there is an optimal, formally describable way of predicting the future, we should be able to construct a machine that continually computes and executes action sequences that maximize expected or predicted reward, thus solving an ancient goal of AI research.

For many decades, however, AI researchers have not paid a lot of attention to the theory of inductive inference. Why not? There is another reason besides the fact that most of them have traditionally ignored theoretical computer science: the theory has been perceived as being associated with excessive computational costs. In fact, its most general statements refer to methods that are optimal (in a certain asymptotic sense) but incomputable. So researchers in machine learning and artificial intelligence have often resorted to alternative methods that lack a strong theoretical foundation but at least seem feasible in certain limited contexts. For example, since the early attempts at building a "General Problem Solver" [35, 42] much work has been done to develop mostly heuristic

⋆ *Based on Technical Report IDSIA-04-03 [55], January 2003, to appear in B. Goertzel and C. Pennachin, eds.: Artificial General Intelligence, 2003.*

machine learning algorithms that solve new problems based on experience with previous problems. Many pointers to *learning by chunking, learning by macros, hierarchical learning, learning by analogy,* etc. can be found in Mitchell's book [33] and Kaelbling's survey [26].

Recent years, however, have brought substantial progress in the field of *computable* and *feasible* variants of optimal algorithms for prediction, search, inductive inference, problem solving, decision making, and reinforcement learning in very general environments. In what follows, I will focus on results predominantly from my own lab at IDSIA.

Sections 3, 4, 7 relate Occam's razor and the notion of simplicity to the shortest algorithms for computing computable objects, and will concentrate on recent *asymptotic* optimality results for universal learning machines with infinite computational power, essentially ignoring issues of practical feasibility. With respect to the grand problem of AI (see abstract), what is missing here is to adapt such results to the case of limited resources, if possible.

Section 5 will then focus on computable (as opposed to noncomputable) optimal predictors based on our recent non-traditional simplicity measure which is *not* based on the shortest but on the *fastest* way of describing objects, and Section 6 will use this measure to derive non-traditional predictions concerning the future of our universe.

Sections 8, 9, 10 will finally address quite pragmatic issues and "true" time-optimality: given a problem and only so much limited computation time, what is the best way of spending it on evaluating solution candidates? In particular, Section 9 will present an optimally fast way of incrementally solving each task in a problem sequence, given a probability distribution (the *bias*) on programs computing solution candidates. Bias shifts are computed by program prefixes that modify the distribution on their suffixes by reusing successful code for previous tasks (stored in non-modifiable memory). No tested program gets more runtime than its probability times the total search time. In illustrative experiments, ours becomes the first general system to *learn* a universal solver for arbitrary n disk *Towers of Hanoi* tasks (minimal solution size $2^n - 1$). It demonstrates the advantages of incremental learning by profiting from previously solved, simpler tasks involving samples of a simple context free language. Section 10 will discuss first ideas on how to use this approach for building universal reinforcement learners, and mention a few important questions that are still open.

2 More Formally

What is the optimal way of predicting the future, given the past? Which is the best way to act such as to maximize one's future expected reward? Which is the best way of searching for the solution to a novel problem, making optimal use of solutions to earlier problems?

Most previous work on these old and fundamental questions has focused on very limited settings, such as Markovian environments where the optimal next action, given past inputs, depends on the current input only [26].

We will concentrate on a much weaker and therefore much more general assumption, namely, that the environment's responses are sampled from a computable probability distribution. If even this weak assumption were not true then we could not even formally specify the environment, leave alone writing reasonable scientific papers about it.

Let us first introduce some notation. B^* denotes the set of finite sequences over the binary alphabet $B = \{0, 1\}$, B^∞ the set of infinite sequences over B, λ the empty string, $B^\sharp = B^* \cup B^\infty$. x, y, z, z^1, z^2 stand for strings in $B^\sharp$. If $x \in B^*$ then xy is the concatenation of x and y (e.g., if $x = 10000$ and $y = 1111$ then $xy = 100001111$). For $x \in B^*$, $l(x)$ denotes the number of bits in x, where $l(x) = \infty$ for $x \in B^\infty$; $l(\lambda) = 0$. x_n is the prefix of x consisting of the first n bits, if $l(x) \geq n$, and x otherwise ($x_0 := \lambda$). log denotes the logarithm with basis 2, f, g denote functions mapping integers to integers. We write $f(n) = O(g(n))$ if there exist positive constants c, n_0 such that $f(n) \leq cg(n)$ for all $n > n_0$. For simplicity let us consider universal Turing Machines [64] (TMs) with input alphabet B and trinary output alphabet including the symbols "0", "1", and " " (blank). For efficiency reasons, the TMs should have several work tapes to avoid potential quadratic slowdowns associated with 1-tape TMs. The remainder of this paper assumes a fixed universal reference TM.

Now suppose bitstring x represents the data observed so far. What is its most likely continuation $y \in B^\sharp$? Bayes' theorem yields

$$P(xy \mid x) = \frac{P(x \mid xy)P(xy)}{P(x)} \propto P(xy) \tag{1}$$

where $P(z^2 \mid z^1)$ is the probability of z^2, given knowledge of z^1, and $P(x) = \int_{z \in B^\sharp} P(xz)dz$ is just a normalizing factor. So the most likely continuation y is determined by $P(xy)$, the *prior probability* of xy. But which prior measure P is plausible? Occam's razor suggests that the "simplest" y should be more probable. But which exactly is the "correct" definition of simplicity? Sections 3 and 4 will measure the simplicity of a description by its length. Section 5 will measure the simplicity of a description by the time required to compute the described object.

3 Prediction Using a Universal Algorithmic Prior Based on the Shortest Way of Describing Objects

Roughly fourty years ago Solomonoff started the theory of universal optimal induction based on the apparently harmless simplicity assumption that P is computable [59]. While Equation (1) makes predictions of the entire future, given the past, Solomonoff [60] focuses just on the next bit in a sequence. Although this provokes surprisingly nontrivial problems associated with translating the bitwise approach to alphabets other than the binary one — this was achieved only recently [20] — it is sufficient for obtaining essential insights. Given an observed bitstring x, Solomonoff assumes the data are drawn according to a recursive measure μ; that is, there is a program for a universal Turing machine that reads $x \in B^*$ and computes $\mu(x)$ and halts. He estimates the probability of the next bit (assuming there will be one), using the remarkable, well-studied, enumerable prior M [59,75,60,16,30]

$$M(x) = \sum_{\substack{program\ prefix\ p\ computes \\ output\ starting\ with\ x}} 2^{-l(p)}. \tag{2}$$

M is *universal*, dominating the less general recursive measures as follows: For all $x \in B^*$,

$$M(x) \geq c_\mu \mu(x) \tag{3}$$

where c_μ is a constant depending on μ but not on x. Solomonoff observed that the conditional M-probability of a particular continuation, given previous observations, converges towards the unknown conditional μ as the observation size goes to infinity [60], and that the sum over all observation sizes of the corresponding μ-expected deviations is actually bounded by a constant. Hutter (on the author's SNF research grant “"Unification of Universal Induction and Sequential Decision Theory") recently showed that the number of prediction errors made by universal Solomonoff prediction is essentially bounded by the number of errors made by any other predictor, including the optimal scheme based on the true μ [20].

Recent Loss Bounds for Universal Prediction. A more general recent result is this. Assume we do know that p is in some set P of distributions. Choose a fixed weight w_q for each q in P such that the w_q add up to 1 (for simplicity, let P be countable). Then construct the Bayesmix $M(x) = \sum_q w_q q(x)$, and predict using M instead of the optimal but unknown p. How wrong is it to do that? The recent work of Hutter provides general and sharp (!) loss bounds [21]:

Let $LM(n)$ and $Lp(n)$ be the total expected unit losses of the M-predictor and the p-predictor, respectively, for the first n events. Then $LM(n) - Lp(n)$ is at most of the order of $\sqrt{Lp(n)}$. That is, M is not much worse than p. And in general, no other predictor can do better than that! In particular, if p is deterministic, then the M-predictor soon won't make any errors any more.

If P contains *all* recursively computable distributions, then M becomes the celebrated enumerable universal prior. That is, after decades of somewhat stagnating research we now have sharp loss bounds for Solomonoff's universal induction scheme (compare recent work of Merhav and Feder [32]).

Solomonoff's approach, however, is uncomputable. To obtain a feasible approach, reduce M to what you get if you, say, just add up weighted estimated future finance data probabilities generated by 1000 commercial stock-market prediction software packages. If only one of the probability distributions happens to be close to the true one (but you do not know which) you still should get rich.

Note that the approach is much more general than what is normally done in traditional statistical learning theory, e.g., [66], where the often quite unrealistic assumption is that the observations are statistically independent.

4 Super Omegas and Generalizations of Kolmogorov Complexity & Algorithmic Probability

Our recent research generalized Solomonoff's approach to the case of less restrictive nonenumerable universal priors that are still computable in the limit [49, 52].

An object X is formally describable if a finite amount of information completely describes X and only X. More to the point, X should be representable by a possibly infinite bitstring x such that there is a finite, possibly never halting program p that computes x and nothing but x in a way that modifies each output bit at most finitely many times; that is, each finite beginning of x eventually *converges* and ceases to change. This constructive notion of formal describability is less restrictive than the traditional notion of computability [64], mainly because we do not insist on the existence of a halting program that computes an upper bound of the convergence time of p's n-th output bit. Formal describability thus pushes constructivism [5,1] to the extreme, barely avoiding the nonconstructivism embodied by even less restrictive concepts of describability (compare computability *in the limit* [17,39,15] and Δ_n^0-describability [41][30, p. 46-47]).

The traditional theory of inductive inference focuses on Turing machines with one-way write-only output tape. This leads to the universal enumerable Solomonoff-Levin (semi) measure. We introduced more general, nonenumerable, but still limit-computable measures and a natural hierarchy of generalizations of algorithmic probability and Kolmogorov complexity [49,52], suggesting that the "true" information content of some (possibly infinite) bitstring x actually is the size of the shortest nonhalting program that converges to x and nothing but x on a Turing machine that can edit its previous outputs. In fact, this "true" content is often smaller than the traditional Kolmogorov complexity. We showed that there are *Super Omegas* computable in the limit yet more random than Chaitin's "number of wisdom" *Omega* [10] (which is maximally random in a weaker traditional sense), and that any approximable measure of x is small for any x lacking a short description.

We also showed that there is a universal cumulatively enumerable measure of x based on the measure of all enumerable y lexicographically greater than x. It is more dominant yet just as limit-computable as Solomonoff's [52]. That is, if we are interested in limit-computable universal measures, we should prefer the novel universal cumulatively enumerable measure over the traditional enumerable one. If we include in our Bayesmix such limit-computable distributions we obtain again sharp loss bounds for prediction based on the mix [49,52].

Our approach highlights differences between countable and uncountable sets. Which are the potential consequences for physics? We argue that things such as *un*countable time and space and *in*computable probabilities actually should not play a role in explaining the world, for lack of evidence that they are really necessary [49]. Some may feel tempted to counter this line of reasoning by pointing out that for centuries physicists have calculated with continua of real numbers, most of them incomputable. Even quantum physicists who are ready to give up the assumption of a continuous universe usually do take for granted the existence of continuous probability distributions on their discrete universes, and Stephen Hawking explicitly said: *"Although there have been suggestions that space-time may have a discrete structure I see no reason to abandon the continuum theories that have been so successful."* Note, however, that all physicists in fact have only manipulated discrete symbols, thus generating finite, describable proofs of their results derived from enumerable axioms. That real numbers really *exist* in a way transcending the finite symbol strings used by everybody may be a figment of imagination — compare Brouwer's constructive mathematics [5,1] and the Löwenheim-Skolem Theorem [31,58] which implies that any first order theory with an uncountable model such as the real numbers also has a countable model. As Kronecker put it: *"Die ganze Zahl schuf der liebe Gott, alles Übrige ist Menschenwerk"* ("God created the integers, all else is the work of man" [6]). Kronecker greeted with scepticism Cantor's celebrated insight [7] about real numbers, mathematical objects Kronecker believed did not even exist.

Assuming our future lies among the few (countably many) describable futures, we can ignore uncountably many nondescribable ones, in particular, the random ones. Adding the relatively mild assumption that the probability distribution from which our universe is drawn is cumulatively enumerable provides a theoretical justification of the prediction that the most likely continuations of our universes are computable through short enumeration procedures. In this sense Occam's razor is just a natural by-product of a computability assumption! But what about falsifiability? The pseudorandomness of our universe might be effectively undetectable in principle, because some approximable and enumerable patterns cannot be proven to be nonrandom in recursively bounded time.

The next sections, however, will introduce additional plausible assumptions that do lead to *computable* optimal prediction procedures.

5 Computable Predictions through the Speed Prior Based on the Fastest Way of Describing Objects

Unfortunately, while M and the more general priors of Section 4 are computable in the limit, they are not recursive, and thus practically infeasible. This drawback inspired less general yet practically more feasible principles of minimum description length (MDL) [68,40] as well as priors derived from time-bounded restrictions [30] of Kolmogorov complexity [27,59,8]. No particular instance of these approaches, however, is universally accepted or has a general convincing motivation that carries beyond rather specialized application scenarios. For instance, typical efficient MDL approaches require the specification of a class of computable models of the data, say, certain types of neural networks, plus some computable loss function expressing the coding costs of the data relative to the model. This provokes numerous *ad-hoc* choices.

Our recent work [54], however, offers an alternative to the celebrated but noncomputable algorithmic simplicity measure or Solomonoff-Levin measure discussed above [59,75,60]. We introduced a new measure (a prior on the computable objects) which is not based on the **shortest** but on the **fastest** way of describing objects.

Let us assume that the observed data sequence is generated by a computational process, and that any possible sequence of observations is therefore computable in the limit [49]. This assumption is stronger and more radical than the traditional one: Solomonoff just insists that the probability of any sequence prefix is recursively computable, but the (infinite) sequence itself may still be generated probabilistically.

Given our starting assumption that data are deterministically generated by a machine, it seems plausible that the machine suffers from a computational resource problem. Since some things are much harder to compute than others, the resource-oriented point of view suggests the following postulate.

Postulate 1 *The cumulative prior probability measure of all x incomputable within time t by any method is at most inversely proportional to t.*

This postulate leads to the Speed Prior $S(x)$, the probability that the output of the following probabilistic algorithm starts with x [54]:

1. Toss an unbiased coin until heads is up; let i denote the number of required trials; set $t := 2^i$.
2. If the number of steps executed so far exceeds t then exit. Execute one step; if this leads to a request for a new input bit (of the growing self-delimiting program, e.g., [29, 30]), toss the coin to determine the bit, and set $t := t/2$.
3. Go to **2.**

Algorithm **GUESS** is very similar to a probabilistic search algorithm used in previous work on applied inductive inference [46, 48]. On several toy problems it generalized extremely well in a way unmatchable by traditional neural network learning algorithms.

