

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
INSTITUTO DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MILTON ROBERTO HEINEN

**A Connectionist Approach for Incremental  
Function Approximation and On-line Tasks**

Thesis presented in partial fulfillment  
of the requirements for the degree of  
Doctor of Computer Science

Prof. Dr. Paulo Martins Engel  
Advisor

Porto Alegre, March 2011

## CIP – CATALOGING-IN-PUBLICATION

Heinen, Milton Roberto

A Connectionist Approach for Incremental Function Approximation and On-line Tasks / Milton Roberto Heinen. – Porto Alegre: PPGC da UFRGS, 2011.

172 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2011. Advisor: Paulo Martins Engel.

1. Machine learning. 2. Artificial neural networks. 3. Incremental learning. 4. Bayesian methods. 5. Gaussian mixture models. 6. Function approximation. 7. Regression. 8. Clustering. 9. Reinforcement learning. 10. Autonomous mobile robots. I. Engel, Paulo Martins. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitor: Prof. Carlos Alexandre Netto

Vice-Reitor: Prof. Rui Vicente Oppermann

Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion

Diretor do Instituto de Informática: Prof. Flávio Rech Wagner

Coordenador do PPGC: Prof. Álvaro Freitas Moreira

Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

*“I do not know what I may appear to the world, but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.”*

— SIR ISAAC NEWTON

*“Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius – and a lot of courage – to move in the opposite direction.”*

*“I have no special talent. I am only passionately curious.”*

— ALBERT EINSTEIN

*“Intelligence is the ability to adapt to change.”*

— STEPHEN HAWKING



## **ACKNOWLEDGEMENTS**

I am grateful to my advisor, Dr. Paulo Martins Engel, for the ideas that led to this thesis, for his guidance, encouragement, valuable comments, support and patience throughout the course of this work. I thank the professors and teachers of the Informatics Institute, for transmitting the knowledge necessary to accomplish this journey, and to Dr. Fernando Santos Osório, for his continuing support and valuable advice. Another great debt is to my colleagues of the Neural Networks group for discussions, comments, and support. Thanks to Edson Prestes for lending us his Pioneer 3-DX robot which was used in some experiments. I also acknowledge the financial support granted by CNPq.

Finally I would like to acknowledge my thanks to my family. My parents, Arno José Heinen and Melânia Heinen, supported me with their encouragement and comprehension. My dearest lovely wife, Alessandra Hüther Heinen, provided me constant support which helped me to overcome many difficulties on the way to completing this journey.

I dedicate this work to my dearest lovely wife, Alessandra Hüther Heinen, who during all these years has always been by my side, especially when I most needed her support, fondness and love.



# CONTENTS

<b>LIST OF ABBREVIATIONS AND ACRONYMS</b> . . . . .	9
<b>LIST OF SYMBOLS</b> . . . . .	11
<b>LIST OF FIGURES</b> . . . . .	15
<b>LIST OF TABLES</b> . . . . .	17
<b>ABSTRACT</b> . . . . .	19
<b>RESUMO</b> . . . . .	21
<b>1 INTRODUCTION</b> . . . . .	23
1.1 <b>Function approximation</b> . . . . .	25
1.2 <b>Theoretical and biological inspiration</b> . . . . .	26
1.3 <b>Main objectives and contributions of this thesis</b> . . . . .	30
1.4 <b>Outline</b> . . . . .	31
<b>2 GAUSSIAN MIXTURE MODELS</b> . . . . .	33
2.1 <b>Gaussian mixture models</b> . . . . .	33
2.2 <b>K-means clustering algorithm</b> . . . . .	34
2.3 <b>The EM algorithm</b> . . . . .	35
2.4 <b>State-of-the-art about learning Gaussian mixture models</b> . . . . .	36
2.5 <b>Incremental Gaussian Mixture Model</b> . . . . .	40
2.5.1 <b>Tackling the Stability-Plasticity Dilemma</b> . . . . .	41
2.5.2 <b>Model update by sequential assimilation of data points</b> . . . . .	42
2.5.3 <b>Creating a new component based on novelty and stability</b> . . . . .	47
2.5.4 <b>Incremental learning from unbounded data streams</b> . . . . .	48
2.5.5 <b>IGMM learning algorithm</b> . . . . .	49
2.5.6 <b>Computational complexity of IGMM</b> . . . . .	50
2.6 <b>Comparative</b> . . . . .	51
<b>3 ARTIFICIAL NEURAL NETWORKS</b> . . . . .	53
3.1 <b>Multi-layer Perceptrons</b> . . . . .	53
3.2 <b>Radial basis function networks</b> . . . . .	54
3.3 <b>Self-organizing maps</b> . . . . .	55
3.4 <b>Adaptive Resonance Theory</b> . . . . .	56
3.5 <b>ARTMAP</b> . . . . .	59
3.6 <b>Probabilistic neural networks</b> . . . . .	61

<b>3.7</b>	<b>General regression neural network</b>	63
<b>3.8</b>	<b>Improvements made over PNN and GRNN</b>	65
<b>3.9</b>	<b>Other neural network models</b>	69
<b>3.10</b>	<b>Comparison among the described ANN models</b>	70
<b>4</b>	<b>INCREMENTAL GAUSSIAN MIXTURE NETWORK</b>	73
<b>4.1</b>	<b>General regression using Gaussian mixture models</b>	73
<b>4.2</b>	<b>IGMN architecture</b>	76
<b>4.3</b>	<b>IGMN operation</b>	78
4.3.1	Learning mode	79
4.3.2	Recalling mode	82
<b>4.4</b>	<b>Dealing with multi-valued target data</b>	84
<b>4.5</b>	<b>IGMN configuration parameters</b>	88
<b>4.6</b>	<b>Final remarks</b>	88
<b>5</b>	<b>FUNCTION APPROXIMATION USING IGMN</b>	91
<b>5.1</b>	<b>Sinusoidal data set</b>	91
5.1.1	Standard $\times$ multivariate representation	92
5.1.2	Assessing the sensibility of the $\delta$ parameter	96
5.1.3	Ordered $\times$ shuffled data sets	96
5.1.4	Estimating the confidence intervals	100
5.1.5	Comparison with other ANN models	101
<b>5.2</b>	<b>Approximating the “Mexican hat” function</b>	102
<b>5.3</b>	<b>Estimating both a and b</b>	104
<b>5.4</b>	<b>Estimating the outputs of a nonlinear plant</b>	109
<b>5.5</b>	<b>Predicting future values of a time series</b>	112
<b>5.6</b>	<b>Final remarks</b>	116
<b>6</b>	<b>ROBOTICS AN OTHER RELATED TASKS</b>	119
<b>6.1</b>	<b>Incremental concept formation</b>	119
6.1.1	Related work	120
6.1.2	Concept formation experiments	122
<b>6.2</b>	<b>Estimating the desired speeds in a mobile robotics application</b>	124
<b>6.3</b>	<b>Computing the inverse kinematics in a legged robot task</b>	128
<b>6.4</b>	<b>Reinforcement Learning using IGMN</b>	130
6.4.1	Related work	131
6.4.2	Selecting the robot actions using IGMN	131
6.4.3	Pendulum with limited torque	132
6.4.4	Robot soccer task	133
<b>6.5</b>	<b>Feature-based mapping using IGMN</b>	135
6.5.1	Related work	136
6.5.2	Incremental feature-based mapping using IGMN	137
6.5.3	Experiments	140
<b>7</b>	<b>CONCLUSION AND FUTURE WORK</b>	147
<b>7.1</b>	<b>Future work</b>	149
	<b>REFERENCES</b>	151
	<b>APPENDIX – PRINCIPAIS CONTRIBUIÇÕES DA TESE</b>	169



## LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AMNN	Attentional Mode Neural Network
ANN	Artificial Neural Network
ARAVQ	Adaptive Resource Allocating Vector Quantization
ART	Adaptive Resonance Theory
ARTMAP	Predictive ART
BP	Back-Propagation
DDA	Dynamic Decay Adjustment
EM	Expectation-Maximization
FPGA	Field Programmable Gate Array
FPNN	Fuzzy Probabilistic Neural Network
GA	Genetic Algorithm
GC	Gaussian Clustering
GF	Generalized Fisher
GMM	Gaussian Mixture Model
GRNN	General Regression Neural Network
GTSOM	Growing Temporal Self Organizing Map
HMM	Hidden Markov Models
IGMM	Incremental Gaussian Mixture Model
IGMN	Incremental Gaussian Mixture Network
INBC	Incremental Naïve Bayes Clustering
IPNN	Incremental Probabilistic Neural Network
LMS	Least Mean Squares
LWR	Locally Weighted Regression
MDL	Minimum Description Length
ML	Maximum Likelihood

MLP	Multi-Layer Perceptron
MPF	Memory Prediction Framework
MSE	Mean Square Error
NN	Neural Network
NRMS	Normalized Root Mean Squared
ODE	Open Dynamics Engine
pdf	Probability density function
PDBNN	Probabilistic Decision-Based Neural Network
PNN	Probabilistic Neural Network
RAN	Resource Allocating Network
RBF	Radial Basis Function
RL	Reinforcement Learning
RMS	Root Mean Squared
R-LLGMN	Recurrent Log-Linearized Gaussian Mixture Network
SLAM	Simultaneous Localization and Mapping
SOM	Self-Organizing Maps
VB	Variational Bayes
WPNN	Weighted Probabilistic Neural Network

## LIST OF SYMBOLS

<b>a</b>	Sensory/motor stimulus of the first cortical region
<b>b</b>	Sensory/motor stimulus of the second cortical region
$\langle \mathbf{b}   \mathbf{a} \rangle$	Regression of <b>b</b> conditioned on <b>a</b>
<b>C</b>	Variance/covariance matrix
$\mathbf{C}_j$	Complete $j$ th covariance matrix of <b>z</b>
$\mathbf{C}_j^{\mathcal{A}\mathcal{A}}$	Submatrix containing the rows and columns of $\mathcal{A}$ in $\mathbf{C}_j$
$\mathbf{C}_j^{\mathcal{A}\mathcal{B}}$	Submatrix containing the rows corresponding to $\mathcal{A}$ and the columns corresponding to $\mathcal{B}$ in $\mathbf{C}_j$
$\mathbf{C}_j^{\mathcal{B}\mathcal{A}}$	Association matrix: $\mathbf{C}_j^{\mathcal{B}\mathcal{A}} = \mathbf{C}_j^{\mathcal{A}\mathcal{B}T}$
$\mathbf{C}_j^{\mathcal{B}\mathcal{A}*}$	New (updated) association matrix
$\mathbf{C}_j^{\mathcal{B}\mathcal{B}}$	Submatrix containing the rows and columns of $\mathcal{B}$ in $\mathbf{C}_j$
$\mathbf{C}_j^{\mathcal{X}}$	Covariance matrix of the $j$ th unit of region $\mathcal{N}^{\mathcal{X}}$
$\mathbf{C}_j^{\mathcal{X}*}$	New (updated) covariance matrix $j$ of region $\mathcal{N}^{\mathcal{X}}$
$\hat{\mathbf{C}}^{\mathcal{X}}$	Estimated covariance matrix of <b>b</b>
$D$	Dimensionality of <b>z</b>
$D^{\mathcal{X}}$	Dimensionality of the sensory/motor stimulus <b>k</b>
$\mathcal{D}_{\mathcal{M}}(\cdot)$	Mahalanobis distance
$E_{lg}$	Discrepancies between the local and global models
$e$	Euler number: $e = 2.718281828$
$f(\cdot)$	Forward function
$f(\cdot)^{-1}$	Inverse function
<b>I</b>	Identity matrix
$j$	Current Gaussian distribution
<b>K</b>	Data sequence of $N$ input vectors: $\mathbf{K} = \{\mathbf{k}^1, \dots, \mathbf{k}^n, \dots, \mathbf{k}^N\}$
$\mathcal{K}$	Sample space of <b>K</b>
<b>k</b>	Sensory/motor stimulus of the $k$ th cortical region

$\hat{\mathbf{k}}$	Estimate of the sensory/motor stimulus $\mathbf{k}$
$\mathcal{L}(\boldsymbol{\theta})$	Likelihood of $\boldsymbol{\theta}$ for a given $\mathbf{K}$
$M$	Number of Gaussian units
$M^*$	New (updated) number of Gaussian units
$N$	Number of training patterns
$\mathcal{N}^{\mathcal{A}}$	First cortical region of IGMN
$\mathcal{N}^{\mathcal{B}}$	Second cortical region of IGMN
$\mathcal{N}^{\mathcal{K}}$	$k$ th cortical region of IGMN
$\mathcal{N}^{\mathcal{S}}$	Last cortical region of IGMN
$\mathcal{N}(\mathbf{k} \boldsymbol{\mu}, \mathbf{C})$	Multivariate Gaussian distribution
$n$	Current training pattern
$\mathcal{P}$	Associative region
$p(\mathbf{a}, \mathbf{b})$	Joint density
$\hat{p}(\mathbf{a}, \mathbf{b})$	Estimate of $p(\mathbf{a}, \mathbf{b})$
$p(\mathbf{b} \mathbf{a})$	Conditional density
$p(j)$	A priori probability of $j$ th Gaussian unit
$p(j)^*$	New (updated) a priori probability of $j$ th Gaussian unit
$p(j \mathbf{a}, \mathbf{b}, \dots, \mathbf{s})$	Joint posterior probability: $p(j \mathbf{a}, \mathbf{b}, \dots, \mathbf{s}) = p(j \mathbf{z})$
$p(j \mathbf{k})$	Posterior probability of $\mathbf{k}$ have been generated from $j$
$p(\mathbf{k} j)$	Probability density function
$p(\mathbf{k} j, \ell)$	Probability density function of the $j$ th non-discarded component
$p(\mathbf{k} \boldsymbol{\theta})$	Component density function
$p(\ell \mathbf{a})$	Probability of the Maximum likelihood (ML) hypothesis $\ell$
$Q_{max}$	Maximum $Q$ value currently stored in the Gaussian units
$Q(s, a)$	Value of the state-action pair $(s, a)$
$\mathbf{s}$	Sensory/motor stimulus of the last cortical region
$sp_j$	Accumulator of the a posteriori probabilities of the $j$ th unit
$sp_j^*$	New (updated) $j$ th accumulator of the a posteriori probabilities
$sp_{max}$	Configuration parameter used to set maximum value of $sp$
$sp_{min}$	Configuration parameter used to identify spurious components
$t$	Current time step
$V(s)$	Value of the state $s$
$\bar{\mathbf{x}}_j^{\mathcal{K}}$	$\bar{\mathbf{x}}_j^{\mathcal{K}} = \boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})$
$(x, y, \phi)$	Pose of a robotic actuator

$\mathbf{z}$	Combination of all sensory/motor stimuli: $\mathbf{z} = \{\mathbf{a}, \mathbf{b}, \dots, \mathbf{s}\}$
$\alpha_{max}$	Maximum difference between the orientations of equivalent clusters
$\alpha_{min}$	Minimum angle between the sensor beam and the cluster orientation
$\beta$	Configuration parameter used to restart the $sp$ accumulators
$\gamma$	Configuration parameter used to restart the $sp$ accumulators
$\delta$	Configuration parameter used to set the initial radius of the Gaussian units
$\epsilon$	Independent Gaussian noise vector
$\varepsilon$	Instantaneous normalized approximation error
$\varepsilon_{max}$	Configuration parameter used to set maximum approximation error
$\eta_{max}$	Maximum Mahalanobis distance between equivalent clusters
$\theta$	Set of parameters of the GMM: $\theta = (\theta_1, \dots, \theta_j, \dots, \theta_M)^T$
$\theta_j$	Set of parameters of the $j$ th Gaussian distribution: $\theta_j = \{\boldsymbol{\mu}_j, \mathbf{C}_j, p(j)\}$
$\mu$	Mean
$\boldsymbol{\mu}_j^{\mathcal{X}}$	Mean vector of the $j$ th unit of region $\mathcal{N}^{\mathcal{X}}$
$\boldsymbol{\mu}_j^{\mathcal{X}*}$	New (updated) mean vector $j$ of the cortical region $\mathcal{N}^{\mathcal{X}}$
$\mu_\epsilon$	Mean of $\epsilon$
$\rho_j$	Density of the $j$ th cluster
$\rho_{min}$	Minimum cluster density (used to verify if a cluster is spurious)
$\sigma$	Standard deviation or <i>spreading</i> parameter of GRNN
$\sigma^2$	Variance
$\boldsymbol{\sigma}_{ini}$	Initial radius of the covariance matrices
$\boldsymbol{\sigma}_j^{\mathcal{X}}$	Vector containing the $j$ th standard deviation of $\mathbf{k}$
$\hat{\boldsymbol{\sigma}}_{\mathcal{X}}^2$	Estimated variance of $\mathbf{k}$
$\sigma_\epsilon$	Standard deviation of $\epsilon$
$\tau_{nov}$	Configuration parameter used to set the minimum likelihood criterion
$\nu_j$	Age of the Gaussian unit $j$
$\nu_{min}$	Configuration parameter used to set the minimum age
$\Omega$	Configuration parameter used to discard secondary branches
$\omega_j$	$\omega_j = p(j \mathbf{z}^t)/sp_j^*$
$\ell$	Maximum likelihood (ML) hypothesis
$\mathbf{0}$	Zero matrix
*	Subscript that indicate the new (updated) parameters



## LIST OF FIGURES

Figure 1.1:	Information flow in traditional connectionist models that follows the <i>information system metaphor</i> . . . . .	27
Figure 1.2:	Information flow in embodied systems: there is a closed loop of information between perception and action . . . . .	28
Figure 1.3:	Information flows up and down sensory hierarchies to form predictions and create a unified sensory experience . . . . .	29
Figure 3.1:	Typical architecture of ART1 . . . . .	57
Figure 3.2:	ARTMAP architecture . . . . .	60
Figure 3.3:	PNN for classification of patterns in two categories . . . . .	62
Figure 3.4:	Architecture of the GRNN model . . . . .	64
Figure 4.1:	IGMN architecture . . . . .	77
Figure 4.2:	Information flow through the neural network during learning . . . . .	80
Figure 4.3:	Information flow in IGMN during recalling . . . . .	83
Figure 4.4:	A typical inverse problem in which the target data are multi-valued . . . . .	86
Figure 5.1:	Sinusoidal data set of size $N = 1000$ corrupted with Gaussian noise . . . . .	92
Figure 5.2:	Experiments performed over the sinusoidal data set using $\varepsilon_{max} = 0.075$ . . . . .	94
Figure 5.3:	Experiments performed over the sinusoidal data set using $\varepsilon_{max} = 0.025$ . . . . .	94
Figure 5.4:	Comparison between standard and multivariate regression . . . . .	95
Figure 5.5:	Assessing the sensibility of $\delta$ – Standard IGMN algorithm . . . . .	97
Figure 5.6:	Assessing the sensibility of $\delta$ – Multivariate IGMN algorithm . . . . .	97
Figure 5.7:	Ordered $\times$ shuffled data sets – Standard algorithm . . . . .	98
Figure 5.8:	Ordered $\times$ shuffled data sets – Multivariate algorithm . . . . .	98
Figure 5.9:	Gaussian units added during learning – Multivariate algorithm . . . . .	99
Figure 5.10:	Gaussian units added during learning – Standard algorithm . . . . .	100
Figure 5.11:	Confidence estimates computed by IGMN over the sinusoidal data set . . . . .	100
Figure 5.12:	Comparative of the NRMS error among ANNs – sinusoidal data set . . . . .	102
Figure 5.13:	Approximating the “Mexican hat” function using IGMN . . . . .	103
Figure 5.14:	Hyperbolic tangent data set corrupted with noise . . . . .	105
Figure 5.15:	Estimating both $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ in a hyperbolic tangent data set . . . . .	105
Figure 5.16:	Experiments performed over the cubic data set . . . . .	106
Figure 5.17:	Estimating both $\hat{\mathbf{a}}$ from $\mathbf{b}$ in the cubic data set . . . . .	106
Figure 5.18:	Circular data set used to evaluate IGMN using multiple hypothesis . . . . .	107
Figure 5.19:	Estimating both $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ on the circular data set . . . . .	108
Figure 5.20:	Returning multiple solutions for the circular data set . . . . .	108
Figure 5.21:	Identification models . . . . .	109

Figure 5.22:	Identification of the nonlinear plant using some previous approaches . . . . .	110
Figure 5.23:	Identification of a nonlinear plant using 1000 training samples . . . . .	111
Figure 5.24:	Assessing the sensibility of IGMN to the $\varepsilon_{max}$ parameter . . . . .	111
Figure 5.25:	Assessing the sensibility of IGMN to the $\delta$ parameter . . . . .	112
Figure 5.26:	Predicting a time series using MLP . . . . .	114
Figure 5.27:	Predicting the time series using Camargo's model . . . . .	115
Figure 5.28:	Predicting the time series using IGMN . . . . .	115
Figure 5.29:	Predicting the time series using GRNN ( $\sigma = 0.075$ ) . . . . .	116
Figure 5.30:	95% confidence intervals computed by IGMN . . . . .	117
Figure 6.1:	Results obtained using the Nolfi and Tani's model in an environment composed by two rooms joined by a short corridor . . . . .	121
Figure 6.2:	Results obtained using ARAVQ in an environment composed by two rooms joined by a short corridor . . . . .	122
Figure 6.3:	Pioneer 3-DX robot with eight sonars and a differential steering drive	122
Figure 6.4:	Segmentation obtained by IGMN in six corridors experiment . . . . .	123
Figure 6.5:	Segmentation obtained by IGMN in the two rooms experiment . . . . .	124
Figure 6.6:	Difference between the speeds of the right and left motors of the robot while it is following the trajectory shown in Figure 6.5. . . . .	125
Figure 6.7:	Trajectory obtained using IGMN to choose the robot actions . . . . .	126
Figure 6.8:	Comparing the approximation errors in the trajectory of Figure 6.5 . . . . .	127
Figure 6.9:	Wall following behavior in a more complex environment. The solid gray line shows the trajectory followed by the robot in the learning mode and the dashed black line shows the trajectory followed by the robot using IGMN to control its actions. . . . .	127
Figure 6.10:	Modeled robot . . . . .	129
Figure 6.11:	Robot walking using IGMN to compute the inverse kinematics . . . . .	130
Figure 6.12:	Pendulum swing up task . . . . .	133
Figure 6.13:	Results obtained by IGMN in the pendulum swing up task . . . . .	133
Figure 6.14:	Results obtained in the robot soccer experiment . . . . .	134
Figure 6.15:	Example of robot trajectory during the task . . . . .	135
Figure 6.16:	General architecture of the proposed mapping algorithm . . . . .	138
Figure 6.17:	Environment used in the feature-based mapping experiments . . . . .	140
Figure 6.18:	Gaussian distributions generated using laser data ( $\tau_{nov} = 10^{-8}$ ) . . . . .	141
Figure 6.19:	Occupancy probabilities of Figure 6.18 model . . . . .	141
Figure 6.20:	Gaussian distributions generated using laser data ( $\tau_{nov} = 10^{-2}$ ) . . . . .	142
Figure 6.21:	Occupancy probabilities of Figure 6.20 model . . . . .	142
Figure 6.22:	Distributions generated using sonar sensors ( $\tau_{nov} = 10^{-2}$ ) . . . . .	143
Figure 6.23:	Occupancy probabilities of Figure 6.22 model . . . . .	143
Figure 6.24:	Grid map generated using the same data of Figure 6.24 . . . . .	143
Figure 6.25:	AMROffice environment . . . . .	144
Figure 6.26:	Results obtained in the AMROffice environment . . . . .	145
Figure 6.27:	Map generated in an irregular environment . . . . .	145
Figure 6.28:	Occupancy probabilities of the irregular environment . . . . .	146



## LIST OF TABLES

Table 2.1:	Characteristics of the GMM algorithms described in this chapter . . .	51
Table 3.1:	Characteristics of the described neural network models . . . . .	71
Table 5.1:	Comparison between the standard and multivariate versions of IGMN	93
Table 5.2:	Assessing the sensibility of the $\delta$ parameter . . . . .	96
Table 5.3:	Comparison between ordered shuffled data sets . . . . .	99
Table 5.4:	Comparative among ANN models using the sinusoidal data set . . . .	102
Table 5.5:	Approximating the “Mexican hat” function using IGMN . . . . .	103
Table 5.6:	Assessing the sensibility of IGMN to the $\varepsilon_{max}$ parameter . . . . .	111
Table 5.7:	Assessing the sensibility of IGMN to the $\delta$ parameter . . . . .	112
Table 6.1:	Comparative among ANNs in follow the trajectory of Figure 6.5 . . .	126
Table 6.2:	Comparative among ANNs in learning the wall-following behavior .	128



## ABSTRACT

This work proposes IGMN (standing for Incremental Gaussian Mixture Network), a new connectionist approach for incremental function approximation and real time tasks. It is inspired on recent theories about the brain, specially the Memory-Prediction Framework and the Constructivist Artificial Intelligence, which endows it with some unique features that are not present in most ANN models such as MLP, RBF and GRNN. Moreover, IGMN is based on strong statistical principles (Gaussian mixture models) and asymptotically converges to the optimal regression surface as more training data arrive. The main advantages of IGMN over other ANN models are: (i) IGMN learns incrementally using a single scan over the training data (each training pattern can be immediately used and discarded); (ii) it can produce reasonable estimates based on few training data; (iii) the learning process can proceed perpetually as new training data arrive (there is no separate phases for leaning and recalling); (iv) IGMN can handle the stability-plasticity dilemma and does not suffer from catastrophic interference; (v) the neural network topology is defined automatically and incrementally (new units added whenever is necessary); (vi) IGMN is not sensible to initialization conditions (in fact there is no random initialization/decision in IGMN); (vii) the same neural network can be used to solve both forward and inverse problems (the information flow is bidirectional) even in regions where the target data are multi-valued; and (viii) IGMN can provide the confidence levels of its estimates. Another relevant contribution of this thesis is the use of IGMN in some important state-of-the-art machine learning and robotic tasks such as model identification, incremental concept formation, reinforcement learning, robotic mapping and time series prediction. In fact, the efficiency of IGMN and its representational power expand the set of potential tasks in which the neural networks can be applied, thus opening new research directions in which important contributions can be made. Through several experiments using the proposed model it is demonstrated that IGMN is also robust to overfitting, does not require fine-tuning of its configuration parameters and has a very good computational performance, thus allowing its use in real time control applications. Therefore, IGMN is a very useful machine learning tool for incremental function approximation and on-line prediction.

**Keywords:** Machine learning, artificial neural networks, incremental learning, Bayesian methods, Gaussian mixture models, function approximation, regression, clustering, reinforcement learning, autonomous mobile robots.



## Uma Abordagem Conexcionista para a Aproximação Incremental de funções e Tarefas de Tempo Real

### RESUMO

Este trabalho propõe uma nova abordagem conexionista, chamada de IGMN (do inglês *Incremental Gaussian Mixture Network*), para aproximação incremental de funções e tarefas de tempo real. Ela é inspirada em recentes teorias do cérebro, especialmente o MPF (do inglês *Memory-Prediction Framework*) e a Inteligência Artificial Construtivista, que fazem com que o modelo proposto possua características especiais que não estão presentes na maioria dos modelos de redes neurais existentes. Além disso, IGMN é baseado em sólidos princípios estatísticos (modelos de mistura gaussianos) e assintoticamente converge para a superfície de regressão ótima a medida que os dados de treinamento chegam. As principais vantagens do IGMN em relação a outros modelos de redes neurais são: (i) IGMN aprende instantaneamente analisando cada padrão de treinamento apenas uma vez (cada dado pode ser imediatamente utilizado e descartado); (ii) o modelo proposto produz estimativas razoáveis baseado em poucos dados de treinamento; (iii) IGMN aprende de forma contínua e perpétua a medida que novos dados de treinamento chegam (não existem fases separadas de treinamento e utilização); (iv) o modelo proposto resolve o dilema da estabilidade-plasticidade e não sofre de interferência catastrófica; (v) a topologia da rede neural é definida automaticamente e de forma incremental (novas unidades são adicionadas sempre que necessário); (vi) IGMN não é sensível às condições de inicialização (de fato IGMN não utiliza nenhuma decisão e/ou inicialização aleatória); (vii) a mesma rede neural IGMN pode ser utilizada em problemas diretos e inversos (o fluxo de informações é bidirecional) mesmo em regiões onde a função alvo tem múltiplas soluções; e (viii) IGMN fornece o nível de confiança de suas estimativas. Outra contribuição relevante desta tese é o uso do IGMN em importantes tarefas nas áreas de robótica e aprendizado de máquina, como por exemplo a identificação de modelos, a formação incremental de conceitos, o aprendizado por reforço, o mapeamento robótico e previsão de séries temporais. De fato, o poder de representação e a eficiência e do modelo proposto permitem expandir o conjunto de tarefas nas quais as redes neurais podem ser utilizadas, abrindo assim novas direções nos quais importantes contribuições do estado da arte podem ser feitas. Através de diversos experimentos, realizados utilizando o modelo proposto, é demonstrado que o IGMN é bastante robusto ao problema de *overfitting*, não requer um ajuste fino dos parâmetros de configuração e possui uma boa performance computacional que permite o seu uso em aplicações de controle em tempo real. Portanto pode-se afirmar que o IGMN é uma ferramenta de aprendizado de máquina bastante útil em tarefas de aprendizado incremental de funções e predição em tempo real.

**Palavras-chave:** Aprendizado de Máquina, Redes Neurais Artificiais, Aprendizado Incremental, Métodos Bayesianos, Modelos de Mistura Gaussianos, Aproximação de Funções, Regressão, Formação de Agrupamentos, Aprendizado por Reforço, Robôs Móveis Autônomos.



# 1 INTRODUCTION

Artificial neural networks (ANNs) (HAYKIN, 2008; FREEMAN; SKAPURA, 1991) are mathematical or computational models inspired by the structure and functional aspects of biological neural networks. They are composed by several layers of massively interconnected processing units, called artificial neurons, which can change their connection strength (i.e., the synaptic weights values) based on external or internal information that flows through the network during learning. Modern ANNs are non-linear machine learning (MITCHELL, 1997) tools frequently used to model complex relationships between inputs and outputs and/or to find patterns in data. In the past several neural network models have been proposed. The most well-known model is the Multi-Layer Perceptron (MLP) (RUMELHART; HINTON; WILLIAMS, 1986), which can be used in function approximation and classification tasks. The MLP supervised learning algorithm, called Backpropagation, uses gradient descent to minimize the mean square error between the desired and actual ANN outputs. Other ANN models, like the Self-Organizing Maps (SOM) (KOHONEN, 1990, 2001), are trained using unsupervised learning to find relationships among the input patterns themselves and/or to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples.

Although in the last decades neural networks have been successfully used in several tasks, including signal processing, pattern recognition and robotics, most ANN models have some disadvantages that difficult their use in incremental function approximation and prediction tasks. The Backpropagation learning algorithm, for instance, requires several scans over all training data, which must be complete and available at the beginning of the learning process, to converge for a good solution. Moreover, after the end of the training process the synaptic weights are “frozen”, i.e., the network loses its learning capabilities. These drawbacks, which also occurs in other ANN models like SOM, highly contrast with the human brain learning capabilities because: (i) we don't need to perform thousands of scans over the training data to learn something (in general we are able to learn using few examples and/or repetitions, the so called *aggressive learning*); (ii) we are always learning new concepts as new “training data” arrive, i.e., we are always improving our performance through experience; and (iii) we don't have to wait until sufficient information arrives to make a decision, i.e., we can use partial information as it becomes available. Besides being not biologically plausible, these drawbacks difficult the use of ANNs in tasks like incremental concept formation, reinforcement learning and robotics, because in this kind of application the training data are just instantaneously available to the learning system, and in general a decision must be made using the information available at the moment.

There are other problems which difficult the use of ANNs, like the definition of the network topology (e.g., number of layers and neurons per layer) and the setting of many

configuration parameters (e.g., learning rate, momentum and weight decay). In fact, the main difficulty of using neural networks in practical problems is to adjust these settings adequately, because they are critical and dependent on the training data and/or current task (HAYKIN, 2008). To tackle these problems some ANN models have been proposed, like the Fahlman's cascade correlation (FAHLMAN; LEBIERE, 1990), which is restricted to supervised classification tasks, and GTSOM (BASTOS, 2007; MENEGAZ; ENGEL, 2009), a temporal and incremental SOM without separate phases for learning and recalling (but which has many configuration parameters and requires several training epochs to converge). Therefore, although on the last decades many ANN models have been proposed, most of these models are unsuitable for incremental function approximation and prediction, i.e., they cannot learn aggressively (i.e., using few training samples) and incrementally from continuous and noisy input data.

This thesis proposes a new artificial neural network model, called IGMN (standing for Incremental Gaussian Mixture Network), which is able to tackle great part of these problems described above. IGMN is inspired on recent theories about the brain, specially the Memory-Prediction Framework (MPF) (HAWKINS, 2005) and the constructivist artificial intelligence (DRESCHER, 1991), which endows it with some unique features that are not present in other neural network models such as MLP and RBF. The main advantages of IGMN over other connectionist approaches are:

- It learns instantaneously through just one scan over the training data;
- It does not require that the complete training data set be available at the beginning of the learning process (each training pattern can be immediately used and discarded);
- The IGMN learning algorithm is very aggressive, i.e., it is able to create good approximations using few training data;
- IGMN operates continuously without separate phases for learning and recalling (the neural network can always improve its performance as new data arrive);
- It handles the stability-plasticity dilemma and does not suffer from catastrophic interference;
- The network topology is defined automatically and incrementally (new units added whenever is necessary);
- It is based on a probabilistic framework (Gaussian mixture models) and approximates the optimal Bayesian decision using the available training data;
- It uses a multivariate representation with full variance/covariance matrices;
- IGMN can predict both the forward and the inverse mappings even in regions where the target data are multi-valued;
- It is not sensible to initialization conditions (in fact there is no random initialization in IGMN);
- The representations created in the cortical regions correspond to natural groupings (i.e. clusters) of the state space that can be interpreted by a human specialist (i.e., IGMN is not a *black box*);
- It provides not only an estimate of the target stimulus but can also inform the confidence level of its estimates;
- IGMN has few non critical configuration parameters that are easy to configure;
- IGMN can be used in supervised, unsupervised or reinforcement learning tasks.

IGMN is also particularly useful in on-line robotic tasks, because it can handle large input data received at high frequencies as the robot explores the environment. Hence, IGMN fulfills the requirements of the so called Embodied Statistical Learning, a desired



but still scarce set of statistical methods compatible to the design principles of Embodied Artificial Intelligence (BURFOOT; LUNGARELLA; KUNIYOSHI, 2008). To the best of our knowledge, IGMN is the first ANN model based on incremental Gaussian Mixture Models (GMMs) that can solve the forward and inverse problems even in regions of the state space where the target function is multi-valued and to inform the confidence of its estimates. The remaining of this chapter is organized as follows. Section 1.1 describes some theoretical concepts about function approximation, regression, classification and clustering. Section 1.2 describes the theoretical and biological inspiration of IGMN. Section 1.3 presents the main objectives and contributions of this work. Finally, Section 1.4 presents the outline of this thesis.

## 1.1 Function approximation

Function approximation consists in finding a mapping  $\mathbb{R}^D \rightarrow \mathbb{R}^O$  given a set of training data vectors, where  $D$  and  $O$  are the dimensionality of the input and output vectors, respectively. It is assumed that the function to be approximated is smooth in some sense, because the problem of function approximation is ill-posed and therefore must be constrained (HAYKIN, 2008). According to Poggio and Girosi (1989) the problem of learning a mapping between an input and an output space is essentially equivalent to the problem of synthesizing an associative memory that retrieves the appropriate output when presented with the input and *generalizes* when presented with new inputs. It also consists in identifying the system that transforms inputs into outputs given a set of examples of input-output pairs (BARRON; BARRON, 1988; OMOHUNDRO, 1987).

A classical framework for this problem is the approximation theory, which deals with the problem of approximating or interpolating a continuous, multivariate function  $f(X)$  by an approximating function  $F(W, X)$  having a fixed number of parameters  $W$ , where  $X$  and  $W$  are real vectors  $X = x_1, x_2, \dots, x_D$  and  $W = w_1, w_2, \dots, w_M$ . For a choice of a specific  $F$ , the problem is then to find the set of parameters  $W$  that provides the best possible approximation of  $f$  on the set of training examples which constitutes the *learning* step. Therefore, it is very important to choose an approximating function  $F$  that can represent  $f$  as well as possible. To measure the quality of the approximation, a distance function  $\rho$  is used to determine the distance  $\rho[f(X), F(W, X)]$  of an approximation  $F(W, X)$  from  $f(X)$ . The approximation problem can then be stated formally as (POGGIO; GIROSI, 1989):

### Approximation problem

*If  $f(X)$  is a continuous function defined on set  $\mathcal{X}$ , and  $F(W, X)$  is an approximating function that depends continuously on  $W \in \mathcal{W}$  and  $X$ , the approximation problem is to determine the parameters  $W^*$  such that*

$$\rho[F(W^*, X), f(X)] < \rho[F(W, X), f(X)]$$

*for all  $W$  in the set  $\mathcal{W}$ .*

There are two main kinds of function approximation according to the characteristics of the output space  $\mathcal{Y}$ . When the target function is continuous, the problem is called regression. The goal of regression is to learn a mapping from the input space,  $\mathcal{X}$ , to the output space,  $\mathcal{Y}$ . This mapping,  $F$ , is called an *estimator*. In general the function to be approximated is not known a priori, i.e., it must be approximated using just the input-output pairs available for training. If the target function is discrete, the problem is called

classification. It consists in predicting categorical class labels, which can be discrete or nominal. The goal of classification is to learn a mapping from the feature space,  $\mathcal{X}$ , to a label space,  $\mathcal{Y}$ . This mapping,  $F$ , is called a *classifier*.

Function approximation is considered a supervised learning task, because the training process in general occurs using a dataset containing the desired outputs for each input data vector. In classification tasks, for instance, the output classes are previously known and fixed. Therefore, the classifier does not have to create new categories, i.e., it just needs to assign each training vector to one of the known classes. But in some tasks the desired outputs are not available to guide the learning process, i.e., the system needs to create the category labels and to assign the data vectors to them using just the information available in the input data. This task is called unsupervised classification or clustering, because the learning system goal is to partition the input space in clusters according to similarities discovered in the data. Unsupervised classification is a very difficult problem because multidimensional data can form clusters with different shapes and sizes, demanding a flexible and powerful modeling tool.

From a theoretical point of view, supervised and unsupervised learning differ only in the causal structure of the model. In supervised learning, the model defines the effect one set of observations, called inputs, has on another set of observations, called outputs. In other words, the inputs are assumed to be at the beginning and outputs at the end of the causal chain. In unsupervised learning, all the observations are assumed to be caused by latent variables, that is, the observations are assumed to be at the end of the causal chain (VALPOLA, 2000).

There are many machine learning tools available for function approximation and clustering (e.g. regression trees (QUINLAN, 1993) and the EM algorithm (DEMPSTER; LAIRD; RUBIN, 1977)), but in this work we are interested just in connectionist approaches, i.e., those based on artificial neural networks.

## 1.2 Theoretical and biological inspiration

Traditional neural network models, such as the Multi-layer Perceptron (MLP) and the Radial Basis Functions (RBF) network, are based on Cybernetics, a science devoted to understand the phenomena and natural processes through the study of communication and control in living organisms, machines and social processes (ASHBY, 1956). Cybernetics had its origins and evolution in the second-half of the 20th century, specially after the development of the McCulloch-Pitts neural model (MCCULLOCH; PITTS, 1943). According to Cybernetics, the brain can be seen as an information system that receives information as input, performs some processing over this information and outcomes the computed results as output. Figure 1.1, reproduced from Pfeifer and Scheier (1994), illustrates this *information system metaphor*. Therefore, in traditional connectionist models the information flow is unidirectional, from the input (e.g., sensory stimulation) to the hidden layer (processing) and then to the output (e.g., motor action) layer.

