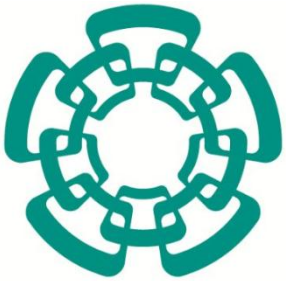

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Control Automático

Máquinas restringidas de Boltzmann para el modelado de sistemas no lineales

Tesis que presenta

M. en C. Erick Dasaev de la Rosa Montero

Para obtener el grado de

Doctor en Ciencias

En la especialidad de

Control Automático

Director de tesis

Dr. Wen Yu Liu

Ciudad de México Abril, 2018

Agradecimientos

Agradezco al cosmos y la casualidad el haberme jugado vertido las aleatoriedades que me permitieron llegar a este punto.

En inteligencia artificial hay un concepto que se titula “aprendizaje por demostración”, ese paradigma es aplicable no solo en un contexto técnico, sino que tiene su fundamentación en la forma con la cual los seres humanos aprenden. Gracias papás por ser esa guía, por demostrarme la manera de conducirme en este complejo mundo, porque su presencia creo un objetivo invisible que siempre quise alcanzar sin saberlo dado que su ejemplo paso a paso me ha conducido a este momento.

También quiero dedicarle unas palabras a mi hermano Iván que simplemente ha sido mi compañero en un sinfín de experiencias y que sin su presencia y motivación este logro habría sido imposible.

Me gustaría especialmente agradecer al Dr. Wen Yu liu. Como mi profesor y mentor, él me ha enseñado mas de lo que le podría dar crédito utilizando estas líneas. Él me ha mostrado, con su ejemplo, lo que un buen investigador (y ser humano) debería ser.

Le doy las gracias al Centro de Investigación y Estudios Avanzados por abrirme las puertas al mundo de la investigación y proporcionarme los medios materiales e intelectuales que me han permitido forjarme como investigador, así como por otorgarme el ambiente ideal en dónde pude forjar relaciones profesionales y personales que durarán toda la vida.

Finalmente agradezco al Consejo Nacional de Ciencia y Tecnología que por medio de su Programa de Posgrados de Calidad y la beca de manutención de doctorado que me fue otorgada me permitieron enfocarme enteramente en mi trabajo de tesis haciendo posible la realización de un proyecto de investigación de calidad.

Restricted Boltzmann machines for nonlinear systems modeling

by

Erick Dasaev de la Rosa Montero

Submitted to the Automatic Control Department
on April 2018, in partial fulfillment of the
requirements for the degree of
Doctor in Philosophy in Automatic Control

Abstract

Deep learning techniques have been state of the art methods during the last decade achieving remarkable results in tasks such as hand written digits classification, speech recognition and behaviour identification introducing new methods to train "deep" architectures making possible to learn high dimensional datasets. Nevertheless, despite of the fact that they have had great success in classification problems, their usage in system identification has not been deeply explored by the artificial intelligence community. In this thesis, there are explored some approaches to solve the regression problem using deep learning algorithms and it is also explained the modifications that have to be done in order to handle the analog nature of the data provided by the sampling process applied on a nonlinear system. The study is focused on the restricted Boltzmann machines (RBMs) as they constitute the building blocks in many deep learning variants. The main problem that an RBM presents is that it cannot handle continuous entries as it is designed to learn a probability distribution over a binary dataset where the allowed values for the input are only $\{0, 1\}$, this problem is handled considering a continuous range in the input domain which changes the way the probability distribution associated with the learning process is presented.

The modified RBMs are tested along with other algorithms such as randomized learning using the weights provided by an RBM in the hidden layer of an one-hidden-layer neural network and a pseudoinverse computing. Furthermore, a probability based clustering method is proposed to partition the hidden features extracted from the RBM, and then fuzzy rules are set with the introduction of a probability measurement for each fuzzy set which gives an extra degree of freedom to the model, making it more accurate.

Finally, it is also argued that even when RBMs have traditionally been used as a pre-training procedure in the machine learning literature they can also be used as models which directly learn the nonlinear system behaviour. In this case, the parameters of the RBM are trained considering the conditional distribution of the provided dataset. Moreover, it is proved the universal approximation capability of the RBMs over any binary conditional distribution. The nonlinear modeling is discussed considering two cases: binary encoding for the input for binary RBMs and continuous conditional probability transformation during the learning process.

Research Head: Wen Yu Liu
Title: Chair, Department of Automatic Control

Restricted Boltzmann machines for nonlinear systems modeling

by

Erick Dasaev de la Rosa Montero

Submitted to the Automatic Control Department
on April 2018, in partial fulfillment of the
requirements for the degree of
Doctor in Philosophy in Automatic Control

Abstract

El aprendizaje profundo es un conjunto de métodos que durante la última década han tenido buenos resultados en tareas tales como clasificación de caracteres a mano alzada, reconocimiento del habla y formación de grupos de conducta sospechosa. Estos métodos incluyen técnicas para entrenar arquitecturas "profundas" haciendo factible aprender conjuntos de entrenamiento con alta dimensionalidad. Sin embargo, a pesar de haber tenido un notable éxito en tareas de clasificación, su uso en la identificación de sistemas no ha sido explorado por la comunidad científica. En esta tesis, se analizan algunas soluciones para resolver el problema de regresión utilizando algoritmos de aprendizaje profundo, además, se explican las modificaciones que deben ser efectuadas para manejar los datos analógicos obtenidos por el proceso de muestreo aplicado en un sistema no lineal. El estudio se enfoca en las máquinas restringidas de Boltzmann (RBMs por sus siglas en inglés) debido a que son el bloque constructor de la mayoría de las variantes del aprendizaje profundo.

Las RBMs modificadas son probadas en conjunto con un aprendizaje aleatorio utilizando los pesos obtenidos de la RBM en la capa oculta de una red neuronal mientras que se calcula la solución por pseudoinversa para los pesos de la capa de salida. Además, un método de agrupamiento probabilístico se propone para obtener las características ocultas extraídas de la RBM y después se crean las reglas difusas con la introducción de una medición de probabilidad para cada conjunto difuso lo que le otorga un grado de libertad adicional al modelo haciéndolo mas preciso.

Finalmente, se argumenta que a pesar de que las RBMs han sido solo utilizadas como procesos de preentrenamiento también pueden ser utilizadas como modelos que directamente aprendan el comportamiento de los sistemas. En este caso, los parametros de una RBM son entrenados considerando la distribución condicional de los datos muestreados del sistema. También se ha probado la capacidad de aproximación universal que poseen las RBMs sobre cualquier distribución condicional binaria. El modelado es expuesto considerando dos casos: codificado binario sobre la entrada de una RBM binaria y una transformación al dominio continuo de la probabilidad condicional.

Contents

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network
BM	Boltzmann Machine
BP	Back Propagation
DA	Denoising Autoencoders
DBN	Deep Belief Network
DNN	Deep Neural Network
ICA	Independent Component Analysis
KL	Kullback-Liebler
MLP	Multilayer Perceptron
MSE	Mean Square Error
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machine
RSE	Root Square Error
SBN	Sigmoid Belief Network
SDA	Stacked Denoising Autoencoders
TK	Takagi-Sugeno
VC	Vapnik-Chervonenkis

Chapter 1

Introduction

The first precedent of the succesful usage of deep learning techniques can be found in [27], Hinton introduced deep belief networks (DBN) utilizing in each layer an RBM for the initial weights selection.

The general principle of a deep architecture behavior is to guide the training of each representation layer using unsupervised learning with a greedy-layer wise algorithm. In order to achieve this goal, each layer is trained using an unsupervised method such as autoencoders [5] and RBMs [27]. Although these techniques are generative models by themselves, they have been utilized to set the initial weights of feed forward neural networks which are discriminative models.

In the present work, there is explored the application of deep learning algorithms on the system identification task and how this application can be combined with supervised algorithms such as gradient descent, fuzzy modeling and randomized methods.

1.1 Motivation

Neural networks use a family of statistical learning algorithms to estimate or approximate functions or nonlinear systems. The most used neural model is the multilayer perceptron (MLP). From universal approximation theory [20], a single hidden layer neural network can approximate any nonlinear function to any prescribed accuracy if sufficient hidden neurons are provided. However, despite of the fact that the previous statement has been proved, a formal procedure to find the optimal parameters of a neural network does not exist. Many algorithms

have been developed to train a neural model but the gradient descent and its variants are the most widely used methods which obtained good results in the early 1980's. Nevertheless, when more layers were added to the model, it was necessary to develop a method which could cope with the new problem, this was the origin of the backpropagation algorithm which is nothing more than the application of the chain rule over each layer of neurons. After its initial success, the backpropagation algorithm faced the problem of gradient vanishing which yielded into poor results in generalization performance by deep neural networks. This was a wall that could not be overcome until Geoffrey Hinton came up with the deep learning idea [27].

In general, deep learning has two goals: a) it guides the weights to regions of minimal norm and, b) it sets the weights in zones of the parameter space where the likelihood of a global minimum is maximum [4]. The results of [23] show that the unsupervised training can drive the neural model away from the local minima for classification problems. However, deep learning methods cannot be applied to system identification directly, because the input/output values are non-binary as in classification problems. Most of deep learning techniques also use binary data, for example the conditional probability transformation in the restricted Boltzmann machines needs binary values [28].

Deep learning techniques for system identification can be regarded as a pre-training stage. Only input data are used for this unsupervised learning stage [22]. The objective of this stage is to learn the probability distribution of input data $P(x)$. This helps to decide the conditional probability distribution $P(y|x)$, which is the objective of system identification [23]. Since the unsupervised deep learning minimizes the variance and introduces bias into the input space X, the supervised learning for X and Y can be improved. This is explained by [4]: in the unsupervised learning stage, the input information is sent to hidden layers to construct useful statistical features. This mechanism improves the corresponding input/output representation. The input distribution $P(x)$ appears in the hidden units via the deep learning method.

1.2 Antecedents

Nonlinear systems identification is a problem that has not been widely addressed by the deep learning community, this is because of the differences that this problem has respecting to classification where the number of classes is well defined. In an identification framework the number of possible outputs that the system can output is infinite as its domain is continuous within a range of possible values. This contrasts with the standard deep learning methods which makes harder to design a transition between both tasks.

Although the difficulties explained previously are troublesome, the regression problem has been tackled by deep learning methods in recent years. [9] uses deep denoising autoencoders to create a model capable of representing the data, however, it is concluded that the encoding stage did not help in the identification task and the procedure has the problem of not dealing with the continuous nature of the variables that describe a physical phenomenon. The problem of handling continuous values by an RBM has been addressed by [5] who defines Gaussian hidden units showing good results for image classification, the Gaussian units assume a gaussian probability distribution over the input data which can be a extremely difficult prior to be fulfilled.

There have been also direct approximations to the problem, In [22] the normalization factor and conditional probabilities between the hidden and visible units are changed defining some domain intervals in $\Re$. After the RBM is trained using continuous values, it is used for setting the initial weights of a gradient descent optimization obtaining good results in three identification tasks. Finally, [40], argue that an RBM can be used successfully as a standalone classifier and not only as a feature extractor. To achieve this, a conditional distribution which models the class-data relationship is learned instead of the usual marginal distribution.

Other techniques that have addressed the nonlinear system identification problem are the randomized algorithms and fuzzy modeling. Randomized algorithms were initially proposed in [63] and deeply studied in [33] for single hidden layer neural networks, where the hidden weights are chosen randomly and the pseudoinverse approach (or least square method) is applied to calculate the output weights. The advantages of using the pseudoinverse are: it gives an optimal solution in the sense of least square and finds the optimal weights with minimal norm. [33] extended the above algorithms to random sampling: the hidden weights are sampled from a continuous distribution. It shows that for the single hidden layer neural

network, the optimization for the hidden layer parameters does not improve the generalization behavior significantly, while updating the output weights is more effective. Randomized algorithms have been successfully applied to nonlinear system identification in [70].

On the other hand both fuzzy models and probability theory can represent and process uncertain systems effectively [14]. Including probability theory in fuzzy modeling can improve the stochastic modeling capability [30]. In [49], the probabilistic is added into the fuzzy relation between the input space and the output space to handle the effect of random noise and stochastic uncertainties. [73] introduces probability distribution in the consequent part of the fuzzy rules improving the fuzzy modeling.

1.3 Contributions

The contributions of this thesis are explained in detail in Chapters 3, 4 and 5, it has to be pointed out that the contributions are divided in three approaches that were taken, a brief summary of these contributions is presented next:

1. In this thesis, the advantages of both deep learning and randomized algorithms are applied on nonlinear system identification. The neural model has deep structure, which increases the quantity of hidden layers and decreases the number of hidden neurons. The complexity of the neural model does not change, while the modeling capacity is improved. The restricted Boltzmann machines are modified to train the hidden weights with input data. Then, the randomized algorithm is used to train the output weights. Three benchmark examples are applied to show that the randomized algorithm with deep learning modification can improve the identification accuracy for nonlinear system identification. This constituted the first approach to deep learning in the system identification context.

2. Another approach that was taken in this thesis was to include probability theory in fuzzy modeling due to the fact that it can improve the stochastic modeling capability [30]. The third contribution of this thesis is to apply probability parameters to classical fuzzy models. Deep learning is introduced in this context using an RBM as a pretraining stage for the training samples. For the consequent part of the fuzzy rules (the output

weights), we use a randomized algorithm to train them. Finally, we use an optimization method to reach maximum probability measures in each fuzzy rule.

With this work, the advantages of deep learning, probability theory, fuzzy modeling, and randomized algorithms are exposed. We use the restricted Boltzmann machine (RBM) and probability theory to overcome some common problems in data based modeling methods. The RBM is modified such that it can be trained with continuous values. A probability based clustering method is proposed to partition the hidden features computed by the RBM, and fuzzy rules are extracted adding probability measurement. Moroever, an extreme learning machine and an optimization method are applied to train the consequent part of the fuzzy rules and the probability parameters. The proposed method is validated with two benchmark problems.

3. In Chapter 5, we first prove the universal approximation property of an RBM over binary conditional distributions, then, we address two approaches that use an RBM for system identification. First, we perform an encoding procedure over the system dataset in order to get a binary representation of the input and output vectors, with this new binary dataset we train an RBM directly maximizing the log-likelihood of the conditional probability (between input and output datasets) turning the RBM into a discriminative model. The second strategy is to modify the input domain that the RBM can handle, we just consider continuous input and output while the hidden variables remain binary. To accomplish this task, the probability distributions of the data are changed using integrals evaluated in the new domain. Finally, we calculate the new conditional probability distribution and train the model to increase it. Two numerical simulations are tested to verify our method.

1.4 Structure

The thesis is divided as follows:

- In Chapter 2, a brief introduction to deep learning is presented. It is explained which is the new paradigm that has been taken by the machine learning community and how it can be related with nonlinear systems identification theory.

- Randomized algorithms are described in Chapter 3. Their advantages over other methods are discussed and it is also explained how RBMs can be introduced and modified to fit within this framework.
- Chapter 4 describes in detail a fuzzy modeling approach where the data has been transformed using an RBM, the defuzzification process is performed adding a probability measure to each fuzzy set.
- The conditional RBM's universal approximation capability is proved in Chapter 5 along with the implementation of its training algorithm.
- Conclusions, insights and opportunities of improvement are discussed in Chapter 6.

1.5 Publications

International Journals

1. E. de la Rosa and W. Yu, Randomized Algorithms for Nonlinear System Identification with Deep Learning Modification, Information Sciences, Vol. 364, 197-212, 2016. **(Impact Factor: 3.364)**

2. E. de la Rosa and W. Yu, Data-Driven Fuzzy Modeling Using Restricted Boltzmann Machines and Probability Theory, IEEE Transaction on Systems, Man, and Cybernetics: Systems **(Impact factor: 1.598)**

3. E. de la Rosa and W. Yu, Nonlinear system identification using conditional probability, Automatica, In Review **(Impact factor: 5.451)**

4. E. de la Rosa and W. Yu, Deep Boltzmann machine for nonlinear system modelling, International Journal of Machine Learning and Cybernetics, In Review (**Impact factor: 1.699**)

International Conferences

1. E.de la Rosa, W.Yu, Nonlinear system identification using deep learning and randomized algorithms, *2015 IEEE International Conference on Information and Automation*

(ICIA 2015), Lijing, China, 274-279, 2015

2. E. de la Rosa and W. Yu, Restricted Boltzmann machine for nonlinear system modeling, *14th IEEE International Conference on Machine Learning and Applications (ICMLA15)*, Miami, USA, 2015

3. E. de la Rosa, W. Yu, X.Li , Nonlinear system modeling with deep neural networks and autoencoders algorithm, *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC16)*, Budapest, Hungary, 2157-2162, 2016

4. E. de la Rosa, W. Yu and H. Sossa, Fuzzy Modeling from Black-Box Data with Deep Learning Techniques, *14th International Symposium on Neural Networks (ISNN 2017)*, Sapporo, Japan

5. E. de la Rosa, W. Yu, X. Li, Probability based fuzzy modeling, IEEE International Conference on Systems, Man, and Cybernetics (SMC17), Banff, Canada, 1633-1638, 2017

Chapter 2

Deep learning

In this chapter we present a brief introduction to the deep learning framework and some of the identification techniques that are addressed in this thesis are presented. First, we explain which were the reasons that motivated the resurgence of deep architectures in the artificial intelligence (AI) community and how they have impacted the addressing of some problems, like face recognition, natural language processing or sentiment analysis. Some parallelisms in processes that the human brain performs in a daily basis are explained and how they can be interpreted with computer science tools.

2.1 Neural networks

In the artificial intelligence framework, researchers usually look for algorithms, methods and techniques capable of solving problems that are easily answered by a human mind. This means that scientists and engineers try to mimic the process that our brains perform in order to come up with a solution in different kinds of environments where the quantity of variables is not measurable.

2.1.1 Biological background

One of the most famous mimetizations that has been carried out by the machine learning scientists is the artificial neural network. Artificial neural networks (ANN) were originally designed to model in some small way the functionality of the biological neural networks which are a part of the human brain. Our brains contain about 10^{11} neurons. Each biological neuron

consists of a cell body, a collection of dendrites which bring electrochemical information into the cell and an axon which transmits electrochemical information out of the cell.

A neuron produces an output along its axon, it *fires* when the collective effect of its inputs reaches a certain threshold. The axon from one neuron can influence the dendrites of another neuron across junctions called synapses. Some synapses will generate a positive effect in the dendrite which encourages its neuron to fire, and others will produce a negative effect which discourages the neuron from firing. A single neuron receives inputs from perhaps 10^5 synapses and the total number of synapses in our brains may be of the order of 10^{15}. It is still not clear exactly how our brains learn and remember but it appears to be associated with the interconnections between the neurons (at the synapses).

Artificial neural nets try to model this low level functionality of the brain. This contrasts with the high level symbolic reasoning in artificial intelligence which tries to model the high level reasoning processes of the brain. When we think, we are conscious of manipulating concepts to which we attach names (or symbols) but we are not conscious of the low level electrochemical processes which are going on underneath. The argument for the neural net approach to AI is that, if we can model the low level activities correctly, the high level functionality may be produced as an emergent property.

A single artificial neuron consists of a processing element which has a number of input connections, each with an associated weight, a transfer function which determines the output, given the weighted sum of the inputs, and the output connection itself. An artificial neural network is a network of interconnected neurons, the network may be trained by adjusting the weights associated with the connections in the net to try and obtain the required outputs for given inputs from a training set. It can be seen from the above that there is an analogy between biological (human) and artificial neural nets. The analogy is summarized in Table 2.1.

Table 2.1: Analogies between an ANN and the human brain

Brain	**ANN**
Neuron	AN
Dendrites	Combining function
Cell body	Activation function
Axons	Output
Synapses	Weights

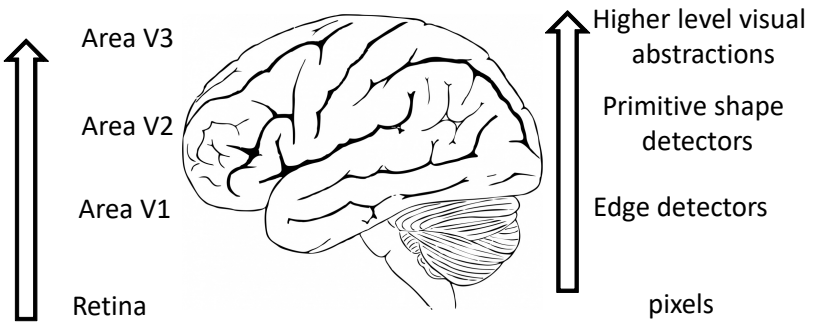

Figure 2-1: Information flow through the brain. The information existing in an image is detected using the retina. The shades, forms and shapes are interpreted using areas V1, V2 and V3

As is known, ANNs have had a huge impact achieving good results in many different areas of application [56]. However, most of the implemented ANNs are shallow (one or two layers of representation) which not entirely relates with how a brain works. Recent studies [5] show that the brain cannot be studied using a shallow architecture, the conclusions of that work are summarized in 5 points.

1. Brains have a deep architecture.

2. Humans organize their ideas hierarchically through the composition of simpler ideas.

3. Insufficiently deep architectures can be exponentially inefficient.

4. Distributed or sparse representations are necessary to achieve true learning

5. Intermediate representations allow sharing statistical strength

One example of the deep processes that take place in our minds is shown in Figure 2-1. There is seen how the data stored in an image travels through different representation layers until it is finally understood as an abstraction or idea.

Automatically, learning features of a system at multiple levels of abstraction allows a system to learn complex functions from the sensory input. This ability is very important when the amount of data and range of machine learning applications methods is growing.

According to [23], *depth of architecture* refers to the number of levels of composition of non-linear operations represented by a given function. The brain appears to process

information using different stages of abstraction and representation. For example, in our visual system the brain refines the shapes that are seen through several steps: edge detection, primitive shapes, known shapes, ..., recognized object or scene.

Under this background, neural network researchers had wanted (from 1980 and onwards) to train deep multi layer networks [6][72]. As has been explained, the backpropagation algorithm offers a solution to train multi layer models, however, it presents some problems which could not be overcome until the year 2006, these problems are summarized in three points.

1. Vanishing effect: the backpropagation signal vanishes as it moves backwards through the net, as consequence, the bottom layers remain almost untrained.

2. Local minima problem. The gradient descent algorithm cannot overcome the local minima regions of the cost function.

3. Shadowing effect. Having too many model parameters can produce a continuous updating through the training epochs which does not converge to any solution

Because of these difficulties, the study of deep architectures was abandoned and researchers focused in only two or three layers of representation. Nevertheless, Hinton introduced in 2006 the concept of Deep Belief Networks (DBNs) [27] which is an algorithm that greedily trains each layer at a time utilizing a restricted Boltzmann machine (RBM). In the next section we explain how deep learning helps to overcome the historical issues that deep architectures have had achieving state of the art performance in various tasks.

2.1.2 Unsupervised learning and deep learning

Deep learning has achieved general success despite of the serious challenge of training models with many layers of adaptive parameters. In virtually all deep learning instances, the goal is to minimize an objective function which is a highly non-convex function of the model parameters with the potential of many distinct local minima in the parameter space. It has been shown that for deep architectures, the classic training schemes which rely on random parameter initialization tend to place the parameters in regions that settle down in regions of the hyperdimensional parameter space that give poor generalization performance as has been frequently observed.

As has been said, the breakthrough to effective training strategies for deep architectures came with the algorithms for training DBNs and stacked denoising autoencoders [5] which are all based on a similar approach: greedy layer-wise unsupervised pre-training followed by supervised fine-tuning.

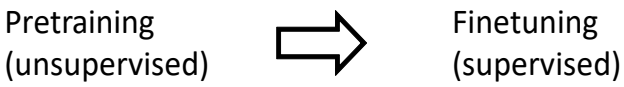

Figure 2-2: The deep learning approach consists of two stages: a pretraining procedure where the parameters are initialized and a finetuning method which uses a supervised criteria to find the best parameters

Each layer is pretrained with an unsupervised learning algorithm learning, i. e., an encoding procedure that captures the main variations of the layer's input. This unsupervised pre-training sets the stage for a final training phase where the deep architecture is fine-tuned with respect to a supervised training criterion with gradient-based optimization. It is important to mentione that despite of the fact that the pretraining impact has been measured with impressive results, the mechanisms underlying its success are not understood. There have been some claims that try to explain why the pretraining stage works, the most important among them are:

- Unsupervised pretraining initializes the model to a point that somehow renders the optimization process more effective achieving a lower minimum of the empirical cost function.

- Pretraining is an unusual form of regularization which minimizes variance and introduce bias towards a configuration of the parameter space that is useful for supervised learning.

The second perspective places unsupervised pretraining among semisupervised algorithms, however, it is unique because it acts by defining a particular initialization point for standard supervised training rather than modifying the supervised objective function or imposing constraints on the parameters throughout training. It has been suggested [9] that in highly non-convex situations as training a deep structure, defining a particular initialization point

imposes constraints on the parameters because it specifies which minima of the cost function are allowed.

Standard training of deep models using gradient descent is difficult. It has to be stated why it is difficult and which are the tasks that a successful algorithm has to accomplish. The main problem is that the model parameters have strong dependencies between them which carry several difficulties during training. This dependency is stronger between parameters across layers as parameters which belong to the same layer are independent in the sense that in a direct calculation their impact in the final result is independent. This problem has to be addressed considering the next two aspects for every new algorithm that we want to use:

- Modify the lower layers in order to provide good data to the upper layers .
- Modify the upper layers to take advantage of the data delivered by the lower layers.

The second problem is easily solved using any supervised learning approach. However, it is not well understood how the first problem can be addressed, moreover, a particular difficulty arises when both sets of layers must be learned at the same time as the gradient of the objective function is limited to a local measure. Furthermore, the training error cannot show the effectiveness of the lower layers training because the upper layers can overfit the training set if they are large enough.

Now, what happens with the online gradient descent implementation? This procedure defines a trajectory in the parameter space that eventually converges (i. e., it reaches a point where the error does not improve anymore); it has been argued [23] that small perturbations on such trajectory have a bigger effect if they are applied early on.

Once the training is initialized it rapidly settles into a basin which defines the local minimum that the algorithm reaches. Early on, small perturbations allow the model parameters to switch from a basin to a nearby one, whereas later on, it is unlikely to escape from the basin attraction. In this sense, unsupervised pretraining interacts with the optimization process and when the number of training examples becomes large, its positive effect is seen not only on generalization error but also on training error. An scheme of this general idea is seen in Figure 2-3.

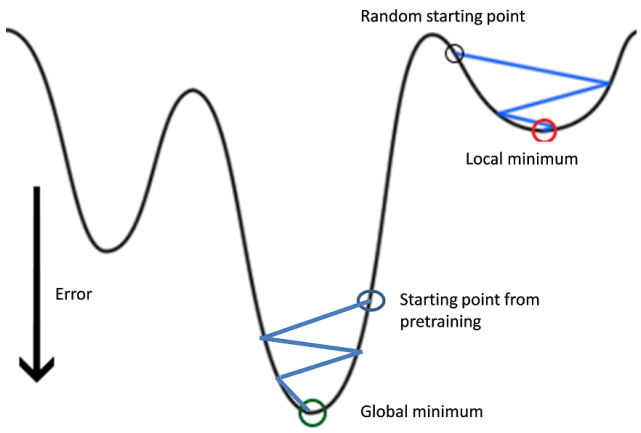

Figure 2-3: The effect of the pretraining stage. If the initial parameters are chosen randomly they most likely will settle down in a local minima. The pretraining stage helps to avoid this.

2.2 Probabilistic models

In any system identification framework, the goal is to find an useful approximation $\widehat{f}(\mathbf{x})$ of the function $f(\mathbf{x})$ that underlies the predictive relationship between inputs and outputs. In the theoretical setting of probability, it has been shown that optimizing a squared error loss leads to the regression function $f(x) = \mathrm{E}(Y|X = \mathbf{x})$. In this case, the modeling object is to find the proper probability distribution that can predict the associated system observations.

2.2.1 Statistical learning

Suppose that in fact, we have a dataset that arose from a statistical model:

$$Y = f(X) + \varepsilon \tag{2.1}$$

where the random error ε has $\mathrm{E}(\varepsilon) = 0$ and is independent of X. Note that for this model, $f(\mathbf{x}) = \mathrm{E}(Y|X = \mathbf{x})$, and in fact the conditional distribution $P(Y|X)$ depends on X only through the conditional mean $f(\mathbf{x})$.

The additive error model is a useful approximation to the real behaviour of nature. For most systems the input–output pairs (X, Y) will not have a deterministic relationship $Y = f(X)$. Generally, there will be other unmeasured variables that also contribute to Y , including measurement error. The additive model assumes that we can capture all these de-

partures from a deterministic relationship via the error ε. For some problems a deterministic relationship does not hold. Many of the classification problems studied in machine learning are of this form, where the learning comes from the training data that consists of examples $\{\mathbf{x}(k), g(k)\}$. Here the function is deterministic, and the randomness enters through the $\mathbf{x}$ location of the training points.

We will see that this problem can be handled by techniques appropriate for error-based models. The assumption in (2.1) that the errors are independent and identically distributed is not strictly necessary, but with such a model it becomes natural to use least squares as a data criterion for model estimation. Simple modifications can be made to avoid the independence assumption; for example, we can have $\mathrm{Var}(Y|X = \mathbf{x}) = \sigma(\mathbf{x})$, and now both the mean and variance depend on X. In general, the conditional distribution $P(Y|X)$ can depend on X in complicated ways, but the additive error model precludes these.

Additive error models are typically not used for qualitative outputs G; in this case the target function $p(X)$ is the conditional density $P(G|X)$, and this is modeled directly. For example, for two-class data, it is often reasonable to assume that the data arise from independent binary trials, with the probability of one particular outcome being $p(X)$, and the other $1 - p(X)$. Thus if Y is the $0 - 1$ coded version of G, then $\mathrm{E}(Y|X = \mathbf{x}) = p(\mathbf{x})$, but the variance depends on $\mathbf{x}$ as well: $\mathrm{Var}(Y|X = \mathbf{x}) = p(\mathbf{x})[1 - p(\mathbf{x})]$.

2.2.2 Supervised Learning and function approximation

Suppose for simplicity that the errors are additive and that the model $Y = f(X) + \varepsilon$ is a reasonable assumption. Supervised learning attempts to learn f by example through a teacher. One observes the system under study, both the inputs and outputs, and assembles a training set of observations $T = (\mathbf{x}(k), y(k))$, $i = 1, ..., N$. The observed input values of the system $\mathbf{x}(k)$ are also fed into an artificial system, known as a learning algorithm, which also produces outputs $\widehat{f}(\mathbf{x}(k))$ in response to the inputs.

The learning algorithm has the property that it can modify its input/output relationship $\widehat{f}$ in response to differences $y(k) - \widehat{f}(\mathbf{x}(k))$ between the original and generated outputs. This process is known as learning by example. Upon completion of the learning process the hope is that the artificial and real outputs will be close enough to be useful for all sets of inputs likely to be encountered in practice.

The learning paradigm of the previous section has been the motivation for research into the supervised learning problem in the fields of machine learning (with analogies to human reasoning) and neural networks (with biological analogies to the brain). The approach taken in applied mathematics and statistics has been from the perspective of function approximation and estimation. Here the data pairs $\{\mathbf{x}(k), y(k)\}$ are viewed as points in a $(p+1)$-dimensional Euclidean space. The function $f(x)$ has domain equal to the p-dimensional input subspace, and is related to the data via a model such as $y(k) = f(\mathbf{x}(k)) + \varepsilon(k)$.

2.3 Data-driven identification

In this section we explain some of the algorithms that have been used along deep learning to identify nonlinear systems. We begin giving a brief description of the gradient descent algorithm which is the most known algorithm that minimizes cost functions, then, we introduce the multilayer perceptron which is the most famous feed forward neural network. Finally, the reader is introduced to the fuzzy logic framework where a heuristic paradigm commands.

2.3.1 Gradient descent algorithm

A very common problem that arises in the vast majority of machine learning problems is the minimization of a cost function which is subject to some parameters to be tuned by a training algorithm. The most used method to minimize the cost function is the *gradient descent algorithm* or *steepest descent method* which is explained in this section.

The problem we are interested in solving is:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & x \in \Re^n, \end{aligned}$$

where $f(x)$ is differentiable. If $x = \overline{x}$ is a given point, $f(x)$ can be approximated by its linear expansion

$$f(\overline{x} + d) \approx f(\overline{x}) + \frac{\partial f(\overline{x})^{\mathrm{T}}}{\partial x} d$$

if d is small. Now notice that if the approximation in the above expression is good, then

we want to choose d so that the inner product $\frac{\partial f(\overline{x})}{\partial x}^{\mathrm{T}} d$ is as small as possible. Let us normalize d so that $\|d\| = 1$. Then among all directions d with norm $\|d\| = 1$, the direction

$$\widetilde{d} = \frac{-\frac{\partial f(\overline{x})}{\partial x}}{\left\| \frac{\partial f(\overline{x})}{\partial x} \right\|}$$

makes the smallest inner product with the gradient $\frac{\partial f(\overline{x})}{\partial x}$. This fact follows from the following inequalities:

$$\frac{\partial f(\overline{x})}{\partial x}^{\mathrm{T}} d \geq -\left\| \frac{\partial f(\overline{x})}{\partial x} \right\| \|d\| = \frac{\partial f(\overline{x})}{\partial x}^{\mathrm{T}} \left(\frac{-\frac{\partial f(\overline{x})}{\partial x}}{\left\| \frac{\partial f(\overline{x})}{\partial x} \right\|} \right) = -\frac{\partial f(\overline{x})}{\partial x}^{\mathrm{T}} \widetilde{d}$$

For this reason the un-normalized direction:

$$\overline{d} = -\frac{\partial f(\overline{x})}{\partial x}$$

is called the *direction of steepest descent* at the point $\overline{x}$.

Note that $\overline{d} = -\frac{\partial f(\overline{x})}{\partial x}$ is a descent direction as long as $\frac{\partial f(\overline{x})}{\partial x} \neq 0$. To see this, simple observe that $\overline{d}^{\mathrm{T}} \frac{\partial f(\overline{x})}{\partial x} = -\left(\frac{\partial f(\overline{x})}{\partial x} \right)^{\mathrm{T}} \frac{\partial f(\overline{x})}{\partial x} < 0$ as long as $\frac{\partial f(\overline{x})}{\partial x} \neq 0$.

Observing this behavior, we have as consequence Algorithm 1, called the steepest descent algorithm

Algorithm 1

1. Given $x(0)$, set $k := 0$
2. $d(k) := -\frac{\partial f(x(k))}{\partial x}$. If $d(k) = 0$, then stop.
3. Solve $\min_\alpha f(x(k) + \alpha d(k))$ for the stepsize $\alpha(k)$, perhaps chosen by an exact or inexact linesearch.
4. Update with $x(k+1) = x(k) + \alpha(k) d(k)$, $k = k + 1$. Return to Step 2.