With S comes a computable method **AS** for predicting optimally within ϵ accuracy [54]. Consider a finite but unknown program p computing $y \in B^\infty$. What if Postulate 1 holds but p is not optimally efficient, and/or computed on a computer that differs from our reference machine? Then we effectively do not sample beginnings y_k from S but from an alternative semimeasure S'. Can we still predict well? Yes, because the Speed Prior S dominates S'. This dominance is all we need to apply the recent loss bounds [21]. The loss that we are expected to receive by predicting according to **AS** instead of using the true but unknown S' does not exceed the optimal loss by much [54].

6 Speed Prior-Based Predictions for Our Universe

"In the beginning was the code."
FIRST SENTENCE OF THE GREAT PROGRAMMER'S BIBLE

Physicists and other inductive scientists make predictions based on observations. Astonishingly, however, few physicists are aware of the theory of *optimal* inductive inference [59, 27]. In fact, when talking about the very nature of their inductive business, many physicists cite rather vague concepts such as Popper's falsifiability [38], instead of referring to quantitative results.

All widely accepted physical theories, however, are accepted not because they are falsifiable—they are not—or because they match the data—many alternative theories also match the data—but because they are simple in a certain sense. For example, the theory of gravitation is induced from locally observable training examples such as falling apples and movements of distant light sources, presumably stars. The theory predicts that apples on distant planets in other galaxies will fall as well. Currently nobody is able to verify or falsify this. But everybody believes in it because this generalization step makes the theory simpler than alternative theories with separate laws for apples on other planets. The same holds for superstring theory [18] or Everett's many world theory [13], which presently also are neither verifiable nor falsifiable, yet offer comparatively simple explanations of numerous observations. In particular, most of Everett's postulated many worlds will remain unobservable forever, but the assumption of their existence simplifies the theory, thus making it more beautiful and acceptable.

In Sections 3 and 4 we have made the assumption that the probabilities of next events, given previous events, are (limit-)computable. Here we make a stronger assumption by adopting **Zuse's thesis** [73, 74], namely, that the very universe is actually being computed deterministically, e.g., on a cellular automaton (CA) [65, 67]. Quantum physics, quantum computation [3, 11, 37], Heisenberg's uncertainty principle and Bell's inequality [2] do **not** imply any physical evidence against this possibility, e.g., [63].

But then which is our universe's precise algorithm? The following method [47] does compute it:

> Systematically create and execute all programs for a universal computer, such as a Turing machine or a CA; the first program is run for one instruction every second step on average, the next for one instruction every second of the remaining steps on average, and so on.

This method in a certain sense implements the simplest theory of everything: *all* computable universes, including ours and ourselves as observers, are computed by the very short program that generates and executes *all* possible programs [47]. In nested fashion, some of these programs will execute processes that again compute all possible universes, etc. [47]. Of course, observers in "higher-level" universes may be completely unaware of observers or universes computed by nested processes, and vice versa. For example, it seems hard to track and interpret the computations performed by a cup of tea.

The simple method above is more efficient than it may seem at first glance. A bit of thought shows that it even has the optimal order of complexity. For example, it outputs our universe history as quickly as this history's fastest program, save for a (possibly huge) constant slowdown factor that does not depend on output size.

Nevertheless, some universes are fundamentally harder to compute than others. This is reflected by the Speed Prior S discussed above (Section 5). So let us assume that our universe's history is sampled from S or a less dominant prior reflecting suboptimal computation of the history. Now we can immediately predict:

1. Our universe will not get many times older than it is now [49] — essentially, the probability that it will last 2^n times longer than it has lasted so far is at most 2^{-n}.

2. Any apparent randomness in any physical observation must be due to some yet unknown but *fast* pseudo-random generator PRG [49] which we should try to discover. **2a.** A re-examination of beta decay patterns may reveal that a very simple, fast, but maybe not quite trivial PRG is responsible for the apparently random decays of neutrons into protons, electrons and antineutrinos. **2b.** Whenever there are several possible continuations of our universe corresponding to different Schrödinger wave function collapses — compare Everett's widely accepted many worlds hypothesis [13] — we should be more likely to end up in one computable by a short *and* fast algorithm. A re-examination of split experiment data involving entangled states such as the observations of spins of initially close but soon distant particles with correlated spins might reveal unexpected, nonobvious, nonlocal algorithmic regularity due to a fast PRG.

3. Large scale quantum computation [3] will not work well, essentially because it would require too many exponentially growing computational resources in interfering "parallel universes" [13].

4. Any probabilistic algorithm depending on truly random inputs from the environment will not scale well in practice.

Prediction **2** is verifiable but not necessarily falsifiable within a fixed time interval given in advance. Still, perhaps the main reason for the current absence of empirical evidence in this vein is that few [12] have looked for it.

In recent decades several well-known physicists have started writing about topics of computer science, e.g., [37, 11], sometimes suggesting that real world physics might allow for computing things that are not computable traditionally. Unimpressed by this trend, computer scientists have argued in favor of the opposite: since there is no evidence that we need more than traditional computability to explain the world, we should try to make do without this assumption, e.g., [73, 74, 14, 47].

7 Optimal Rational Decision Makers

So far we have talked about passive prediction, given the observations. Note, however, that agents interacting with an environment can also use predictions of the future to compute action sequences that maximize expected future reward. Hutter's recent *AIXI model* [22] (author's SNF grant 61847) does exactly this, by combining Solomonoff's M-based universal prediction scheme with an *expectimax* computation.

In cycle t action y_t results in perception x_t and reward r_t, where all quantities may depend on the complete history. The perception x'_t and reward r_t are sampled from the (reactive) environmental probability distribution μ. Sequential decision theory shows how to maximize the total expected reward, called value, if μ is known. Reinforcement learning [26] is used if μ is unknown. AIXI defines a mixture distribution ξ as a weighted sum of distributions $\nu \in M$, where M is any class of distributions including the true environment μ.

It can be shown that the conditional M probability of environmental inputs to an AIXI agent, given the agent's earlier inputs and actions, converges with increasing length of interaction against the true, unknown probability [22], as long as the latter is recursively computable, analogously to the passive prediction case.

Recent work [24] also demonstrated AIXI's optimality in the following sense. The Bayes-optimal policy p^ξ based on the mixture ξ is self-optimizing in the sense that the average value converges asymptotically for all $\mu \in \mathcal{M}$ to the optimal value achieved by the (infeasible) Bayes-optimal policy p^μ which knows μ in advance. The necessary condition that $\mathcal{M}$ admits self-optimizing policies is also sufficient. No other structural assumptions are made on $\mathcal{M}$. Furthermore, p^ξ is Pareto-optimal in the sense that there is no other policy yielding higher or equal value in *all* environments $\nu \in \mathcal{M}$ and a strictly higher value in at least one [24].

We can modify the AIXI model such that its predictions are based on the ϵ-approximable Speed Prior S instead of the incomputable $\mathcal{M}$. Thus we obtain the so-called *AIS model.* Using Hutter's approach [22] we can now show that the conditional S probability of environmental inputs to an AIS agent, given the earlier inputs and actions, converges against the true but unknown probability, as long as the latter is dominated by S, such as the S' above.

8 Optimal Universal Search Algorithms

In a sense, searching is less general than reinforcement learning because it does not necessarily involve predictions of unseen data. Still, search is a central aspect of computer science (and any reinforcement learner needs a searcher as a submodule—see Section 10). Surprisingly, however, many books on search algorithms do not even mention the following, very simple asymptotically optimal, "universal" algorithm for a broad class of search problems.

Define a probability distribution P on a finite or infinite set of programs for a given computer. P represents the searcher's initial bias (e.g., P could be based on program length, or on a probabilistic syntax diagram).

```
Method LSEARCH: Set current time limit T=1. WHILE problem not
solved DO:
    Test all programs $q$ such that $t(q)$, the maximal time spent on
    creating and running and testing $q$, satisfies $t(q) < P(q)\ T$. Set
    $T := 2T$.
```

LSEARCH (for *Levin Search*) may be the algorithm Levin was referring to in his 2 page paper [28] which states that there is an asymptotically optimal universal search method for problems with easily verifiable solutions, that is, solutions whose validity can be quickly tested. Given some problem class, if some unknown optimal program p requires $f(k)$ steps to solve a problem instance of size k, then LSEARCH will need at most $O(P(p)f(k)) = O(f(k))$ steps — the constant factor $P(p)$ may be huge but does not depend on k. Compare [30, p. 502-505] and [23] and the fastest way of computing all computable universes in Section 6.

Recently Hutter developed a more complex asymptotically optimal search algorithm for *all* well-defined problems, not just those with with easily verifiable solutions [23]. HSEARCH cleverly allocates part of the total search time for searching the space of proofs to find provably correct candidate programs with provable upper runtime bounds, and at any given time focuses resources on those programs with the currently best proven time bounds. Unexpectedly, HSEARCH manages to reduce the unknown constant slowdown factor of LSEARCH to a value of $1 + \epsilon$, where ϵ is an arbitrary positive constant.

Unfortunately, however, the search in proof space introduces an unknown *additive* problem class-specific constant slowdown, which again may be huge. While additive constants generally are preferrable over multiplicative ones, both types may make universal search methods practically infeasible.

HSEARCH and LSEARCH are nonincremental in the sense that they do not attempt to minimize their constants by exploiting experience collected in previous searches. Our method *Adaptive* LSEARCH or ALS tries to overcome this [57] — compare Solomonoff's related ideas [61, 62]. Essentially it works as follows: whenever LSEARCH finds a program q that computes a solution for the current problem, q's probability $P(q)$ is substantially increased using a "learning rate," while probabilities of alternative programs decrease appropriately. Subsequent LSEARCHes for new problems then use the adjusted P, etc. A nonuniversal variant of this approach was able to solve reinforcement learning (RL) tasks [26] in partially observable environments unsolvable by traditional RL algorithms [71, 57].

Each LSEARCH invoked by ALS is optimal with respect to the most recent adjustment of P. On the other hand, the modifications of P themselves are not necessarily optimal. Recent work discussed in the next section overcomes this drawback in a principled way.

9 Optimal Ordered Problem Solver (OOPS)

Our recent OOPS [53, 51] is a simple, general, theoretically sound, time-optimal way of searching for a universal behavior or program that solves each problem in a sequence of computational problems, continually organizing and managing and reusing earlier acquired knowledge. For example, the n-th problem may be to compute the n-th event from previous events (prediction), or to find a faster way through a maze than the one found during the search for a solution to the $n-1$-th problem (optimization).

Let us first introduce the important concept of bias-optimality, which is a pragmatic definition of time-optimality, as opposed to the asymptotic optimality of both LSEARCH and HSEARCH, which may be viewed as academic exercises demonstrating that the $O()$ notation can sometimes be practically irrelevant despite its wide use in theoretical computer science. Unlike asymptotic optimality, bias-optimality does not ignore huge constant slowdowns:

Definition 1 (BIAS-OPTIMAL SEARCHERS). *Given is a problem class $\mathcal{R}$, a search space $\mathcal{C}$ of solution candidates (where any problem $r \in \mathcal{R}$ should have a solution in $\mathcal{C}$), a task dependent bias in form of conditional probability distributions $P(q \mid r)$ on the candidates $q \in \mathcal{C}$, and a predefined procedure that creates and tests any given q on any $r \in \mathcal{R}$ within time $t(q, r)$ (typically unknown in advance). A searcher is n-*bias-optimal *($n \geq 1$) if for any maximal total search time $T_{max} > 0$ it is guaranteed to solve any problem $r \in \mathcal{R}$ if it has a solution $p \in \mathcal{C}$ satisfying $t(p, r) \leq P(p \mid r)\ T_{max}/n$. It is* bias-optimal *if $n = 1$.*

This definition makes intuitive sense: the most probable candidates should get the lion's share of the total search time, in a way that precisely reflects the initial bias. Now we are ready to provide a general overview of the basic ingredients of OOPS [53, 51]:

Primitives. We start with an initial set of user-defined primitive behaviors. Primitives may be assembler-like instructions or time-consuming software, such as, say, theorem provers, or matrix operators for neural network-like parallel architectures, or trajectory generators for robot simulations, or state update procedures for multiagent systems, etc. Each primitive is represented by a token. It is essential that those primitives whose runtimes are not known in advance can be interrupted at any time.

Task-specific prefix codes. Complex behaviors are represented by token sequences or programs. To solve a given task represented by task-specific program inputs, OOPS tries to sequentially compose an appropriate complex behavior from primitive ones, always obeying the rules of a given user-defined initial programming language. Programs are grown incrementally, token by token; their beginnings or *prefixes* are immediately executed while being created; this may modify some task-specific internal state or memory, and may transfer control back to previously selected tokens (e.g., loops). To add a new token to some program prefix, we first have to wait until the execution of the prefix so far *explicitly requests* such a prolongation, by setting an appropriate signal in the internal state. Prefixes that cease to request any further tokens are called *self-delimiting* programs or simply programs (programs are their own prefixes). *Binary* self-delimiting programs were studied by [29] and [9] in the context of Turing machines [64] and the theory of Kolmogorov complexity and algorithmic probability [59, 27]. OOPS, however, uses a more practical, not necessarily binary framework.

The program construction procedure above yields *task-specific prefix codes* on program space: with any given task, programs that halt because they have found a solution or encountered some error cannot request any more tokens. Given the current task-specific inputs, no program can be the prefix of another one. On a different task, however, the same program may continue to request additional tokens. This is important for our novel approach—incrementally growing self-delimiting programs are unnecessary for the asymptotic optimality properties of LSEARCH and HSEARCH, but essential for OOPS.

Access to previous solutions. Let p^n denote a found prefix solving the first n tasks. The search for p^{n+1} may greatly profit from the information conveyed by (or the knowledge embodied by) $p^1, p^2, \ldots, p^n$ which are stored or *frozen* in special *non*modifiable memory shared by all tasks, such that they are accessible to p^{n+1} (this is another difference to *non*incremental LSEARCH and HSEARCH). For example, p^{n+1} might execute a token sequence that calls p^{n-3} as a subprogram, or that copies p^{n-17} into some internal *modifiable* task-specific memory, then modifies the copy a bit, then applies the slightly edited copy to the current task. In fact, since the number of frozen programs may grow to a large value, much of the knowledge embodied by p^j may be about how to access and edit and use older p^i $(i < j)$.

Bias. The searcher's initial bias is embodied by initial, user-defined, task dependent probability distributions on the finite or infinite search space of possible program prefixes. In the simplest case we start with a maximum entropy distribution on the tokens, and define prefix probabilities as the products of the probabilities of their tokens. But prefix continuation probabilities may also depend on previous tokens in context sensitive fashion.