However, in the last decades this information processing point of view has been considered obsolete in face of the new scientific discoveries in the neurosciences, and consequently new theories for explaining how the brain works have been proposed (CRICK, 1994; DAMASIO, 1994; PINKER, 1997; DENNETT, 1996; PFEIFER; SCHEIER, 1999; RAMACHANDRAN, 2003; HAWKINS, 2005). One of these theories is the *embodied intelligence* (PFEIFER; SCHEIER, 1999; PFEIFER; IIDA; BONGARD, 2005), which has been used in the design of autonomous agents in simulated and real environments

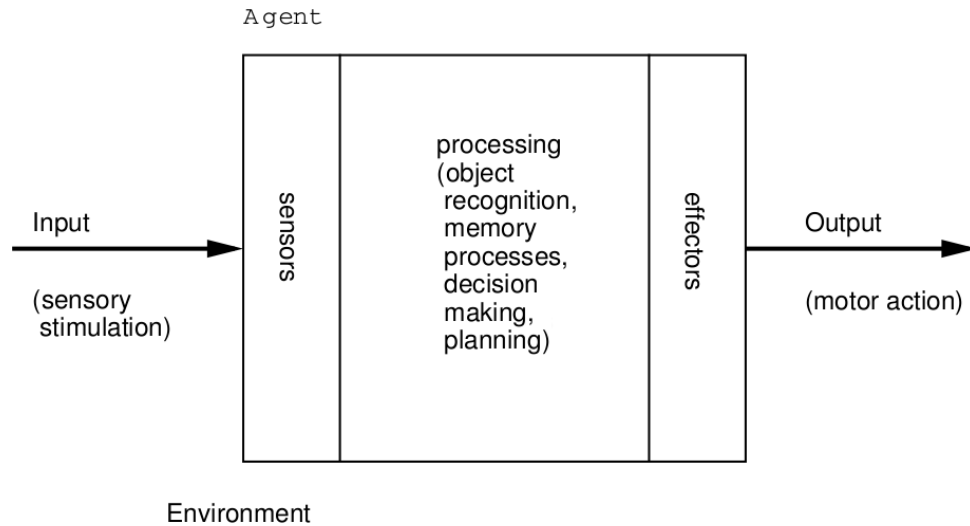


Figure 1.1: Information flow in traditional connectionist models that follows the *information system metaphor*

(PFEIFER; SCHEIER, 1994, 1997; PFEIFER; LUNGARELLA; IIDA, 2007; NOLFI; PARISI, 1999; NOLFI; FLOREANO, 2000; KRICHMAR; EDELMAN, 2003, 2005). According to this theory, instead of a unidirectional flow of information (open loop), there is a continuous tight interaction (closed loop) between the motor system and the various sensory systems, i.e. a sensory-motor coordination. Therefore, the sensory perceptions of an agent influence its actions, and these actions also change the agent's perception, as shows Figure 1.2 (reproduced from Pfeifer et al. (2007)). It can be seen in this figure that it is the environment that closes the loop between perception and action in embodied systems, and therefore the agent must be complete, i.e.: embodied, autonomous, self-sufficient and situated (PFEIFER; SCHEIER, 1999).

Another interesting theory about how the brain works, called Memory-Prediction Framework (MPF), is presented in Hawkins (2005). This theory, which is inspired by the work of other researchers such as Stephen Grossberg (GROSSBERG, 1976a,b, 2000), Vernon B. Mountcastle (MOUNTCASTLE, 1978) and Gerald M. Edelman (EDELMAN, 1978, 1987; KRICHMAR; EDELMAN, 2002), attempts to provide an overall theoretical understanding of the neocortex – the part of the human brain responsible for “intelligence” (HAWKINS, 2005). According to MPF, the brain is a *probabilistic machine* whose function is to make continuous predictions about future events. Moreover, it is organized in a hierarchy of levels (i.e., cortical regions), as shown in Figure 1.3 reproduced from Hawkins (2005). The lower levels in the hierarchy (near to the sensory areas) learn more basic (concrete) concepts, and the higher levels (composed by the union of many lower-level concepts) learn more general (abstract) and invariant concepts (HAWKINS, 2005; PINTO, 2009).

An important aspect of MPF is that there are two kinds of connections with distinct roles in the neocortex: the bottom-up (feedforward) connections, which provide predictions from the lower to the higher levels, and the top-down (feedback) connections, which provide expectations to the lower levels of the hierarchy. Although feedback dominates most connections throughout the neocortex, in most neural network models (e.g., MLP and RBF) it is simply ignored. But according to the MPF theory these feedback connections are very important, because they provide top-down expectations to the lower levels of the neocortex hierarchy. These expectations interact with the bottom-up signals to both

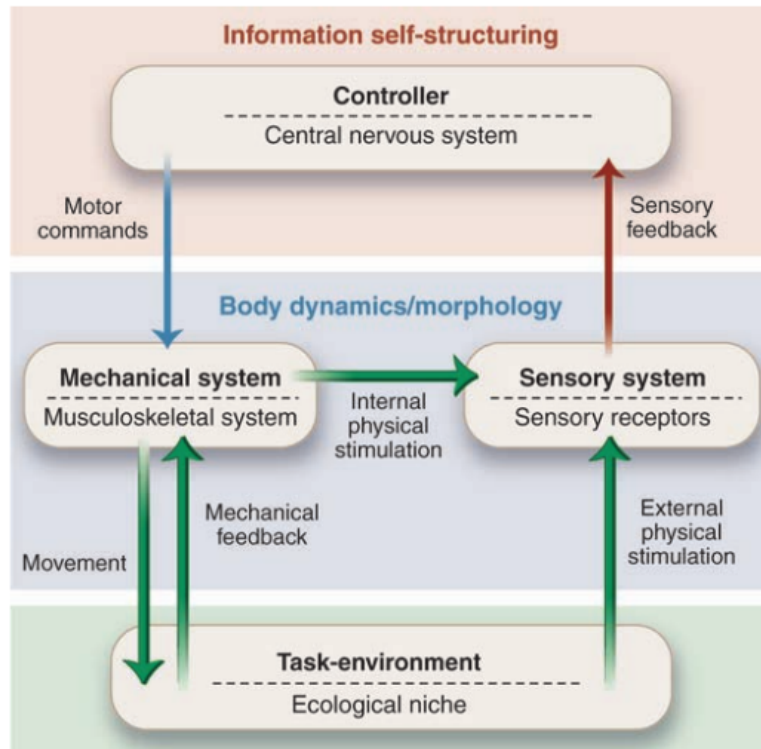


Figure 1.2: Information flow in embodied systems: there is a closed loop of information between perception and action

analyze those inputs and generate predictions of subsequent expected inputs. Higher levels predict future inputs by matching partial sequences and projecting their expectations to the lower levels, thus helping these lower level to make better predictions (HAWKINS, 2005).

Another particularity of MPF is the kind of relationship that occurs among different sensory stimuli in the neocortex. As Figure 1.3 shows, in the neocortex each sense is processed by a cortical region, and these regions are connected to the association areas (the higher levels in the hierarchy) through bottom-up and top-down connections. Consequently, a stimulus in a sensory modality (e.g., hearing) can help to comprehend another stimulus in other areas (e.g., vision) and/or to make more accurate predictions about future events. If we hear a bark, for instance, we will expect to see a dog in the vicinity (PINTO, 2009). The relationship between sensory and motor processing is also an important aspect of MPF. According to MPF, the motor areas of cortex consist of a behavioral hierarchy similar to the sensory hierarchy, with the lowest levels consisting of explicit motor commands to musculature, and the highest levels corresponding to abstract prescriptions (e.g. “catch the ball”). Therefore, sensory and motor hierarchies are tightly coupled, with behavior giving rise to sensory expectations and sensory perceptions driving motor processes. Moreover, according to Hawkins (2005), a common function, i.e., a common algorithm, is performed by all the cortical regions in the neocortex. Thus, vision is not considered different from hearing, which is not considered different from motor output. What makes the vision regions different of hearing regions is the kind of stimuli received, not the brain structure itself (MOUNTCASTLE, 1978).

Another theory of intelligence that has been developed in the last few decades is the Constructivist Artificial Intelligence (DRESCHER, 1991), which comprises all works on Artificial Intelligence (AI) that refer to the constructivist psychological theory (PIAGET,

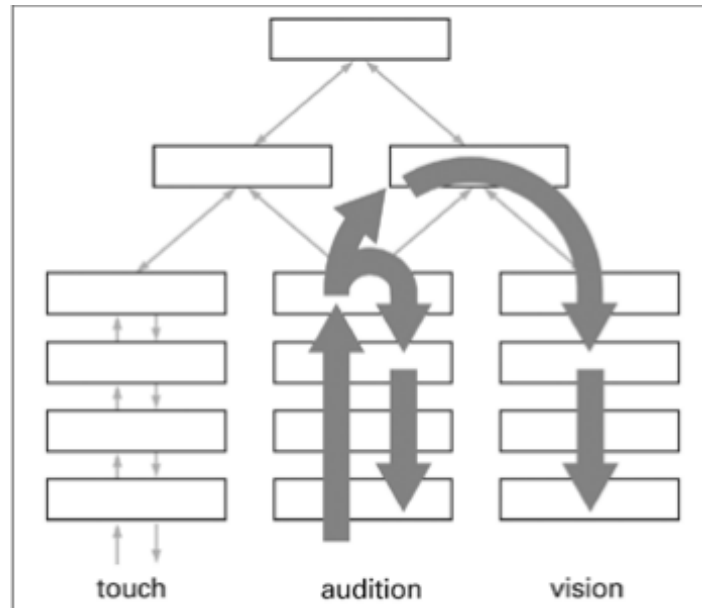


Figure 1.3: Information flows up and down sensory hierarchies to form predictions and create a unified sensory experience

1954). The constructivist conception of intelligence was brought to the field of AI by the Drescher's pioneer work (DRESCHER, 1991), and some improvements of this theory are presented in Chaput (2004), Perotto and Álvares (2006; 2007) and Perotto (2010). The key concepts of Piaget's constructivist theory that are applicable for the learning processes of both humans and artificial agents are (PIAGET, 1952, 1954):

- **Assimilation:** occurs when the agent perceives new objects or events in terms of existing schemas or operations. According to Piaget (1954), people tend to apply any mental structure that is available to assimilate a new event, and actively seek to use this newly acquired mental structure;
- **Accommodation:** refers to the process of changing internal mental structures to provide consistency with the external reality. It occurs when new schemas must be created to account for a new experience;
- **Equilibration:** refers to the biological drive to produce an optimal state of equilibrium between the agents's cognitive structures and their environment. It involves both assimilation and accommodation;
- **Schemata:** refers to the mental representation of an associated set of perceptions, ideas, and/or actions.

In a constructivist neural network, for instance, the *schemata* would correspond to the knowledge stored in the synaptic weights, the *equilibration* is the adjustment of the ANN parameters to reduce the prediction error, the *assimilation* process is equivalent to make small adjusts in the synaptic weights to assimilate a new event in terms of the existing schemas and the *accommodation* process corresponds to create new neurons to account for a new experience that is not well explained by the current schemata.

It is important to note that IGMN is just *inspired* in these theories, i.e., it does not implement all aspects of these models neither aims to reproduce some functionality of the human brain. Our main concern in developing IGMN was to create an efficient ANN for on-line and continuous tasks, and these theories were useful just to provide a theoretical background and a *new way* to design connectionist systems.

### 1.3 Main objectives and contributions of this thesis

This section summarizes the main contributions of this thesis and also highlights the innovations resulting from this work. The main objective of this thesis is to develop a new connectionist approach for incremental function approximation that learns incrementally from data flows and thus can be used in incremental regression and on-line prediction and robotic tasks. The first contribution of this thesis is a new neural network model, called IGMN, which has some similarities (e.g., instantaneous learning capabilities) with the Specht's Probabilistic Neural Network (PNN) (SPECHT, 1990) and General Regression Neural Network (GRNN) (SPECHT, 1991). However, IGMN is fundamentally different from these models because it is based on parametric probabilistic models (Gaussian mixture models) (MCLACHLAN; PEEL, 2000), rather than nonparametric Parzen's window estimators (PARZEN, 1962). Parametric probabilistic models have nice features from the representational point of view, describing noisy environments in a very parsimonious way, with parameters that are readily understandable.

IGMN can be seen as a supervised learning extension of the IGMM algorithm, published in Engel and Heinen (2010a; 2010b) and presented in Section 2.5, but it has unique features from the statistical point of view that endow it with the capacity of making on-line predictions for both *forward* and *inverse* problems at same time (the information flux is not unidirectional). In fact, the same IGMN neural network can be used to solve a forward problem and its corresponding inverse problem even in regions where the target data are multi-valued. Moreover, IGMN can inform the confidence of its estimates, thus allowing better decisions based on a confidence interval. To the best of our knowledge IGMN is the first neural network model endowed with these capabilities, thus making the proposed ANN model very suitable for on-line regression and prediction tasks.

The second contribution of this thesis is the use of IGMN in many machine learning and robotic tasks. In fact, the efficiency of IGMN opens new possibilities and research directions where relevant developments can be made. One of these directions is in incremental concept formation, which consists in identifying natural groupings in a sequence of noisy sensory data. Another possibility is in the reinforcement learning (RL) field, where IGMN can be used to approximate continuous state and action values ( $V(s)$  and  $Q(s, a)$  in the RL terminology). Moreover, a new action selection mechanism, based on statistical principles, is proposed allowing for an efficient exploration of the input space without requiring *ad-hoc* parameters.

This thesis also presents a new indoor feature-based mapping algorithm which uses IGMN to represent the characteristics of the environment, i.e., the features are represented using multivariate Gaussian mixture models. The main advantages of this approach are: (i) GMMs can represent the environment using variable size and shape structures, which is more precise and parsimonious than the usual grid-like maps; (ii) it is inherently probabilistic, which according to Thrun et al. (2006) is required for efficient robotic algorithms; and (iii) it is computationally efficient even in large environments.

Other applications in which IGMN can be successfully used are time series prediction, robotic control and solving the inverse kinematics problem. Summing up, the main contributions of this thesis are:

- A new ANN model, called IGMN, that learns incrementally from data flows using a single scan over the training data, produces valid answers even in regions where the target data is multi-valued and can inform the confidence of its estimates;
- The use of IGMN in many important machine learning and robotic tasks outper-

forming some of the existing connectionist approaches;

- A new reinforcement learning algorithm for continuous spaces that allows an efficient exploration mechanism which does not require *ad-hoc* choices;
- A new feature-based mapping algorithm, based on IGMN, that uses Gaussian mixture models to represent the environment structures.

To the best of our knowledge all these contributions are new and relevant to the state-of-the-art of their respective research areas. Part of them have been published in many important national and international events and journals (HEINEN; ENGEL, 2011, 2010a,b,c,d,e,f,g, 2009a,b,c; ENGEL; HEINEN, 2010a,b).

## 1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 presents several topics about Gaussian mixture models, which are used in the core of the neural network model proposed in this thesis. Chapter 3 describes some related work about artificial neural networks and presents many well-known state-of-the-art ANN models, such as MLP, RBF, ART and GRNN, their characteristics and limitations in the kind of task that we are interested, i.e., incremental function approximation and on-line prediction.

Chapter 4 presents the new neural network model proposed in this thesis, called IGMN, which is the main contribution of this work. Throughout that chapter the IGMN neural architecture is presented, as well as its mathematical derivation, the incremental learning algorithm and some other important features that are not available in other ANN models such as MLP, RBF and GRNN.

Chapter 5 describes several experiments to evaluate the performance of IGMN in function approximation and prediction tasks and also exploring some basic aspects of the proposed model such as: the advantage of a multivariate representation; if the order of presentation of data affects the results; the sensibility of IGMN to its configuration parameters; how it can be used to solve forward and inverse problems at same time; and to compute the confidence intervals of its estimates. That chapter also presents some experiments in which IGMN is used to identify a nonlinear plant and to predict the future values of a cyclic time series.

Chapter 6 presents the second contribution of this thesis, which is the use of IGMN in many potential applications of neural networks such as incremental concept formation, on-line robotic control, to solve the inverse kinematics problem, reinforcement learning and feature-based mapping. Moreover, that chapter demonstrates that IGMN is a very useful machine learning tool that allows to expand the horizon of tasks in which the artificial neural networks can be successfully used.

Finally, Chapter 7 concludes this monograph summarizing the main concepts and contributions of this thesis and suggesting fruitful directions for further work.





## 2 GAUSSIAN MIXTURE MODELS

This chapter presents several topics about Gaussian mixture models (GMM), which are used in the core of the neural network model proposed in this thesis. Initially Section 2.1 describes Gaussian mixture models and their main characteristics. Sections 2.2 and 2.3 present, respectively, the  $K$ -means clustering algorithm and the Expectation-Maximization (EM) algorithm, which are commonly used to learn the set of parameters of a GMM. Section 2.4 presents some state-of-the-art algorithms for learning Gaussian mixture models. Finally, Section 2.5 describes the IGMM algorithm, which was developed by us (ENGEL; HEINEN, 2010a,b) to learn GMMs in an incremental and continuous way.

### 2.1 Gaussian mixture models

A Gaussian mixture model (GMM) (MCLACHLAN; BASFORD, 1988; MCLACHLAN; PEEL, 2000) is a statistical modeling tool that has been successfully used in a number of important problems involving both pattern recognition tasks of supervised classification and unsupervised classification (JAIN; DUIN; MAO, 2000). In supervised classification, an observed pattern, viewed as a  $D$ -dimensional feature vector, is probabilistic assigned to a set of predefined classes. In this case, the main task is to discriminate the incoming patterns based on the predefined class model. In unsupervised classification, classes are not predefined but are learned based on the similarity of patterns. In this case, the recognition problem is posed as a categorization task, or clustering, consisting in finding natural groupings (i.e., clusters) in multidimensional data, based on measured similarities among the patterns (TAN; STEINBACH; KUMAR, 2006). Unsupervised classification is a very difficult problem because multidimensional data can form clusters with different shapes and sizes, demanding a flexible and powerful modeling tool (FUKUNAGA, 1990).

Suppose that we have observed a set of  $N$  samples  $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^n, \dots, \mathbf{x}^N\}$  up to the current time  $t$ . The problem of modeling samples by a probability density function can be posed as a problem of Gaussian mixture estimation (BISHOP, 1995). A GMM is represented as a *mixture distribution* (TITTERINGTON; SMITH; MAKOV, 1985), i.e., a linear combination of  $M$  Gaussian component densities as given by the equation

$$p(\mathbf{x}) = \sum_{j=1}^M p(j) p(\mathbf{x}|j), \quad (2.1)$$

where  $\mathbf{x}$  is a  $D$ -dimensional continuous-valued data vector,  $p(\mathbf{x}|j)$  is the component density function and  $p(j)$  is the prior probability that the data point  $\mathbf{x}$  has been generated from the  $j$ th mixture component (BISHOP, 1995). The priors  $p(j), \forall j \in M$ , are chosen

to satisfy the constraints:

$$\begin{aligned} \sum_{j=1}^M p(j) &= 1 \\ 0 \leq p(j) &\leq 1. \end{aligned} \quad (2.2)$$

Each component density  $p(\mathbf{x}|j)$  is a  $D$ -variate Gaussian function of the form:

$$p(\mathbf{x}|j) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{C}_j) = \frac{1}{(2\pi)^{D/2} \sqrt{|\mathbf{C}_j|}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_j)^T \mathbf{C}_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j) \right\} \quad (2.3)$$

with mean vector  $\boldsymbol{\mu}_j$  and covariance matrix  $\mathbf{C}_j$ . The component density functions  $p(\mathbf{x}|j)$  are normalized so that

$$\int_{-\infty}^{+\infty} p(\mathbf{x}|j) dx = 1. \quad (2.4)$$

The *posterior* probability that  $\mathbf{x}$  has been generated from the  $j$ th Gaussian Component is expressed using Bayes' theorem in the form

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{q=1}^M p(\mathbf{x}|q)p(q)}, \quad (2.5)$$

and these posterior probabilities must satisfy the constraint:

$$\sum_{j=1}^M p(j|\mathbf{x}) = 1. \quad (2.6)$$

The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and priors from all distributions. These parameters are collectively represented by the  $\boldsymbol{\theta}$  notation:

$$\boldsymbol{\theta} = (\theta_1, \dots, \theta_j, \dots, \theta_M)^T, \quad (2.7)$$

where  $\theta_j$  represents the parameters of the  $j$ th Gaussian component, i.e.:

$$\theta_j = \{\boldsymbol{\mu}_j, \mathbf{C}_j, p(j)\}. \quad (2.8)$$

The covariance matrices,  $\mathbf{C}_j$ , can be full rank or constrained to be diagonal. Additionally, parameters can be shared, or tied, among the Gaussian components, such as having a common covariance matrix for all components. In general the GMM parameters  $\boldsymbol{\theta}$  are estimated from training data  $\mathbf{X}$  using the EM algorithm, presented in Section 2.3. Next section describes the  $K$ -means clustering algorithm, which according to Bishop (1995) “can be seen as a particular limit of the EM optimization of a Gaussian mixture model”.

## 2.2 K-means clustering algorithm

$K$ -means (MACQUEEN, 1967) is one of the simplest batch-mode, unsupervised learning algorithms to solve the so called clustering problem (TAN; STEINBACH; KUMAR, 2006). It follows a simple and easy way to classify a given dataset through a number  $K$  of clusters, where  $K \leq N$  is previously fixed, and each observation belongs to the cluster with the nearest center. Therefore,  $K$ -means aims to partition  $N$  observations into  $K$  sets  $\mathbf{S} = \{S^1, \dots, S^j, \dots, S^K\}$  minimizing the sum-of-squares criterion

$$J = \sum_{j=1}^K \sum_{n \in S_j} \|\mathbf{x}^n - \boldsymbol{\mu}_j\|^2, \quad (2.9)$$

where  $\mathbf{x}^t$  is a vector representing a data point received at time  $t$  and  $\boldsymbol{\mu}_j$  is the geometric centroid of the  $N_j$  data points belonging to  $S_j$ , i.e.:

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{n \in S_j} \mathbf{x}^n. \quad (2.10)$$

The  $K$ -means algorithm begins by assigning the points at random to  $K$  sets and then computing the mean vectors  $\boldsymbol{\mu}$  for all sets. Next, each point is re-assigned to a new set, according to what is the nearest mean vector, and then means of all sets are recomputed. This procedure is repeated until there is no further change in the groupings (BISHOP, 1995). In general the algorithm does not achieve the global minimum of  $J$  over the assignments, but it can achieve a local minimum.

The main disadvantage of  $K$ -means is that, like other distance based clustering algorithms, its induced model is equivalent to a set of equiprobable spherical distributions sharing the same variance, what badly fits to a data flow with temporal correlation. To describe this kind of data is better to use elongated elliptical distributions, like those produced by the EM algorithm, described in the next section.

### 2.3 The EM algorithm

The Expectation-Maximization (EM) algorithm (DEMPSTER; LAIRD; RUBIN, 1977; BISHOP, 1995; MCLACHLAN; KRISHNAN, 1997; FIGUEIREDO; JAIN, 2002) is the most used method to fit finite mixture models to observed data. It performs an efficient iterative procedure to compute the Maximum Likelihood (ML) estimation in the presence of missing or hidden data. The aim of ML estimation is to find the model parameters which maximize the likelihood of the GMM given the training data, i.e., to find a local maxima of  $\log p(\mathbf{X}|\boldsymbol{\theta})$ , where  $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^n, \dots, \mathbf{x}^N\}$  is a set of  $n$  independent and identically distributed samples. EM assumes the following problem definition: we have two sample spaces  $\mathcal{X}$  and  $\mathcal{Z}$ , such that there is a many-one mapping  $\mathbf{X} = f(\mathbf{Z})$  from an observation  $\mathbf{Z}$  in  $\mathcal{Z}$  to an observation  $\mathbf{X}$  in  $\mathcal{X}$ . We define  $\mathcal{Z}(\mathbf{X}) = \{\mathbf{Z} : f(\mathbf{Z}) = \mathbf{X}\}$ , where  $\mathbf{Z}$  is the complete data, and  $\mathbf{X}$  is the observed training data. If the distribution  $f(\mathbf{Z}|\boldsymbol{\theta})$  is well defined then the probability of  $\mathbf{X}$  given  $\boldsymbol{\theta}$  is

$$p(\mathbf{X}|\boldsymbol{\theta}) = \int_{\mathcal{Z}(\mathbf{X})} f(\mathbf{Z}|\boldsymbol{\theta}) d\mathbf{Z} \quad (2.11)$$

EM is an iterative optimization algorithm which attempts to solve the problem of finding the maximum-likelihood estimate  $\boldsymbol{\theta}_{\text{ML}}$  which maximizes  $\mathcal{L}(\boldsymbol{\theta}) = \log p(\mathbf{X}|\boldsymbol{\theta})$ , given that a sample from  $\mathbf{X}$  is observed, but the corresponding  $\mathbf{Z}$  is unobserved. In general,  $\log f(\mathbf{Z}|\boldsymbol{\theta})$  can be solved analytically, but maximization of  $\mathcal{L}(\boldsymbol{\theta})$  has no analytic solution (TITTERINGTON, 1984). EM defines a sequence of parameter settings through a mapping  $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}_{t+1}$  such that  $\mathcal{L}(\boldsymbol{\theta}_{t+1}) \geq \mathcal{L}(\boldsymbol{\theta}_t)$  with equality holding only at stationary points of  $\mathcal{L}(\boldsymbol{\theta})$ . Thus EM can be considered a hill-climbing algorithm which, at least under certain conditions, will converge to a stationary point  $\mathcal{L}(\boldsymbol{\theta})$  (COLLINS, 1997). At each iteration EM produces an estimate  $\boldsymbol{\theta}_t$  by alternatively applying two steps:

1. **Expectation (E) step:** Calculate the expected value of the log likelihood function, with respect to the conditional distribution of  $\mathbf{Z}$  given  $\mathbf{X}$  under the current estimate of the parameters  $\boldsymbol{\theta}_t$ :

$$Q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t) = E_{p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}_t)} [\log p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}')] \quad (2.12)$$

2. **Maximization (M) step:** Updates the parameter estimates according to

$$\boldsymbol{\theta}_{t+1} = \arg \max_{\boldsymbol{\theta}'} Q(\boldsymbol{\theta}', \boldsymbol{\theta}_t). \quad (2.13)$$

For Gaussian mixtures, the updated set of GMM parameters  $\boldsymbol{\theta}^*$  is computed at each iteration using the following equations (BISHOP, 1995; TAN; STEINBACH; KUMAR, 2006), where the superscript ‘\*’ indicates the new (updated) parameter values:

$$\boldsymbol{\mu}_j^* = \frac{\sum_{n=1}^N p(j|\mathbf{x}^n) \mathbf{x}^n}{\sum_{n=1}^N p(j|\mathbf{x}^n)} \quad (2.14)$$

$$\mathbf{C}_j^* = \frac{\sum_{n=1}^N p(j|\mathbf{x}^n) (\mathbf{x}^n - \boldsymbol{\mu}_j^*) (\mathbf{x}^n - \boldsymbol{\mu}_j^*)^T}{\sum_{n=1}^N p(j|\mathbf{x}^n)} \quad (2.15)$$

$$p(j)^* = \frac{1}{N} \sum_{n=1}^N p(j|\mathbf{x}^n). \quad (2.16)$$

EM is useful for several reasons: its conceptual simplicity, ease of implementation, and the fact that each iteration improves  $\mathcal{L}(\boldsymbol{\theta})$  (MCLACHLAN; KRISHNAN, 1997). The rate of convergence on the first few steps is typically quite good (XU; JORDAN, 1996), but can become slow as the solution approaches a local optimum. The main drawbacks of the EM algorithm are (FIGUEIREDO; JAIN, 2002):

- It is sensitive to initialization because the likelihood function of a mixture model is not unimodal;
- The number of Gaussian distributions must be fixed and known at the beginning of the learning process;
- It requires that the complete training set is previously known and fixed, which prevents its use in on-line applications.

In general the  $K$ -means clustering algorithm, described in the previous section, is used to initialize the EM Gaussian distributions, thus alleviating the first problem. Next section describes several improvements made over the original EM algorithm in order to tackle the limitations described above.

## 2.4 State-of-the-art about learning Gaussian mixture models

In the past several attempts have been made to create more efficient algorithms for learning Gaussian mixture models. In Stauffer and Grimson (1999), for instance, an incremental adaptive mixture model was proposed to be used in real-time tracking applications. This model is implemented as an on-line  $K$ -means approximation, where only the data which match the model (in the sense of the nearest neighbor) are included in the estimation. The prior weights  $p(k)$  of the  $K$  distributions are adjusted as follows

$$p(k)^* = (1 - \alpha) p(k) + \alpha \mathbf{m}_k, \quad (2.17)$$

where  $\alpha$  is the learning rate and  $\mathbf{m}_k$  is 1 for the nearest model which is matched and 0 for the remaining models. The mean  $\boldsymbol{\mu}$  and standard deviation  $\boldsymbol{\sigma}$  parameters for unmatched distributions remain the same. The parameters of the distribution which matches the new observation are updated as follows

$$\boldsymbol{\mu}_k^* = (1 - \rho) \boldsymbol{\mu}_k + \rho \mathbf{x}^n \quad (2.18)$$

$$\boldsymbol{\sigma}_k^{2*} = (1 - \rho)\boldsymbol{\sigma}_k^2 + \rho(\mathbf{x}^n - \boldsymbol{\mu}_k^*)^T(\mathbf{x}^n - \boldsymbol{\mu}_k^*). \quad (2.19)$$

The parameter  $\rho$  is computed through the following equation

$$\rho = \alpha \mathcal{N}(\mathbf{x}^n | \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k), \quad (2.20)$$

where  $N(\mathbf{x}^n | \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k)$  is a Gaussian probability density function with covariance matrix in the form  $\mathbf{C}_k = \boldsymbol{\sigma}_k^2 \mathbf{I}$ , i.e., it is assumed that the observed variables are mutually conditional independent. The main drawback of this approach is that, as occurs with the batch-mode  $K$ -means algorithm (MACQUEEN, 1967), it cannot represent multivariate distributions neither account for clusters with different sizes.

In Figueiredo and Jain (2002), an unsupervised algorithm for learning finite mixture models from multivariate data is proposed. This algorithm is an improvement of the standard EM algorithm because: (i) it selects the number of components in an automatic way; (ii) it is less sensible to initialization than EM; and (iii) it avoids the boundary of the parameter space. The estimation of the number of categories is based on Rissanen's Minimum Description Length (MDL) criteria (RISSANEN, 1989):

$$c_{\text{MDL}} = \arg \min_c \left\{ -\log p(\mathbf{X} | \boldsymbol{\theta}_c) + \frac{c}{2} \log N \right\}, \quad (2.21)$$

whose two-part code interpretation is the following (FIGUEIREDO; JAIN, 2002): the data code-length is  $-\log p(\mathbf{X} | \boldsymbol{\theta}_c)$ , while each of the  $c$  components  $\boldsymbol{\theta}_c$  requires a code-length proportional to  $1/2 \log N$ . The mean vectors  $\{\boldsymbol{\mu}^1, \dots, \boldsymbol{\mu}^j, \dots, \boldsymbol{\mu}^M\}$  are initialized to randomly chosen data points, and the initial covariances are made proportional to the identity matrix, i.e.,  $\mathbf{C}_{ini} = \sigma^2 \mathbf{I}$ , where  $\sigma^2$  is computed as a fraction  $\delta$  (e.g.,  $1/5$  or  $1/10$ ) of the mean of the variances along each dimension of the data, i.e.:

$$\sigma^2 = \frac{1}{\delta D} \text{trace} \left( \frac{1}{N} \sum_{n=1}^N (\mathbf{x}^n - \mathbf{m})(\mathbf{x}^n - \mathbf{m})^T \right), \quad (2.22)$$

where trace is defined as the sum of the elements on the main diagonal of a square matrix and  $\mathbf{m} \equiv 1/N \sum_{n=1}^N \mathbf{x}^n$  is the global data mean. The main disadvantage of this algorithm is that it is not incremental, i.e., it learns by processing all training data in batch-mode.

In Jepson et al. (2003) an on-line version of the EM algorithm is proposed and used in a visual tracking system. This approach considers the data observations under an exponential envelope located at the current time, thus constituting a temporal window in which the distribution parameters are assumed approximately constant. The updated set of GMM parameters  $\boldsymbol{\theta}^*$  is computed in the M-step through:

$$\begin{aligned} \boldsymbol{\mu}_j^* &= \frac{\boldsymbol{\omega}_1}{\boldsymbol{\omega}_0} \\ \boldsymbol{\sigma}_j^* &= \frac{\boldsymbol{\omega}_2}{\boldsymbol{\omega}_0} - (\boldsymbol{\mu}_j^*)^2 \\ p(j)^* &= \alpha o_j(\mathbf{x}^n) + (1 - \alpha) p(j), \end{aligned} \quad (2.23)$$

where  $\alpha$  is the learning rate,  $o_j(\mathbf{x}^n)$  are the membership probabilities for each observation  $\mathbf{x}^n$  and  $\boldsymbol{\omega}_i$  are the membership weighted,  $i$ th-order data moments approximated by

$$\boldsymbol{\omega}_i^* = \alpha \mathbf{x}^n o_s(\mathbf{x}^n) + (1 - \alpha) \boldsymbol{\omega}_i. \quad (2.24)$$

The main drawbacks of this algorithm are: (i) the distribution components are modeled by spherical Gaussian densities, i.e., just considering the mean  $\boldsymbol{\mu}_j$  and a single variance

$\sigma_j$  for each component  $j$ ; (i) the number of Gaussian components is previously known and fixed; (iii) it assumes that the distribution parameters are slowly variable; and (iv) it requires an initial guess for the optimization process and also occasional restart guesses when the estimator becomes trapped at local extrema.

In Gomes et al. (2008) an algorithm for learning nonparametric Bayesian mixture models is proposed, in which the current best estimate is computed using Variational Bayes (VB) (ATTIAS, 2000; BLEI; JORDAN, 2005). In the Variational Bayes approach, intractable posterior distributions, due to the large number of possible grouping of data, are approximated with simpler proxy distributions that are chosen so that they are tractable to compute. Given the data  $\mathbf{x}^t$  observed at time  $t$ , the VB algorithm optimizes the variational Free Energy functional:

$$\mathcal{F}(\mathbf{x}^t; q) = \int_{dW} q(\mathbf{v}, \Phi, \mathbf{z}^t) \log \frac{p(\mathbf{v}, \Phi, \mathbf{z}^t, \mathbf{x}^t | \eta, \nu, \alpha)}{q(\mathbf{v}, \Phi, \mathbf{z}^t)}, \quad (2.25)$$

which is a lower bound on the log-evidence  $\log p(\mathbf{x}^t | \eta, \nu, \alpha)$ . In Equation 2.25,  $\mathbf{z}^t$  are the assignment variables,  $\eta$  and  $\nu$  are the natural parameters for the conjugate prior,  $\alpha > 0$  is the concentration parameter of the Dirichlet Process (FERGUSON, 1973),  $\Phi$  is a set of component parameter vectors,  $\mathbf{v}$  is a set of *stick breaking* random variables (SETHURAMAN, 1994) and the proxy distributions

$$q(\mathbf{v}, \Phi, \mathbf{z}^t) = \prod_{k=1}^K q(\mathbf{v}_k; \xi_{k,1}, \xi_{k,2}) q(\phi_k; \zeta_{k,1}, \zeta_{k,2}) \prod_{t=1}^T q(z_t) \quad (2.26)$$

are products of Beta distributions for the stick breaking variables (with hyper-parameters  $\xi$ ), component distributions (with hyper-parameters  $\zeta$ ), and assignment variables, respectively. Update equations for each proxy distribution can be cycled in an iterative coordinate ascent and are guaranteed to converge to a local maximum of the free energy (GOMES; WELLING; PERONA, 2008). The main drawback of this algorithm is that, accordingly to incremental requirements described in Domingos and Hulten (2003), it is not really incremental, because it assumes that new data comes in blocks, representable by GMMs, which are then merged in the current model estimate.

In Arandjelovic and Cipolla (2005), an incremental algorithm for learning temporally-coherent GMMs which does not assume that new data comes in blocks is proposed. This algorithm keeps in memory two GMMs, the current GMM estimate and an old estimate called *Historical GMM*. The algorithm makes the strong assumption that component likelihoods do not change much with the inclusion of novel information  $\mathbf{x}^t$  in the model, i.e:

$$p^*(j | \mathbf{x}^t) \approx p(j | \mathbf{x}^t). \quad (2.27)$$

In the first stage of this algorithm, the current GMM  $\mathcal{G}$  is updated under the *constraint of fixed model complexity* (i.e. fixed number of Gaussian components) through:

$$p(j)^* = \frac{E_j + p(j | \mathbf{x}^t)}{N + 1} \quad \boldsymbol{\mu}_j^* = \frac{\boldsymbol{\mu}_j E_j + \mathbf{x}^t p(j | \mathbf{x}^t)}{E_j + p(j | \mathbf{x}^t)} \quad (2.28)$$

$$\mathbf{C}_j^* = \frac{(\mathbf{C}_j + \boldsymbol{\mu}_j \boldsymbol{\mu}_j^T - \boldsymbol{\mu}_j \boldsymbol{\mu}_j^{*T} - \boldsymbol{\mu}_j^* \boldsymbol{\mu}_j^T + \boldsymbol{\mu}_j^* \boldsymbol{\mu}_j^{*T}) E_j + (\mathbf{x}^t - \boldsymbol{\mu}_j^*)(\mathbf{x}^t - \boldsymbol{\mu}_j^*)^T p(j | \mathbf{x}^t)}{E_j + p(j | \mathbf{x}^t)}, \quad (2.29)$$

where the superscript ‘\*’ indicates the new (updated) parameters,  $p(j|\mathbf{x}^t)$  is the probability of the  $j$ th component conditioned on data point  $\mathbf{x}^t$  and  $E_j \equiv \sum_{n=1}^N p(j|\mathbf{x}^t)$ .

In the second stage of this algorithm, new Gaussian clusters are postulated based on the parameters of the current parameter model estimate  $\mathcal{G}$  and the *Historical GMM*  $\mathcal{G}^{(h)}$ . Therefore, for each pair of corresponding components  $(\boldsymbol{\mu}_j, \mathbf{C}_j)$  and  $(\boldsymbol{\mu}_j^{(h)}, \mathbf{C}_j^{(h)})$  it computes the “difference” component, and using the assumption in (2.27) the  $j$ th difference component parameters become:

$$\alpha_j^{(n)} = \frac{E_j - E_j^{(h)}}{N - N^{(h)}} \quad \boldsymbol{\mu}_j^{(n)} = \frac{\boldsymbol{\mu}_j E_j - \boldsymbol{\mu}_j^{(h)} E_j^{(h)}}{E_j - E_j^{(h)}} \quad (2.30)$$

$$\begin{aligned} \mathbf{C}_j^{(n)} = & \frac{\mathbf{C}_j E_j - (\mathbf{C}_j^{(h)} + \boldsymbol{\mu}_j^{(h)} \boldsymbol{\mu}_j^{(h)T}) E_j^{(h)} + (\boldsymbol{\mu}_j^{(h)} \boldsymbol{\mu}_j^T + \boldsymbol{\mu}_j \boldsymbol{\mu}_j^{(h)T}) E_j^{(h)} - \boldsymbol{\mu}_j \boldsymbol{\mu}_j^T E_j}{E_j - E_j^{(h)}} \\ & + \boldsymbol{\mu}_j^{(n)} \boldsymbol{\mu}_j^T + \boldsymbol{\mu}_j \boldsymbol{\mu}_j^{(n)T} - \boldsymbol{\mu}_j^{(n)} \boldsymbol{\mu}_j^{(n)T}. \end{aligned} \quad (2.31)$$

This algorithm also performs merging operations to reduce the model complexity, i.e., to minimize the expected model description length:

$$L(\boldsymbol{\theta}|\{\mathbf{x}^t\}) = \frac{1}{2} N_E \log_2(N) - \log_2 P(\{\mathbf{x}^t\}|\boldsymbol{\theta}), \quad (2.32)$$

where  $\boldsymbol{\theta}$  is the set of parameters of the GMM and  $N_E$  is the number of free parameters.