Note from Step 3 and the fact that $d(k) = -\frac{\partial f(x(k))}{\partial x}$ is a descent direction, it follows that $f(x(k+1)) < f(x(k))$. A graphical representation of this process is shown in Figure 2-4, it is shown how the values of x are changing towards a local minimum where eventually the updating process will cease to work.

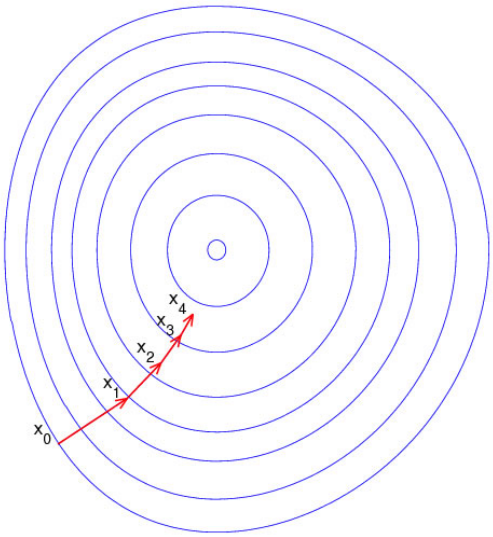

Figure 2-4: Gradient descent algorithm

The original error back-propagation algorithm implements a steepest descent method. In each iteration, the weights of the multilayer perceptron are updated by a fixed percentage in the negative direction. In literature, the gradient descent techniques can be summarized in three main variants of the classic algorithm, these variants are explained next.

Batch gradient descent

Consider a training set $D = \{x(k)\}_{k=1}^{n}$, both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$J(\Lambda) = \frac{1}{n}\sum_{k=1}^{n} J(k, \Lambda) \tag{2.2}$$

where the parameter Λ which minimizes $J(\Lambda)$ is to be estimated. Each summand function $J(k, \Lambda)$ is typically associated with the k-th observation in the dataset (used for training). When used to minimize the above function, a standard (or batch) gradient descent method would perform the following iterations:

$$\Lambda = \Lambda - \eta\frac{\partial J(\Lambda)}{\partial \Lambda} = \Lambda - \eta\frac{1}{n}\sum_{k=1}^{n}\frac{\partial J(k, \Lambda)}{\partial \Lambda} \tag{2.3}$$

where η is the learning rate. The most important feature of this variant is that the up-

dating process is only done once the summation of the contributions of each training example has been calculated. In many cases, evaluating the sum-gradient may require expensive evaluations of the gradients from all summand functions. When the training set is enormous and no simple formulas exist, evaluating the sums of gradients becomes very expensive, because evaluating the gradient requires evaluating all the summand functions' gradients. To economize on the computational cost at every iteration, stochastic gradient descent samples a subset of summand functions at every step. This is very effective in the case of large-scale machine learning problems.

Stochastic gradient descent

In stochastic (or "on-line") gradient descent, the true gradient of $J(\Lambda)$ is approximated by a gradient at a single example:

$$\Lambda = \Lambda - \eta \frac{\partial J(k, \Lambda)}{\partial \Lambda} \qquad \text{for } k = 1, .., n \tag{2.4}$$

As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive learning rate so that the algorithm converges.

Minibatch gradient descent

A compromise between computing the true gradient and the gradient at a single example, is to compute the gradient against more than one training example (called a *mini-batch*) at each step. This can perform significantly better than true stochastic gradient descent because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step uses more training examples.

Consider then that the training set D is divided in q disjoint subsets $\{H_1, H_2, ..., H_q\}$. Each of this subsets is a minibatch, the learning rule is formulated as:

$$\Lambda = \Lambda - \eta \frac{1}{n_j} \sum_{k|x(k) \in H_j} \frac{\partial J(k, \Lambda)}{\partial \Lambda} \qquad \text{for } j = 1, ..., q \tag{2.5}$$

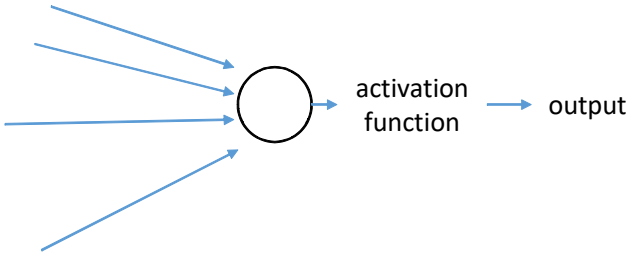

Figure 2-5: Model of a perceptron

where n_j is the number of elements in the minibatch H_j.

2.3.2 Feed Forward Neural Networks

The most common learning mechanism associated with all artificial neural networks is, by far, the supervised paradigm. Multilayer perceptrons (MLP) are the most widely known type of ANNs. It has been shown that they constitute *universal approximators* [20], both with one hidden layer and with two hidden layers. Before describing MLPs let us describe the single perceptron.

The Perceptron

Perceptrons were first introduced by Frank Rosenblatt, working at Cornell Aeronautical Labs, intended to be computational models of the retina. The basic model is shown in Figure 2-5.

The typical application of Rosenblat was to activate an appropriate response unit for a given input pattern or a class of patterns. For this reason, the activation function is a threshold function. The inputs, outputs and training patterns were binary values (0 or 1). The basic rule is to alter the value of the weights when an error exists between the network output and the desired output. The heuristic learning rule is as follows:

Algorithm 2

1. If the input is 1 and should be 1, or if the output is 0 and should be 0, do nothing;
2. If the output is 0 and should be 1, increment all the weights in all active lines;
3. If the output is 1 and should be 0, decrement the weights in all active lines.

Considering a perceptron with just 1 output, the weight vector, W, is updated as:

$$W(k+1) = W(k) + \Delta W \tag{2.6}$$

where ΔW is the change made to the weight vector, as:

$$\Delta w_i = \alpha(\widehat{y}(k) - y(k))x_i(k) \tag{2.7}$$

In (2.7), α is the learning rate, $\widehat{y}(k)$ and $y(k)$ are the desired and actual output, respectively, at time k, $x_i(k)$ is the $i-$th element of the input vector. Some variations have been made to this simple perceptron model: First, some models do not employ a bias; the inputs to the net may be real valued, bipolar $(+1, -1)$, as well as binary and the outputs may be bipolar.

Multilayer perceptrons

The error back-propagation (BP) algorithm is the best known learning algorithm for performing the tuning of the MLP parameters. In fact, MLPs and the BP algorithm are so intimately related that it is usual to find in the literature that this type of artificial neural network is referred to as *back-propagation neural network*.

The MLP can be explained as the composition of nonlinear functions applied to inner products. Its mathematical model is given by (2.8).

$$\widehat{y}\,(k) = \beta\phi_p\left\{W_p\phi_{p-1}\left[\ldots W_3\phi_2\left\{W_2\phi_1\left[W_1x\,(k) + b_1\right] + b_2\right\} + b_3 \ldots + b_{p-1}\right] + b_p\right\} \tag{2.8}$$

where $\widehat{y}\,(k) \in \Re^m$ is the output of the neural model, $W_1 \in \Re^{l_1 \times n}$, $b_1 \in \Re^{l_1}$, $W_2 \in \Re^{l_2 \times l_1}$, $b_2 \in \Re^{l_2}$, $W_p \in \Re^{l_p \times l_{p-1}}$, $b_p \in \Re^{l_p}$, p is the number of hidden layers, l_i $(i = 1 \cdots p)$ are the node numbers in each layer, $\phi_i \in \Re^{l_i}$ $(i = 1 \cdots p)$ are active vector functions, $\beta = \left[\beta_1 \cdots \beta_{l_p}\right]$, $\beta \in \Re^{m \times l_p}$ is the weight matrix of the output layer. The active functions are in sigmoid form,

$$\phi_i\,(\omega_j) = a_i / \left(1 + e^{-b_i^{\mathrm{T}}\omega_j}\right) - c_i$$

where $i = 1 \cdots p$, $j = 1 \cdots l_i$, a_i, b_i, and c_i are prior defined positive constants, ω_j are the input variables to the sigmoid functions. A flow chart of the MLP is seen in Figure 2-6.

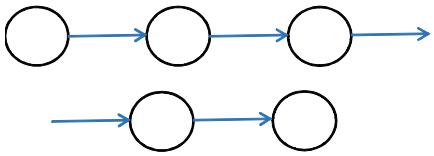

Figure 2-6: Multilayer Perceptron

Let Λ represents any of the parameters that define the MLP, the learning procedure follows the same rules of gradient descent. The learning rule can be given by (2.3), (2.4) or (2.5) depending on the type of approach we are taking. It is important to note that the very famous backpropagation algorithm is just an application of the chain rule that allows to calculate the term $\frac{\partial J(k,\Lambda)}{\partial \Lambda}$ when Λ is a parameter that does not belong to the output layer. In this thesis we compute the gradient using a numerical package which enables us to skip the usage of the backpropagation method.

2.3.3 Randomized algorithms

Randomized algorithms are feedforward neural networks for classification or regression with a single layer of hidden nodes, where the weights connecting inputs to hidden nodes are randomly assigned and never updated. The weights between hidden nodes and outputs are learned in a single step, which essentially amounts to learning a linear model.

According to their creators, these models are able to produce good generalization performance and learn thousands of times faster than networks trained using backpropagation [32].

The simplest randomized training algorithm learns a model of the form

$$\widehat{y} = \beta\phi(Wx) \tag{2.9}$$

where W is the matrix of input-to-hidden-layer weights, ϕ is some activation function, and β is the matrix of hidden-to-output-layer weights. Algorithm 3 explains how the training is done:

Algorithm 3

1. Fill W with Gaussian random noise;

2. Estimate β by least-squares fit to a matrix of response variables Y, computed using the pseudoinverse $(\cdot)^+$+ given a design matrix X:

$$W = \phi(WX)^+ Y$$

It is seen the incredible simplicity of the model which is the main reason that has made this model so popular in recent years. It is important to point out that this model has received numerous critics that emphasize the lack of proper training in the hidden layer as a weakness that decreases the generalization capabilities of the net. However, it has been shown that the generalization is not compromised if the random selection of the hidden weights is performed over certain classes of probability distributions [31]. In Chapter 3 it is proposed a method that initializes the hidden weights using a deep learning technique: the restricted Boltzmann machine.

2.3.4 An historical view of the perceptron and randomized algorithms

Rosenblat [60] stated that a MLP can enable artificial systems to perform human-like activities such as speaking, walking, writing and even being aware of its own existence. However, it was shown [53] that a perceptron without hidden layers could not even handle the modeling of the simple XOR function, this fact made researchers give up in the field of machine learning but as it is going be explained in this thesis, the failure exposed in [53] can be avoided if a new architecture is introduced.

The XOR counterexample given in [53] used a feedforward network with input and output layers but without hidden layers, this model can be empirically understood as *brain* which has input layers or *sensors* (eyes, nose, ears,...) and output layers or *actuators* (muscles, bones, ...) but it lacks of *neurons.* It is easily seen that this brain is an empty shell and has no learning or cognition capabilities at all. Then, it is clear that a hidden layer is necessary to provide a true learning system, this realization had as consequence that thousands of researches started to look for learning algorithms capable of tuning a new set of hidden layers. Such human effort did not have good results despite of the huge number of scientists that got involved, eventually it reached a point where some researchers began to think about

ways to avoid the hidden layer training by assuming that it was not necessary. Finally, such beliefs and philosophy in both machine learning and biological learning resulted in techniques referred as *randomized algorithms*, in this approach the problem of training a MLP is addressed by considering that the existence of hidden layers is necessary to achieve good results but their tuning is not.

Randomized algorithms represent a suite of machine learning techniques (including single hidden feedforward networks and multilayer feedforward networks) in which hidden neurons do not need to be tuned with the consideration of neural networks generalization theory. It has been argued that randomized algorithms reflect the true nature of some biological learning mechanisms as it has been found that the stimulus propagation in some areas of the brain cortex occurs randomly. Their universal approximation capabilities (proved for a network in which a hidden node may be a subnetwork of several nodes with almost nonlinear piecewise continuous neurons) was shown in [31]. Their concrete biological evidence subsequently appears in [66].

The target of randomized algorithms is not only the single layer feedforward neural networks but also the generalized multilayer feedforward neural networks in which a node may be a subnetwork consisting of other hidden nodes. The randomized algorithms framework also covers wide types of neural networks including but not limited to sigmoid networks and radial basis functions, it also aims to implement the five fundamental operations of learning in an homogenous architecture, these operations are:

1. Compression
2. Feature learning
3. Clustering
4. Regression (modeling)
5. Classification

Thus, from this point of view, the coexistence of globally structured architectures and locally random hidden neurons happens to have fundamental learning capabilities in the five tasks exposed above. This may have addressed John von Neumann's puzzle. Biological learning mechanisms are sophisticated, and it is believed in [31] that learning without

tuning hidden neurons is one of the fundamental biological learning mechanisms in many modules of learning systems. Furthermore, random hidden neurons and random wiring are only two specific implementations of such *learning without tuning hidden neurons* learning mechanisms.

2.4 Fuzzy models

Th idea of a *fuzzy logic* was introduced by L. A. Zadeh in 1965. This concept allows imprecise and qualitative information to be expressed and used in an exact way. It also implied a generalization of the concept of *set* which was included in the more general term of *fuzzy set*. This new concept offered the flexibility to be able to contain with uncertainty objects and ideas. A mathematical model which in some ways 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 predicates, such as:

IF *heating is fast* THEN *temperature increase is fast*

This rule defines in a rather qualitative way the relationship between the heating an 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 Mandami (or linguistic) and the Takagi-Sugeno model.

2.4.1 Mamdani model

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

$$\mathrm{R}^j\text{: IF } x \text{ is } A^j \text{ THEN } y \text{ is } B^j, \qquad j = 1, 2, .., K \tag{2.10}$$

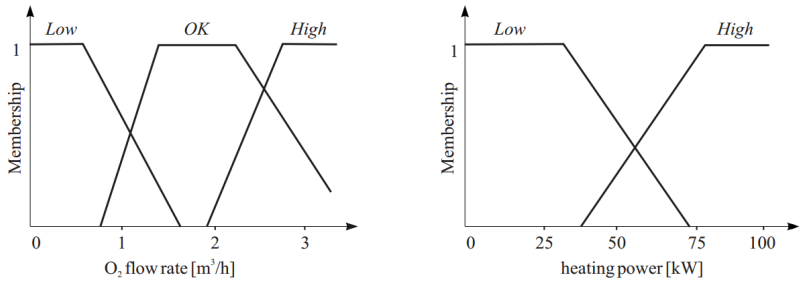

Figure 2-7: Membership funtions for the Mamdani model

Here A^j and B^j are the antecedent and consequent linguistic terms (such as *small*, *large*, etc.), represented by fuzzy sets, and K is the number of rules that exist in the model. The linguistic fuzzy model is useful for representing qualitative knowledge, this is illustrated in the following example.

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

$$\begin{array}{llllll} \mathrm{R}^1\text{:} & \text{IF} & O_2 \text{ flow rate} & \text{is } \textit{Low} & \text{THEN} & \text{heating power is } \textit{Low} \\ \mathrm{R}^2\text{:} & \text{IF} & O_2 \text{ flow rate} & \text{is } \textit{OK} & \text{THEN} & \text{heating power is } \textit{OK} \\ \mathrm{R}^3\text{:} & \text{IF} & O_2 \text{ flow rate} & \text{is } \textit{High} & \text{THEN} & \text{heating power is } \textit{Low} \end{array}$$

The meaning of the linguistic terms {*Low, OK, High*} and {*Low, High*} is defined by membership functions such as the ones depicted in Figure 2-7. 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 arbitrarily).

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.

2.4.2 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:

$$\mathrm{R}^j\text{: IF } x \text{ is } A^j \text{ THEN } y^j = a_j^{\mathrm{T}} x + b_j, \qquad j = 1, 2, .., K \tag{2.11}$$

where a_j is the consequent parameter vector and b_j 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:

$$y = \frac{\sum_{j=1}^{K} \beta^j(x) y^j}{\sum_{j=1}^{K} \beta^j(x)} = \frac{\sum_{j=1}^{K} \beta^j(x)(a_j^{\mathrm{T}} x + b_j)}{\sum_{j=1}^{K} \beta^j(x)}$$

where $\beta^j(x)$ is the degree of fulfillment of the j-th rule. For the rule (2.11), $\beta^j(x) = \mu_{A^j}(x)$, 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 $a_j(x)$ 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. This is illustrated by the next example:

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 Figure 2-8.

The corresponding rules are given next:

$$\begin{aligned}
\mathrm{R}^1\text{: IF } u \text{ is } \textit{Negative} \text{ THEN } y^1 &= a^1 x - b^1 \\
\mathrm{R}^2\text{: IF } u \text{ is } \textit{Zero} \text{ THEN } y^2 &= a^2 x - b^2 \\
\mathrm{R}^3\text{: IF } u \text{ is } \textit{Positive} \text{ THEN } y^3 &= a^3 x - b^3
\end{aligned}$$

As the consequent parameters are first-order polynomials in the input variables, the model

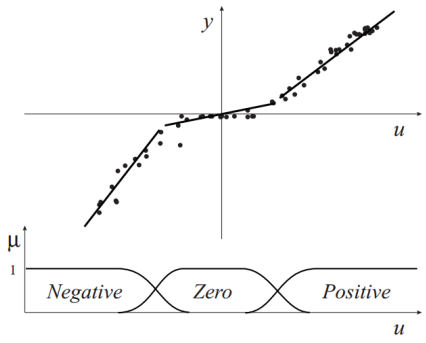

Figure 2-8: A Takagi-Sugeno fuzzy model as a piece-wise linear approximation of a nonlinear system

(2.11) 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.

$$\mathrm{R}^j\text{: IF } x \text{ is } A^j \text{ THEN } y^j \ = b_j, \qquad j = 1, 2, .., K \tag{2.12}$$

2.4.3 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* (conjuction), *or* (disjunction) and *not* (complement). In fuzzy set theory, several families of operators have been introduced for these logical connectives. Table 2.2 shows the two most common ones.

Table 2.2: 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:

$$\mathrm{R}^j\text{: IF } x_1 \text{ is } A_1^j \text{ and } x_2 \text{ is } A_2^j \text{ and } \cdots \ x_m\,(k) \ \text{ is } A_m^j \text{ THEN } y^j \ = a_j^{\mathrm{T}} x + b_j$$

with the degree of fulfillment

$$\beta^j(x) = \min\left(\mu_{A_1^j}(x_1), \mu_{A_2^j}(x_2), ..., \mu_{A_m^j}(x_m)\right)$$

or

$$\beta^j(x) = \mu_{A_1^j}(x_1) \cdot \mu_{A_2^j}(x_2), ..., \mu_{A_m^j}(x_m)$$

for the minimum and product conjunction operators, respectively. The complete set of rules divides 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 set.

2.5 Kullback–Leibler divergence

The Kullback–Leibler divergence (also called relative entropy) is a measure of how one probability distribution diverges from a second, expected probability distribution. In the simple case, a Kullback–Leibler divergence of 0 indicates that we can expect similar, if not the same, behavior of two different distributions, while a Kullback–Leibler divergence of 1 indicates that the two distributions behave in such a different manner that the expectation given the first distribution approaches zero.

Definition

Consider two discrete probability distributions P and Q, the Kullback-Leibler divergence from Q to P is defined to be

$$D_{KL}(P||Q) = -\sum_i P(i) \log \frac{Q(i)}{P(i)} \tag{2.13}$$

which is equivalent to

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{2.14}$$

In other words, it is the expectation of the logarithmic difference between the probabilities

P and Q, where the expectation is taken using the probabilities P. the Kullback-Leibler divergence is defined only if for all i, $Q(i) = 0$ implies $P(i) = 0$ (absolute continuity). Whenever $P(i)$ is zero the contribution of the i-th term is interpreted as zero because $\lim_{x \to 0} \log(x) = 0$.

For distributions P and Q of a continuous random variable, the Kullback-Leibler divergence is defined to be the integral:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx, \tag{2.15}$$

where p and q denote the densities of P and Q.

More generally, if P and Q are probability measures over a set X, and P is absolutely continuous with respect to Q, then the Kullback-Leibler divergence from Q to P is defined as

$$D_{KL}(P||Q) = \int_{X} \log \frac{dP}{dQ} dP, \tag{2.16}$$

where $\frac{dP}{dQ}$ is the Radon-Nikodym derivative of P with respect to Q, and provided the expression on the right-hand side exists. Equivalently, this can be written as

$$D_{KL}(P||Q) = \int_{X} \log \left(\frac{dP}{dQ} \right) \frac{dP}{dQ} dQ, \tag{2.17}$$

which is the entropy of P relative to Q. Continuing in this case, if μ is any measure on X for which $p = \frac{dP}{d\mu}$ and $q = \frac{dQ}{d\mu}$ exist (meaning that p and q are absolutely continuous with respect to μ) then the Kullback-Leibler divergence from Q to P is given as

$$D_{KL}(P||Q) = \int_{X} p \log \frac{p}{q} d\mu, \tag{2.18}$$

The logarithms in these formula are taken to base 2 if information is measured in units of bits, or to base e if information is measured in nats. Most formulas involving the Kullback-Leibler divergence hold regardless of the base of the logarithm.

Motivation

In information theory, the Kraft–McMillan theorem establishes that any directly decodable coding scheme for coding a message to identify one value x_i out of a set of possibilities X can be seen as representing an implicit probability distribution $q(x_i) = 2^{-l_i}$ over X, where l_i is the length of the code for x_i in bits. Therefore, the Kullback–Leibler divergence can be interpreted as the expected extra message-length per datum that must be communicated if a code that is optimal for a given (wrong) distribution Q is used, compared to using a code based on the true distribution P.

$$\begin{aligned} D_{KL}(P||Q) &= -\sum_x p(x) \log q(x) + \sum_x p(x) \log p(x) \\ &= H(P,Q) - H(P) \end{aligned} \tag{2.19}$$

where $H(P,Q)$ is the cross entropy of P and Q, and $H(P)$ is the entropy of P.

Properties

1. The Kullback-Leibler divergence is always non-negative, $D_{KL}(P||Q) \geq 0$, a result known as Gibb's inequality, $D_{KL}(P||Q)$ zero if and only if $P = Q$ almost everywhere.

2. The Kullback-Leibler divergence remains well-defined for continuous distributions, and furthermore is invariant under parameter transformations. For example, if a transformation is made from variable x to variable $y(x)$, then, since $P(x)dx = P(y)dy$ and $Q(x)dx = Q(y)dy$ the Kullback-Leibler divergence may be written:

$$\begin{aligned} D_{KL}(P||Q) &= \int_{x_a}^{x_b} P(x) \log \left(\frac{P(x)}{Q(x)} \right) dx \\ &= \int_{y_a}^{y_b} P(y) \log \left(\frac{P(y)dy/dx}{Q(y)dy/dx} \right) dy \\ &= \int_{y_a}^{y_b} P(y) \log \left(\frac{P(y)}{Q(y)} \right) dy \end{aligned} \tag{2.20}$$

where $y_a = y(x_a)$ and $y_b = y(x_b)$.

The Kullback-Leibler divergence is additive for independent distributions in much the same way as Shannon entropy. If P_1, P_2 are independent distributions, with the joint distribution $P(x, y) = P_1(x)P_2(y)$, and Q, Q_1, Q_2 likewise, then

$$D_{KL}(P||Q) = D_{KL}(P_1||Q_1) + D_{KL}(P_2||Q_2) \tag{2.21}$$

The Kullback-Leibler divergence $D_{KL}(P||Q)$ is convex in the pair of probability mass functions (p, q), i.e. if (p_1, q_1) and (p_2, q_2) are two pairs of probability mass functions, then

$$\begin{aligned} & D_{KL}(\lambda p_1 + (1-\lambda)p_2||\lambda q_1 + (1-\lambda)q_2) \\ \leq \; & \lambda D_{KL}(p_1||q_1) + (1-\lambda)D_{KL}(p_2||q_2) \end{aligned} \tag{2.22}$$

for $0 \leq \lambda \leq 1$.

2.6 State of the art

Several works have been done regarding deep learning, randomized algorithms and probabilistic methods, a brief summary is presented in the following.

2.6.1 Randomized modeling, local minima and deep learning

As has been said, randomized algorithms have been initially proposed in [63] and deeply studied in [33] for single hidden layer neural networks, where the hidden weights are chosen randomly and the pseudoinverse approach (or least square method) is applied to calculate the output weights. The advantages of using the pseudoinverse are: it gives an optimal solution in the sense of least square and finds the optimal weights with minimal norm. [33] extended the above algorithms to random sampling: the hidden weights are sampled from a continuous distribution. It shows that for the single hidden layer neural network, the optimization for the hidden layer parameters does not improve the generalization behavior significantly, while updating the output weights is more effective. Randomized algorithms have been successfully applied to nonlinear system identification in [70].

The parameter identification of neural models is usually addressed by some gradient descent variants, e.g., the least squares algorithm, back-propagation, and Levenberg-Marquardt method. Even though these methods have been widely used, they may converge very slowly and have the local minima problem [34]. Since the identification error space is unknown, the neural model can be settled down in a local minimum easily if the initial weights of the neural model are not suitable [26]. There are some techniques to overcome the local minima in the error space and to force the neural model near the global minimum, such as noise-shaping modification for the gradient descent algorithm [12], adding momentum term [52], and combining nonlinear clustering [49]. These algorithms modify the gradient descent algorithms to avoid the local minima problem, but they do not solve the key problem of the local minima: wrong initial weights.

The pseudoinverse approach of the randomized algorithm can solve the local minima problem without considering the hidden weights [2]. By the sensitivity ratio analysis, [65] gives a method to calculate the initial weights of a recurrent neural network. In [76], the initial weights are obtained by finding the support vectors of the input data. However, the above papers do not consider one important issue: the initial hidden weights depend on the statistical features of the input data [33].

In general, a deep neural network has the same structure as a MLP where the depth of the neural network is defined as the number of hidden layers [4]. In order to be considered *deep*, a structure has to have at least two hidden layers [4], this depth usually gives the network the advantage of needing fewer neurons (or weights) than a shallow MLP [28]. However, increasing the number of hidden layers causes exponentially increasing model complexity and requires more training examples [23]. On the other hand, restricted Boltzmann machines [28] use energy-based learning models whose training process is unsupervised, *i.e.*, it uses input information.

Deep learning has two goals: a) it guides the weights to regions of minimal norm, and b) it sets the weights in zones of the parameter space where the likelihood of a global minimum is maximum [4]. The results of [23] show that the unsupervised training can drive the neural model away from the local minima for classification problems. However, deep learning methods cannot be applied to system identification directly, because the input/output values are non-binary as in classification problems. Most of deep learning techniques also use binary

data, for example the conditional probability transformation in the restricted Boltzmann machines needs binary values [28].

Deep learning techniques for system identification can be regarded as a pre-training stage where only input data are used. The objective of this stage is to learn the probability distribution of the input data $P(x)$. This helps to decide the conditional probability distribution $P(y|x)$, which is the objective of system identification [23]. Since the unsupervised deep learning minimizes the variance and introduces bias into the input space X, the supervised learning for X and Y can be improved. This is explained by [4]: in the unsupervised learning stage, the input information is sent to hidden layers to construct useful statistical features. This mechanism improves the corresponding input/output representation. The input distribution $P(x)$ appears in the hidden units via the deep learning method.

In this thesis, we take both advantages of the deep learning and the randomized algorithm for nonlinear system identification. We modify the learning rule of a special kind of restricted Boltzmann machine to train the hidden weights with input data. Then we use the randomized algorithm to train the output weights. Three benchmark examples are applied to show that the randomized algorithm with deep learning modification can improve the identification accuracy for nonlinear system identification.

2.6.2 Fuzzy logic and probability theory for system identification

A fuzzy model can approximate a large class of nonlinear systems, while keeping linguistic propositions of human thinking [79]. Moreover, a fuzzy model can be regarded as an universal estimator as it can approximate any nonlinear function to any prescribed accuracy, provided that sufficient fuzzy rules are available [11][48]. It is often claimed that fuzzy models are more robust than nonfuzzy methods against the sensitivity of variations of the data, or varying dynamics of nonlinear systems [38].

Data-driven fuzzy modeling uses observed data to construct a fuzzy model automatically. It needs two processes: 1) extracting suitable fuzzy rules from the data and deriving a fuzzy model; 2) updating the parameters of the fuzzy model with the data. The first process is called structure identification while the second process is called parameter identification. The key problem of the structure identification is the extraction of the fuzzy rules. The fuzzy rules can be obtained from mechanistic prior knowledge of nonlinear systems [45], from the

knowledge of experts [11], or from data [48][80]. However, it is difficult to obtain mechanistic prior knowledge for many nonlinear processes, and the expert method needs the un-bias criterion and the trial-and-error technique [58], which can only be applied off-line. The data-driven fuzzy modeling is very effective to identify a wide class of complex nonlinear systems when we have no complete model information, or even when we consider the nonlinear system as a black box [57].

Extraction of fuzzy rules from the input/output data usually uses the partition method, which is also called fuzzy grid [41]. Many data clustering methods are applied for structure identification, such as fuzzy C-means clustering [54], mountain clustering [54], and subtractive clustering [16]. These approaches require that the data is ready before the modeling. On-line clustering with a recursively calculated spatial proximity measure is given in [3]. The combination of on-line clustering and genetic algorithms for fuzzy systems is proposed in [36]. In [77] the input space is automatically partitioned into fuzzy subsets by adaptive resonance theory. Besides these clustering approaches, fuzzy rule extraction can also be realized by neural networks [76], genetic algorithms [58], singular-value decomposition [15] and support vector machines [19]. These data based clustering methods do not use the probability distribution information of the data.

In the sense of probability theory, the object of system modeling is to obtain a conditional probability distribution $P(y|\mathbf{x})$ [23], where $\mathbf{x}$ is the input and y is the output. Recent results show that deep learning techniques can learn the probability distribution $P(x)$ of the input space with an unsupervised learning method. [4] shows that in the unsupervised learning stage, the input information is sent to hidden layers to construct useful statistical features. This mechanism improves the corresponding input/output representation while the input distribution $P(x)$ appears in the hidden units via the deep learning method.

As seen in Chapter 3, RBMs [28] are main deep learning methods that use energy-based learning models. It has been shown that they can be used as nonlinear transformations which extract useful features from the input data that are more suitable for classification or regression tasks than the raw data themselves. Moreover, fuzzy modeling can be improved if the input data is transformed first (using RBMs) instead of being presented directly to the regression model. In this thesis, we first measure the benefits of using an unsupervised stage as an entry process for fuzzy modeling.

Both fuzzy models and probability theory can represent and process uncertain data effectively [14]. The dynamics and uncertainty of the data set in many cases have probabilistic nature [25]. The clustering methods discussed above partition the data directly by calculating Euclidean distances. These clusters do not include the distribution properties of the input/output data. They also do not scale well with large datasets due to the quadratic computational complexity of calculating all the pair-wise distances [47]. The clustering methods based on probability theory and statistical models are more powerful for big and uncertain data [24]. On the other hand, we use a restricted Boltzmann machine (RBM) to obtain the hidden features of the joint vectorial space of the input/output pairs. The data obtained from the RBM used for clustering are in the form of probability distributions. The second contribution of this thesis is that a probability based clustering method is proposed to extract fuzzy rules.

Including probability theory in fuzzy modeling can improve the stochastic modeling capability [30]. In [49], the probabilistic nature is added into the fuzzy relation between the input space and the output space to handle the effect of random noise and stochastic uncertainties. [73] introduces a probability distribution in the consequent part of the fuzzy rules improving the fuzzy classifiers. In this thesis, we introduce a probability parameter in each fuzzy rule. This idea comes from the Z-number [78], where a probability measure is included into the fuzzy number to make the decision fruitful based on human knowledge. The third modification proposed in this work is that we apply probability parameters to classical fuzzy model and train these parameters.

2.6.3 Deep conditional RBMs

The most popular deep learning models are the well known deep belief networks (DBN) [27], convolutional neural networks (CNN) [43], and deep Boltzmann machines (DBM) [62]. By using a deep structure, feature extraction, unsupervised learning, and probabilistic analysis, these models successfully solve many problems in machine learning. Unlike a DBN whose top two layers are restricted Boltzmann machines, a DBM uses a restricted Boltzmann machine in its whole net, so the inference and training of a DBM are in both directions. These allow the DBM to extract features from the ambiguous and complex input better than DBNs and CNNs. However, the training of DBMs is more difficult and slow than the one used by DBNs

and CNNs [61]

DBMs are generative energy based models. They learn the probability distribution of the input data through the usage of latent or hidden variables. The latent variables capture features of the data, which helps DBMs to obtain better representations of the empirical distribution. A DBM can be used as a stand-alone classifier, not only as feature extractor [40]. [44] shows that with sufficient hidden nodes a DBM can approximate any marginal distribution with any desired accuracy. A DBM is a very successful method for feature extractions from image and text data. It is also an excellent pre-training tool to set the initial parameters for discriminative models [23]. These two properties of DBMs have been widely used for solving classification problems in the past years [28].

DBMs as predictive models, are also applied for data regression and time series modeling [80]. The time series are the input to the DBM and the output of the DBM is the predicted values of the time series. Since the hidden and visible units of the DBM are binary, the prediction results for continuous values are not satisfied [39]. [59] uses denoising autoencoders to pre-train the model. The prediction results are better than no pre-training learning methods. However, [9] points out that the denoising autoencoder may not improve prediction results if the time series is not sufficiently large.

There are two correlation time series in system identification, named input x and output y. In the sense of probability theory, the objective of system identification is to find the best conditional probability distribution $P(y|x)$ [51]. As shown in [27][5], a DBM can learn the probability distribution among the input data, and obtain their hidden features. The time series modeling only for output y does not give the dynamic properties between the input x and the output y. Recent results show that deep learning techniques can be applied for nonlinear system modeling by learning the probability distribution of the input space [22]. The unsupervised learning is used to obtain the input features and send them to hidden layers. This mechanism improves the corresponding input/output representation, i.e., the modeling accuracy can be improved.

Chapter 3

Restricted Boltzmann machines and randomized algorithms

Restricted Boltzmann machines are the main building block of deep architectures, they are used as a pretraining stage of each hidden layer during the training of a MLP, however, they are modeling architectures by themselves which encourage their usage along with other techniques like randomized algorithms, fuzzy modeling or as stand alone structures. In this chapter, we first introduce the reader to the randomized algorithms framework presenting the history behind their emergence and the advantages they offer, then a formal definition of system identification and restricted Boltzmann machines is given. Finally, both algorithms are used together to create a modeling environment that takes the best from the two paradigms.