Self-computed suffix probabilities. In fact, we permit that any executed prefix assigns a task-dependent, self-computed probability distribution to its own possible continuations. This distribution is encoded and manipulated in task-specific internal memory. So unlike with ALS [57] we do not use a prewired learning scheme to update the probability distribution. Instead we leave such updates to prefixes whose online execution modifies the probabilities of their suffixes. By, say, invoking previously frozen code that redefines the probability distribution on future prefix continuations, the currently tested prefix may completely reshape the most likely paths through the search space of its own continuations, based on experience ignored by *non*incremental LSEARCH and HSEARCH. This may introduce significant problem class-specific knowledge derived from solutions to earlier tasks.

Two searches. Essentially, OOPS provides equal resources for two near-*bias-optimal* searches (Def. 1) that run in parallel until p^{n+1} is discovered and stored in non-modifiable memory. The first is exhaustive; it systematically tests all possible prefixes on all tasks up to $n + 1$. Alternative prefixes are tested on all current tasks in parallel while still growing; once a task is solved, we remove it from the current set; prefixes that fail on a single task are discarded. The second search is much more focused; it only searches for prefixes that start with p^n, and only tests them on task $n+1$, which is safe, because we already know that such prefixes solve all tasks up to n.

Bias-optimal backtracking. HSEARCH and LSEARCH assume potentially infinite storage. Hence they may largely ignore questions of storage management. In any practical system, however, we have to efficiently reuse limited storage. Therefore, in both searches of OOPS, alternative prefix continuations are evaluated by a novel, practical, token-oriented backtracking procedure that can deal with several tasks in parallel, given some *code bias* in the form of previously found code. The procedure always ensures near-*bias-optimality* (Def. 1): no candidate behavior gets more time than it deserves, given the probabilistic bias. Essentially we conduct a depth-first search in program space, where the branches of the search tree are program prefixes, and backtracking (partial resets of partially solved task sets and modifications of internal states and continuation probabilities) is triggered once the sum of the runtimes of the current prefix on all current tasks exceeds the prefix probability multiplied by the total search time so far.

In case of unknown, infinite task sequences we can typically never know whether we already have found an optimal solver for all tasks in the sequence. But once we unwittingly do find one, at most half of the total future run time will be wasted on searching for alternatives. Given the initial bias and subsequent bias shifts due to $p^1, p^2, \ldots$, no other bias-optimal searcher can expect to solve the $n+1$-th task set substantially faster than OOPS. A by-product of this optimality property is that it gives us a natural and precise measure of bias and bias shifts, conceptually related to Solomonoff's *conceptual jump size* of [61, 62].

Since there is no fundamental difference between domain-specific problem-solving programs and programs that manipulate probability distributions and thus essentially rewrite the search procedure itself, we collapse both learning and metalearning in the same time-optimal framework.

An example initial language. For an illustrative application, we wrote an interpreter for a stack-based universal programming language inspired by FORTH [34], with initial primitives for defining and calling recursive functions, iterative loops, arithmetic operations, and domain-specific behavior. Optimal metasearching for better search algorithms is enabled through the inclusion of bias-shifting instructions that can modify the conditional probabilities of future search options in currently running program prefixes.

Experiments. Using the assembler-like language mentioned above, we first teach OOPS something about recursion, by training it to construct samples of the simple context free language $\{1^k 2^k\}$ (k 1's followed by k 2's), for k up to 30 (in fact, the system discovers a universal solver for all k). This takes roughly 0.3 days on a standard personal computer (PC). Thereafter, within a few additional days, OOPS demonstrates incremental knowledge transfer: it exploits aspects of its previously discovered universal $1^k 2^k$-solver, by rewriting its search procedure such that it more readily discovers a universal solver for all k disk *Towers of Hanoi* problems—in the experiments it solves all instances up to $k = 30$ (solution size $2^k - 1$), but it would also work for $k > 30$. Previous, less general reinforcement learners and *non*learning AI planners tend to fail for much smaller instances.

Future research may focus on devising particularly compact, particularly reasonable sets of initial codes with particularly broad practical applicability. It may turn out that the most useful initial languages are not traditional programming languages similar to the FORTH-like one, but instead based on a handful of primitive instructions for massively parallel cellular automata [65, 67, 74, 72], or on a few nonlinear operations on matrix-like data structures such as those used in recurrent neural network research [69, 43, 4]. For example, we could use the principles of OOPS to create a non-gradient-based, near-bias-optimal variant of Hochreiter's successful recurrent network metalearner [19]. It should also be of interest to study probabilistic *Speed Prior*-based OOPS variants [54] and to devise applications of OOPS-like methods as components of universal reinforcement learners (see below). In ongoing work, we are applying OOPS to the problem of optimal trajectory planning for robotics in a realistic physics simulation. This involves the interesting trade-off between comparatively fast program-composing primitives or *"thinking primitives"* and time-consuming *"action primitives"*, such as *stretch-arm-until-touch-sensor-input*.

10 OOPS-Based Reinforcement Learning

At any given time, a reinforcement learner [26] will try to find a *policy* (a strategy for future decision making) that maximizes its expected future reward. In many traditional reinforcement learning (RL) applications, the policy that works best in a given set of training trials will also be optimal in future test trials [50]. Sometimes, however, it won't. To see the difference between searching (the topic of the previous sections) and reinforcement learning (RL), consider an agent and two boxes. In the n-th trial the agent may open and collect the content of exactly one box. The left box will contain $100n$ Swiss Francs, the right box 2^n Swiss Francs, but the agent does not know this in advance. During the first 9 trials the optimal policy is *"open left box."* This is what a good searcher should find, given the outcomes of the first 9 trials. But this policy will be suboptimal in trial 10. A good reinforcement learner, however, should extract the underlying regularity in the reward generation process and predict the future reward, picking the right box in trial 10, without having seen it yet.

The first general, asymptotically optimal reinforcement learner is the recent AIXI model [22, 24] (Section 7). It is valid for a very broad class of environments whose reactions to action sequences (control signals) are sampled from arbitrary computable probability distributions. This means that AIXI is far more general than traditional RL approaches. However, while AIXI clarifies the theoretical limits of RL, it is not practically feasible, just like HSEARCH is not. ¿From a pragmatic point of view, what we are really interested in is a reinforcement learner that makes optimal use of given, limited computational resources. In what follows, we will outline how to use OOPS-like bias-optimal methods as components of universal yet feasible reinforcement learners.

We need two OOPS modules. The first is called the predictor or world model. The second is an action searcher using the world model. The life of the entire system should consist of a sequence of *cycles* 1, 2, ... At each cycle, a limited amount of computation time will be available to each module. For simplicity we assume that during each cyle the system may take exactly one action. Generalizations to actions consuming several cycles are straight-forward though. At any given cycle, the system executes the following procedure:

1. For a time interval fixed in advance, the predictor is first trained in bias-optimal fashion to find a better world model, that is, a program that predicts the inputs from the environment (including the rewards, if there are any), given a history of previous observations and actions. So the n-th task ($n = 1, 2, \ldots$) of the first OOPS module is to find (if possible) a better predictor than the best found so far.
2. Once the current cycle's time for predictor improvement is used up, the current world model (prediction program) found by the first OOPS module will be used by the second module, again in bias-optimal fashion, to search for a future action sequence that maximizes the predicted cumulative reward (up to some time limit). That is, the n-th task ($n = 1, 2, \ldots$) of the second OOPS module will be to find a control program that computes a control sequence of actions, to be fed into the program representing the current world model (whose input predictions are successively fed back to itself in the obvious manner), such that this control sequence leads to higher predicted reward than the one generated by the best control program found so far.
3. Once the current cycle's time for control program search is used up, we will execute the current action of the best control program found in step 2. Now we are ready for the next cycle.

The approach is reminiscent of an earlier, heuristic, non-bias-optimal RL approach based on two adaptive recurrent neural networks, one representing the world model, the other one a controller that uses the world model to extract a policy for maximizing expected reward [45]. The method was inspired by previous combinations of *non*recurrent, *reactive* world models and controllers [70, 36, 25].

At any given time, until which temporal horizon should the predictor try to predict? In the AIXI case, the proper way of treating the temporal horizon is not to discount it exponentially, as done in most traditional work on reinforcement learning, but to let the future horizon grow in proportion to the learner's lifetime so far [24]. It remains to be seen whether this insight carries over to OOPS-based RL. In particular, is it possible to prove that certain OOPS-RL variants are a near-bias-optimal way of spending a given amount of computation time on RL problems? Or should we instead combine OOPS and Hutter's time-bounded AIXI(t, l) model?

We observe that the grand problem of AI (as defined in the abstract) is not yet solved, but promising approaches along the lines above suggest themselves.

11 Conclusion

Recent theoretical and practical advances are currently driving a renaissance in the fields of universal learners and optimal search [56]. A new kind of AI is emerging. Does it really deserve the attribute *"new,"* given that its roots date back to the 1960s, just two decades after Zuse built the first general purpose computer in 1941? An affirmative answer seems justified, since it is the recent results on practically feasible computable variants of the old incomputable methods that are currently reinvigorating the long dormant field. The "new" AI is new in the sense that it abandons the mostly heuristic or non-general approaches of the past decades, offering methods that are both general and theoretically sound, and provably optimal in a sense that *does* make sense in the real world.

We are led to claim that the future will belong to universal or near-universal learners that are more general than traditional reinforcement learners / decision makers depending on strong Markovian assumptions, or than learners based on traditional statistical learning theory, which often require unrealistic i.i.d. or Gaussian assumptions. Due to ongoing hardware advances the time has come for optimal search in algorithm space, as opposed to the limited space of reactive mappings embodied by traditional methods such as artificial feedforward neural networks.

It seems safe to bet that not only computer scientists but also physicists and other inductive scientists will start to pay more attention to the fields of universal induction and optimal search, since their basic concepts are irresistibly powerful and general and simple. How long will it take for these ideas to unfold their full impact? A very naive and speculative guess driven by wishful thinking might be based on identifying the *"greatest moments in computing history"* and extrapolating from there. Which are those "greatest moments"? Obvious candidates are:

1. *1640:* first mechanical calculator (Pascal, France).
2. *Two centuries later:* concept of a programmable computer (Babbage, UK).
3. *One century later:* first working programmable computer (Zuse, Berlin), plus fundamental theoretical work on universal integer-based programming languages and the limits of proof and computation (Gödel, Austria, reformulated by Turing, UK). (The next 50 years saw many theoretical advances as well as faster and faster switches—relays were replaced by tubes by transistors by chips—but arguably this was rather predictable, incremental progress without radical shake-up events.)
4. *Half a century later:* the World Wide Web (UK's Berners-Lee, Switzerland).

This list seems to suggest that each major breakthrough tends to come twice as fast as the previous one. Extrapolating the trend, optimists should expect another radical change by 2015, which happens to coincide with the date when the fastest computers will match brains in terms of raw computing power, according to frequent estimates based on Moore's law. The author is confident that the coming 2015 upheaval (if any) will involve universal learning algorithms and optimal incremental search in algorithm space—possibly laying a foundation for the remaining series of faster and faster additional revolutions culminating in an "Omega point" expected around 2040.

12 Acknowledgments

Hutter's frequently mentioned work was funded through the author's SNF grant 2000-061847 "Unification of universal inductive inference and sequential decision theory." Over the past three decades, numerous discussions with Christof Schmidhuber (a theoretical physicist) helped to crystallize the ideas on computable universes—compare his notion of *"mathscape"* [44].

References

1. M. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, Heidelberg, 1985.
2. J. S. Bell. On the problem of hidden variables in quantum mechanics. *Rev. Mod. Phys.*, 38:447–452, 1966.
3. C. H. Bennett and D. P. DiVicenzo. Quantum information and computation. *Nature*, 404(6775):256–259, 2000.
4. C. M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, 1995.
5. L. E. J. Brouwer. Over de Grondslagen der Wiskunde. Dissertation, Doctoral Thesis, University of Amsterdam, 1907.
6. F. Cajori. *History of mathematics (2nd edition).* Macmillan, New York, 1919.
7. G. Cantor. Über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen. *Crelle's Journal für Mathematik*, 77:258–263, 1874.
8. G.J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.
9. G.J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.
10. G.J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, Cambridge, 1987.
11. D. Deutsch. *The Fabric of Reality.* Allen Lane, New York, NY, 1997.
12. T. Erber and S. Putterman. Randomness in quantum mechanics – nature's ultimate cryptogram? *Nature*, 318(7):41–43, 1985.
13. H. Everett III. 'Relative State' formulation of quantum mechanics. *Reviews of Modern Physics*, 29:454–462, 1957.
14. E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.
15. R. V. Freyvald. Functions and functionals computable in the limit. *Transactions of Latvijas Vlasts Univ. Zinatn. Raksti*, 210:6–19, 1977.
16. P. Gács. On the relation between descriptional complexity and algorithmic probability. *Theoretical Computer Science*, 22:71–93, 1983.
17. E. M. Gold. Limiting recursion. *Journal of Symbolic Logic*, 30(1):28–46, 1965.
18. M.B. Green, J.H. Schwarz, and E. Witten. *Superstring Theory.* Cambridge University Press, 1987.

19. S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *Lecture Notes on Comp. Sci. 2130, Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2001)*, pages 87–94. Springer: Berlin, Heidelberg, 2001.
20. M. Hutter. Convergence and error bounds of universal prediction for general alphabet. *Proceedings of the 12th European Conference on Machine Learning (ECML-2001)*, (TR IDSIA-07-01, cs.AI/0103015), 2001.
21. M. Hutter. General loss bounds for universal sequence prediction. In C. E. Brodley and A. P. Danyluk, editors, *Proceedings of the* 18th *International Conference on Machine Learning (ICML-2001)*, pages 210–217. Morgan Kaufmann, 2001. TR IDSIA-03-01, IDSIA, Switzerland, Jan 2001, cs.AI/0101019.
22. M. Hutter. Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decisions. *Proceedings of the 12th European Conference on Machine Learning (ECML-2001)*, (TR IDSIA-14-00, cs.AI/0012011):226–238, 2001.
23. M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(3):431–443, 2002.
24. M. Hutter. Self-optimizing and Pareto-optimal policies in general environments based on Bayes-mixtures. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 364–379, Sydney, Australia, 2002. Springer.
25. M. I. Jordan and D. E. Rumelhart. Supervised learning with a distal teacher. Technical Report Occasional Paper #40, Center for Cog. Sci., Massachusetts Institute of Technology, 1990.
26. L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: a survey. *Journal of AI research*, 4:237–285, 1996.
27. A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.
28. L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
29. L. A. Levin. Laws of information (nongrowth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10(3):206–210, 1974.
30. M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition)*. Springer, 1997.
31. L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915.
32. N. Merhav and M. Feder. Universal prediction. *IEEE Transactions on Information Theory*, 44(6):2124–2147, 1998.
33. T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
34. C. H. Moore and G. C. Leach. FORTH - a language for interactive computing, 1970. http://www.ultratechnology.com.
35. A. Newell and H. Simon. GPS, a program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963.
36. Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press, 1989.
37. R. Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.
38. K. R. Popper. *The Logic of Scientific Discovery*. Hutchinson, London, 1934.
39. H. Putnam. Trial and error predicates and the solution to a problem of Mostowski. *Journal of Symbolic Logic*, 30(1):49–57, 1965.