The main drawbacks of this algorithm are: (i) it must keep in memory two GMMs, (the current and *historical* GMMs); (ii) for creating new distributions, the algorithm first assumes that the complexity of the model (number of Gaussian mixture components) is fixed, and afterwards changes the number of components using splitting and merging techniques; (iii) it fails (produce unsatisfactory results) when newly available data is well explained by the Historical GMM; and (iv) it requires a strong temporal coherency among the input data (the temporal patterns cannot abruptly change).

In Kristan et al. (2008) another incremental algorithm for estimating Gaussian mixture models is proposed. This algorithm does not assume specific forms of the target probability density functions (pdf) neither temporal constraints on the observed data. It starts from a known pdf  $\hat{p}_{t-1}(x)$  from the previous time-step, and in the current time-step observes a unidimensional sample  $x_t$ . A Gaussian kernel corresponding to  $x_t$  is then calculated and used to update  $\hat{p}_{t-1}(x)$  to yield a new pdf  $\hat{p}_t(x)$ .

Let  $x_t$  be the currently observed sample and  $\hat{p}_{t-1}(x)$  be an approximation to the underlying distribution  $p(x)$  from the previous time-step. The bandwidth  $\hat{h}_t$  of the kernel  $K_{\hat{h}_t}(x - x_t)$  corresponding to  $x_t$  is obtained by approximating the unknown distribution  $p(x) \approx \hat{p}_t(x)$  through:

$$\hat{h}_t = c_{scale} [2\sqrt{\pi} R(\hat{p}_t''(x)) N]^{-1/5}, \quad (2.33)$$

where  $N$  is the number of training points received until time  $t$  and  $c_{scale} \in [1, 1.5]$  is a configuration parameter used to increase the bandwidth and thus avoid under-smoothing. The resulting kernel  $K_{\hat{h}_t}(x - x_t)$  is then combined with  $\hat{p}_{t-1}(x)$  into an improved estimate of the unknown distribution

$$\hat{p}_t(x) = \left(1 - \frac{1}{N}\right) \hat{p}_{t-1}(x) + \frac{1}{n_t} K_{\hat{h}_t}(x - x_t). \quad (2.34)$$

Next, the improved estimate  $\hat{p}_t(x)$  from (2.34) is sent back to (2.33) for re-approximating  $\hat{h}_t$ , and equations (2.33) and (2.34) are iterated until convergence. It is important to notice that, for each observed sample, the number of components in the mixture model increases. Therefore, in order to maintain low complexity, the model is compressed from time to time into a model with a smaller number of components. According to Kristan et al. (2008), "the presented approach deals with one-dimensional distributions", i.e., this algorithm was tested only in situations where  $C = \sigma^2$ .

There are other algorithms proposed to incrementally learn Gaussian mixture models (TITTERINGTON, 1984; NEAL; HINTON, 1998; WANG; ZHAO, 2006; CAPPÉ; MOULINES, 2008), but most of them require several data points to the correct estimation of the covariance matrices, assume that the input data come in blocks and/or does not handle the stability-plasticity dilemma. So, although several attempts have been made to create unsupervised algorithms to incrementally learn Gaussian mixture models, all of them have important drawbacks which prevents their use in on-line and robotic tasks. Next section presents a new GMM learning algorithm proposed by to tackle these limitations.

## 2.5 Incremental Gaussian Mixture Model

IGMM (standing for Incremental Gaussian Mixture Model) is an incremental algorithm developed by us for estimating the GMM parameters  $\theta$  from data flows (ENGEL; HEINEN, 2010a,b). IGMM was specially designed for the so called unsupervised incremental learning, which considers building a model, seen as a set of concepts of the environment describing a data flow, where each data point is just instantaneously available to the learning system (FISHER, 1987; GENNARI; LANGLEY; FISHER, 1989). In this case, the learning system needs to take into account these instantaneous data to update its model of the environment.

As the batch-mode EM algorithm, IGMM follows the mixture distribution modeling. However, its model can be effectively *expanded* with new components (i.e. concepts) as new relevant information is identified in the data flow. Moreover, IGMM adjusts the parameters of each distribution after the presentation of every single data point according to recursive equations that are approximate incremental counterparts of the batch-mode update equations used by the EM algorithm. The main advantages of IGMM over the existing approaches described above are: (i) it does not need to keep in memory a *Historical GMM* or the previous training data (each pattern can be immediately used and discarded); (ii) it does not assume strong temporal coherencies among the input data; (iii) it does not require any kind of special initialization; (iv) the obtained results are similar to those produced by the batch-mode EM algorithm; and (v) it handles the stability-plasticity dilemma properly.

IGMM tackles the stability-plasticity dilemma by means of a novelty criterion, based on the likelihood of the mixture components, and a stability criterion, that assumes a stable dominant component for the current data, i.e, the data must be locally stable. The requirement of a stable dominant component means that a new component is added to the model only if it remains dominant, in the sense of the posterior probabilities, for at least a user specified minimum sequence length of data. This is equivalent to say that a new concept is added to the model only if it explains a minimum sequence stretch of the data flow. This is a reasonable requirement for the kind of task we are interested in, since we assume that the data flow is explainable by a set of persistent concepts. It is important to say that IGMM does not assume that  $p^*(j|\mathbf{x}^t) \approx p(j|\mathbf{x}^t)$ , i.e., unlike the algorithm



presented in Arandjelovic and Cipolla (2005) IGMM can properly handle abrupt changes in the input data. The requirement of a stable dominant component only states that these changes remain for a specified period of time – otherwise the current input pattern is considered *noise*. Next subsection describes how IGMM tackles the stability-plasticity dilemma in details.

### 2.5.1 Tackling the Stability-Plasticity Dilemma

An important issue in unsupervised incremental learning is the stability-plasticity dilemma (GROSSBERG, 2000, 2003; HAYKIN, 2008), i.e., whether a new presented data point must be assimilated in the current model or cause a structural change in the model to accommodate the new information that it bears, i.e., a new concept. IGMM uses a probabilistic approach for modeling the environment, and so, it can rely on solid arguments to handle this issue.

IGMM assumes that the data stream corresponds to observations of a non-stationary environment whose dynamics is composed by a number of different local stationary dynamics that succeed one another. Each local dynamics is called a mode of the environment dynamics (CHOI; YAN-YEUNG; ZHANG, 2001) or simply a context (SILVA et al., 2006; BASSO; ENGEL, 2009). This is a valid assumption in many application domains, specially in mobile robotics. On the other hand, it also assures that the environment is local stable in a certain minimum segment of the data sequence, and so, by the succession of the contexts, the learning system eventually receives a sufficient number of observations to properly identify each context from the whole data sequence. So, we can now rely on a stability criterion that, together with a novelty criterion, can help us to overcome the problem of the model complexity selection, related to the decision whether a new component should be added to the current model.

This subsection presents the novelty criterion used by IGMM. In Subsection 2.5.2 we derive the recursive equations used to update the model parameters. Based on these equations we discuss then the stability criterion in Subsection 2.5.3.

The mixture model starts with a single component with unity prior, centered at the first input data, with a baseline covariance matrix specified by default, i.e.,  $\boldsymbol{\mu}_1 = \mathbf{x}^1$ , meaning the value of  $\mathbf{x}$  for  $t = 1$ , and  $\mathbf{C}_1 = \sigma_{ini}^2 \mathbf{I}$ , where  $\sigma_{ini}$  is a user defined fraction  $\delta$  of the overall variance (e.g.,  $\delta = 1/100$ ) of the corresponding attributes, estimated from the range of these values according to

$$\sigma_{ini} = \delta [\max(\mathbf{x}) - \min(\mathbf{x})]. \quad (2.35)$$

New components are added by demand. IGMM uses a *minimum likelihood* criterion to recognize a vector  $\mathbf{x}$  as belonging to a mixture component. For each incoming data point the algorithm verifies whether it minimally fits any mixture component. A data point  $\mathbf{x}$  is not recognized as belonging to a mixture component  $j$  if its probability  $p(\mathbf{x}|j)$  is lower than a previously specified *minimum likelihood-* (or *novelty-*) *threshold*. In this case,  $p(\mathbf{x}|j)$  is interpreted as a *likelihood function* of the  $j$ th mixture component. If  $\mathbf{x}$  is rejected by all density components, meaning that it bears new information, a new component is added to the model, appropriately adjusting its parameters. The novelty-threshold value affects the sensibility of the learning process to new concepts, with higher threshold values generating more concepts. It is more intuitive for the user to specify a minimum value for the acceptable likelihood,  $\tau_{nov}$ , as a *fraction* of the maximum value of the likelihood function, making the novelty criterion independent of the covariance matrix. Hence, a new mixture component is created when the instantaneous data point  $\mathbf{x} = (x_1, \dots, x_i, \dots, x_D)$

matches the *novelty criterion* written as

$$p(\mathbf{x}|j) < \frac{\tau_{nov}}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \quad \forall j \quad (2.36)$$

Before explaining what happens when a new component is created, next subsection presents how IGMM adjusts the values of the distribution parameters as new data points are sequentially acquired.

### 2.5.2 Model update by sequential assimilation of data points

An instantaneous data point that does not match the novelty criterion needs to be assimilated by the current mixture distribution, causing an update in the values of its parameters due to the information it bears. IGMM follows an incremental version for the usual iterative process to estimate the parameters of a mixture model (i.e., the EM algorithm) based on two steps: an estimation step (E) and a maximization step (M). The update process begins computing the posterior probabilities of component membership for the data point,  $p(j|\mathbf{x})$ , the *estimation* step. These can be obtained through Bayes' theorem, using the current component-conditional densities  $p(\mathbf{x}|j)$  and priors  $p(j)$  as follows:

$$p(j|\mathbf{x}) = \frac{p(\mathbf{x}|j)p(j)}{\sum_{q=1}^M p(\mathbf{x}|q)p(q)} \quad \forall j \quad (2.37)$$

The posterior probabilities can then be used to compute new estimates for the values of the mean vector  $\boldsymbol{\mu}_j^*$  and covariance matrix  $\mathbf{C}_j^*$  of each component density  $p(\mathbf{x}|j)$ , and the priors  $p(j)^*$  in the *maximization* step. Next, we derive the recursive equations used by IGMM to successively estimate these parameters.

The parameters  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_M)^T$ , corresponding to the means  $\boldsymbol{\mu}_j$ , covariances matrices  $\mathbf{C}_j$  and priors  $p(j)$ ,  $\forall j \in M$ , of a mixture model involving  $D$ -dimensional Gaussian distributions  $p(\mathbf{x}|j)$ , can be estimated from a sequence of  $t$  data vectors,  $\mathbf{X} = \{\mathbf{x}^1, \dots, \mathbf{x}^n, \dots, \mathbf{x}^t\}$  assumed to be drawn independently from this mixture distribution. The estimates of the parameters are random vectors whose statistical properties are obtained from their joint density function. Starting from an initial “guess”, each observation vector is used to update the estimates according to a successive estimation procedure.

IGMM follows the Robbins-Monro stochastic approximation method to derive the recursive equation used to successively estimate the priors (ROBBINS; MONRO, 1951). Notice that it is a derivation based on “first principles”, not starting from the EM update equations, as in Arandjelovic and Cipolla (2005). For this, in the maximization step the parameters of the current model are updated based on the maximization of the likelihood of the data. In this case, the *likelihood* of  $\boldsymbol{\theta}$  for the given  $\mathbf{X}$ ,  $\mathcal{L}(\boldsymbol{\theta})$ , is the joint probability density of the whole data stream  $\mathbf{X}$ , given by

$$\mathcal{L}(\boldsymbol{\theta}) \equiv p(\mathbf{X}|\boldsymbol{\theta}) = \prod_{n=1}^t p(\mathbf{x}^n|\boldsymbol{\theta}) = \prod_{n=1}^t \left[ \sum_{j=1}^M p(\mathbf{x}^n|j)p(j) \right] \quad (2.38)$$

The technique of maximum likelihood sets the value of  $\boldsymbol{\theta}$  by maximizing  $\mathcal{L}(\boldsymbol{\theta})$ . From (2.38), the maximum likelihood value of a specific parameter,  $\boldsymbol{\theta}^*$ , is given by a solution of

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \prod_{n=1}^t \left[ \sum_{j=1}^M p(\mathbf{x}^n|j)p(j) \right] \right\} \Bigg|_{\boldsymbol{\theta}} = 0 \quad (2.39)$$

If we consider the logarithm of the likelihood function, we can also write

$$\frac{1}{t} \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \sum_{n=1}^t \ln \left[ \sum_{j=1}^M p(\mathbf{x}^n | j) p(j) \right] \right\} \Big|_{\boldsymbol{\theta}} = 0 \quad (2.40)$$

Introducing the factor of  $1/t$  allows us to take the limit  $t \rightarrow \infty$  and hence obtain the expectation ( $E$ )

$$\lim_{N \rightarrow \infty} \frac{1}{t} \sum_{n=1}^t \left\{ \frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x}^n | j) p(j) \right] \right\} = E \left\{ \frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \right\} \quad (2.41)$$

Thus, the maximum likelihood solution is asymptotically equivalent to finding a solution of

$$E \left\{ \frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \right\} = 0 \quad (2.42)$$

From the Robbins-Monro approximation method, this can be solved using an iterative scheme of the form

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + a_t \frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \Big|_t \quad (2.43)$$

The coefficients  $\{a_t\}$  represent a sequence of positive numbers of type  $1/t$ , which satisfy the following three conditions:

$$\begin{cases} \lim_{t \rightarrow \infty} a_t = 0 \\ \sum_{t=1}^{\infty} a_t = \infty \\ \sum_{t=1}^{\infty} a_t^2 < \infty \end{cases} \quad (2.44)$$

It can be shown that the sequence of estimates  $\boldsymbol{\theta}_t$  converges to the root  $\boldsymbol{\theta}^*$  with probability 1 (ROBBINS; MONRO, 1951).

To find the desired equation to be used by IGMM to update the priors, we begin applying the chain rule of the derivatives to the second term of the sum in (2.43) yielding

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \Big|_t = \frac{1}{\sum_{j=1}^M p(\mathbf{x}^t | j) p(j)} \frac{\partial}{\partial \boldsymbol{\theta}} \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \Big|_t \quad (2.45)$$

where  $p(j)$  refers to the value of the prior at instant  $t$ , before updating. Using Bayes' theorem, equation (2.37), we can write

$$\frac{1}{\sum_{j=1}^M p(\mathbf{x}^t | j) p(j)} = \frac{p(j | \mathbf{x}^t)}{p(\mathbf{x}^t | j) p(j)} \quad (2.46)$$

Entering with this equation in (2.45) follows that

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \Big|_t = \frac{p(j | \mathbf{x}^t)}{p(\mathbf{x}^t | j) p(j)} \frac{\partial}{\partial \boldsymbol{\theta}} \left[ \sum_{j=1}^M p(\mathbf{x} | j) p(j) \right] \Big|_t \quad (2.47)$$

Considering now the derivative of the summation in (2.47) for the prior  $\theta = p(j)$  we obtain:

$$\frac{\partial}{\partial p(j)} \left[ \sum_{j=1}^M p(\mathbf{x}|j)p(j) \right] \Big|_t = p(\mathbf{x}^t|j) \quad (2.48)$$

Entering with (2.48) in (2.47) we can write

$$\frac{\partial}{\partial p(j)} \ln \left[ \sum_{j=1}^M p(\mathbf{x}|j)p(j) \right] = \frac{p(j|\mathbf{x}^t)}{p(\mathbf{x}^t|j)p(j)} p(\mathbf{x}^t|j) = \frac{p(j|\mathbf{x}^t)}{p(j)} \quad (2.49)$$

Now, to satisfy the conditions described by (2.44), we choose arbitrarily the coefficients (BISHOP, 1995):

$$a_t = \frac{1}{t} p(j) \quad (2.50)$$

and we end up with the following expression for updating the prior  $p(j)$ :

$$p(j) = p(j)^{old} + \frac{1}{t} p(j|\mathbf{x}^t) \quad (2.51)$$

After updating, the priors are adjusted through normalization to satisfy constraints (2.2) by

$$p(j)^* = \frac{p(j)}{\sum_{q=1}^M p(q)}. \quad (2.52)$$

Although the maximum likelihood technique for estimating the priors is straightforward, as shown above, it becomes quite complex when applied to estimate the mean vector and the covariance matrix directly from (2.3). Instead, we follow the natural conjugate technique to estimate these parameters (KEEHN, 1965). When  $\boldsymbol{\mu}$  and  $\mathbf{C}$  are estimated by the sample mean vector and sample covariance matrix, and  $\mathbf{X}$  is a normally distributed random vector, the joint density function  $p(\boldsymbol{\mu}, \mathbf{C}|\mathbf{X})$  is known to be the reproducible Gauss-Wishart distribution, the natural conjugate density for the model of (2.3) (KEEHN, 1965). In this case, when we estimate both the expected vector and the covariance matrix of a single distribution, starting with a priori distribution with an expected vector  $\boldsymbol{\mu}^0$  and covariance matrix  $\mathbf{C}^0$ , these parameters are transformed through  $n$  observations in the following manner (KEEHN, 1965; FUKUNAGA, 1990):

$$\omega^1 = \omega^0 + n \quad v^1 = v^0 + n$$

$$\boldsymbol{\mu}^1 = \frac{\omega^0 \boldsymbol{\mu}^0 + n \langle \mathbf{X} \rangle}{\omega^0 + n} \quad (2.53)$$

$$\mathbf{C}^1 = \frac{\left( v^0 \mathbf{C}^0 + \omega^0 \boldsymbol{\mu}^0 (\boldsymbol{\mu}^0)^T \right) + n \langle \mathbf{X} \rangle \langle \mathbf{X} \rangle^T - \omega^1 \boldsymbol{\mu}^1 (\boldsymbol{\mu}^1)^T}{v^0 + n} \quad (2.54)$$

where  $\omega^0$  and  $v^0$  reflect the confidence about the initial estimates of  $\boldsymbol{\mu}^0$  and  $\mathbf{C}^0$  respectively, corresponding to the number of samples used to compute these initial estimates.

On the other hand, when the probability density of the input data is a Gaussian mixture model with  $M$  components, an observation  $\mathbf{x}^t$  is probabilistic assigned to a distribution  $j$  by the corresponding posterior probability  $p(j|\mathbf{x}^t)$ . In this case, the equivalent number of samples used to compute the parameter estimates of the  $j$ th distribution component corresponds to the sum of posterior probabilities that the data presented so far were generated

from component  $j$ , the so called *0th-order moment of  $p(j|\mathbf{x})$  over the data*, or simply the *0th-order data moment for  $j$* . IGMM stores this summation as the variable  $sp_j$ , i.e.:

$$sp_j = \sum_{n=1}^t p(j|\mathbf{x}^n). \quad (2.55)$$

The instantaneously presented data point  $\mathbf{x}^t$  contributes to this sum with its posterior probability to the component  $j$ , written as

$$\sum_{n=1}^t p(j|\mathbf{x}^n) = p(j|\mathbf{x}^t) + \sum_{n=1}^{t-1} p(j|\mathbf{x}^n). \quad (2.56)$$

The left term and the last right term correspond respectively to the *new* and *old* values of the variable  $sp_j$ ,

$$sp_j^* = p(j|\mathbf{x}^t) + sp_j \quad (2.57)$$

Since IGMM updates the distribution parameters of the  $M$  components after a single observation  $\mathbf{x}^t$ , the equivalent number of observations for a specific component  $j$ , corresponding to  $n$  in (2.53) and (2.54), is given by  $p(j|\mathbf{x}^t)$ . In this case, we can rewrite (2.53) and (2.54) considering:

$$\langle \mathbf{X} \rangle = \mathbf{x}^t \quad (2.58)$$

and

$$n_j = p(j|\mathbf{x}^t) \quad (2.59)$$

Moreover, from these definitions we recognize that, for a Gaussian mixture model, the confidence constants for the  $j$ th component appearing in (2.53) and (2.54) correspond to:

$$\omega_j^0 = v_j^0 = \sum_{n=1}^{t-1} p(j|\mathbf{x}^n) = sp_j \quad (2.60)$$

and

$$\omega_j^1 = v_j^1 = sp_j^* \quad (2.61)$$

Entering with (2.58), (2.59), (2.60) and (2.61) in (2.53) and (2.54) and rearranging the terms we obtain the desired recursive equations for updating the mean vector and the covariance matrix of the  $j$ th mixture component respectively as

$$\boldsymbol{\mu}_j^* = \boldsymbol{\mu}_j + \frac{p(j|\mathbf{x}^t)}{sp_j^*} (\mathbf{x}^t - \boldsymbol{\mu}_j) \quad (2.62)$$

$$\mathbf{C}_j^* = \mathbf{C}_j - (\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j)^T + \frac{p(j|\mathbf{x})}{sp_j^*} [(\mathbf{x} - \boldsymbol{\mu}_j^*)(\mathbf{x} - \boldsymbol{\mu}_j^*)^T - \mathbf{C}_j] \quad (2.63)$$

Note that the total number of presented data points, equivalent to  $t$ , can be computed summing all  $sp_j$ , and so no extra counter is needed to store  $t$ , as can be shown by

$$t = \sum_{j=1}^M sp_j \quad (2.64)$$

So, making use of  $sp_j$ , we can observe that for  $t \gg 1$ , Equation (2.51) can be replaced by

$$p(j)^* = \frac{sp_j^*}{\sum_{q=1}^M sp_q^*} \quad (2.65)$$

These equations formalize the maximization step of IGMM.

One important property of the recursive equations (2.62), (2.63) and (2.65) is the fact that, for non-stationary but locally stable environments, they continuously compute a useful instantaneous approximation of the parameters that represent the mixture distribution. Next, we show that these recursive expressions are approximate incremental counterparts of the equations used by the EM algorithm (DEMPSTER; LAIRD; RUBIN, 1977). To see this, we begin iterating equation (2.51) from  $n = 1$  until  $t$ , obtaining

$$p(j)^* = \frac{1}{t} \sum_{n=1}^t p(j|\mathbf{x}^n) \quad (2.66)$$

This expression is the same equation used by the EM algorithm to update the priors in the step M. As a matter of fact,  $sp_j$  represents an approximate estimate of the desired summation, since by each new presented data point the distribution parameters change. This actualization error is analog to the one made by the EM algorithm when the new parameter values are computed using posteriors computed by old parameters. In the case of the EM algorithm, this error is minimized through successive presentations of the whole data set, while in the case of IGMM the error is minimized through the successive presentation of new data. If the learning task is *episodic*, meaning that it evolves in cycles from a starting state to a terminal state, it is very likely that all distribution components of the IGMM model will be properly updated, equivalently to the repeated presentations of the whole data set for the EM algorithm. However, as the posteriors of the past observations are not updated by IGMM, in *continuing, non-episodic* tasks this approximation may fail if the model parameters change rapidly. This makes IGMM suitable either for episodic tasks as well as for infinite horizon tasks in non-stationary but locally stable environments.

Focusing now on the update of the mean vector, we can rewrite equation (2.62) as

$$\boldsymbol{\mu}_j^* = \frac{p(j|\mathbf{x}^t)\mathbf{x}^t + \sum_{n=1}^{t-1} p(j|\mathbf{x}^n)\boldsymbol{\mu}_j}{\sum_{n=1}^t p(j|\mathbf{x}^n)} \quad (2.67)$$

Note that the subtraction of  $\boldsymbol{\mu}_j$  in the last term of (2.62) makes the summation on the numerator in (2.67) to run just until  $n = t - 1$ . So, we can think of the first term in the numerator of (2.67) as the contribution to the mean of the instantaneous data point, while the second term corresponds to the accumulation of the information about the mean of all past data points. Iterating this summation from  $n = 1$ , and making  $\boldsymbol{\mu}_j^1 = \mathbf{x}^1$ , we obtain

$$\boldsymbol{\mu}_j^* = \frac{\sum_{n=1}^t p(j|\mathbf{x}^n)\mathbf{x}^n}{\sum_{n=1}^t p(j|\mathbf{x}^n)} \quad (2.68)$$

We recognize now that this is the same equation used by the EM algorithm to update the means. Similarly, we can rewrite equation (2.63) for updating the covariance matrix as

$$\mathbf{C}_j^* = \left[ p(j|\mathbf{x}^t) (\mathbf{x}^t - \boldsymbol{\mu}_j^*) (\mathbf{x}^t - \boldsymbol{\mu}_j^*)^T + \sum_{n=1}^{t-1} p(j|\mathbf{x}^n) \mathbf{C}_j - \sum_{n=1}^{t-1} p(j|\mathbf{x}^n) (\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j) (\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j)^T \right] \left[ \sum_{n=1}^t p(j|\mathbf{x}^n) \right]^{-1} \quad (2.69)$$

Here too, we recognize the first term in the first brackets as the contribution of the present data point to the covariance matrix, while the second term corresponds to the

accumulation of the information about the covariance matrix of all past data points. The third term is generated from the new estimates of the mean. When the model is local stable, this last term is negligible. Finally, iterating backwards this expression from  $t$  until  $n = 1$ , and making  $(\mathbf{C}_j)^1 = \sigma_{ini}^2 \mathbf{I}$ , as the initial covariance matrix, and considering the third term in (2.69) as negligible, we obtain

$$\mathbf{C}_j^* = \frac{\sum_{n=1}^t p(j|\mathbf{x}^n) (\mathbf{x}^t - \boldsymbol{\mu}_j^*) (\mathbf{x}^t - \boldsymbol{\mu}_j^*)^T}{\sum_{n=1}^t p(j|\mathbf{x}^n)} \quad (2.70)$$

This is the same expression used by the EM algorithm to update the covariance matrix. As pointed out above, this derivation is also an approximation, since the distribution parameters change as new data points are considered. Concluding, we can say that the recursive equations used by IGMM described by (2.62), (2.63) and (2.65) are indeed approximate incremental counterparts of the update equations used by the EM algorithm. Similar equations can be derived from those used by the (batch-mode) EM algorithm, like in Arandjelovic and Cipolla (2005), but here we decided to derive the incremental formulae to re-estimate  $\boldsymbol{\theta}$  from basic principles. Next section describes the stability criteria used by IGMM for creating new components.

### 2.5.3 Creating a new component based on novelty and stability

To create a new component, IGMM uses the novelty criterion given by (2.36), but also a stability criterion that tests if there is already a recent created component that should assimilate the current presented data point. This means that there is a *stable* dominant component for the current data, which is equivalent to say that the environment can be decomposed in stable contexts, the *concepts* of the environment. To this end, we store the *age* of each model component  $j$ ,  $v_j$ , consisting of the number of data that have been presented to the learning system since the component was created. A new component is created only if there is no model component whose age is less than a specified threshold  $v_{min}$ , i.e., the stability criterion can be defined by

$$v_j > v_{min}, \quad \forall j.$$

If the data change too fast, these data points are assimilated by the current model as noise and no new component is created. As a matter of fact, the stability criterion avoids the successive creation of components in a noisy environment, but it cannot avoid the creation of an eventual spurious component. However, a spurious component can be readily identified if its  $sp_j$  remains very small, bellow a specified parameter  $sp_{min}$ , after some time steps after its creation, given by the parameter  $v_{min}$ . A good choice for  $sp_{min}$  is  $D + 1$ , because according to Trávén (1991), a minimum of more than  $D$  samples are required to obtain a nonsingular estimate of an unconstrained covariance matrix. Based on the same principle  $v_{min}$  can be set to any value large than  $sp_{min}$  such as  $v_{min} = 2D$ . Once identified, a spurious component can be deleted from the current model. The condition to identify and delete a spurious component can be written as

**if**  $v_j > v_{min}$  **and**  $sp_j < sp_{min}$  **then**  
     **delete** the  $j$ th component  
     **adjust**  $p(q)$  for all  $q \in M, q \neq j$ , using (2.65)  
**end if**

Whenever a data point  $\mathbf{x}^t$  matches the novelty criterion given by (2.36), and the stability criterion is fulfilled, a new component  $k$  is created, centered at that data vector and

with the baseline covariance matrix, i. e.,  $\boldsymbol{\mu}_k = \mathbf{x}^t$  and  $\mathbf{C}_k = \boldsymbol{\sigma}_{ini}^2 \mathbf{I}$ . After that, we must adjust all  $p(j)$  to satisfy constraints (2.2) by (2.65). Since  $sp_k$  is initialized with 1, the prior for this new component is equivalent to:

$$p(k) = \frac{1}{\sum_{j=1}^{M^*} sp_j}. \quad (2.71)$$

In Arandjelovic and Cipolla (2005) is affirmed that “Intuitively, if all information that is available at any time is the current GMM estimate, a single novel point never carries enough information to cause an increase in the number of Gaussian components”. We disagree with this affirmation, because in IGMM we allow for an increase in the number of components, but using the stability criterium, we delete latter on the spurious ones. In our opinion this is better than maintaining a *historical GMM* in memory or to use splitting and merging operations to change the complexity of the GMM.

#### 2.5.4 Incremental learning from unbounded data streams

IGMM was designed to process data streams creating on-line models of eventually huge amounts of data based on accumulation of information in the  $sp$  variables. This unbounded accumulation may lead to an eventual saturation of the variables that store the corresponding summations, depending on the specific limitation of the available numeric representation. To avoid this problem, a discount factor given by a parameter  $\alpha$  is applied to the instantaneous value of the corresponding variable. For the accumulator of the posteriors in the  $j$ th unit we obtain:

$$sp_j^* = p(j|\mathbf{x}^t) + \alpha sp_j. \quad (2.72)$$

Since the instantaneous sum of  $p(j|\mathbf{x}^t)$  over all components  $j$  is one, if we choose  $0 < \alpha < 1$  as

$$\alpha = 1 - \frac{1}{sp_{max}}, \quad (2.73)$$

where  $sp_{max}$  is a large number within the available numeric representation, the accumulation over the time of all  $sp_j$  according to the recursive equation (2.72) is bounded to  $sp_{max}$ , as shown by

$$\lim_{t \rightarrow \infty} \sum_{n=1}^t \sum_{j=1}^M sp_j^n = sp_{max}. \quad (2.74)$$

A large  $sp_{max}$  is desirable to keep the overall strategy stable. In some previous experiments, described in Engel (2009), it was observed that  $sp_{max} \geq 10^6$  produced negligible effects in the generated models. Moreover, we observe from these long-term equations that when the model is at a local minimum, its parameters are dependent from the *relation* among the values of the accumulators but not on the values themselves. So, to better explore the available numeric range keeping the system responsive to new components, whenever long-term mode is reached, identified as a fraction  $\beta$  of  $sp_{max}$ , the accumulators are restarted to a fraction  $\gamma$  of their corresponding value, i.e.:

$$\text{if } (\sum_{j=1}^M sp_j) \geq \beta sp_{max} \text{ then } sp_j^* = \gamma sp_j, \quad \forall j.$$

In all experiments good results were obtained using  $sp_{max} = 10^8$ ,  $\beta = 0.8$  and  $\gamma = 0.5$ . In fact, these parameters are not critical, because the obtained results were practically the same using several different values for  $sp_{max}$ ,  $\beta$  and  $\gamma$  (ENGEL, 2009).



### 2.5.5 IGMM learning algorithm

Algorithm 1 presents a detailed pseudocode description of the IGMM algorithm, which works as follows. When the first training data vector arrives, the first Gaussian mixture component is created centered on this input vector, i.e., the distribution parameters are initialized through

$$M = 1; \quad v_1 = 1; \quad sp_1 = 1.0; \quad p(1) = 1; \quad \boldsymbol{\mu}_1 = \mathbf{x}^1; \\ \mathbf{C}_1 = \boldsymbol{\sigma}_{ini}^2 \mathbf{I}; \quad p(\mathbf{x}^1|1) = \mathcal{N}(\mathbf{x}^1|\boldsymbol{\mu}_1, \mathbf{C}_1) = (\boldsymbol{\sigma}_{ini}^2 2\pi)^{-D/2},$$

where  $p(\mathbf{x}^1|1) = (\boldsymbol{\sigma}_{ini}^2 2\pi)^{-D/2}$  because as  $\boldsymbol{\mu}_1 = \mathbf{x}^1$ , the exponential term in (2.3) reduces to one, and as  $\mathbf{C}_1$  is diagonal,  $\sqrt{|\mathbf{C}_1|} = \boldsymbol{\sigma}_{ini}^D$ .

---

#### Algorithm 1 Summary of the IGMM algorithm

---

```

for all new training data  $\mathbf{x}^t$  do
  {Compute the probability of  $\mathbf{x}^t \in j$ th mixture component}
  for all mixture component  $j$  do
     $p(\mathbf{x}^t|j) = \mathcal{N}(\mathbf{x}^t|\boldsymbol{\mu}_j, \mathbf{C}_j)$ 
  end for
  {Create a new mixture component  $k$  if necessary}
  if  $M < 1$  or ( $v_j > v_{min}$  and  $p(\mathbf{x}^t|j) < \tau_{nov}/(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}$ ,  $\forall j$ ) then
     $M^* = M + 1; \quad v_k = 1; \quad sp_k = 1.0$ 
     $\boldsymbol{\mu}_k = \mathbf{x}^t; \quad \mathbf{C}_k = \boldsymbol{\sigma}_{ini}^2 \mathbf{I}$ 
     $p(\mathbf{x}^t|k) = \mathcal{N}(\mathbf{x}^t|\boldsymbol{\mu}_k, \mathbf{C}_k)$ 
     $p(j) = sp_j / \sum_{q=1}^k sp_q, \quad \forall j$ 
  end if
  {Compute the posterior probabilities}
  for all mixture component  $j$  do
     $p(j|\mathbf{x}^t) = \frac{p(\mathbf{x}^t|j)p(j)}{\sum_{q=1}^M p(\mathbf{x}^t|q)p(q)}$ 
  end for
  {Incremental estimation step}
  for all mixture component  $j$  do
     $sp_j^* = \alpha sp_j + p(j|\mathbf{x}^t)$ 
     $\boldsymbol{\mu}_j^* = \boldsymbol{\mu}_j + \frac{p(j|\mathbf{x}^t)}{sp_j^*} (\mathbf{x}^t - \boldsymbol{\mu}_j)$ 
     $\mathbf{C}_j^* = \mathbf{C}_j - (\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_j^* - \boldsymbol{\mu}_j)^T + \frac{p(j|\mathbf{x}^t)}{sp_j^*} [(\mathbf{x}^t - \boldsymbol{\mu}_j^*)(\mathbf{x}^t - \boldsymbol{\mu}_j^*)^T - \mathbf{C}_j]$ 
  end for
   $p(j)^* = sp_j^* / \sum_{q=1}^M sp_q^*, \quad \forall j$ 
  {Restart the accumulators and delete all spurious components}
  if ( $\sum_{j=1}^M sp_j^*$ )  $\geq \beta sp_{max}$  then  $sp_j^* = \gamma sp_j^*, \quad \forall j$ 
   $v_j = v_j + 1, \quad \forall j$ 
  if  $v_j > v_{min}$  and  $sp_j < sp_{min}$  then delete the  $j$ th component
end for

```

---

When a new training vector arrives, the component densities  $p(\mathbf{x}|j)$  are computed using the  $D$ -variate Gaussian function (2.3). The algorithm then decides if it is necessary to create a new component for the current data point  $\mathbf{x}^t$  based on the minimum likelihood criterium (2.36). If this criterium returns true, the new input vector  $\mathbf{x}$  is not considered

as a member of any existing component. In this case, a new Gaussian component  $k$  is created centered on  $\mathbf{x}^t$ , i.e.:

$$M^* = M + 1; \quad v_k = 1; \quad sp_k = 1.0; \quad \boldsymbol{\mu}_k = \mathbf{x}^t; \\ \mathbf{C}_k = \sigma_{ini}^2 \mathbf{I}; \quad p(\mathbf{x}^t|k) = \mathcal{N}(\mathbf{x}^t|\boldsymbol{\mu}_k, \mathbf{C}_k) \equiv (\sigma_{ini}^2 2\pi)^{-D/2} .$$

and all prior probabilities  $p(j), \forall j$ , are adjusted to satisfy constraints (2.2) by (2.65).

Otherwise (if Equation 2.36 returns false for at least one mixture component), the posterior probabilities are computed using the Bayes' theorem (Equation 2.37), and the existing mixture model is updated by (2.72), (2.62) and (2.63). At last, the  $sp$  accumulators are restarted and the spurious components are deleted if necessary.

IGMM has seven configuration parameters ( $sp_{min}$ ,  $sp_{max}$ ,  $\beta$ ,  $\gamma$ ,  $v_{min}$ ,  $\delta$  and  $\tau_{nov}$ ). Three of them, related to restarting of the accumulators ( $sp_{max}$ ,  $\beta$  and  $\gamma$ ), are actually constants, i.e., they are not critical and can always be set to the default values presented in Subsection 2.5.4. The configuration parameters related to the stability criterium ( $v_{min}$  and  $sp_{min}$ ) can be changed under special conditions (such as when the data are not temporarily correlated or are very noisy), but in general they can be set to  $sp_{min} = D + 1$  and  $v_{min} = 2D$  or even to zero (in this case the stability criterium will be ignored).

The  $\delta$  parameter, used to define the initial radius  $\sigma_{ini}$  of the covariance matrices  $\mathbf{C}_k$ , must be defined more carefully. The main requirement for  $\delta$  is to be large enough to avoid singularities, but if  $\sigma_{ini}$  is too large all training points would fall inside the decision frontier of the cluster, and consequently just a single cluster will be created. In general good results are achieved using  $0.005 \leq \delta \leq 0.05$ , but this may vary according to the statistical distribution of the training dataset.

The most critical configuration parameter is  $\tau_{nov}$ , which indicates how distant  $\mathbf{x}$  must be from  $\boldsymbol{\mu}_j$  to be considered a non-member of  $j$ . For instance,  $\tau_{nov} = 0.01$  indicates that  $p(\mathbf{x}|j)$  must be lower than one percent of the Gaussian height (probability at the mean) for  $\mathbf{x}$  to be considered a non-member of  $j$ . If  $\tau_{nov} \ll 0.01$ , for instance, few Gaussian distributions will be created (the GMM will be composed by few large clusters), and if  $\tau_{nov} > 0.01$ , more Gaussian distributions will be created and consequently the mixture model will be composed by many small clusters. In the limit, if  $\tau_{nov} = 1$  one distribution per training pattern will be created, and using  $\tau_{nov} = 0$  just the initial Gaussian distribution is created.

### 2.5.6 Computational complexity of IGMM

As each training pattern  $\mathbf{x}^n$  is processed just once by IGMM, its computational complexity is linear in  $N$ , that is, the execution time does not depend on the number of training patterns  $N$ . However, to compute  $p(\mathbf{x}|j)$  using the  $D$ -variate Gaussian function (2.3) is necessary to invert the covariance matrix  $\mathbf{C}_j$ , and this requires  $D^{\log_2 7}$  operations using the Strassen algorithm (STRASSEN, 1969), which is the fastest implementable algorithm<sup>1</sup> for square matrix multiplication (ROBINSON, 2005). Thus, the computational complexity of IGMM is  $O(NMD^{\log_2 7})$ , where  $M$  is the number of Gaussian components.

It can be noticed that, although IGMM is linear in  $N$ , the number of operations for each training pattern increases as new Gaussian components are added. Moreover, if the number of input features  $D$  is large ( $D \gg 10$ ), the execution times will be high.

<sup>1</sup>The asymptotically fastest known algorithm for square matrix multiplication is the Coppersmith-Winograd algorithm (COPPERSMITH DON; WINOGRAD, 1990), but unlike the Strassen algorithm (STRASSEN, 1969), it is not used in practice because it only provides an advantage for matrices so large that they cannot be processed by modern hardware (ROBINSON, 2005).

A solution for this problem is to use diagonal matrices in  $C$ , which corresponds to the *naïve* hypothesis explored in INBC (ENGEL, 2009). Another solution, presented in Sato and Ishii (2000), is to compute the inverse matrix  $C^{-1}$  directly, and in this case is not necessary to invert  $C$ . But in this case the recursive equations for updating  $\theta$  must be changed accordingly, and the interpretation of the results may require the inversion of  $C^{-1}$ . A large number of input features  $D$  may also result in numeric imprecisions (in (2.3), for instance, the denominator of the first term will be very small), but this can be easily solved using a logarithmic formulation, e.g.:  $a/b = \exp(\log a - \log b)$ .