3.1 A simple deep learning scheme for nonlinear system identification

As explained in previous sections, deep learning has many advantages over the algorithms that perform on shallow architectures. In this section we present a simple approach that takes advantages of the pretraining stage in order to get a good model for nonlinear system identification. We use the ideas presented in [22] which serve as a starting point to understand how deep learning can be used. Consider the scheme shown in Figure 3-1, it represents the classical approach to model a nonlinear system. The input is presented to both: the proposed model and the real system in order to get their respective outputs. Once the outputs are

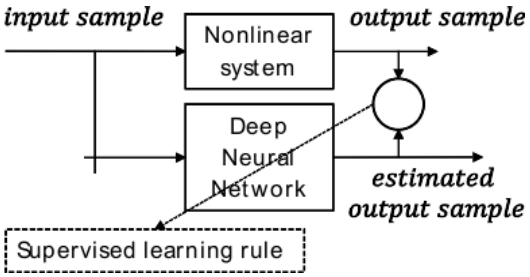

Figure 3-1: Deep identification structure

calculated, the error is measured substracting both quantities; an estimate of the system performance is obtained using the error information over several training examples. With the gathered information the model parameters are updated to fit better on the data.

The approach presented in Figure 3-1 is not exclusive of deep architectures, it constitutes the general approach that is used for almost every nonlinear system model (support vector machines, neural networks, state space models, ...). However, what makes deep learning different is the presence of a pretraining stage, usually, the regression process consists on a supervised learning method where the model parameters are chosen randomly at first, then the parameters are updated following a learning rule that tries to minimize a cost function which depends on the error obtained by the model. The approach taken in [22] differs in the sense that it incorporates two new stages as shown in Figure 3-2.

The system identification algorithm is split into four stages: random hyperparameter selection, pretraining stage, supervised stage and testing. A brief description of each stage is given next, it is not given a formal definition as the purpose of this text is only to present the framework in where this thesis was developed.

1. Hyperparameter selection: When a certain regression model is chosen to learn the behavior of a system, a number of priors have to be assumed. In the case of a deep neural network these priors are gathered in what is called the hyperparameter set, this set includes parameters as:

   (a) Number of layers.

   (b) Number of neurons in each layer.

   (c) Activation functions in each layer.

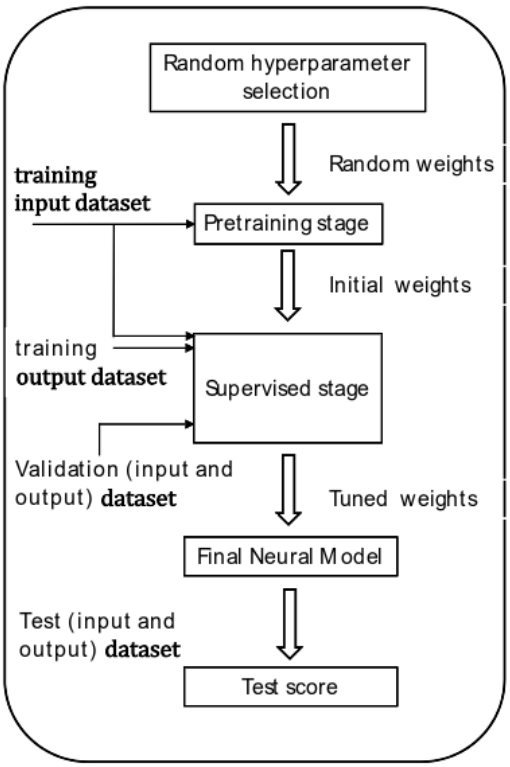

Figure 3-2: Flow of data of a deep training model

(d) Output function .

(e) Learning rates for pretraining and fine tuning procedures.

(f) Number of epochs.

(g) Batch size (if stochastic gradient descent is applied).

(h) Early stopping threshold.

These parameters are usually chosen using the developer criteria as there are not analytical methods to select the best ones. One simple approach is to prove all possible combinations of hyperparameters using a *grid search,* this kind of approximation is an exhaustive one because it takes into account every plausible value for each hyperparameter until it finally finds the optimal hyperparameter set. Although the *grid search* finds out the best solution, it is intractable when the number of hyperparameters to choose is high as the addition of a new hyperparameter increases exponentially the number of alternatives. To overcome this bottle neck, [7] has shown that a *random search* can achieve the same performance while only using some samples of the whole

sampling space. Once a set of hyperparameters is selected, the pretraining stage is implemented.

2. Pretraining stage. As it has been discussed, it consists on an unsupervised learning algorithm that is greedily applied to each layer of the net. The goal of the pretraining is to guide the weights to regions of the parameter space where they are more likely to achieve a better local minimum during the supervised training. Usually the unsupervised algorithm tries to train the layers as associative memories or encoders. The most important algorithms are restricted Boltzmann machines and denoising autoencoders.

   (a) Restricted Boltzmann machines: They are energy models that have an associated energy measure that has to be decreased by the training process. Their structure is divided in two: hidden units and visible units. Their architecture and functioning is explained later in this thesis.

   (b) Denoising autoencoders: They basically are one-layer neural networks whose output is their input. They are trained to reconstruct a *noisy* input, historically they have been used due to their simpler structure that constrasts with the probability scheme presented by the RBMs, a simple utilization of this algorithm for system identification can be found in [22].

3. Supervised stage: After the pretraining stage is finished and the initial weights are chosen, a supervised criteria is applied over the training set. The stochastic gradient descent and its variants are the most common methods to train deep neural models, in particular, backpropagation is used to train them because it allows to transmit the gradient effect through the network layers. In addition, early stopping criteria is sometimes used to avoid overfitting.

4. Testing: The final model is tested using a test dataset. The final performance of the model is usually measured utilizing the average squared error.

The above four points constitute a traditional deep learning modeling structure which is deeply analyzed in [22]. In this thesis we explore alternatives in the usage of deep learning combining it with other algorithms taking the best from each one. In the next chapters, deep

learning has been used along randomized algorithms, fuzzy modeling and as a stand alone procedure to identify nonlinear behaviors.

3.2 Nonlinear system identification framework

Consider the following unknown discrete-time nonlinear system

$$\bar{x}(k+1) = f\left[\bar{x}\left(k\right), u\left(k\right)\right], \quad y(k) = g\left[\bar{x}\left(k\right)\right] \tag{3.1}$$

where $u\left(k\right) \in \Re^u$ is the input vector, $\bar{x}\left(k\right) \in \Re^x$ is an internal state vector, and $y\left(k\right) \in \Re^m$ is the output vector. f and g are general nonlinear smooth functions $f, g \in C^\infty$. Denoting $Y(k) = \left[y^{\mathrm{T}}\left(k\right), y^{\mathrm{T}}\left(k+1\right), \cdots y^{\mathrm{T}}\left(k+n-1\right)\right]^{\mathrm{T}}, U(k) = \left[u^{\mathrm{T}}\left(k\right), u^{\mathrm{T}}\left(k+1\right), \cdots u^{\mathrm{T}}\left(k+n-2\right)\right]^{\mathrm{T}}$, if $\frac{\partial Y}{\partial \bar{x}}$ is non-singular at $\bar{x} = 0$, $U = 0$, this leads to the following NARMA model

$$y(k) = \Gamma\left[x\left(k\right)\right] \tag{3.2}$$

where

$$x\left(k\right) = [y^{\mathrm{T}}\left(k-1\right), y^{\mathrm{T}}\left(k-2\right), \cdots u^{\mathrm{T}}\left(k\right), u^{\mathrm{T}}\left(k-1\right), \cdots]^{\mathrm{T}}$$

$\Gamma\left(\cdot\right)$ is an unknown nonlinear difference equation representing the plant dynamics, $u\left(k\right)$ and $y\left(k\right)$ are measurable scalar input and output. The nonlinear system (3.2) is a NARMA model. We can also regard the input of the nonlinear system as $x\left(k\right) = \left[x_1 \cdots x_n\right]^{\mathrm{T}} \in \Re^n$, and the output as $y(k) \in \Re^m$

Now we use the MLP given by (2.8) to identify the unknown nonlinear system (3.2)

>From the Stone-Weierstrass theorem, if the number of nodes of a one hidden layer neural network is large enough, the neural model can approximate the nonlinear function Γ to any degree of accuracy for all $x\left(k\right)$. In this chapter, instead of increasing the number of nodes l_i of the single hidden layer, we increase the layer number p. We use a deep structure, *i.e.,* $p \geq 2$ (at least 2 hidden layers), for the multilayer neural model (2.8), such that we can use some existing deep learning techniques for system identification.

The object of the neural identification is to find a suitable structure (number of layers p, number of nodes in each layer l_i), the weights $W_1 \cdots W_p$, and β, such that the neural

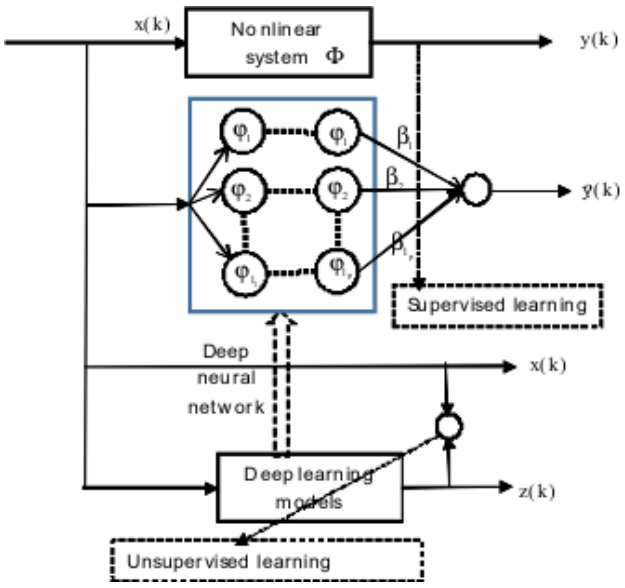

Figure 3-3: Randomized algorithms with deep learning for nonlinear system identification

identification error

$$e\left(k\right)=\widehat{y}\left(k\right)-y\left(k\right) \tag{3.3}$$

is minimized.

The randomized algorithms [33] use random weights in the single hidden layer (they use $p=1$) to avoid the problems of many supervised learning procedures, such as gradient descent and Hessian methods. In this chapter, we use the input data and an RBM as a unsupervised learning method, *i.e.,* deep learning, to solve the same problems of the neural modeling. We will show that the randomized algorithm with a deep learning modification can improve the modeling accuracy effectively. The neural modeling structure using randomized algorithms and deep learning techniques is shown in Figure 3-3, where φ denotes the parameters of each layer W and b. In the following sections, we will show how to use the restricted Boltzmann machines to find the structure and the initial weights $W_1 \cdots W_p$ with input data.

3.3 Restricted Boltzmann for system identification

Having random weights in the hidden layers of a MLP can be useful because in this way, the learning procedure can be focused in tuning the output layer. Nevertheless, random selection may not be the best configuration as shown in this chapter, we use the input $x\left(k\right)$ in (2.8)

to construct better hidden weights.

The restricted Boltzmann machine (RBM) is a deep learning method [28], which trains the weights under a probability distribution by using only the input dataset. The goal of an RBM is to create a stochastic machine capable of reconstructing the input x from a distribution $P(x)$, which denotes the reconstruction probability of x. The RBM training process tries to maximize $P(x)$ along all training examples, this reconstruction algorithm can be applied to all hidden layers of (2.8) in order to set good initial weights for the supervised training stage [28].

In this section, we present a formal definition for a restricted Boltzmann machine as an energy based model. The training procedure which is an application of the stochastic gradient descent algorithm is developed along with a modification for the handling of non-binary inputs.

3.3.1 Contrastive divergence

In order to apply gradient descent over an energy based function we need to find a way to find the expectation of our distribution, we first explain contrastive divergence (CD) as an approximate Maximum-Likelihhod (ML) learning algorithm that was proposed by Geoffrey Hinton. Suppose we would like to model the probability distribution of a data point x using a function of the form $f(x;\Theta)$, where Θ is a vector of model parameters. The probability of x, $p(x;\Theta)$ must integrate to 1 over all x, therefore:

$$p(x;\Theta) = \frac{1}{Z(\Theta)} f(x;\Theta) \tag{3.4}$$

where $Z(\Theta)$, known as the partition function, is defined as

$$Z(\Theta) = \int f(x;\Theta)dx \tag{3.5}$$

The model parameters Θ, are learned by maximizing the probability of a training set of data, X=$x_{1,\ldots,K}$, given as

$$p(\mathrm{X};\Theta) = \prod_{k=1}^{K} \frac{1}{Z(\Theta)} f(x_k;\Theta) \tag{3.6}$$

or, equivalently, by by minimizing the negative log of $p(\mathrm{X};\Theta)$, denoted $E(\mathrm{X};\Theta)$, which we shall call the energy:

$$E(\mathrm{X};\Theta) = \log Z(\Theta) - \frac{1}{K}\sum_{k=1}^{K} \log f(x_k;\Theta) \tag{3.7}$$

First, choose the probability model function, $f(x;\Theta)$, to be the pdf of a normal distirbution $N(x;\mu,\sigma)$, so that $\Theta = \{\mu,\sigma\}$. The integral of the pdf is 1, so that $\log Z(\Theta) = 0$. Differentiating equation 3.7 with respect to μ is the mean of the training data X, and a similar calculation with respect to σ shows that the optimal σ is the square root of the variance of the training data.

Sometimes, as in this case, a method exists that can exactly minimize the particular energy function. Now let choose the probability model function $f(x;\Theta)$, to be the sum of N normal distributions, so that $\Theta = \left\{\mu_{1,\ldots,K}, \sigma_{1,\ldots,K}\right\}$ and

$$f(x;\Theta) = \sum_{i=1}^{N} N(x;\mu_i,\sigma_i) \tag{3.8}$$

This is equivalent to a sum of experts or mixture model, with equal weights on all the experts; having different weights is a trivial extension to the model. Again using the fact that a normal distribution integrates to 1, we can see from equation 3.8 that $\log Z(\Theta) = \log N$. However, now differentiating equation 3.7 with respect to each of the model parameters produces equations dependent on other model parameters, so we cannot calculate the optimal model parameters straight off. Instead we can use the partial differential equations and a gradient descent method with line search to find a local minimum of energy in the parameter space.

Choose the probability model function $f(x;\Theta)$, to be the product of N normal distributions, so that

$$f(x;\Theta) = \sum_{i=1}^{N} N(x;\mu_i,\sigma_i) \tag{3.9}$$

This is equivalent to a product of experts model. The partition function, $Z(\Theta)$, is now no longer a constant. We can see this by considering a model consisting of two normal distributions, both with $\sigma = 1$. If $\mu_1 = -\infty$ and $\mu_2 = \infty$ then $Z(\Theta) = 0$, while if $\mu_1 = \mu_2$

then $Z(\Theta) = \frac{1}{2}\sqrt{\pi}$.

While it is possible, in this case, to compute the partition function exactly given Θ, suppose that the integration part is not algebraically tractable (as will be tha case with other probability model functions). In this case we would need to use a numerical integration in parameter space, and use a gradient descent method to find a local minimum. For high dimensional data spaces the integration time is crippling, and a high-dimensional parameter space compounds this problem. This leads to a situation where a situation where we are trying to minimize an energy function that we cannot evaluate.

Even though we cannot evaluate the energy function itself, CD provides a way to estimate the gradient of the energy function. CD effectively gives us a sense of balance, by taking very small steps in the direction of steepest gradient we can then find a local minimum.

As explained, CD estimates the energy function's gradient, given a set of model parameters, Θ, and the training data, X. We derive the partial derivative of Equation 3.7:

$$\begin{aligned} \frac{\partial E(\mathrm{X}, \Theta)}{\partial \Theta} &= \frac{\partial \log Z(\Theta)}{\partial \Theta} - \frac{1}{K} \sum_{i=1}^{K} \frac{\partial \log f(x_i; \Theta)}{\partial \Theta} \\ &= \frac{\partial \log Z(\Theta)}{\partial \Theta} - \mathrm{E}_{\mathrm{X}} \left(\frac{\partial \log f(x_i; \Theta)}{\partial \Theta} \right) \end{aligned} \tag{3.10}$$

where $\mathrm{E}_{\mathrm{X}}(\cdot)$ is the expectation of $\cdot$ given the data distribution X.

The first term on the right-hand side comes from the partition function, $Z(\Theta)$, which, as Equation 3.5 shows, involves an integration over x. Substituting this in, we get

$$
\begin{aligned}
\frac{\partial \log Z(\Theta)}{\partial \Theta} &= \frac{1}{Z(\Theta)} \frac{\partial \log Z(\Theta)}{\partial \Theta} \\
&= \frac{1}{Z(\Theta)} \frac{\partial}{\partial \Theta} \int f(x;\Theta) dx \\
&= \frac{1}{Z(\Theta)} \int \frac{\partial f(x;\Theta)}{\partial \Theta} dx \\
&= \frac{1}{Z(\Theta)} \int f(x;\Theta) \frac{\partial \log f(x;\Theta)}{\partial \Theta} dx \\
&= \int p(x;\Theta) \frac{\partial \log f(x;\Theta)}{\partial \Theta} dx \\
&= \mathrm{E}_{p(x;\Theta)} \left(\frac{\partial \log f(x;\Theta)}{\partial \Theta} \right)
\end{aligned}
\tag{3.11}
$$

As discussed, this integration is generally algebraically intractable. However, in the form of Equation 3.11, it is clear that it can be numerically approximated by drawing samples from the proposed distribution, $p(x;\Theta)$.

Samples cannot be drawn directly from $p(x;\Theta)$ as we do not know the value of the partition function, but we can use many cycles of Markov Chain Monte Carlo (MCMC) sampling to transform the training data (drawn from the target distribution) into data drawn from the proposed distribution. This is possible as the transfromation only involves calculating tha ratio of two probabilities, $p(x';\Theta)/p(x;\Theta)$, so the partition function cancles out. X^n represents the training data transformed using n cycles of MCMC, such that $\mathrm{X}^0 \equiv \mathrm{X}$. Putting this back into Equation 3.10, we get:

$$
\frac{\partial E(\mathrm{X},\Theta)}{\partial \Theta} = \mathrm{E}_{\mathrm{X}^\infty} \left(\frac{\partial \log f(x;\Theta)}{\partial \Theta} \right) - \mathrm{E}_{\mathrm{X}^0} \left(\frac{\partial \log f(x;\Theta)}{\partial \Theta} \right) \tag{3.12}
$$

We still have a computational hurdle to overcome, i.e., the several MCMC bucles required to compute an accurate gradient will take a long time. Hinton's assertion was that only a few MCMC cycles would be needed to calculate an approximate gradient. The intuition behind this is that after a few iterations the data will have moved from the target distribution towards the proposed distribution, and so give an idea in which direction the proposed distribution should move to better model the training data. Empirically, Hinton has found that even 1 cycle of MCMC is sufficient for the algorithm to converge to the ML answer.

$$\Theta_{t+1} = \Theta_t + \eta \left[\mathrm{E}_{\mathrm{X}^0} \left(\frac{\partial \log f(x;\Theta)}{\partial \Theta} \right) - \mathrm{E}_{\mathrm{X}^1} \left(\frac{\partial \log f(x;\Theta)}{\partial \Theta} \right) \right] \tag{3.13}$$

As such, bearing in mind that we wish to minimize the energy function, the parameter update equation may be written as Equation 3.13 where η is the step size factor, which should be chosen experimentally, based on convergence time and stability.

3.3.2 Standard RBMs and their training procedure

An RBM is an energy-based model, which is defined by a probability distribution. This probability distribution depends on the current configuration (or energy) of the model. Consider a training example $x(k)$ (for simplicity denoted as x), the training goal is to maximize the following probability function of the model,

$$P(x) = \sum_h P(x,h) = \sum_h \frac{e^{-E(x,h)}}{Z} \tag{3.14}$$

where x is the input to the model, h is the hidden representation, Z is a partition function defined as $Z = \sum_h \sum_x e^{-E(x,h)}$, and $P(x)$ is the probability distribution of x. $E(x,h)$ is the energy function which is defined by

$$E\left(x,h\right) = -c^{\mathrm{T}}x - b^{\mathrm{T}}h - h^{\mathrm{T}}Wx \tag{3.15}$$

In (3.14), $\sum_h$ and $\sum_x$ denote the sums of over all possible values of h and x . They are tractable when the input and hidden spaces are discrete (or binary). In continuous spaces, the summations become integrals that have to be evaluated. For identification purposes, W and b can be regarded as the weights and bias of some layer i in (2.8), and c is a bias vector of appropriate dimension. In this way, x represents the input of layer i with size l_{i-1}, and $h \in \Re^{l_i}$ is the hidden representation. W is called the hidden layer weigths, b and c are called visible and hidden bias respectively.

In order to maximize $P(x)$ with respect to the weights, we have to redefine $P(x)$. The probability distribution of such a model is given by the following concept of free energy

$$F\left(x\right) = -\log \sum_h e^{-E(x,h)} \tag{3.16}$$

With this definition, (3.14) becomes

$$P(x) = \frac{e^{-F(x)}}{Z}, \quad Z = \sum_x e^{-F(x)} \tag{3.17}$$

Let expand x as $x = [x_1 \cdots x_t \cdots x_{l_{i-1}}]^{\mathrm{T}}$ with $t = 1, 2, ..., l_{i-1}$ and $h = [h_1 \cdots h_s \cdots h_{l_i}]^{\mathrm{T}}$ with $s = 1, 2, ..., l_i$. Substituting (3.15) into (3.16), the free energy becomes

$$F(x) = -c^{\mathrm{T}}x - \sum_{s=1}^{l_i} \log \sum_{h_s} e^{h_s(b_s + W_s x)} \tag{3.18}$$

In (3.18), W has been divided as $W = \left[W_1^{\mathrm{T}} \cdots W_s^{\mathrm{T}} \cdots W_{l_i}^{\mathrm{T}}\right]^{\mathrm{T}}$, $s = 1, ..., l_i$, where W_s are row vectors of size l_{i-1}. In some simple cases, such as classification and dimensionality reduction, x_t takes binary values, *i.e.,* $x_t \in \{0, 1\}$, and the binary hidden units are $h_s \in \{0, 1\}$. The probabilistic version of the neural model becomes

$$\begin{aligned} P\left(h_s = 1|x\right)_{s=1\cdots l_i} &= \phi\left[W_s x + b_s\right] \\ P\left(x_t = 1|h\right)_{t=1\cdots l_{i-1}} &= \phi\left[W_t^T h + c_t\right] \end{aligned} \tag{3.19}$$

where W_t is the t-th column of W and ϕ is the sigmoid function $\phi(x) = 1/(1 + e^{-x})$. Here the hidden units and the visible units are conditionally independent. So the conditional probabilities of them are

$$\begin{aligned} P(h|x) &= \prod_{s=1\cdots l_i} P(h_s|x) \\ P(x|h) &= \prod_{t=1\cdots l_{i-1}} P(x_t|h) \end{aligned}$$

The free energy for binary visible and hidden units becomes

$$F(x) = -c^{\mathrm{T}}x - \sum_{s=1}^{l_i} \log\left(1 + e^{(b_s + W_s x)}\right) \tag{3.20}$$

We introduce the learning rate $\eta_1 > 0$. The weights and biases of RBM are updated using the gradient descent algorithm which will minimize the function $-\log P(x)$,

$$\Lambda(k+1) = \Lambda(k) - \eta_1 \frac{\partial - \log P(x)}{\partial \Lambda(k)} \tag{3.21}$$

where Λ denotes the updated parameters, which can be $W_{s,t}$, b_s or c_t, $t = 1, ..., l_{i-1}$, $s =$

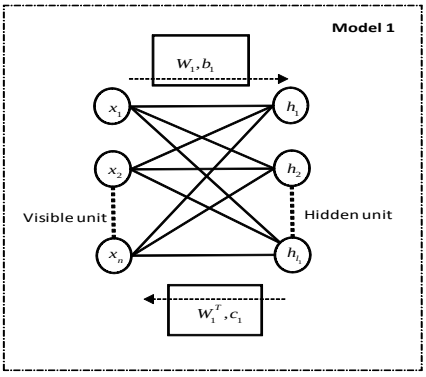

Figure 3-4: Markov sampling in a restricted Boltzmann machine

$1, ..., l_i$. If we denote z as the reconstruction of x, z is sampled from the RBM, $\sum_z$ indicates a sum along the entire sampling space of z. The log-likelihood gradient with respect to Λ is

$$\frac{\partial \log P(x)}{\partial \Lambda(k)} = \sum_z P(z) \frac{\partial F(z)}{\partial \Lambda(k)} - \frac{\partial F(x)}{\partial \Lambda(k)}$$

In this section, we estimate $\sum_z P(z) \frac{\partial F(z)}{\partial \Lambda(k)}$ with a set S which includes s finite samples [28] among the probability distribution. Considering each sample with equal probability $1/s$, we get

$$\sum_z P(z) \frac{\partial F(z)}{\partial \Lambda(k)} \approx \frac{1}{s} \sum_{z \in S} \frac{\partial F(z)}{\partial \Lambda(k)}$$

Here the samples S are obtained with a Monte Carlo algorithm with contrastive divergence [1]. The approximation capability is improved when the number of hidden units increases [23]. Figure 3-4 shows the first layer of a deep RBM based model. The transformation (3.19) is repeated s times, which generates s samples for the learning process, see Figure 3-4.

A sample z from a training example x using a $k-$steps sampling process of the Monte Carlo chain is obtained using Algorithm 4:

Algorithm 4

1. Calculate $P(h|x)$ using the current W and b.

2. Sample h using the conditional distribution $P(h|x)$.

3. Calculate $P(x|h)$ using the current W and c.

4. Sample z using the conditional distribution $P(x|h)$.

5. Repeat steps 1-4 k times using the new sample z obtained in step 4, and the the new x in step 1. After k times, we get a sample z for the set S.

After s sampling processes, the samples S and the input $x(k)$ are used to update the parameters W, b and c. Then a new training example $x(k+1)$ is presented to the model. One training epoch consists of q examples.

For an RBM model we can calculate the hidden representation $h(k)$ associated with the input $x(k)$. It is convenient to compute it as the conditional probabilities of the distribution $z(k)$. Similarly with a MLP model, the hidden representation of the Model 1 in Figure 3-4, which is directly associated with the input $x(k)$, is calculated by

$$h_1(k) = \phi_1 [W_1 x(k) + b_1]$$

This is the input of Model 2 in Figure 3-5.

3.3.3 Conditional probability transformation for non-binary values

For nonlinear system identification, the visible units cannot be binary values, thus, the conditional probability transformation cannot always be the form of (3.19). We consider three domains for the input $x(k)$: 1) interval $[0, \infty)$, for unbounded positive inputs; 2) $[0, 1]$, for normalized inputs; and 3)$[-\delta, \delta]$, for bounded inputs.

1) The visible and hidden units are in the interval $[0, \infty)$.

The conditional probability for the energy function of the model (3.14) is

$$P(x_i|h) = \frac{e^{(W_t^{\mathrm{T}} h + c_t)x_t}}{\int_{x_t} e^{(W_t^T h + c_t)x_t} dx_t} \tag{3.22}$$

If $x_t(k) \in (-\infty, \infty)$, the integral term has an algebraic form that does not converge. Let denote

$$a_t(h) = W_t^{\mathrm{T}} h + c_t \tag{3.23}$$

where a_t is the term applied to the visible units, see Figure 3-4. If the terms $x_t(k)$ are

non-negative, $x_t(k) \in [0, \infty)$, (3.22) becomes

$$P(x_t|h) = \frac{e^{a_t x_t}}{\int_0^\infty e^{a_t x_t} dx_t} = \frac{e^{a_t x_t}}{\frac{1}{a_t} e^{a_t x_t}|_0^\infty} \tag{3.24}$$

(3.24) has finite value if $a_t(h) < 0$, $\forall h$. The conditional probability distribution is

$$P(x_t|h) = -a_t(h)\, e^{a_t(h) x_t} > 0 \tag{3.25}$$

The visible units which have a probability distribution as (3.25) are called exponential units.

In order to perform the sampling process, we need to calculate the cumulative probability distribution $P_C(x_t|h)$.

$$P_C(x_t|h) = \int_0^{x_t} P(x_t|h) dx_i = \int_0^{x_t} -a_t e^{a_t x_t} dx_t = 1 - e^{a_t x_t} \tag{3.26}$$

So $P_C(x_t|h)$ always increases. The sampling process is possible by using the inverse function of cumulative probability P_C^{-1},

$$z_t(k) = \frac{\ln(1 - P_C)}{a_i} \tag{3.27}$$

If $o(k)$ is a value from a sampling process on a uniform distribution, then we can associate it with the cumulative density value P_C. The value of the corresponding visible unit is

$$z_t(k) = \frac{\ln(1 - o(k))}{a_t} \tag{3.28}$$

The expected value according to the distribution $P(x_t|h)$ is

$$\mathrm{E}[x_t] = \int_0^\infty P(x_t|h) x_t dx_t = -a_t \int_0^\infty e^{a_t x_t} x_t dx_t = -\frac{1}{a_t(h)} \tag{3.29}$$

2) The visible and hidden units are in the interval $[0, 1]$

The positive range $[0, \infty)$ is not needed most of the time as physical systems are always constrained in some sense. We use a normalization method to force the input to fit in $[0, 1]$. The probability distribution is also bounded. (3.22) is

$$P(x_t|h) = \frac{e^{a_t x_t}}{\int_0^1 e^{a_t x_t} dx_t} = \frac{e^{a_t x_t}}{\frac{1}{a_t} e^{a_t x_t}|_0^1} = \frac{a_t e^{a_t x_t}}{e^{a_t} - 1} \tag{3.30}$$

To use the Gibbs sampling process, the conditional probability $P_C(x_t|h)$ is computed

$$P_C(x_t|h) = \int_0^{x_t} P(x_t|h)dx_t = \frac{a_t}{e^{a_t}-1}\int_0^{x_t} e^{a_t x_t}dx_t = \frac{e^{a_t x_t}-1}{e^{a_t}-1} \tag{3.31}$$

This leads to

$$z_t(k) = \frac{\log\left[1 + P_C(e^{a_t}-1)\right]}{a_t}$$

With a sample unit from the uniform distribution $o(k)$, the new value $z_t(k)$ with respect to P_C is

$$z_t(k) = \frac{\log\left[1 + o(k)\left(e^{a_t}-1\right)\right]}{a_t} \tag{3.32}$$

Finally the expected value of the distribution is calculated as

$$\mathrm{E}[x_t] = \int_0^1 P(x_t|h)x_t dx_t = \frac{a_t}{e^{a_t}-1}\int_0^1 e^{a_t x_t}x_t dx_t = \frac{1}{1-e^{-a_t}} - \frac{1}{a_t} \tag{3.33}$$

3) The visible and hidden units are in the interval $[-\delta, \delta]$

If the input set $x(k)$ can be positive and negative, we define the operating interval as $[-\delta, \delta]$ for each visible unit. In this case the conditional probability is truncated exponential. (3.22) transforms into

$$P(x_t|h) = \frac{e^{a_t x_t}}{\int_{-\delta}^{\delta} e^{a_t x_t}dx_t} = \frac{e^{a_t x_t}}{\frac{1}{a_t}e^{a_t x_t}|_{-\delta}^{\delta}} = \frac{a_t e^{a_t x_t}}{e^{a_t\delta} - e^{-a_t\delta}} \tag{3.34}$$

The cumulative probability distribution is

$$P_C(x_t|h) = \int_{-\delta}^{x_t} P(x_t|h)dx_t = \frac{a_t}{e^{a_t\delta} - e^{-a_t\delta}}\int_{-\delta}^{x_t} e^{a_t x_t}dx_t = \frac{e^{a_t x_t} - e^{-a_t\delta}}{e^{a_t\delta} - e^{-a_t\delta}} \tag{3.35}$$

The sampling process that uses the inverse function of P_C and $o(k)$ in the uniform distribution is

$$z_t(k) = \frac{\log\left[e^{-a_t\delta} + o(k)\left(e^{a_t\delta} - e^{-a_t\delta}\right)\right]}{a_t} \tag{3.36}$$

The expected value of this distribution is

$$\mathrm{E}[x_t] = \int_{-\delta}^{\delta} P(x_t|h)x_t dx_t = \frac{a_t}{e^{a_t\delta} - e^{-a_t\delta}}\int_{-\delta}^{\delta} e^{a_t x_t}x_t dx_t = \delta\frac{e^{a_t\delta} + e^{-a_t\delta}}{e^{a_t\delta} - e^{-a_t\delta}} - \frac{1}{a_t} \tag{3.37}$$

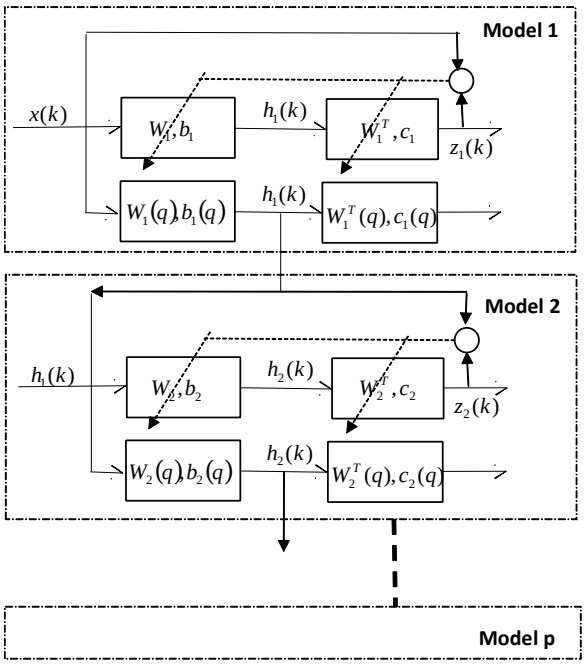

Figure 3-5: Deep RBM Model

3.3.4 Deep identification model

The unsupervised training for the deep RBM model is described in Algorithm 5:

Algorithm 5

1. The input and the hidden representation of the first model are $x(k) \in \Re^n$ and $h_1(k) \in \Re^{l_1}$. We use q data to train the weights of the first model $W_1 \in \Re^{l_1 \times n}$, $b_1 \in \Re^{l_1}$, and $c_1 \in \Re^n$.

2. After the first model is trained, their weights are fixed. The code or hidden representation of the first model is computed with fixed weights to generate q examples, which are the input of the second model.

3. The second model is trained using as input $h_1(k) \in \Re^{l_1}$ and it generates the hidden representation $h_2(k) \in \Re^{l_2}$, which is the input of the third model.

4. Then we train the third model, we keep repeating the procedure until all p models are trained. This training process is shown in Figure 3-5.

The RBM model in Figure 3-5 has a structure similar to that of the neural identification model (2.8). It uses input the $x(k)$ to update the model, while the identification model (2.8) uses the output $y(k)$ to train its weights.