40. J. Rissanen. Stochastic complexity and modeling. *The Annals of Statistics*, 14(3):1080–1100, 1986.
41. H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967.
42. P. S. Rosenbloom, J. E. Laird, and A. Newell. *The SOAR Papers.* MIT Press, 1993.
43. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
44. C. Schmidhuber. Strings from logic. Technical Report CERN-TH/2000-316, CERN, Theory Division, 2000. http://xxx.lanl.gov/abs/hep-th/0011065.
45. J. Schmidhuber. Reinforcement learning in Markovian and non-Markovian environments. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 500–506. Morgan Kaufmann, 1991.
46. J. Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A. Prieditis and S. Russell, editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA, 1995.
47. J. Schmidhuber. A computer scientist's view of life, the universe, and everything. In C. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential - Theory - Cognition*, volume 1337, pages 201–208. Lecture Notes in Computer Science, Springer, Berlin, 1997. Submitted 1996.
48. J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.
49. J. Schmidhuber. Algorithmic theories of everything. Technical Report IDSIA-20-00, quant-ph/0011122, IDSIA, Manno (Lugano), Switzerland, 2000.
50. J. Schmidhuber. Sequential decision making based on direct search. In R. Sun and C. L. Giles, editors, *Sequence Learning: Paradigms, Algorithms, and Applications.* Springer, 2001. Lecture Notes on AI 1828.
51. J. Schmidhuber. Bias-optimal incremental problem solving. In *Advances in Neural Information Processing Systems 15.* MIT Press, Cambridge, MA, 2002. To appear.
52. J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 13(4):587–612, 2002.
53. J. Schmidhuber. Optimal ordered problem solver. Technical Report IDSIA-12-02, arXiv:cs.AI/0207097 v1, IDSIA, Manno-Lugano, Switzerland, July 2002.
54. J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, Sydney, Australia, 2002.
55. J. Schmidhuber. The new AI: General & sound & relevant for physics. In B. Goertzel and C. Pennachin, editors, *Real AI: New Approaches to Artificial General Intelligence.* Plenum Press, New York, 2003. To appear. Also available as TR IDSIA-04-03, cs.AI/0302012.
56. J. Schmidhuber and M. Hutter. NIPS 2002 workshop on universal learning algorithms and optimal search. Additional speakers: R. Solomonoff, P. M. B. Vitányi, N. Cesa-Bianchi, I. Nemenmann. Whistler, CA, 2002.
57. J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.

58. T. Skolem. Logisch-kombinatorische Untersuchungen über Erfüllbarkeit oder Beweisbarkeit mathematischer Sätze nebst einem Theorem über dichte Mengen. *Skrifter utgit av Videnskapsselskapet in Kristiania, I, Mat.-Nat. Kl.*, N4:1–36, 1919.
59. R.J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.
60. R.J. Solomonoff. Complexity-based induction systems. *IEEE Transactions on Information Theory*, IT-24(5):422–432, 1978.
61. R.J. Solomonoff. An application of algorithmic probability to problems in artificial intelligence. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 473–491. Elsevier Science Publishers, 1986.
62. R.J. Solomonoff. A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527. Tel Aviv, Israel, 1989.
63. G. 't Hooft. Quantum gravity as a dissipative deterministic system. Technical Report SPIN-1999/07/gr-gc/9903084, http://xxx.lanl.gov/abs/gr-qc/9903084, Institute for Theoretical Physics, Univ. of Utrecht, and Spinoza Institute, Netherlands, 1999. Also published in *Classical and Quantum Gravity 16*, 3263.
64. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 41:230–267, 1936.
65. S. Ulam. Random processes and transformations. In *Proceedings of the International Congress on Mathematics*, volume 2, pages 264–275, 1950.
66. V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.
67. J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illionois Press, Champain, IL, 1966.
68. C. S. Wallace and D. M. Boulton. An information theoretic measure for classification. *Computer Journal*, 11(2):185–194, 1968.
69. P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.
70. P. J. Werbos. Learning how the world works: Specifications for predictive networks in robots and brains. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics, N.Y.*, 1987.
71. M.A. Wiering and J. Schmidhuber. Solving POMDPs with Levin search and EIRA. In L. Saitta, editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 534–542. Morgan Kaufmann Publishers, San Francisco, CA, 1996.
72. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.
73. K. Zuse. Rechnender Raum. *Elektronische Datenverarbeitung*, 8:336–344, 1967.
74. K. Zuse. *Rechnender Raum.* Friedrich Vieweg & Sohn, Braunschweig, 1969.
75. A. K. Zvonkin and L. A. Levin. The complexity of finite objects and the algorithmic concepts of information and randomness. *Russian Math. Surveys*, 25(6):83–124, 1970.

Part II
Students Short Papers

Damage Identification using Soft-computing Techniques

Luis Eduardo Mujica[1] and Josep Vehí[2]
Departament d'Electrònica, Informàtica i Automàtica (EIA)
Universitat de Girona
Montilivi. Girona - Spain
Phone: +34-972 418 487, Fax: +34-972418976
email:{lemujica,vehi}@eia.udg.es

1 INTRODUCTION

The damage identification problem in structural analysis is usually based on the phenomenon of elastic strain wave propagation. An excitation signal is applied and the dynamic response is examined. Many works analyses the perturbations to the original signal due to structural damage. However the currently used methods encounter problems with obtaining the proper solution to damage identification and the related numerical cost is considerable.

We propose an approach using Case-Based Reasoning (CBR), Self Organizing Maps (SOM) and Wavelet Transform (WT) in order to obtain an initial diagnostic exploiting the data generated by the modeling structure and the data acquired by the sensors once the system has started, creating an incremental database (since a new experience is retained each time a problem has been solved) in order to use in diagnosing future situations by analogy.

2 CASE BASED REASONING

Reasoning based on experience is a powerful procedure frequently used by human beings to solve problems, both in day-to-day life and in situations requiring more expertise. People rely on similar previous experience when they need to solve a problem, reusing solutions without thinking about the situation so much. . In any field, when tackling problem, a professional with many years of experience is generally considered to be more suitable than a recent graduate with brilliant grades. Daily life continually presents opportunities to apply case based reasoning. CBR systems, instead of being exclusively based on general knowledge of the domain of a problem or establishing associations through a set of generalized relations among descriptors of problems and conclusions, use the specific knowledge of previous experiences in concrete situations. To reach that goal, CBR methodology proposes the cycle of the 4 R's (see Figure 1) [2][4].

Retrieve the most similar cases (a new problem is grouped with other similar problems saved in a case-base)

Reuse the solutions proposed in the cases to solve the problem

Revise the proposed solution (if necessary)

Retain the new solution as a part of a new case once it has been confirmed or validated

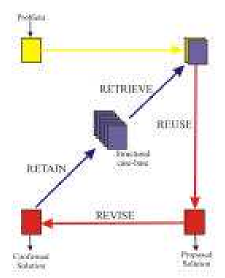


Figure 1. Conventional CBR cycle

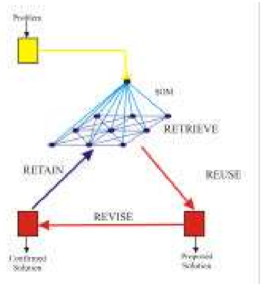


Figure 2. Purposed CBR cycle

3 SELF ORGANIZING MAPS

Self-Organizing Maps (SOM) are the largest representation of artificial neural networks. An SOM is a classifier that can be visualized as a two-dimensional neural network arrangement. The principle used by Kohonen [1] to develop the self-organizing maps is based on the organization of neurons according to the features of the received stimulus. The greatest strength of the self-organizing maps lies in the possibilities they have to model and analyze complex experimental data vectors. The self-organizing maps are non-linear projection methods from a high-dimensional input space to a bi-dimensional space, where it is easier to classify and visualize as vectors. The reduction in the number of dimensions could permit the visualization of important relations between the data that would not be appreciated in any other way.

[1] Research student
[2] Advisor

4 WAVELET TRANSFORM

A wavelet transform is similar to a Fourier transform. The Fourier transform the signal is broken up or decomposed into sine waves of various frequencies. The Wavelet transform is the procedure by which a signal is broken up in a sum of translations (shifting) and dilations (scaling) of a function, called *mother wavelet.* The continuous wavelet transform (CWT) is defined as the sum over all time of the signal multiplied by scaled, shifted versions of the wavelet function ψ:

$$C(scale, position) = \int_{-\infty}^{\infty} f(t)\Psi(scale, position, t)dt$$

The result of the CWT is many wavelet coefficients *C*, which are a function of scale and position (see Figure 3). Multiplying each coefficient by the appropriately scaled and shifted wavelet yields the constituent wavelets of the original signal.

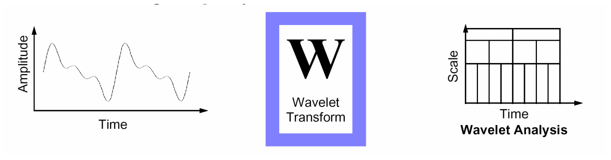


Figure 3. Wavelet Transform

In wavelet-based feature extraction for signal interpretation, the wavelet coefficients are grouped into clusters in an unsupervised mode. The procedure divides the scheme of all computed wavelet coefficients into disjoint clusters U_1, U_2, ..., U_c for each of which a single robust feature u_i $(i = 1, 2, ..., c)$ can be computed. The so obtained feature vector $(u_1, u_2, ..., u_c)$ serves as an input pattern to a signal interpretation procedure such as a neural network [3].

5 ORIGINAL CONTRIBUTION

5.1 HOW IS DAMAGE IDENTIFIED?

We propose using Case-Based Reasoning methodology in damage detection, taking advantage of experience and the model of the structure, exploiting the data acquired by sensors in real practice and the outcomes given in known models simulations. The goal is to use Soft Computing techniques (SOM,WT) to relate the data stored in the memory with representative situations as cases to be used in a later diagnosis by analogy.

Bearing in mind that Case-Based Reasoning is a methodology [5], Figure 2 shows our CBR system, it has a casebase that consists of a Self-Organizing Map. For each new case, the SOM retrieves the group of old cases with same features. These features are extracted using Wavelet Analysis [3].

5.2 OUR APPROACH APPLIED IN A TRUSS STRUCTURE

5.2.1 Description

Figure 4 shows a cantilever truss structure to be considered. Materials and geometric specifications have previously been assigned. The opposite sine excitation to the phase is applied to elements 36 and 38. Member 1 was chosen as the sensor receiving the propagated wave.

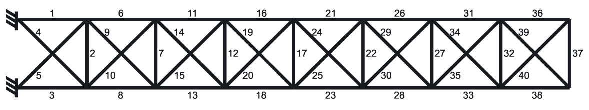


Figure 4. Cantilever Structure

5.2.2 How to build cases?

A case is defined by defect in the structure and the principal features of the elastic wave either modeled or detected by the sensor. For example we have a case with damage in the element 13, the elastic wave is shown in Figure 5 and the principal features in the Figure 6.

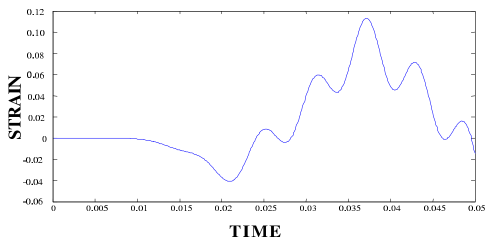


Figure 5. Elastic wave detected

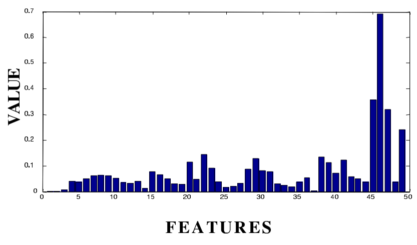


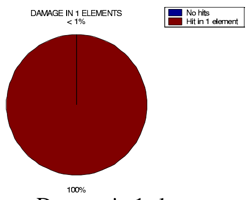
Figure 6. Principal features of the wave

5.2.3 How should the solution be retrieved?

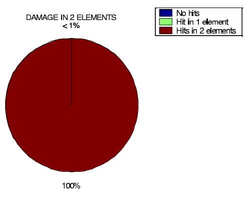
In this way, when a new case is occurred, we don't know the damage, but we have its principal features, The SOM retrieve a set of old cases with most similar features, from this set we propose a solution (its damage). When this solution is validated, it is stored like a new case into the SOM.

5.2.4 Outcomes presentation

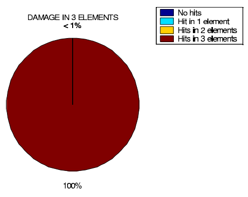
In order to build the casebase, it is necessary to generate damage patterns and to obtain the elastic wave simulated or detected by sensor. Taking into account of the structure in the previous example, we have generated cases of simultaneous damage of 1,2,3,4 and 5 elements into casebase. So as to evaluate the approach, we have generated tests of simultaneous damage of 1,2,3,4,5,6,7,8 and 9 elements. The following figures show the percentage of accurate detections of each test divided by the number of detected defective elements. For example the picture with 6 defective elements Figure 7f, we are detected 3 defectives elements from 6 (hit in 3 elements) in the 14% of the cases, and we are detected 4 elements from 6 (hit in 4 elements) in the 39%.



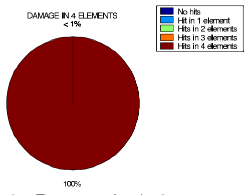
a. Damage in 1 element



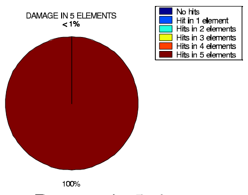
b. Damage in 2 element



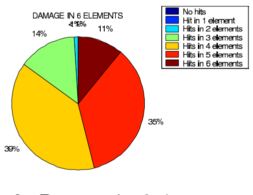
c. Damage in 3 element



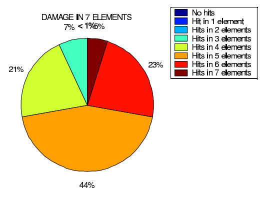
d. Damage in 4 element



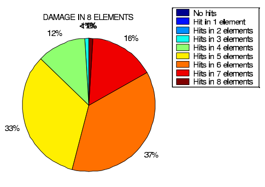
e. Damage in 5 element



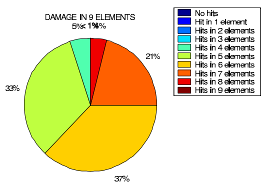
f. Damage in 6 element



g. Damage in 7 element



h. Damage in 8 element



i. Damage in 9 element

Figure 7. Accurate detections in each test

5.2.5 Outcomes analysis

The casebase includes damages up to 5 horizontal elements, therefore up to 5 defective horizontal elements are totally detected (100%). If there are damages too much of 5 elements, it isn't detected completely, however the system adapts the solution and it is able to detect up 8 and 9 defective elements, although still in low percentage.