## 2.6 Comparative

Table 2.1 summarizes the main characteristics of the algorithms for learning Gaussian mixture models presented in this chapter. The characteristics required for incremental function approximation are shown in blue and in uppercase. The first column (*Main*

Table 2.1: Characteristics of the GMM algorithms described in this chapter

Main reference	Sensitive to initialization	Fixed number of distrib.	Batch learning	Merging splitting	Data come in blocks	Slowly variable	Covariance matrix
(MACQUEEN, 1967)	NO	yes	yes	NO	NO	NO	-
(DEMPSTER; LAIRD; RUBIN, 1977)	yes	yes	yes	NO	NO	NO	FULL C
(FIGUEIREDO; JAIN, 2002)	NO	NO	yes	NO	NO	NO	FULL C
(STAUFFER; GRIMSON, 1999)	NO	yes	NO	NO	NO	NO	Diagonal
(JEPSON; FLEET; EL-MARAGHI, 2003)	yes	yes	NO	NO	NO	yes	Diagonal
(GOMES; WELLING; PERONA, 2008)	NO	NO	NO	yes	yes	NO	-
(ARANDJELOVIC; CIPOLLA, 2005)	yes	NO	NO	yes	NO	yes	FULL C
(KRISTAN; SKOCAJ; LEONARDIS, 2008)	yes	NO	NO	yes	NO	NO	Diagonal
(ENGEL; HEINEN, 2010a,b)	NO	NO	NO	NO	NO	yes	FULL C

*reference*) shows the main reference of the algorithm. The second column (*Sensitive to initialization*) informs if the learning algorithm is sensitive to initialization, i.e., if it starts from an existing GMM and/or requires an initial guess about the Gaussian distributions. The third column (*Fixed number of distributions*) indicates if the number of Gaussian distributions must be previously known and fixed. The fourth column (*Batch-mode learning*) informs if the GMM uses a batch-mode learning algorithm, i.e., if it requires several scans over the complete training dataset to converge. The fifth column (*Merging splitting*) indicates if the learning algorithm uses merging and splitting operations to control the number of Gaussian components. The sixth column (*Data come in blocks*) indicates if the corresponding model assumes that the data come in blocks. The seventh column (*Slowly variable*) informs if the algorithm requires temporal coherency among the input data, and the eighth column (*Covariance matrix*) indicates the kind of covariance matrix used in the Gaussian components (the hyphen symbol indicates that the corresponding model does not use covariance matrices, i.e., it is an distance-based algorithm) according to following description:

- **Diagonal:** The covariance matrices are diagonal;
- **FULL C:** The corresponding model uses full covariance matrices.

In can be noticed that all algorithms for learning GMMs presented in this chapter assume at least one restriction. Three of them (MACQUEEN, 1967; DEMPSTER; LAIRD; RUBIN, 1977; FIGUEIREDO; JAIN, 2002) are off-line solutions, and thus cannot be used in incremental function approximation. The algorithm presented in Gomes et al. (2008)

assumes that the training data come in blocks, which is a strong constraint for many applications. Other algorithms (STAUFFER; GRIMSON, 1999; JEPSON; FLEET; ELMARAGHI, 2003) assume that the number of Gaussian distributions is previously known and fixed, which prevent them to adapt for environment changes and/or learning new concepts. The GMM learning algorithms proposed in Jepson et al. (2003), Arandjelovic and Cipolla (2005) and Kristan et al. (2008) are sensitive to initialization and/or starts from existing GMMs, i.e., they do not tackle the problem of how to initialize the Gaussian distributions. Moreover, the algorithms presented in Arandjelovic and Cipolla (2005), Gomes et al. (2008) and Kristan et al. (2008) use *batch-mode* operations (e.g., merging and splitting) to change the number of distributions.

The GMM learning algorithm proposed in this thesis (last row at Table 2.1), on the other hand, starts from scratch, learns incrementally from data flows, does not assume that the data come in blocks, is able to create new components on demand and does not use merging and splitting operations. In fact, the only constraint assumed by IGMM is that the environment must be locally stable, but this restriction is less severe than that assumed in Arandjelovic and Cipolla (2005), for instance, because IGMM can deal even with abrupt changes in the input data.

## 3 ARTIFICIAL NEURAL NETWORKS

This chapter presents some state-of-the-art artificial neural network (ANN) models related to this thesis, their main characteristics and limitations. Initially Section 3.1 describes the Multi-layer Perceptrons (MLP) (RUMELHART; HINTON; WILLIAMS, 1986) architecture, which is the most well known and used neural network model. Then Section 3.2 presents the radial basis functions (RBF) network (POWELL, 1985, 1987a,b), which is an important kind of neural network used mainly in non-linear regression. Section 3.3 describes the self-organizing map (SOM) (KOHONEN, 1990, 2001), an unsupervised ANN model characterized by the formation of a topological map from the input patterns. Sections 3.4 and 3.5 describe, respectively, the ART (CARPENTER; GROSSBERG, 1987) and ARTMAP (CARPENTER; GROSSBERG; REYNOLDS, 1991) models, which are ANNs based on the adaptive resonance theory (GROSSBERG, 1976a,b, 2000). Sections 3.6 and 3.7 present, respectively, the PNN (SPECHT, 1988, 1990) and GRNN (SPECHT, 1991) networks, which are probabilistic models that learn instantaneously using a single presentation of each training sample. Section 3.8 shows some improvements made over PNN and GRNN to tackle their main limitations, specially the requirement of a separate neuron for each training pattern. Section 3.9 describes other not well known but interesting ANN models. Finally, Section 3.10 summarizes and compares the neural network models described in this chapter.

### 3.1 Multi-layer Perceptrons

Multi-layer Perceptrons (MLP) (RUMELHART; HINTON; WILLIAMS, 1986) represent the most prominent and well researched class of ANNs for classification and function approximation, implementing a feed-forward and supervised learning paradigm. It consists of several layers of neurons, interconnected through weighted acyclic arcs from each preceding layer to the following, without lateral or feedback connections. Each node computes a transformed weighted combination of its inputs of the form

$$y_j(n) = \varphi(v_j(n)) \quad (3.1)$$

where  $\varphi(\cdot)$  is the activation function and  $v_j(n)$  is the induced local field of the neuron  $j$ , defined by

$$v_j(n) = \sum_{i=0}^D w_{ji}(n)y_i(n) \quad (3.2)$$

where  $D$  is the total number of inputs applied to neuron  $j$ , and  $w_{ji}$  is the synaptic weight connecting neuron  $i$  to neuron  $j$ , and  $y_i(n)$  is the input signal of neuron  $j$  or equivalently,

the function signal appearing at the output of neuron  $i$ . A commonly used activation function is the logistic function given by:

$$y_j(n) = \frac{1}{1 + \exp(-v_j(n))}. \quad (3.3)$$

The algorithm used to train MLP networks, called Backpropagation (RUMELHART; HINTON; WILLIAMS, 1986), is a supervised learning method based on gradient descent which generalizes the Delta rule proposed by Widrow and Hoff (1960).

MLPs are useful in function approximation because they can approximate any given continuous function on any compact subset to any degree of accuracy, provided that a sufficient number of hidden layer neurons is used (POGGIO; GIROSI, 1989). This is proved by the *universal approximation theorem*, which asserts that the standard multilayer feed-forward networks with a single hidden layer that contains finite number of hidden neurons, and with arbitrary activation function<sup>1</sup> are universal approximators in  $C(\mathbb{R}^D)$  (CSÁJI, 2001). According to Hornik (1991), it is the multilayer feed-forward architecture itself which gives artificial neural networks the potential of being universal approximators, and this potential does not depend on a specific choice of the activation function. It is important to say that although the MLP architecture can represent any continuous function given a sufficient number of hidden neurons, this function cannot necessarily be learned by the Backpropagation algorithm, i.e., the convergence in Backpropagation learning is not guaranteed (HAYKIN, 2008).

MLPs can be used in classification and regression, but generally not in clustering tasks. The main disadvantages of MLP are: (i) the Backpropagation algorithm requires several scans (called epochs) over all training data to converge for a good solution, i.e., to reduce to an acceptable value the error between the desired and actual outputs; (ii) it suffers from the so called “catastrophic interference” (FRENCH, 2003), which occurs when the newly learned patterns suddenly and completely erase the network memory of the previously learned patterns; (iii) the neural network topology (i.e., number of hidden layers and the number of neurons in each hidden layer) must be defined a priori and kept fixed; (iv) the learning algorithm has many critical parameters (e.g., learning rate, momentum, weight decay) which must be finely tuned for the learning process to properly occur. The neural network topology and the configuration parameters are usually set in a trial and error process, which requires several hours of a human specialist, and the obtained configuration is specific for the current task (there is no default values that work in most cases). Some of these problems are tackled by incremental ANN models, like the Fahlman’s Cascade Correlation (FAHLMAN; LEBIERE, 1990), which automatically sets the neural architecture and the learning parameters. But these models still require several scans over the entire training dataset to converge.

### 3.2 Radial basis function networks

Radial basis functions (RBF) networks (POWELL, 1985, 1987a,b) constitute a widely used and researched tool for nonlinear function approximation. They typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a

---

<sup>1</sup> $\varphi$  is an activation function if and only if  $\varphi$  is bounded and  $\lim_{x \rightarrow +\infty} \varphi(x) = a$ ,  $\lim_{x \rightarrow -\infty} \varphi(x) = b$ ,  $a \neq b$ .

linear output layer. The output,  $\varphi : \mathbb{R}^D \rightarrow \mathbb{R}$ , of the network is thus

$$\varphi(\mathbf{x}) = \sum_{i=1}^M w_i \rho(\|\mathbf{x} - \mu_i\|) \quad (3.4)$$

where  $M$  is the number of neurons in the hidden layer,  $\mu_i$  is the center vector of the neuron  $i$ , and  $w_i$  are the weights of the linear output neuron. In its basic form all inputs are connected to each hidden neuron. The norm is typically the Euclidean distance and the basis functions are generally Gaussian, i.e.,

$$\rho(\|\mathbf{x} - \mu_i\|) = \exp[-\beta \|\mathbf{x} - \mu_i\|^2]. \quad (3.5)$$

The Gaussian basis functions are local in the sense that changing parameters of one neuron has only a small effect for input values that are far away from the center of that neuron, i.e.:

$$\lim_{\|\mathbf{x}\| \rightarrow \pm\infty} \rho(\|\mathbf{x} - \mu_i\|) = 0 \quad (3.6)$$

This local effect is specially useful in tasks like reinforcement learning (SUTTON; BARTO, 1998), because it prevents that the knowledge acquired in a region not recently visited be destroyed by changes made in other regions of the state space (SMART, 2002).

RBF networks are universal approximators on a compact subset of  $\mathbb{R}^D$  (POWELL, 1987a; SHAHSAVAND, 2009). This means that a RBF network with enough hidden neurons can approximate any continuous function with arbitrary precision. The parameters  $w_i$ ,  $\mu_i$ , and  $\beta$  are determined in a manner that optimizes the fit between  $\varphi$  and the data. In general RBF networks are used just in regression tasks.

### 3.3 Self-organizing maps

A self-organizing map (SOM), also called a Kohonen map (KOHONEN, 1990, 2001), is an artificial neural network model that is trained using unsupervised learning to produce a low-dimensional, discretized representation of the input space. In a self-organizing map, the neurons are placed at the nodes of a *lattice*, that is usually one-dimensional or two-dimensional. Through a competitive learning process, the neurons of the lattice become ordered with respect to each other in such a way that a meaningful coordinate system for input features is created over the lattice (KOHONEN, 2001). SOM is therefore characterized by the formation of a *topographic map* of the input patterns, in which the spatial locations of the neurons indicate intrinsic statistical features of the input patterns (HAYKIN, 2008).

The competitive learning process used by SOM works as follows. Initially the weight vectors  $\{\mathbf{w}_j\}_{j=1}^M$  are chosen either to small random values or sampled evenly from the available set of input training vectors  $\{\mathbf{x}_i\}_{i=1}^N$ . Using the latter alternative the learning process is usually faster because the initial weights are already good approximations of the SOM weight vectors (KOHONEN, 2001). When a training sample  $\mathbf{x}^t$  is fed to the network, the neuron with weight vector most similar to the input vector, i.e., the winning neuron  $i(\mathbf{x})$ , is computed through

$$i(\mathbf{x}) = \arg \min_j \|\mathbf{x}^t - \mathbf{w}_j\|, \quad j = 1, 2, \dots, M. \quad (3.7)$$

After this, the synaptic weight vectors of all neurons are adjusted using the following update equation:

$$\mathbf{w}_j^* = \mathbf{w}_j + \eta_t h_{j,i(\mathbf{x})}(\mathbf{x}^t - \mathbf{w}_j), \quad (3.8)$$

where  $\eta_t$  is the learning rate parameter and  $h_{j,i(\mathbf{x})}$  is the neighborhood function centered around the winning neuron  $i(\mathbf{x})$ . A common choice for the neighborhood function is the Gaussian function:

$$h_{j,i(\mathbf{x})} = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2}\right), \quad (3.9)$$

where  $\sigma$  is a parameter that controls the size of the topology and  $d_{j,i}$  is the Euclidean distance between the lattice positions of neuron  $j$ ,  $\mathbf{r}_j$ , and the winning neuron  $i(\mathbf{x})$ ,  $\mathbf{r}_i$ :

$$d_{j,i} = \|\mathbf{r}_j - \mathbf{r}_i\|. \quad (3.10)$$

A requirement of the SOM learning algorithm is that the size of the topological neighborhood shrinks with time, which is achieved using an exponential decay of  $\sigma$ :

$$\sigma_t = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right), \quad t = 0, 1, 2, \dots \quad (3.11)$$

where  $\sigma_0$  is the initial value of  $\sigma$  and  $\tau_1$  is a time constant set by the user. Another requirement of the learning algorithm is that the learning rate  $\eta_t$  must decrease gradually with increasing time  $t$ , which is also achieved using an exponential decay:

$$\eta_t = \eta_0 \exp\left(-\frac{t}{\tau_2}\right), \quad t = 0, 1, 2, \dots \quad (3.12)$$

where  $\eta_0$  is the initial value of  $\eta$  and  $\tau_2$  is another time constant set by the user. The learning process is repeated for each input vector  $\mathbf{x}$  for a (usually large) number of epochs until no noticeable changes in the feature map are observed (KOHONEN, 2001).

SOM is very useful for clustering and visualizing high-dimensional data, because it describes a mapping from a higher dimensional input space to a lower dimensional, typically two-dimensional, map space. Its main limitations are: (i) the learning algorithm requires several epochs to converge; (ii) there are separate phases for training and mapping, i.e., the standard SOM cannot learn continuously; and (iii) the neural network architecture (number of neurons in the lattice) must be previously configured and kept fixed. Some of these drawbacks are tackled by variations of the SOM architecture, like GTSOM (BAS-TOS, 2007; MENEGAZ; ENGEL, 2009), which can automatically set the neural network architecture during learning in an incremental way. Moreover, GTSOM has no separate phases for training and mapping, i.e., it can learn continuously and perpetually. However, GTSOM still requires many training epochs to converge.

### 3.4 Adaptive Resonance Theory

Adaptive resonance theory (ART) is a theory developed by Grossberg (1976a; 1976b) on aspects of how the brain processes information (GROSSBERG, 1982). It describes a number of neural network models (e.g., ART1, Fuzzy ART, ARTMAP), that uses supervised and unsupervised learning methods, and addresses problems such as pattern recognition and prediction (GROSSBERG, 2000). A central feature of ART is a pattern matching process, that compares an external input  $\mathbf{I}$  with the internal memory of an active code. This matching process leads either to a *resonant* state, which persists long enough to allow learning, or to a parallel memory search. If the search ends at an established code, the memory representation may either remain the same or incorporate new information from



matched portions of the current input  $I$ . If the search ends at a new code, the memory representation learns the current input  $I$  (CARPENTER; GROSSBERG, 2003).

The simplest ART model is an unsupervised learning network called ART1, which accepts only binary input data. Figure 3.1, reproduced from Carpenter and Grossberg (1987), shows its typical architecture, that consists of:

- **Two layers of binary neurons:** the comparison layer, called  $F_1$ , and the recognition layer, called  $F_2$ ;
- **A vigilance parameter**,  $\rho$ , which specifies the minimum fraction of the input that must remain in the matched pattern for resonance to occur;
- **A reset module**, that compares the strength of the recognition match to the vigilance parameter  $\rho$ .

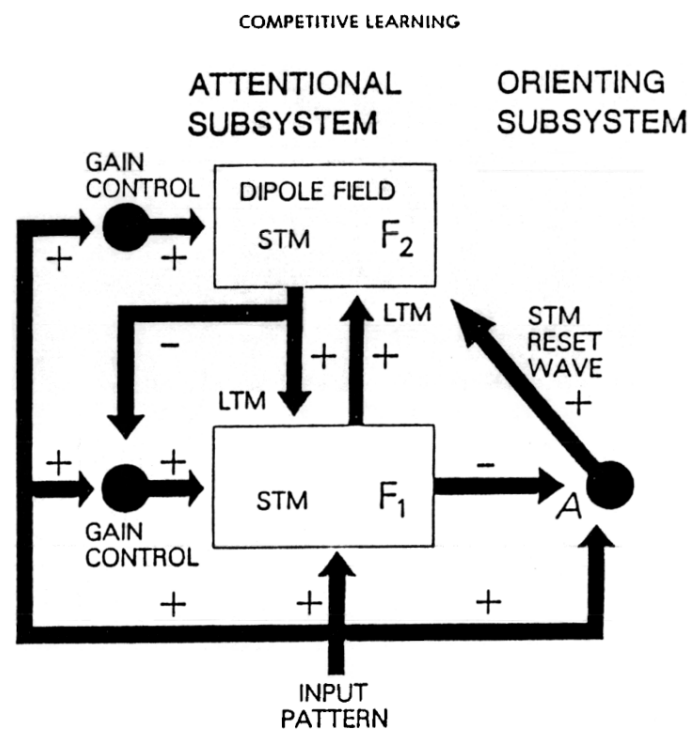


Figure 3.1: Typical architecture of ART1

The vigilance parameter  $\rho$  has considerable influence on the system: higher vigilance produces highly detailed memories (many, fine-grained categories), while lower vigilance results in more general memories (fewer, more-general categories). By varying  $\rho$  the system can recognize both abstract categories, such as faces and cats, and individual examples of these categories such as my cat (CARPENTER; GROSSBERG, 2003).

There are two sets of synaptic weights connecting  $F_1$  and  $F_2$  in the ART1 model:

- **Bottom-up weights:** connecting  $F_1 \rightarrow F_2$ ;
- **Top-down weights:** connecting  $F_2 \rightarrow F_1$ .

In the following the basic operation of the ART1 learning algorithm is described. To simplify our notation, in this section the subscripts  $i$  and  $j$  will be used exclusively to refer to the layers  $F_1$  and  $F_2$ , respectively. Therefore, the neurons on the  $F_1$  and  $F_2$  layers will be called, respectively,  $v_i$  and  $v_j$ , the top-down ( $F_2 \rightarrow F_1$ ) weights will be called  $w_{ji}$  and

the bottom-up ( $F_1 \rightarrow F_2$ ) weights  $w_{ij}$ . This notation follows that presented in Freeman and Skapura (1991).

Before the learning process starts, it is necessary to initialize the weights and the neural activities of  $F_1$  and  $F_2$ . The weights of the top-down connections ( $v_j \rightarrow v_i$ ),  $w_{ji}$ , are initialized according to

$$w_{ji}(0) > \frac{B_1 - 1}{D_1}, \quad (3.13)$$

where  $B_1$  and  $D_1$  are configuration parameters chosen according to the following constraints:

$$\begin{aligned} D_1 &\geq 0 \\ \max\{D_1, 1\} &< B_1 < D_1 + 1. \end{aligned} \quad (3.14)$$

Other configuration parameters are  $A_1 \geq 0$ ,  $C_1 \geq 0$  and the vigilance parameter  $\rho$ , which must be chosen in the interval  $0 < \rho \leq 1$ . The bottom-up weights, connecting  $F_1 \rightarrow F_2$ , are initialized according to

$$0 < w_{ij}(0) < \frac{L}{L - 1 + M}, \quad (3.15)$$

where  $L > 1$  is an ART1 constant and  $M$  is the number of units on  $F_1$ . The activities on  $F_2$  are then initialized to zero, and the activities on  $F_1$  are initialized in the following way:

$$x_{1i}(0) = \frac{-B_1}{1 + C_1}. \quad (3.16)$$

When a new binary input vector  $\mathbf{I} \in \{0, 1\}$  is applied to  $F_1$ , the activities on  $F_1$  are computed according to

$$x_{1i} = \frac{I_i}{1 + A_1(I_i + B_1) + C_1}, \quad (3.17)$$

and the output vector for  $F_1$ ,  $\mathbf{S} = \{s_1, \dots, s_i, \dots, s_M\}$ , is computed through

$$s_i = h(x_{1i}) = \begin{cases} 1 & \text{if } x_{1i} > 0 \\ 0 & \text{if } x_{1i} \leq 0. \end{cases} \quad (3.18)$$

This output vector  $\mathbf{S}$  is propagated to  $F_2$ , and the activities  $T_j, \forall j \in M$ , are computed:

$$T_j = \sum_{i=1}^M s_i w_{ij}. \quad (3.19)$$

These  $T_j$  activities are used to compute the outputs on  $F_2$ :

$$u_j = \begin{cases} 1 & T_j = \max_k \{T_k\}, \forall k \\ 0 & \text{otherwise} \end{cases} \quad (3.20)$$

It is important to notice that only the ‘‘winning’’ node of  $F_2$ ,  $v_j$ , has a nonzero output. The output from  $F_2$  is propagated back to  $F_1$ , and the *net* inputs from  $F_2$  to the  $F_1$  units,  $V_i$ , are given by:

$$V_i = \sum_{j=1}^N u_j w_{ji}, \quad (3.21)$$

the new activities on  $F_1$  are recomputed through

$$x_{1i} = \frac{I_i + D_1 V_i - B_1}{1 + A_1(I_i + D_1 V_i) + C_1}, \quad (3.22)$$

and the new output vector  $\mathbf{S}$  is recomputed using (3.18). After this, it is necessary to determine the degree of match between the input pattern  $\mathbf{I}$  and the top-down template, i.e.:

$$\frac{|\mathbf{S}|}{|\mathbf{I}|} = \frac{\sum_{i=1}^M s_i}{\sum_{i=1}^M I_i}. \quad (3.23)$$

If  $|\mathbf{S}|/|\mathbf{I}| < \rho$ , then  $v_j$  is marked as inactive, the outputs of  $F_2$  are set to zero and the algorithm returns to the first step using the original input pattern  $\mathbf{I}$ . Otherwise, the learning process continues updating the bottom-up weights on  $v_j$  (the “winning” node) only:

$$w_{ij} = \begin{cases} \frac{L}{L-1+|\mathbf{I}|} & \text{if } v_i \text{ is active} \\ 0 & \text{if } v_i \text{ is inactive,} \end{cases} \quad (3.24)$$

and the top-down weights coming from  $v_j$  only to all  $F_1$  units are updated through:

$$w_{ji} = \begin{cases} 1 & \text{if } v_i \text{ is active} \\ 0 & \text{if } v_i \text{ is inactive.} \end{cases} \quad (3.25)$$

Finally, the input pattern is removed, all inactive units on  $F_2$  are restored and the algorithm proceeds from the beginning using a new input pattern (FREEMAN; SKAPURA, 1991).

As mentioned above, the basic ART1 architecture deals only with binary input patterns. But an improved ART architecture, called Fuzzy ART (CARPENTER; GROSSBERG; ROSEN, 1991), generalizes ART1 for learning stable recognition categories in response to both analog and binary input patterns. This improvement is achieved by incorporating computations from fuzzy set theory (ZADEH, 1965) into the ART1 neural network. In Fuzzy ART, the *crispy* intersection operator ( $\cap$ ) is replaced by the *fuzzy* MIN operator ( $\wedge$ ), and the *crispy* union operator ( $\cup$ ) is replaced by the *fuzzy* MAX operator ( $\vee$ ). The fast learning operation, for instance, which in ART1 is given by an intersection between  $\mathbf{I}$  and  $\mathbf{w}$ :  $\mathbf{w}^* = \mathbf{I} \cap \mathbf{w}$ , in Fuzzy ART is computed using  $\mathbf{w}^* = \mathbf{I} \wedge \mathbf{w}$ , where the MIN operator ( $\wedge$ ) of fuzzy set theory replaces the intersection operator ( $\cap$ ) used in ART1.

To prevent category proliferation, Fuzzy ART uses a normalization procedure called complement coding, which actually doubles the number of input components, presenting to the network both the original feature vector and its complement. According to Carpenter et al. (1991), “complement coding leads to a symmetric theory in which the MIN operator ( $\wedge$ ) and the MAX operator ( $\vee$ ) of fuzzy set theory play complementary roles”. Moreover, it is generally necessary to normalize the input vectors at a preprocessing stage to prevent category proliferation.

The main advantages of the ART1 and Fuzzy ART are (CARPENTER; GROSSBERG, 2003): (i) they can learn incrementally and continuously, what make them well suited for on-line learning of large and evolving databases; (ii) the adaptive resonance theory tackles the stability-plasticity dilemma and does not suffer from catastrophic interference; (iii) the learning process can be fast, thus enabling a system to adapt quickly to inputs that rarely occur but that may require immediate accurate recall. The main drawbacks of these unsupervised ART models are (SARLE, 1995): (i) they depend critically upon the order in which the training data are processed; and (ii) the estimates produced by them do not possess the statistical property of consistency.

### 3.5 ARTMAP

The ARTMAP model, also called predictive ART (CARPENTER; GROSSBERG; REYNOLDS, 1991), is a neural network classifier that combines two ART modules into

a supervised learning structure, where the first module ( $ART_a$ ) takes the input data  $\mathbf{a}$  and the second module ( $ART_b$ ) takes the correct output data  $\mathbf{b}$ . ARTMAP uses these modules to make the minimum possible adjustment of the vigilance parameter of the first module ( $\rho_a$ ) in order to make the correct classification or prediction. The main elements of the ARTMAP architecture are shown in Figure 3.2 (reproduced from Carpenter et al. (CARPENTER; GROSSBERG; REYNOLDS, 1991)).

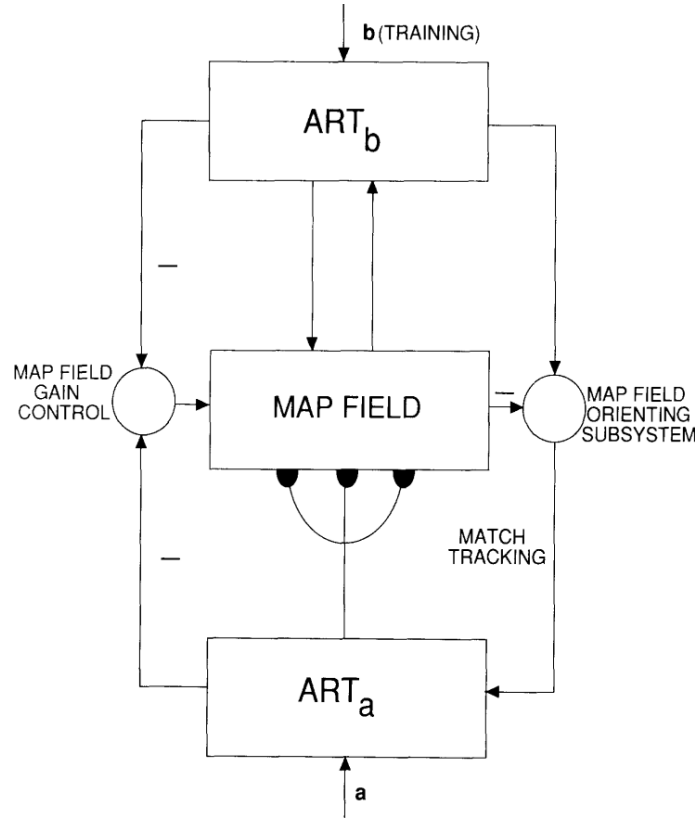


Figure 3.2: ARTMAP architecture

$ART_a$  and  $ART_b$  modules are connected by an inter-ART module, which is in many aspects similar to an ART1 network (e.g., it receives only binary data). This inter-ART module includes a so called *map field*  $F^{ab}$ , that controls the learning of an associative map from  $ART_a$  to  $ART_b$  recognition categories. The map field also controls the match tracking of the  $ART_a$  vigilance parameter  $\rho_a$ . A mismatch at the map field between the  $ART_a$  category activated by  $\mathbf{a}$  and the  $ART_b$  category activated by  $\mathbf{b}$  increases  $\rho_a$  by the minimum amount needed for the system to search for and, if necessary, learn a new  $ART_a$  category whose prediction matches the  $ART_b$  category (CARPENTER; GROSSBERG; REYNOLDS, 1991).

The match tracking process of ARTMAP works as follows. Assume that  $\mathbf{x}$  is the output vector of the map field  $F^{ab}$ ,  $\rho$  is the vigilance parameter of  $F^{ab}$ ,  $\mathbf{y}^b$  is the output vector of  $F_2^b$  (the second layer of  $ART_b$ ),  $\mathbf{x}^a$  is the output vector of  $F_1^a$  (the first layer of  $ART_a$ ),  $\mathbf{z}_j^a$  is the top-down input of  $F_1^b$  and  $\mathbf{w}_j$  is a weight vector that links the  $j$ th node of  $F_2^a$  to the nodes in  $F^{ab}$ . When a new data sample  $\{\mathbf{a}, \mathbf{b}\}$  is presented to ARTMAP, the vigilance parameter of the  $ART_a$  module,  $\rho_a$ , is set to be equal a baseline vigilance  $\bar{\rho}_a$ . If  $|\mathbf{x}| < \rho|\mathbf{y}^b|$ , then  $\rho_a$  is increased until it is slightly larger than  $|\mathbf{a} \cap \mathbf{z}_j^a| |\mathbf{a}|^{-1}$ . Then

$$|\mathbf{x}^a| = |\mathbf{a} \cap \mathbf{z}_j^a| < \rho_a |\mathbf{a}|, \quad (3.26)$$

where  $\mathbf{a}$  is the current  $\text{ART}_a$  input vector and  $j$  is the index of the active  $F_2^a$  node. When this occurs,  $\text{ART}_a$  search leads either to activation of a new  $F_2^a$  node  $j$  with

$$|\mathbf{x}^a| = |\mathbf{a} \cap \mathbf{z}_j^a| \geq \rho_a |\mathbf{a}| \quad (3.27)$$

and

$$|\mathbf{x}| = |\mathbf{y}^b \cap \mathbf{w}_j| \geq \rho |\mathbf{y}^b|, \quad (3.28)$$

or, if this node does not exist, to the shut-down of  $F_2^a$  for the remainder of the input presentation. ARTMAP can be seen as a self-organizing system that calibrates the selectivity of its hypotheses based upon predictive success. As a result, rare but important events can be distinguished even if they are similar to frequent events with different consequences (CARPENTER; GROSSBERG; REYNOLDS, 1991).

To prevent category proliferation, ARTMAP employs the same complement coding procedure introduced in the Fuzzy ART model described above (CARPENTER; GROSSBERG; ROSEN, 1991). ARTMAP has also a *fuzzy* version, called Fuzzy ARTMAP (CARPENTER et al., 1992), which accepts both analog and binary input patterns (the standard ARTMAP accepts only binary data). The main advantages of ARTMAP are the same of ART1 and Fuzzy ART, i.e., it can learn fast and continuously, solves the stability-plasticity dilemma and does not suffer from catastrophic interference. The main drawbacks of ARTMAP and Fuzzy ARTMAP are: (i) they are restricted to supervised classification and prediction using discrete categories; (ii) their estimates are not statistically consistent (SARLE, 1995); and (iii) the error-driven learning used in the match tracking process can result in a catastrophic allocation of new nodes, specially in overlapping decision regions (HAMKER, 2001).

### 3.6 Probabilistic neural networks

The probabilistic neural network (PNN) model (SPECHT, 1988, 1990) is a feed-forward ANN based on nonparametric Parzen's window estimators (PARZEN, 1962) of conditional probability density functions (pdf). It is used to classify patterns in order to minimize the "expected risk" according to the Bayes' strategy for decision making (BERGER, 1980; TAN; STEINBACH; KUMAR, 2006). Consider the situation in which the state of the environment  $\mathcal{S}$  is known to be either  $\mathcal{S}_A$  or  $\mathcal{S}_B$ , and we must decide whether  $\mathcal{S} = \mathcal{S}_A$  or  $\mathcal{S} = \mathcal{S}_B$  based on a set of measurements represented by a  $D$ -dimensional input vector  $\mathbf{x}$ . In this case, the Bayes' decision rule becomes:

$$\begin{aligned} d(\mathbf{x}) &= \mathcal{S}_A & \text{if } h_A l_A f_A(\mathbf{x}) > h_B l_B f_B(\mathbf{x}) \\ d(\mathbf{x}) &= \mathcal{S}_B & \text{if } h_A l_A f_A(\mathbf{x}) < h_B l_B f_B(\mathbf{x}) \end{aligned}$$

where  $h_A$  is the a priori probability of occurrence of patterns from category  $A$ ,  $h_B = 1 - h_A$  is the a priori probability of occurrence of patterns from category  $B$ ,  $f_A(\mathbf{x})$  is the probability density function (pdf) for class  $A$ ,  $f_B(\mathbf{x})$  is the pdf for class  $B$ ,  $l_A$  is the loss function associated with the decision  $d(\mathbf{x}) = \mathcal{S}_B$  when  $\mathcal{S} = \mathcal{S}_A$ , and  $l_B$  is the loss associated with the decision  $d(\mathbf{x}) = \mathcal{S}_A$  when  $\mathcal{S} = \mathcal{S}_B$ . The losses associated with correct decisions are taken to be equal to zero (SPECHT, 1990).

Figure 3.3, reproduced from Specht (1992), shows a neural network organization for classification of the input pattern  $\mathbf{x}$  into two categories,  $A$  and  $B$ . The typical PNN architecture is composed of many interconnected processing units, i.e., artificial neurons, organized in four successive layers. As occurs in other ANN models, the input layer unit

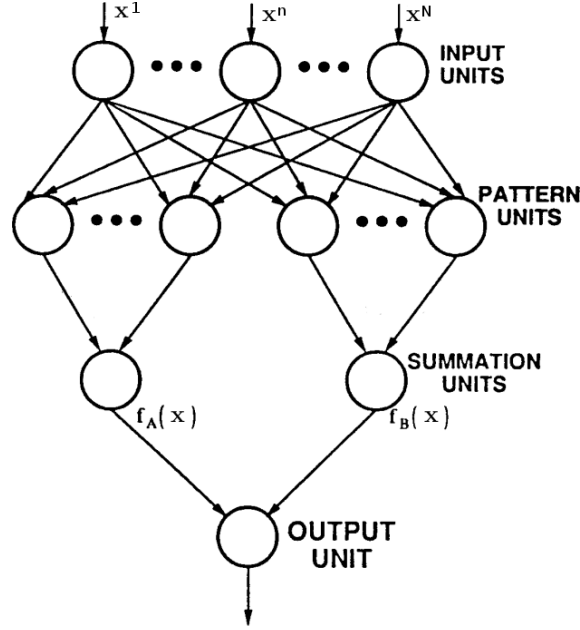


Figure 3.3: PNN for classification of patterns in two categories

does not perform any computation – it is composed merely by distribution units that supply the same input values to all units of the pattern layer. Each pattern unit  $j$  performs a dot product of the input vector  $\mathbf{x}$  with a weight vector  $\mathbf{w}_j$ , i.e.,  $\mathbf{z} = \mathbf{x} \cdot \mathbf{w}_j$ , and then perform the nonlinear operation  $\exp[(\mathbf{z}_j - 1)/\sigma^2]$  before sending its output to the next layer. Assuming that both  $\mathbf{x}$  and  $\mathbf{w}_j$  are normalized to unit length, which is necessary because the pattern units have the same variance  $\sigma^2$  in each direction, the output of the  $j$ th unit becomes

$$p(\mathbf{x}|j) = \exp \left[ -\frac{(\mathbf{x} - \mathbf{w}_j)^T (\mathbf{x} - \mathbf{w}_j)}{2\sigma^2} \right], \quad (3.29)$$

where the standard deviation  $\sigma$  is the so called “smoothing parameter”. Equation 3.29 falls into the class of consistent estimators proposed by Parzen (1962) and extended to the multidimensional case by Cacoullos (1966). The summation units simply sum the inputs from the pattern units corresponding to the category from which the training pattern was selected, i.e.:

$$f_i(\mathbf{x}) = \sum_{j=1}^{N_i} p(\mathbf{x}|j), \quad (3.30)$$

where  $N_i$  denotes the total number of samples in category  $i \in \{A, B\}$ . The decision units produce a binary network output. They have only a single variable weight,  $C$ , computed through

$$C = \frac{h_B l_B}{h_A l_A} \cdot \frac{N_A}{N_B}, \quad (3.31)$$

where  $N_A$  and  $N_B$  are the number of training patterns from categories  $A$  and  $B$ , respectively. If  $N_A$  and  $N_B$  are obtained in proportion to their a priori probabilities, then this equation can be simplified to  $C = l_B/l_A$ , i.e.,  $C$  is determined only from the significance of the decision. Moreover, if the losses associated with making an incorrect decision are the same for both classes, then  $C$  can be simplified to  $-1$  (an inverter).

The most important advantage of the probabilistic neural network is that training is easy and instantaneous. In fact, it is accomplished by just setting each training pattern  $\mathbf{x}$

equal to the  $w_j$  weight vector in one of the pattern units, and then connecting the output of the pattern unit to the appropriate summation unit. Another advantage of PNN is that it is guaranteed to asymptotically approach the Bayes' optimal decision surface provided that the class pdfs are smooth and continuous (RUTKOWSKI, 2004a).

The main drawback of PNN is that a separate neuron (pattern unit) is required for every training pattern, so in large databases an enormous amount of storage memory may be required (BHATTACHARYYA et al., 2008). Moreover, the amount of computation necessary to classify an unknown pattern  $\mathbf{x}$  is proportional to the size of the training set. Another limitation of PNN is that all pattern units are "isotropic Gaussians", i.e., they have the same width  $\sigma$  in each dimension, which makes PNN not robust with respect to affine transformations of feature space (MONTANA, 1992). Moreover, the smoothing parameter  $\sigma$  is critical and must be properly configured to obtain good results. Specht suggests to use the *holdout method* (FUKUNAGA, 1990; TAN; STEINBACH; KUMAR, 2006) for setting the  $\sigma$  parameter, but this method requires repeating the training process  $N$  times using the entire dataset.

### 3.7 General regression neural network

The general regression neural network (GRNN), proposed by Specht (1991), falls into the category of probabilistic neural networks described above. It is a memory-based network based on the theory of nonlinear kernel regression (BISHOP, 1995), which provides estimates of continuous variables and converges to the underlying regression surface. The mathematical formulation of GRNN is given as follows. Assume that  $f(\mathbf{X}, Y)$  represents the known joint continuous pdf of a vector random variable,  $\mathbf{X}$ , and a scalar random variable,  $Y$ . Let  $\mathbf{x}$  be a particular measured value of  $\mathbf{X}$ . The conditional mean of  $Y$  given  $\mathbf{x}$ , also called the regression of  $Y$  on  $\mathbf{x}$ , is given by

$$E[Y|\mathbf{x}] = \frac{\int_{-\infty}^{\infty} Y f(\mathbf{x}, Y) dY}{\int_{-\infty}^{\infty} f(\mathbf{x}, Y) dY}. \quad (3.32)$$

When the density  $f(\mathbf{X}, Y)$  is not known, it must be usually estimated from the observations  $\mathbf{x}$  and  $y$ . GRNN uses the class of consistent estimators proposed by Parzen (1962) and extended to the multidimensional case by Cacoullos (1966) for a nonparametric estimate of  $f(\mathbf{X}, Y)$ . Using these nonparametric estimators, the conditional mean  $\hat{y}$  becomes:

$$\hat{y} = \frac{\sum_{i=1}^N y^i \exp\left(-\frac{(\mathbf{x} - \mathbf{x}^i)^T(\mathbf{x} - \mathbf{x}^i)}{2\sigma^2}\right)}{\sum_{i=1}^N \exp\left(-\frac{(\mathbf{x} - \mathbf{x}^i)^T(\mathbf{x} - \mathbf{x}^i)}{2\sigma^2}\right)}, \quad (3.33)$$

where  $N$  is the number of sample observations and  $\mathbf{x}^i$  is the  $i$ th observation of the vector random variable  $\mathbf{X}$ .