Now, there are two options: 1) Consider the weights $W_1 \cdots W_p$ obtained from the pretraining process as initial values, and use the supervised learning to train both hidden weights $W_1 \cdots W_p$ and the output weight β; or 2) Use randomized algorithms to keep $W_1 \cdots W_p$ unchanged, and only train the output weight β.

For the first choice, we use the following square error

$$\Omega(k) = \|y(k) - \hat{y}(k)\|^2 \tag{3.38}$$

where $y(k)$ is the output of the unknown plant (3.2), $\hat{y}(k)$ is the output of the neural model (2.8). The weights W_i and β are updated by

$$W_i(k+1) = W_i(k) - \eta_2 \frac{\partial \Omega(k)}{\partial W_i(k)}, \quad i = 1 \cdots p \tag{3.39}$$

where $\eta_2 > 0$ is the learning rate of the supervised learning, $k = 1, 2 \cdots q$, q is the number of training examples, $W_i(0) = W_i(q)$, $W_i(q)$ are the final trained weights of the unsupervised stage model.

By the studies of [33] and [63], if the Moore-Penrose inverse is applied into the output layer, a training procedure in the hidden layers may worsen the modeling results by some supervised learning problems. Problems as overfitting and shadowing may also appear.

Noise (or disturbance) is an important issue in the system identification context, an external disturbance can be regarded as measurement noise or/and input noise. Within the deep learning environment, the input noises are included feedforward through each layer while the output noise (measurement noise) is enlarged due to the backpropagation of the identification error. This also affects the modeling accuracy.

In this chapter, we use the second choice, the weights of the hidden layers $W_1 \cdots W_p$ are not changed after they have been pretrained. Only the weights β in the output layer are trained by the randomized algorithm as in [33].

3.4 RBMs with randomized algorithms

We rewrite the neural model (2.8) as the following form

$$\widehat{y}(k) = \beta\Phi(k) \tag{3.40}$$

where $\Phi(k) = \phi_p\left\{W_p\phi_{p-1}\left[\ldots W_3\phi_2\left\{W_2\phi_1\left[W_1x(k)+b_1\right]+b_2\right\}+b_3\ldots+b_{p-1}\right]+b_p\right\}$. $\Phi(k)$ has been determined by RBMs. (3.40) is a linear-in-parameters system in the form of $y = Ax$. Here A may be singular and/or be not square, the solution x can be solved by the Moore-Penrose generalized inverse, which is defined as follows.

Definition 6 *The matrix $A^+ \in \Re^{n\times m}$ is the Moore-Penrose generalized inverse of $A \in \Re^{m\times n}$ if*

$$AA^+A = A, \quad A^+AA^+ = A^+, \quad \left(AA^+\right)^{\mathrm{T}} = AA^+, \quad \left(A^+A\right)^{\mathrm{T}} = A^+A \tag{3.41}$$

In particular, when A has full column rank,

$$A^+ = \left(A^{\mathrm{T}}A\right)^{-1}A^{\mathrm{T}} \tag{3.42}$$

When A has full row rank

$$A^+ = A^T\left(AA^T\right)^{-1} \tag{3.43}$$

Definition 7 *$x_0 \in \Re^n$ is said to be a minimum norm least-squares solution of the linear system $y = Ax$ if*

$$\|x_0\| \le \|x\|, \quad \forall x \in \{x : \|Ax-y\| \le \|Az-y\|, \forall z \in \Re^n\} \tag{3.44}$$

where $y \in \Re^m$.

For a linear system $y = Ax$, x_0 is a least-squares solution if

$$\|Ax_0 - y\| = \min_x \|Ax - y\| \tag{3.45}$$

where $\|\cdot\|$ is a norm in Euclidean space. If By is a minimum norm least-squares solution of the linear system $y = Ax$, then it is necessary and sufficient that $B = A^+$. Here A^+ is the Moore-Penrose generalized inverse of matrix A, which is defined in (3.41).

For our identification model, W_i and b_i in $\Phi(k)$ are fixed. The goal of the training algorithm is to find β such that the following cost function is minimized

$$J = \sum_k \|y(k) - \widehat{y}(k)\|^2 \tag{3.46}$$

The training data are $y(k)$ and $\Phi(k)$, $k = 1, 2 \cdots q$, q is the total training data number. In the best case, $J = 0$, then $\widehat{y}(k) = y(k) = \beta\Phi(k)$ for all k. Considering the entire training set,

$$\hat{Y} = \begin{bmatrix} \hat{y}(1) & \hat{y}(2) & \cdots & \hat{y}(q) \end{bmatrix} = \begin{bmatrix} \beta\Phi(1) & \beta\Phi(2) & \cdots & \beta\Phi(q) \end{bmatrix} = \beta\Psi \tag{3.47}$$

where $\Psi = [\Phi(1), \Phi(2), \cdots, \Phi(q)]$. Or in another form:

$$Y = \begin{bmatrix} y(1) & y(2) & \cdots & y(q) \end{bmatrix} = \begin{bmatrix} \beta\Phi(1) + e(1) & \beta\Phi(2) + e(2) & \cdots & \beta\Phi(q) + e(q) \end{bmatrix}$$
$$Y = \beta\Psi + E \tag{3.48}$$

where $e(k)$ is the modeling error $e(k) = y(k) - \widehat{y}(k)$, and $E = [e(1), e(2), \cdots, e(q)]$. To obtain $\min_\beta J$, we need $\frac{\partial J}{\partial \beta} = 0$. From (3.43)

$$\beta^* = Y\Psi^{\mathrm{T}} \left(\Psi\Psi^T\right)^{-1} = Y\Psi^+ \tag{3.49}$$

So β^* can minimize the index J in (3.46).

Since β^* is one of the least-squares solutions of the system $Y = \beta\Psi + E$, it reaches the smallest approximation error on the training dataset, and it is unique. The solution β^* has the smallest norm for a least-squares solution of $Y = \beta\Psi$. [63] shows that for feedforward networks, small norm of the weights is more important than the number of nodes to obtain small generalization error. Since the norms of the hidden weights which are generated by deep learning are small [4], the combination of deep learning and least-squares can provide good generalization performance.

The final training procedure follow Algorithm 8:

Algorithm 8

1. Construct a deep neural model (2.8) with $p \geq 2$.

2. Use the input data and the deep learning algorithm to train the hidden weights $W_1 \cdots W_p$

3. Calculate the output weight β^* in (3.49) with Ψ in (3.47)

In order to obtain good approximation capability, the distributions of the random hidden weights and biases should be defined in advance [33]. Arbitrary assignment of the hidden weights may lead to poor performances. The deep learning technique discussed in this thesis can be regarded as an alternative method to find the distributions of the hidden weights of the randomized algorithm. The restricted Boltzmann machines (RBM) for random hidden weights works as Algorithm 9:

Algorithm 9

1. The hidden weights are randomly assigned in $[-1, 1]$.

2. An RBM is applied to learn the probability distribution of the input $P(x)$. This information is sent to the hidden layers.

3. The visible units of RBM are encoded into three types: $[0, 1]$, $[0, \infty)$, and $[-d, d]$, $d \neq 1$.

4. The conditional distribution of the hidden weights and biases are updated by Monte Carlos algorithm.

The deep learning technique provides a possible selection manner of hidden weights for randomized algorithms with the distribution of the input data. After the RBM pre-training, the hidden weights are not longer in $[-1, 1]$. The examples in the next section show how the hidden weights are expanded from $[-1, 1]$ by different input distributions, and the identification errors are influenced by these areas.

3.5 Simulations and comparisons

In this section, we use three benchmark examples to show the effectiveness of the combination of deep learning techniques and randomized algorithms for nonlinear system identification.

Gas furnace data

The gas furnace dataset is a commonly used benchmark [10]. The input $u(k)$ is the flow rate of the methane gas, while the output $y(k)$ is the concentration of CO_2 in the gas mixture under a steady air supply. The dataset has 296 samples at a fixed interval of 9 seconds. [10] used a time-series based approach to develop a linear model while [67] and [75] used this dataset to evaluate their fuzzy modeling methods.

In this example, we use the same data structure as [67][75], the recursive input data for the model is $X(k) = [y(k-1), \cdots y(k-4),\ u(k), \cdots u(k-5)]^{\mathrm{T}}$, the model output is $\hat{y}(k)$. 200 samples are applied for training. In order to use a restricted Boltzmann machine, the training values of $X(k)$ and y are normalized to match the conditions of (3.50). The gas furnace dataset has the form of (3.2) with $n = 10$, $m = 1$.

We use three types of restricted Boltzmann machine. The input $x(k)$ data are encoded into: 1) binary input (DN_BI); 2) in the interval $[0, 1]$ (DN_NO); 3) in the interval $[-1, 1]$ (DN_NE). For the interval $[0, 1]$, $x(k)$ is normalized as

$$x(k) = \frac{X(k) - \min_k \{X(k)\}}{\max\{X(k)\} - \min_k \{X(k)\}} \tag{3.50}$$

We use 200 data to train the deep learning model. The structure parameters of the neural model, layer number p and node number of each layer l_i $(i = 1 \cdots p)$, are obtained by the random search method [7]. The results have shown that choosing 2 hidden layers $(p = 2)$ and $l_i = 20$ $(i = 1, 2)$, the model has an optimal configuration. The training rate for the restricted Boltzmann machine in (3.21) is $\eta_1 = 0.1$. The initial hidden weights of the restricted Boltzmann machine are selected in $[-1, +1]$ randomly. Table 3.1 shows how the restricted Boltzmann machines change the distributions of the hidden weights from $[-1, 1]$ to the other zones.

We define the squared error as

$$E = \frac{1}{N}\Sigma_{k=1}^{N} e^2(k), N = 1 \cdots 96 \tag{3.51}$$

The testing squared errors and the hidden weight distributions of the four models are shown in Table 3.1. 200 samples are used for training, $l_i = 20$, and $N = 96$ in (3.51).

Table 3.1: Testing results of deep learning with pseudoinverse (gas furnace)

	DN_BI	DN_NO	DN_NE	Random
Testing squared errors ($\times 10^{-4}$)	3.217	16.573	0.778	133.318
Hidden weight distributions	$[-0.493, 0.951]$	$[-1.21, 0.61]$	$[-0.316, 2.279]$	$[-1, 1]$

For this particular model, the deep learning modification decreases the modeling error almost 100 times by changing the hidden weight distributions from $[-1, 1]$ to $[-0.316, 2.279]$, here the input data are encoded in the interval $[-1, 1]$ (DN_NE).

The next experiment is to find general performance with respect to the change of the number of hidden neurons l_i in (2.8). The training data size is 200, the testing data size is 96. The testing results are shown in Figure 3-6. Both, randomized algorithm with deep learning modification and normal randomized algorithm improve the identification accuracy when the hidden node number is less than 25. Figure 3-6 shows that the randomized algorithm has a worst performance when the hidden node number is between 25 and 30, while the three deep learning methods have better performances when the hidden node number increases. When the input data are in the interval $[0, 1]$ (DN_NO), the deep leaning technique cannot improve the randomized algorithm effectively with a small hidden node number (less than 10). So the $[0, 1]$ encode method is not suitable for nonlinear system identification. However, when the input data are encoded as binary (DN_BI) or in the interval $[-1, 1]$ (DN_NE), deep learning can improve the randomized algorithm dramatically. As [28] states, RBM increases the likelihood of the output weights by using the input data.

The last experiment shows the relation of the training data number and the modeling error. The hidden node number is fixed as $l_i = 20$. The training data size is changed from 50 to 250. The simulation results are shown Figure 3-7. The deep learning modifications have better testing results, while the best one is DN_NE. This experiment shows the feature extraction capability of deep learning.

This improvement is more clear when the hidden node number l_i is increased. Even the $[0, 1]$ code (DN_NO) is better than the normal randomized algorithm. We can conclude that the pre-training stage is very effective for nonlinear system identification.

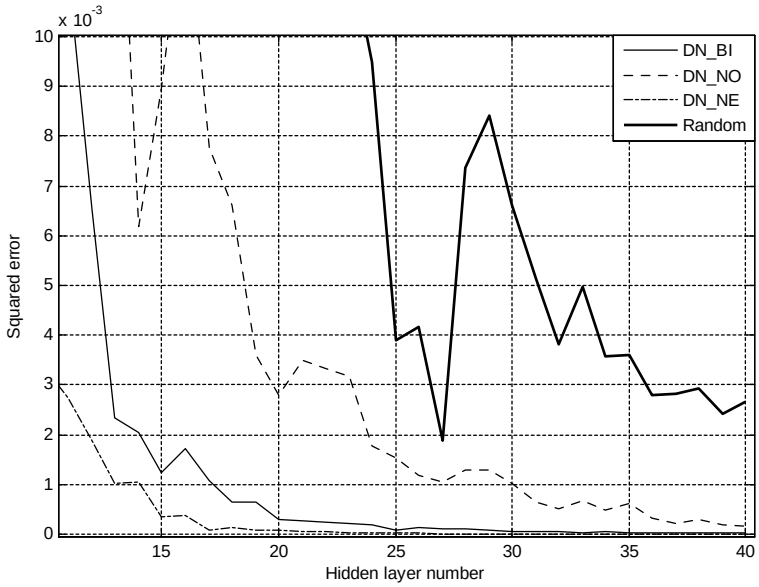

Figure 3-6: Testing errors vs hidden neuron number (gas furnace)

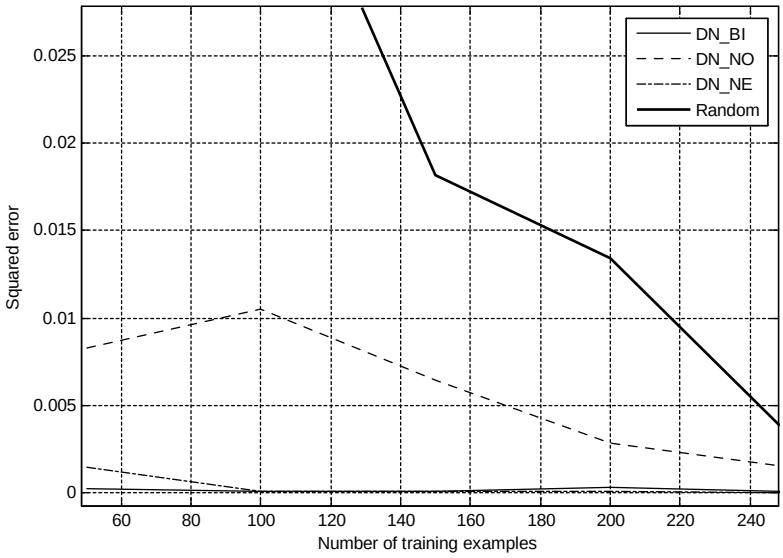

Figure 3-7: Testing squared errors vs training data number (gas furnace)

First order nonlinear system

This benchmark example was proposed in [56]. It is a simple nonlinear system,

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{3.52}$$

where $u(k)$ is a periodic input, which has different form in the training and the testing processes

$$u(k) = A\sin\left(\frac{\pi k}{50}\right) + B\sin\left(\frac{\pi k}{20}\right) \tag{3.53}$$

In the training stage, $A = B = 1$. In the testing stage, $A = 0.9$, $B = 1.1$.

The unknown nonlinear system (3.52) has the form of (3.2). $\bar{x}\,(k) = [y(k), u\,(k)]^{\mathrm{T}}$, $n = 2$, $m = 1$. The values of x and y are normalized to match the conditions of the restricted Boltzmann machine (3.50). The deep learning model is shown in Figure 3-5 and (2.8). Here we use two hidden layers $p = 2$. Similar with the gas furnace, we also use three types of encode methods for input data: DN_BI, DN_NO and DN_NE.

Similar with the above example, the initial hidden weights of the restricted Boltzmann machine are selected in $[-1, +1]$ randomly. Table 3.2 shows the testing squared errors and the hidden weight distributions of these models. Here the training data are 120 examples and the hidden node number is $l_i = 15$.

Table 3.2: Testing results of deep learning with pseudoinverse (first order nonlinear system)

	DN_BI	DN_NO	DN_NE	Random
Testing squared errors ($\times 10^{-3}$)	8.9	10.5	9.7	12.4
Hidden weight distributions	$[-0.369, 0.453]$	$[-1.387, 2.295]$	$[-0.137, 1.354]$	$[-1, 1]$

For this example we have the same conclusion as the gas furnace dataset. However, the binary encode method (DN_BI) for the input data is the best.

Then, we show the effectiveness of the hidden neuron number. They are drawn from the interval $10 \leq l_i \leq 30$. We use 80 data ($q = 80$) for training, and 100 samples ($N = 100$) for testing. The squared errors defined as (3.51) are shown in Figure 3-8. When the hidden nodes increase, the modeling errors become smaller, because the deep learning needs sufficient parameters to learn the probability distribution of the input. We can see that all encode

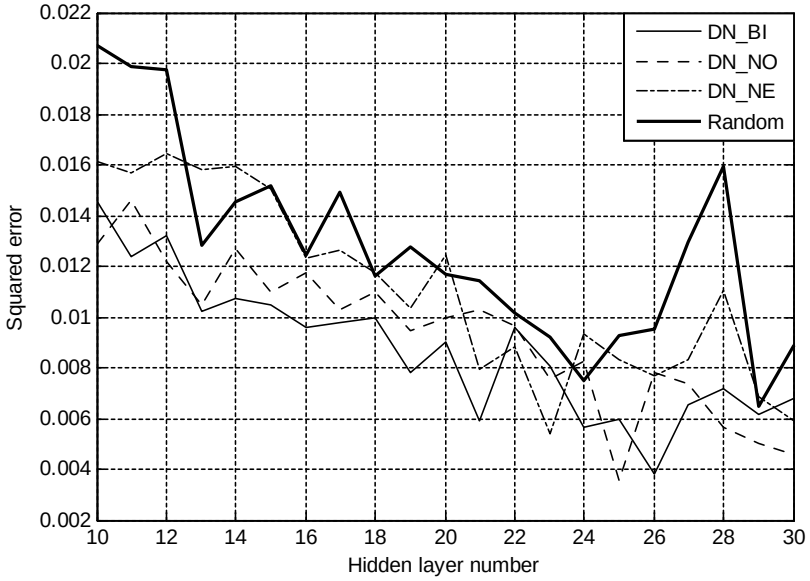

Figure 3-8: Testing errors vs hidden neuron number (first-order system)

methods of the deep learning modification are better than normal randomized algorithm (Random) for this example. This can be explained that the deep learning can guide the weights of the hidden layers into better regions of the parameter space with input data.

Finally, we use experiments to show the influence of the number of training examples. The number of training data is drawn from 80 to 200. The hidden node numbers are fixed in $l_i = 15$, $i = 1, 2$. Figure 3-9 shows the squared modelling error. We see that when the training data number is near 160, the normal randomized algorithm (Random) is better than the deep learning modification, because the self nature of the random selection can obtain good initial weights in some cases. Otherwise, the deep learning modification improves identification accuracy because it forces the neural model to a better parameter space region, and extracts enough features from the input.

Wiener-Hammerstein benchmark

A Wiener-Hammerstein system is a series connection of three parts: a linear system, a static nonlinearity and another linear system. The data of the Wiener-Hammerstein benchmark is generated from an electrical circuit which consists on three cascade blocks [64]. There is not direct measurement of the static nonlinearity, because it is located between two unknown

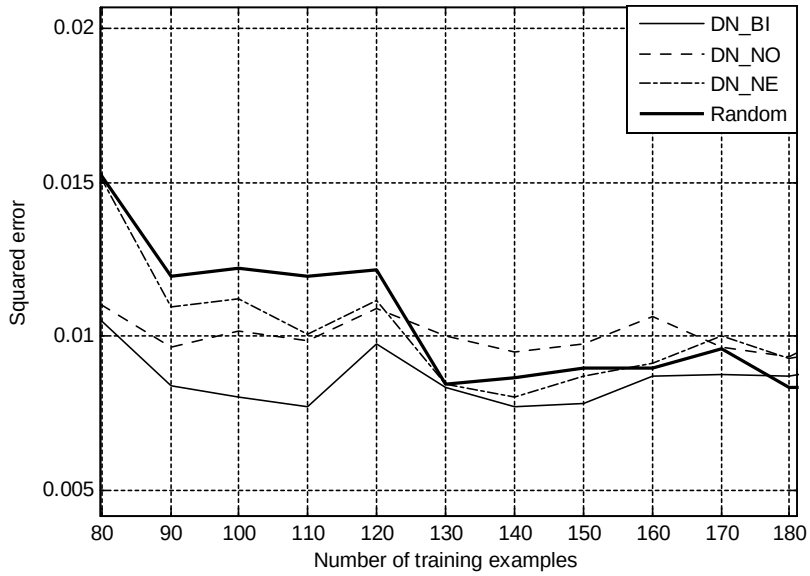

Figure 3-9: Squared errors vs training examples (first-order system)

linear dynamic systems.

The benchmark dataset consists of $188,000$ input/output pairs. This dataset is divided in two parts [29]: $100,000$ sample pairs are for the training stage and $88,000$ samples are for testing. Let $u(k)$ be the input and $y\,(k)$ be the output. We define the recursive input vector to the model as $X(k) = [y(k-1)\cdots y(k-4)\ u(k)\cdots u(k-5)]^{\mathrm{T}}$. So the Wiener-Hammerstein benchmark is

$$y\,(k) = f\,[y(k-1)\cdots y(k-4)\ u(k)\cdots u(k-5)] \tag{3.54}$$

Similar to the above two examples, $X(k)$ and $y\,(k)$ are also normalized as (3.50).

In this example, the deep learning modification has the same structure as normal randomized algorithms, *i.e.*, both of them have one hidden layer. We first test how the hidden node number affects the modeling error. The hidden node number l_1 is chosen from 100 to 500.

In this example, the input $x\,(k)$ data are coded into: 1) binary input (DN_BI); 2) in the interval $[0, 1]$ (DN_NO). The interval $[-1, 1]$ (DN_NE) for input data does not work well and is not reported. Table 3.3 shows the testing squared errors and the hidden weight distributions of these three models. Here, the training data are $50,000$ and the hidden node number is $l_i = 500$.

Table 3.3: Testing results of deep learning with pseudoinverse (Wiener-Hammerstein benchmark)

	DN_BI	DN_NO	Random
Testing squared errors ($\times 10^{-3}$)	2.639	2.819	2.724
Hidden weight distributions	$[-0.639, 0.621]$	$[-0.071, 0.003]$	$[-1, 1]$

We can see that the normal randomized algorithm (Random) is better in generalization results than the input encode method (DN_NO). For this example, the deep learning does not improve modeling accuracy with a good margin. It seems that the hidden weight distribution $[-1, 1]$ is suitable for the Wiener-Hammerstein benchmark problem.

The binary encode (DN_BI) has good modeling performance for this benchmark. Figure 3-10 gives the comparison results of the binary encoding (DN_BI) and the randomized algorithm. When the hidden nodes are chosen from 100 to 200, both models perform well for the Wiener-Hammerstein benchmark. After that, the deep learning modification is better because it needs more parameters to learn the probability distribution of the input. When the hidden nodes are 400, the deep learning modification becomes worse, this indicates that an overfitting problem of the neural model is happening.

Then we show how the number of training samples influences the modeling errors. The training examples are chosen between $10,000$ and $100,000$. The hidden nodes are fixed in 500. The squared modeling errors with respect to different training examples are shown in Figure 3-11. We can see that the errors decrease when the number of training examples increases for both methods. In the most cases, the deep learning modification is better.

Finally we compare these methods with a support vector machine (SVM) [21] and multilayer perceptrons with gradient learning algorithm (MLP) [56]. We use three types of kernels for SVMs: linear kernel (SVM-L), polynomial kernel (SVM-P), and radial basis function (SVM-R). In order to work well, the input recursive vector is modified as $X(k) = [y(k-1) \cdots y(k-10)\ u(k) \cdots u(k-5)]^T$. The squared modeling errors are shown in Table 3.4.

Compared with MLPs and SVMs, the normal randomized algorithm (Random) is much better for this benchmark, and deep learning can improve the modeling accuracy further.

Table 3.4: Comparison of MSE error with different learning techniques over the W-H benchmark($\times 10^{-3}$)

MLP	SVM-L	SVM-P	SVM-R	DN_BI	Random
56.03	43.01	6.01	4.71	2.65	2.82

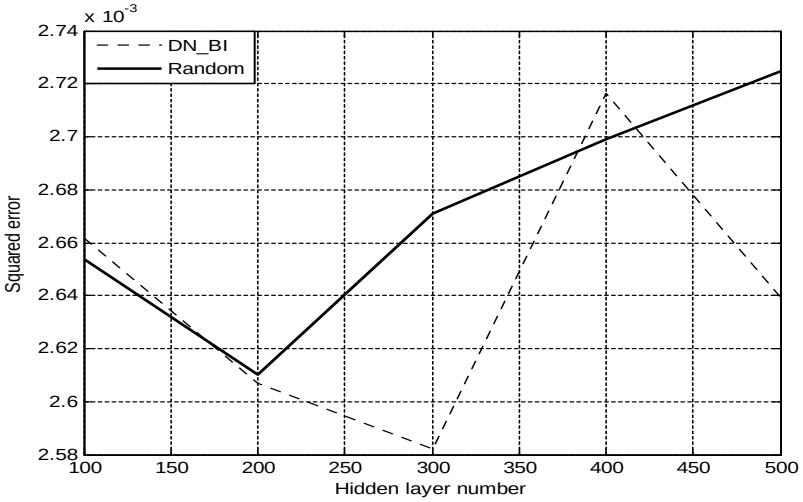

Figure 3-10: Binary encode (DN_BI) deep learning modification and the normal randomized algorithm. (W-H)

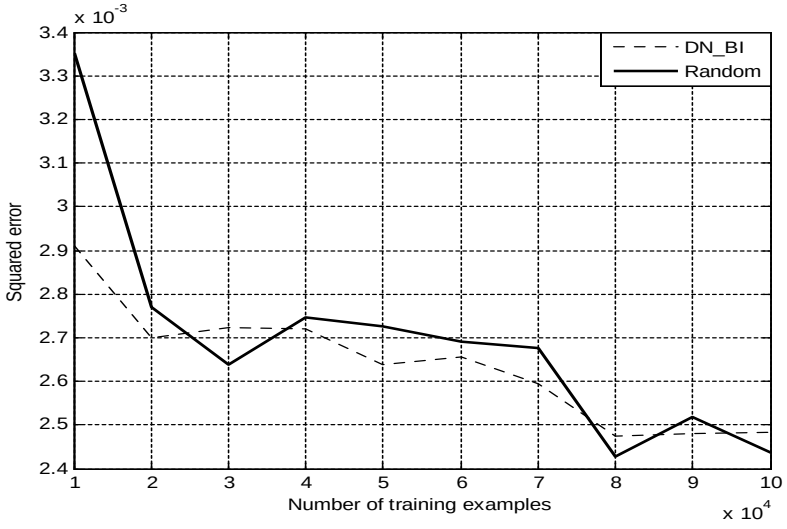

Figure 3-11: Squared modeling errors vs training examples (W-H)

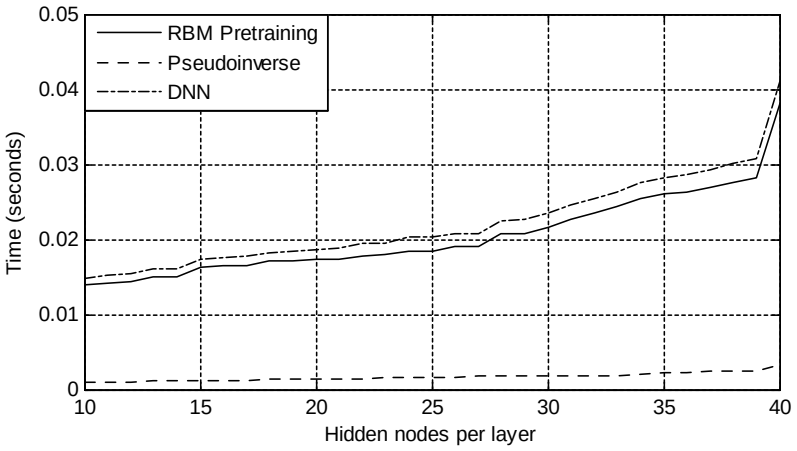

Figure 3-12: Training times for the gas furnace dataset with number of training examples $q = 150$

Computational complexity

The above three examples show that the identification accuracy increases with more hidden nodes and training examples. However, these also increases the training time of the neural models. Obviously, the computational complexity increases with the deep learning modifications, especially for online applications.

Both, random algorithms and deep learning techniques are batch processes. They cannot be applied for online updating for every input/output. Figure 3-12 shows the training time of the gas furnace for different hidden nodes. Here DNN is the total training time, DNN=RBM pretraining+Pseudoinverse. The training time of the deep learning modification (RBM pretraining) is significantly higher than the randomized algorithm (Pseudoinverse). One reason is that the input data are have higher dimensionality than the output, and RBM pretraining only uses input data. The training time is also affected by the other pre-training parameters, such as the number of epochs, learning rate, and stopping criterion.

For the second example, we obtain similar results, see Figure 3-13. Here the hidden layer number is $p = 2$. The pre-training time is similar as the one used in the Example 1, because the training number is similar. The computational time of DNN does not increase, because the size of the weight matrix β is the same.

For Example 3, the deep model just has one hidden layer $p = 1$. So the structure of the deep learning modification is the same as the randomized algorithm. The training time

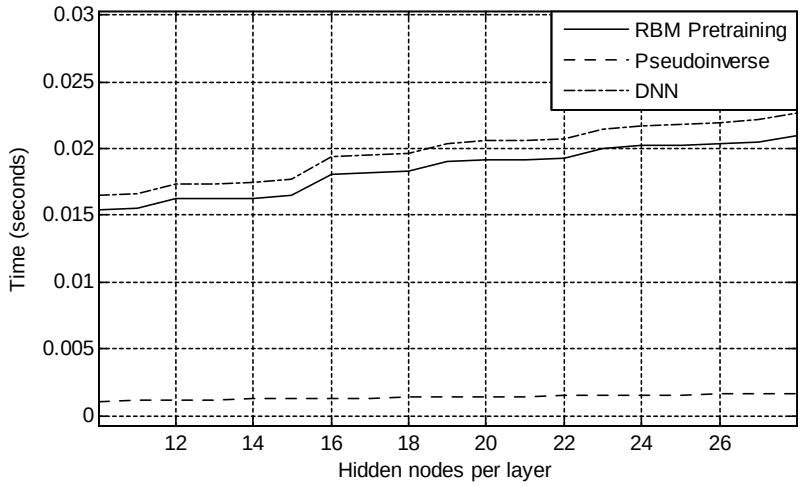

Figure 3-13: Training times for the first order system with number of training examples $q = 180$

is shown in Figure 3-14. We need more training time for this Wiener-Hammerstein model, because the training dataset has $50,000$ examples. When there are not many hidden nodes (less than 400), the training time of the deep learning modification does not increase, while the modeling accuracy is improved significantly.

>From the above three experiments, we see that the training time of the deep learning modification does not increase drastically for the randomized algorithm. Although adding the pre-training stage in the randomized algorithm increases computational complexity, the identification accuracy is improved, and the testing time for all methods is almost the same.

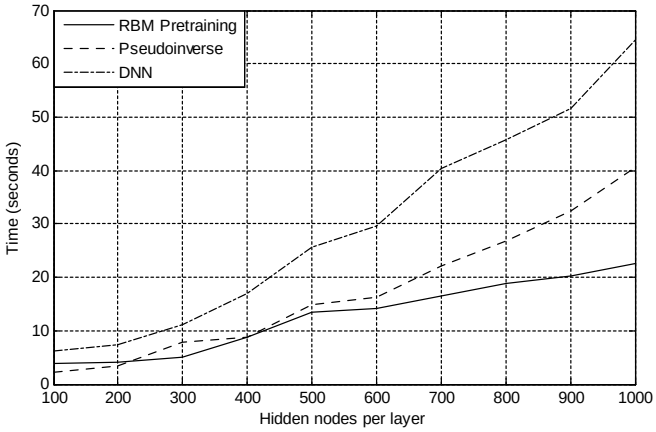

Figure 3-14: Training times for the W-H dataset with number of training examples $q = 50,000$

Chapter 4

Restricted Boltzmann machines and probabilistic fuzzy systems

In chapter 3, we have shown the effectiveness of deep learning when used with other methods for nonlinear system identification. The improvements can be explained analyzing the effects that the pretraining stage has over the identification model. These effects can be surveyed as:

1. RBMs guide the model parameters to regions of the configuration space where the model is more *likely* to reach a *good* local minimum.

2. The pretraining stage takes advantage of the statistical information that is hidden in the training data, it modifies the weights of the model to learn this variational features.

3. Unsupervised learning acts as a preprocessing stage where the training data is transformed into an encoded representation (in the case of an RBM this *code* corresponds to the hidden units). This *code* can be used by the output layer in the sense that it contains features of the input data that can be easily interpreted by a learning system.

The third effect is specially observed if we think about Ψ in (3.48) as a vector that contains features that the hidden layers have extracted from the input data. In this sense, deep learning helps to get meaningful features from the training dataset that can be used during the fine tuning stage. The success of the deep learning algorithms has been reported in Chapter 3 where the pretrained models surpassed the non-pretrained ones when both of them used as final layer a linear model which was solved finding the least square solution.

After RBMs have proved to be helpful in the identification framework, we explore, in this chapter, the consequences of having the data pretraining stage when using other identification algorithms such as fuzzy logic.

In order to use RBMs along with fuzzy rules we need to calculate the consequent part of the fuzzy model (the output weights). We simply apply the pseudoinverse calculation exposed in Section 3.4. In order to accomplish that, we consider the probability distributions computed by the RBMs as the hidden weights of a randomized model (the premise part of the fuzzy rules). In this sense, we interpret the fuzzy modeling as a randomized algorithm where the techniques used to train the second can also be implemented. Finally, we use an optimization method to reach maximum probability measures in each fuzzy rule. The proposed data-driven fuzzy modeling process is shown in Figure 4-1.