6 CONCLUSIONS

There are several advantages to the CBR systems approach using SOM. It most closely resembles the human decision making process. This means that it does not require a complete set of data in order to solve a problem. The knowledge is stored in memory as separate "cases" defined only by the defect in the structure, this is important because it allows fast construction of a knowledge base. It also allows for easier system maintenance because new cases can easily be entered into memory and old cases can be totally revised or deleted.

The ability of the CBR system to provide a quick answer is also desirable. The system indexes important information in the case and looks for a similar case. If there is an exact case in the knowledge base, almost instantaneously the solution can be displayed and implemented.

It is very important to determine which are the real damages presented in the structure, coherent and logical damages. In fact the system is able to train with a lot of cases (infinite), however in practice it is not certain, due to storage limitations. Therefore, it is not appropriate to load the system with damages that never will happen.

The main value or innovation of this system is the exploitation of the model of the structure to pre-load the casebase. In this way, when the system is put in operating mode, it is able to detect damages given a very good performance even before loading any real damage in the casebase.

7 REFERENCES

[1] Kohonen, T.: 1990 "The Self – Organizing Map", Proceedings IEEE, vol 78, No 9.

[2] Lopez de Mantaras, R; Plaza, E.: 1997. "Case-Based Reasoning: An Overview". AI Communications Vol. 10(1), pp 21-29

[3] Pittner, S., Kamarthi, S.V.: "Feature extraction from wavelet coefficients for pattern recognition tasks". Pattern Analysis and Machine Intelligence, IEEE Transactions on, Volume: 21 Issue: 1, pp. 83-88 (1999)

[4] Watson I.: 1995, "An introduction to case-based reasoning". Springer-Verlag,

[5] Watson I.: 1998, "CBR is a Methodology not a Technology". In, Research & Development in Expert Systems XV. Miles,R., Moulton, M. & Bramer, M. (Eds), pp 213-223. Springer-London. ISBN 1-85233-086-4

Investigating the feasibility of using fuzzy logic-based method for transients classification in nuclear power plants

Piero Baraldi

The early identification of the causes for the onset of an unscheduled and meaningful departure from steady state behaviour is an essential step for the operation, control and accident management in nuclear power plants. The basis for the identification is that different system faults and anomalies lead to different patterns of evolution of the involved process variables. Given the safety and economical importance of the problem, several approaches for fault identification have been investigated and many efforts are continuously devoted to the improvement of the results thus far obtained.

The problem of fault identification may be tackled as a problem of classification. The classes are the different faults or anomalies of the plant, while the signals upon which the classification is based are the plant process variables. Our work has concerned the investigation of the capabilities of fuzzy logic in this area. One of the advantages of approaching the classification problem by fuzzy clustering is that the membership values found can serve as a confidence measure in the classification: for example, if a vector is assigned 0.9 membership in one class and 0.05 membership in two other classes we can be reasonably sure the class of 0.9 membership is the class to which the vector belongs. On the other hand, if a vector is assigned 0.55 membership in class one, 0.44 membership in class two, and 0.01 membership in class three, then we should be hesitant to assign the vector based on these results.

The key issue is the definition of a set of fuzzy *if-then* rules capable of associating the correct fault class to the various process variables transients which may occur. To this aim, we have developed a method of supervised training which automatically generates the proper rule corresponding to a given transient. We consider c possible transient-causing faults and suppose that a set of L numerical input/output vector pairs $(\overrightarrow{x_l}, q_l)$, $l = 1, ..., L$ is available from the plant. Each component x_l^i of $\overrightarrow{x_l}$ is a process variable and the corresponding q_l is an integer denoting the particular fault that has lead to the pattern evolution $\overrightarrow{x_l}$. These data are used to generate a set of fuzzy *if-then* rules representative of the correspondence between the input space of $\overrightarrow{x}$ and the output space of the fault class q. Once the training is completed, the fuzzy model is defined and one can feed it with a new input vector

$\vec{x}^*$ to determine the corresponding class q^*, i.e. the fault that has caused the plant transient. The method developed has been successfully applied to the classification of the causes of the transients in a steam genarator of a Pressurized Water Reactor (PWR): based on the measured signals, the forcing function responsible for the transient is readily classified.

The main disadvantage of the approach is the large number of rules of the resulting model. In our case, we obtained 266 *if-then* rules which are not physically interpretable so that the model is a "black box" not easily interpretable by the plant operators.

To improve this aspect, we are investigating the feasibility of using neuro-fuzzy systems and fuzzy clustering methods. In particular, with respect to the latter approach we would like to partition the process variables data into c clusters such that each cluster corresponds to one of the c fault classes.

In this area we have approached our investigation by looking at the popular Fuzzy C Means (FCM) method which searches for hyper-spheres or hyper-ellipsoidal clusters in the space of the input data. The FCM algorithm finds the centers of the c clusters and the degrees of membership of each of the L training data to each cluster, by iteratively minimizing an appropriately defined function which measures the distance, usually in an Euclidean metric, between the L data and the centers of the c clusters. The approach, however, is limited to a well defined geometric partition of the input data, thus depending on the metric assumed, and gives no a priori account to the fault class to which the data belongs. In our experience, this results in only only few of the identified clusters containing data actually belonging to a single class, the remaining clusters containing data belonging to more than one class. In this respect we are developing a method of data classification in which an evolutionary algorithm is employed to search for the optimal Mahalanobis metric on the basis of which the FCM algorithm derives a partition of the data set which accounts also for information on the fault class and is as close as possible to a priori known faults classification.

A Fuzzy Web System for Community Building in Complex Environment

Raffaele Giordano
Dept. of Architecture and Urban Planning
Polytechnic of Bari
Bari - Italy

1 Complex Environment and Territorial Relations Network

This contribution deals with the definition of a web-intelligent system able to support social interaction in complex environment, like an Industrial District (ID). Industrial districts, whose main character is the agglomeration of medium/small industries, represented in the past and still represent a successful model of industrial production organization, a sort of Italian way to overcome the difficulties found by the big industry (Albino, 2002).

Several research studies have been realized dealing with ID, emphasizing the role of *spatial aggregation* phenomena, able to create external economies and competitive advantages for firms located inside the ID.

In this perspective, the local territory is able to play an important role in ID development, connecting locale and global through a *versatile integration* process (Beccatini e Rullani, 1993), which create exchanges between local knowledge created by ID firms and knowledge in global network. Therefore, territorial contest could be considered as an experiential contest which allows a continue renewal of practices through innovation processes supported both by social dynamic and interactions with contextual cultural sphere (Camagni, 1989).

The territory represents the physical infrastructure which allows the reiteration of contacts among different local actors. In this perspective, both inter-firm relations and firms-territory relations characterize the ID.

2 Relations Network and Collective Learning Process

In the current economic scenario, characterized by a growing global economic competition among companies and countries, knowledge seems to be a relevant comparative factor: the success is related both to learning capacities and to capabilities to use learning process better than others.

The introduction of learning concept seems interesting because it allows to amply explication models of performances of both individuals and organizations, allowing to adopt knowledge as a causal factor (Calafati, 2002b).

When dealing with complex systems, like ID, I becomes fundamental to adopt a collective perspective in learning process, that is a learning process involving a community of agents is needed. The collective learning concept seems be useful to comprehend relations between changes at individual level to changes at system level (Calafati, 2002b).

Therefore, a systemic approach is requested to analyse cognitive process in complex system as ID. In this approach, intelligence of a human system is not enough to create improvement condition. On one hand, obstacles to implementation of strategy and solution have to be considered. On the other, strategies efficacy of open system depends on environmental characteristics. In this perspective, IDs respect complex systems peculiarity: system characteristics are not derivable from single part characteristics (von Bertalanffy, 1969).

System concept requests to pay attention to system intelligence, that is, to mechanisms able to control system evolution trajectory considering environment constraints-possibilities matrix (Calafati, 2001). As complex systems, local systems evolution is governed by feedback cycles among their elements: each of them has an own evolution dynamic (Bertuglia e Staricco, 2000).

In this perspective, agent interactions become fundamental and learning process could e defined as a change of system relations structure (Calafati, 2002a). Therefore, learning processes are strongly related with communication interaction. In fact, the information emerged from communication improve mental process of the agents enhancing learning process.

Using this approach it can be possible to understand the importance of physical proximity in ID. In fact, along with both shared language and shared social relations, it facilitates agent interactions and structures information flows, influencing, therefore, learning process.

Innovations in Information and Communication Technology seem particularly interesting in our study domain because thet are involving a fundamental systemic component: relation networks (Chiarversio e Micelli, 2000). In fact, ICT tools diffusion could create a remarkable increment of both quantity and type of interactions, between, individuals, organizations and external environment, provoking an increment of evolution dynamics complexity of local systems.

The more relevant potentiality of ICT for local complex system, as ID, regards its capability to greatly amply the individual relational spectrum by:

- Sustaining already existing relational network;
- Improving the creation of contacts with new interlocutors.

Technological innovation processes configure new form of communication processes management which redesign relational system, both effective and potential (Calafati, 2002a). They generate conditions for creation of new cooperation environment, not necessarily local (Chiarversio e Micelli, 2000).

3 Fuzzy Web-based System for Community Creation

This research work, moving from concepts descript in previous pages, deals with definition of a web-base system able to facilitate interaction inside a local community, particularly Industrial District. Research focus is on improvement of logistic in ID by improvement of communication among agents.

To improve those interactions an emarketplace can be used. It could be defined as an electronic agora where a set of persons or agents can be involved in exchanging services and information. The e-marketplaces are today primarily focused on the matchmaking of buyers and sellers. To define the e-marketplace, primarily the two interest communities have to be built up (sellers and providers). To this aim, what is in specific technical literature about the creation of a coalition among intelligent agents is used. A coalition is a set of agents, each one with his own interests, who draw up a cooperation agreement to carry out a piece of work or achieve a goal (Sheory et al., 1998). A group has to be entrusted with a task when single agents cannot carry out sufficiently or at all the same task (Sheory and Kraus, 1998).

The cooperation process can be divided in the following stages:

- finding someone to collaborate with;
- making contact with the selected people;
- building a common understanding: that is, the identification of a goal and the way to reach this goal;
- coordinating activities and work plans through communication among co-workers.

First two stages are emphasized in this research work. In this sense, the communityware research field seems very interesting. In fact, the communityware can be defined as an electronic medium that facilitates the contact with collaborators who have similar interests and preferences, but do not know each other (Raisch, 2001). The communityware has to include essentially three different functions to encourage interactions:

- knowing each other;
- sharing preferences and knowledge;
- generating consensus.

The basic idea is that multiple Internet Agents can form groups of people who share the same interests by analyzing the individual's interests. Resulting clusters can be used for cooperative solution of problems.

As we stated previously, the first step to allow cooperation among unknown agents concerns the contact facilitation, that is the individuation of agents sharing same interests, and making easier the contacts among them. In order to support this process, attributes describing the individuals are required (Raisch, 2001). In this paper, the individuals' attributes concern the attributes of transportation demands: shipment time, destination, type of product, quantity of product, etc.

The algorithm for the coalition formation has some attributes difficult to define in a rigorous way. In fact, to identify the agents able to form the customers' coalition, customers with similar requests, with regard to the shipment date and destinations, have to be considered. The similarity concept seems to be difficult to define using crisp values. In the real world, the human reasoning is based on approximate values or linguistic statements instead of numeric or precise values. Therefore, it seems appropriate to use the fuzzy logic to define the attributes for the coalition formation process. In fact, the fuzzy logic allows performing operation on variables defined in an approximate way and handling variables defined in linguistic terms.

To understand better the use of fuzzy logic in implementation of the proposed system, we focused our attention on the formation process of customers' coalition. The system identifies the possible coalition members comparing different requests and finding possible similarity. In particular, in our research we refer to the Fuzzy Clustering.

Let n be the number of customers, included in the Customers Interest Community, who expressed a request. Split now, on the basis of significant indicators that should characterize each request, these customers into c homogenous subsets (clusters), with $2 \leq c < n$. The customers belonging to anyone of the clusters should be similar to each other, and as dissimilar as possible from the objects of different clusters (Zimmermann, 1991). Classical clustering algorithm, based on bi-value logic, generate partition in which each elements belong or not to a data class. In real world, classes of elements are fuzzy rather than crisp (Dumitrescu et al., 2000). In this sense, "strictly" assign an element to a cluster could be lead to a mistake, because elements are often located *between* classes (Zimmermann, 1991) rather *inside* them.

Currently, this research work deals with definition of a process of fuzzy clustering based on intra-class similarity measure able to build community of interest in ID transportation problem.

Application of a Genetic Algorithm to a Scheduling Assignement Problem

Amândio Marques[a] and Francisco Morgado[b]
[a]CISUC - Center of Informatics and Systems of University of Coimbra, 3030 Coimbra, Portugal
[b]Polytechnic Institute of Viseu, 3500 Viseu, Portugal
email: amandio@dei.uc.pt, Tel: 239790057

Abstract

A set of heuristics are used successfully in a schedulling problem within the framework of healthcare medical services. Emphasis is given to the genetic algorithm which looks for the best schedule problem solution.

1 Introduction

The main objective of this work is the development of an intelligent system based on genetic algorithms to assist the planning of shifts scheduling in a local Hospital. The system will ease the current scheduling edition acting as an advisory to prevent uncorrected distributions assignment which lead to not enough resting periods of the health professionals and to a lack of parity concerning time and type of service. Clearly, these reasons cause inappropriate medical service care.

The specific objectives of the developed application are:

1. To take into account the number of working hours in excess or missing of the healthcare professional;
2 To allow a generic specification of the health care service requirements;
3 To visualize all the healthcare professionals assigned to a specific day shift;
4. To propose shift schedules sought for proper parity and balanced distribution for mid and long term;
5. To allow the adjustment of the proposed shift schedule.

2 Problem Formulation

The following parameters have to be defined:

F	- Set of healthcare professionals;
D	- Number of days of the schedule period;
T	- Number of Shifts;
Nec_{dt}	- Healthcare needs wrt Shift t of day d;
TRD	- Number of Shifts per day;

The problem solution can be formalized by the following variables which express each healthcare professional assignment to the care specific needs:

$$X_{fdt} = \begin{cases} 1 \text{ if professional } f \text{ does Shift } t \text{ of day } d \\ 0 \text{ otherwise} \end{cases}$$

with $f \in F$, $d \in D$, $t \in T$.

2.1 Problem Constraints

The solution admissibility is impose by the following constraints:

$$\sum_{f \in F} X_{fdt} = Nec_{dt} \qquad d \in D, t \in T \tag{1}$$

In each shift day the number of healthcare professionals have to satisfy the service needs.

$$\frac{\sum_{d \in D, f \in F} X_{fdt}}{\#D} = TRD \qquad f \in F \tag{2}$$

Each healthcare professional should fulfill the shifts specified in his working contract.