The architecture of a typical GRNN consists of four successive layers of processing units, as shows Figure 3.4 reproduced from Specht (1991). The input layer is composed merely by distribution units, but it is usually necessary to scale all input variables such that they have approximately the same ranges or variances. This is necessary because

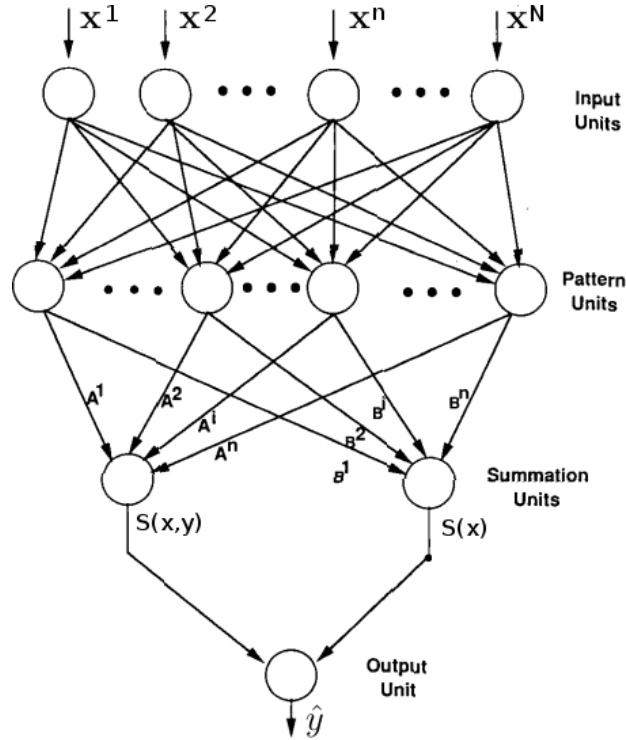


Figure 3.4: Architecture of the GRNN model

the underlying pdf is estimated with a kernel that has the same width in each dimension (SPECHT, 1991). A pattern neuron  $j$  receives the data vector  $\mathbf{x}$  from the input layer and computes an output  $p(\mathbf{x}|j)$  using (3.29), i.e., PNN and GRNN have the same kind of pattern neurons. The standard GRNN model also uses a separate neuron for each training data in the pattern layer.

The summation units perform a dot product between a weight vector and a vector composed of the signals from the pattern units. There are two types of summations performed in this layer: simple arithmetic summations  $s(\mathbf{x})$  and weighted summations  $s(\mathbf{x}, y)$ , which are computed, respectively, by the following equations:

$$s(\mathbf{x}) = \sum_{j=1}^M p(\mathbf{x}|j) \quad (3.34)$$

$$s(\mathbf{x}, y) = \sum_{j=1}^M C_j p(\mathbf{x}|j) \quad (3.35)$$

where  $C_j$  is a weight connecting the  $j$ th pattern neuron to the weighted summation unit. The sums calculated by the summation neurons are sent to the fourth layer, i.e., the output neuron, which merely divides  $s(\mathbf{x}, y)$  by  $s(\mathbf{x})$  to yield the desired estimate  $\hat{y}$ .

The learning process is accomplished by just setting each training pattern  $\mathbf{x}$  equal to the  $\mathbf{w}_j$  weight vector in one of the pattern units, connecting the pattern unit's output to the summation units and setting the weight  $C_j$  equal to the desired response  $y^i$  of the corresponding training pattern  $\mathbf{x}^i$  (SPECHT, 1991).

The main advantages of GRNN are: (i) the neural network learns instantaneously using a single pass over the training data; (ii) the estimate asymptotically converges to the optimal regression surface as more training data arrive; and (iii) it needs only a fraction of



the training samples a MLP neural network would need to converge. The main disadvantage of the standard GRNN is the amount of computation required to evaluate new points, which is proportional to the size of the training dataset.

To overcome this problem, Specht (1991) suggest to use clustering algorithms, like  $K$ -means averaging (TOU; GONZALEZ, 1974) or adaptive  $K$ -means (MOODY; DARKEN, 1989), to group samples so that the group can be represented by a single pattern unit. In this case, each weight vector  $\mathbf{w}_j$  represents the center of a cluster and (3.33) can be rewritten as

$$\hat{y} = \frac{\sum_{j=1}^M A^j \exp\left(-\frac{(\mathbf{x} - \mathbf{w}_j)^T(\mathbf{x} - \mathbf{w}_j)}{2\sigma^2}\right)}{\sum_{j=1}^M B^j \exp\left(-\frac{(\mathbf{x} - \mathbf{w}_j)^T(\mathbf{x} - \mathbf{w}_j)}{2\sigma^2}\right)}, \quad (3.36)$$

where  $M < N$  is the number of clusters,  $A^j$  is the sum of the  $y$  values and  $B^j$  is the number of samples assigned to cluster  $j$ .  $A^j$  and  $B^j$  are incremented each time a training observation  $y^i$  for the  $j$ th cluster is encountered, i.e.:

$$\begin{aligned} A^{j*} &= A^j + y^i \\ B^{j*} &= B^j + 1. \end{aligned} \quad (3.37)$$

In Specht (1991) other cluster techniques to set the pattern units are suggested, but those techniques: (i) are restricted to distance-based algorithms (e.g., the  $K$ -means and its variations); (ii) require that the number of clusters be previously known and fixed; and/or (iii) are based in ad-hoc choices like the definition of a radius of influence  $r$ . Another problem of the GRNN model is the configuration of the smoothing parameter  $\sigma$ , which as in PNN must be done off-line using iterative techniques like the *holdout method* (TAN; STEINBACH; KUMAR, 2006).

### 3.8 Improvements made over PNN and GRNN

As mentioned above, the standard PNN and GRNN architectures have several limitations which makes the computation very slow for large databases and/or prevents their use in on-line and continuous tasks. This section describes some state-of-the-art approaches to tackle these limitations. Most part of these approaches use some kind of clustering algorithm to group the input patterns, so that the group can be represented by a single unit in the pattern layer.

In Trávén (1991), a Gaussian clustering (GC) algorithm is proposed for substantially reducing the amount of information necessary to consider during classification. This algorithm uses radially symmetric Gaussian components whose parameters are estimated by a stochastic gradient descent procedure (WOLFE, 1970). In this procedure, the mean of each Gaussian distribution (i.e., cluster)  $j$  is updated through:

$$\boldsymbol{\mu}_j^* = \boldsymbol{\mu}_j + \eta_j(\mathbf{x}^t - \boldsymbol{\mu}_j), \quad (3.38)$$

where  $\mathbf{x}^t$  is the current input pattern and  $\eta_j$  is given by:

$$\eta_j = \frac{p(j|\mathbf{x}^t)}{Np(j)}, \quad (3.39)$$

where  $p(j|\mathbf{x}^t)$  is the posterior probability of  $j$  given  $\mathbf{x}^t$ ,  $N$  is the number of training patterns received until the current time  $t$  (i.e.,  $N \equiv t$ ) and  $p(j)$  is the *a priori* probability of  $j$ . GC assumes that the prior probabilities of all clusters are the same, i.e.,  $p(j) = 1/M$ , where  $M$  is the number of Gaussian distributions. It also assumes that the covariance matrices are in the form of a constant times the identity matrix, i.e.,  $\mathbf{C} = \sigma^2 \mathbf{I}$ . The update equation for the variance  $\sigma^2$  is:

$$\sigma_j^{2*} = \sigma_j^2 + \eta_j \left[ \frac{2}{D} (1 - \mathbf{x}^t \cdot \boldsymbol{\mu}_j^T) - \sigma_N^2 \right], \quad (3.40)$$

where the superscript ‘ $T$ ’ denotes the transpose and  $D$  is the dimensionality of  $\mathbf{x}$ . A peculiarity of this algorithm is that the parameters of the pattern units depend only on the input densities, which according to Tråvén (1991) “in some cases may result in somewhat less efficient solutions compared with methods where the placement of the hidden units can be supervised”. The main limitations of this algorithm are: (i) it uses only radially symmetric pdf’s; (ii) it assumes that all prior probabilities are equal; (iii) the number of clusters must be previously defined and fixed; and (iv) there are separate phases from training and recalling, that prevents its use in on-line and continuous tasks.

In Ćwik and Koronacki (1996), the Tråvén’s GC algorithm is extended so that no constraints are imposed on the covariance matrices, i.e., the updated covariance matrix  $\mathbf{C}_j^*$  of the  $j$ th Gaussian distribution is computed through:

$$\mathbf{C}_j^* = \mathbf{C}_j + \eta_j [(\mathbf{x}^t - \boldsymbol{\mu}_j)(\mathbf{x}^t - \boldsymbol{\mu}_j)^T - \mathbf{C}_j]. \quad (3.41)$$

where  $\eta_j$  is computed using (3.39) above. Unfortunately this algorithm still requires that the number of clusters must be previously known and fixed. Moreover, The Gaussian components are initialized using the off-line  $K$ -means algorithm (MACQUEEN, 1967).

In Montana (1992) a genetic algorithm (GOLDBERG, 1989; MITCHELL, 1996) is used to optimize the covariance matrices of the pattern units, thus allowing anisotropic Gaussians units, i.e. Gaussians whose covariance is not a multiple of the identity matrix, in the pattern layer. This optimization method, called weighted probabilistic neural network (WPNN), works as follows. For  $i$ th exemplar of class  $j$ ,  $\mathbf{x}_i^j$ , let  $\mathcal{L}_j(\mathbf{x}_i^j)$ , for  $j = 1, \dots, M$ , denote the class likelihoods obtained upon withholding this exemplar and applying  $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \mathbf{C}_j)$ , and let  $\tilde{P}_j(\mathbf{x}_i^j)$  be the probabilities obtained from these likelihoods through:

$$\tilde{P}_j(\mathbf{x}_i^j) = \mathcal{L}_j(\mathbf{x}_i^j) / \sum_{q=1}^M \mathcal{L}_q(\mathbf{x}_i^j) \quad (3.42)$$

Then, the fitness of the GA is defined as

$$E = \sum_{j=1}^M \sum_{i=1}^{N_j} ((1 - \tilde{P}_j(\mathbf{x}_i^j))^2 + \sum_{q \neq j} (\tilde{P}_q(\mathbf{x}_i^j))^2). \quad (3.43)$$

where  $N_j$  is the number of training samples of class  $j$ . In Mao et al. (2000) another GA-based algorithm to determine the structure of a PNN network is proposed. This algorithm consists of two parts and runs in an iterative way. The first part identifies an appropriate smoothing parameter using a genetic algorithm, while the second part determines suitable pattern layer neurons using a forward regression orthogonal algorithm (CHEN; COWAN; GRANT, 1991). The main disadvantage of these GA-based methods is that they require a large number of generations to evolve a good solution.

In Streit and Luginbuhl (1994), a maximum likelihood method for training probabilistic networks is proposed. This training method, called Generalized Fisher (GF) training, is an iterative procedure that computes stationary points of the posterior likelihood function  $\mathcal{L}$  without taking gradients or derivatives. It begins with an initial guess,  $\tilde{\theta}$ , for the optimum parameters, and each iteration gives a new parameter estimate,  $\theta^+$ , that is guaranteed to increase the value of the posterior likelihood function  $\mathcal{L}$  unless  $\tilde{\theta}$  is a stationary value of  $\mathcal{L}$ . By restarting the GF training algorithm with different initial guesses  $\tilde{\theta}$ , and choosing the best of the local maxima so obtained, a satisfactory maximum likelihood estimate for  $\theta$  can be found. GF is based on the premise that all classes are multivariate Gaussian random variables with a common covariance matrix  $\mathbf{C}$ , but different mean vectors  $\boldsymbol{\mu}_j$ . Initially the labeled training set  $\mathbf{X}$  is partitioned into the disjoint subsets  $\mathbf{X} = \mathbf{X}^1 \cup \mathbf{T}^2 \cup \dots \cup \mathbf{X}^M$ , where  $M$  is the number of classes and  $\mathbf{X}^j$  comprises those samples in  $\mathbf{X}$  with class label  $j$ . The sample means of the  $j$ th class are estimated through

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{x}^i \quad (3.44)$$

where  $\mathbf{x}^i$  is the  $i$ th training pattern of class  $j$  and  $N_j$  is the number of training samples in  $j$ . The common covariance matrix  $\mathbf{C}$  is estimated by Fisher's within-class scatter matrix:

$$\hat{\mathbf{C}} = \frac{1}{N} \sum_{j=1}^M \sum_{i=1}^{N_j} (\mathbf{x}^i - \hat{\boldsymbol{\mu}}_j)(\mathbf{x}^i - \hat{\boldsymbol{\mu}}_j)^T. \quad (3.45)$$

where  $N$  denotes the total number of samples of the training dataset. The estimation error for  $\mathbf{C}$  is reduced by pooling the sample data, i.e., by using all samples in the training set  $\mathbf{X}$ . The *a priori* probabilities are computed without iteration using  $p(j) = N_j/N$ . The main limitations of this method are: (i) all pattern units share the same covariance matrix, which works well only when "samples from different classes have broadly similar correlational structure" (STREIT; LUGINBUHL, 1994); (ii) it is an iterative procedure whose results depend strongly on the initial guess  $\tilde{\theta}$ ; (iii) the number of Gaussian units must be specified before the actual training can take place; and (iv) it is somewhat restricted to classification, because the training dataset must be partitioned into the disjoint subsets.

In Berthold and Diamond (1998), a learning algorithm that constructs the topology of a probabilistic neural network during training is proposed. This algorithm, called dynamic decay adjustment (DDA), has two configuration parameters, the thresholds  $\theta^+$  and  $\theta^-$ .  $\theta^+$  determines the minimum correct-classification probability for training patterns of the correct class, and  $\theta^-$  is used to avoid misclassifications. The operation requires two distinct phases: training and classification. The training phase works as follows. Initially all prototype weights  $A$  are set to zero, i.e.,  $A_i^k = 0, \forall p_i^k$ , where  $p_i^k$  is the  $i$ th prototype (candidate) neuron of the output class  $k$ . Next all training patterns are presented to the neural network. If a pattern  $(\mathbf{x}, k)$  is classified correctly, i.e.,  $\exists p_i^k : \mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_i^k, \sigma_i^k) \geq \theta^+$ , the weight of the prototype with the largest variance,  $A_i^k$ , is increased ( $A_i^k = A_i^k + 1.0$ ). Otherwise a new prototype is introduced ( $M^k = M^k + 1$ ) having the new pattern as its center ( $\boldsymbol{\mu}_{M^k}^k = \mathbf{x}$ ), a prototype weight  $A_{M^k}^k$  equal to 1 ( $A_{M^k}^k = 1$ ), and its initial radius  $\sigma_{M^k}^k$  is set as large as possible without misclassifying an already existing prototype of conflicting class, i.e.:

$$\sigma_{M^k}^k = \min_{\substack{l \neq k \\ 1 \leq j \leq M^l}} \left\{ \sqrt{-\frac{\|\boldsymbol{\mu}_j^l - \boldsymbol{\mu}_{M^k}^k\|^2}{\ln \theta^-}} \right\}. \quad (3.46)$$

After this all the prototypes  $1 \leq j \leq M^l$  of the conflicting classes  $l \neq k$  are shrunk if their activations are too high for this specific pattern, i.e.:

$$\sigma_j^l = \min \left\{ \sigma_j^l, \sqrt{-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j^l\|^2}{\ln \theta^-}} \right\}, \forall j \in M^k. \quad (3.47)$$

The learning process is repeated for some epochs until the network architecture settles, thus indicating the completion of the training phase. After training is complete, the normalized output weights  $w_j^k$  are computed for each class  $k$  from the prototype weights  $A_j^k$  through:

$$w_j^k = \frac{A_j^k}{\sum_{q=1}^{M^k} A_q^k}, \quad \forall j \in M^k. \quad (3.48)$$

The main limitations of this algorithm are: (i) it is a batch-mode algorithm which requires that the entire training dataset must be previously known and fixed; (ii) there are distinct phases for training and classification; (iii) it uses just radially symmetric Gaussian kernels; and (iv) the center  $\boldsymbol{\mu}$  of a pattern unit never changes after its introduction, i.e., just the standard deviation  $\sigma$  of each distribution is adjusted during learning.

Delgosha and Menhaj (2001) propose a modified PNN learning algorithm, called Fuzzy PNN (FPNN), which aims to improve the reliability of classification when the output classes overlap. This is accomplished by making a soft decision in the training phase instead of the traditional hard decision made by traditional classifiers. In the soft decision, the effect of overlapping portions is lessened, because “the classifier is intentionally made blind with respect to overlapping sections” (DELGOSHA; MENHAJ, 2001). The soft decision is performed by the following smooth penalty function:

$$\rho(m; b, s) = \begin{cases} 0, & m < 0 \\ 1/(1 + e^{-(m-b)/s}), & m \geq 0 \end{cases} \quad (3.49)$$

where  $b$  is a bias introduced for more robustness,  $s$  is the softness parameter and  $m = l_j - l_q$  is the difference between the losses of a class  $j$  ( $l_j$ ) and the class  $q \neq j$  with the minimum risk  $l_q$  among all classes. The training steps of FPNN are completely similar to those of PNN except the last step: FPNN counts the total number of misclassified sample vectors and repeats the training procedure until all the sample vectors are correctly classified.

There are other improvements made over probabilistic networks to adapt them to specific situations. Rutkowski, for instance, derives a new mathematical formulation for GRNN (RUTKOWSKI, 2004b) and PNN (RUTKOWSKI, 2004a) in time-varying environments, but his formulation has the same drawbacks of the Specht’s models described above (e.g., a separate neuron is required for every training pattern). Polat and Yildirim (2010) describe a hardware implementation of a general regression neural network using field programmable gate array (FPGA) circuits. Chang et al. (2009) present a PNN model which has data imputation capabilities for machine-fault classification. This model uses the standard EM algorithm to train the neural network and a global k-means algorithm is used prior to EM to find the number of clusters based on minimizing the clustering error.

We can conclude that, although many algorithms have been proposed to tackle the main restrictions of PNN and GRNN, most of them have at least one of the following limitations: (i) they are restricted to classification tasks; (ii) they require that the complete

training dataset is previously known and fixed; (iii) the learning process requires several scans over the entire dataset; (iv) they impose limitations on the form of the pattern units; (v) they have separate phases for training and recalling; or (vi) the neural network topology must be defined a priori and kept fixed. Therefore, these algorithms are not useful for applications such as incremental function approximation, on-line prediction and mobile robotics.

### 3.9 Other neural network models

This section presents other neural network models which are less known and used than the traditional models (e.g., MLP, RBF, SOM) described in the previous sections. In Lin et al. (1997), an ANN model called probabilistic decision-based neural network (PDBNN) is proposed. PDBNN has a modular architecture composed by several subnets connected to a decision unit. Learning in PDBNN is divided into two phases: locally unsupervised and globally supervised. In the locally unsupervised learning phase, PDBNN adopts the EM algorithm (DEMPSTER; LAIRD; RUBIN, 1977) to maximize the likelihood function

$$\mathcal{L}(\mathbf{X}) = \sum_{t=1}^N \log \left[ \sum_{j=1}^M p(k) p(\mathbf{x}^t | k, \boldsymbol{\theta}_j) \right] \quad (3.50)$$

where  $k$  is the output class,  $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^N\}$  denotes the set of  $N$  independent and identically distributed training patterns,  $\boldsymbol{\theta}_j = \{\boldsymbol{\mu}_j, \mathbf{C}_j, p(k)\}$  represents the parameters of the  $j$ th mixture component,  $M$  is the number of mixture components and  $p(\mathbf{x}^t | k, \boldsymbol{\theta}_j)$  is the class likelihood function computed using a  $D$ -variate Gaussian distribution. In the globally supervised training phase, target values are used to fine-tune the decision boundaries. Specifically, when a training pattern is misclassified to the  $j$ th class, reinforced and/or anti-reinforced learning (SUTTON; BARTO, 1998) are applied to update the mean vectors and covariance matrices of subnet  $j$ . The main drawbacks of this algorithm are the same of great part of the ANN models, i.e.: (i) the learning process requires several scans over the entire dataset; and (ii) there are separate phases for training and recalling.

In Williamson (1997) a probabilistic version of Fuzzy ARTMAP, called Gaussian ARTMAP, is proposed for mixture modeling and classification. Gaussian ARTMAP differs from Fuzzy ARTMAP significantly in a few respects. First, it represents category  $j$  as a Gaussian density function, defined by two vectors: its mean  $\boldsymbol{\mu}_j = \{\mu_{j1}, \mu_{j2}, \dots, \mu_{jM}\}$  and its standard deviation  $\boldsymbol{\sigma}_j = \{\sigma_{j1}, \sigma_{j2}, \dots, \sigma_{jM}\}$ , which together replace prototype vector  $\mathbf{w}_j$ . Second, a scalar,  $n_j$ , accumulates the amount of relative activation obtained by  $F_2$  node  $j$  on training set patterns. During training, the number of committed  $F_2$  nodes,  $N_c$ , is initially set to 0. Newly-committed  $F_2$  nodes increment  $N_c$ , and undergo the initialization step:

$$\boldsymbol{\mu}_J = \mathbf{A}; \quad \boldsymbol{\sigma}_{Ji} = \gamma \quad w_{JK}^{ab} = 1; \quad n_J = 1,$$

where  $\mathbf{A}$  represents the input pattern,  $w_{JK}^{ab}$  is the  $F^{ab}$  prototype vector connecting nodes  $J$  and  $K$  (the inter-ART module still has a prototype vector  $\mathbf{w}$ ) and  $\gamma$  is the initial standard deviation assigned to newly-committed  $F_2$  nodes. Committed  $F_2$  nodes that pass the following vigilance test for pattern  $\mathbf{a}$ :

$$G_j(\mathbf{A}) = \exp \left\{ -\frac{1}{2} \sum_{i=1}^M \frac{(A_i - \mu_{ji})^2}{\sigma_{ji}^2} \right\} > \rho, \quad (3.51)$$

where  $\rho$  are the choice is the baseline vigilance parameter, are allowed to activate, and distribute a pattern of activity  $\mathbf{y}$ , i.e.:

$$y_j = \frac{g_j}{0.01 + \sum_{l=1}^{N_c} g_l}. \quad (3.52)$$

Match tracking and learning are performed according to the relative activation over the “ensemble”  $E_K$  of  $F_2$  nodes linked to the predicted  $F^{ab}$  node  $K$ , i.e.:

$$\rho' = \exp \left\{ -\frac{1}{2} \sum_{j \in E_K} y_j^* \sum_{i=1}^M \frac{(A_i - \mu_{ji})^2}{\sigma_{ji}^2} \right\} + \varepsilon. \quad (3.53)$$

The relative activation over  $E_K$  is defined by the distributed pattern  $\mathbf{y}^*$ , where  $y_j^* = y_j / \sum_{l \in E_K} y_l$  only if  $j \in E_K$ , and  $y_j^* = 0$  otherwise. The prototype update is computed through the following equations:

$$n'_j = n_j + y_j^* \quad (3.54)$$

$$\mu'_{ji} = \left( 1 - \frac{y_j^*}{n_j} \right) \mu_{ji} + \frac{y_j^*}{n_j} A_i \quad (3.55)$$

$$\sigma'_{ji} = \sqrt{\left( 1 - \frac{y_j^*}{n_j} \right) \sigma_{ji}^2 + \frac{y_j^*}{n_j} (A_i - \mu_{ji})^2}. \quad (3.56)$$

Although the Gaussian ARTMAP can achieve better results than Fuzzy ARTMAP in incremental classification tasks (GRANGER; CONNOLLY; SABOURIN, 2008), it has many critical parameters (the “standard” Fuzzy ARTMAP parameters plus  $\varepsilon$ ,  $\rho$ ,  $\gamma$ ), which must be properly configured before the learning process starts. Moreover, it uses diagonal covariance matrices in the Gaussian units, i.e., just the variance  $\sigma^2$  is computed.

Other interesting neural network models are: (i) the recurrent log-linearized Gaussian mixture network (R-LLGMN), proposed by Tsuji et al. (2003), which uses Gaussian mixtures and hidden Markov models (HMM) for classifying temporal series. But unfortunately this model is not on-line nor incremental, just recurrent; and (ii) the resource allocating network (RAN), proposed by Platt (1991), which adds new neurons incrementally by choosing what training vectors it must store. RAN is based on a distance based algorithm, and like  $K$ -means the induced model is equivalent to a set of equiprobable spherical distributions sharing the same variance. Moreover, the training algorithm uses the same (slow) least mean squares (LMS) algorithm (WIDROW; HOFF, 1960) used by Backpropagation (RUMELHART; HINTON; WILLIAMS, 1986).

### 3.10 Comparison among the described ANN models

Table 3.1 summarizes the main characteristics of the neural network models presented in this chapter. These characteristics refer to the standard architectures described in the corresponding papers, but of course some of these models have variations which improves some of these characteristics and/or extends them for using in other tasks.

The first (*neural network*) and second (*main reference*) columns of Table 3.1 present, respectively, the acronyms of the neural network models and the corresponding references. The third column (*main tasks*) describes the tasks in which the ANN model can be used (classification, regression or clustering). The fourth column (*incr. arch.*) indicates if the ANN model has an incremental architecture: “YES” means that the neural network

Table 3.1: Characteristics of the described neural network models

Neural network	Main reference	Main tasks	Incr. arch.	Cont. learn.	Inst. learn.	Init. dep.	Prob. mod.	Covar. matrix	Many units	Many conf.
MLP	(RUMELHART et al., 1986)	Class., regr.	no	no	no	yes	no	-	NO	yes
Cascade Correlat.	(FAHLMAN; LEBIERE, 1990)	Class.	YES	YES	no	NO	no	-	NO	NO
RBF	(POWELL, 1985)	Regr.	no	no	no	yes	no	Unique $\sigma$	NO	yes
SOM	(KOHONEN, 1990)	Clust.	no	no	no	yes	no	-	NO	yes
GTSOM	(BASTOS, 2007)	Clust.	YES	YES	no	NO	no	-	NO	yes
ART	(CARPENTER et al., 1987)	Clust.	YES	YES	YES	yes	no	-	NO	yes
ARTMAP	(CARPENTER et al., 1991)	Regr., clust.	YES	YES	YES	yes	no	-	NO	yes
PNN	(SPECHT, 1988, 1990)	Class.	YES	YES	YES	NO	YES	Unique $\sigma$	yes	NO
GRNN	(SPECHT, 1991)	Regr.	YES	YES	YES	NO	YES	Unique $\sigma$	yes	NO
Traven's GC	(TRÁVÉN, 1991)	Class., regr.	no	no	no	yes	YES	Radially	NO	NO
Cwik's GC	(CWIK; KORONACKI, 1996)	Class., regr.	no	no	no	yes	YES	FULL C	NO	NO
WPNN	(MONTANA, 1992)	Class.	YES	no	no	yes	YES	FULL C	NO	NO
Mao's PNN	(MAO; TAN; SER, 2000)	Class.	no	no	no	NO	YES	Unique $\sigma$	NO	NO
GF	(STREIT; LUGINBUHL, 1994)	Class.	YES	no	no	yes	YES	Unique C	NO	NO
DDA	(BERTHOLD; DIAMOND, 1998)	Class.	no	no	no	NO	YES	Radially	NO	NO
FPNN	(DELGOSHA; MENHAJ, 2001)	Class.	YES	no	no	NO	YES	Unique C	NO	NO
Rutkowski's GRNN	(RUTKOWSKI, 2004a)	Regr.	YES	YES	YES	NO	YES	Unique $\sigma$	yes	NO
Rutkowski's PNN	(RUTKOWSKI, 2004b)	Class.	YES	YES	YES	NO	YES	Unique $\sigma$	yes	NO
FPGA GRNN	(POLAT; YILDIRIM, 2010)	Regr.	YES	YES	YES	NO	YES	Unique $\sigma$	yes	NO
Chang's PNN	(CHANG; LOO; RAO, 2009)	Class.	no	no	no	NO	YES	FULL C	NO	NO
PDBNN	(LIN; KUNG; LIN, 1997)	Class.	YES	no	no	yes	YES	FULL C	NO	yes
Gaussian ARTMAP	(WILLIAMSON, 1997)	Regr., clust.	YES	YES	YES	NO	YES	Diagonal	NO	yes
R-LLGMN	(TSUJI et al., 2003)	Class.	YES	no	no	yes	YES	FULL C	NO	yes
RAN	(PLATT, 1991)	Regr.	YES	YES	YES	NO	no	-	NO	yes
<b>IGMN</b>	<b>(THIS THESIS)</b>	Regr., clust.	YES	YES	YES	NO	YES	FULL C	NO	NO

incrementally adds neurons whenever necessary and “no” means that it uses a predefined and fixed architecture. The fifth column (*cont. learn.*) indicates if the neural network can learn continuously without separate phases for training and using. The sixth column (*inst. learn.*) indicates if the model can learn instantaneously with a single presentation of each training pattern (“YES”) or it requires several scans over the entire training database (i.e., epochs) to converge (“no”). The seventh column (*init. dep.*) indicates if the results obtained using the corresponding model varies according to the initial conditions, e.g., if the random initialization of the model parameters affects significantly the obtained results (which occurs, for instance, when the ANN is sensible to local minima). The eighth column informs if the neural network follows a probabilistic framework or not. In this thesis we are mainly interested in probabilistic models, because they allow better results in some kind of tasks likes robotics (THRUN; BURGARD; FOX, 2006). The Ninth column (*covar. matrix*) indicates the kind of covariance matrix used by each probabilistic model (the hyphen symbol indicates that the corresponding model does not use covariance matrices) according to following description:

- **Unique  $\sigma$ :** all Gaussian units are radially symmetric and share the same standard deviation  $\sigma$ ;
- **Radially:** each Gaussian unit  $j$  has its own standard deviation parameter  $\sigma_j$ , but these units are still radially symmetric;
- **Diagonal:** the covariance matrices are diagonal, i.e., the input features are considered conditionally independent (that corresponds to the *naïve* Bayes hypothesis);
- **Unique C:** all Gaussian units share the same (full) covariance matrix  $C$ ;
- **FULL C:** each Gaussian unit  $j$  has its own (full) covariance matrix  $C_j$ ; i.e., no restrictions are imposed to the shape of the Gaussian distributions.

The tenth column (*many units*) indicates if the neural network requires a separate neuron for every training pattern, such as in Specht’s PNN and GRNN. The last column (*many*

*conf.*) indicates if the ANN has many (critical) configuration parameters which must be properly configure before learning.

Observing Table 3.1 it can be noticed that neither of these previous approaches satisfy all requirements (written in uppercase in the respective columns) for using them successfully in incremental function approximation and on-line tasks. The neural network model proposed in this thesis (last row in Table 3.1), on the other hand, satisfies all these requirements, and thus is a suitable tool to be used in this kind of application. Next chapter describes the proposed neural network model, called IGMN, in details.



## 4 INCREMENTAL GAUSSIAN MIXTURE NETWORK

This chapter presents the neural network model proposed in this thesis, called IGMN<sup>1</sup> (standing for Incremental Gaussian Mixture Network) (HEINEN; ENGEL, 2010a,b), that is the main contribution of this thesis. It is based on parametric probabilistic models (Gaussian mixture models), that have nice features from the representational point of view, describing noisy environments in a very parsimonious way, with parameters that are readily understandable. IGMN can be seen as a supervised learning extension of the IGMM algorithm, presented in Section 2.5, but it has unique features from the statistical point of view that endow it with the ability to make on-line predictions for *forward* and *inverse* problems and also to compute the confidence estimates of its predictions, as will be shown throughout this thesis.

This chapter is structured as follows. Section 4.1 provides the mathematical derivation of the regression algorithm used by IGMN to approximate functions and make predictions. Section 4.2 presents the neural network architecture of IGMN. Section 4.3 describes the operation of IGMN during learning and recalling. Section 4.4 extends the proposed ANN model for multi-valued inverse problems which are not characterized by a *functional* (i.e. single-valued) mapping. Section 4.5 discusses the IGMN configuration parameters and how they can be set. Finally, Section 4.6 concludes this chapter summarizing the main characteristics and limitations of IGMN.

### 4.1 General regression using Gaussian mixture models

This section describes the statistical basis of IGMN and the mathematical derivation of the general regression algorithm which enables IGMN to approximate functions and make predictions. Suppose that we have a noisy training data set  $\mathbf{z}$  composed by two data vectors,  $\mathbf{a} \in \mathcal{A}$  and  $\mathbf{b} \in \mathcal{B}$ , that correspond to the observed (i.e., the independent variables) and missing (i.e., the target values) stimuli, respectively. According to Bishop (1995), the goal of learning is to find a smooth mapping from  $\mathbf{a}$  to  $\mathbf{b}$  which captures the underlying systematic aspects of the data, without fitting the noise on the data. In Bishop (1995) it is shown that, under many circumstances, the optimal mapping is given by forming the regression, or conditional average  $\langle \mathbf{b} | \mathbf{a} \rangle$ , of the target data  $\mathbf{b}$ , conditioned on the input variables. This can be expressed in terms of the conditional density  $p(\mathbf{b} | \mathbf{a})$ , and hence in terms of the joint density  $p(\mathbf{a}, \mathbf{b})$ , as follows:

$$\hat{\mathbf{b}} = \langle \mathbf{b} | \mathbf{a} \rangle$$

---

<sup>1</sup>Initially IGMN was called IPNN (standing for Incremental Probabilistic Neural Network), but the name was changed to avoid misunderstandings with the Specht's PNN model, which is based on nonparametric Parzen's window estimators (PARZEN, 1962) rather than Gaussian mixture models.

$$\begin{aligned}
&= \int \mathbf{b} p(\mathbf{b}|\mathbf{a}) d\mathbf{b} \\
&= \frac{\int \mathbf{b} p(\mathbf{a}, \mathbf{b}) d\mathbf{b}}{\int p(\mathbf{a}, \mathbf{b}) d\mathbf{b}}.
\end{aligned} \tag{4.1}$$

The probability density  $p(\mathbf{a}, \mathbf{b})$  can be modeled using Parzen kernel estimators (PARZEN, 1962), for instance, as occurs in GRNN (SPECHT, 1991). In this case, the density  $p(\mathbf{a}, \mathbf{b})$  is modeled using a Gaussian kernel function in the form:

$$\hat{p}(\mathbf{a}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi\sigma^2)^{(D^a+D^b)/2}} \exp \left\{ -\frac{\|\mathbf{a} - \mathbf{a}^n\|^2}{2\sigma^2} - \frac{\|\mathbf{b} - \mathbf{b}^n\|^2}{2\sigma^2} \right\}, \tag{4.2}$$

where  $D^a$  and  $D^b$  are the dimensions of  $\mathbf{a}$  and  $\mathbf{b}$ , respectively. Substituting (4.2) into (4.1) we obtain the following expression for the regression of  $\hat{\mathbf{b}}$  (BISHOP, 1995):

$$\hat{\mathbf{b}} = \frac{\sum_{n=1}^N \mathbf{b}^n \exp\{-\|\mathbf{a} - \mathbf{a}^n\|^2/2\sigma^2\}}{\sum_{n=1}^N \exp\{-\|\mathbf{a} - \mathbf{a}^n\|^2/2\sigma^2\}}, \tag{4.3}$$

which is known as the Nadaraya-Watson estimator (NADARAYA, 1964; WATSON, 1964), used by GRNN (SPECHT, 1991). If we replace the kernel estimator by a Gaussian mixture model with diagonal matrices, the joint density function becomes:

$$\hat{p}(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^M p(j) \frac{1}{(2\pi\sigma_j^{a2})^{D^a/2} (2\pi\sigma_j^{b2})^{D^b/2}} \exp \left\{ -\frac{\|\mathbf{a} - \boldsymbol{\mu}_j^a\|^2}{2\sigma_j^{a2}} - \frac{\|\mathbf{b} - \boldsymbol{\mu}_j^b\|^2}{2\sigma_j^{b2}} \right\}, \tag{4.4}$$

where  $\boldsymbol{\mu}_j^a$  and  $\boldsymbol{\mu}_j^b$  are the mean vectors of  $\mathbf{a}$  and  $\mathbf{b}$ , respectively,  $\sigma_j^{a2}$  is the variance of  $\mathbf{a}$  and  $\sigma_j^{b2}$  is the variance of  $\mathbf{b}$ . Replacing (4.4) into (4.1) and performing the indicated integrations, we obtain the following expression for the regression:

$$\hat{\mathbf{b}} = \frac{\sum_{j=1}^M p(j) \boldsymbol{\mu}_j^b \exp\{-\|\mathbf{a} - \boldsymbol{\mu}_j^a\|^2/2\sigma_j^{a2}\}}{\sum_{j=1}^M p(j) \exp\{-\|\mathbf{a} - \boldsymbol{\mu}_j^a\|^2/2\sigma_j^{a2}\}}, \tag{4.5}$$

which can be viewed as a normalized GRNN expansion in which the pattern units are no longer constrained to be symmetrically circular nor to coincide with the data points and in which the number of pattern units is typically smaller than the number of data points. Moreover, we can recognize the terms  $\exp\{-\|\mathbf{a} - \boldsymbol{\mu}_j^a\|^2/2\sigma_j^{a2}\}$  in (4.5) as univariate Gaussian distributions:

$$p(\mathbf{a}|j) = \frac{1}{\sqrt{2\pi\sigma_j^{a2}}} \exp \left\{ -\frac{\|\mathbf{a} - \boldsymbol{\mu}_j^a\|^2}{2\sigma_j^{a2}} \right\}, \tag{4.6}$$

but without the normalizing terms  $(2\pi\sigma_j^{a2})^{-1/2}$ , which are unnecessary because the denominator in (4.5) already normalizes this expression. Replacing (4.6) into (4.5) we obtain:

$$\hat{\mathbf{b}} = \frac{\sum_{j=1}^M \boldsymbol{\mu}_j^b p(j) p(\mathbf{a}|j)}{\sum_{j=1}^M p(j) p(\mathbf{a}|j)}. \tag{4.7}$$

Using some algebraic manipulation we can rewrite this expression as:

$$\hat{\mathbf{b}} = \sum_{j=1}^M \left[ \frac{p(j)p(\mathbf{a}|j)}{\sum_{q=1}^M p(q)p(\mathbf{a}|q)} \right] \boldsymbol{\mu}_j^{\mathcal{B}}, \quad (4.8)$$

where we identify the term between brackets as the a posteriori probability  $p(j|\mathbf{a})$  computed using the Bayes' rule (2.37). Hence (4.8) can be rewritten as

$$\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{a}) \boldsymbol{\mu}_j^{\mathcal{B}}, \quad (4.9)$$

where  $\hat{\mathbf{b}}$  is estimated using the a posteriori probabilities  $p(j|\mathbf{a})$  computed over  $\mathbf{a}$  only and the mean vector of  $\mathbf{b}$ ,  $\boldsymbol{\mu}_j^{\mathcal{B}}$ . This equation results in a soft interpolation procedure among the Gaussian centers weighted by the corresponding a posteriori probabilities. Extending this result to multivariate Gaussian mixture models, (4.9) becomes (GHAHRAMANI; JORDAN, 1994a,b):

$$\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{a}) [\boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})], \quad (4.10)$$

where  $\mathbf{C}_j^{\mathcal{A}\mathcal{A}}$  is a submatrix containing just the rows and columns corresponding to  $\mathcal{A}$  in  $\mathbf{C}_j$  and  $\mathbf{C}_j^{\mathcal{B}\mathcal{A}}$  is a submatrix containing the rows corresponding to  $\mathcal{B}$  and the columns corresponding to  $\mathcal{A}$  in  $\mathbf{C}_j$ , i.e.:

$$\mathbf{C}_j = \left( \begin{array}{c|c} \mathbf{C}_j^{\mathcal{A}\mathcal{A}} & \mathbf{C}_j^{\mathcal{A}\mathcal{B}} \\ \hline \mathbf{C}_j^{\mathcal{B}\mathcal{A}} & \mathbf{C}_j^{\mathcal{B}\mathcal{B}} \end{array} \right) \quad \text{and} \quad \boldsymbol{\mu}_j = \begin{pmatrix} \boldsymbol{\mu}_j^{\mathcal{A}} \\ \boldsymbol{\mu}_j^{\mathcal{B}} \end{pmatrix}.$$

If the covariance matrices are constrained to be diagonal, then (4.10) simplifies to (4.9). It is important to notice that the approximation level obtained using (4.10) is higher than that obtained using (4.9), and as will be shown in the next chapter this is one of the main reasons why IGMN has a superior performance than other ANN models (e.g., GRNN) that use (4.9). In a previous version of IGMN, presented in Heinen and Engel (2010a; 2010b), just (4.9) has been used to estimate  $\hat{\mathbf{b}}$ , which resulted in a rougher approximation than that obtained using (4.10). To avoid confusion, in this thesis we will call this previous IGMN version as *standard* IGMN, and the new IGMN version proposed in this thesis will be called *multivariate* IGMN.