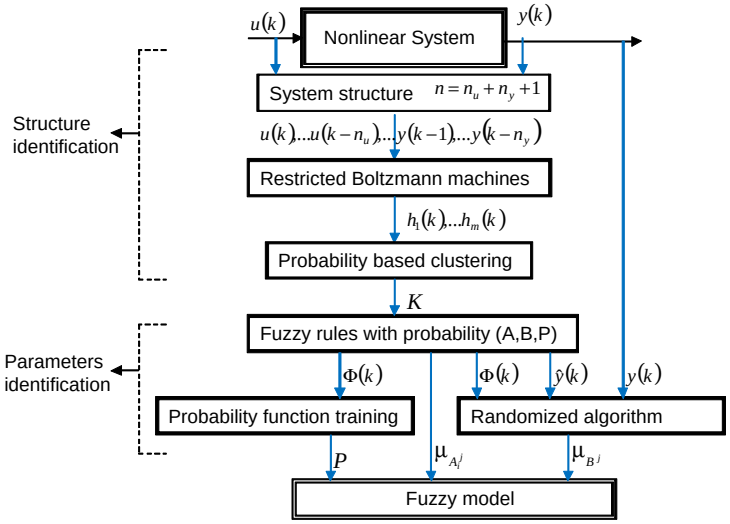

Figure 4-1: Data-driven fuzzy modeling

4.1 Data-driven deep fuzzy identification with restricted Boltzmann machines

In this chapter we aim again to identify a similar plant than the one described in (3.2). This plant can describe a wide family of discrete nonlinear systems whose behavior depends on previous states of their output. In this case we explicitly indicate the form of the input vector

in (4.1).

$$\mathbf{x}(k) = [y(k-1), y(k-2), \cdots y(k-n_y), u(k), u(k-1), \cdots u(k-n_u)]^T \tag{4.1}$$

where $u(k)$ and $y(k)$ are the measurable scalar input and output of the nonlinear plant, n_y and n_u correspond to the system order (or delays), $\mathbf{x}(k) \in \Re^n$ can be regarded as a new input to the nonlinear function $f(\cdot)$, with $n = n_y + n_u + 1$.

The objective of the fuzzy modeling is to use the input and output data set $[y(k), \mathbf{x}(k)]$ (or $[y(k), u(k)]$) of the nonlinear system (3.2), and construct a fuzzy model

$$\hat{y}(k) = F_{fuzz}[\mathbf{x}(k), k]$$

such that $\hat{y}(k) \to y(k)$, here $\hat{y}(k)$ is the output of the fuzzy model $F_{fuzz}[\cdot]$.

This data-driven modeling scheme needs two basic processes: structure identification and parameter identification. The structure identification consists in partitioning the input and output data of the nonlinear system and extract fuzzy rules.

As shown in Chapter 3, an RBM can learn the probability distribution among the input data, and obtain their hidden features which dramatically improves the performance of a regression model.

In this chapter of the thesis, we first use an RBM to transfer the input data to their feature space, and obtain the hidden features of the input. The RBM transformation allows us to model the system in the probability theory frame, such that the model is not sensitive to the noises and disturbances.

Consider an RBM as presented in Section 3.3. The input data of the RBM is $\mathbf{x}(k) = [x_1 \cdots x_n]$ and the hidden output of the RBM is $\bar{\mathbf{h}} = \left[\bar{h}_1 \cdots \bar{h}_m\right]$. In this chapter we consider a stand-alone RBM instead of a cascade (as proposed previously) where n is the dimension of the input and m is the dimension of the hidden layer. If $t = 1, ..., m$ and $s = 1, ..., n$, the $t-th$ hidden node and the $s-th$ visible node are sampled using the conditional distribution given by (3.19).

It is observed that we have changed the variable $\mathbf{h}$ into $\bar{\mathbf{h}}$. This modification was made in order to allow us to use $\mathbf{h}$ to represent another aspect of the RBM. Then, we define the

probability vector $\mathbf{h}$ as

$$\mathbf{h} = \left[p\left(\bar{h}_1 = 1 \mid \mathbf{x}\right) \cdots p\left(\bar{h}_m = 1 \mid \mathbf{x}\right)\right] = [h_1 \cdots h_m]$$

In this sense, we are considering the probability measures as the new output of the model, this is done because the sampling process takes away meaningful data from the probability vector, thus, instead of taking an already decided vector, we use the vector from where it was drawn from. The standard RBM model requires that both $\bar{h}_s$ and x_t to be binary values but, for nonlinear system identification, the visible units $\mathbf{x}$ cannot be binary values. The solution that we used to overcome this problem is presented as follows:

Consider the RBM parameters gathered as $\Lambda = [V, b, c]$ where V is the weight matrix between the input and hidden layers, b is the bias that interacts with the raw input and c is the hidden bias that works to sample new visible vectors from the learned distribution.

We have said that nonlinear system identification demands that the RBM is capable of dealing with continuous valued input. Following that thinking, we use the modified RBM whose conditional probability distribution $P_C(x_t|h)$ is given by (3.30) which makes the strong assumption that $\mathbf{x}$ is normalized.

When $x_t \in [0, 1]$, from (3.31), the probability distribution with $a_t = V_t^T \overline{h} + c_t$ is,

$$P(x_t|\bar{\mathbf{h}}) = \frac{a_t e^{a_t x_t}}{e^{a_t} - 1} \tag{4.2}$$

And from (3.31) the cumulative conditional probability from where a sampling process can be made is computed by

$$P_C(x_t|\bar{\mathbf{h}}) = \frac{e^{a_t x_t} - 1}{e^{a_t} - 1} \tag{4.3}$$

We use the data set, $\mathbf{x}(k) \in D_1$ (training set), $k = 1 \cdots M$, to train the RBM. The training procedure obeys the same steps given by Algorithm 4 and the gradient descent minimization in (3.21).

After the RBM is trained, the parameters Λ are fixed. Then we use another data set, $\mathbf{x}(k) \in D_2$, $k = 1 \cdots q$, to compute the data-driven fuzzy modeling where q is the number of training examples. Now the RBM transforms the input data into their hidden feature space.

It is a common practice that the dataset D_1 and D_2 are the same as they share the same statistical properties, this is done in order to take full advantage of the deep learning encoding

that the RBM is performing. Nevertheless, this practice (although it seems to be convenient) have some flaws because of the lack of representativeness that the datasets present when they try to provide the model with sufficient statistical information.

Once the RBM has been trained we perform a probabilistic clustering method Because the features of the input data are in the form of probability distributions, we use the following probability based clustering method to obtain the fuzzy rules.

4.2 Fuzzy rules extraction with probability based clustering

The input data $\mathbf{x}(k) \in D_2$ are mapped to the hidden features $H = \{\mathbf{h}(k)\}_{k=1}^{q}$ by the trained RBM. We assume that each sample $\mathbf{h}(k)$ belongs to a specific cluster whose labels are given by $L = \{l(k)\}_{k=1}^{q}$, $l(k) \in \{1, ..., K\}$, where K is the number of clusters. The object of the probability based clustering is to find the correlation between the input instances and their respective cluster parameters. The higher correlation between an instance and a cluster, the more possible it will be assigned to that cluster. We use the following objective function, which is similar as [24],

$$P(L, \{\delta_j\}_{j=1}^{K}|H) \propto p(L) \left[\prod_{k=1}^{N} p(\mathbf{h}(k)|\delta_{l(k)})\right] \prod_{j=1}^{K} p(\delta_j) \tag{4.4}$$

where $p(L)$ is the marginal clustering distribution probability, δ_j are the clustering model parameters, $p(\mathbf{h}(k)|\delta_{l(k)})$ is the likelihood of the hidden code $\mathbf{h}(k)$, $\delta_{l(k)}$ is the cluster parameter, $p(\delta_j)$ is the Gaussian prior for all δ_j with $j = 1...K$.

The parameters $\{\delta_j\}_{j=1}^{K}$ are estimated by the following Gibbs sampling with respect to the label $l(k)$ and hidden feature $\mathbf{h}(k)$. Given the set of codes $H = \{\mathbf{h}(k)\}_{k=1}^{q}$ and its cluster labels L, the Gibbs sampling allow us to obtain samples from the conditional probability distribution while keeping other variables fixed. So for each label $l(k)$, the conditional posterior is

$$p\left[l(k) = j|l(-k), \mathbf{h}(k), \{\delta_j\}_{j=1}^{K}, \alpha, \psi, \lambda\right] \propto p\left[l(k) = j|l(-k), \alpha, \psi\right] p\left[\mathbf{h}(k)|\delta_j\right] \tag{4.5}$$

where $l(-k)$ denotes all other indices but k.

$p\left[l(k)=j|l(-k),\alpha\right]$ is determined by a Chinese restaurant process with concentration parameter α and discount parameter ψ. The probability of each cluster given by the Chinese restaurant process is calculated as follows: suppose that we have K different clusters at time $k+1$, then $\mathbf{h}(k)$ would be assigned at an empty new cluster G_{K+1} with probability $\frac{\psi+K\alpha}{k+\psi}$. For an existing cluster G_j with n_j existing elements, the probability is $\frac{n_j-\alpha}{k+\psi}$.

$p(\mathbf{h}(k)|\delta_j)$ is the likelihood of the current instance k and $\mathbf{h}(k)$ in its cluster. It is directly proportional to the correlation between $\mathbf{h}(k)$ and δ_j. It can be calculated as $\mathbf{h}(k)^T\delta_j$. Taking into account a weight penalization $\lambda\left\|\delta_j\right\|^2$, it can also be calculated as

$$p(\mathbf{h}(k)|\delta_j) \propto \exp(\mathbf{h}(k)^T\delta_j - \lambda\left\|\delta_j\right\|^2) \tag{4.6}$$

where λ is a penalization constant to control the weights size, $\lambda\left\|\delta_j\right\|^2$ represents the maximum margin to separate clusters [24].

(4.6) is regarded as a set of exponential functions, which have similar statistical properties. Substituting the assumption (4.5) into (4.6),

$$p\left[l(k)=j|l(-k),\mathbf{h}(k),\{\delta_j\}_{j=1}^K,\alpha,\lambda\right] \propto p\left[l(k)=j|l(-k),\alpha\right]\exp(\mathbf{h}(k)^T\delta_j - \lambda\left\|\delta_j\right\|^2) \tag{4.7}$$

A larger correlation between $\mathbf{h}(k)$ and δ_j indicates a higher probability that $\mathbf{h}(k)$ belongs to cluster G_j. If the probability is less than a probability threshold, a new virtual cluster G_{K+1} with random parameters δ_{K+1} is generated, $K=K+1$.

$\mathbf{h}(k)$ is assigned into this new cluster. The probability of a new cluster is calculated by the Chinese restaurant process. The correlation is calculated by (4.7). δ_{K+1} is drawn from a multi-variate t-distribution. So the clustering object is to maximize (4.4) as

$$\max\left\{p(L)\left[\prod_{k=1}^{N}p(\mathbf{h}(k)|\delta_{l(k)})\right]\prod_{j=1}^{K}p(\delta_j)\right\} \tag{4.8}$$

The probabilities $p(\delta_j)$ are calculated by the following maximum margin learning rule. The maximum margin learning rule uses the passive aggressive algorithm (PA) to update the cluster parameters [18]. At time k, the label $l(k)$ is determined by the Gibbs sampling process described in (4.7).

We concatenate the cluster parameters $\{\delta_j\}_{j=1}^K$ as a vector $\Delta=[\delta_1,...,\delta_K]$, or $\Delta^{l(k)}=\delta_{l(k)}$.

If we define the concatenating vector $\Phi\left[\mathbf{h}(k), l(k)\right]$ where the $l(k)-th$ element is set to be $\mathbf{h}(k)$, while the others are set to be vectors 0 we calculate at time k the margin vector $\Delta(k)$ as:

$$M\left[\Delta(k);(\mathbf{h}(k), l(k))\right] = \Delta(t)\cdot\Phi\left[\mathbf{h}(k), l(k)\right] - \Delta(k)\cdot\Phi\left[\mathbf{h}(k), \widehat{l}(k)\right] \tag{4.9}$$

where $\widehat{l}(k)$ is the prediction label from the model and $\mathbf{h}(k)$,

$$\widehat{l}(k) = \arg\max_{j} \mathbf{h}(k)^T \delta_j \tag{4.10}$$

The updating process is designed to optimize the following objective function

$$\begin{aligned} \Delta(k+1) &= \arg\min_{\Delta} \tfrac{1}{2}\left\|\Delta - \Delta(k)\right\|^2 + C\xi \\ \text{Subject to:} \quad & l_2\left[\Delta;(\mathbf{h}(k), l(k))\right] \leq \xi \end{aligned} \tag{4.11}$$

where $C > 0$ is a penalty constant, ξ is the threshold of the hinge-loss function, $l_2\left[\cdot\right]$ is the hinge-loss function defined by

$$l_2\left[\Delta(k);(\mathbf{h}(k), l(k))\right] = \begin{cases} 0 & \text{if } M\left[\Delta(k);(\mathbf{h}(k), l(k))\right] \geq 1 \\ 1 - M\left[\Delta(k);(\mathbf{h}(k), l(k))\right] & \text{otherwise} \end{cases} \tag{4.12}$$

where $M\left[\cdot\right]$ is the margin function (4.9).

Using the passive aggressive algorithm [18], the parameters are updated as

$$\begin{aligned} \Delta^{l(k)}(k+1) &= \Delta^{l(k)}(k) + \tau(k)\mathbf{h}(k) \\ \Delta^{\widehat{l}(k)}(k+1) &= \Delta^{\widehat{l}(k)}(k) - \tau(k)\mathbf{h}(k) \end{aligned} \tag{4.13}$$

where $\tau(k) = \min\{C, \frac{hl[\Delta(k);(\mathbf{h}(k),l(k))]}{\|\mathbf{h}(k)\|^2}\}$.

For each iteration k, δ_k is estimated by (4.13), (4.12), and (4.9), such that the maximum margin increases. This probability based clustering is similar to the nonparametric maximum margin clustering [24]. However, the goal in this thesis is to identify a nonlinear system which leads to a time series clustering algorithm that can be applied on-line.

After the training set D_2 has been divided, we have K different clusters G_j, $j = 1\cdots K$.

We assign one fuzzy rule for each cluster G_j as

$$\mathrm{R}^j\text{: IF } h_1(k) \text{ is } A_1^j \text{ and } h_2(k) \text{ is } A_2^j \text{ and } \cdots h_m(k) \text{ is } A_m^j \text{ THEN } y(k) \text{ is } B^j \tag{4.14}$$

where $A_1^j, \cdots A_m^j$ and B^j are standard fuzzy sets, they are represented by the following Gaussian membership functions

$$\mu_{A_s^j}\left[h_s(k)\right] = \exp\left(-\frac{\left[h_s(k) - c_{js}\right]^2}{\sigma_{js}^2}\right) \tag{4.15}$$

where $k = 1 \cdots q$, $s = 1 \cdots m$, $j = 1 \cdots K$.

By using product inference, center-average and singleton fuzzifier, the output of the fuzzy system is expressed as [74]

$$\hat{y}(k) = \left(\sum_{j=1}^{K} w_j \left[\prod_{s=1}^{m} \mu_{A_s^j}\right]\right) / \left(\sum_{j=1}^{K} \left[\prod_{s=1}^{m} \mu_{A_s^j}\right]\right) \tag{4.16}$$

where w_j is the point at which $\mu_{B^j} = 1$. If we define $\phi_j = \prod_{s=1}^{m} \mu_{A_i^j} / \sum_{j=1}^{K} \prod_{s=1}^{m} \mu_{A_s^j}$, (4.16) can be expressed in matrix form

$$\hat{y}(k) = \mathbf{W}(k)\, \Phi\left[\mathbf{h}(k)\right] \tag{4.17}$$

with parameters $\mathbf{W}(k) = [w_1 \cdots w_K]$ and data vector $\Phi\left[\mathbf{h}(k)\right] = [\phi_1 \cdots \phi_K]^T$.

From the restricted Boltzmann machine, we obtain the hidden features $h_s(k)$ and their dimension m. From the probability based clustering, we obtain the fuzzy rule number K and the data distributions. So the structure of the fuzzy model is ready. The fuzzy rules extraction with the on-line clustering and the probability based clustering is shown in Figure 4-2.

The probability based clustering not only gives the distribution of the data $h_s(k)$, but also provides the relations of the data in probability forms. The fuzzy rule (4.14) only represents the data distribution. In order to include the flexibility of this probability relation in the

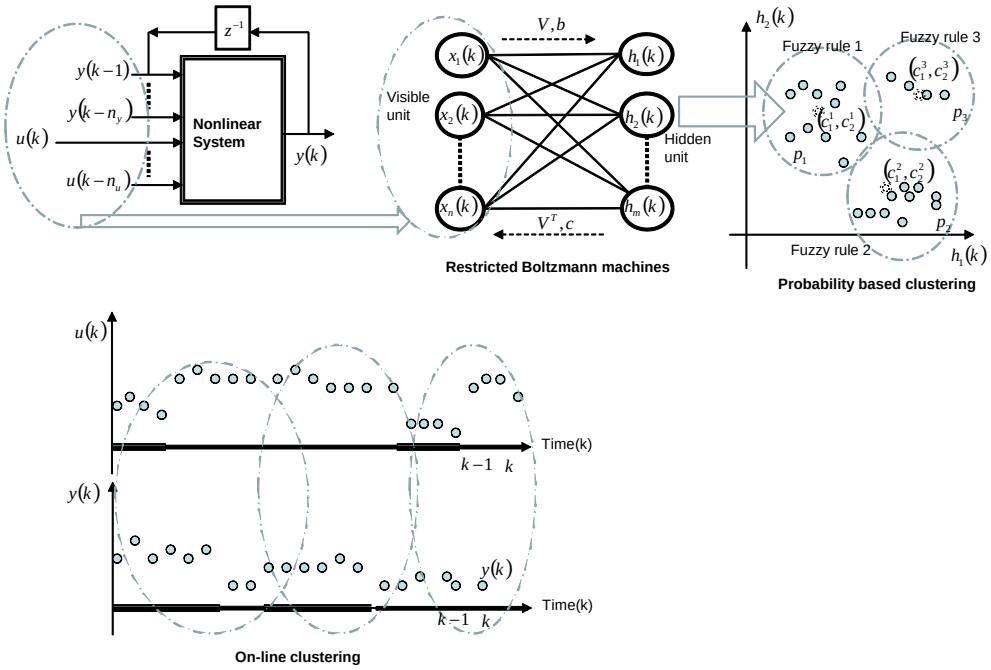

Figure 4-2: Fuzzy rules extraction with the on-line culstering and the probability based clustering

data, we assign probability factors $p_{j,i}$ into each rule (4.14) as

$$\begin{aligned} \mathrm{R}^j\text{: IF } h_1(k) \text{ is } A_1^j \text{ and } h_2(k) \text{ is } A_2^j \text{ and } \cdots h_m(k) \text{ is } A_n^j \text{ THEN} \\ y(k) \text{ is } B^1 \text{ with prob. } p_{j,1} \text{ and} \\ y(k) \text{ is } B^2 \text{ with prob. } p_{j,2} \text{ and} \\ \ldots \\ y(k) \text{ is } B^K \text{ with prob. } p_{j,K} \end{aligned} \tag{4.18}$$

where $p_{j,i} \geq 0$, $\sum_{i=1}^{K} p_{j,i} = 1$ with $i, j = 1, ..., K$. This means the consequent $y(k)$ is established in the probability given by $p_{j,i}$. So the fuzzy set of the consequent, B^j, must satisfy

$$p(B^j|\mathbf{h}(k)) = \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)] p_{i,j} \tag{4.19}$$

4.3 Data-Driven Fuzzy Modeling

The fuzzy model of the probability based fuzzy rules is not longer (4.17). We use the following process to extract the fuzzy model from the feature space $\mathbf{h}(k)$. $\phi_j[\mathbf{h}(k)]$ in (4.19), it can be regarded as a normalized vectorial membership function of $\mathbf{h}(k)$ to the fuzzy sets $A_1^j, A_2^j, \cdots, A_m^j$, $p(y|\mathbf{h}(k))$ is calculated by

$$p(y|\mathbf{h}(k)) = \sum_{j=1}^{K} p(y|B^j)p(B^j|\mathbf{h}(k)) \tag{4.20}$$

where $p(y|B^j)$ is estimated as

$$p(y|B^j) = \frac{\mu_{B^j}(y)}{\int \mu_{Bj}(y)dy} \tag{4.21}$$

This is a probability measurement of the membership function μ_{B^j}. The output of the probability based fuzzy model is

$$\widehat{y}(k) = \mathrm{E}(y|\mathbf{h}(k)) = \int yp(y|\mathbf{h}(k))dy = \sum_{j=1}^{K} p(B^j|\mathbf{h}(k))\mathrm{E}(y|B^j) \tag{4.22}$$

where $\mathrm{E}(y|B^j) = \frac{\int y\mu_{Bj}(y)dy}{\int \mu_{Bj}(y)dy}$. The last term is just the centroid of the fuzzy set B^j.

Compared with the standard fuzzy model (4.17), where w_j is the point at which $\mu_{B^j} = 1$, (4.22) can be formed as

$$\widehat{y}(k) = \sum_{j=1}^{K} w_j p(B^j|\mathbf{h}(k)) = \sum_{j=1}^{K}\sum_{i=1}^{K} \phi_i[\mathbf{h}(k)]\, p_{i,j} w_j \tag{4.23}$$

or

$$\hat{y}(k) = \mathbf{W}(k)\,\Phi[\mathbf{h}(k)] \tag{4.24}$$

where the parameter $\mathbf{W}(k) = [w_1 \cdots w_K]$ and the data vector is given by

$$\Phi[\mathbf{h}(k)] = \left[\sum_{i=1}^{K} \phi_i[\mathbf{h}(k)]\, p_{i,1} | \cdots | \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)]\, p_{i,K}\right]^T$$

4.3.1 Randomized algorithms for membership functions training

For the probability based fuzzy model (4.24), $\Phi\left[\mathbf{h}\left(k\right)\right]$ is determined by a restricted Boltzmann machine and probability based clustering as we presented above. (4.24) is a linear-in-parameter system, the parameter $\mathbf{W}\left(k\right)$ may be singular and/or be not square, the solution can be solved by the Moore-Penrose generalized inverse as explained in Section 3.4. The pseudoinverse calculation is performed as follows:

For a linear system $\hat{y}\left(k\right)=\mathbf{W}\Phi$, $\mathbf{W}_0$ is a least-squares solution if

$$\left\|\mathbf{W}_0\Phi-y\left(k\right)\right\|=\min_{W}\left\|\mathbf{W}\Phi-y\left(k\right)\right\| \tag{4.25}$$

where $\left\|\cdot\right\|$ is a norm in Euclidean space. If By is a minimum norm least-squares solution of the linear system $\hat{y}=\mathbf{W}\Phi$, then it is necessary and sufficient that $B=\Phi^{+}$. Here Φ^{+} is the Moore-Penrose generalized inverse of matrix Φ, which is defined in (3.41).

For our fuzzy model, the goal of the training algorithm is to find the parameter $\mathbf{W}\left(k\right)$ such that the following cost function is minimized

$$J=\sum_{k}\left\|y\left(k\right)-\widehat{y}\left(k\right)\right\|^{2} \tag{4.26}$$

where $\Psi=\left[\Phi\left(1\right),\Phi\left(2\right),\cdots,\Phi\left(q\right)\right]$. Or in another form:

$$Y=\left[\begin{array}{cccc} y\left(1\right) & y\left(2\right) & \cdots & y\left(N\right)\end{array}\right]=\left[\begin{array}{cccc} \mathbf{W}\Phi\left(1\right)+e(1) & \mathbf{W}\Phi\left(2\right)+e(2) & \cdots & \mathbf{W}\Phi\left(N\right)+e(N)\end{array}\right] \tag{4.27}$$

$$Y=\mathbf{W}\Psi+E \tag{4.28}$$

where $e\left(k\right)$ is the modeling error $e(k)=y\left(k\right)-\widehat{y}\left(k\right)$, and $E=\left[e\left(1\right),e\left(2\right),\cdots,e\left(N\right)\right]$. To obtain $\min\limits_{\beta}J$, we need $\frac{\partial J}{\partial\mathbf{W}}=0$. From (3.43)

$$\mathbf{W}^{*}=Y\Psi^{T}\left(\Psi\Psi^{T}\right)^{-1}=Y\Psi^{+} \tag{4.29}$$

So $\mathbf{W}^{*}$ can minimize the index J in (4.26).

Since $\mathbf{W}^{*}$ is one of the least-squares solutions of the system $Y=\mathbf{W}\Psi+E$, it reaches the

smallest approximation error on the training dataset, and it is unique. The solution $\mathbf{W}^*$ has the smallest norm for a least-squares solution of $Y = \mathbf{W}\Psi$. [63] shows that for feedforward networks, small norm of the weights is more important than the number of nodes to obtain a small generalization error.

Although random weights in the hidden layers are better than backpropagation training in many cases, sometimes random weights may lead to poor performances [33]. The restricted Boltzmann machine and the probability based clustering provide possible selection manners of hidden weights with the distribution of the input data. The distributions of the random hidden weights are defined in advance to improve the modeling accuracy.

For the fuzzy model, the premise membership functions $A_1^j, \cdots A_m^j$ are given by the probability based clustering. A_s^j is in the form of a Gaussian function (4.15). Its two parameters c_{js} and σ_{js} are determined as:

- The terms c_{js} are selected as equal as the center of each cluster
- The parameters σ_{js} are assigned randomly in$(0, 1)$

As we do not have the values of $p_{j,i}$ we cannot calculate W. We set the parameters $p_{j,i} = 1$ for $i = j$ and $p_{j,i} = 0$ for $i \neq j$ which reduces the probabilistic model (4.18) into the model (4.14). With this consideration we can compute W. The next section shows how to estimate the probability parameters $p_{j,i}$.

4.3.2 Probability functions training

The purpose of training the probabilities $p_{j,i}$ of each fuzzy rule (4.18) is to maximize the likelihood of the desired output with respect to its input. From (4.19) and (4.20), the parameters $p_{i,j}$ satisfy

$$p(y|\mathbf{h}(k)) = \sum_{j=1}^{K} p(y|B^j) \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)] p_{i,j} \tag{4.30}$$

Because $p_{i,K} = 1 - \sum_{j=1}^{K-1} p_{i,j}$,

$$p(y|\mathbf{h}(k)) = \sum_{j=1}^{K-1} p(y|B^j) \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)] p_{i,j} + p(y|B^K) \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)] \left(1 - \sum_{j=1}^{K-1} p_{i,j}\right) \tag{4.31}$$

Then the global log-likelihood function of the training set D such that $\{\mathbf{h}(k), y(k)\} \in D$ is

$$\begin{aligned}\mathcal{L}(D,P) &= \log\left(\prod_{k=1}^{N} p(y(k)|\mathbf{h}(k))\right) = \sum_{k=1}^{N} \log p(y(k)|\mathbf{h}(k)) \\ &= \sum_{k=1}^{N} \log \left[\begin{array}{c} \sum_{j=1}^{K-1} p(y|B^j) \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)]\, p_{i,j} + \\ p(y|B^K) \sum_{i=1}^{K} \phi_i[\mathbf{h}(k)] \left(1 - \sum_{j=1}^{K-1} p_{i,j}\right) \end{array}\right]\end{aligned}$$

where P is a $K \times K$ dimension matrix which contains the probability parameters $p_{j,i}$,

$$P = \left[\begin{array}{ccc} p_{1,1} & \cdots & p_{1,K} \\ \vdots & \ddots & \vdots \\ p_{K,1} & \cdots & p_{K,K} \end{array}\right] \tag{4.32}$$

The fuzzy set B^j has the form of a Gaussian function (4.15) with $c_j = w_j$,

$$\mu_{B^j}(y(k)) = \exp\left(-\frac{(y(k) - c_j)^2}{\sigma_{B^j}^2}\right)$$

By using $\int \mu_{B^j}(y)dy = \sqrt{\pi}\sigma_{B^j}$, we can evaluate $p(y(k)|B^i)$.

In order to obtain P, we need to solve the following minimization problem

$$\left\{\begin{array}{c} \min_P \{-\mathcal{L}(D,P)\} \\ \text{Subject to} \quad p_{i,j} > 0 \forall\ i,j \text{ and } \sum_{j=1}^{K-1} p_{i,j} \leq 1 \end{array}\right. \tag{4.33}$$

Here we do not use the last column $p_{i,K}$ of P, because it is calculated as a consequence of the rest of the values of P.

The minimization (4.33) can be formed into the following linear programming program as

$$\left\{\begin{array}{c} \min_{P_v} -\mathcal{L}(D,P_v) \\ \text{Subject} \quad AP_v \leq b \text{ and } l_b \leq P_v \end{array}\right. \tag{4.34}$$

where

$$P_v = [p_1, \cdots, p_{K-1} | \cdots | p_K, \cdots, p_{K,K-1}]^T$$

$$A = \left[\begin{array}{cccc} \overrightarrow{1} & \overrightarrow{0} & \cdots & \overrightarrow{0} \\ \overrightarrow{0} & \overrightarrow{1} & \cdots & \overrightarrow{0} \\ \vdots & \vdots & \ddots & \vdots \\ \overrightarrow{0} & \overrightarrow{0} & \cdots & \overrightarrow{1} \end{array}\right]$$

$\overrightarrow{0}$, $\overrightarrow{1}$ $\in \mathbb{R}^{K-1}$ are row vectors with $\overrightarrow{0}$ $= [0...0]$ and $\overrightarrow{1}$ $= [1...1]$, $l_b, b \in \mathbb{R}^{K-1}$ such that $b = [1...1]^T$ and $l_b = [0...0]^T$. The minimization problem of (4.33) is solved by a standard linear programming toolbox of Matlab.

4.4 Comparisons with other fuzzy modeling methods

In this section, we use two benchmark examples to show the effectiveness of our data-driven fuzzy modeling method which combines restricted Boltzmann machines, probability based clustering, and probability fuzzy rules.

Gas furnace fuzzy modeling

We use the same dataset as in Section 3.5. Here we aim to identify the next general model:

$$y(k) = f[y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u)] = f\left[\mathbf{x}\left(k\right)\right]$$

where n_y and n_u are the regression delays for input and output.

Here, we also use the random search method [7][17] to decide the best n_y and n_u. The regression delays are assumed in the interval $[1, 10]$, the training data are 200 examples while the rest are used for validation. Finally, we have $n_y = 4$, $n_u = 5$.

The dataset is first normalized using (3.50) for comparison purposes. In this example, the data-driven fuzzy modeling has the following four steps:

1. Features extraction. The normalized input data are sent to an RBM: The contrastive divergence uses 1-step Gibbs sampling and 10 training epochs, the learning rate is $\eta = 0.2$. After the training, the parameters of the RBM V and b are then used to compute the hidden representation of the model ($\mathbf{h}$). The number of hidden units is chosen as $n_y + n_u + 1$, such that the hidden and the visible unit numbers are the same.

2. Clustering. After the features are extracted by the RBM, we used the probability based clustering. The hyper parameters are chosen as $\alpha = 0.8$, $\psi = 10$, $\lambda = 5$ and $C = 0.001$. Here α and ψ determine the probabilities which are obtained by the Chinese restaurant process. α is close to 1. When ψ increases, the number of clusters K also grows. The penalization parameter λ decreases the probability of the cluster, while keeps $\|\delta_j\|^2$ low

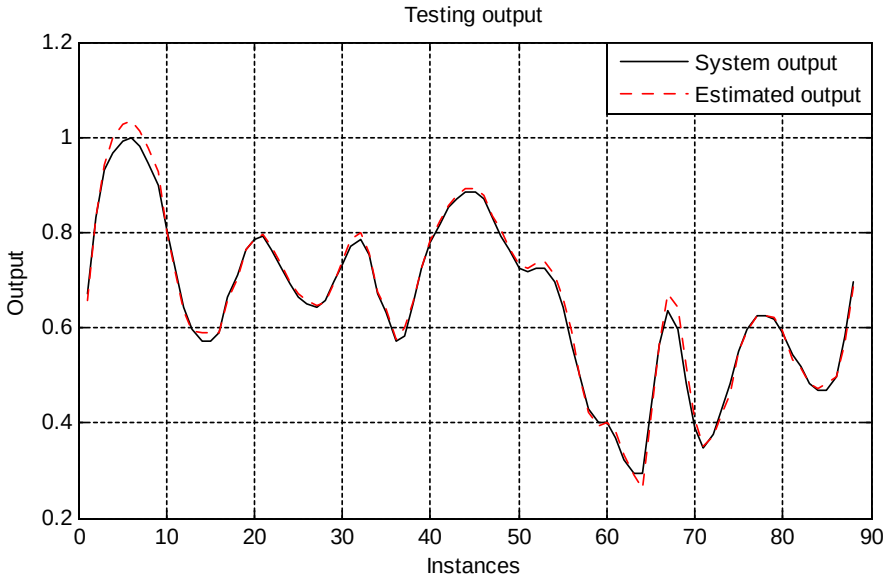

Figure 4-3: Testing results of the gas furnace modeling.

. In our experiments, the probability based clustering divided the dataset $\mathbf{h}(k)$ into 10 clusters. Without the RBM, the same clustering method extracts 12 clusters from the original data $\mathbf{x}(k)$.

3. Membership functions training. In order to improve modeling accuracy, the membership functions of the fuzzy model are updated with the input and output data. The centers of the membership function are the cluster centers which are obtained in Step 2. The parameters W are computed using the ELM approach (we calculate the pseudoinverse using a vector which contains the parameters Φ).

4. Probability training. Once the minimization problem is set, The probability parameters $p_{i,j}$ are estimated by the standard linear programming toolbox, "fmincon" and "sqp". The initial value of the matrix P is the identity matrix $\mathbf{I}_K$, *i. e.*, we start from a standard fuzzy rule and the probability parameters are introduced to minimize the possibility of the modeling error procedure.

In order to test the generalization capabilities of our model, we use the remaining 96 data for testing after the training phase is finished. The final testing results are shown in Figure 4-3.

We compared our method with the following three fuzzy modeling algorithms:

1. Adaptive fuzzy modeling approach (ANFIS) [74]. It may be the most popular fuzzy modeling method. In this experiment, we also use 8 fuzzy rules. The Gaussian membership functions are selected randomly at first.

2. Fuzzy modeling via online clustering [36][71][3]. Here we do not consider the temporal interval problem [77] and use all data to train each group. All thresholds for the output and the input are 1.5. Finally ,we obtain five fuzzy rules.

3. Fuzzy logic with data clustering [54][16]. It is another popular fuzzy modeling method. In this comparison, only the input is partitioned. With the threshold 1.0, we have 15 groups in the input space. So 15 fuzzy rules are constructed.

The root mean square (RMS) testing error for each method is $RMS_1 = 0.019$ (our fuzzy modeling with RBMs), $RMS_2 = 0.031$ (fuzzy modeling with clustering) and $RMS_3 = 0.09$ (ANFIS).

In order to show the effectiveness of the hidden feature extraction with RBMs, we compare the testing error of the $\mathbf{h}(k)$ clustering after RBMs and the $\mathbf{x}(k)$ clustering without RBMs. Figure 4-4 gives these testing errors.

It is observed that the clustering procedure using the features from the RBM gives better representation for the input data. Once the fuzzy rules are trained, the hidden features can be observed by the RBM, and the probabilistic fuzzy model improves the modeling accuracy.