2.2 Objective Functions

The two objective functions to be minimized are as follows. Equation (3) is the objective function designated hereby Disorder:

$$Z_1 = \frac{\sum_{d \in D, f \in F, t \in T} (X_{fdt} \times penalty(fdt))}{\#F \times \#D \times \#T} \tag{3}$$

which corresponds to the mean of the penalties of the bad assigned shifts. The objective function Unfairness is given by (4):

$$Z_2 = 1 - \frac{\sum_{d \in D, t \in T} \left(Ptd_{td} \frac{1}{1 + mean\ dev(Ttr_{fdt}\ f \in F)} \right)}{\sum_{d \in D, t \in T} Ptd_{dt}} \tag{4}$$

It is calculated by the weighted mean of the inverse of the dispersion resulting from the service distribution per shift and type of day.

3 Heuristics versus Optimization Method

Although we have linear constraints, the problem can be cast in the context of multiobjective nonlinear binary programming, due to the nonlinearity of the objective functions.

Since linear methods are not adequate to solve this type of problems, other approaches seem adequate. Therefore, an heuristic based approach was here successfully applied and it will be described next.

```
For each day ( d )
   Calculates Shift Pattern (d,d+5)
   For all Shifts(t), from n down to 1
        While needs (d,t) are not fullfilled
          Select professional (f) more appropriate
          Assign professional (f) to the needs (d,t)
         End While
   End For
End For
```

4 Finding the Best Solution

We claim that for this problem (i) the performance of the proposed solutions can be properly evaluated, (ii) the problem is complex being NP complete; (iii) it has not been found yet an exact method to determine the best solution; (iv) the problem involves a large number of variables, thus occurring the curse of dimensionality.

For a problem with such characteristics the best approach to be used relies on Genetic Algorithms.

5 Genetic Algorithms

Genetic Algorithms (GAs) perform a stochastic global search method that mimics the metaphor of natural biological evolution. GAs operate iteratively on a population of individuals (solutions). In each iteration all the members are evaluated according to a fitness function. The lowest fitness individuals are eliminated and from the crossover of the remaining ones, a new generation of solutions is created following a mutation which is realized in a small percentage completing thus the cycle. This cycle is repeated until a stop condition is reached (see Figure 1).

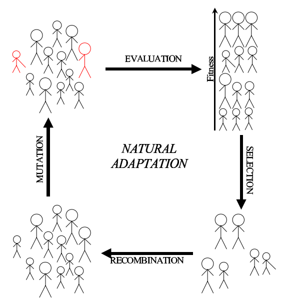


Figure 1: Genetic Algorithm Schema.

The individuals representation assumes an important role in any genetic algorithm approach In this case, the individual, a shift scheduling, is represented by:

$$X_{fdt} \qquad f \in F, d \in D, t \in T$$

The fitness of a specific solution is given the weighted sum of the objective Disorder and Unfairness. Since we have a multiobjective problem two strategies are presented.

In order to find a solution that minimizes a weighted average of the two objective functions, the ranking is obtained by sorting the solutions according to $Z = pZ_1 + (1-p)Z_2, 0 < p < 1$.

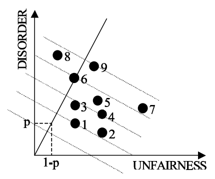


Figure 2: Function F_1.

To find the best solutions, either the Disorder or the Unfairness, the ranking should be done iteratively, capturing the trade-off between these objectives, the so-called Pareto curve (see Figure 2), inserting the found solutions in the ranking and removing them from the initial list. The process is repeated until any other solution can be found.

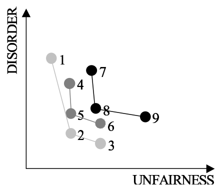


Figure 3: Function F_2.

The crossover operator combines two solutions (progenitors) from the actual generation with identical fitness and it generates two solutions (descendents) recombining portions of both parents. We take two solutions $XP1$ and $XP2$, and from a random $x \in D$ two new solutions $XD1$ and $XD2$ are generated as follows:

$$
\begin{aligned}
XD1_{fdt} &= XP1_{d<x} \cup XP2_{d \geq x} \; f \in F, d \in D, t \in T \\
XD2_{fdt} &= XP1_{d \geq x} \cup XP2_{d<x} \; f \in F, d \in D, t \in T
\end{aligned}
$$

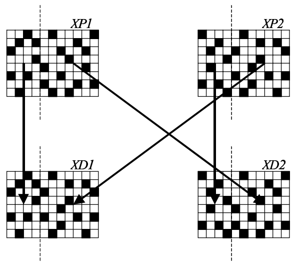


Figure 4: Crossover operation.

The mutation operates randomly on the chromosomes of an individual. To allow the convergence of an algorithm this operator is used with a low frequency. The mutation is achieved choosing randomly two heathcare professionals f_1, $f_2 \in F$, two days $d1, d2 \in D$, and two shifts $t1$,$t2 \in T$ that verify the following condition:

$$X_{f_1 d_1 t_1} = 1 \wedge X_{f_2 d_2 t_2} = 1 \wedge X_{f_1 d_2 t_2} = 0 \wedge X_{f_2 d_1 t_1} = 0$$

and changing their values, respectively.

$$X_{f_1d_1t_1} = 0 \wedge X_{f_2d_2t_2} = 0 \wedge X_{f_1d_2t_2} = 1 \wedge X_{f_2d_1t_1} = 1$$

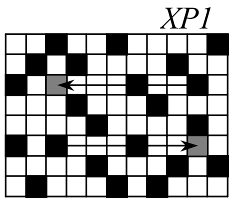


Figure 5: Mutation operation.

The evolution is associated with diversity. To achieve a good diversity on the initial population, based on the heuristic proposed, new heuristics were derived to generate purely random solutions, or random solutions to favoring the decreasing Disorder or to favoring the decreasing Unfairness.

6 Conclusion

The proposed approach solving a schedule shift problem base on a standard genetic algorithm was successfully implemented. The new heuristics generated a valid good shift scheduling. The genetic algorithm optimized the solutions found with the defined heuristics. The results show good agreement with the needs of the medical care service as well as the personal preferences of healthcare professionals.

7 References

1. Melanie Mitchell (1996). *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts.

2. David E. Goldberg (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley Longman.

3. Stroustrup, Bjarne (1995). *The C++ Programing Language.* Second Edition, Addison-Wesley.

4. Guerreiro, Jorge et al.(1985), *Programação Linear*, Vol II, Lisboa, Mcgraw-Hill.

5. Niklaus Wirth (1975) . *Algorithms & Data Structures.* Prentice-Hall.

Analysis of RBS data by Artificial Neural Networks: a systematic approach

Armando Vieira* Helder Pinho

ISEP, R. S. Tomé, 4200 Porto, Portugal

Rutherford backscattering (RBS) is a non-destructive, fully quantitative, technique for accurately determining the compositional depth profile of thin films. The inverse RBS problem, which is to determine from the data the corresponding sample structure, is however in general ill-posed. Skilled analysts use their knowledge and experience to recognize recurring features in the data and relate them to features in the sample structure. This is then followed by a detailed quantitative analysis. Artificial Neural Network (ANN) have already been sucessufuly applied to data analysis of implantations of Ge in Si, and Er in saphire among others. In this work we show the first results of using neural networks to a more general problem, namely implantations of *any element* in *any substract* under *any experimental conditions.* This is a very hard problem for a ANN where we used housands of constructed spectra of samples for which the structure is known. We used a efficient algorithm to extract features from the 512 channel spectra, thus reducing drastically the dimensionality of the data. The ANN learns how to interpret the spectrum of a given sample, without any knowledge of the physics involved. The ANN was then applied to experimental data from samples of unknown structure. The quantitative results obtained were compared with those given by traditional analysis methods, and are excellent. The major advantage of ANNs over those other methods is that, after the time-consuming training phase, the analysis is instantaneous, which opens the door to automated on-line data analysis. Furthermore, the ANN was able to distinguish two different classes of data which are experimentally difficult to analyze. This opens the door to automated on-line optimization of the experimental conditions.

*Corresponding author: asv@isep.ipp.pt

Estimation of Distribution Algorithms

Petr Pošík

Czech Technical University, Faculty of Electrical Engineering, Dept. of Cybernetics
Technická 2, 166 27 Prague 6, Czech Republic
E-mail: `posik@labe.felk.cvut.cz`, phone: +420-2-24357228

1 Introduction

Evolutionary algorithms (EAs) are known in many areas as a powerful and robust optimization and searching tool. Classical EAs rely on the well-known two phases: selection and variation. Variation is usually carried out by means of perturbation of promissing individuals (searching local neigbourhoods), or by means of combining two promising individuals together (creating offsprings which embody some characteristics of both parents). However, classical EAs suffer from several problems. The linkage problem belongs among the most severe ones. It arises in situations when the individual components of chromosomes are not statistically independent of each other with respect to the fitness function. There exists no general way of EA modification that would enable the modified EA to account for the dependencies at hand. Usually, this problem is solved by constructing special crossover and mutation types of operators and by incorporating some problem-specific knowledge in them. The classical EA then looses its flavor of general problem solver and quickly becomes an algorithm highly specialized to the given problem.

2 Estimation of Distribution Algorithms

Recently, a new type of EAs emerged — *Estimation of Distribution Algorithms* (EDAs) [1]. Some researchers use names as *Probabilistic Model Building Genetic Algorithms* (PMBGAs), or *Iterated Density Estimation Algorithms* (IDEAs), but all these names describe basically the same concept. These algorithms don't rely on the 'genetic' principles anymore; instead, in each generation, they build an explicit probabilistic model of distribution of 'good' individuals in the search space. New individuals are created by sampling from this distribution. The *model-sample* step of EDA can be thought of as a generalized type of multiparent crossover operator. The strengths and weaknesses of a particular EDA are mainly determined by the used probabilistic model.

2.1 Probabilistic Models for Discrete Variables

The probabilistic models differ for EDAs in discrete and continuous spaces. The first EDAs were developed for the discrete spaces. They range from simple *Univariate Marginal Density Algorithm* (UMDA), which is comparable to simple genetic algorithm, to *Bayesian Optimization Algorithm* (BOA) [2] which uses Bayesian net as the underlying probabilistic model. Bayesian nets are able to encode general type of discrete probabilistic distribution, however, their learning from data involves either sophisticated methods for statistical dependency detection, or they are learnt by searching the space of possible Bayesian nets (usually by a greedy algorithm).

2.2 Probabilistic Models for Continuous Variables

In continuous spaces, the situation is even more complicated. The simplest continuous EDAs (continuous UMDAs) use models in which the joint probability density function (PDF) is factorized into a product of marginal univariate PDFs which take various forms: empirical histograms, normal (or any other well-known) distribution, finite mixtures of univariate Gaussians, etc. To take into account the dependencies between variables, we have to employ more complex models like Gaussian nets (GN), which results in *Estimation of Gaussian Networks Algorithm* (EGNA) [1]. GN has the power to encode general multidimensional Gaussian distribution, however, very often this type of probabilistic model is not sufficient. Then we should use even more flexible models which are empowered by (hard- or soft-) clustering, e.g. finite mixture of multidimensional Gaussians. To be objective, one must say that these models are capable in covering various types of interactions, however, learning them is not a trivial task. It is usually very time consuming and it must be performed using a kind of iterative learning scheme (usually by a variant of the expectation-maximization algorithm).

3 Marginal Models in EAs

My research is aimed at the EDAs in continuous spaces. I have examined the UMDA in continuous domain. The individual components of promising solutions are supposed to be statistically independent of each other. This means that the global distribution of promising solutions in the search space can be modeled by a set of univariate marginal distributions, i.e. the global model can be factorized as

$$p(\mathbf{x}) = \prod_{d=1}^{D} p_d(x_d), \tag{1}$$

where the $p(\mathbf{x})$ is the global multivariate density and the $p_d(x_d)$'s are the univariate marginal densities. I compared the suitability of four different marginal probability models, namely the equi-width histogram (HEW), equi-height histogram (HEH), max-diff histogram (HMD), and univariate mixture of Gaussians (MOG) (for the differences of individual models, see fig. 1). For the suite of test functions, see [3].

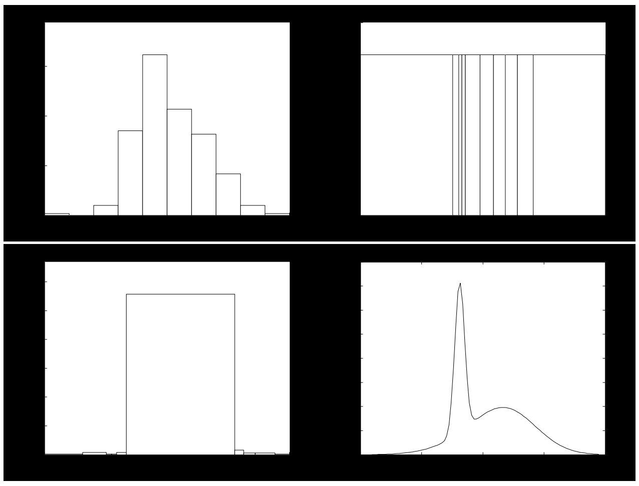


Figure 1: Equi-width, equi-height, and max-diff histograms with 10 bins, and mixture of gaussians with 3 components

In the experiments, I varried the population size (200, 400, 600, 800 individuals), the number of bins for histogram models (120 and 60 bins), or the number of components for the case of MOG model (6 and 3 components). Furthermore, all models were compared to the *line search* heuristic [4], which is very efficient for high-dimensional separable problems.

In each generation, new *PopSize* individuals were created, joined with the old population, and using truncation selection, the population was reduced to its original size. For each of possible factor combination I run the algorithm 20 times. Each run continued until the number of 50,000 evaluations was reached. Let's say the algorithm found the global optimum if for each variable x_d the following relation holds: $|x_d^{best} - x_d^{opt}| < 0.1$ (if the difference of the best found solution x^{best} from the optimal solution x^{opt} is lower then 0.1 in each of coordinates). In all experiments, we track three statistics:

- The number of runs in which the algorithm succeeded in finding the global optimum (*NoFoundOpts*).
- The average number of evaluations needed to find the global optimum computed from those runs in which the algorithm really found the optimum (*AveNoEvals*).
- The average fitness of the best solution the algorithm was able to find in all 20 runs (*AveBest*).