Following the same line of argument as before, we can likewise evaluate the variance of  $\hat{\mathbf{b}}$  using the following expression (BISHOP, 1995):

$$\begin{aligned} \hat{\boldsymbol{\sigma}}_{\mathcal{B}}^2 &= \langle \|\mathbf{b} - \langle \mathbf{b}|\mathbf{a} \rangle\|^2 | \mathbf{a} \rangle \\ &= \sum_{j=1}^M p(j|\mathbf{a}) \left\{ \boldsymbol{\sigma}_j^{\mathcal{B}} + \|\boldsymbol{\mu}_j^{\mathcal{B}} - \hat{\mathbf{b}}\|^2 \right\}. \end{aligned} \quad (4.11)$$

Extending this result to the multivariate case, we can estimate the covariance matrix  $\hat{\mathbf{C}}^{\mathcal{B}}$  using the following expression :

$$\hat{\mathbf{C}}^{\mathcal{B}} = \sum_{j=1}^M p(j|\mathbf{a}) \left\{ \mathbf{C}_j^{\mathcal{B}\mathcal{B}} - \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}\mathcal{A}-1} \mathbf{C}_j^{\mathcal{B}\mathcal{A}T} + \|\bar{\mathbf{x}}_j^{\mathcal{B}} - \hat{\mathbf{b}}\|^2 \right\}, \quad (4.12)$$

where the term  $\mathbf{C}_j^{\mathcal{B}\mathcal{A}}\mathbf{C}_j^{\mathcal{A}\mathcal{A}^{-1}}\mathbf{C}_j^{\mathcal{B}\mathcal{A}T}$  is added to (4.11) to extend it to general covariance matrices (GHAHRAMANI; JORDAN, 1994a,b) and  $\bar{\mathbf{x}}_j^{\mathcal{B}}$  corresponds to the term between brackets in (4.10), i.e.:

$$\bar{\mathbf{x}}_j^{\mathcal{B}} = \boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}}\mathbf{C}_j^{\mathcal{A}\mathcal{A}^{-1}}(\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}}). \quad (4.13)$$

If the covariance matrices are constrained to be diagonal, then (4.12) simplifies to (4.11). Next sections describe how these equations are used by IGMN for approximating functions and make predictions.

## 4.2 IGMN architecture

This section describes the general architecture and topology of IGMN. As said before in Section 1.2, most neural network models (e.g. MLP, RBF and GRNN) are based on the Cybernetic paradigm, and therefore the information flow is unidirectional from the input to the output layer<sup>2</sup>. In some ANN models (JORDAN, 1986; ELMAN, 1990) there are recurrent connections, i.e., the activations at time  $t$  are sent back to the hidden layer at time  $t + 1$ . But these recurrent models still follow the *information processing metaphor*, because the input is received in the first layer, it is processed in the hidden layer(s) and the results are obtained in the output layer (i.e., the information flow is unidirectional and the loop is still open) (PFEIFER; SCHEIER, 1994).

The neural network model proposed in this thesis, on the other hand, is based on IGMM (Section 2.5) and inspired on more recent theories about the brain (Section 1.2), specially the Memory-Prediction Framework (MPF) (HAWKINS, 2005) and the constructivist AI (DRESCHER, 1991). Therefore, in IGMN it is not quite correct to use the words *input* and *output* to represent the data features of a training sample such as  $\mathbf{z} = \{\mathbf{a}, \mathbf{b}\}$ , for instance. Instead, we consider that the data vectors  $\mathbf{a}$  and  $\mathbf{b}$  are different sensory and/or motor modalities with distinct domains, and one modality (e.g.  $\mathbf{a}$ ) can be used to estimate another (e.g.  $\hat{\mathbf{b}}$ ). Example:  $\mathbf{a}$  can be a sensor reading received at time  $t$  and used to estimate the best action  $\mathbf{b}$  at time  $t + 1$ . An important feature of IGMN is that the information flow is not unidirectional, i.e.,  $\mathbf{a}$  can be used to estimate  $\hat{\mathbf{b}}$ ,  $\mathbf{b}$  can be used to estimate  $\hat{\mathbf{a}}$  and both can be used to compute the *joint* a posteriori probability  $p(j|\mathbf{z})$ .

Figure 4.1 shows the IGMN architecture. It is composed by an association region  $\mathcal{P}$  (in the top of this figure) and many cortical regions,  $\mathcal{N}^{\mathcal{A}}, \mathcal{N}^{\mathcal{B}}, \dots, \mathcal{N}^{\mathcal{S}}$ . All regions have the same number of neurons,  $M$ . Initially there is a single neuron in each region (i.e.,  $M = 1$ ), but more neurons are incrementally added when necessary using an error driven mechanism. Each cortical region  $\mathcal{N}^{\mathcal{X}}$  receives signals from the  $k$ th sensory/motor modality,  $\mathbf{k}$  (in IGMN there is no difference between sensory and motor modalities), and hence there is a cortical region for each sensory/motor modality.

Another important feature of IGMN is that all cortical regions  $\mathcal{N}$  execute a common function, i.e., they have the same kind of neurons and use the same learning algorithm. Moreover, all cortical regions can run in parallel, which improves the performance specially in parallel architectures. Each neuron  $j$  of region  $\mathcal{N}^{\mathcal{X}}$  performs the following operation:

$$p(\mathbf{k}|j) = \mathcal{N}(\mathbf{k}|\boldsymbol{\mu}_j^{\mathcal{X}}, \mathbf{C}_j^{\mathcal{X}}) = \frac{1}{(2\pi)^{D^{\mathcal{X}}/2} \sqrt{|\mathbf{C}_j^{\mathcal{X}}|}} \exp \left\{ -\frac{1}{2}(\mathbf{k} - \boldsymbol{\mu}_j^{\mathcal{X}})^T \mathbf{C}_j^{\mathcal{X}^{-1}} (\mathbf{k} - \boldsymbol{\mu}_j^{\mathcal{X}}) \right\}, \quad (4.14)$$

<sup>2</sup>Important exceptions are the ART models (Sections 3.4 and 3.5), which are based on the Grossberg's adaptive resonance theory (GROSSBERG, 1976a,b) instead of the Cybernetic paradigm.

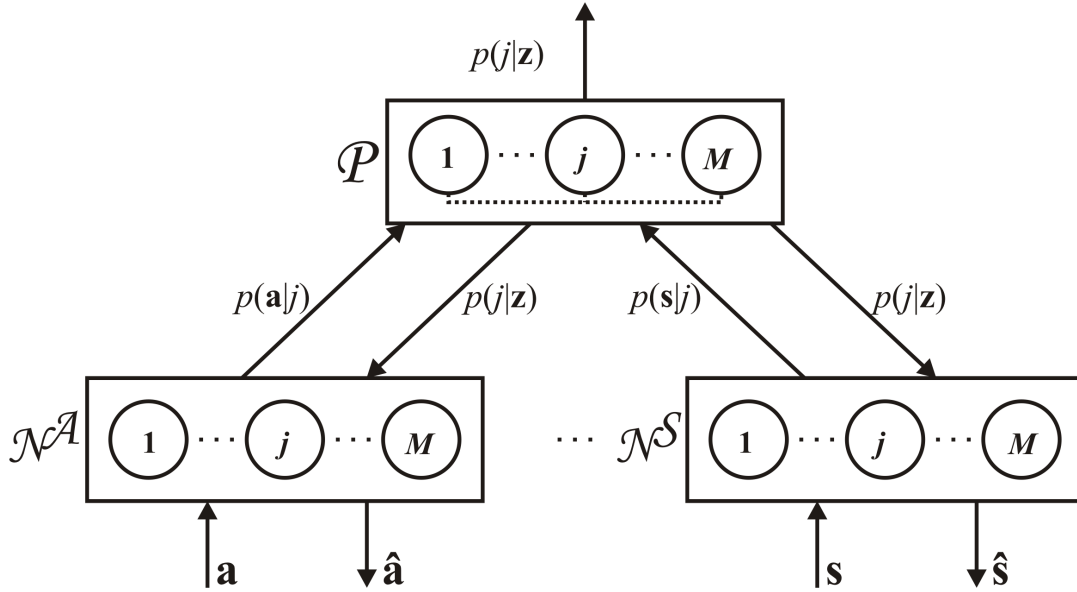


Figure 4.1: IGMN architecture

i.e., a multivariate Gaussian distribution, where  $D^{\mathbf{k}}$  is the dimensionality of  $\mathbf{k}$  (different sensory/motor modalities  $\mathbf{k}$  can have different dimensions  $D^{\mathbf{k}}$ ). Each neuron  $j$  maintains a mean vector  $\boldsymbol{\mu}_j^{\mathbf{k}}$  and a covariance matrix  $\mathbf{C}_j^{\mathbf{k}}$ .

In IGMN the regions are not fully connected, i.e., the neuron  $j$  of  $\mathcal{N}^{\mathbf{x}}$  is connected just to the  $j$ th neuron of  $\mathcal{P}$ , but this connection is bidirectional. It is important to notice that there are no *synaptic weights* in these connections, i.e., all IGMN parameters are stored in the neurons themselves. A bottom-up connection between  $\mathcal{N}^{\mathbf{x}}$  and  $\mathcal{P}$  provides the component density function  $p(\mathbf{k}|j)$  to the  $j$ th neuron in  $\mathcal{P}$ . Therefore, a neuron  $j$  in the association region  $\mathcal{P}$  is connected with the  $j$ th neuron of all cortical regions  $\mathcal{N}$  via bottom-up connections and computes the a posteriori probability of  $j$  using the Bayes' rule:

$$p(j|\mathbf{z}) = \frac{p(\mathbf{a}|j) p(\mathbf{b}|j) \dots p(\mathbf{s}|j) p(j)}{\sum_{q=1}^M p(\mathbf{a}|q) p(\mathbf{b}|q) \dots p(\mathbf{s}|q) p(q)}, \quad (4.15)$$

where it is considered that the neural network has an arbitrary number,  $s$ , of cortical regions and  $\mathbf{z} = \{\mathbf{a}, \mathbf{b}, \dots, \mathbf{s}\}$ . The dotted lines in Figure 4.1 above indicate the lateral interaction among the association units in order to compute the denominator in (4.15).

Each neuron  $j$  of the association region  $\mathcal{P}$  maintains its own age,  $v_j$ , the a priori probability,  $p(j)$ , an accumulator of the a posteriori probabilities,  $sp_j$ , and an association matrix to store the correlations among each sensory/motor modality. If a neural network has two cortical regions,  $\mathcal{N}^{\mathbf{a}}$  and  $\mathcal{N}^{\mathbf{b}}$ , for instance, then the association matrix  $\mathbf{C}_j^{\mathbf{a}\mathbf{b}}$  will have two dimensions and size  $D^{\mathbf{a}} \times D^{\mathbf{b}}$ . Note that it is not necessary to maintain  $\mathbf{C}_j^{\mathbf{b}\mathbf{a}}$  because  $\mathbf{C}_j^{\mathbf{b}\mathbf{a}} = \mathbf{C}_j^{\mathbf{a}\mathbf{b}T}$ .

The top-down connections between  $\mathcal{P}$  and  $\mathcal{N}^{\mathbf{x}}$ , on the other hand, returns *expectations* to  $\mathcal{N}^{\mathbf{x}}$  which are used to estimate  $\hat{\mathbf{k}}$  when  $\mathbf{k}$  is missing. Actually this expectations are the a posteriori probabilities computed using all sensory/motor modalities except  $\mathbf{k}$ , and an estimate  $\hat{\mathbf{k}}$  is computed using Equation (4.10) above.

The IGMN architecture shown in Figure 4.1 is based on the supposition of conditional independence among domains, which works as follows. Let  $\mathcal{A}$ ,  $\mathcal{B}$  and  $\mathcal{C}$  denote three sets of random variables. The variables in  $\mathcal{A}$  are said to be conditionally independent of  $\mathcal{B}$ ,

given  $\mathcal{C}$ , if the following condition holds (TAN; STEINBACH; KUMAR, 2006):

$$P(\mathcal{A}|\mathcal{B}, \mathcal{C}) = P(\mathcal{A}|\mathcal{C}). \quad (4.16)$$

This condition also implies that

$$P(\mathcal{A}, \mathcal{B}|\mathcal{C}) = P(\mathcal{A}|\mathcal{C}) \times P(\mathcal{B}|\mathcal{C}), \quad (4.17)$$

i.e., the *joint* posterior probability  $P(\mathcal{A}, \mathcal{B}|\mathcal{C})$  can be obtained by the product of the individual posterior probabilities  $P(\mathcal{A}|\mathcal{C})$  and  $P(\mathcal{B}|\mathcal{C})$  (PEREIRA; RAO, 2009). Therefore, IGMN adopts the strategy of using distinct covariance matrices to represent each modality, which is a reasonable constraint because in fact different sensory/motor modalities (e.g., a and b) have distinct domains. However, in IGMN the cortical regions are not fully independent, because as described above the correlations among cortical regions are kept on the association region  $\mathcal{P}$ .

This architecture is inspired on the memory-prediction framework (MPF) (HAWKINS, 2005), which states that different cortical regions are not fully connected in the neocortex. Instead, they are linked to the association areas  $\mathcal{P}$  through bottom-up and top-down connections, thus providing predictions and expectations, respectively, to all cortical regions  $\mathcal{N}^K$ . The main advantage of this strategy is to speed up IGMN and make it more suitable to real-time and critical applications, because it is much faster to invert two covariance matrices of size  $M$  than a single covariance matrix of size  $2M$ . Moreover, a large number of samples is required to obtain good estimates from a large covariance matrix, and therefore using this strategy IGMN becomes more aggressive, i.e., it is able to provide good estimates using few training samples.

Another advantage of this strategy is that it does not degrade the function approximation performed by IGMN, because in fact Equation 4.10 does not require a full covariance matrix  $\mathbf{C}_j$  to compute its estimates – but just the submatrices  $\mathbf{C}_j^{\mathcal{A}\mathcal{A}}$ ,  $\mathbf{C}_j^{\mathcal{B}\mathcal{B}}$  and  $\mathbf{C}_j^{\mathcal{B}\mathcal{A}}$  are required, and these submatrices correspond to the covariance matrices of the cortical regions plus the association matrix  $\mathbf{C}_j^{\mathcal{A}\mathcal{B}} = \mathbf{C}_j^{\mathcal{B}\mathcal{A}T}$ . Therefore using this strategy we can speed up the IGMN performance without reducing its representational power and regression quality. Next section describes the IGMN operation in details.

### 4.3 IGMN operation

This section describes the basic operation of IGMN, i.e., how the information flows through the bottom-up and top-down connections as the data are processed. Like other supervised ANN models, IGMN has two operation modes, called *learning* and *recalling*. But unlike most ANN models, in IGMN these operations don't need to occur separately, i.e., the learning and recalling modes can be intercalated. In fact, even after the presentation of a single training pattern the neural network can already be used in the recalling mode (i.e., IGMN can immediately use the acquired knowledge), and the produced estimates become more precise as more training data are presented. Moreover, the learning process can proceed perpetually, i.e., the neural network parameters can always be updated as new training data arrive.

As described in Subsection 2.5.1, the incremental Gaussian mixture model (IGMM) uses a minimum likelihood criterion (Equation 2.36) to decide if it is necessary to add a new Gaussian distribution to the mixture model. Although this criterion works fine in an unsupervised clustering algorithm, in supervised function approximation it may be difficult to adjust the approximation level using the  $\tau_{nov}$  parameter, because  $\tau_{nov}$  actually

controls the granularity of the model (i.e., the number of neurons in each region), which just *indirectly* changes the approximation error  $\varepsilon$ . Therefore, using the minimum likelihood criterion in IGMN may require to fine tune the  $\tau_{nov}$  parameter for achieving the required approximation level.

To prevent this problem, IGMN adopts an error-driven mechanism to decide if it is necessary to add a neuron in each region for *explaining* a new data vector  $\mathbf{z}^t$ . This error-driven mechanism is inspired on the Constructivist IA (DRESCHER, 1991; CHAPUT, 2004; PEROTTO, 2010), described in Section 1.2, where the accommodation process occurs when it is necessary to change the neural network structure (i.e. to add a neuron in each region) to account for a new *experience* which is not explained for the current schemata (i.e., the current ANN structure), and the assimilation process occurs when the new experience is well explained in terms of the existing schemata (PIAGET, 1954). In mathematical terms, the neural network structure is changed if the instantaneous approximation error  $\varepsilon$  is larger than a user specified threshold  $\varepsilon_{max}$ . Hence, the most critical IGMM parameter,  $\tau_{nov}$ , is replaced in IGMN by another parameter,  $\varepsilon_{max}$ , equally important (it still controls the granularity of the model) but easier to adjust, because in general we know the maximum error allowed in a given task, specially if this error is normalized.

The following subsections describe the IGMN operation during learning and recalling. To simplify our explanation, we will consider that the neural network has just two cortical regions,  $\mathcal{N}^a$  and  $\mathcal{N}^b$ , that receive the stimuli  $\mathbf{a}$  and  $\mathbf{b}$ , respectively. It will be also considered that we are estimating  $\hat{\mathbf{b}}$  from  $\mathbf{a}$  in the recalling mode. But it is important to remember that: (i) IGMN can have more than two cortical regions (one for each sensory/motor stimulus  $\mathbf{k}$ ); and (ii) after training it can be used to estimate either  $\hat{\mathbf{a}}$  or  $\hat{\mathbf{b}}$  (i.e., there is no difference between *inputs* and *outputs* in IGMN).

### 4.3.1 Learning mode

The learning algorithm used by IGMN is based on IGMM, presented in Subsection 2.5.5, but it has many modifications which adapt it to supervised tasks such as incremental function approximation and prediction. Figure 4.2 shows how the information flows through IGMN during learning. Before learning starts the neural network is empty, i.e., all regions have  $M = 0$  neurons. When the first training pattern  $\mathbf{z}^1 = \{\mathbf{a}^1, \mathbf{b}^1\}$  arrives (the superscript ‘1’ indicates the time  $t = 1$ ), a neuron in each region is created centered on  $\mathbf{z}^1$  and the neural network parameters are initialized as follows:

$$M = 1; \quad sp_1 = 1.0; \quad v_1 = 0; \quad p(1) = 1.0; \quad \mathbf{C}_1^{AB} = \mathbf{0}; \\ \boldsymbol{\mu}_1^a = \mathbf{a}^1; \quad \boldsymbol{\mu}_1^b = \mathbf{b}^1; \quad \mathbf{C}_1^a = \boldsymbol{\sigma}_{ini}^a \mathbf{I}; \quad \mathbf{C}_1^b = \boldsymbol{\sigma}_{ini}^b \mathbf{I},$$

where the subscript ‘1’ indicates the neuron  $j = 1$  in each region,  $M$  is the number of neurons in each region (all regions have the same size  $M$ ),  $sp$  is the accumulator of posterior probabilities maintained in the association region  $\mathcal{P}$ ,  $v_1$  the *age* of the first neuron,  $\mathbf{0}$  is a zero matrix of size  $D^a \times D^b$ ,  $\boldsymbol{\sigma}_{ini}^a$  and  $\boldsymbol{\sigma}_{ini}^b$  are diagonal matrices that define the initial radius of the covariance matrices (the pdf is initially circular but it changes to reflect the actual data distribution as new training data arrive) and  $\mathbf{I}$  denotes the identity matrix. As in IGMM,  $\boldsymbol{\sigma}_{ini}^a$  and  $\boldsymbol{\sigma}_{ini}^b$  are initialized in IGMN using a user defined fraction  $\delta$  of the overall variance (e.g.,  $\delta = 1/100$ ) of the corresponding attributes, estimated from the range of these values according to:

$$\boldsymbol{\sigma}_{ini}^k = \delta [\max(\mathbf{k}) - \min(\mathbf{k})], \quad (4.18)$$

where  $[\min(\mathbf{k}), \max(\mathbf{k})]$  defines the domain of a sensory/motor modality  $\mathbf{k}$  (throughout this chapter the symbol  $k$  will be used to indicate any sensory/motor modality, i.e., either  $a$

or  $b$  in this case). It is important to say that it is not necessary to know the *exact* minimum and maximum values along each dimension to compute  $\sigma_{ini}^{\mathcal{K}}$  (which would require that the entire training database be available before learning), but instead just the approximate domain of each feature. In a Pioneer 3-DX mobile robot platform, for instance, the sonar readings are limited to the interval  $(0, 5000)$ , and therefore we can set  $\min(\mathbf{k}) = 0$  and  $\max(\mathbf{k}) = 5000$  in this case.

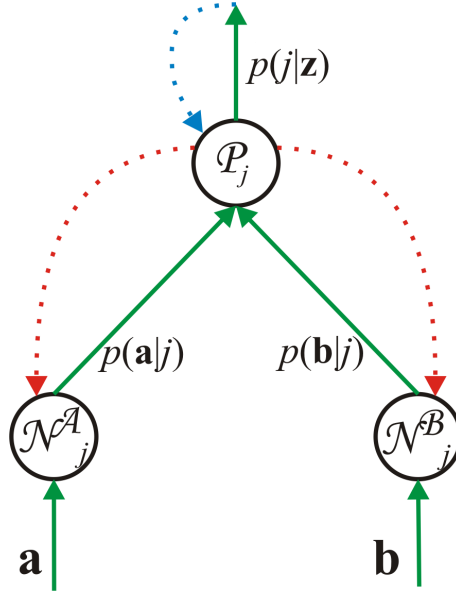


Figure 4.2: Information flow through the neural network during learning

When a new training pattern  $\mathbf{z}^t$  arrives, all cortical regions are activated, i.e.,  $p(\mathbf{k}|j)$  is computed using Equation 4.14 above, and the probabilities  $p(\mathbf{z}^t|j)$  are sent to the association region  $\mathcal{P}$ , which computes the *joint* posterior probabilities  $p(j|\mathbf{z}^t)$  using the Bayes' rule in (4.15). After this, the posterior probabilities  $p(j|\mathbf{z}^t)$  are sent back to all cortical regions, i.e.,  $\mathcal{N}^A$  and  $\mathcal{N}^B$ , which compute their estimates as follows:

$$\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{z}^t) [\boldsymbol{\mu}_j^B + \mathbf{C}_j^{BA} \mathbf{C}_j^{A-1} (\mathbf{a}^t - \boldsymbol{\mu}_j^A)], \quad (4.19)$$

$$\hat{\mathbf{a}} = \sum_{j=1}^M p(j|\mathbf{z}^t) [\boldsymbol{\mu}_j^A + \mathbf{C}_j^{AB} \mathbf{C}_j^{B-1} (\mathbf{b}^t - \boldsymbol{\mu}_j^B)], \quad (4.20)$$

where  $\mathbf{C}_j^{AB}$  is the  $j$ th association matrix maintained in association region  $\mathcal{P}$  and  $\mathbf{C}_j^{BA}$  is its transpose. Note that as the covariance matrices  $\mathbf{C}_j^A$  and  $\mathbf{C}_j^B$  were already inverted when  $\mathcal{N}^A$  and  $\mathcal{N}^B$  were activated in the bottom-up direction, we don't need to invert them again, i.e., we can temporarily store the corresponding inverse matrices to speed up the IGMN learning algorithm.

Equations 4.19 and 4.20 are similar to (4.10), but the joint a posteriori probability  $p(j|\mathbf{z}^t)$ , rather than  $p(j|\mathbf{b})$  and  $p(j|\mathbf{a})$ , is used to compute the estimates. Although the joint probability is not required to compute the estimates, if  $p(j|\mathbf{z}^t)$  is available it is better to use it because it prevents some problems which occur if the association  $\mathbf{a} \leftrightarrow \mathbf{b}$  is not a mathematical function in both directions, as will be described in Section 4.4. Using the



estimates  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  the normalized approximation error  $\varepsilon$  is given by:

$$\varepsilon = \max_{\mathbf{k} \in \mathbf{z}} \left\{ \max_{i \in D^{\mathcal{X}}} \left[ \frac{\|k_i^t - \hat{k}_i\|}{\max(k_i) - \min(k_i)} \right] \right\} \quad (4.21)$$

where  $[\min(k_i), \max(k_i)]$  defines the domain of the sensory/motor feature  $k_i$ . Again  $\min(k_i)$  and  $\max(k_i)$  do not need to be the exact minimum and maximum values of  $\mathbf{k}$  – they may be just approximations of the domain of each  $k_i$  feature (in fact  $\min(k_i)$  and  $\max(k_i)$  are used just to make IGMN more independent from the range of the data features). If  $\varepsilon$  is larger than a user specified threshold,  $\varepsilon_{max}$ , than  $\mathbf{z}^t$  is not considered as *represented* by any existing cortical neuron in  $\mathcal{N}^{\mathcal{X}}$ . In this case, a new unit  $j$  is created in each region and centered on  $\mathbf{z}^t$ , i.e.:

$$M^* = M + 1; \quad v_j = 0; \quad sp_j = 1.0; \quad \mathbf{C}_j^{\mathcal{A}\mathcal{B}} = \mathbf{0}; \\ \boldsymbol{\mu}_j^{\mathcal{A}} = \mathbf{a}^t; \quad \boldsymbol{\mu}_j^{\mathcal{B}} = \mathbf{b}^t; \quad \mathbf{C}_j^{\mathcal{A}} = \boldsymbol{\sigma}_{ini}^{\mathcal{A}2} \mathbf{I}; \quad \mathbf{C}_j^{\mathcal{B}} = \boldsymbol{\sigma}_{ini}^{\mathcal{B}2} \mathbf{I},$$

and all priors of the association region  $\mathcal{P}$  are adjusted to satisfy constraints (2.2) by:

$$p(j)^* = \frac{sp_j}{\sum_{q=1}^{M^*} sp_q}. \quad (4.22)$$

Otherwise (if  $\mathbf{z}$  is well explained by any of the existing Gaussian units), the a posteriori probabilities  $p(j|\mathbf{z}^t)$  are added to the current value of the  $sp(j)$  on the association region:

$$sp_j^* = sp_j + p(j|\mathbf{z}^t), \quad \forall j, \quad (4.23)$$

and the priors  $p(j)$  are recomputed using (4.22). Then  $\omega_j = p(j|\mathbf{z}^t)/sp_j^*$  is sent back to all cortical regions (as shown by the dashed lines in Figure 4.2), and the parameters of all neurons in  $\mathcal{N}^{\mathcal{X}}$  are updated using the IGMM recursive equations (Subsection 2.5.2) reproduced here (with an adapted notation) for convenience:

$$\boldsymbol{\mu}_j^{\mathcal{X}*} = \boldsymbol{\mu}_j^{\mathcal{X}} + \omega_j (\mathbf{z}^t - \boldsymbol{\mu}_j^{\mathcal{X}}) \quad (4.24)$$

$$\mathbf{C}_j^{\mathcal{X}*} = \mathbf{C}_j^{\mathcal{X}} - (\boldsymbol{\mu}_j^{\mathcal{X}*} - \boldsymbol{\mu}_j^{\mathcal{X}})(\boldsymbol{\mu}_j^{\mathcal{X}*} - \boldsymbol{\mu}_j^{\mathcal{X}})^T + \omega_j [(\mathbf{z} - \boldsymbol{\mu}_j^{\mathcal{X}*})(\mathbf{z} - \boldsymbol{\mu}_j^{\mathcal{X}*})^T - \mathbf{C}_j^{\mathcal{X}}] \quad (4.25)$$

where the superscript ‘\*’ refers to the new (updated) values. Finally the association matrix  $\mathbf{C}_j^{\mathcal{A}\mathcal{B}}$  is updated using the following recursive equation:

$$\mathbf{C}_j^{\mathcal{A}\mathcal{B}*} = \mathbf{C}_j^{\mathcal{A}\mathcal{B}} - (\boldsymbol{\mu}_j^{\mathcal{A}*} - \boldsymbol{\mu}_j^{\mathcal{A}})(\boldsymbol{\mu}_j^{\mathcal{B}*} - \boldsymbol{\mu}_j^{\mathcal{B}})^T + \omega_j [(\mathbf{a}^t - \boldsymbol{\mu}_j^{\mathcal{A}*})(\mathbf{b}^t - \boldsymbol{\mu}_j^{\mathcal{B}*})^T - \mathbf{C}_j^{\mathcal{A}\mathcal{B}}], \quad (4.26)$$

which is derived using the same principles described in Subsection 2.5.2. This equation is another important contribution of this thesis, because it allows computing the covariances among distinct cortical regions incrementally, and thus estimating a missing stimulus  $\mathbf{k}$  by (4.10) without having to maintain and invert a complete variance/covariance matrix. In fact, using (4.26) the complete variance/covariance matrix is broken down in separate submatrices that can be efficiently computed and/or inverted.

Algorithm 2 presents a detailed pseudocode of the learning algorithm used by IGMN. This algorithm can proceed perpetually as new training data arrive, i.e., IGMN can learn continuously and indefinitely without suffering from catastrophic interference. Algorithm 2 also shows that IGMN uses the same mechanism described in Subsection 2.5.4 to prevent an eventual saturation of the  $sp$  accumulators and the mechanism presented in Subsection 2.5.3 to identify and delete spurious components. However, the stability criterion is not used by IGMN to control the creation of new units (just the error driven mechanism described above is enough). Next subsection describes the IGMN operation during recalling.

---

**Algorithm 2** Summary of the IGMN learning algorithm
 

---

**for all** training data  $\mathbf{z} = \{\mathbf{a}, \mathbf{b}\}$  **do**  
 {Activate all cortical regions  $\mathcal{N}^{\mathcal{X}}$  via bottom-up connections}  
 $p(\mathbf{k}|j) = \mathcal{N}(\mathbf{k}|\boldsymbol{\mu}_j^{\mathcal{X}}, \mathbf{C}_j^{\mathcal{X}}), \forall j, \mathcal{N}^{\mathcal{X}}$   
 {Activate the association region  $\mathcal{P}$ }  
**for all** neuron  $j$  **do**  

$$p(j|\mathbf{z}) = \frac{p(\mathbf{a}|j)p(\mathbf{b}|j)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q)p(\mathbf{b}|q)p(q)}$$
  
**end for**  
 {Reactivate all cortical regions via top-down connections and compute  $\varepsilon$ }  
 $\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{z}^t)[\boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}}\mathbf{C}_j^{\mathcal{A}-1}(\mathbf{a}^t - \boldsymbol{\mu}_j^{\mathcal{A}})]$   
 $\hat{\mathbf{a}} = \sum_{j=1}^M p(j|\mathbf{z}^t)[\boldsymbol{\mu}_j^{\mathcal{A}} + \mathbf{C}_j^{\mathcal{A}\mathcal{B}}\mathbf{C}_j^{\mathcal{B}-1}(\mathbf{b}^t - \boldsymbol{\mu}_j^{\mathcal{B}})]$   
 $\varepsilon = \max_{\mathbf{k} \in \mathbf{z}} \{ \max_{i \in D^{\mathcal{X}}} [ \|k_i^t - \hat{k}_i\| / (\max(k_i) - \min(k_i)) ] \}$   
 {Create a new neuron  $j$  in each region if necessary}  
**if**  $M < 1$  **or**  $\varepsilon > \varepsilon_{max}$  **then**  
 $M^* = M + 1; \quad v_j = 0; \quad sp_j = 1.0; \quad p(j) = sp_j / \sum_{q=1}^M sp_q;$   
 $\mathbf{C}_j^{\mathcal{A}\mathcal{B}} = \mathbf{0}; \quad \boldsymbol{\mu}_q^{\mathcal{X}} = \mathbf{k}; \quad \mathbf{C}_j^{\mathcal{X}} = \sigma_{ini}^{\mathcal{X}^2} \mathbf{I}, \quad \forall \mathcal{N}^{\mathcal{X}}$   
**end if**  
 {Adjust the neural network parameters of all regions}  
**for all** neuron  $j$  **do**  
 $sp_j^* = \alpha sp_j + p(j|\mathbf{z}); \quad p(j)^* = sp_j^* / \sum_{q=1}^M sp_q^*; \quad \omega_j = p(j|\mathbf{z}) / sp_j^*$   
 $\mathbf{C}_j^{\mathcal{A}\mathcal{B}*} = \mathbf{C}_j^{\mathcal{A}\mathcal{B}} - (\boldsymbol{\mu}_j^{\mathcal{A}*} - \boldsymbol{\mu}_j^{\mathcal{A}})(\boldsymbol{\mu}_j^{\mathcal{B}*} - \boldsymbol{\mu}_j^{\mathcal{B}})^T + \omega_j [(\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}*})(\mathbf{b} - \boldsymbol{\mu}_j^{\mathcal{B}*})^T - \mathbf{C}_j^{\mathcal{A}\mathcal{B}}]$   
**for all** cortical region  $\mathcal{N}^{\mathcal{X}}$  **do**  
 $\boldsymbol{\mu}_j^{\mathcal{X}*} = \boldsymbol{\mu}_j^{\mathcal{X}} + \omega_j (\mathbf{k} - \boldsymbol{\mu}_j^{\mathcal{X}})$   
 $\mathbf{C}_j^{\mathcal{X}*} = \mathbf{C}_j^{\mathcal{X}} - (\boldsymbol{\mu}_j^{\mathcal{X}*} - \boldsymbol{\mu}_j^{\mathcal{X}})(\boldsymbol{\mu}_j^{\mathcal{X}*} - \boldsymbol{\mu}_j^{\mathcal{X}})^T + \omega_j [(\mathbf{k} - \boldsymbol{\mu}_j^{\mathcal{X}*})(\mathbf{k} - \boldsymbol{\mu}_j^{\mathcal{X}*})^T - \mathbf{C}_j^{\mathcal{X}}]$   
**end for**  
**end for**  
 {Restart the accumulators and delete all spurious components}  
**if**  $(\sum_{j=1}^M sp_j) \geq \beta sp_{max}$  **then**  $sp_j^* = \gamma sp_j, \forall j$   
 $v_j = v_j + 1, \forall j$   
**if**  $v_j > v_{min}$  **and**  $sp_j < sp_{min}$  **then delete** the  $j$ th component in all regions  
**end for**

---

### 4.3.2 Recalling mode

In the recalling mode, a stimulus (e.g.,  $\mathbf{a}$ ) is presented to a *partially* trained neural network (as the learning process proceeds perpetually, in IGMN we never consider that the training process is over), which computes an estimate for another stimulus (e.g.,  $\hat{\mathbf{b}}$ ). As said before, IGMN can be used to estimate either  $\hat{\mathbf{a}}$  (Figure 4.3(b)) or  $\hat{\mathbf{b}}$  (Figure 4.3(a)) – the training process is unique for both estimations – but to simplify our explanation in this and the following sections we will consider that we are estimating  $\hat{\mathbf{b}}$  from  $\mathbf{a}$ . Figure 4.3 shows how the information flows through IGMN during recalling.

Initially the stimulus  $\mathbf{a}$  is received in the cortical region  $\mathcal{N}^{\mathcal{A}}$ , where each neuron  $j$  computes  $p(\mathbf{a}|j)$  using (4.14). These predictions are sent to the association region  $\mathcal{P}$  through the bottom-up connections, which is activated using just  $p(\mathbf{a}|j)$ :

$$p(j|\mathbf{a}) = \frac{p(\mathbf{a}|j)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q)p(q)}. \quad (4.27)$$

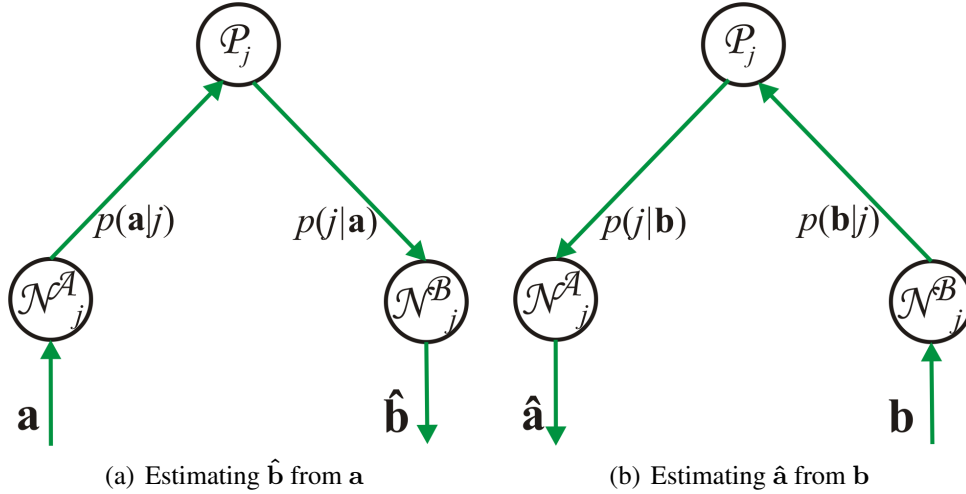


Figure 4.3: Information flow in IGMN during recalling

After this,  $p(j|\mathbf{a})$  is sent to the cortical region  $\mathcal{N}^B$  via the top-down connections, and  $\mathcal{N}^B$  computes the estimated stimulus  $\hat{\mathbf{b}}$  using (4.10), here reproduced with an adapted notation for convenience:

$$\bar{\mathbf{x}}_j^B = \boldsymbol{\mu}_j^B + \mathbf{C}_j^{BA} \mathbf{C}_j^{A-1} (\mathbf{a} - \boldsymbol{\mu}_j^A), \quad (4.28)$$

$$\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{a}) \bar{\mathbf{x}}_j^B, \quad (4.29)$$

A useful particularity of IGMN is that it allows to estimate not just the expected value of  $\mathbf{b}$  (i.e., a punctual prediction) but also the variance/covariance matrix  $\hat{\mathbf{C}}^B$ . This is possible because IGMN maintains in  $\mathcal{N}^B$  not just the mean vector  $\boldsymbol{\mu}_j^B$ , but also the covariance matrix  $\mathbf{C}_j^B$ . Hence, we can estimate the covariance matrix  $\hat{\mathbf{C}}^B$  at  $\hat{\mathbf{b}}$  using Equation 4.12, here also reproduced for convenience:

$$\hat{\mathbf{C}}^B = \sum_{j=1}^M p(j|\mathbf{a}) \left\{ \mathbf{C}_j^B - \mathbf{C}_j^{BA} \mathbf{C}_j^{A-1} \mathbf{C}_j^{BA^T} + \|\bar{\mathbf{x}}_j^B - \hat{\mathbf{b}}\|^2 \right\}, \quad (4.30)$$

where  $\bar{\mathbf{x}}_j^B$  is computed using (4.28). If  $\mathbf{b}$  is an unidimensional vector then  $\hat{\mathbf{C}}^B$  is reduced to the sample variance  $\sigma^2$ . Using  $\hat{\mathbf{C}}^B$  we can also compute other statistical estimators such as the confidence intervals and the hypothesis test. To estimate the confidence interval  $\hat{\mathbf{C}}$ , for instance, we can use the following equation (COX; HINKLEY, 1974):

$$\hat{\mathbf{C}} = \hat{\mathbf{b}} \pm t_{\alpha, n-1} \frac{\hat{\mathbf{C}}^B}{\sqrt{n}} \quad (4.31)$$

where  $t_{\alpha, n-1}$  is Student's  $t$ -distribution with  $n - 1$  degrees of freedom and  $n$  can be estimated from the  $sp$  accumulators. Hence, IGMN not only provides an estimate of the target stimulus but it can also inform the confidence of their estimates, and this functionality is not found in most existing ANN models. Moreover, the confidence estimates computed by IGMN are very useful to make decisions, for instance, because they allow us to plan the actions using not just an estimate of  $\mathbf{b}$  but also the confidence levels of its estimates. Algorithm 3 summarizes this recalling procedure described above, that we will call *special IGMN recalling algorithm* to differentiate it from the *general IGMN recalling algorithm* presented in the next section.