Now, we discuss how the probability parameters work in the consequences of the fuzzy rules (4.18). Figure 4-5 shows the training errors with standard fuzzy rules and probabilistic fuzzy rules. We see that the probabilistic parameters give more freedom and robustness to adjust the model with the data, the testing errors decrease most of time.

The mean square errors (MSE) while using RBMs for the clustering and probability parameters of the fuzzy rules are given in Table 4.1. We see how the use of each stage clearly helps with the decreasing of the testing error.

Table 4.1: Testing results of the deep fuzzy modeling (gas furnace) ($\times 10^{-3}$)

	Training		Testing	
	No RBM	RBM	No RBM	RBM
Standard fuzzy rule	5.10	3.35	26.2	23.7
Probabilistic fuzzy rule	3.25	3.11	22.5	19.3

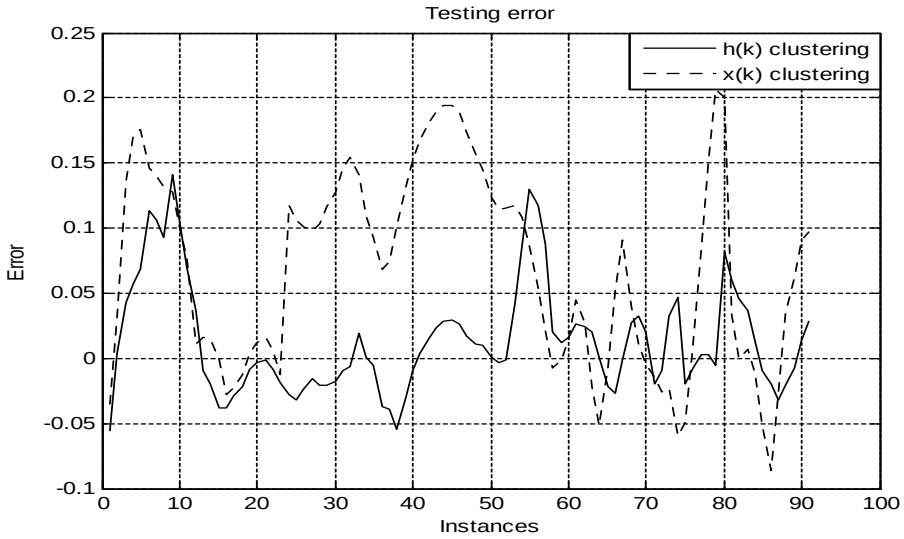

Figure 4-4: Testing errors with RBMs and without RBMs

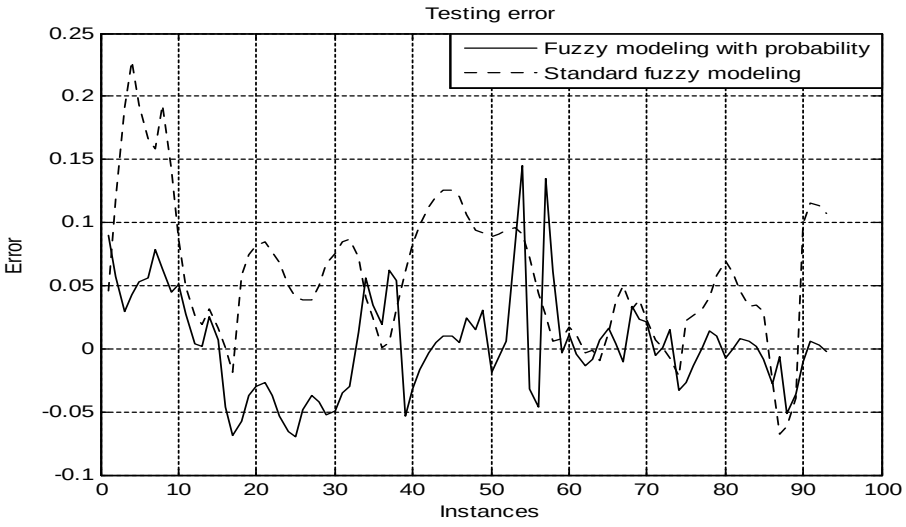

Figure 4-5: GAS testing error using probabilistic parameters

Wiener-Hammerstein benchmark fuzzy modeling

We use the same benchmark that was used in Section 3.5. Let $u(k)$ be the input and $y\,(k)$ be the output. We define the recursive input vector of the model as $\mathbf{x}(k) = [y(k-1)\cdots y(k-n_y)\ u(k)\cdots u($
So the Wiener-Hammerstein benchmark is

$$y\,(k) = f\,[y(k-1)\cdots y(k-n_y)\ u(k)\cdots u(k-n_u)] \tag{4.35}$$

Similar as the previous example, $u(k)$ and $y\,(k)$ are also normalized. The delays n_y and n_u are drawn again from a uniform interval $[1, 10]$. The fuzzy modeling process also has the following four steps:

1. Features extraction. We also train the RBM with contrastive divergence with 1-step Gibbs sampling and 10 training epochs. The learning rate is $\eta = 0.1$. Due to the quantity of data, we utilize a smaller learning rate. The number of hidden units is also chosen as $n_y + n_u + 1$.

2. Clustering. We set $\alpha = 0.95$, $\psi = 100$, $\lambda = 5$ and $C = 0.001$. α and ψ determine the probability given by the Chinese restaurant process, α is chosen close to 1 to ensure that a big number of clusters are created, ψ also increases to accomplish the same objective. The hidden feature $\mathbf{h}(k)$ is divided into 13 clusters, while the original data $\mathbf{x}(k)$ is partitioned into 11 clusters.

3. Membership functions training. The parameters W are again computed using the pseudoinverse approach.

4. Probability training. We used Matlab functions: fmincon and sqp, to compute the parameters $p_{i,j}$. P is initialized as $\mathbf{I}_K$.

Our data-driven fuzzy modeling method for the W-H data is shown in Figure 4-6. To see how the RBM helps to decrease the modeling error, Figure 4-7 shows the testing errors for $\mathbf{x}(k)$ and $\mathbf{h}(k)$ clustering. We can see that clustering directly over $\mathbf{x}(k)$ gives a good testing performance but its MSE is greater.

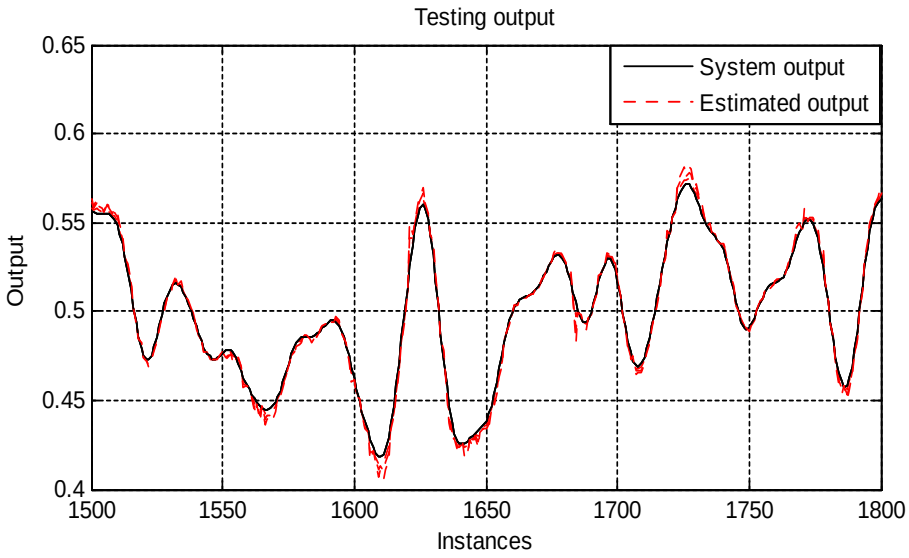

Figure 4-6: Data-driven fuzzy modeling method for the W-H data

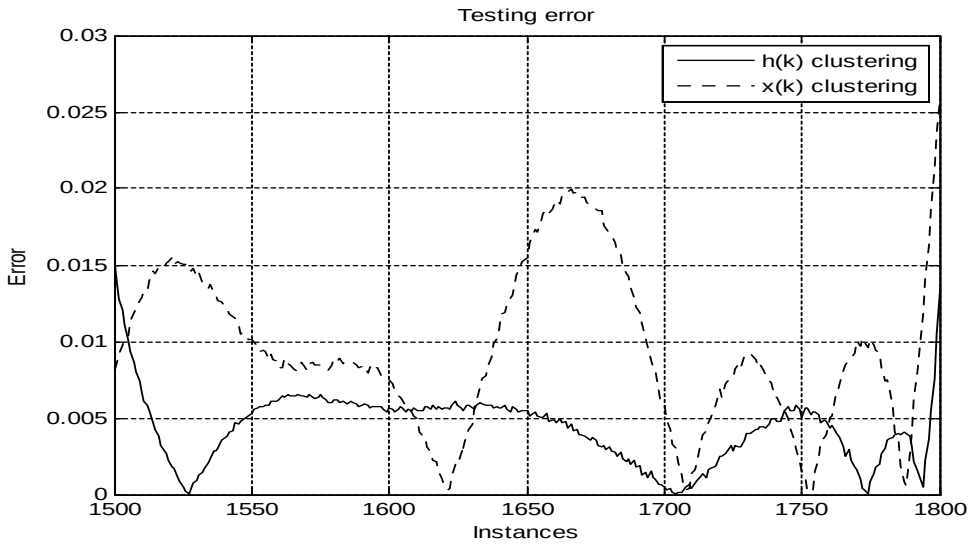

Figure 4-7: Testing errors using RBM and without RBM

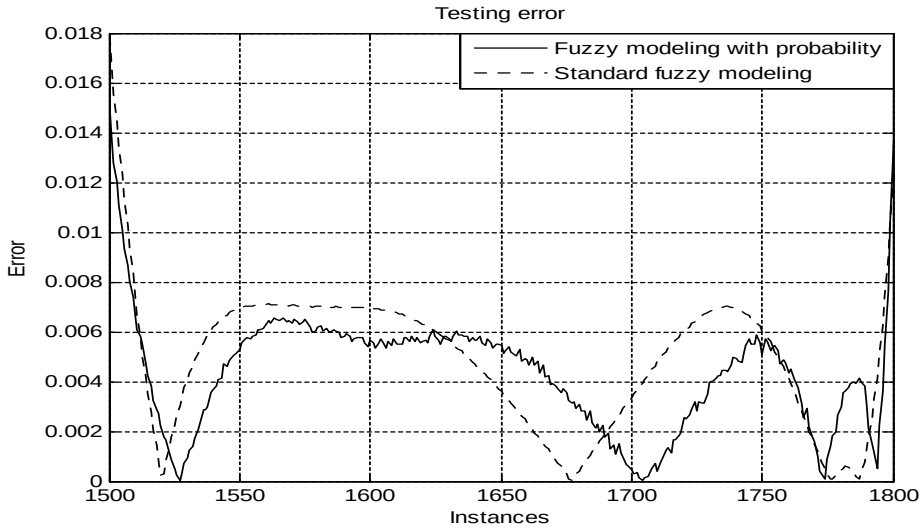

Figure 4-8: Testing errors using probabilistic parameters and standard fuzzy rules

Figure 4-8 shows the effect of the fuzzy probability parameters $p_{i,j}$. We see that as the number of clusters K increased the computational time of the model decreased, this is due to the linear programming method for calculation of P. The MSE decreases when we use probabilistic fuzzy rules.

By combining restricted Boltzmann machines and probability theory, our data-driven fuzzy modeling method has outstanding properties, see Table 4.2.

Table 4.2: Testing results of the deep fuzzy modeling (Wiener-Hammerstein benchmark) ($\times 10^{-3}$)

	Training		Testing	
	No RBM	RBM	No RBM	RBM
Standard fuzzy rule	18.9	17.7	26.4	22.8
Probabilistic fuzzy rule	16.2	14.1	23.6	19.3

We find that the modeling accuracy of the W-H benchmark does not improve so much as the gas furnace by using the probabilistic tuning. While the RBM gives better results when more data are available.

Chapter 5

Conditional continuous restricted Boltzmann machines

In this chapter, the conditional probability distributions that exist between the system input and output are modeled using an RBM based framework, we extent the RBM modeling to capabilities beyond the reach of the simple hidden-visible units relationship. We prove that DBMs have the desired universal approximation property with binary conditional distributions of the input and output data. In order to identify dynamic systems with DBMs, we modify the DBM on two ways. First, we perform an encoding procedure over the continuous input and output to get a binary representation of the input and output vectors. Then, we design a probability gradient algorithm to train the weights of the DBM in order to maximize the log-likelihood of the conditional probability between the input and output. The second method consists in modifying the variable domain such that input and output data are continuous, while the hidden variables remain binary. The probability distributions are changed by using integral evaluation in the new variable domain. The comparisons on these two methods and with the other normal nonlinear modeling methods are carried out with two benchmark problems.

For an known discrete-time nonlinear system, we use the representation described by (3.1) and (4.1) where we described the general form of the following three models introduced in

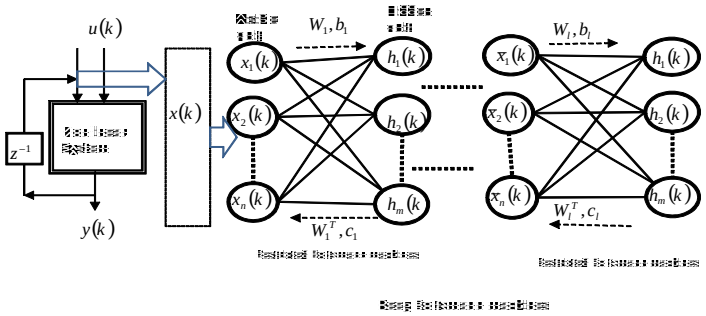

Figure 5-1: Input features extraction with deep Boltzmann machines for nonlinear system modeling

[56]

$$y(k) = \sum_{i=1}^{n_y} a_i y(k-i) + \psi\left[u(k), \cdots u(k-n_u)\right]$$

$$y(k) = \varphi\left[y(k-1), \cdots y(k-n_y)\right] + \sum_{i=1}^{n_u} b_i u(k-i)$$

$$y(k) = \varphi\left[y(k-1), \cdots y(k-n_y)\right] + \psi\left[u(k), \cdots u(k-n_u)\right]$$

where $\varphi(\cdot)$ and $\psi(\cdot)$ are unknown nonlinear functions, a_i and b_i are unknown coefficients.

At time k, there are two time series: the input series $\{u(1), \cdots, u(k)\}$, and the output series $\{y(1), \cdots, y(k)\}$. The objective of system modeling is to use these input and output series, to construct a model $\hat{\mathbf{y}}(k) = F\left[\mathbf{x}(k), k\right]$, such that $\hat{\mathbf{y}}(k) \to y(k)$, where $\hat{\mathbf{y}}(k)$ is the output of the model $F[\cdot]$.

A DBM can be regarded as a stochastic artificial neural network, which is made of several restricted Boltzmann machines (RBMs), see Fig.5-1. We will first use DBMs to extract the features from the input data $\mathbf{x}(k)$ which are combined by some output $y(k)$ and control $u(k)$ as in (4.1). The DBMs allow us to identify the nonlinear system (**??**) within the probability theory frame, such that the models are not sensitive to noise and disturbances. Then, we will model the system by using the conditional distribution $p\left[y(k) | \mathbf{x}(k)\right]$.

The modeling capability with RBMs is the basic issue for nonlinear system identification. The following theorem states the universal approximation capability of a DBM, *i.e.*, if the number of hidden units of a DBM is enough, that DBM can model any discriminative distribution $p\left[y(k) | \mathbf{x}(k)\right]$.

In order to prove the universal approximation of DBMs for dynamic system identification,

we need to encode the input variable $\mathbf{x}$. The binary assumption , $\mathbf{x} \in \{0,1\}$, makes the probability calculation of a DBM simple. Almost all classification tasks with DBMs use binary values. However, the visible units $\mathbf{x}$ in nonlinear systems do not have binary values in general. In [22], we change the DBM's structure, such that we can calculate the probability of the control $u(k)$ and the output $y(k)$. In this chapter, we use a simpler method, we use a binary encoding method to transform the continuous values into binary ones $\{0,1\}$.

We first encode $y(k)$ into a binary representation with a resolution of m bits. This means that $y(k)$ is encoded into 2^m different levels. These levels represent the range from 0 to 1 (binary 0 to 2^m-1) with a step of $1/(2^m-1)$. Similarly, the control $u(k)$ is encoded into 2^m different levels. So

$$\begin{aligned} \mathbf{x} \in \Re^n &\longrightarrow \mathbf{x} \in \{0,1\}^r \\ y \in \Re &\longrightarrow y \in \{0,1\}^m \end{aligned} \tag{5.1}$$

where $r = n \times m$. The hidden variables are assumed to be binary. With this encoding stage finished we are ready to develop the universal approximation properties of the RBMs .

5.1 Universal approximation theory for conditional RBMs

We first will show how an RBM can approximate any marginal binary distribution $p(\mathbf{x})$, we based our results in part on the work of [44]. The final goal is to demonstrate that an RBM can model any binary conditional probability distribution $p(y|\mathbf{x})$. The next lemma works as a preliminary result that will lead to our final objective.

Lemma 10 *Let $p(\mathbf{x})$ be the distribution over binary vectors $\mathbf{x}$, obtained with an RBM $R(p)$ and let $p_{(w,c)}$ be the distribution obtained when adding a hidden unit with weights w and bias c to $R(p)$. Then $\forall p(\mathbf{x}), \forall w \in \mathbb{R}^d, p(\mathbf{x}) = p_{(w,-\infty)}$*

Proof. Denoting $\widetilde{\mathrm{h}} = \begin{bmatrix} \mathrm{h} \\ h_{n+1} \end{bmatrix}$, $\widetilde{W} = \begin{bmatrix} W \\ w^T \end{bmatrix}$ and $\widetilde{C} = \begin{bmatrix} C \\ c \end{bmatrix}$ where w^T denotes the tranpose of w and introducing $z(\mathbf{x}, \mathrm{h}) = \exp\left(\mathrm{h}^T W \mathbf{x} + B^T \mathbf{x} + C^T \mathrm{h}\right)$, we can express $p(\mathbf{x}, \mathrm{h})$ and $p_{(w,c)}(\mathbf{x},\widetilde{\mathrm{h}})$ as follows:

$$
\begin{aligned}
p(\mathbf{x}, \mathrm{h}) &\propto z(\mathbf{x}, \mathrm{h}) \\
p_{(w,c)}(\mathbf{x},\widetilde{\mathrm{h}}) &\propto \exp\left(\widetilde{\mathrm{h}^T}\widetilde{W}\mathbf{x} + B^T\mathbf{x} + \widetilde{C^T}\widetilde{\mathrm{h}}\right) \\
&\propto z(\mathbf{x}, \mathrm{h}) \exp(h_{n+1} w^T \mathbf{x} + c h_{n+1})
\end{aligned} \tag{5.2}
$$

If $c = -\infty$, $p_{(w,c)}(\mathrm{v},\widetilde{\mathrm{h}}) = 0$ if $h_{n+1} = 1$. Thus, we can discard the terms where $h_{n+1} = 1$, keeping only those where $h_{n+1} = 0$. Marginalizing over the hidden units, we have:

$$
\begin{aligned}
p(\mathbf{x}) &= \frac{\sum_{\mathrm{h}} z(\mathbf{x}, \mathrm{h})}{\sum_{\mathrm{h}^{(0)}, \mathbf{x}^0} z(\mathbf{x}^0, \mathrm{h}^{(0)})} \\
p_{(w,-\infty)}(\mathbf{x}) &= \frac{\sum_{\widetilde{\mathrm{h}}} z(\mathbf{x}, \mathrm{h}) \exp(h_{n+1} w^T \mathbf{x} + c h_{n+1})}{\sum_{\widetilde{\mathrm{h}^{(0)}}, \mathbf{x}^0} z(\mathbf{x}^0, \mathrm{h}^{(0)}) \exp(h^{(0)}_{n+1} w^T \mathbf{x} + c h^{(0)}_{n+1})} \\
&= \frac{\sum_{\mathrm{h}} z(\mathbf{x}, \mathrm{h}) \exp(0)}{\sum_{\mathrm{h}^{(0)}, \mathbf{x}^0} z(\mathbf{x}^0, \mathrm{h}^{(0)}) \exp(0)} = p(\mathbf{x})
\end{aligned} \tag{5.3}
$$

Theorem 11 *If $p(\mathbf{x}) \neq p_0(\mathbf{x})$, there exists a pair (w, c) such that $KL(p_0(\mathbf{x})||p_{(w,c)}(\mathbf{x})) < KL(p_0(\mathbf{x})||p(\mathbf{x}))$*

Proof. Expanding the expression of $p_{(w,c)}(\mathbf{x})$ and regrouping the terms similar to the expression of $p(\mathbf{x})$, we get:

$$
\begin{aligned}
p_{(w,c)}(\mathbf{x}) &= \frac{\sum_{\widetilde{h}} \exp(\mathrm{h}^T W \mathbf{x} + h_{n+1} w^T \mathbf{x} + B^T \mathbf{x} + C^T \mathrm{h} + c h_{n+1})}{\sum_{\widetilde{\mathrm{h}^{(0)}}, \mathbf{x}^0} \exp(\mathrm{h}^{(0)^T} W \mathbf{x}^0 + \mathrm{h}^{(0)^T}_{n+1} w \mathbf{x}^0 + B^T \mathbf{x}^0 + C^T \mathrm{h}^{(0)} + c h^{(0)}_{n+1})} \\
&= \frac{\sum_{\mathrm{h}} z(\mathbf{x}, \mathrm{h})(1 + \exp(w^T \mathbf{x} + c))}{\sum_{\mathrm{h}^{(0)}, \mathbf{x}^0} z(\mathbf{x}^0, \mathrm{h}^{(0)})(1 + \exp(w^T \mathbf{x}^0 + c))} \\
&= \frac{(1 + \exp(w^T \mathbf{x} + c)) \sum_h z(\mathbf{x}, \mathrm{h})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} (1 + \exp(w^T \mathbf{x}^0 + c)) z(\mathbf{x}^0, \mathrm{h}^{(0)})}
\end{aligned} \tag{5.4}
$$

Therefore, we have:

$$
\begin{aligned}
KL(p_0||p_{(w,c)}) &= \sum_{\mathbf{x}} p_0(\mathbf{x}) \log p_0(\mathbf{x}) - \sum_{\mathbf{x}} p_0(\mathbf{x}) \log p_{_{w,c}}(\mathbf{x}) \qquad (5.5)\\
&= -H(p_0(\mathbf{x})) - \sum_{\mathbf{x}} p_0(\mathbf{x}) \log \left(\frac{(1+\exp(w^T\mathbf{x}+c)) \sum_h z(\mathbf{x},\mathrm{h})}{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} (1+\exp(w^T\mathbf{x}^0+c)) z(\mathbf{x}^0,\mathrm{h}^{(0)})} \right)\\
&= -H(p_0(\mathbf{x})) - \sum_{\mathbf{x}} p_0(\mathbf{x}) \log(1+\exp(w^T\mathbf{x}+c)) - \sum_{\mathbf{x}} p_0(\mathbf{x}) \log \sum_{\mathrm{h}} z(\mathbf{x},\mathrm{h})\\
&\quad + \sum_{\mathbf{x}} p_0(\mathbf{x}) \log \left(\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} (1+\exp(w^T\mathbf{x}^0+c)) z(\mathbf{x}^0,\mathrm{h}^{(0)}) \right)
\end{aligned}
$$

Assuming $w^T\mathbf{x}+c$ is a very large negative value for all $\mathbf{x}$, we can make a Taylor expansion of the first and the last term. The first term becomes

$$
\sum_{\mathbf{x}} p_0(\mathbf{x}) \log(1+\exp(w^T\mathbf{x}+c)) = \sum_{\mathbf{x}} p_0(\mathbf{x}) \exp(w^T\mathbf{x}+c) + o_{c\to-\infty}(\exp(c))
$$

and the last term becomes

$$
\begin{aligned}
&\left(\sum_{\mathbf{x}} p_0(\mathbf{x}) \right) \log \left(\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} (1+\exp(w^T\mathbf{x}^0+c)) z(\mathbf{x}^0,\mathrm{h}^{(0)}) \right)\\
&= \log \left(\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)}) \right) + \log \left(1 + \frac{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} \exp(w^T\mathbf{x}^0+c) z(\mathbf{x}^0,\mathrm{h}^{(0)})}{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)})} \right)\\
&= \log \left(\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)}) \right) + \frac{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} \exp(w^T\mathbf{x}^0+c) z(\mathbf{x}^0,\mathrm{h}^{(0)})}{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)})} + o_{c\to-\infty}(\exp(c))
\end{aligned}
$$

But

$$
\begin{aligned}
\frac{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} \exp(w^T\mathbf{x}^0+c) z(\mathbf{x}^0,\mathrm{h}^{(0)})}{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)})} &= \sum_{\mathbf{x}} \exp(w^T\mathbf{x}+c) \frac{\sum_{\mathrm{h}^{(0)}} z(\mathbf{x},\mathrm{h}^{(0)})}{\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)})}\\
&= \sum_{\mathbf{x}} \exp(w^T\mathbf{x}+c) p_{(w,-\infty)}(\mathbf{x})\\
&= \sum_{\mathbf{x}} \exp(w^T\mathbf{x}+c) p(\mathbf{x})
\end{aligned}
$$

Putting all terms back together, we have

$$\begin{aligned} KL(p_0(\mathbf{x})||p_{(w,c)}(\mathbf{x})) &= -H(p_0) - \sum_{\mathbf{x}} p_0(\mathbf{x}) \exp(w^T\mathbf{x}+c) + \sum_{\mathbf{x}} \exp(w^T\mathbf{x}+c) + o_{c\to-\infty}(\exp(c)) \\ &\quad - \sum_{\mathbf{x}} p_0(\mathbf{x}) \log\left(\sum_{\mathrm{h}} z(\mathbf{x},\mathrm{h})\right) + \log\left(\sum_{\mathbf{x}^0,\mathrm{h}^{(0)}} z(\mathbf{x}^0,\mathrm{h}^{(0)})\right) \\ &= KL(p_0(\mathbf{x})||p(\mathbf{x})) + \sum_{\mathbf{x}} \exp(w^T\mathbf{x}+c)(p(\mathbf{x})-p_0(\mathbf{x})) + o_{c\to-\infty}(\exp(c)) \end{aligned} \tag{5.6}$$

Finally, we have

$$KL(p_0||p_{w,c}) - KL(p_0||p) = \exp(c)\sum_{\mathbf{x}} \exp(w^T\mathbf{x})(p(\mathbf{x})-p_0(\mathbf{x})) + o_{c\to-\infty}(\exp(c)) \tag{5.7}$$

The question now becomes: can we find a w such that $\sum_{\mathbf{x}} \exp(w^T\mathbf{x})(p(\mathbf{x})-p_0(\mathbf{x}))$ is negative?

As $p_0(\mathbf{x}) \neq p(\mathbf{x})$, there is a $\widehat{\mathbf{x}}$ such that $p(\widehat{\mathbf{x}}) < p_0(\widehat{\mathbf{x}})$. Then there exists a positive scalar a such that $\widehat{w} = a\left(\widehat{\mathbf{x}} - \frac{1}{2}e\right)$(with $e = [1...1]^T$) yields $\sum_{\mathbf{x}} \exp(\widehat{w}^T\mathbf{x})(p(\mathbf{x})-p_0(\mathbf{x})) < 0$. Indeed, for $\mathbf{x} \neq \widehat{\mathbf{x}}$, we have

$$\begin{aligned} \frac{\exp(\widehat{w}^T\mathbf{x})}{\exp(\widehat{w}^T\widehat{\mathbf{x}})} &= \exp(\widehat{w}^T(\mathbf{x}-\widehat{\mathbf{x}})) \\ &= \exp(a\left(\widehat{\mathbf{x}} - \frac{1}{2}e\right)^T(\mathbf{x}-\widehat{\mathbf{x}})) \\ &= \exp\left(a\sum_i \left(\widehat{\mathbf{x}}_i - \frac{1}{2}\right)(\mathbf{x}_i - \widehat{\mathbf{x}}_i)\right) \end{aligned} \tag{5.8}$$

For i such that $\mathbf{x}_i - \widehat{\mathbf{x}}_i > 0$, we have $\mathbf{x}_i = 1$ and $\widehat{\mathbf{x}}_i = 0$. Thus, $\widehat{\mathbf{x}}_i - \frac{1}{2} = -\frac{1}{2}$ and the term inside the exponential is negative (since a is positive). For i such that $\mathbf{x}_i - \widehat{\mathbf{x}}_i < 0$, we have $\mathbf{x}_i = 0$ and $\widehat{\mathbf{x}}_i = 1$. Thus, $\widehat{\mathbf{x}}_i - \frac{1}{2} = \frac{1}{2}$ and the term inside the exponential is also negative. Furthermore, the terms come close to 0 as a goes to infinity. Since the sum can be decomposed as

$$\sum_{\mathbf{x}} \exp(\widehat{w}^T\mathbf{x})(p(\mathbf{x}) - p_0(\mathbf{x})) = \exp(\widehat{w}^T\widehat{\mathbf{x}}) \left(\sum_{\mathbf{x}} \frac{\exp(\widehat{w}^T\mathbf{x})}{\exp(\widehat{w}^T\widehat{\mathbf{x}})}(p(\mathbf{x}) - p_0(\mathbf{x})) \right) \tag{5.9}$$

$$= \exp(\widehat{w}^T\widehat{\mathbf{x}}) \left(p(\widehat{\mathbf{x}}) - p_0(\widehat{\mathbf{x}}) + \sum_{\mathbf{x}\neq\widehat{\mathbf{x}}} \frac{\exp(\widehat{\omega}^T\mathbf{x})}{\exp(\widehat{\omega}^T\widehat{\mathbf{x}})}(p(\mathbf{x}) - p_0(\mathbf{x})) \right)$$

we have

$$\sum_{\mathbf{x}} \exp(\widehat{w}^T\mathbf{x})(p(\mathbf{x}) - p_0(\mathbf{x}))\tilde{}_{a\to+\infty} \exp(\widehat{w}^T\widehat{\mathbf{x}})(p(\widehat{\mathbf{x}}) - p_0(\widehat{\mathbf{x}})) > 0. \tag{5.10}$$

Therefore, there is a value $\widehat{a}$ such that, if $a > \widehat{a}$, $\sum_{\mathbf{x}} \exp(w^T\mathbf{x})(p(\mathbf{x}) - p_0(\mathbf{x})) > 0$. This concludes the proof.

Using the above, we obtain the main theorem:

Theorem 12 *Let p_0 be an arbitrary distribution over $\{0,1\}^n$ and let R_p be an RBM with marginal distribution $p(\mathbf{x})$ over the visible units such that $KL(p_0||p) > 0$. Then there exists an RBM $R_{p_{(w,c)}}$ composed of R_p and an additional hidden unit with parameters (w, c) whose marginal distribution $p_{(w,c)}$ over the visible units achieves $KL(p_0(\mathbf{x})||p_{(w,c)}(\mathbf{x})) < KL(p_0(\mathbf{x})||p(\mathbf{x}))$.*

Proof. Choose a $\widehat{\mathbf{x}}$ such that $p(\widehat{\mathbf{x}}) < p_0(\widehat{\mathbf{x}})$. Pick $\widehat{w} = a\left(\widehat{\mathbf{x}} - \frac{1}{2}e\right)$ with a large enough and c such that $w^T\mathbf{x} + c$ is a large enough negative value for all $\mathbf{x}$. Then adding a hidden unit parameters ($\widehat{w}$,c) gives the desired result.

The next theorem is for the limit case when the number of hidden units is very large, so that we can represent any discrete distribution exactly.

Theorem 13 *Any distribution over $\{0,1\}^n$ can be approximated arbitrarily well (in the sense of the KL divergence) with an RBM with $k+1$ hidden units where k is the number of input vectors whose probability is not 0.*

Proof. In the previous proof, we had

$$p_{(w,c)}(\mathbf{x}) = \frac{(1 + \exp(w^T\mathbf{x} + c)) \sum_h z(\mathbf{x}, \mathrm{h})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} (1 + \exp(w^T\mathbf{x}^0 + c)) z(\mathbf{x}^0, \mathrm{h}^{(0)})} \tag{5.11}$$

Let $\widetilde{\mathbf{x}}$ be an arbitrary $\mathbf{x}$ and $\widehat{w}$ defined in the same way as before, i.e. $\widehat{w} = a\left(\widetilde{\mathbf{x}} - \frac{1}{2}\right)$. Now define $\widehat{c} = -\widehat{w}^T\widetilde{\mathbf{x}} + \lambda$ with $\lambda \in \mathbb{R}$. We have:

$$\begin{aligned} \lim_{a\to\infty} 1 + \exp(\widehat{w}^T\mathbf{x} + \widehat{c}) &= 1 \qquad \text{for } \mathbf{x} \neq \widetilde{\mathbf{x}} \\ 1 + \exp(\widehat{w}^T\mathbf{x} + \widehat{c}) &= 1 + \exp(\lambda) \end{aligned} \tag{5.12}$$

Thus, we can see that, for $\mathbf{x} \neq \widetilde{\mathbf{x}}$:

$$\begin{aligned} \lim_{a\to\infty} p_{(\widehat{w},\widehat{c})}(\mathrm{v}) &= \frac{\sum_h z(\mathbf{x}, \mathrm{h})}{\sum_{\mathbf{x}^0 \neq \widetilde{\mathbf{x}}, \mathrm{h}^{(0)}} z(\mathbf{x}^0, \mathrm{h}^{(0)}) + \sum_{\mathrm{h}^{(0)}} (1 + \exp(\widehat{w}^T\widetilde{\mathbf{x}} + \widehat{c})) z(\widetilde{\mathbf{x}}, \mathrm{h}^{(0)})} \\ &= \frac{\sum_h z(\mathrm{v}, \mathrm{h})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} z(\mathbf{x}^0, \mathrm{h}^{(0)}) + \sum_{\mathrm{h}^{(0)}} \exp(\lambda) z(\widetilde{\mathbf{x}}, \mathrm{h}^{(0)})} \\ &= \frac{\sum_h z(\mathbf{x}, \mathrm{h})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} z(\mathbf{x}^0, \mathrm{h}^{(0)})} \frac{1}{1 + \exp(\lambda) \frac{\sum_{\mathrm{h}^{(0)}} z(\widetilde{\mathbf{x}}, \mathrm{h}^{(0)})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} z(\mathbf{x}^0, \mathrm{h}^{(0)})}} \end{aligned} \tag{5.13}$$

Remembering $p(\mathbf{x}) = \frac{\sum_h z(\mathbf{x}, \mathrm{h})}{\sum_{\mathbf{x}^0, \mathrm{h}^{(0)}} z(\mathbf{x}^0, \mathrm{h}^{(0)})}$, we have for $\mathbf{x} \neq \widetilde{\mathbf{x}}$

$$\lim_{a\to\infty} p_{(\widehat{w},\widehat{c})}(\mathbf{x}) = \frac{p(\mathbf{x})}{1 + \exp(\lambda) p(\widetilde{\mathbf{x}})} \tag{5.14}$$

Similarly, we can see that

$$\lim_{a\to\infty} p_{(\widehat{w},\widehat{c})}(\widetilde{\mathbf{x}}) = \frac{\left[1 + \exp(\lambda)\right] p(\widetilde{\mathbf{x}})}{1 + \exp(\lambda) p(\widetilde{\mathbf{x}})} \tag{5.15}$$

Depending on the value of λ, one can see that adding a hidden unit allows one to increase the probability of an arbitrary $\widetilde{\mathbf{x}}$ and to uniformly decrease the probability of every other $\mathbf{x}$ by a multiplicative factor. However, one can also see that, if $p(\widetilde{\mathbf{x}}) = 0$, then $p_{(\widehat{w},\widehat{c})}(\widetilde{\mathbf{x}}) = 0$ for all λ.