The results can be found in table 1. From the experiments the following conclusion can be made: the HEW model is the least flexible one and the behaviour of EDAs with this model is unsatisfactory in comparison with the other models. The performance of HEH and HMD histograms was comparable. The MOG model showed a bit worse performance, however, it used considerably less components than the histogram models and offers other advantages over the histogram models (easy extension to mixture

Function	Model	Number of Bins (Components)								
		Low				High				
		Population Size								
		200	400	600	800	200	400	600	800	Statistics
Two Peaks	LS	20								NoFoundOpts
		2421								AveNoEvals
		0,00000								AveBest
	HEW	0	1	0	1	1	17	19	20	NoFoundOpts
			13200		47200	5600	10306	15189	19600	AveNoEvals
		5,00560	4,82110	5,05280	4,95510	3,48040	2,51180	2,50030	2,52340	AveBest
	HEH	20	20	20	20	20	20	20	20	NoFoundOpts
		6530	11080	16050	20760	7720	10620	15570	20400	AveNoEvals
		0,10710	0,02070	0,01040	0,07490	0,03740	0,00690	0,00430	0,03520	AveBest
	HMD	20	20	20	20	20	20	20	20	NoFoundOpts
		6270	14920	27960	47080	6770	10980	17490	25280	AveNoEvals
		0,00003	0,00008	0,04990	2,20380	0,00260	0,00001	0,00230	0,05740	AveBest
	MOG	7	20	20	19	16	20	20	20	NoFoundOpts
		6571	11860	17130	23032	5613	10480	15780	20720	AveNoEvals
		0,87390	0,00011	0,00930	0,13970	0,19650	0,00008	0,00660	0,06370	AveBest
Griewangk	LS	18								NoFoundOpts
		14056								AveNoEvals
		0,00250								AveBest
	HEW	13	17	16	14	1	15	18	19	NoFoundOpts
		15954	20353	25763	27886	5800	12320	18867	24926	AveNoEvals
		0,00370	0,00210	0,00230	0,00320	0,00500	0,00081	0,00095	0,00083	AveBest
	HEH	18	18	18	20	16	18	20	20	NoFoundOpts
		6667	12867	18967	25000	6650	12711	18270	23920	AveNoEvals
		0,00074	0,00074	0,00074	0,00000	0,00150	0,00074	0,00000	0,00000	AveBest
	HMD	17	17	18	16	18	17	19	20	NoFoundOpts
		6482	12376	19200	25850	6456	12235	18221	23720	AveNoEvals
		0,00110	0,00110	0,00074	0,00140	0,00074	0,00110	0,00037	0,00000	AveBest
	MOG	15	15	18	19	19	19	17	18	NoFoundOpts
		5693	10613	16100	21726	5894	12084	17682	23644	AveNoEvals
		0,00190	0,00180	0,00074	0,00037	0,00037	0,00037	0,00110	0,00074	AveBest

Table 1: Results of carried out experiments

of multidimensional Gaussians). Typical tracks of evolution of bin boundaries for histogram models and component centers of MOG for one of the test functions is shown in figure 2.

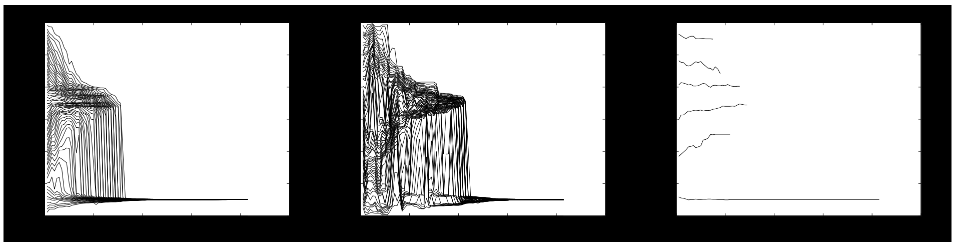


Figure 2: Two Peaks function — Evolution of bin boundaries for equi-height and max-diff histogram models and evolution of component centers for mixture of Gaussians model.

4 Vestibulo-Ocular Reflex Analysis

The above described algorithm was succesfully applied to vestibulo-ocular reflex (VOR) signal processing. By analyzing the VOR signal, physicians can recognize some pathologies of the vestibular organ of a patient in a non-invasive way. The principle is simple: the patient is situated in a chair which is then rotated in a defined way (following some reference signal – sine wave or sum of sine waves). The patient is said to visually track some points on surrounding walls and the movements of his eyes are monitored. The resulting eye signal must be first processed (it is distorted by the fast eye movements) to get 'eye-filtered' response to the reference signal. The differences in amplitudes and phases of the sine waves are the indicators of the vestibular organ pathologies. EDA was applied in the signal processing phase in a co-evolutionary manner, i.e. the following two parts were iteratively alternated: (1) one population searched for the best biases of individual signal segments (when fitted to the best representant of estimated signal

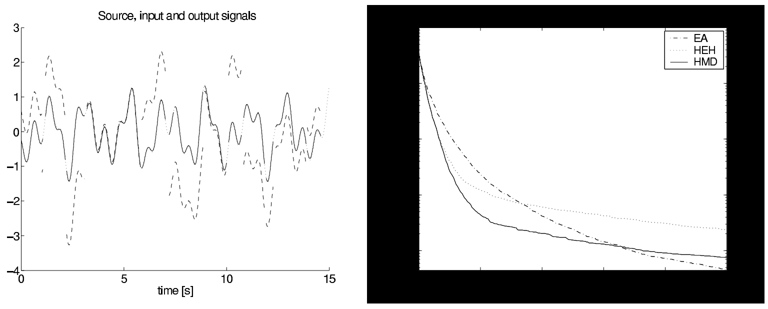


Figure 3: VOR signals and their evolution.

parameters), while (2) the other population searched for the best signal parameters (when fitted to the best set of biases).

I compared two of the above mentioned UMDAs (HEH and HMD models) with an ordinary EA (truncation selection, 2-point crossover, mutation with probability 0.05 by means of adding a random value from distribution $\mathcal{N}(0, 0.1)$). The results are presented in fig. 3. On the left-hand side of the picture, we can see the original signal (dotted line, not known to the EA), the signal segments before EA started (dashed line), and the same signal segments after EA processing (solid line). We can see, that all the segments are very precisely arranged so that they follow the original signal very closely. Both UMDAs were much faster than the EA in the initial phase of evolution. The EA is able to discover more accurate solution, however, the final differences are not very large – measured in residual sum of squares (RSS), the EA reached the score of 0.44 on average, the UMDA/HMD reached 0.75, and the UMDA/HEH reached 2.36 (solutions of these scores are almost identical when compared by human eye). In spite of these differences, all these EAs reach much more accurate results (in terms of RSS) than conventionally used methods based on some form of interpolation.

5 Future Work

In the near future, I would like to implement an EDA using mixture of principal component analysers (MPCA) and test it on several artificial and practical problems (e.g. on Hough's transformation used in image processing, or for hidden Markov models training). The aim of these comparative studies is to find out if it is worth to use such complex models (e.g. MPCA), in other words, if the time spent on learning the model each generation is lower than the time the simple EA needs to find a solution of comparable quality.

Acknowledgements

The project was supported by the Grant agency of the Czech Rep. with the grant No. GACR 102/02/0132 entitled "Use of Genetic Principles in Evolutionary Algorithms", and was supervised by Jiří Lažanský.

References

[1] Larrañaga, P., Lozano, J.A. Estimation of Distribution Algorithms. Kluwer Academic Publishers, 2001.

[2] Pelikan, M. Bayesian Optimization Algorithm: From Single Level to Hierarchy. IlliGAL Report No. 2002023. University of Illinois at Urbana-Champaign. 2002.

[3] Pošík, P. Optimization of Evolutionary Algorithms. Technical Report GL 164-03, Department of Cybernetics, Czech Technical University in Prague, 2003.

[4] Whitley, D., Mathias, K., Rana, S., Dzubera, J. Evaluating Evolutionary Algorithms. Artificial Intelligence Volume 85, p. 245-276, 1996.

Evolving Takagi-Sugeno Fuzzy Models for Data Mining

J. Victor Ramos

Centro de Informática e Sistemas da Universidade de Coimbra
Grupo de Computação Adaptativa
Pólo II – DEI, 3030-290 Coimbra
Email: zevictor@dei.uc.pt

Escola Superior de Tecnologia e Gestão de Leiria
Departamento de Engenharia Informática
Morro do Lena – Alto do Vieiro, 2411-901 Leiria
Email: zevictor@estg.ipleiria.pt

Abstract - We are studying and experiencing approaches for adaptive on-line learning of fuzzy rules and their application for prediction problems in the context of data mining. Takagi-Sugeno (TS) fuzzy models are used for knowledge representation and the mechanism for on-line learning is based on algorithms that recursively update the model structure and parameters by combining supervised and unsupervised (hybrid) learning. The structure and parameters of the model continually evolve by adding new rules and by modifying existing rules and parameters during the operation of the system. The work is based on developments from the original contributions of Stephen Chiu, Plamen Angelov and Nikola Kasabov.

Key words: Takagi-Sugeno fuzzy models, fuzzy clustering, rule-base adaptation, on-line learning.

1. Evolving Takagi-Sugeno Fuzzy Models

Takagi-Sugeno fuzzy models have recently become a powerful practical engineering tool for modelling of complex systems. Evolving rule-based models use methods for learning models from data are based on the idea of consecutive structure and parameter identification. Structure identification includes estimation of the focal points of the rules (antecedent parameters) by fuzzy clustering. With fixed antecedent parameters, the TS fuzzy model transforms into a linear model. Parameters of the linear models associated with each of the rule antecedents are obtained by applying the recursive least-squares (RLS) method or the weighted recursive least-squares (wRLS) method.

For on-line learning of the TS fuzzy models it is necessary an on-line clustering method responsible for the model structure learning. Angelov proposed a new method inspired on the subtractive clustering algorithm that allows the recursive calculation of the informative potential of the data, which represents a spatial proximity measure used to define the focal points of the rules. Evolving rule based models use the information potential of the new data sample as a trigger to update the rule base.

The evolution mechanism is basically the following: If the information potential of the new data sample is higher than the potential of the existing rules a new focal point (rule) is created. If the new focal point is too close to a previously existing rule then the old rule is replaced by the new one. The advantage of using the information potential instead of the distance to a certain rule centre only for forming the rule base is that the spatial information and history are not ignored, but are part of the decision whether to upgrade or modify the rule base.

The recursive procedure for on-line learning of evolving TS fuzzy models includes the following stages:

Stage 1: Initialization of the rule-base structure (antecedent part of the rules);

Stage 2: Reading the next data sample;

Stage 3: Recursive calculation of the potential of each new data sample;

Stage 4: Recursive up-date of the potentials of old centres taking into account the influence of the new data sample;
Stage 5: Possible modification or up-grade of the rule-base structure based on the potential of the new data sample in comparison to the potential of the existing rules centres;
Stage 6: Recursive calculation of the consequence parameters;
Stage 7: Prediction of the output for the next time step.

Despite of the merits the algorithm still needs some major improvements. The conditions to modify and upgrade the fuzzy rules are being studied more deeply since they influence the number of created and modified rules and it is not easy to adjust the definitions for a specific problem.
Another vital issue is the on-line clustering procedure, particularly the function for recursive calculation of the potential of each new data sample. Angelov used different functions (Cauchy type function of first order, exponential with the summation in the exponent) for recursive calculation of the potential but all present limitations because the local maxima of the potential do not cover all the regions of interest. New functions or estimators from information theory need to be tested to achieve a better placement for focal points covering not only the regions of higher density of points but also other regions of interest (a disturbance or a new operating mode).

2. Experimental Results

The approaches and its developments were tested on a benchmark problem, the Mackey-Glass chaotic time series prediction. The data set has been used as a benchmark example in areas of fuzzy systems, neural networks and hybrid systems. Several models were built for different parameters of the algorithm and particularly for the conditions necessary to create and modify the fuzzy rules. The results obtained were compared with other methods (ESOM, EFuNN and DENFIS) and one of the conclusions is that it is possible to obtain identical values for NDEI with a lower number of rules, i.e. more transparent models.
There are a few parameters (radii, Omega for (w)RLS) and conditions that need to be specified, which give the flexibility to tune the search. There are several possibilities for the definitions to create and modify fuzzy rules and different models will be obtained. It is quite difficult to define one condition that is the best for all types of problems.

3. Conclusion

The approaches we are studying and experimenting for on-line identification of evolving Takagi-Sugeno fuzzy models are computationally effective and despite the necessary improvements the adaptive nature of these models, in combination with the highly transparent and compact form of fuzzy rules, makes them a useful tool for on-line modelling.

References

P. Angelov and D. Filev (2003). An Approach to On-Line Identification of Takagi-Sugeno Fuzzy Models (to appear).
N. Kasabov and Q. Song (2002). DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and Its Application for Time-Series Prediction. IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, pp. 144-154, April.
S. Liu (1994). Fuzzy Model Identification Based on Clustering Estimation. Journal of Intelligent and Fuzzy Systems, vol. 2, pp. 267-278.

IS EXTRA-TERRESTRIAL INTELLIGENCE POSSIBLE? AN APPROACH USING COMPLEX SYSTEMS THEORY ON THE COMPUTATION OF THE PROBABILITY FOR THE EMERGENCE OF INTELLIGENCE.

André S. Ribeiro[1,2,3,*]

[1]Universidade Independente, Faculdade de Ciências de Engenharia, Av. Marechal Gomes da Costa, Lote 9, 1800-255 Lisboa, Portugal Tel.: (+351 21 83 61 900), Fax: (+351 21 83 61 922),E-mail: A_Ribeiro@uni.pt
[2]IADE, Instituto de Artes Visuais Design e Marketing, Avª D. Carlos I, nº4 ,1200-649 Lisboa,Tel.: (+351 21 393 96 00), Fax: (+351 21 397 85 61), Email: unidcom@iade.pt ;
[3] TM: (914702877), Email; Andre_S_Ribeiro@clix.pt

Abstract

Much has been said about the possibility of extra-terrestrial life. Some state that, due to the almost impossibility of extra-terrestrial life existence, extra-terrestrial intelligence emergence is virtually impossible.

In this paper we state that intelligence is not only possible in planets with any kind of life but also that, once life emerges, intelligence is an almost inevitable consequence [1].

If we consider a universe of elements with the capability of interacting we are in the presence of a system [2]. Using this concept of system we can establish the optimal states, in a neighbourhood, for which intelligence emergence is possible. It is then possible to obtain the probability of a system to reach those optimal states.

In the end, we prove that reaching such points is actually inevitable since they are the most probable cases because two driving forces will imply reaching optimal states.

Such forces are the entropy maximization necessity, which is a direct consequence of the second principle of thermodynamics, and the relativistic information maximization [3][4][5][6][7], which will be a measure of the system intelligence and, therefore, is used as a

---

[*] *With a PhD scholarship from the Fundação Para a Ciencia e a Tecnologia at Instituto Superior Técnico de Lisboa.*

natural selection factor. The optimal states, due to the influence of such forces, are, therefore, stable points in evolution [8][9][10].

Other principles must be taken in account. To do so is to diminish the set of optimal solutions of any m agents, n connections system. Such principles, such as the least energy principle [11], applicable to the required energy to create the system, will function as secondary forces to choose a smaller set of optimal solutions from the set of solutions previously obtained.

Each time a new condition is added the set of optimal solutions becomes smaller. Nevertheless it is not expected to reduce such set to only one solution.