---

**Algorithm 3** *Summary of the special IGMN recalling algorithm*


---

**for all data  $\mathbf{a}$  do**

{Activate the cortical region  $\mathcal{N}^{\mathcal{A}}$ }

**for all neuron  $j$  do**

$$p(\mathbf{a}|j) = \mathcal{N}(\mathbf{a}|\boldsymbol{\mu}_j^{\mathcal{A}}, \mathbf{C}_j^{\mathcal{A}}) = \frac{1}{(2\pi)^{D^{\mathcal{A}/2}} \sqrt{|\mathbf{C}_j^{\mathcal{A}}|}} \exp \left\{ -\frac{1}{2}(\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})^T \mathbf{C}_j^{\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}}) \right\}$$

**end for**

{Activate the association region  $\mathcal{P}$ }

**for all neuron  $j$  do**

$$p(j|\mathbf{a}) = \frac{p(\mathbf{a}|j) p(j)}{\sum_{q=1}^M p(\mathbf{a}|q) p(q)}$$

**end for**

{Activate the cortical region  $\mathcal{N}^{\mathcal{B}}$  to compute the estimates  $\hat{\mathbf{b}}$  and  $\hat{\mathbf{C}}^{\mathcal{B}}$ }

**for all neuron  $j$  do**

$$\bar{\mathbf{x}}_j^{\mathcal{B}} = \boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})$$

**end for**

$$\hat{\mathbf{b}} = \sum_{j=1}^M p(j|\mathbf{a}) \bar{\mathbf{x}}_j^{\mathcal{B}}$$

$$\hat{\mathbf{C}}^{\mathcal{B}} = \sum_{j=1}^M p(j|\mathbf{a}) \left\{ \mathbf{C}_j^{\mathcal{B}} - \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}-1} \mathbf{C}_j^{\mathcal{B}\mathcal{A}T} + \|\bar{\mathbf{x}}_j^{\mathcal{B}} - \hat{\mathbf{b}}\|^2 \right\}$$

**end for**

---

The special IGMN recalling algorithm described above can be used to predict either  $\hat{\mathbf{a}}$  or  $\hat{\mathbf{b}}$  if both  $f(\cdot)$  and  $f(\cdot)^{-1}$  are mathematical functions (EVES, 1990), i.e., the relationship  $\mathbf{b} \leftrightarrow \mathbf{a}$  is single-valued in both directions. However, if this relationship is not a function in one of these directions (or both) then the approximation computed by (4.29) will be poor in those parts of the domain where the target data are multi-valued. Next section presents an extension of this algorithm, called *general IGMN recalling algorithm*, which produces valid answers even in those parts of the state space where either  $f(\cdot)$  or  $f(\cdot)^{-1}$  (or both) does not represent mathematical functions (i.e., the target data are multi-valued).

#### 4.4 Dealing with multi-valued target data

As said before, one of the main advantages of IGMN is that we can use the same partially trained neural network for estimating either  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  provided that both  $f(\cdot)$  and  $f(\cdot)^{-1}$  are mathematical functions (EVES, 1990). However, in many potential applications of neural networks there is a well-defined *forward* problem which is characterized by a *functional* (i.e. single-valued) mapping, but the corresponding *inverse* problem is multi-valued (BISHOP, 1995). A typical application that falls into this category is robot kinematics (SCIAVICCO; SICILIANO, 1996), where the forward problem consists in computing the position of a robot manipulator from the angles of its joints, and the inverse problem consists in computing the angles required to place the manipulator in a given position. In this application the forward problem is single-valued, but frequently the inverse problem has multiple valid answers, i.e., there are many combinations of joint angles that place the manipulator into the required position (DUDEK; JENKIN, 2000; HEINEN; OSÓRIO, 2007a, 2006a).

If a traditional ANN model such as MLP, RBF and GRNN is used to solve an inverse problem, the least squares approach (HAYKIN, 2008) will approximate the conditional

average of the target data, and this will frequently lead to a poor performance since the average of many solutions is not necessarily itself a solution (BISHOP, 1995). The same result is obtained using the special IGMN recalling algorithm described above (Algorithm 3), as illustrates Figure 4.4(a). In this figure the regression is shown using a gray line, the Gaussian components are represented by ellipses whose width is equivalent to a Mahalanobis distance of two and the mean of each distribution is shown by solid circles in the center of the respective ellipses. It can be noted in this figure that the function  $b = \sin(a)^{-1}$  has two valid answers in the intervals  $[-1.0, -0.58]$  and  $[0.58, 1]$  of its domain, and the approximation is poor in these intervals because the estimates lie between the valid answers and don't belong to the target function.

According to Bishop (1995), this problem cannot be solved by modifying the network architecture or the training algorithm, since it is a fundamental consequence of using a sum-of-squares error function. Hence, if we are using a traditional neural network model such as MLP and RBF the only solution is to restrict the domain in which this problem is evaluated. However, in many problems of high dimensionality, where the visualization of the data is not straightforward, it can be very difficult to identify those regions of the state space where the target data are multi-valued (BISHOP, 1995).

Using IGMN, on the other hand, this problem can be easily identified and tackled because IGMN goes beyond the Gaussian description of the distribution of the target data, thus adopting a more general model for the conditional density. More specifically, IGMN can tackle this problem using one of the following approaches:

1. Do not provide an answer in those regions of the state space where the target data are multi-valued (i.e., to return an "I don't know" flag);
2. To return the most likely hypothesis when the problem has multiple answers;
3. To return all possible answers and their corresponding probabilities.

Although the third solution is more general, it requires a change in the neural network structure in order to provide a vector containing all possible answers rather than a single scalar value as output. However, in many problems such as control applications and robot kinematics we are interested in finding just one valid answer, and therefore the second solution is better suitable.

The strategy used by IGMN to solve the problem described above consists in discarding those distributions that are very far (e.g., a Mahalanobis distance  $\geq 5$ ) from the center of the maximum likelihood (ML) hypothesis  $\ell$  (the Gaussian distribution with the largest a posteriori probability) *before* estimating  $\hat{\mathbf{b}}$ . As a matter of fact, if a distribution is farther than a Mahalanobis distance of five then its influence is negligible. Hence, in regions of the state space where the target data are single-valued only distributions with a posteriori probabilities close to zero will be discarded and the resulting approximation will be practically identical to that computed by the special recalling algorithm.

On the other hand, if very distant distributions have considerable a posteriori probabilities, then they necessarily belong to different branches of the mapping and the target data are multi-valued in that region of the state space. In this case we can discard those distributions that don't belong to the ML branch, and hence IGMN will return the valid answer which has the largest "probability mass", i.e., it will compute a valid solution using a soft interpolation among those hypothesis of the ML branch. Figure 4.4(b) shows the resulting approximation computed using this method.

In mathematical terms this strategy is implemented as follows. Suppose that we are estimating a sensory/motor modality  $\hat{\mathbf{b}}$  from  $\mathbf{a}$  and the mapping  $\mathbf{a} \rightarrow \mathbf{b}$  is multi-valued.

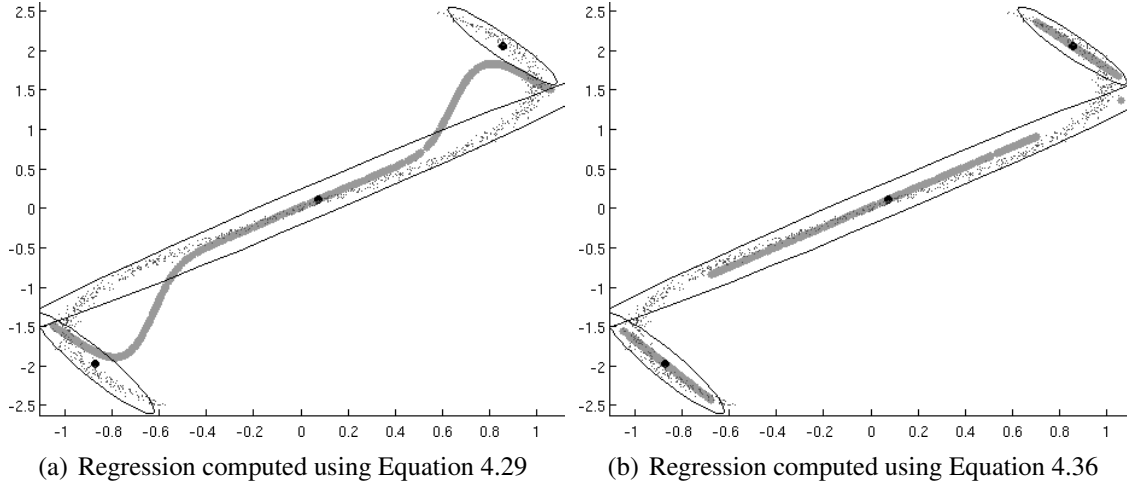


Figure 4.4: A typical inverse problem in which the target data are multi-valued

Initially  $\mathbf{a}$  is propagated through  $\mathcal{N}^{\mathcal{A}}$  as usual, and the corresponding  $p(\mathbf{a}|j)$  are sent to the association region  $\mathcal{P}$ , which computes  $p(j|\mathbf{a})$  using the Bayes' rule (4.27). Then the ML hypothesis  $\ell$  is computed using the following equation:

$$\ell = \arg \max_j [p(j|\mathbf{a})], \quad (4.32)$$

and the Mahalanobis distances among the center of the ML hypothesis and the remaining distributions,  $\mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_{\ell}^{\mathcal{B}}, \boldsymbol{\mu}_j^{\mathcal{B}})$ , are computed through:

$$\mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_{\ell}^{\mathcal{B}}, \boldsymbol{\mu}_j^{\mathcal{B}}) = \sqrt{(\boldsymbol{\mu}_{\ell}^{\mathcal{B}} - \boldsymbol{\mu}_j^{\mathcal{B}})^T \mathbf{C}_j^{\mathcal{B}-1} (\boldsymbol{\mu}_{\ell}^{\mathcal{B}} - \boldsymbol{\mu}_j^{\mathcal{B}})}. \quad (4.33)$$

If  $\mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_{\ell}^{\mathcal{B}}, \boldsymbol{\mu}_j^{\mathcal{B}})$  is larger than a user-specified Mahalanobis distance  $\Omega$ , then the distribution  $j$  is *discarded*, i.e., it is not used to compute  $\hat{\mathbf{b}}$ . Hence, a new vector  $p(\mathbf{a}|j, \ell)$  is created using just the pdfs of the non-discarded components, i.e.:

$$p(\mathbf{a}|j, \ell) = \begin{cases} p(\mathbf{a}|j) & \text{if } \mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_{\ell}^{\mathcal{A}}, \boldsymbol{\mu}_j^{\mathcal{A}}) > \Omega \\ 0 & \text{otherwise} \end{cases}. \quad (4.34)$$

After this the a posteriori probabilities of the ML branch are computed using the Bayes' rule:

$$p(j|\mathbf{a}, \ell) = \frac{p(\mathbf{a}|j, \ell)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q, \ell)p(q)}, \quad (4.35)$$

and the estimate of the most likely hypothesis is computed using the following equation:

$$\hat{\mathbf{b}}_{\ell} = \sum_{j=1}^M p(j|\mathbf{a}, \ell) [\boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})]. \quad (4.36)$$

Finally, the corresponding probability of the hypothesis  $\ell$  is given by:

$$p(\ell|\mathbf{a}) = \frac{\sum_{j=1}^M p(\mathbf{a}|j, \ell)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q)p(q)}. \quad (4.37)$$

Note that if the mapping  $\mathbf{a} \rightarrow \mathbf{b}$  is single-valued, then the prediction  $\hat{\mathbf{b}}_{\ell}$  will be very similar to that produced by (4.10) provided that  $\Omega$  is sufficient large. A good choice is

$\Omega = 5$ , because the probability that a point belonging to a distribution  $j$  falls outside a Mahalanobis distance of five is  $\approx 5.7 \times 10^{-7}$ , and hence the influence of  $j$  over the current hypothesis  $\ell$  is negligible. In fact, a Mahalanobis distance of five is a common choice in visual applications that use Gaussian kernels as filters (HEINEN; ENGEL, 2009d,e). In our experiments, described in the next chapters, we have set  $\Omega = 6$  to guarantee that the mechanism described above does not influence the estimates computed in those regions where the target function is single-valued.

If we are estimating a sensory/motor modality  $\hat{\mathbf{a}}$  from  $\mathbf{b}$  the procedure described above will be exactly the same – only the roles of  $\mathbf{a}$  and  $\mathbf{b}$  will be interchanged. Moreover, if we need to find out all possible answers for a given inverse problem, on the other hand, then the procedure above can be iterated using as the current (i.e., ML) hypothesis the *discarded* distribution with the largest a posteriori probability  $p(j|\mathbf{b})$ . It is important to note that this mechanism does not need to be used during learning because as  $p(j|\mathbf{z})$  is computed using both  $p(\mathbf{a}|j)$  and  $p(\mathbf{b}|j)$  then just the Gaussian units of the current branch have significant a posteriori probabilities, i.e., if we are informing all sensory/motor stimuli it is not possible to have multiple valid solutions.

Algorithm 4 presents the IGMN recalling algorithm that uses the mechanism described above, that is called *general IGMN recalling algorithm* to differentiate it from the special IGMN recalling algorithm presented in Algorithm 3. This algorithm returns almost the same predictions computed by the Algorithm 3 in single-valued regions (provided that  $\Omega \geq 5$ ), and in multi-valued regions the general algorithm will provide the most likely prediction instead of the average of all valid answers.

---

**Algorithm 4** *Summary of the general IGMN recalling algorithm*

---

**for all** data  $\mathbf{a}$  **do**

  { Activate the cortical region  $\mathcal{N}^{\mathcal{A}}$  }

**for all** neuron  $j$  **do**

$$p(\mathbf{a}|j) = \mathcal{N}(\mathbf{a}|\boldsymbol{\mu}_j^{\mathcal{A}}, \mathbf{C}_j^{\mathcal{A}}) = \frac{1}{(2\pi)^{D^{\mathcal{A}}/2} \sqrt{|\mathbf{C}_j^{\mathcal{A}}|}} \exp \left\{ -\frac{1}{2}(\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})^T \mathbf{C}_j^{\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}}) \right\}$$

**end for**

  { Activate the association region  $\mathcal{P}$  }

**for all** neuron  $j$  **do**

$$p(j|\mathbf{a}) = \frac{p(\mathbf{a}|j)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q)p(q)}$$

**end for**

  { Compute the a posteriori probabilities for branch of the ML hypothesis  $\ell$  only }

$$\ell = \arg \max_j [p(j|\mathbf{a})]$$

$$\mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_\ell^{\mathcal{B}}, \boldsymbol{\mu}_j^{\mathcal{B}}) = \sqrt{(\boldsymbol{\mu}_\ell^{\mathcal{B}} - \boldsymbol{\mu}_j^{\mathcal{B}})^T \mathbf{C}_j^{\mathcal{B}-1} (\boldsymbol{\mu}_\ell^{\mathcal{B}} - \boldsymbol{\mu}_j^{\mathcal{B}})}$$

$$p(j|\mathbf{a}, \ell) = \frac{p(\mathbf{a}|j, \ell)p(j)}{\sum_{q=1}^M p(\mathbf{a}|q, \ell)p(q)}$$

  { Activate the cortical region  $\mathcal{N}^{\mathcal{B}}$  to compute the estimates  $\hat{\mathbf{b}}_\ell$  and  $\hat{\mathbf{C}}_\ell^{\mathcal{B}}$  }

**for all** neuron  $j$  **do**

$$\bar{\mathbf{x}}_j^{\mathcal{B}} = \boldsymbol{\mu}_j^{\mathcal{B}} + \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}-1} (\mathbf{a} - \boldsymbol{\mu}_j^{\mathcal{A}})$$

**end for**

$$\hat{\mathbf{b}}_\ell = \sum_{j=1}^M p(j|\mathbf{a}, \ell) \bar{\mathbf{x}}_j^{\mathcal{B}}$$

$$\hat{\mathbf{C}}_\ell^{\mathcal{B}} = \sum_{j=1}^M p(j|\mathbf{a}, \ell) \left\{ \mathbf{C}_j^{\mathcal{B}} - \mathbf{C}_j^{\mathcal{B}\mathcal{A}} \mathbf{C}_j^{\mathcal{A}-1} \mathbf{C}_j^{\mathcal{B}\mathcal{A}T} + \|\bar{\mathbf{x}}_j^{\mathcal{B}} - \hat{\mathbf{b}}_\ell\|^2 \right\}$$

**end for**

---

Using the general recalling algorithm IGMN can solve the forward and inverse problems using the same partially trained neural network even in regions where the target function is multi-valued. To the best of our knowledge, IGMN is the first ANN model endowed with this capability, although some possibilities have been pointed out in the context of Gaussian mixture models (BISHOP, 1995). Next section presents a detailed description of all configuration parameters used by IGMN and how to set them.

## 4.5 IGMN configuration parameters

IGMN has a total of eight configuration parameters. Three of them are related to restarting of the accumulators  $sp_j$  and can be configured as follows:  $sp_{max} = 10^8$ ,  $\beta = 0.8$  and  $\gamma = 0.5$ . These parameters are actually constants that have no influence in the IGMN performance: they only prevent an eventual saturation of the corresponding accumulators.

The configuration parameters  $sp_{min}$  and  $v_{min}$  are inherited from the IGMM stability criterion. In IGMM they control the addition of new components and remove the spurious ones. In IGMN, on the other hand, they are used only to remove spurious components. As described in Subsection 2.5.3, a natural choice for  $sp_{min}$  is  $D + 1$ , because according to Trávén (1991) a minimum  $D + 1$  samples are required to obtain a nonsingular estimate of an unconstrained covariance matrix. The  $v_{min}$  parameter, on the other hand, can be set to any value larger than  $D + 1$  such as 100 or 1000. Actually the only result of deleting spurious components is to speed up a little bit the ANN performance, because as the spurious components have low prior probabilities they don't have any influence in the approximation computed by IGMN.

The configuration parameter  $\Omega$ , used to discard distributions of other branches (Subsection 4.4), can always be set to  $\Omega = 6$ . This parameter is actually a constant, because it can be kept fixed even in problems where the target function is single-valued. Another configuration parameter of IGMN is  $\delta$ , used to set the initial radius of the covariance matrices,  $\sigma_{ini}^x$ . This parameter is not critical and can be set to any value in the interval  $0.005 \leq \delta \leq 0.05$  (its default value is 0.01). The only critical parameter of IGMN is the maximum approximation error  $\varepsilon_{max}$ , that controls the level of generalization of the neural network. This parameter must be adjusted according to the maximum allowed error in a given task. However, as the approximation error  $\varepsilon$  is normalized,  $\varepsilon_{max}$  is usually set in the interval  $0.01 \leq \varepsilon_{max} \leq 0.1$ .

Summing up, unless otherwise noted in all experiments described in this thesis the IGMN configuration parameters were set to  $sp_{max} = 10^{18}$ ,  $\beta = 0.8$ ,  $\gamma = 0.5$ ,  $sp_{min} = D + 1$ ,  $v_{min} = 100$ ,  $\Omega = 6$ ,  $\delta = 0.01$  and  $\varepsilon_{max} = 0.05$ . We conclude that, although IGMN has many configuration parameters, just one of them must really be adjusted by the user: the maximum approximation error  $\varepsilon_{max}$ . Moreover, this parameter is easier to adjust, because in general the user knows the normalized approximation error accepted in a given task. Next section concludes this chapter presenting a summary of the main characteristics and limitations of IGMN.

## 4.6 Final remarks

This chapter described the neural network model proposed in thesis, called IGMN, its main characteristics, mathematical basis, topology and learning algorithm. As described before, IGMN extends the incremental Gaussian mixture model (IGMM), described in Section 2.5, in the following aspects:



- IGMN can be used in supervised function approximation rather than only in unsupervised clustering tasks;
- It uses an error driven mechanism rather than the minimum likelihood criterion to add new units;
- IGMN can provide the confidence levels of its estimates;
- The same partially trained neural network can be used to estimate either  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  even when the target data are multi-valued;
- The computational complexity is broken down by using separate covariance matrices for distinct sensory/motor stimuli;
- Equations 4.10 and 4.26 allow a multivariate representation without having to maintain and/or invert full variance/covariance matrices.

Moreover, most of these characteristics are not present in other neural network models such as MLP, RBF and GRNN, which makes IGMN an useful tool for incremental function approximation and on-line prediction tasks.

The main drawback of IGMN is that it is necessary to invert the covariance matrices  $\mathbf{C}_j^{\mathbf{x}}$  for each training sample  $(\mathbf{a}, \mathbf{b}, \dots, \mathbf{k})$ , and therefore its computational complexity is  $O(NM(D^{\mathbf{a}}D^{\mathbf{b}} \dots D^{\mathbf{s}})^{\log_2 7})$ . Subsection 2.5.6 presents some ways to reduce this complexity, such as to compute the inverse matrix  $\mathbf{C}^{-1}$  directly using the method proposed by Sato and Ishii (2000). Another way to reduce the computational complexity is to separate the data features in more sensory/motor areas, thus dividing a big covariance matrix in many matrices of smaller size. Although this may degrade the computed approximation (specially when all features are highly correlated), it is still a better solution than to consider all features as conditionally independent. Next chapter presents some experiments performed to evaluate IGMN in function approximation and prediction tasks.



## 5 FUNCTION APPROXIMATION USING IGMN

This chapter describes several experiments to evaluate the performance of IGMN in function approximation and prediction tasks. The first set of experiments, described in Section 5.1, uses a sinusoidal data set to explore some basic aspects of the proposed model such as the advantages of the multivariate representation over the conditional independence hypothesis, how the order of presentation of data affects the results and the sensibility of the proposed model to its configuration parameters. Section 5.2 describes some experiments in which the performance of IGMN is evaluated using a more complex, three-dimensional “Mexican hat” function. Section 5.3 describes how IGMN can be used to predict both  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  from the same trained neural network, i.e., how IGMN can be used to solve the forward and inverse problems at the same time. Section 5.4 presents some experiments in which IGMN is used to identify a nonlinear plant originally proposed by Narendra and Parthasarathy (1990). Section 5.5 demonstrates how IGMN can be used for predicting the future values of a cyclic time series. Finally, Section 5.6 presents some final remarks about these experiments.

In practically all experiments presented in this chapter the performance of IGMN is compared against other ANN models, such as MLP and GRNN, available in the MATLAB Neural Network (NN) Toolbox software. The IGMN prototype was implemented in the C ANSI computer programming language. It uses POSIX threads to allow a parallel execution in multi-processed platforms. The inverse matrices and determinants are computed using the LU decomposition algorithm presented in Press et al. (1992) whose computational complexity is  $D^{\log_2 7}$ . The computer platform used in all experiments is a Dell Optiplex 755, equipped with an Intel(R) Core(TM)2 Duo CPU 2.33GHz processor, 64 bits architecture, 1.95GB of RAM memory, GPU Intel and operating system Ubuntu Linux 10.04 LTS of 64 bits.

### 5.1 Sinusoidal data set

This section describes a set of experiments to evaluate the performance of IGMN using a simply and easy to visualize two-dimensional function. More specifically, the data set used in this set of experiments is composed by  $N = 1000$  data samples generated using the following function:

$$\mathbf{b} = \sin(\mathbf{a}) + \boldsymbol{\epsilon}, \quad (5.1)$$

where  $\mathbf{a}$  is drawn from a uniform distribution in the interval  $[-\pi, \pi]$  and  $\boldsymbol{\epsilon}$  is an independent Gaussian noise vector with mean  $\mu_{\boldsymbol{\epsilon}} = 0$  and standard deviation  $\sigma_{\boldsymbol{\epsilon}} = 0.05$ . Figure 5.1 shows this data set, where the black dots represent the  $(a, b)$  sample pairs and the gray line represents the *target* function, i.e.,  $b = \sin(a)$  without noise.

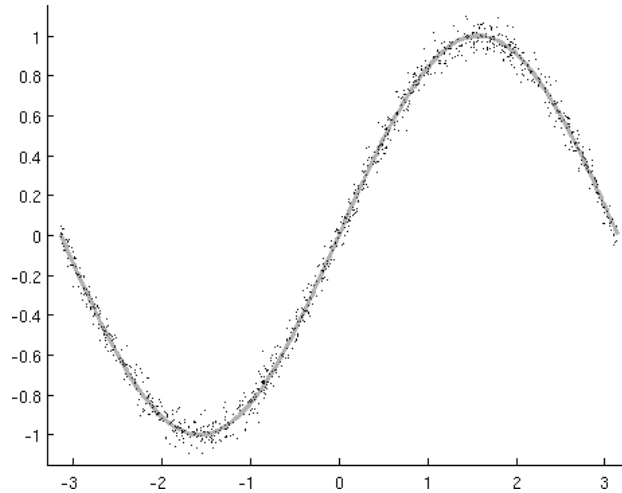


Figure 5.1: Sinusoidal data set of size  $N = 1000$  corrupted with Gaussian noise

Next subsections describe some experiments using this sinusoidal data set to evaluate many aspects of the IGMN architecture such as: the advantages of a multivariate representation, how the order of presentation of data affects the results and the sensibility of IGMN to its configuration parameters  $\varepsilon_{max}$  and  $\delta$ . Moreover, IGMN is compared with other connectionist approaches such as MLP, RBF and GRNN.

### 5.1.1 Standard $\times$ multivariate representation

The first experiment of this chapter is devised to compare the performance of IGMN using a complete multivariate representation (Equation 4.10) against the assumption of conditional independence among domains (4.9) used in the previous IGMN version presented in Heinen and Engel (2010a; 2010b). To perform this experiment we have implemented two distinct versions of IGMN:

- **Standard:** assumes that the cortical regions are conditionally independent (i.e., the cortical regions are not correlated) and estimates the missing stimuli using (4.9);
- **Multivariate:** assumes a full multivariate hypothesis and estimates the missing stimuli using (4.10).

Therefore we have used both IGMN versions for learning the sinusoidal data set shown in Figure 5.1 using different settings of the  $\varepsilon_{max}$  parameter. Table 5.1 shows the results obtained in this experiment. In this experiment the  $\delta$  parameter was set to  $\delta = 0.01$  (its default value) and the training data were presented to the neural network in ascending order of  $\mathbf{a}$ . As IGMN does not have any random initialization and/or decision, it is not necessary to repeat each experiment many times (the results are always identical for the same data set and configuration).

Each row on Table 5.1 represents a pair of experiments performed using a specific setting of the  $\varepsilon_{max}$  parameter. The first column indicates the  $\varepsilon_{max}$  setting used in the corresponding pair of experiments. The following columns show, respectively, the NRMS error, the number of neurons ( $M$ ) added to each region during training and the time required for learning the sinusoidal data set using the corresponding  $\varepsilon_{max}$  setting. Columns 2, 3, and 4 correspond to the standard IGMN version and columns 5, 6 and 7 correspond to the multivariate version of IGMN.

Observing Table 5.1 many important conclusions can be drawn from this experiment. First, the performance of the standard and multivariate versions of IGMN is quite good,

Table 5.1: Comparison between the standard and multivariate versions of IGMN

$\varepsilon_{max}$	Standard version			Multivariate version		
	NRMS	$M$	Time	NRMS	$M$	Time
0.2500	0.105653	12	0.010s	0.079194	3	0.000s
0.1000	0.032439	18	0.015s	0.040956	8	0.005s
0.0750	0.026288	24	0.020s	0.028581	7	0.010s
0.0500	0.023269	30	0.030s	0.023555	29	0.050s
0.0250	0.022063	57	0.075s	0.022111	57	0.080s
0.0100	0.022004	134	0.355s	0.022203	140	0.430s
0.0075	0.022024	173	0.630s	0.022099	173	0.800s

because using the original function  $b = \sin(a)$  to compute  $d$  (the desired value) the NRMS error is 0.024596. In fact, as the NRMS error is computed using the following equation:

$$NRMS = \frac{RMS}{\max(\mathbf{d}) - \min(\hat{\mathbf{b}})}$$

and the root mean square (RMS) error is given by the square root of the mean square error (MSE):

$$RMS = \sqrt{MSE}$$

$$MSE = \frac{1}{N} \sum_{n=1}^N (d_i - \hat{b}_i)^2,$$

then the expected NRMS error will be  $\approx 0.025$  because the noise vector has a standard deviation  $\sigma_\varepsilon = 0.05$  and  $\max(\mathbf{b}) - \min(\mathbf{b}) \approx 2$ . Therefore we can suppose that the best setting of  $\varepsilon_{max}$  is about 0.05 because it is supposed that the instantaneous normalized error  $\varepsilon$  will be bellow  $RMS \times 2 \approx 0.05$  for 98% of the training samples (this supposition is based on the fact that a Mahalanobis distance of two defines a contour encompassing approximately 98% of the training samples). Of course if the noise level is not known it is not possible to configure  $\varepsilon_{max}$  using this heuristic.

Observing Table 5.1 it can be noticed that in fact the best value of  $\varepsilon_{max}$  is  $\approx 0.05$ , because using lower values the neural network tries without success to reduce the error adding more neurons to each region. Nevertheless, it can be noticed that even using 173 neurons the proposed model does not suffer from overfitting (HAYKIN, 2008; PEREIRA; RAO, 2009), because (4.10) computes  $\hat{b}$  as a soft interpolation among the Gaussian centers weighted by the posterior probabilities. Hence the Gaussian neurons added exclusively to model the noise will have low a posteriori probabilities and will not affect significantly the predictions (it is supposed that the noise is Gaussian, has zero mean and the extreme values are infrequent).

Another interesting conclusion drawn from this experiment is that both IGMN versions achieve the global optimum using  $\varepsilon_{max} = 0.05$  and approximately 30 neurons in each region, but using high  $\varepsilon_{max}$  values (e.g., 0.075) the multivariate version requires less neurons to achieve the same performance of the standard version. To understand why this occurs Figures 5.2 and 5.3 show the regression curves of the standard and multivariate versions of IGMN using  $\varepsilon_{max} = 0.075$  and  $\varepsilon_{max} = 0.025$ , respectively. In this figure, the Gaussian components are represented by ellipses whose width is equivalent to a Mahalanobis distance of two, and the mean of each distribution is shown by solid circles in the center of the respective ellipses. It can be noticed that IGMN creates some Gaussian units

with low variances (i.e., small ellipses) to model the noise in the data set, but as these units have low a priori probabilities (i.e., they represent few data samples) they do not influence significantly the approximation curves. As described in Chapter 4, these spurious components can be easily identified and deleted, but in these figures we rather kept all Gaussian units added by IGMN to better understand how the learning process occurs.

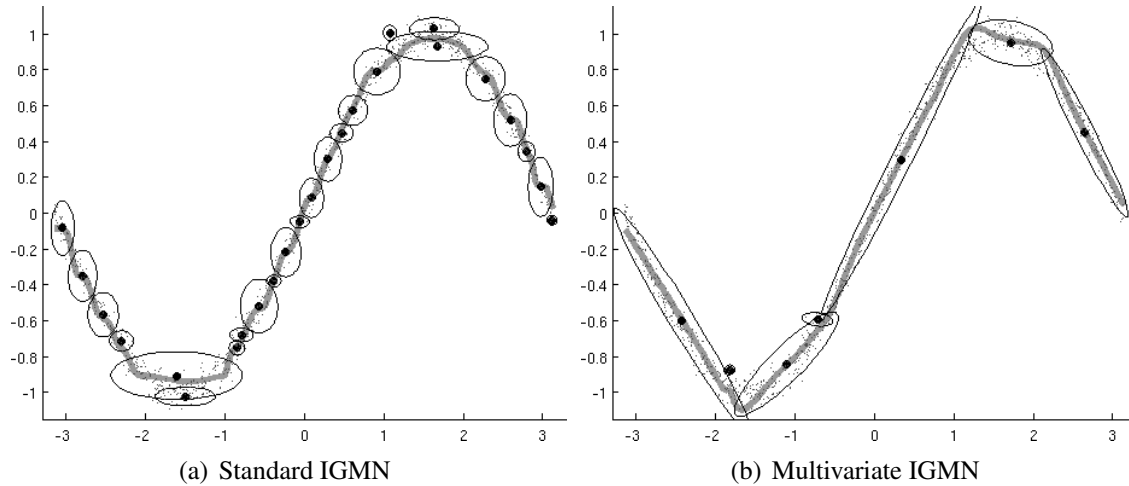


Figure 5.2: Experiments performed over the sinusoidal data set using  $\varepsilon_{max} = 0.075$

It can be noticed in Figure 5.2(b) that the multivariate algorithm can learn the target function using less neurons because the Gaussian units are aligned with the actual distribution of the training data set. However, if we want a more precise approximation IGMN will add more units to change the shape of the approximation curve. Hence if the  $\varepsilon_{max}$  is too small the benefit of having a multivariate representation is lost and consequently the number of Gaussian units will be similar for both IGMN versions. Nevertheless, a multivariate representation is still very useful in state spaces of more dimensions because it allows the representation of the target function using few units. Moreover, using the multivariate algorithm the predictions are not bounded by the minimum and maximum values of the training data set, as we will be described below in Section 5.5.

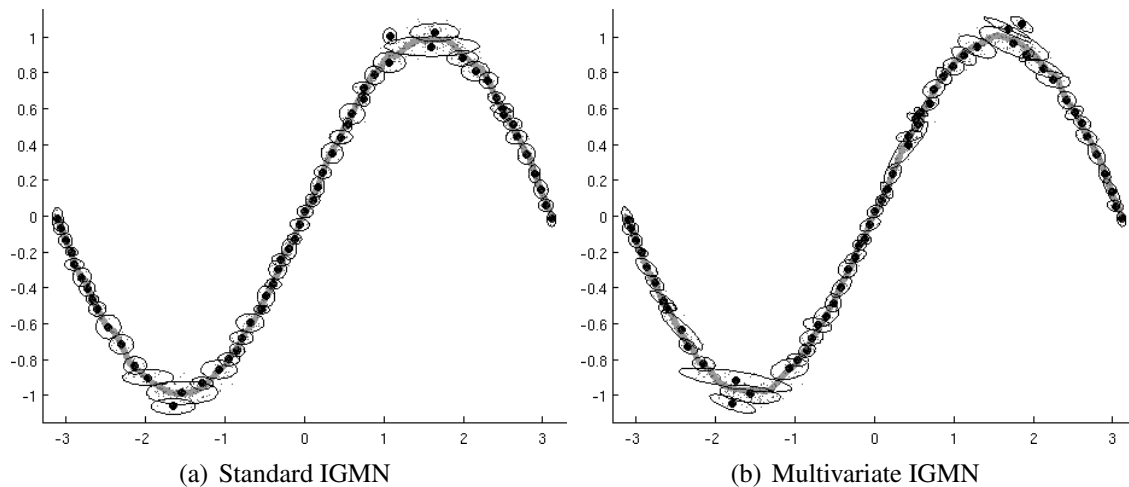


Figure 5.3: Experiments performed over the sinusoidal data set using  $\varepsilon_{max} = 0.025$

In relation to the time required for learning (columns *time* in Table 5.1) it can be noticed that both IGMN versions are very fast. In fact the multivariate version of IGMN

has learned the sinusoidal data set composed by  $N = 1000$  data samples in less than a second even using  $\varepsilon_{max} = 0.0075$ , i.e., using 173 multivariate neurons in each region. Of course in problems of higher dimensionality the multivariate version will require more time to invert the larger covariance matrices.

Although these results are very impressive, the NRMS error computed over the training data set is not a good estimator of the approximation error because it can be influenced by overfitting. Hence, this experiment was repeated using the 10-fold cross validation method (HAYKIN, 2008), in which the learning data set is randomly divided into 10 subsets. The learning process is repeated 10 times using a different subset for testing (i.e., to compute the generalization error) and the remaining 9 subsets for learning. The boxplot graphs<sup>1</sup> of Figure 5.4 show the results computed using the 10-fold cross validation procedure. The configuration parameters used in this experiment are  $\delta = 0.01$  and  $\varepsilon_{max} = 0.05$ .

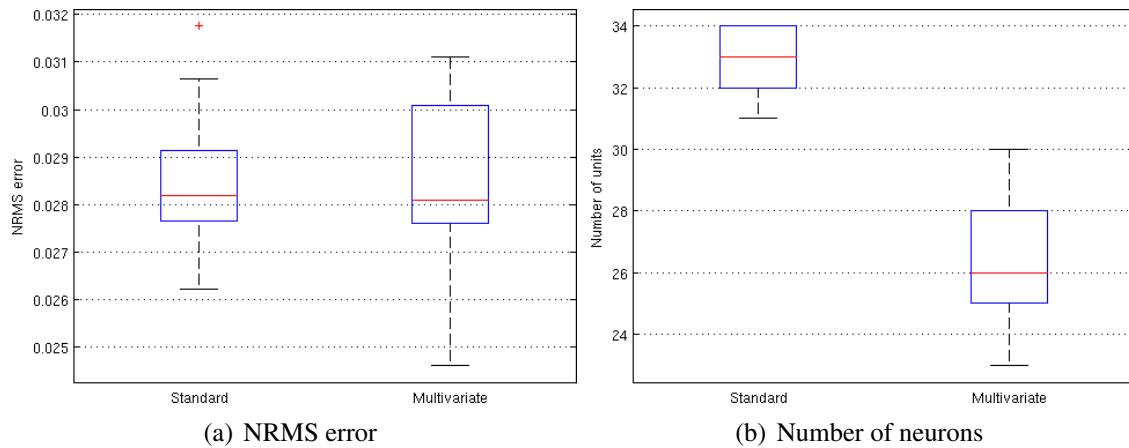


Figure 5.4: Comparison between standard and multivariate regression

It can be noticed in Figure 5.4(a) that the confidence intervals are overlapped, and therefore it is considered that both algorithms have the same generalization performance. The average NRMS error is 0.028379 for the standard algorithm and 0.028501 for the multivariate algorithm. Observing Figure 5.4(b), on the other hand, it can be noticed that the multivariate algorithm is really more parsimonious than the standard algorithm: the average number of neurons added during learning is 32.9 for the standard algorithm and 26.3 for the multivariate algorithm. Throughout this section we will still evaluate the performance of both versions of IGMN, but in the next sections we will use just the multivariate version of IGMN because its representational power is superior than that of the standard algorithm.

Another interesting conclusion that can be drawn from the experiment above is the fact that different implementations of IGMN (standard and multivariate) can approximate the target function with almost the same precision, using the same configuration parameters but different number of neurons in each region. This occurs due to the error-driven mechanism used by IGMN that simply adds so many neurons as necessary to achieve the specified approximation level, no matter what kind of neuron is used. This is a strong point of the current version of IGMN, because using the  $\tau_{nov}$  parameter to control the ap-

<sup>1</sup>A boxplot is a convenient way statistical tool for graphically depicting groups of numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile, median, upper quartile, largest observation (sample maximum) and outliers (MASSART et al., 2005).

proximation level each IGMN version would be carefully adjusted to achieve the desired performance. In fact the  $\tau_{nov}$  parameter is related to the shape and size of the Gaussian units, not to the approximation level of the target function. Of course in clustering tasks the number and the shape of the Gaussian distributions is essential, but in regression the goal is to achieve the required precision no matter how many neurons are used.