We can therefore build the desired RBM as follows. Let us index the $\mathbf{x}$'s over the integers from 1 to 2^n and sort them such that

$$p^0(\mathbf{x}_{k+1}) = ... = p^0(\mathbf{x}_{2^n}) = 0 < p^0(\mathbf{x}_1) \leq p^0(\mathbf{x}_2) \leq ... \leq p^0(\mathbf{x}_k) \tag{5.16}$$

Let us denote p^i the distribution of an RBM with i hidden units. We start with and RBM whose weights and biases are all equal to 0. The marginal distribution over the visible units induced by that RBM is the uniform distirbution. Thus,

$$p^0(\mathbf{x}_1) = ... = p^0(\mathbf{x}_{2^n}) = 2^{-n} \tag{5.17}$$

We define $w_1 = a_1(\mathbf{x}_1 - \frac{1}{2})$ and $c_1 = -w_1^T\mathbf{x}_1 + \lambda_1$.

As shown before, we now have:

$$\begin{aligned} \lim_{a_1 \to +\infty} p^1(\mathbf{x}_1) &= \frac{[1+\exp(\lambda_1)]\,2^{-n}}{1+\exp(\lambda_1)2^{-n}} \\ \lim_{a_1 \to +\infty} p^1(\mathbf{x}_i) &= \frac{2^{-n}}{1+\exp(\lambda_1)2^{-n}} \quad \forall i \geq 2 \end{aligned} \tag{5.18}$$

As we can see, we can set $p^1(\mathbf{x}_1)$ to a value arbitrarily close to 1, with a uniform distribution over $\mathbf{x}_1, ..., \mathbf{x}_{2^n}$. Then, we can choose λ_2 such that $\frac{p^2(\mathbf{x}_2)}{p^2(\mathbf{x}_1)} = \frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)}$. This is possible since we can arbitrarily increase $p^2(\mathbf{x}_2)$ while multiplying the other probabilities by a constant factor and since $\frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)} \geq \frac{p^1(\mathbf{x}_2)}{p^1(\mathbf{x}_1)}$. We can continue the procedure until $p^k(\mathbf{x}_k)$. The ratio $\frac{p^i(\mathbf{x}_j)}{p^i\mathbf{x}_{(j-1)}}$ does not depend on the value of i as long $i > j$. (because at each such step i, the two probabilities are multiplied by the same factor). We will then have

$$\begin{aligned} \frac{p^k(\mathbf{x}_k)}{p^k(\mathbf{x}_{k-1})} &= \frac{p(\mathbf{x}_k)}{p(\mathbf{x}_{k-1})}, ..., \frac{p^k(\mathbf{x}_2)}{p^k(\mathbf{x}_1)} = \frac{p(\mathbf{x}_2)}{p(\mathbf{x}_1)} \\ p^k(\mathbf{x}_{k+1}) &= ... = p^k(\mathbf{x}_{2^n}) \end{aligned} \tag{5.19}$$

From that, we can deduce $p^k(\mathbf{x}_1) = \nu_k p(\mathbf{x}_1), ..., p^k(\mathbf{x}_k) = \nu_k p(\mathbf{x}_k)$ with $\nu_k = 1 - (2^n - k)p^k(\mathbf{x}_{2^n})$

We also have $\frac{p^k(\mathbf{x}_1)}{p^k(\mathbf{x}_{2^n})} = \frac{p^1(\mathbf{x}_1)}{p^1(\mathbf{x}_{2^n})} = 1 + \exp(\lambda_1)$.

Thus, $p^k(\mathbf{x}_1) = p(\mathbf{x}_1)[1 - (2^n - k)p^k(\mathbf{x}_{2^n})] = (1 + \exp(\lambda_1))p^k(\mathbf{x}_{2^n})$.

Resolving the above equations, we have

$$
\begin{aligned}
p^k(\mathbf{x}_i) &= \frac{p(\mathbf{x}_1)}{1+\exp(\lambda_1)+p(\mathbf{x}_1)(2^n-k)} \quad \text{for } i>k \\
p^k(\mathbf{x}_i) &= p(\mathbf{x}_i)\frac{1+\exp(\lambda_1)}{1+\exp(\lambda_1)+p(\mathbf{x}_1)(2^n-k)} \quad \text{for } i>k
\end{aligned} \tag{5.20}
$$

Making a Taylor expansion to the first order of $KL(p||p^k)$ when λ_1 goes to infinity, we have

$$
KL(p(\mathbf{x})||p^k(\mathbf{x})) = \sum_i p(\mathbf{x}_i)\frac{(2^n-k)p(\mathbf{x}_i)}{1+\exp(\lambda_1)} + o(\exp(-\lambda_1)) \rightarrow_{\lambda_1\rightarrow\infty} 0 \tag{5.21}
$$

This concludes the proof. This set of lemmas and theorems have concluded that an RBM can effectively represent any kind of marginal probability distribution $p(\mathbf{x})$. However, the marginal distributions are not suitable for system modeling, they are good from the perspective of generative models but when a discriminative task has to be carried out, one may want to make use of the input $\mathbf{x}$ in order to estimate the output y. Using the previous results we can finally state the main Theorem of this section.

Theorem 14 *Any conditional distribution $p\left[y\left(k\right)|\mathbf{x}\left(k\right)\right]$ over $\{0,1\}^m \times \{0,1\}^r$ can be approximated arbitrarily well in the sense of the Kullback-Leibler divergence by an DBM with $r_0 \times m_0 + 1$ hidden units, where r_0 and m_0 are the number of different inputs and outputs respectively, such that the pair $(\mathbf{x}, y)$ has a probability $p(\mathbf{x}, y)$ greater than 0.*

Proof. For each RBM, the training object is

$$
\min KL\left(p||q\right) \rightarrow \max \sum \log p\left(\mathbf{x}\right) \tag{5.22}
$$

where $KL\left(p||q\right)$ is the the Kullback-Liebler divergence, which is the distance from the RBM probability distribution $p(\mathbf{x})$ to the probability distribution $q\left(\mathbf{x}\right)$. It is

$$
KL\left(p||q\right) = \sum_x q\left(\mathbf{x}\right)\log\frac{q\left(\mathbf{x}\right)}{p\left(\mathbf{x}\right)} \tag{5.23}
$$

Any marginal probability distribution $p(\mathbf{x})$ over $\{0,1\}^r$ can be approximated arbitrarily well in the sense of the Kullback-Leibler divergence (5.23) by an RBM with $k+1$ hidden units,

where k is the number of input vectors whose probability is not 0, the proof can be found [44]. So an RBM with enough hidden units can model any given marginal probability distribution $p(\mathbf{x})$. Consider a well known nonlinear system which can be represented by a binary conditional distribution $p(y|\mathbf{x})$, the vectors $\mathbf{x}$ and y take values from the finite sets $\{\mathbf{x}_1, ..., \mathbf{x}_k, ..., \mathbf{x}_{r_0}\}$ and $\{y_1, ..., y_l, ..., y_{m_0}\}$ respectively. The distribution has probability measures $p(y_l|\mathbf{x}_k)$, for each pair $(\mathbf{x}_k,\ y_l)$, with $k = 1, ..., r_0$ and $l = 1, ...m_0$. It is also known that the conditional probability of each pair is

$$p(y_l|\mathbf{x}_k) = \frac{p(\mathbf{x}_k, y_l)}{p(\mathbf{x}_k)} = \frac{p(\mathbf{x}_k, y_l)}{\sum_j p(\mathbf{x}_k, y_j)} \tag{5.24}$$

From (5.24) we have

$$p(y_l|\mathbf{x}_k) \sum_{j \neq l} p(\mathbf{x}_k, y_j) + [p(y_l|\mathbf{x}_k) - 1]\, p(\mathbf{x}_k, y_l) = 0 \tag{5.25}$$

Since we know the value of every term $p(y_l|\mathbf{x}_k)$, we can form $r_0 \times m_0$ equations with the same form of (5.25) with different indexes k and l. These linear equations can be solved, and the solutions are $p(\mathbf{x}_k, y_l)$. That leads to the desired conditional distribution $p(y|\mathbf{x})$. With each $p(\mathbf{x}_k, y_l)$, we have created a distribution that contains the original conditional distribution $p(y|\mathbf{x})$. Finally we consider the pair $(\mathbf{x}_k, y_l)$ as a single random variable z_{kl}, such that $p(z_{kl}) = p(\mathbf{x}_k, y_l)$. As stated in [44], we can construct an DBM with $r_0 \times m_0 + 1$ hidden units, which models the distribution given by $p(z_{kl})$. ■

Thus, a DBM can model any marginal distribution $p(\mathbf{x})$ with $\mathbf{x} \in \{0, 1\}^r$ being a binary vector. It is similar to the universal approximation theory of neural networks [20][32].

Input features extraction

The training process of the DBMs is as follows: 1) The training data and the hidden representation of the first RBM are $x(k) \in \Re^n$ and $h_1(k) \in \Re^{l_1}$. We use q data to train the weights of the first model $W_1 \in \Re^{l_1 \times n}$, $b_1 \in \Re^{l_1}$, and $c_1 \in \Re^n$. 2) After the first model is trained, their weights are fixed. The code or hidden representation of the first model is computed with fixed weights to generate q examples, which are the input to the second model. 3) The second model is trained using as input $h_1(k) \in \Re^{l_1}$ and it generates the hidden representation $h_2(k) \in \Re^{l_2}$, which is the input of the third model. 4) Then we train the third model, until

all l models are trained. This training process is shown in Fig.3-5.

For each RBM, the training object is (5.22). The Kullback-Liebler divergence (5.23) is [1]

$$KL(p,q) = \sum_x q(\mathbf{x}) \log q(\mathbf{x}) - \sum_x q(\mathbf{x}) \log p(\mathbf{x})$$

The training process for each RBM is carried out utilizing the procedure shown in section 3.3.

5.2 DBM training with binary representation of input and output

The DBM training method discussed in the previous section can generate the probability distribution and extract the features of $\mathbf{x}(k)$ for system identification. This method is widely applied in classification and regression tasks [27][28]. Although $\mathbf{x}(k)$ includes the input $u(k)$ and the output $y(k)$, the probability distribution of $\mathbf{x}(k)$ does not lead directly to the conditional probability distribution $p[y(k)|\mathbf{x}(k)])$ which is the system modeling goal.

In this session we use the inherent conditional distribution $p[y(k)|\mathbf{x}(k)]$ of the data to train the DBMs for nonlinear system modeling. The conditional distribution will be calculated from the joint distribution of the inputs $\mathbf{x}(k)$ and associated output $y(k)$. This idea has been applied in classification task in [40].

Joint distribution for DBM training

Consider a training set denoted as $D = \{\mathbf{x}(k), y(k)\}$, here $\mathbf{x}(k)$ and $y(k)$ are the k-th training input vector and output respectively. In this session, we encode $\mathbf{x}(k)$ and $y(k)$ into binary representations with resolutions of r bits and m bits respectively, see (5.1). After the encoding we have $\mathbf{x}(k) \in \Re^r$, $y(k) \in \Re^m$. The loss function of the DBM training is defined as

$$J_c(D) = -\sum_k^D \log p[\mathbf{x}(k), y(k)] \tag{5.26}$$

The training object is as (5.22): $\min_D J_c(D)$.

The DBM model gives the joint probability distribution between the observed variables $\{\mathbf{x},y\}$ and the hidden features $h \in \Re^s$, see Fig.5-1. The joint probability is also an energy function

$$\begin{aligned} p(\mathbf{x},y,h) &\propto e^{-E(\mathbf{x},y,h)} \\ E(\mathbf{x},y,h) &= -h^T W\mathbf{x} - b^T\mathbf{x} - c^T h - D^T y - h^T V y \end{aligned} \tag{5.27}$$

We define the model parameters as $\Lambda = \{W,b,c,D,V\}$. By the binary encoding (5.1), the conditional distributions of $\mathbf{x}$ are

$$\begin{aligned} p(\mathbf{x}|h) &= \prod_i p(x_i|h) \\ p(x_i=1|h) &= \text{sign}\left(b_i + \textstyle\sum_j W_{ji}h_j\right) \end{aligned} \tag{5.28}$$

where sign$(\cdot)$ is the sign function. The conditional distributions of y are

$$\begin{aligned} p(y|h) &= \prod_\kappa p(y_\kappa|h) \\ p(y_\kappa=1|h) &= \text{sign}\left(D_\kappa + \textstyle\sum_j V_{j\kappa}h_j\right) \end{aligned} \tag{5.29}$$

Clearly, h is the key variable that captures the relationship between $\mathbf{x}$ and y. The inverse relationship is

$$\begin{aligned} p(h|\mathbf{x},y) &= \prod_j p(h_j|\mathbf{x},y) \\ p(h_j=1|\mathbf{x},y) &= \text{sign}\left(c_j + \textstyle\sum_\kappa V_{j\kappa}y_\kappa + \sum_i W_{ji}\bar{x}_i\right) \end{aligned} \tag{5.30}$$

In order to minimize the loss function (5.26), the gradient of $J_c(D)$ with respect to the parameters Λ is

$$\begin{aligned} \frac{\partial \log p[\mathbf{x}(k),y(k)]}{\partial\lambda} &= -E_{h|\mathbf{x}(k),y(k)}\left[\frac{\partial E(\mathbf{x}(k),y(k),h)}{\partial\lambda}\right] \\ &\quad + E_{\mathbf{x},y,h}\left[\frac{\partial E(\mathbf{x},y,h)}{\partial\lambda}\right] \end{aligned} \tag{5.31}$$

where $\lambda \in \Lambda = \{W,b,c,D,V\}$. Each parameter, W,b,c,D, and V, should be applied to (5.31).

To calculate the gradient (5.31), the standard stochastic gradient descent approach can be implemented [4]. The computation of the exact value of (5.31) is not tractable and we use the contrastive divergence (CD) method in order to infer it. This estimation replaces the expectation with a sample from a k_G- steps Gibbs sampling process. This process is initiated

by considering the training examples $\{\mathbf{x}(k), y(k)\}$ as the initial state of the visible variables. Then, we can choose $k_G = 1$ to improve the training speed with a small bias during the whole learning process.

Conditional distribution for DBM training

The joint distribution $p(\mathbf{x}, y)$ of the DBM can be used to predict the system output by giving specified input data, such as time series regression. However, for dynamic system identification, the conditional distribution $p(y|\mathbf{x})$ is needed. As shown in Theorem 1, there always exists a DBM which can represent the conditional distribution over a given training set $\{\mathbf{x}(k),y(k)\}$ in binary units.

The loss function of the conditional distribution $p(y|\mathbf{x})$ is defined as

$$J_o(D) = -\sum_{k}^{D} \log p\left(y(k)|\mathbf{x}(k)\right) \tag{5.32}$$

The object of the dynamic system identification with DBMs is

$$\arg\max_{\Lambda} \left[\sum_{k}^{D} \log p\left(y(k)|\mathbf{x}(k)\right)\right] \quad \text{or} \quad \arg\min_{\Lambda} J_o(D) \tag{5.33}$$

Since J_o uses the conditional distribution, the training algorithm is different than J_c in (5.26) that uses the joint distribution.

The conditional probability $p(y|\mathbf{x})$ is also an energy function. From (5.27), it is calculated by

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x},y)}{p(\mathbf{x})} = \frac{\sum_{h} e^{-E(\mathbf{x},y,h)}}{\sum_{y,h} e^{-E(\mathbf{x},y,h)}} \tag{5.34}$$

So we can directly increase the conditional probability along the data distribution by increasing the value of $p(y|\mathbf{x})$ for each instance. To accomplish this, the negative log-likelihood should be minimized by a stochastic gradient descent variant. Because

$$-\log p(y|\mathbf{x}) = \log p(\mathbf{x}) - \log p(\mathbf{x},y) \tag{5.35}$$

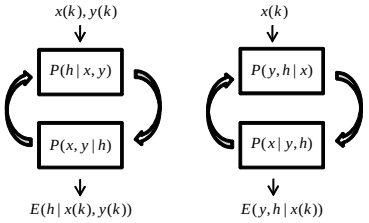

Figure 5-2: Gibbs sampling for $p(y|\mathbf{x})$ calculation

Considering the instance k, for $[\mathbf{x}(k), y(k)]$ we have

$$\begin{array}{l} -\log p\left[\mathbf{x}(k), y(k)\right] = \log \sum_{y,h} e^{-E[\mathbf{x}(k),y(k),h]} \\ -\log \sum_{h} e^{-E[\mathbf{x}(k),y(k),h]} \end{array} \tag{5.36}$$

Then, the gradient of the negative log-likelihood with respect to the parameter λ is

$$\begin{array}{l} -\frac{\partial \log p[y(k)|\mathbf{x}(k)]}{\partial \lambda} = \frac{\sum_h e^{-E[\mathbf{x}(k),y(k),h]} \frac{\partial E[\mathbf{x}(k),y(k),h]}{\partial \lambda}}{\sum_h e^{-E(\mathbf{x}(k),y(k),h)}} \\ -\frac{\sum_{y,h} e^{-E[\mathbf{x}(k),y(k),h]} \frac{\partial E[\mathbf{x}(k),y(k),h]}{\partial \lambda}}{\sum_{y,h} e^{-E[\mathbf{x}(k),y(k),h]}} \end{array} \tag{5.37}$$

In the form of mathematical expectation,

$$\begin{array}{l} -\frac{\partial \log p[y(k)|\mathbf{x}(k)]}{\partial \lambda} = E_{h|(\mathbf{x}(k),y(k))}\left[\frac{\partial E(\mathbf{x}(k),y(k),h)}{\partial \lambda}\right] \\ -E_{(y,h)|\mathbf{x}(k))}\left[\frac{\partial E(\mathbf{x}(k),y,h)}{\partial \lambda}\right] \end{array} \tag{5.38}$$

Both probability expectations of (5.38) can be computed using Gibbs sampling and the contrastive divergence (CD). The CD algorithm can be done for the first term using alternation sampling processes over the distributions (5.30), (5.28) and (5.29) respectively.

However, there are not compact expressions for $p\left[h|(\mathbf{x}(k), y(k)\right]$ and $p\left[(y,h)|\mathbf{x}(k)\right]$. Both of them are needed for the second term of (5.38). The big number of output elements makes the gradient computation intractable. We use CD algorithm to approximate the gradient with only one iteration ($k_G = 1$).The sampling process is explained in Fig. 5-2. The estimation results are satisfied by the conditional distribution calculations.

In order to implement the CD algorithm for $E_{(y,h)|\mathbf{x}(k))}\left[\frac{\partial E(\mathbf{x}(k),y,h)}{\partial \lambda}\right]$, we need to calculate

$p\left[(y,h)|\mathbf{x}(k)\right]$ as follows:

$$\begin{aligned} p\left[(y,h)|\mathbf{x}(k)\right] &= \frac{e^{-E(\mathbf{x}(k),y,h)}}{\sum_{y,h} e^{-E(\mathbf{x}(k),y,h)}} \\ &= \frac{e^{h^T W \mathbf{x}(k) + b^T \mathbf{x}(k) + c^T h + dy + h^T V y}}{\sum_{y,h} e^{h^T W \mathbf{x}(k) + b^T \mathbf{x}(k) + c^T h + dy + h^T V y} dy} \end{aligned} \tag{5.39}$$

The calculation of (5.39) is expensive, because it requires to calculate more than 2^{m+l} possible values. However, it is tractable for system identification. After this distribution is obtained, we just should sample all possible values for y and h.

Once $p\left[(y,h)|\mathbf{x}(k)\right]$ is calculated, we need to find an expression for $p\left[\mathbf{x}|(y,h)\right]$ to complete the Gibbs sampling,

$$p\left[\mathbf{x}|(y,h)\right] = \frac{p(\mathbf{x},y,h)}{p(y,h)} = \prod_i p(\bar{x}_i|h) \tag{5.40}$$

(5.40) is calculated with (5.28).

The training algorithm for the conditional distribution is

$$\lambda\left(k+1\right) = \lambda\left(k\right) - \eta \frac{\partial \log p\left(y|\mathbf{x}\right)}{\partial \lambda} \tag{5.41}$$

where $\eta > 0$ is the training factor, $\lambda \in \Lambda = \{W, b, c, d, V\}$, $\frac{\partial \log p(y|\mathbf{x})}{\partial \lambda}$ is calculated by (5.38).

5.3 DBM training with continuous values of input and output

In order to model nonlinear systems with continuous values, we use the binary encoding method to calculate the conditional distribution $p\left(y|\mathbf{x}\right)$ for the DBM model. However, the training data are enlarged dramatically, for example the dimension of $\mathbf{x}(k)$ increases from n to $2^{n\times r}$. In this session, we modify the learning algorithm of RBMs, such that nonlinear system can be modeled by DBM with continuous values.

In order to train the parameters in (5.41), we need to calculate the following six conditional probabilities

Probability of x given h

The conditional probability of $\mathbf{x}(k) \in \Re^n$ given $h(k) \in \Re^s$ does not have explicit expression from the input and output domains,

$$\begin{aligned} p(\mathbf{x}|h) &= \frac{p(\mathbf{x},h)}{p(h)} = \frac{\int_{\bar{y}} p(\mathbf{x},h,\bar{y})d\bar{y}}{\int_{\bar{y}}\int_{\bar{\mathbf{x}}} p(\bar{\mathbf{x}},h,\bar{y})d\bar{\mathbf{x}}d\bar{y}} \\ &= \frac{\int_{\bar{y}} e^{h^T W\mathbf{x}+b^T\mathbf{x}+c^T h+D^T\bar{y}+h^T V\bar{y}}d\bar{y}}{\int_{\bar{y}}\int_{\bar{\mathbf{x}}} e^{h^T W\bar{\mathbf{x}}+b^T\bar{\mathbf{x}}+c^T h+D^T\bar{y}+h^T V\bar{y}}d\bar{\mathbf{x}}d\bar{y}} \\ &= \frac{e^{h^T W\mathbf{x}+b^T\mathbf{x}}}{\int_{\bar{\mathbf{x}}} e^{h^T W\bar{\mathbf{x}}+b^T\bar{\mathbf{x}}}d\bar{\mathbf{x}}} = \prod_i p(\bar{x}_i|h) \end{aligned}$$

where $\bar{h}$, $\bar{y}$ and $\bar{\mathbf{x}}$ denote the silent variables h, y and $\mathbf{x}$, which will be used for integral evaluations along the domain of the hidden, output and input vectors respectively.

$$p(x_i|h) = \frac{e^{x_i\left(b_i+\sum_j w_{ji}h_j\right)}}{\int_{\bar{x}_i} e^{\bar{x}_i\left(b_i+\sum_j w_{ji}h_j\right)}d\bar{x}_i} \tag{5.42}$$

Now we explore different cases for the domain of x_i, we study three intervals: $[0,\infty)$, $[0,1]$ and $[-\delta,\delta]$ where $\delta \in \Re^+$.

For the case of $x_i \in [0,\infty)$, if we define $\alpha_i(h) = b_i + \sum_j w_{ji}h_j$, then we can directly evaluate the integral taking into account that $\alpha_i(h) < 0$, $\forall h$.

In order to ensure that the integral converges, the evaluations of the three cases are presented in Table 5.1.

Table 5.1: Probability expressions for $p(\mathbf{x}|\mathbf{h})$

Indicator/Interval	$[0,\infty)$	$[0,1]$	$[-\delta,\delta]$
$p(x_i\|\mathbf{h})$	$-\alpha_i e^{\alpha_i x_i}$	$\frac{\alpha_i e^{\alpha_i x_i}}{e^{\alpha_i}-1}$	$\frac{\alpha_i e^{\alpha_i x_i}}{e^{\delta\alpha_i}-e^{-\delta\alpha_i}}$
$P_c(x_i\|\mathbf{h})$	$1-e^{\alpha_i x_i}$	$\frac{e^{\alpha_i x_i}-1}{e^{\alpha_i}-1}$	$\frac{e^{\alpha_i x_i}-e^{-\delta\alpha_i}}{e^{\delta\alpha_i}-e^{-\delta\alpha_i}}$
$E[x_i\|\mathbf{h}]$	$-\frac{1}{\alpha_i}$	$\frac{1}{1-e^{-\alpha_i}}-\frac{1}{\alpha_i}$	$\delta\frac{e^{\delta\alpha_i}+e^{-\delta\alpha_i}}{e^{\delta\alpha_i}-e^{-\delta\alpha_i}}-\frac{1}{\alpha_i}$

Probability of y given h

After we have $p(\mathbf{x}|h)$, we need an expression for $p(y|h)$. We follow a similar procedure using the general formula of $p(y|h)$, and evaluate the integral for different intervals as follows

$$\begin{aligned} p(y|h) &= \frac{p(y,h)}{p(h)} = \frac{\int_{\bar{x}_i} p(\mathbf{x},h,\bar{y})d\bar{\mathbf{x}}}{\int_{\bar{y}}\int_{\bar{\mathbf{x}}} p(\bar{\mathbf{x}},h,\bar{y})d\bar{\mathbf{x}}d\bar{y}} \\ &= \frac{\int_{\bar{x}_i} e^{h^T W\mathbf{x}+b^T\mathbf{x}+c^T h+D^T\bar{y}+h^T V\bar{y}}d\bar{y}}{\int_{\bar{y}}\int_{\bar{\mathbf{x}}} e^{h^T W\bar{\mathbf{x}}+b^T\bar{\mathbf{x}}+c^T h+D^T\bar{y}+h^T V\bar{y}}d\bar{\mathbf{x}}d\bar{y}} \\ &= \frac{e^{h^T Vy+D^T y}}{\int_{\bar{y}} e^{h^T V\bar{y}+d^T\bar{y}}d\bar{y}} = \prod_\kappa p(y_\kappa|h) \end{aligned} \tag{5.43}$$

where

$$p(y|h) = \frac{e^{(h^T V+D)y}}{\int_{\bar{y}} e^{(h^T V'+D)\bar{y}}d\bar{y}} \tag{5.44}$$

If we define $\gamma(h) = h^T V + D$, the evaluations of $p(y|h)$ are presented in Table 5.2:

Table 5.2: Probability expressions for $p(\mathbf{y}|\mathbf{h})$

Indicator/Interval	$[0,\infty)$	$[0,1]$	$[-\delta,\delta]$
$p(\mathbf{y}\|\mathbf{h})$	$-\gamma e^{\gamma\mathbf{y}}$	$\frac{\gamma e^{\gamma\mathbf{y}}}{e^{\gamma}-1}$	$\frac{\gamma e^{\gamma\mathbf{y}}}{e^{\delta\gamma}-e^{-\delta\gamma}}$
$P_c(\mathbf{y}\|\mathbf{h})$	$1-e^{\gamma\mathbf{y}}$	$\frac{e^{\gamma\mathbf{y}}-1}{e^{\gamma}-1}$	$\frac{e^{\gamma\mathbf{y}}-e^{-\delta\gamma}}{e^{\delta\gamma}-e^{-\delta\gamma}}$
$E[\mathbf{y}\|\mathbf{h}]$	$-\frac{1}{\gamma}$	$\frac{1}{1-e^{-\gamma}}-\frac{1}{\gamma}$	$\delta\frac{e^{\delta\gamma}+e^{-\delta\gamma}}{e^{\delta\gamma}-e^{-\delta\gamma}}-\frac{1}{\gamma}$

In this chapter, we only deal with the case where the entries of $\mathbf{x}$ and y belong to the same domain.

Probability of h given $\mathbf{x}$ and y

We let the hidden units have binary values, while $\mathbf{x}$ and y have continuous values. So $p(h|\mathbf{x},y)$ is

$$p(h|\mathbf{x},y) = \frac{p(\mathbf{x},y,h)}{p(\mathbf{x},y)} = \prod_j p(h_j|\mathbf{x},y)$$

and

$$p(h_j = 1|\mathbf{x},y) = \text{sign}\left(\sum_i w_{ji}\bar{x}_i + v_j y + c_j\right)$$

where v_j denotes the j-th element of the vector V. Obviously, no any modification is needed for this conditional probability for the continuous visible units.

Probability of y given x

When we consider the scalar case for y,the bias variable D becomes a real number, and the weight matrix V is a real valued vector. We do the same procedure as in the previous steps,

$$\begin{aligned} p(y|\mathbf{x}) &= \frac{p(\mathbf{x},y)}{p(\mathbf{x})} = \frac{\sum_{\bar{h}} p(\mathbf{x},y,\bar{h})}{\int_{\bar{y}} \sum_{\bar{h}} p(\mathbf{x},\bar{y},\bar{h}) d\bar{y}} \\ &= \frac{\sum_{\bar{h}} e^{\bar{h}^T W\mathbf{x} + b^T\mathbf{x} + c^T\bar{h} + dy + \bar{h}^T V y}}{\int_{\bar{y}} \sum_{\bar{h}} e^{h^T W\mathbf{x} + b^T\mathbf{x} + c^T\bar{h} + d\bar{y} + \bar{h}^T V\bar{y}} d\bar{y}} \end{aligned} \tag{5.45}$$

Using Fubini's Theorem, the integral and the sum are interchangeable. We then evaluate the sums on the numerator and denominator defining the term:

$$\tau_j(\mathbf{x}, y) = \sum_i w_{ji}\bar{x}_i + v_j y + c_j \tag{5.46}$$

and substituting it in (5.45),

$$p(y|\mathbf{x}) = \frac{e^{dy}\prod_j \left(1 + e^{\tau_j(\mathbf{x},y)}\right)}{\int_{\bar{y}} e^{d\bar{y}}\prod_j \left(1 + e^{\tau_j(\mathbf{x},\bar{y})}\right) d\bar{y}} \tag{5.47}$$

Probability of (y, h) given x

The second term of the negative log-likelihood can be computed as

$$\begin{aligned} p\left[(y,h)|\mathbf{x}(k)\right] &= \frac{e^{-E(\mathbf{x}(k),\bar{y},\bar{h})}}{\sum_{\bar{y},\bar{h}} e^{-E(\mathbf{x}(k),\bar{y},\bar{h})}} \\ &= \frac{e^{h^T W\mathbf{x}(k) + b^T\mathbf{x}(k) + c^T h + dy + h^T V y}}{\int_{\bar{y}} \sum_{\bar{h}} e^{h'^T W\mathbf{x}(k) + b^T\mathbf{x}(k) + c^T\bar{h} + d\bar{y} + h'^T V\bar{y}} d\bar{y}} \end{aligned} \tag{5.48}$$

The numerator can be easily computed but the integral in the denominator needs to be expanded in the following. First, we use the definition (5.46) of $\tau_j(\mathbf{x}(k), \bar{y})$,

$$\int_{\bar{y}} e^{d\bar{y}} \prod_j \left(1 + e^{\tau_j(\mathbf{x}(k),\bar{y})}\right) d\bar{y} \tag{5.49}$$

In order to find a closed form of the solution of the integral, we define $F = \{\tau_1, \tau_2, ..., \tau_n\}$ which yields to the incomplete power set $P(F)$. It is incomplete, because the empty set is not included in $P(F)$. If $P(F) = \{P_{F1}, P_{F2}, ...\}$, the elements P_{Fi} contain all possible combinations of any length of elements τ_j. The finite product $\prod_j \left(1 + e^{\tau_j(\mathbf{x}(k),\bar{y})}\right)$ can be

expressed as

$$\prod_j (1 + e^{\tau_j}) = 1 + \sum_{P_F i} e^{\sum \tau_\gamma} \tag{5.50}$$

where γ is an index for τ, which takes values such that $\tau_\gamma \in P_F i$. The integral then becomes

$$\begin{array}{c} \int_{\bar{y}} e^{d\bar{y}} \left(1 + \sum_{P_F i} e^{\sum \tau_\gamma(\mathbf{x}(k),\bar{y})}\right) d\bar{y} \\ \int_{\bar{y}} \left(e^{d\bar{y}} + \sum_{P_F i} e^{d\bar{y}+\sum \tau_\gamma(\mathbf{x}(k),\bar{y})}\right) d\bar{y} \end{array} \tag{5.51}$$

Because $\tau_j = \sum_i w_{ji}\bar{x}_i + v_j y + c_j$, we can define the vector $w_j = [w_{j1}...w_{jl}]$. So $\tau_j = w_j x + v_j y + c_j$. Considering this expression for τ_j, the value of the integral is

$$\int_{\bar{y}} \left(e^{d\bar{y}} + \sum_{P_F i} e^{\sum w_\gamma \mathbf{x}(k)+c_\gamma} e^{(d+\sum v_\gamma)\bar{y}} \right) d\bar{y} \tag{5.52}$$

The same intervals, as in the previous sections (5.53), (5.54) and (5.55), are studied for y,

- Interval $[0, \infty)$

$$-\frac{1}{D} - \sum_{P_F i} \frac{1}{D + \sum v_\gamma} e^{\sum w_\gamma \mathbf{x}(k)+c_\gamma} \tag{5.53}$$

- Interval $[0, 1]$

$$\frac{1}{d}\left(e^D - 1\right) + \sum_{P_F i} \frac{e^{\sum w_\gamma \mathbf{x}(k)+c_\gamma}}{F + \sum v_\gamma}\left(e^{D+\sum v_\gamma} - 1\right) \tag{5.54}$$

- Interval $[-\delta, \delta]$

$$\begin{array}{c} \frac{1}{d}\left(e^{D\delta} - e^{-D\delta}\right) \\ + \sum_{P_F i} \frac{e^{\sum w_\gamma \mathbf{x}(k)+c_\gamma}}{D+\sum v_\gamma}\left(e^{(D+\sum v_\gamma)\delta} - e^{-(D+\sum v_\gamma)\delta}\right) \end{array} \tag{5.55}$$

The sum $\sum_{P_F i}$ is performed along the elements of the power set which is computational expensive. The number of elements is 2^n, which represents all possible combinations. For system identification, the number of visible and hidden units is no so big, so the procedure becomes tractable.