Complex Systems are known for having the ability to exhibit many "unexpected behaviours" and adopt many solutions. Therefore, our goal consists only in determining optimal states within small neighbourhoods of possible states.

The determination of all optimal states involves many difficulties, mainly topological, due to the huge number of possible states [12]. Therefore a method or algorithm, able to simplify such quest is imposed.

Our method allows the establishment of the optimal number of interactions, considering as optimal number the one for which thermodynamic entropy is minimal and capacity to store information is maximal [10][11]. Such method consists in:

1) From previous works we know the conditions and, therefore, the rules, to obtain optimal states for any m agents, n connections system, capable of adopting any structure.

The two most important conditions are given by:

$Ns = [m.(m-1) / 2] + 1$ (1) – Number of states

$\forall k \in N, \forall 0 < n_k < m.(m-1)/2: n_k = k.m \wedge n_k = n_{max} - n_k$ (2) - Optimal States

2) With such knowledge we are capable to estimate the number of all possible states of the system.

3) We determine, according to the conditions for intelligence emergence, the number of optimal states and, from that, its probability of occurrence in a system with no preferential structure.

Since states are not all equally probable and establishing the driving forces of evolution, we end by proving that the most probable states are also the optimal ones, thereby explaining its relatively fast emergence on Earth and predict the probable emergence anywhere else where agents can connect themselves to create systems, provided that simple agents can start creating any sort of interaction between them.

[1] "Investigations", Stuart A. Kauffman, Oxford University Press, 2000.
[2] "Hidden Order", J. Holland, Addison Wesley, 1995.
[3] Carvalho Rodrigues, F., 1989. "A proposed entropy measure for assessing combat degradation", J. Opl. Res. Soc. (UK), **40** (8): 789-93.
[4] Carvalho Rodrigues, F., Dockery, J., Woodcock, A.E.R. 1992, "Models of combat with embedded command and control: Casualty based entropy calculations as a combat predictor", chapter 2.5, in: *The Military Landscape, Mathematical Models of Combat*, Woodhead Press, Cambridge, UK.
[5] Carvalho Rodrigues, F., Dockery, J. and Rodrigues, T., 1993b, "Entropy of Plagues: A Measure Assessing the Loss of social Cohesion Due to Epidemics", European J. of Operational Research, **71**, 45-60.
[6] Carvalho Rodrigues, F., Dockery, J., 1996, "Defining Systems Based on Information Exchange: Structure from Dynamics", BioSystems **38**,229-234.
[7] Carvalho Rodrigues, F.,1990/1991, "Função de Coesão de Estruturas baseadas em Informação", Memórias da Academia, TOMO XXXI, Academia das Ciências de Lisboa.
[8] "Comparative Mammalian Brain Collections",
http://www.neurophys.wisc.edu/brain/evolution/index.html/
[9] "Fossil Hominids: the evidence for human evolution",
http://www.talkorigins.org/faqs/homs/
[10] "Cálculo do Consumo Energético e Capacidade de Armazenamento de Informação em Sistemas Complexos", F. Carvalho Rodrigues, André S. Ribeiro, Setembro de 2002, Conferencia Nacional de Física 2002 in Portugal.
[11] The Cost in Energy and in Time of the Intelligence", F. Carvalho Rodrigues, André S. Ribeiro, Scientific Annals of UnI, Vol 2, no. 3, Winter 2001
[12] H. J. Bremermann, "Complexity and transcomputability", in *The Encyclopedia of Ignorance*, M. Duncan, ed. Pergamon Press, Oxford (1977).

INNER PRODUCT NETWORKS

Research student: Gonçalo Xufre Silva[(a)]**;**

Advisor: António José Rodrigues[(b)]

[(a)] Centro de Investigação Operacional – FCUL e Centro de Matemática – ISEL, Rua Conselheiro Emídio Navarro, Nº1, 1949-014 Lisboa, Portugal, Email: goncalo@dem.isel.ipl.pt;

[(b)] Centro de Investigação Operacional – Faculdade de Ciências da Universidade de Lisboa, 1749-016 Lisboa, Portugal, Email: ajrodrigues@fc.ul.pt;

1. Introduction

The two types of neural networks most used for supervised learning problems are multilayer perceptrons (MLPs) and radial basis function (RBF) networks. The main difference between them is that RBFs are linear in the parameters and MLPs are not. The only way to have MLPs linear in the parameters is to impose that the weights of the connections between input and hidden units are pre-defined and fixed during all the process of training (in a three layer network). Also, the output units have to be a linear combination of the hidden unit outputs. When we study a novel type of learning models, the support vector machines, we observe that after the learning process these models have a structure similarly to neural networks. They use a kernel function, $K(X,Y)$, that represents the inner product between $\phi(X)$ and $\phi(Y)$ where $\phi(\cdot)$ is a transformation of the training data to a higher dimension space. $K(X,Y)$ allow us to calculate the inner products between $\phi(X)$ and $\phi(Y)$ without having to know the explicitly form of the transformation function. The most used kernel functions are:

$$K(X,Y)=e^{-\frac{\|X-Y\|^2}{2\sigma^2}} \tag{1.1}$$

$$K(X,Y)=\tanh(k\cdot X\cdot Y-\delta) \tag{1.2}$$

$$K(X,Y)=\left(X\cdot Y\right)^p \tag{1.3}$$

These functions satisfy the Mercer condition [Vapnik 95] that guarantees to represent an inner product for the transformation of the variables into a higher dimension space. In the case of (1.2) this is true only for some values of k and δ. A relevant difference between SVMs and neural networks is that the structure of a SVM doesn't have to be pre-defined but is determined during raining.

If we use the kernel (1.1) the SVM has a structure identical to a radial basis function. If the kernel used is (1.2) we have a structure similar to a multilayer perceptron where the weights form the connections between input and hidden units form the inner product between input training vectors. The observation of this last structure inspired us to think in a multilayer perceptron where the weights from the connections between input and hidden units are input training vectors. The only parameters to be estimated will be the weights from the connections between hidden and output units. In this way we have a multilayer perceptron that is linear in the parameters.

2. Inner product networks

An initialization method for the weights of a multilayer perceptron was proposed by [Denoeux 93] that consists in the use of the input training vectors after one type of normalization. This initialization procedure was motivated by the fact that if the training vectors were normalized their inner product reflects the distance between them. Using the training vectors (after normalization) for the weights of the connections between input and hidden units, the neural networks that we propose and will refer as Inner Product networks presents hidden units with local influence as RBFs units, but with the ability to influence the entire input training space has the MLPs. On the other hand, the network proposed is linear in the parameters that we have to estimated, what can be achieve with the calculation of the pseudo-inverse matrix. These networks have an activation function given by $f(X)=\tanh\left(k\left(1-X\cdot W\right)\right)$, meaning that we have a model hyper parameter, k, that we need to define.

3. Application examples

In order to test the network proposed we used two artificial problems. The first one, a regression type problem, that we call "function detection problem", has an input training set of 100 points uniformly randomly chosen in the interval $[-5,5]$ where the desired outputs where generated by $y_i=5\sin(x_i)+0.3x_i^2+\varepsilon_i$ where $x_i=[-5,5]$, and $\varepsilon_i \cap N(0,1)$ is gaussian noise, added to the generation model function. An Inner Product network was used in order to estimate the generation model of the data.

After calculating the weights for the connections between hidden and output units with the pseudo-inverse matrix, we obtain the network performance by calculating the medium square error (MSE) in a new set, the validation set, composed by 100 other points generated the same way as the training set.
In this first study of the Inner Product networks we were interested in finding out the influence of the model hyper parameter, k, used in the activation function. We analysed the relationship between the network performance (the MSE in the validation set) and the variation of k. In table 1 we have some of the results obtained.

Value of k	**MSE in the training set**	**MSE in the validation set**
0,00001	14,330	15,208
0,0001	1,611	3,689
0,001	1,611	3,688
0,01	0,856	1,023
0,1	0,841	1,1747
1,0	0,647	213990,903

Table 1 – Variation of the network performance (MSE in the training and the validation set) in function of the value used for *k*.

It's possible to observe a regularization behaviour in the value of k. For small values of k the network doesn't have the capability to learn the structure of the data generation model (ex: $k = 0.00001$), on the other hand, for higher values of k the network presents an over learning behaviour (ex: $k = 0.00001$).The choice of the value of k becomes a critical point for the Inner Product networks. We are convicted that the optimal value, of k, is problem dependent and we suggest to use an intensive searching procedure with cross validation for choice criterion. Let us refer that for the "function detection problem" the optimal value found for k (in the sense of minimizing the MSE in the validation set) was 0.0014.

The second problem used was a classification type problem. The input data are $\Re^2$ values classified in one of two classes. The frontier separating the two classes forms a double "F". We generate 800 data points following a uniformly randomly distribution over the $[0,3]\times[0,4]$ rectangle. For the validation set, another 800 points were generated using the some procedure. Several Inner Product networks were trained for different values of k. For each case the MSE in the training and validation set were calculated as well the classification error in both sets. In table 2 we may observe the results obtained for some values of k.

Value of k	**MSE in the training set**	**Classification error in the training set**	**MSE in the validation set**	**Classification error in the validation set**
0,0001	0,378263	13,125%	0,375909	13,333%
0,001	0,319434	12,000%	0,316013	12,000%
0,01	0,266083	8,750%	0,291842	10,333%
0,1	0,208813	4,750%	0,238269	5,000%
1,0	0,118934	2,633%	0,145862	3,670%
5,0	0,062200	0,750%	0,561323	1,333%
10,0	0,041209	0,250%	0,300351	2,670%
15,0	0,026433	0,250%	17,203581	3,000%
20,0	0,019240	0,000%	99,613418	4,333%

Table 2 – Variation of the MSE and classification error in the training and validation set, for some values of *k*.

References

[Denoeux93] T. Denoeux, R. Lengellé, "Initializing Back Propagation Networks With Prototypes" Neural Networks Vol. 6, pp. 351-363, 1993.

[Vapnik95] V. Vapnik, "The Nature of Statistical Learning Theory", Springer, 1995.

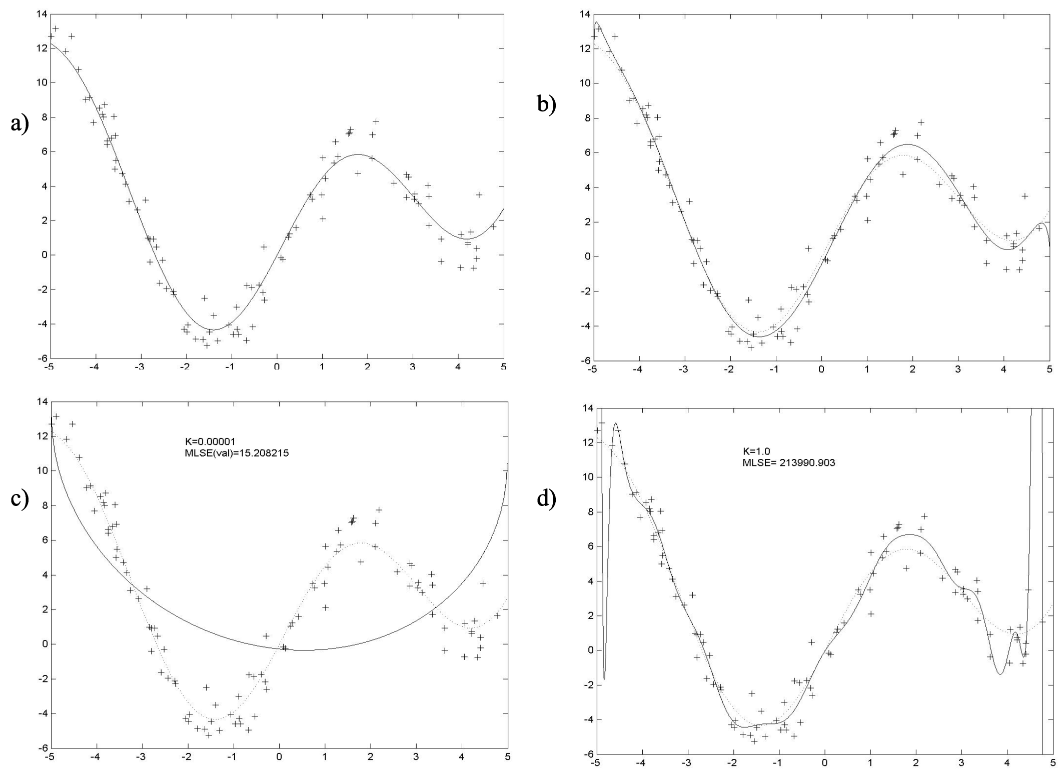


Figure 1 – a) Training data and data generating function. b), c) and d) Function estimated by an IP network whit k=0.0014, k=0.00001 and k= 1.0, respectively.

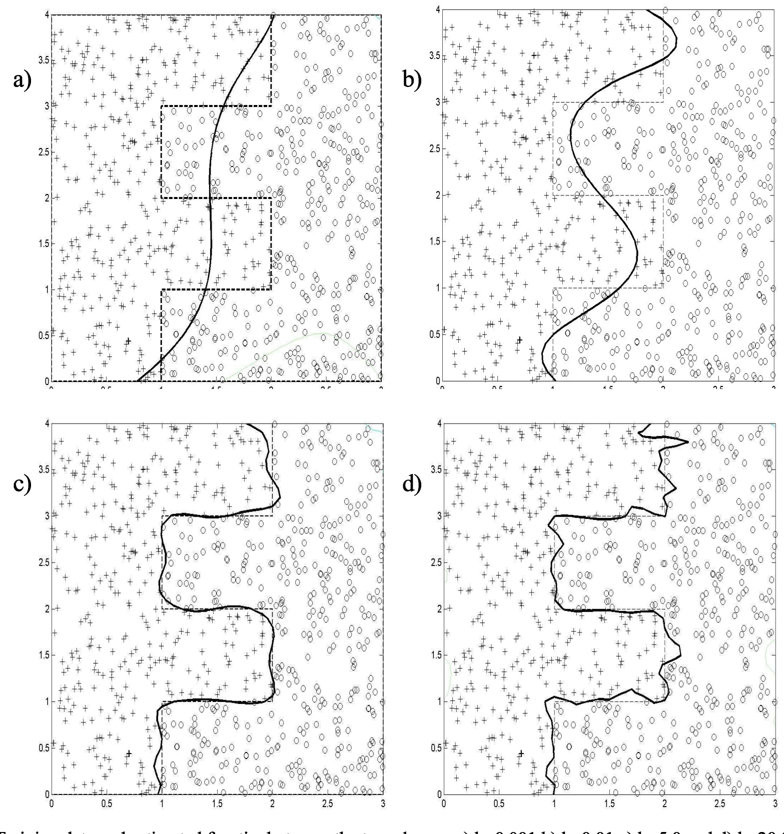


Figure 2 – Training data and estimated frontier between the two classes. a) k=0.001 b) k=0.01 c) k=5.0 and d) k=20.0