Probability of x given (y,h)

We have shown that $p\left[\mathbf{x}|(y,h)\right] = \prod_i p(x_i|h)$. In the intervals $[0, \infty)$, $[0, 1]$ and $[-\delta, \delta]$, we get the same expressions presented in Table 1 for $p(\mathbf{x}|h)$. We use the following algorithm to

calculate $p\left[\mathbf{x}|(y,h)\right]$.

The complete algorithm is shown as follows:

Algorithm 15 *1.- For each training pair* $(\mathbf{x}(k), y(k))$ *and with learning rate* σ

2. We enter into a Gibbs sampling step.

Positive phase

3.- Assign the first values $y^{0a} = y(k)$ *and* $x^{0a} \leftarrow x(k)$

4.- Sample h^{0a} *from* $p(h|x^{0a}, y^{0a})$

5.- Assign $x^{0b} = x(k)$

6.- Sample y^{0b}, h^{0b} *from* $p(y, h|x^{0b})$

Negative phase

7.- Sample y^{1a} *from* $p(y|h^{0a})$ *and* x^{1a} *and* $p(x|h^{0a})$

8.- Sample h^{1a} *from* $p(h|x^{1a}, y^{1a})$

9.- Sample x^{1b} *from* $p(x|y^{0b}, h^{0b})$ *and* y^{1b}, h^{1b} *from* $p(y, h|x^{1b})$

Update

10 .- Apply the learning rule

$$\lambda = \lambda - \sigma \left(\frac{\partial}{\partial \lambda} E\left(x^{0a}, y^{0a}, h^{0a}\right) - \frac{\partial}{\partial \lambda} E\left(x^{0b}, y^{0b}, h^{0b}\right) \right)$$

5.4 Simulations

Gas furnace data set

One of the most utilized benchmark examples in system identification is the famous gas furnace data from the Box-Jenkins textbook [10]. In this example, the air and methane are mixed to create gas mixture which contains carbon dioxide. The control $u(k)$ of the system is methane gas, while the output $y(k)$ is CO_2 concentration. The gas furnace are sampled continuously in 9 second intervals. The data set is composed by 296 successive pairs of $[u(k), y(k)]$, where $u(k) = 0.6 - 0.4z(k)$.

The model of the gas furnace is

$$\begin{aligned} y(k) &= f[y(k-1), \ldots, y(k-n_y), u(k), \ldots, u(k-n_u)] \\ &= f[\mathbf{x}(k)] \end{aligned} \tag{5.56}$$

where the regression steps n_y and n_u are 5 and 1. 200 samples are used as training data, the rest 96 samples are the testing. The stopping criteria is not used to train the RBMs. The random search method [7] is applied to determine the structure parameters of the DBM. and each layer size l_i $(i = 1 \cdots p)$, is obtained by the random search method [7]. The search range of the layer number l is $10 \geq l \geq 3$, the node number p is $40 \geq p \geq 5$. We choose $l = 3$ and $p = 20$. The following steps are applied in dynamic system identification using RBMs and conditional distribution:

A) Normalization: the data are normalized into the interval $[0, 1]$ using

$$\mathbf{x}(k) = \frac{\mathbf{x}(k) - \min_k \{\mathbf{x}(k)\}}{\max\{\mathbf{x}(k)\} - \min_k \{\mathbf{x}(k)\}}, y = \frac{y - y_{\max}}{y_{\min} - y_{\max}} \tag{5.57}$$

the formula (5.57)

B) Coding: after the data has been normalized, we code them into a binary representation. In our experiments we used two resolutions, 4 bits and 8 bits, for $\mathbf{x}$ and y. The input number is $n_y + n_u = 6$. The resolutions of the input are 24 and 48, $\mathbf{x} \in \Re^{24}$ or $\Re^{48}$, and $y \in \Re^4$ or $\Re^8$. This new training dataset is used to train the discriminative DBM to obtain the conditional probability $p(y|\mathbf{x})$.

C) Training: the conditional DBM is trained using the coded dataset, the step number of the Gibbs sampling is $k_G = 1$, the learning rate is $\eta = 0.001$. Stochastic gradient descent (5.41) is applied over the dataset. The algorithm has 100 training epochs.

D) Decoding: in the testing phase, the output of the neural model is taken from the probability distribution $p(y|\mathbf{x})$, which is learned from the DBM conditional probability. The output data are sampled from $p(y|\mathbf{x})$ and decoded to continuous equivalent values.

The testing results are displayed in Fig. 5-3. For 4 bits and 8 bits encoding, the mean squared errors (MSE) are 11.3×10^{-3} and 8.2×10^{-3}. For this example the binary encoding has good approximation results. The high precise encoding helps to improve the model accuracy. However, adding one bit in the encoding procedure immediately doubles the computation time.

In order to show the advantages of using Boltzmann machines and conditional distributions, we added noises to the raw dataset, to show the robust and the noise resistance of our

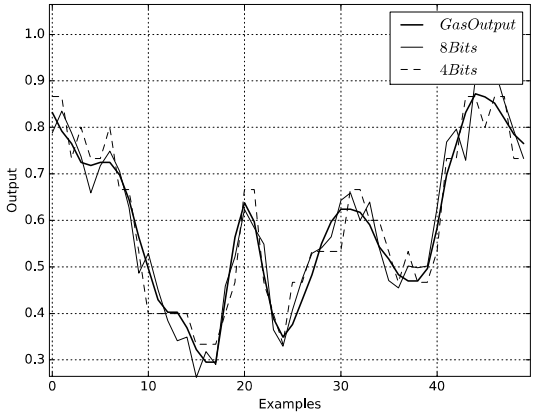

Figure 5-3: DBM modeling using 8 bits and 4 bits encoding for the gas furnace data.

models compared with standard neural network models,

$$\mathbf{x}(k) = \mathbf{x}(k) + 0.2\mathbf{z}(k) \tag{5.58}$$

where $\mathbf{z}(k) \in N(0, 0.01)$, $N(0, 0.01)$ is a normal distribution with 0 average and 0.01 standard deviation. The comparison results are shown in Table 5.3.

Table 5.3: MSEs of different identification models ($\times 10^{-3}$)

MLP	SVM-L	SVM-P	SVM-R	4 bits	8 bits
30.03	41.01	11.7	14.70	9.54	8.05

Here MLP is the multilayer perceptrons which have the same structure as our DBM, the learning algorithm is the usual backpropagation. SVM-L is the support vector machine (SVM) with linear kernel. SVM-P is the SVM with polynomial kernel. SVM-R is the SVM radial basis function kernel.

It can be seen that our models, 4 bits and 8 bits encoding DBMs, have distinctive advantages over the noises and disturbances in dynamic system identification. The main reason is that we model the probability distributions of the input and output, the noises and outliers in the data do not affect the conditional distributions significantly.

Another advantage of our models is the feature extraction by the unsupervised learning,

which is applied in most deep learning methods. The following experiments show the impact of the feature extraction in the DBM and conditional distribution. We use the same noisy input (5.58) and output data as before. The l RBMs are trained in sequence as presented in Fig. 3-5. Here we try $l = 1, 2, 3, 4$, to show the impact of each layer. The results are given in Table 5.4.

Table 5.4: MSEs of different hidden layers ($\times 10^{-3}$)

MSE	1 layer	2 layers	3 layers	4 layers
4 bits	11.62	10.36	9.54	15.67
8 bits	8.61	8.21	8.05	9.75

By adding the new feature extraction layer, the MSE drops significantly. If the hidden layer number is more than 3, the MSE becomes worse. This means that it is no longer necessary to add a new layer to extract more system information.

Now we use the continuous valued algorithm for the gas dataset. The training data are the same noisy input (5.58). We use a three layered DBM, $l = 3$, see Fig. 3-5. For the two hidden layers, the training parameters are $k_G = 1$, $\eta_1 = 0.001$, and 200 training epochs for each layer. For the output layer, we use the coded features $\mathbf{h}^3$ with $k_G = 1$ and $\eta = 0.001$, and 100 training epochs.

The testing MSE is 8.05×10^{-3}, which is better than the error obtained with the binary encoding method, this happened because it provides more information on real axis than the encoding procedure. The modeling results are shown in Fig. 5-4. There are fewer nodes because the size of the input vector is dramatically decreased without the encoding.

Wiener-Hammerstein benchmark

Wiener-Hammerstein systems [64] have a static nonlinearity surrounded by two unknown dynamic systems. Instead of a direct measurement, the samples are taken from the output of the three systems as a whole. The signal-to-noise ratio in the benchmark has big nonlinear behavior. It is a good case of study to test nonlinear system modeling techniques.

The benchmark dataset consists in $188,000$ input/output pairs. This dataset is divided in two parts: $100,000$ sample pairs are for training and $88,000$ samples are for testing. Let

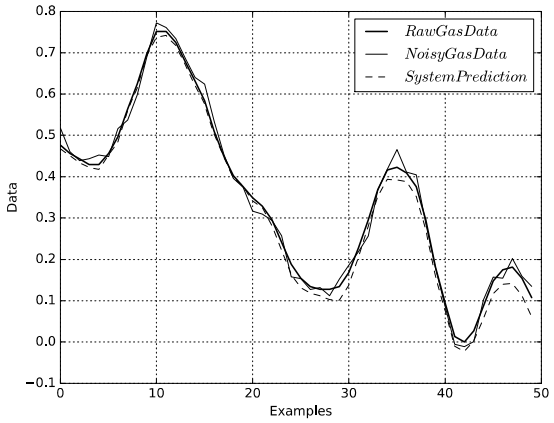

Figure 5-4: DBM modeling using continuous values for the gas furnace data.

$u(k)$ be the control and $y\,(k)$ be the output. We define the recursive input vector to the model as $\mathbf{x}(k) = [y(k-1)\cdots y(k-n_y)\ u(k)\cdots u(k-n_u)]^T$, $n_y = 10$ and $n_u = 5$.

Similar as the previous example, the random search method is used to obtain the DBM structure, here the layer number $20 \geq l \geq 2$, the node number $70 \geq p \geq 10$. This intervals are wider than the previous example, because we have more data to work. We have $l = 5$ and $p = 50$. We normalize the data to fit into the $[0, 1]$ interval. The DBM is trained using the real valued data set, the step number of the Gibbs sampling is $k_G = 1$ while the the learning rate is $\eta = 0.001$. Stochastic batch gradient descent is applied over the dataset following Algorithm 1. It has 10 training epochs. The $100,000$ sample pairs are divided into batch packages, all packages have the same size. We compare the following four types calculation methods of the probability distributions: binary, in the interval $[0, \infty)$, in the interval $[0, 1)$, and in the interval $[-\delta, \delta]$ with $1 \geq \delta > 0$.

We first show how the batch size affects the training. Fig. 5-5 shows the training performance with a batch size of 1000. The training error is high for the first 60 batches, then it decreases as more training samples are presented. The DBM cannot model the probability distribution properly with a few samples. Fig. 5-6 shows a batch size of 500. No remarkable changes appear, but there are small error increases during the training. When the batch size is increased to 5000, the fluctuations on the training error vanish, the interval $[-\delta, \delta]$ becomes unstable, see Fig. 5-7. Large batch size can affect the distribution learning in the sense that some particular training samples mislead the results. The binary DBM can work, but the

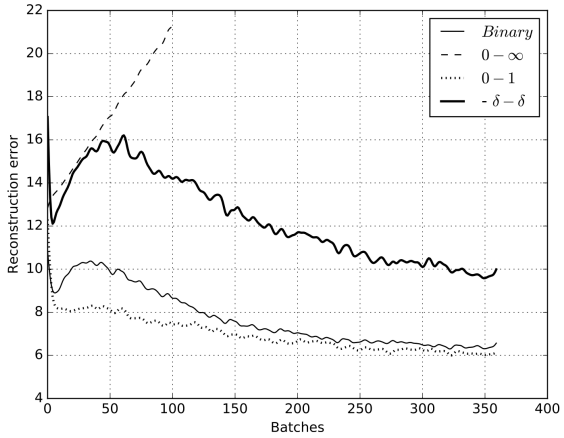

Figure 5-5: Training errors with batch size of 1000 ($\times 10^{-3}$)

MSE is high as 15.2×10^{-3}. The interval $[-\delta, \delta]$ also works, but its performance offers almost twice the error. The interval $[0, \infty)$ shows instability in all the experiments, which has the same problems as in our previous observations [**?**]. The problem may come from the integral convergence. The interval $[0, 1]$ is the best. For this example, we use the continuous valued method within the interval $[0, 1]$, the batch size is 1000.

The testing result is shown in Fig.5-8 and the testing error MSE is 5.6×10^{-3}.

Table 5.5. shows how the hidden features extraction affect and improve the identification accuracy for this benchmark problem. To see the noise influence, we also add noise as (5.58). Here $l = 4$ and $p = 50$, $k_G = 1$, $\eta_1 = 0.001$, 200 training epochs per each hidden layer, 10 training epochs for the output layer, the interval is $[0, 1)$, the batch size is 1000.

Table 5.5: MSEs of different hidden layers for WH($\times 10^{-3}$)

MSE	1 layer	2 layers	3 layers	4 layers
Without noise	5.6	4.8	4.5	6.2
With noise	6.1	5.3	4.7	6.3

As the gas furnace dataset, adding new layers to the feature extractor improves the model accuracy by eliminating the noise influence of the data. We can see that even for the best result obtained by the SVM-R, our DBM is more tolerant than others with respect to big uncertainties.

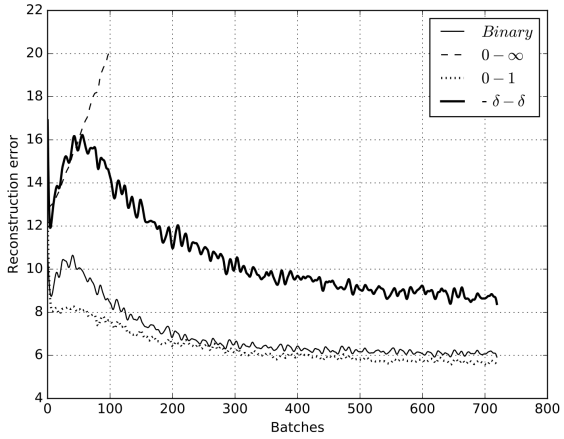

Figure 5-6: Training errors with batch size of 500 ($\times 10^{-3}$)

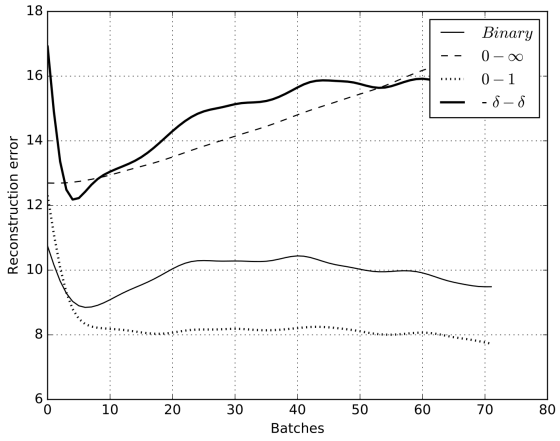

Figure 5-7: Training errors with batch size of 5000 ($\times 10^{-3}$)

Table 5.6: MSE over the WH benchmark ($\times 10^{-3}$)

MLP	SVM-L	SVM-P	SVM-R	DBM
56.03	43.01	8.01	5.71	4.70

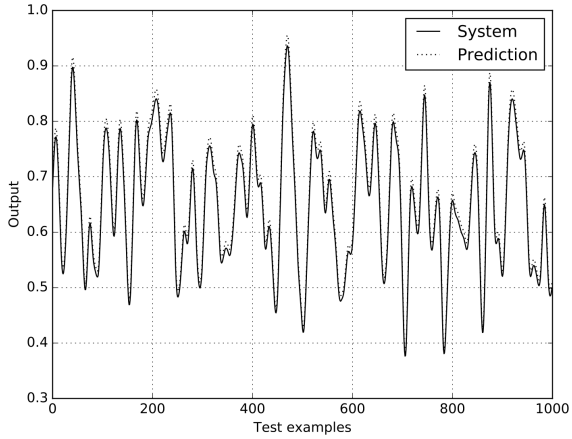

Figure 5-8: Testing error within the interval $[0, 1]$ and batch size of 1000.

Chapter 6

Conclusions

Deep learning has solved many of the problems that deep architectures had in the past The main advantage that deep architectures possess is the ability to represent complex models using fewer parameters, this feature is possible because having many layers of representation allows the model to represent high nonlinear functions compactly. It is not then surprisingly that deep architectures have become the state of the art techniques in many machine learning disciplines.

In this thesis, we have increased the reach that deep learning has by addressing the nonlinear system identification problem. We have not only shown how to use RBMs and pretraining stages to achieve better testing results but we have also used deep learning along with other famous identification techniques such as fuzzy logic and randomized algorithms. In this chapter, it is presented the final remarks about the work that has been carried out.

About deep learning and randomized algorithms

Randomized algorithms usually rely on the usage of random selection of parameters in the hidden layer, it has also been argued that this random assignment does not decrease the generalization capabilities of the model, however, we have shown that the performance of an arbitrarily selection can be easily surpassed if an RBM is chosen as a pretraining stage that finds suitable hidden weights. This is not an unexpected result because (as has been discussed before) random initialization of parameters usually leads to poor generalization performance due to the presence of local minima, therefore, using an RBM helps to move the minimization problem to a region where reaching a good local minimum is feasible for

the supervised training algorithm (in this case the pseudoinverse calculation). Moreover, we have shown that RBMs can be easily modified when the nature of the system input is continuous, this is achieved by considering a new domain in the input space that was calculated through its probability measure. With this modification an RBM is then capable of handling continuous values such as the ones that a nonlinear system produces.

A point that should be unlighted is that not all available domains of the system input worked. The solution of the normalization integral is not guaranteed for every possible domain which helps to explain why some of them failed to decrease the identification error. In conclusion, the right domain have to be chosen according to the system nature with the only restriction that it should be restricted and bounded.

About fuzzy modeling with deep learning and probability theory

It has been shown that standard fuzzy modeling can be easily improved if the input is conditionated before entering to the model. This conditioning was performed using an RBM, as has been said, it can extract useful features from the dataset that give meaningful information of the probability space from where the training examples where sampled. Then, it was proved that a probabilistic clustering method outperforms classic k-means and c-means, it seems that the features delivered by the RBM can be fully utilized by the probabilistic clustering method which come up with clusters that entirely represent the information extracted from the dataset. These clusters are more useful that the ones delivered by other clustering methods as their accuracy error suggest. Finally, it was also viewed that giving another degree of freedom to a fuzzy model can improve its efficiency, introducing a new set of probabilistic parameters allowed the model to overcome difficulties such as the multiple selection of fuzzy sets and the modeling of high variant datasets.

About conditional continuous RBMs

RBMs were proven to be conditional universal approximators that can model any conditional probability distribution if sufficient units are providen. It was also shown that binary RBMs can also deal with the identification problem if the data are encoded into binary representations (with any desire accuracy) before enter the model, this is done utilizing conditional RBMs that maximizes directly the conditional probability $p(x|y)$ which transforms the RBMs

into self-contained identification models. Moreover, conditional RBMs were also changed to deal with continuous entries which was solved by evaluating the normalization integral, this transformation achieved good testing results but with a high computational cost that grows exponentially when the number inputs increases.

Future work

Several opportunity areas were found while writing this thesis, as deep learning continues being the state of the art techniques in the machine learning community the next problems should be solved:

- There has not been done any research about the Vapnik-Chervonenkis (VC) dimensions of the deep learning algorithms. This research could help to understand the underlying mechanisms that are behind the generalization improvement caused by a pretraining stage. VC-dimensions would also give precise measurements of the deep learning boundaries as it would offer a theoretical framework where deep learning could be studied and analyzed.

- An universal approximation property for RBMs is provided in this thesis, however, it only works for binary conditional distributions. A more general universal approximation conjecture should be proved, this property would give certainty to researchers when RBMs are used to model data distributions.

- The RBMs used in this text made use of what is called *a Boltzmann distribution*, other distributions should be explored. It is possible that the election of the right distribution is one of the main factors to achieve a low testing error.

- In this thesis we only used simple stochastic gradient descent to train the RBMs, however there exist other methods that should be tested. These methods include Hessian techniques, regularization, early stopping and cross-validation criteria.

- A nonlinear system is of course a dynamic one that can be represented using a differential equation (continuous case) or a equation in differences (discrete case). These representations are not included in the deep learning environment which suggests that

deep learning should be amplified including ideas such as deep recurrent neural networks or deep dynamic neural models. These additions would make possible to predict better the behavior of a system given that their respective natures could be represented with more precision.

Bibliography

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, A learning algorithm for boltzmann machines, *Cognitive Science*, 9 (1985) 147-169.

[2] M. Alhamdoosh, D.Wang, Fast decorrelated neural network ensembles with random weights, *Information Sciences*, 264 (2014) 104-117.

[3] P.Angelov, An approach for fuzzy rule-base adaptation using on-line clustering, *International Journal of Approximate Reasoning*, Vol.35, No.3,275-289, 2004.

[4] Y. Bengio and O. Delalleau, Justifying and generalizing contrastive divergence, *Neural Computation*, 21 (6) (2009) 1601-1621.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, Greedy layer-wise training of deep networks, *Advances in Neural Information Processing Systems (NIPS'06)*, pp. 153-160, MIT Press, 2007.

[6] Y. Bengio and Y. LeCun, Scaling learning algorithms towards AI, *Large Scale Kernel Machines*, MIT Press, 2007.

[7] J. Bergstra, Y. Bengio, Random Search for Hyper-Parameter Optimization, *Journal of Machine Learning Research*, (2011) 281-305.

[8] G. Box, G. Jenkins, G. Reinsel, *Time Series Analysis: Forecasting and Control*, 4th Ed, Wiley, 2008.

[9] E. Busseti, I. Osband and S. Wong, Deep learning for time series modeling, *CS 229 Technical Report*, Stanford University, 2012

[10] G. Box, G. Jenkins, G. Reinsel, *Time Series Analysis: Forecasting and Control*, 4th Ed, Wiley, 2008.

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

[12] S. Chakrabartty, ; R. K. Shaga, K. Aono, Noise-Shaping Gradient Descent-Based Online Adaptation Algorithms for Digital Calibration of Analog Circuits, *IEEE Transactions on Neural Networks and Learning Systems*, 24 (4) (2013) 554-565.

[13] S. Chen and S.A. Billings, Neural networks for nonlinear system modelling and identification, *International Journal of Control*, 1992, 56(2), pp. 319-346.

[14] P.Chen, C.Y.Zhang, L.Chen, M.Gan , Fuzzy Restricted Boltzmann Machine for the Enhancement of Deep Learning, *IEEE Transactions on Fuzzy Systems,* Vol.23, No.6, pp.2163-2173, 2015.

[15] J-H. Chiang, P-Y. Hao, Support Vector Learning Mechanism for Fuzzy Rule-Based Modeling: A New Approach, *IEEE Transactions on Fuzzy Systems*, Vol. 12, No. 1, 2004.

[16] S.L.Chiu, Fuzzy Model Identification based on cluster estimation, *Journal of Intelligent and Fuzzy Systems*, Vol.2, No.3, 1994.

[17] R. Collobert and J. Weston, A unified architecture for natural language processing: Deep neural networks with multitask learning, *25th International Conference on Machine Learning*, pp. 160-167, ACM, 2008.

[18] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y., Online passive-aggressive algorithms, *JMLR* pp. 551-585, 2006

[19] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines:* Cambridge Univ. Press, 2000.

[20] G.Cybenko, Approximation by Superposition of Sigmoidal Activation Function, *Math.Control, Sig Syst*, Vol.2, 303-314, 1989

[21] K. De Brabanter, P. Dreesen, P. Karsmakers, K. Pelckmans, J. De Brabanter, J.A.K. Suykens and B. De Moor, Fixed-size LS-SVM applied to the Wiener-Hammerstein benchmark, *In Proceedings of the 15th IFAC Symposium on System Identification,* (2009) 826-831.

[22] E. de la Rosa, Deep learning for nonlinear systems identification (Masters dissertation), CINVESTAV-IPN, 2014, Retrieved from *http://www.ctrl.cinvestav.mx/~yuw/pdf/MaTesER.pdf*

[23] D. Erhan, Y. Bengio, A. Courville, P. A. Manzagol, P.Vincent, Why Does Unsupervised Pre-training Help Deep Learning?, *Journal of Machine Learning Research*, 11 (2010) 625-660.

[24] G. Chen, Deep learning with nonparametric clustering, *arXiv:1501.03084*, 2015

[25] X.Gu, F-L.Chung, Hi.Ishibuchi, S.Wang, Imbalanced TSK Fuzzy Classifier by Cross-Class Bayesian Fuzzy Clustering and Imbalance Learning, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, DOI: 10.1109/TSMC.2016.2598270, 2016.

[26] S.Haykin, *Neural Networks- A Comprehensive Foundation*, Macmillan College Publ. Co., New York, 1994.

[27] G. E. Hinton, S. Osindero, and Y. Teh, A fast learning algorithm for deep belief nets, *Neural Computation*, vol. 18, pp. 1527-1554, 2006.

[28] G. E. Hinton and T. J. Sejnowski, Learning and relearning in Boltzmann machines, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, Cambridge, MA: MIT Press, (1986) 282-317.

[29] H. Hjalmarsson,C.R.Rojas, D.E.Rivera, System identification: A Wiener-Hammerstein benchmark, *Control Engineering Practice*, 20 (2012) 1095-1096,

[30] C-H.Hu, X-S.Si, J-B.Yang, Z-J.Zhou, Online Updating With a Probability-Based Prediction Model Using Expectation Maximization Algorithm for Reliability Forecasting, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* Volume: 41, Issue: 6, Pages: 1268 - 127, 2011

[31] G.B. Huang, L. Chen, and C. K. Siew, Universal approximation using incremental feedforward networks with arbitrary input weights, *Technical Report ICIS/46/2003, (School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore)*, 2003.

[32] G-B.Huang, Q-Y.Zhu, C-K.Siew, Extreme learning machine: theory and applications, *Neurocomputing,* vol. 70, no.1, pp.489-501, 2006.

[33] B. Igelnik and Y-H.Pao, Stochastic Choice of Basis Functions in Adaptive Function Approximation and the Functional-Link Net, *IEEE Transactions on Neural Networks*, 6 (2) (1995) 1320-1329.

[34] S. Jagannathan and F. L. Lewis, Identification of Nonlinear Dynamical Systems Using Multilayered Neural Networks, *Automatica*, 32 (12) (1996) 1707-1712.

[35] J. S. Jang, ANFIS: Adaptive-network-based fuzzy inference system, *IEEE Transactions on Systems, Man and Cybernetics*, 23 (1993) 665–685.

[36] C.F.Juang, Combination of on-line clustering and Q-value based GA for reinforcement fuzzy system design, *IEEE Transactions on Fuzzy Systems*, Vol.13, No.3, 289- 302, 2005

[37] X.Jin, J.Shao, X.Zhang, W.An, R.Malekian, Modeling of nonlinear system based on deep learning framework, *Nonlinear Dynamics,* Volume 84, Issue 3, pp 1327-1340, 2016

[38] M.Kumar, A.Insan, N.Stoll, K.Thurow, R.Stoll, Stochastic Fuzzy Modeling for Ear Imaging Based Child Identification, *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* Volume: 46, Issue: 9 Pages: 1265 - 1278, 2016.

[39] M. Längkvist, L. Karlsson, and A. Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters* 42: 11-24. 2014.

[40] H. Larochelle and Y. Bengio, Classification using discriminative restricted Boltzmann machines, *Proceedings of the 25th International Conference on Machine Learning*, 536-543, 2008

[41] H.K.Lam, Design of stable fuzzy controller for non-linear systems subject to imperfect premise matching based on grid-point approach, *IET Control Theory & Applications*, Vol.4 , No.12, 2770-2780, 2010

[42] Y.LeCun, Y.Bengio, G. E.Hinton, Deep learning, *Nature*, 521 (7553): 436-444, 2015

[43] Y. LeCun, L.Bottou, Y.Bengio, and P.Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol.86, No.11, 2278-2324,1998

[44] N. Le Roux and Y. Bengio, Representational power of restricted Boltzmann machines and deep belief networks, *Neural Computation*, Vol. 20, 1631-1649, 2008

[45] J.M. Leski, TSK-Fuzzy Modeling Based on ε-Insensitive Learning, *IEEE Trans. on Fuzzy System*, vol. 13, no. 2, pp181-193, 2005.

[46] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, *IEEE Transactions on Neural Networks*, 14 (2003) 79-88.

[47] J.Li, S.Ray, B.G.Lindsay, A Nonparametric Statistical Approach to Clustering via Mode Identification, *Journal of Machine Learning Research*, Vol.8, 1687-1723, 2007

[48] C.-T. Lin and C-.S.G. Lee, Neural network-based fuzzy logic control and decision system, *IEEE Trans. Comput.*, vol 40 pp. 1320-1336, 1991.

[49] Y. Liu, K.Chan, K.A.Hua, Hybrid Manifold Embedding, *IEEE Transactions on Neural Networks and Learning Systems*, 25 (12) (2014) 2295 - 2302.

[50] Z.Liu and H-X.Li, Probabilistic Fuzzy Logic System for Modeling and Control, *IEEE Trans. on Fuzzy System*, vol. 13, no. 6, pp848-859, 2005.

[51] L.Ljung, *System Identification-Theory for User*, Prentice Hall, Englewood Cliffs, NJ 07632, 1987.

[52] D. Marquardt, An Algorithm for Least-Squares Estimation of Nonlinear Parameters, *SIAM Journal on Applied Mathematics*, 11 (2) (1963) 431-441.

[53] M. Minsky, S. Papert, *Perceptrons: an introduction to computational geometry*, Cambridge: MIT Press; 1969.

[54] S. Mitra adn Y. Hayashi, Neuro–fuzzy rule generation: survey in soft computing framework, IEEE Transactions on Neural Networks 11 (3) (2000) 748-769.

[55] T.M. Nabhan, A.Y. Zomaya, Toward generating neural network structures for function approximation, *Neural Networks*, Vol.7, No.1, pp. 89-99, 1994

[56] K. S. Narendra and K. Parthasarathy, Gradient methods for optimization of dynamical systems containing neural networks, *IEEE Transactions on Neural Networks,* 3 (2) (1991) 252-262.

[57] K.Noori, K.JenabFuzzy Reliability-Based Traction Control Model for Intelligent Transportation Systems, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Volume: 43, Issue: 1, Pages: 229 - 234, 2013.

[58] I. Rivals and L. Personnaz, Neural-network construction and selection in nonlinear modeling, *IEEE Transactions on Neural Networks*, 14 (4) (2003) 804-820.

[59] P. Romeu, et al. Time-Series Forecasting of Indoor Temperature Using Pre-trained Deep Neural Networks. *Artificial Neural Networks and Machine Learning–ICANN 2013*. Springer Berlin Heidelberg, 451-458. 2013.

[60] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, *Psychol Rev*, 1958;65(6):386–408.

[61] R.Salakhutdinov, G. E.Hinton, An Efficient Learning Procedure for Deep Boltzmann Machines, *Neural Computation,* Vol.24, 1967-2006, 2012

[62] R.Salakhutdinov, G. E.Hinton, Deep Boltzmann Machines, *12th International Conference on Artificial Intelligence and Statistics (AISTATS),* Clearwater Beach, Florida, USA, 2009

[63] W. F. Schmidt, M. A. Kraaijveld, R. P. W. Duin, Feedforward neural networks with random weights, *11th IAPR International Conference on Pattern Recognition*, pp. 1-4, The Hague, Netherlands , 1992.

[64] J. Schoukens, J. Suykens, L. Ljung, Wiener-Hammerstein benchmark, *15th IFAC Symposiumon System Identification*, Saint-Malo, France, 2009.

[65] Q. Song, Robust Initialization of a Jordan Network With Recurrent Constrained Learning, *IEEE Transactions on Neural Networks*, 22 (12) (2011) 2460-2473.

[66] D. Sosulski, M. Bloom ML, T. Cutforth, R. Axel , S. Datta, Distinct representations of olfactory information in different cortical centres, *Nature*, 2011;472:213–6.

[67] M. Sugeno, T. Yasukawa, A Fuzzy Logic Based Approach to Qualitative Modeling, *IEEE Trans.on Fuzzy Systems*, 1 (1) (1993) 7-31.

[68] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Syst., Man. and Cybern.*, vol. 1, pp. 116-132, Jan. 1985.

[69] S. Tamura and M. Tateishi, Capabilities of a four-layered feedforward neural network: Four layers versus three, *IEEE Transactions on Neural Networks,* vol. 8, no. 2, pp. 251—255, 1997.

[70] J. Tapson and A. van Schaik, Learning the pseudoinverse solution to network weights, *Neural Networks*, 45 (2013) 94-100.

[71] S.G.Tzafestas and K.C.Zikidis, NeuroFAST: On-line neuro-fuzzy ART-based structure and parameter learning TSK model, *IEEE Transactions on Systems, Man and Cybernetics*, Part B, Vol.31, No.5, 797-803, 2001.

[72] P. E. Utgoff and D. J. Stracuzzi, Many-layered learning, *Neural Computation*, vol. 14, pp. 2497–2539, 2002.

[73] L.Waltman, U.Kaymak, J.Berg, Maximum likelihood parameter estimation in probabilistic fuzzy classifiers, *14th IEEE International Conference on Fuzzy Systems*, 1098-1103, 2005.

[74] L.X.Wang, *Adaptive Fuzzy Systems and Control*, Englewood Cliffs NJ: Prentice-Hall, 1994.

[75] L. Wang and R. Langari, Complex Systems Modeling via Fuzzy Logic, *IEEE Trans. on Syst., Man, and Cybernetics*, 26 (1) (1996) 100-106.

[76] W. Yu, X. Li, Automated Nonlinear System Modeling with Multiple Fuzzy Neural Networks and Kernel Smoothing, *International Journal of Neural Systems*, 20 (5) (2010) 429-435.

[77] W. Yu, X. Li, Online fuzzy modeling with structure and parameter learning, *Expert Systems With Applications*, Vol. 36, 7484-7492, 2009

[78] L.A. Zadeh, A note on Z-numbers, *Information Sciences,* Vol. 181, pp.2923-2932, 2011.

[79] L.A. Zadeh, "Fuzzy sets". *Inf. Control*, vol 8, pp 338-353, Aug, 1998.

[80] H.Zhang, M.Li, J.Yang, D.Yang, Fuzzy Model-Based Robust Networked Control for a Class of Nonlinear Systems,*IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* Volume: 39, Issue: 2, Pages: 437 - 447, 2009.