### 5.1.2 Assessing the sensibility of the $\delta$ parameter

In the previous experiments we have changed the  $\varepsilon_{max}$  parameter to see how IGMN is affected by this configuration parameter, but we have set  $\delta = 0.01$ . In this section we present some experiments in which the  $\varepsilon_{max}$  is set to 0.05 (its best configuration according to Table 5.1) and the  $\delta$  parameter is varied in order to assess its sensibility. Table 5.2 shows the results obtained in this experiment using both versions of IGMN and the 10-fold cross validation procedure.

Table 5.2: Assessing the sensibility of the  $\delta$  parameter

$\delta$	Standard		Multivariate	
	NRMS	$M$	NRMS	$M$
0.0010	0.042843	41.0	0.031196	70.7
0.0025	0.033888	65.8	0.028613	51.4
0.0050	0.028703	46.4	0.028325	34.7
0.0075	0.029878	34.4	0.027900	32.0
0.0100	0.026773	29.7	0.026554	27.0
0.0250	0.026055	29.7	0.025388	24.9
0.0500	0.027096	38.0	0.027186	43.7
0.0750	0.045344	170.2	0.039672	94.6

For better understanding these results, Figures 5.5 and 5.6 show the boxplot graphs computed over this experiment using the standard and multivariate versions of IGMN, respectively. It can be noticed that although extreme values of  $\delta$  can degrade the approximation and/or require many Gaussian units, using  $0.005 \leq \delta \leq 0.05$  the results are practically the same using both the standard and multivariate algorithms, and this indicates that IGMN is not sensible to this parameter. Moreover, as  $\sigma_{ini}$  is computed using  $\delta$  and the minimum and maximum values of each feature (Equation 2.35), then the  $\delta$  parameter is not affected by resizing and normalization procedures.

Actually the  $\delta$  parameter can always be set to 0.01 without problems, as will be demonstrated throughout this thesis. In the remaining experiments of this section  $\delta$  will be kept fixed at 0.01, but in Section 5.4 we will again vary  $\delta$  to demonstrate that it is really uncritical. Next section describes an experiment performed to verify if the order of presentation of data affects the results produced by IGMN.

### 5.1.3 Ordered $\times$ shuffled data sets

In the previous experiments the training data were always presented in ascending order of  $\mathbf{a}$ , i.e., the training data set was ordered. The next experiment is devised to verify if the order of presentation of data significantly affects the approximation computed by IGMN. Hence, we will perform two experiments using both versions of IGMN each, the first experiment presenting the training data in ascending order of  $\mathbf{a}$  (as in the previous experiments) and the second presenting the data in a random order. To simplify our ex-



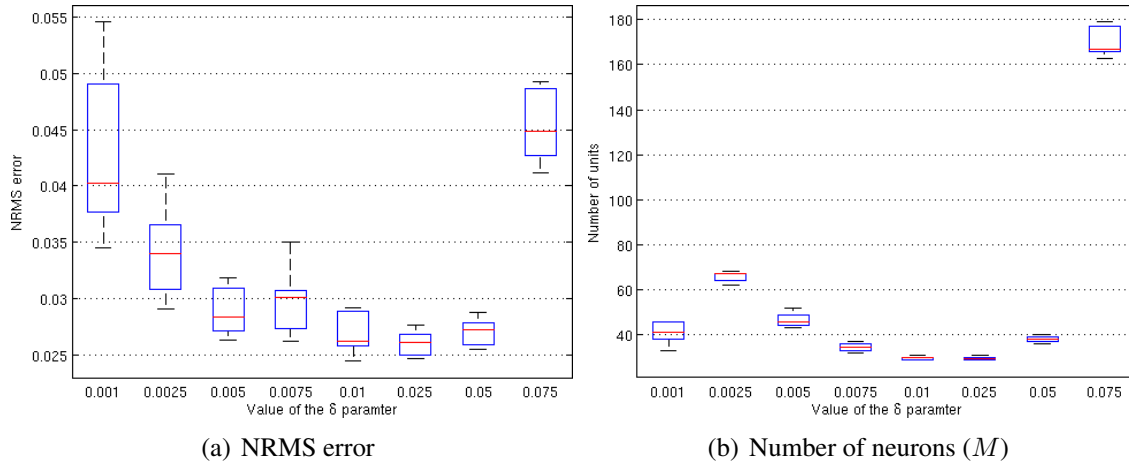


Figure 5.5: Assessing the sensibility of  $\delta$  – Standard IGMN algorithm

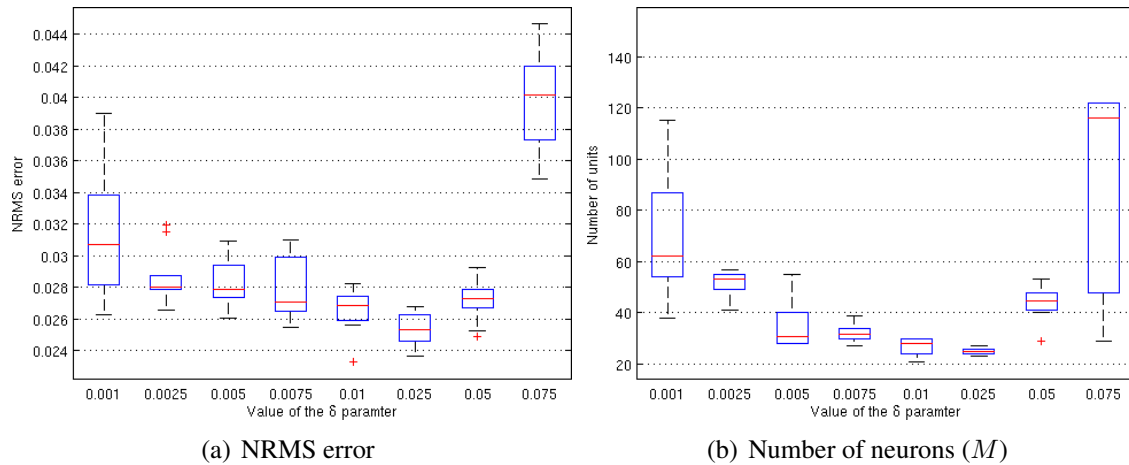


Figure 5.6: Assessing the sensibility of  $\delta$  – Multivariate IGMN algorithm

planation, we will use the term *ordered data set* to refer to the presentation of data in ascending order of  $a$  and *shuffled data set* to refer to the presentation of data in a random order, although the data samples themselves are the same in both data sets. Figures 5.7 and 5.8 show the results obtained in these experiments computed using the 10-fold cross validation procedure. The IGMN configuration parameters were set in these experiments to  $\varepsilon_{max} = 0.05$  and  $\delta = 0.01$ .

It can be noticed in Figure 5.7 that the order of presentation of data affects strongly the standard algorithm. In fact the NRMS error computed over the shuffled data set is almost twice the error computed over the ordered data set. Moreover, the number of Gaussian units added during learning is significantly larger when the shuffled data set is used. Observing the boxplot graphs shown in Figure 5.8, on the other hand, we can notice that the multivariate algorithm is not strongly affected by the order of presentation of data, i.e., the differences between the results computed over each data set are not statistically significant (the confidence intervals overlap).

Based on the boxplot graphs of Figure 5.8 we can affirm that the multivariate algorithm is not affected by the order of presentation of data when  $\varepsilon_{max} = 0.05$ , but we don't know if this robustness stands using other settings of  $\varepsilon_{max}$ . To answer this question this experiment was repeated using other  $\varepsilon_{max}$  settings, and the results computed using the 10-fold cross validation procedure are shown in Table 5.3. Based on these results we can

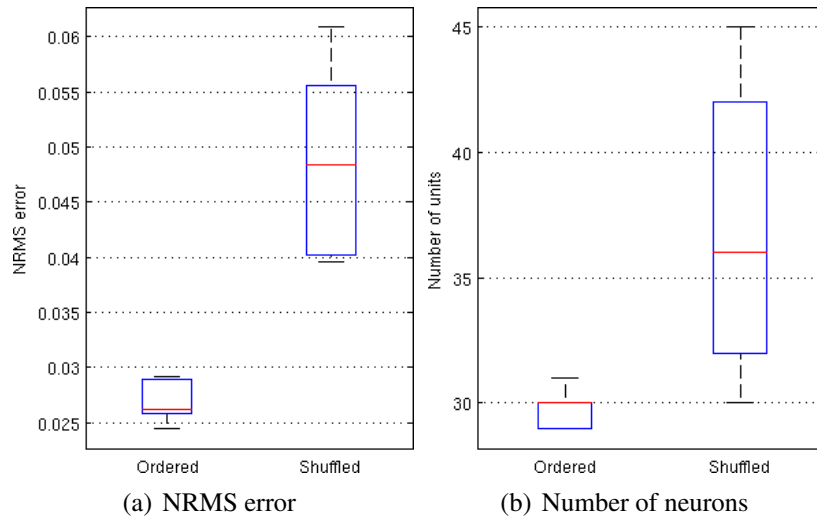


Figure 5.7: Ordered  $\times$  shuffled data sets – Standard algorithm

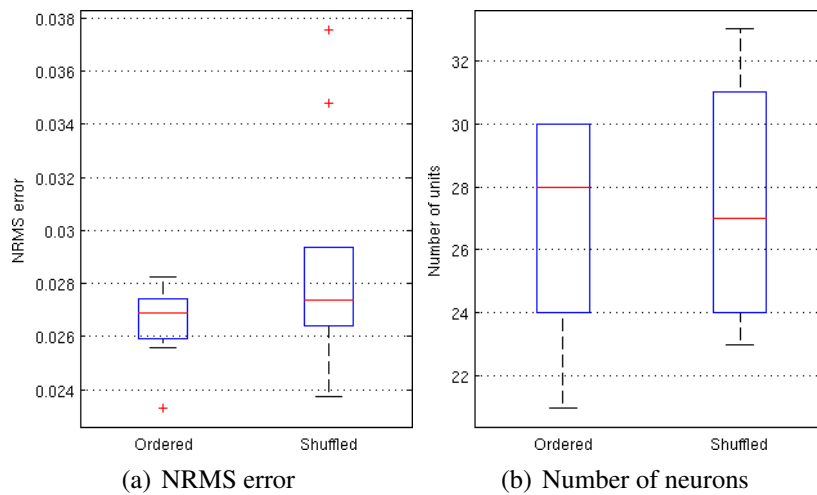


Figure 5.8: Ordered  $\times$  shuffled data sets – Multivariate algorithm

now affirm that the multivariate algorithm is really robust to the order of presentation of data, because as can be noticed in Table 5.3 the results (number of units added during learning and NRMS error) computed over both data sets are similar. But the standard IGMN algorithm, on the other hand, is really affected by the order of presentation of data, but the differences are reduced as the  $\varepsilon_{max}$  parameter is lowered.

The explanation why just the standard algorithm is affected by the order of presentation of data is the following. When the multivariate algorithm is used the Gaussian units rapidly assume the shape and size of the training data, and thus the state space is rapidly covered and clustered according to the actual distribution of these training data. Hence, after the presentation of some training patterns each data vector falls into the most likely cluster (i.e., a Gaussian distribution that is close to the actual distribution of data in that region of the state space) no matter in which order these training vectors are presented. Figure 5.9 illustrates this situation, where the Gaussian units obtained using the ordered and shuffled data sets are shown (in this experiment  $\varepsilon_{max}$  was set to 0.05). Observing this figure we notice that the Gaussian units added during learning are not exactly the same for both data sets, but the resulting approximation and the number of neurons are practically the same in both experiments.

Table 5.3: Comparison between ordered shuffled data sets

		Standard		Multivariate	
$\varepsilon_{max}$	Data set	NRMS	$M$	NRMS	$M$
0.075	Ordered	0.029947	22.6	0.031741	7.4
0.075	Shuffled	0.059026	27.6	0.029370	12.5
0.050	Ordered	0.026773	29.7	0.026554	27.0
0.050	Shuffled	0.048697	37.0	0.028602	27.7
0.025	Ordered	0.025927	54.9	0.025516	57.5
0.025	Shuffled	0.036486	59.5	0.025882	58.5

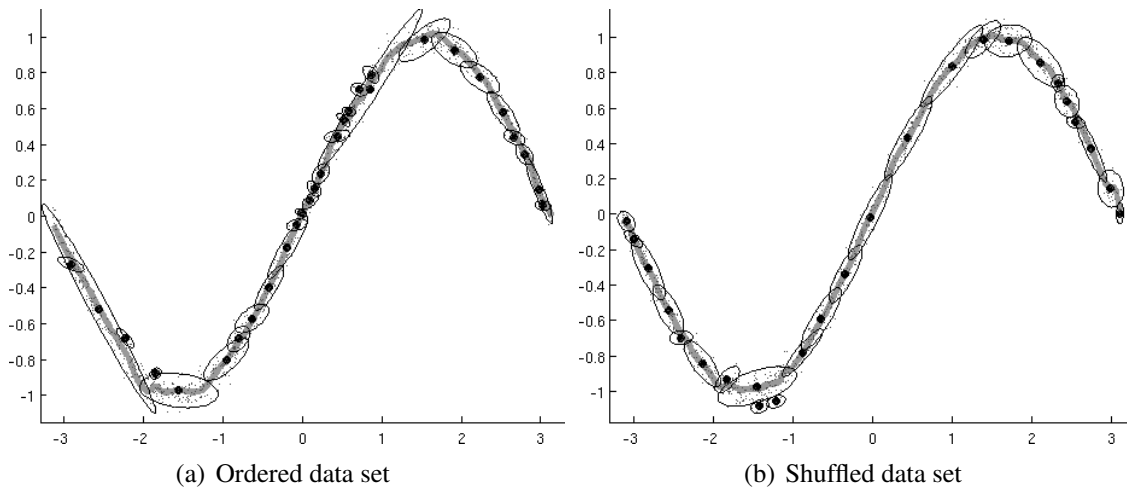


Figure 5.9: Gaussian units added during learning – Multivariate algorithm

Using the standard algorithm, on the other hand, many small clusters are created to model the regions of the state space where the target function is not parallel to the coordinate axis, as can be noticed in Figure 5.2(a) above. If the training data are presented in order the Gaussian units will be added evenly to cover the state space, as shows Figure 5.10(a). But if the training data is presented in a random order then many Gaussian units will be created irregularly, as shows Figure 5.10(b). We can notice in this figure that some units assume shapes and sizes that are not similar to the distribution of the training data. Moreover, many spurious components are created to model the noise and some distributions become very large, as is clearly seen in Figure 5.10(b). These irregular distributions created by the standard algorithm difficult the approximation of the target function, and therefore the results are worse when the training data is presented in a random order. Nevertheless, for the kind of application we are interested in (incremental function approximation) the training data generally arrive in a specific order (e.g., when the data consist on sensory readings of a mobile robot navigating through an environment), and therefore the standard IGMN algorithm can be used in these applications without problems.

Based on these experiments we conclude that the standard algorithm is significantly affected by the order or presentation of data, but this does not occur when we are using the multivariate algorithm, i.e., the multivariate algorithm is relatively robust to the order of presentation of data. As described in Chapter 4 the main drawback of the multivariate representation is that it has a higher computational cost which is bearable only if this representation provides a real benefit to the learning algorithm. But based on the experiments

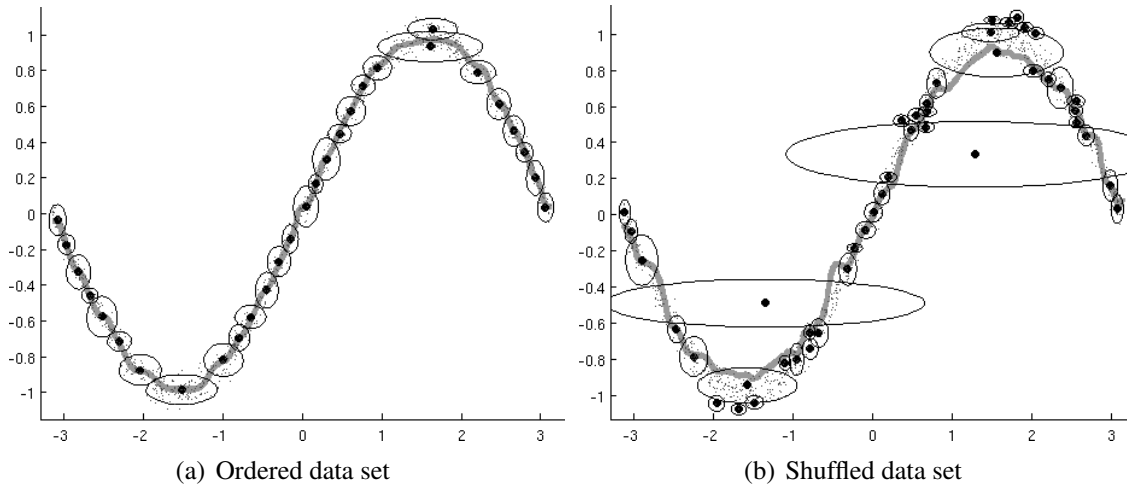


Figure 5.10: Gaussian units added during learning – Standard algorithm

described above we can affirm that the multivariate representation is really worthwhile. Hence, in the remaining of this thesis just the multivariate version of IGMN will be used.

#### 5.1.4 Estimating the confidence intervals

An advantage of IGMN over other connectionist approaches is that it can estimate not only the expected value of a sensory/motor modality (e.g.,  $b$ ) but also the variance/covariance structures at that point. This is illustrated in Figure 5.11(a), that shows the standard deviation computed by IGMN over the sinusoidal data set. The solid gray line in this figure represents the estimate of  $\hat{b}$  computed over the interval  $[-\pi, \pi]$ , and the dashed lines show a standard deviation above and below  $\hat{b}$ , i.e., the dashed lines are given by  $\hat{b} \pm \sigma_b^2$ , where  $\sigma_b^2$  is computed using (4.12). It can be noticed in Figure 5.11(a) that the standard deviation is larger in those points far from the center of the Gaussian distributions. Therefore using (4.12) we do not risk to underestimate the variance in the regions of the state space that are not well covered by the Gaussian units.

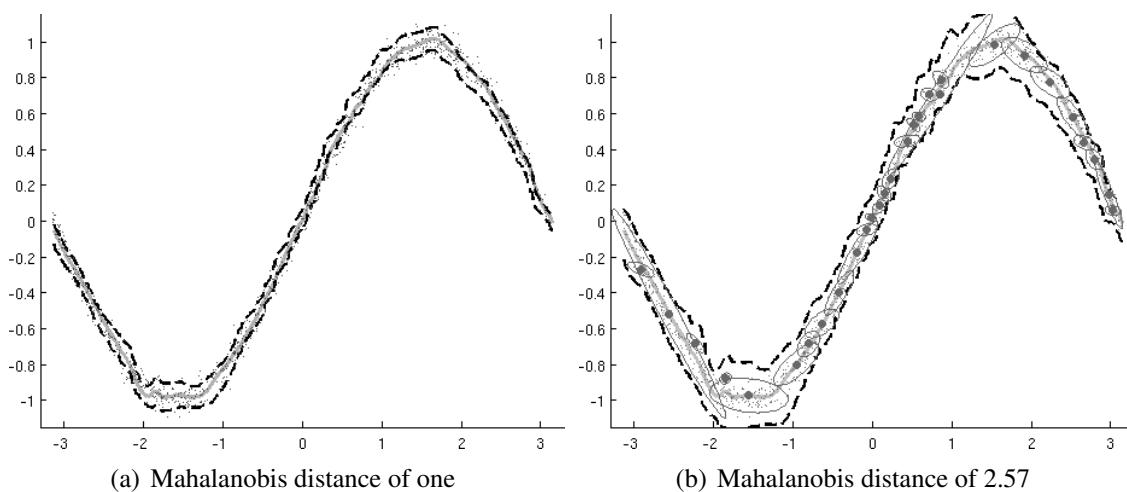


Figure 5.11: Confidence estimates computed by IGMN over the sinusoidal data set

Using  $\sigma_b^2$  it is also possible to compute the confidence intervals of the estimates, and thus IGMN can outcome not just a scalar  $b$  but also the confidence levels of its estimates. In the example above, if we want to know the region in which 99% of the sample data

will fall, for instance, we can plot contours with a Mahalanobis distance of 2.57, i.e.,  $\hat{b} \pm 2.57\sigma_b^2$ , as shows Figure 5.11(b). The confidence estimates computed by IGMN are very useful to make decisions, for instance, because they allow us to plan the actions using not just an estimate of  $b$  but also the confidence intervals. These confidence estimates are very useful in real time applications operating in hazardous environments, because in this kind of task we cannot take the risk of making a wrong decision. Next subsection presents a comparison among IGMN and other ANN models.

### 5.1.5 Comparison with other ANN models

This section presents some experiments performed to compare IGMN with other existing connectionist approaches. To perform a comparison using the sinusoidal data set we have selected the following ANN models:

- Multi-layer Perceptron (MLP) trained using the Resilient Propagation (RPROP) algorithm (RIEDMILLER; BRAUN, 1993);
- MLP trained by the Levenberg-Marquardt (LM) algorithm (HAGAN; MENHAJ, 1994), which is a second order method that learns faster than the standard back-propagation algorithm and RPROP;
- Radial basis functions (RBF) network (POWELL, 1985, 1987a,b) created using the ‘newrb’ method of the MATLAB Neural Network Toolbox (this method incrementally adds neurons until an specified *goal* is achieved);
- General regression neural network (GRNN) (SPECHT, 1991).

These neural network models were chosen because they are well known and available in the MATLAB Neural Network Toolbox software. Moreover, they are good representatives of the class of connectionist methods used for function approximation and prediction. Another possibility would be some of the improved GRNN models presented in Section 3.8, but as most part of these models just reduces the number of pattern units (i.e., the generalization level is not improved) we decided to use only the standard GRNN in these experiments. To make the comparison fair, we performed an exhaustive trial and error search to find out the best configuration of each ANN model listed below:

- MLP: a single hidden layer with 8 hidden neurons;
- RBF: goal (minimum MSE) = 0.0025 and spread ( $\sigma$ ) = 0.25;
- GRNN: spread ( $\sigma$ ) = 0.05;
- IGMN:  $\varepsilon_{max}$  = 0.05 and  $\sigma$  = 0.01.

The stop criterion used in these experiments for the MLP and RBF models was the early stopping criterion based on the best generalization epoch. Table 5.4 shows the results obtained in these experiments. The first column presents the ANN model. The following columns show, respectively, the number of hidden/pattern units, the NRMS error computed using the 10-fold cross validation procedure, the average number of training epochs and the time required to perform each replication, i.e. 1/10 of the time required by the entire 10-fold cross validation procedure. To facilitate our comparison, the last rows on Table 5.4 show the results obtained using both IGMN versions.

It is important to notice that the number of hidden/pattern units (the second column on Table 5.4) is just informative, because the ANN models are based on different principles (MLP is a global model, RBF has local receptive fields, GRNN is non-parametric, etc.) and therefore it is not possible to evaluate the model complexity (number of free parameters) based on this information. Similarly, the time required for learning (last column on

Table 5.4) is not a fair measure of the computational complexity because it depends on the implementation details of each architecture. Nevertheless, we can notice that IGMN is very fast, because even using full covariance matrices it performed each replication in less than 50 milliseconds.

Table 5.4: Comparative among ANN models using the sinusoidal data set

ANN model	Units	NRMS	Epochs	Time
MLP – RPROP	8.0	0.031384	107.1	5.444s
MLP – LM	8.0	0.024866	20.3	3.196s
RBF	24.5	0.025631	24.5	6.673s
GRNN	1000.0	0.025569	1.0	0.481s
IGMN – Standard	29.7	0.026773	1.0	0.028s
IGMN – Multivariate	27.0	0.026554	1.0	0.041s

Figure 5.12 shows a boxplot graph comparing the NRMS error obtained in each experiment. Observing this figure and Table 5.4 we can notice that all ANN models (excepting MLP – RPROP) have a similar performance (the confidence intervals overlap). Therefore, we can affirm that IGMN is a very good regression tool because it achieves a similar level of generalization using an incremental architecture and a single scan over the training data. Moreover, using IGMN we don't need to inform the number of hidden neurons neither to fine-tune many configuration parameters (actually it is much easier to configure IGMN than the other ANN models).

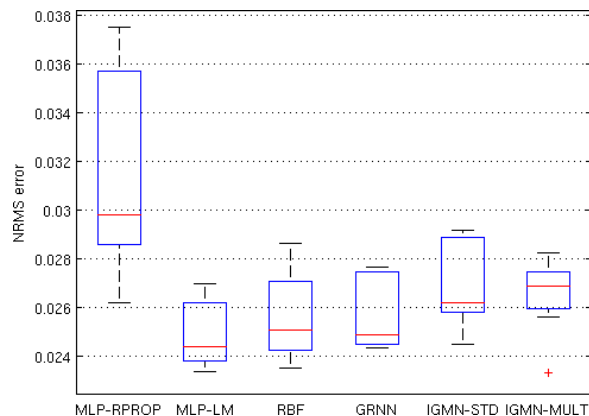


Figure 5.12: Comparative of the NRMS error among ANNs – sinusoidal data set

## 5.2 Approximating the “Mexican hat” function

In the previous experiments we have used a training data set composed by just two data features,  $a$  and  $b$ . The next experiment uses a three-dimensional data set composed by  $N = 5000$  data samples given by:

$$b = \sin(a_1)/a_1 \sin(a_2)/a_2, \quad (5.2)$$

were  $a_1$  and  $a_2$  are randomly chosen in the interval  $[-10, 10]$ . Figure 5.13(a) shows the target surface, also known as “Mexican hat”. The IGMN net used to learn this data set has two cortical regions:  $\mathcal{N}^a$ , composed by two data features ( $a_1$  and  $a_2$ ), and  $\mathcal{N}^b$ , composed

by a single feature (b). This data set was randomly divided into two subsets, called training and testing data sets, each of them composed by 2500 data samples. Moreover, the training data was presented to IGMN in a random order (i.e., the data set was shuffled).

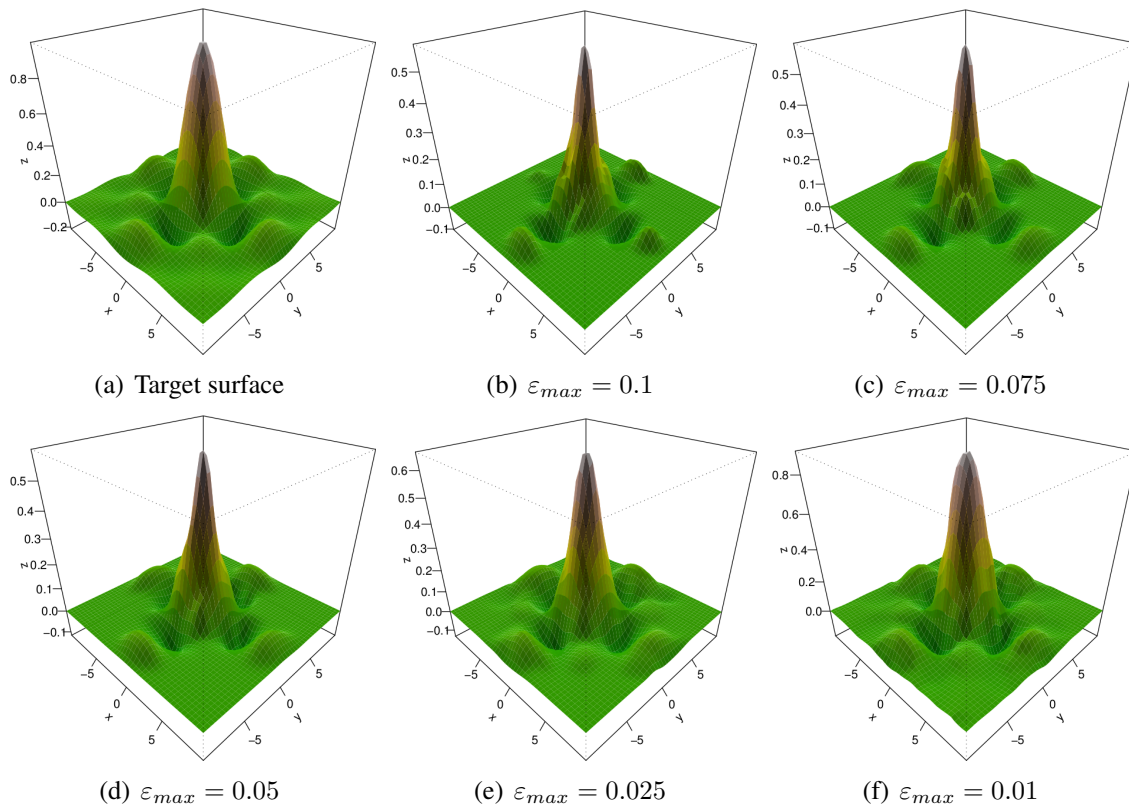


Figure 5.13: Approximating the “Mexican hat” function using IGMN

To assess the sensibility of IGMN to  $\varepsilon_{max}$  over this data set we have repeated this experiment using different configurations of  $\varepsilon_{max}$ , and the results obtained in this experiment are shown in Table 5.5. The first row shows the configuration of the  $\varepsilon_{max}$  parameter. The following rows show, respectively, the NRMS error, number of neurons added during learning ( $M$ ) and the learning time. The  $\delta$  parameter was kept fixed at 0.01, but the results are practically the same using  $\delta$  in the interval  $0.005 \leq \delta \leq 0.05$ .

Table 5.5: Approximating the “Mexican hat” function using IGMN

$\varepsilon_{max}$	0.1	0.075	0.05	0.025	0.01
NRMS	0.071302	0.067699	0.065824	0.051815	0.014288
$M$	19	25	37	83	184
Time	0.088s	0.104s	0.188s	0.656s	2.292s

Table 5.5 shows that the NRMS error is reduced as the  $\varepsilon_{max}$  parameter is decreased, whilst the number of neurons added during learning is increased. Using  $\varepsilon_{max} = 0.01$ , for instance, the NRMS error is just 0.014288, but 184 neurons are added to each region during learning. If  $\varepsilon_{max} \approx 0$  then the approximation error will be almost zero (remember that this data set is noise-free), but a huge number of neurons will be added during learning. Just for illustration purposes, Figures 5.13(b) to 5.13(f) show the surface approximated by IGMN using each  $\varepsilon_{max}$  setting. We can notice in these figures that using just 21 neurons

(Figure 5.13(b)) the proposed model is able to learn the general structure of the target surface, and using 184 neurons (Figure 5.13(f)) the approximated surface is quite good.

Using a MLP neural network trained with the Levenberg-Marquardt (LM) algorithm and 20 hidden neurons, the average NRMS error computed over the testing data set was 0.015265 (we needed to repeat this experiment 10 times due to the random initialization of the synaptic weights in a MLP network), and the time required for learning was 20.633 seconds. However, using MLP we had to perform an exhaustive search to find out the best number of hidden neurons. Using GRNN with  $\sigma = 1.0$ , on the other hand, the NRMS error was 0.017395 and the learning time was 0.189 seconds. As said before, the number of neurons added by GRNN is equal to the number of pattern neurons, i.e., 2500 pattern neurons in this case.

Based on this experiment we conclude that IGMN can learn a three-dimensional surface with arbitrary precision, and the number of neurons added during learning is proportional to the required approximation level. Next section describes how IGMN can be used to estimate both  $\mathbf{a}$  and  $\mathbf{b}$  using the same neural network.

### 5.3 Estimating both $\mathbf{a}$ and $\mathbf{b}$

In the previous experiments we have always used a specific sensory/motor signal (e.g.,  $a$ ) to predict another (e.g.,  $\hat{b}$ ), i.e., IGMN has learned and predicted just the *forward* function. However, we can use the same partially trained IGMN network (as IGMN can learn continuously we never consider that the training process has finished) to predict both the *forward* and the *inverse* function, i.e., to predict both  $\hat{\mathbf{b}}$  from  $\mathbf{a}$  and  $\hat{\mathbf{a}}$  from  $\mathbf{b}$  as well. To demonstrate this IGMN capability, initially we describe an experiment using a data set of size  $N = 1000$  generated by the following function:

$$\mathbf{b} = \tanh(\mathbf{a}) + \epsilon, \quad (5.3)$$

where  $\tanh(\cdot)$  is the hyperbolic tangent function,  $\epsilon$  is an independent Gaussian noise vector with mean  $\mu_\epsilon = 0$  and standard deviation  $\sigma_\epsilon = 0.025$  and  $\mathbf{a}$  is drawn from a uniform distribution in the interval  $[-e, e]$ , where  $e = 2.718281828$  is the Euler number. Figure 5.14(a) shows this data set, where the gray line represents the *target* function, i.e.,  $b = \tanh(a)$  without noise. An interesting point about this function is that both  $f(\cdot)$  and  $f(\cdot)^{-1}$  are mathematical functions in the interval  $[-e, e]$ .

This data set was learned by IGMN as usual:  $\Omega$  was set to 6,  $\varepsilon_{max}$  to 0.05,  $\delta$  to 0.01, and 6 Gaussian units were added during learning, as shows Figure 5.14(b). After learning this neural network was used to estimate  $\hat{\mathbf{b}}$  from  $\mathbf{a}$ , as shows the solid gray line on Figure 5.15(a). The estimates of  $\hat{\mathbf{b}}$  were computed by propagating  $\mathbf{a}$  into the cortical region  $\mathcal{N}^{\mathcal{A}}$ , computing  $p(j|\mathbf{a})$  in the association region  $\mathcal{P}$  and estimating  $\hat{\mathbf{b}}$  on region  $\mathcal{N}^{\mathcal{B}}$ .

We can notice in Figure 5.15(a) that the approximation performed by IGMN is quite good: the NRMS error computed over  $\hat{\mathbf{b}}$  is 0.013664, which is almost  $\sigma_\epsilon / [\max(\mathbf{b}) - \min(\mathbf{b})]$ , i.e., the *expected* NRMS error. Moreover, this same neural network can be used to estimate  $\hat{\mathbf{a}}$  from  $\mathbf{b}$  without retraining, as shows the gray line in Figure 5.15(b). We can notice in this figure that the approximation of the inverse function performed by IGMN is also good: the NRMS error computed over  $\hat{\mathbf{a}}$  is 0.033378. The main difference occurs at the top-right extremum, because in that region the distribution is almost parallel to the  $a$  axis.



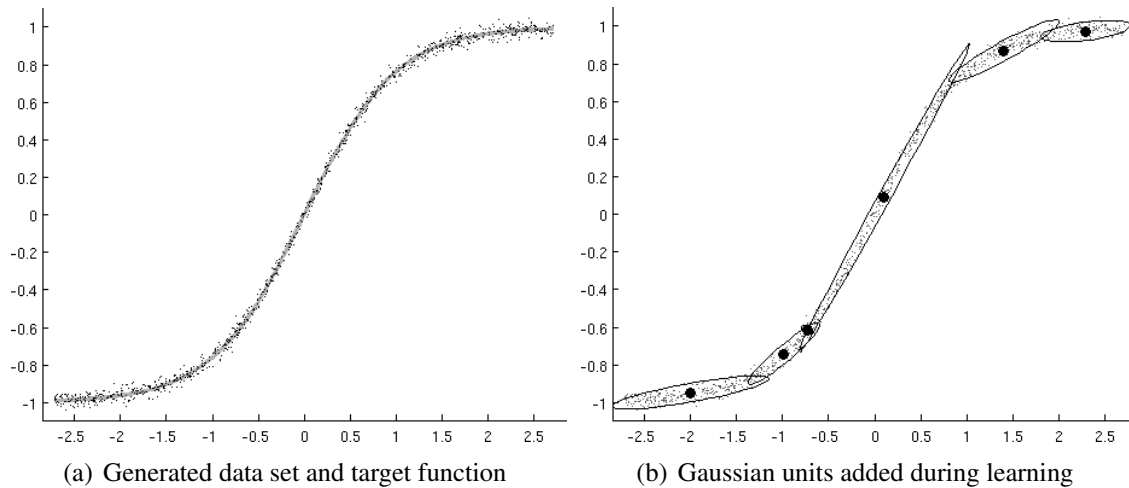
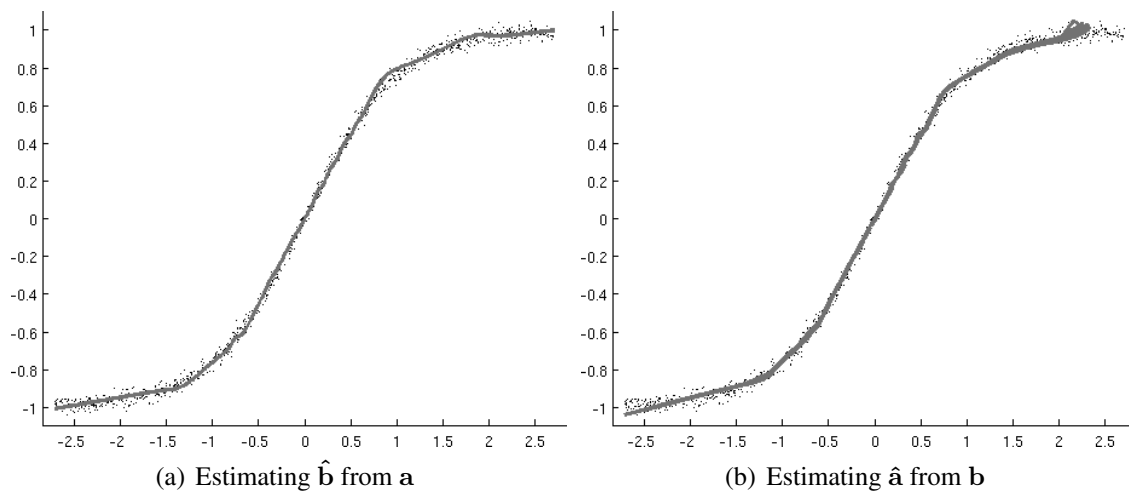


Figure 5.14: Hyperbolic tangent data set corrupted with noise

Figure 5.15: Estimating both  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  in a hyperbolic tangent data set

This experiment has demonstrated that IGMN can approximate both  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  if  $f(\cdot)$  and  $f(\cdot)^{-1}$  are functions in  $\mathbb{R}$ , but this constraint is not really necessary. If  $f(\cdot)^{-1}$  is a function in a more limited domain, for instance, IGMN can also predict  $\hat{\mathbf{a}}$  in the regions where  $f(\cdot)^{-1}$  is a mathematical function. To demonstrate this functionality we have used a data set composed by  $N = 500$  data samples generated using the following cubic function:

$$\mathbf{b} = \frac{\mathbf{a}^3}{5} + \frac{3\mathbf{a}^2}{4} - 3\mathbf{a} - 2 \quad (5.4)$$

where  $\mathbf{a}$  is randomly chosen in the interval  $[-8, 6]$ . Figure 5.16(a) shows this data set and the Gaussian units added by IGMN during learning. For estimating the forward function  $\mathbf{a} \rightarrow \hat{\mathbf{b}}$  in the domain  $\mathbb{R}$ , we can use any connectionist approach such as MLP, RBF, GRNN and IGMN, of course. Figure 5.16(b) shows the regression of  $f(\cdot)$  in the interval  $[-8, 6]$  performed by IGMN. The NRMS error computed over  $\hat{\mathbf{b}}$  is 0.009131, and the configuration parameters were set to  $\varepsilon_{max} = 0.025$ ,  $\delta = 0.01$  and  $\Omega = 6$  (actually in all experiments of this thesis  $\Omega$  was set to 6).

However, if we need to estimate the inverse function  $f(\cdot)^{-1}$  using a connectionist approach the procedure is more complicated, because  $f(\cdot)^{-1}$  leads to many solutions in

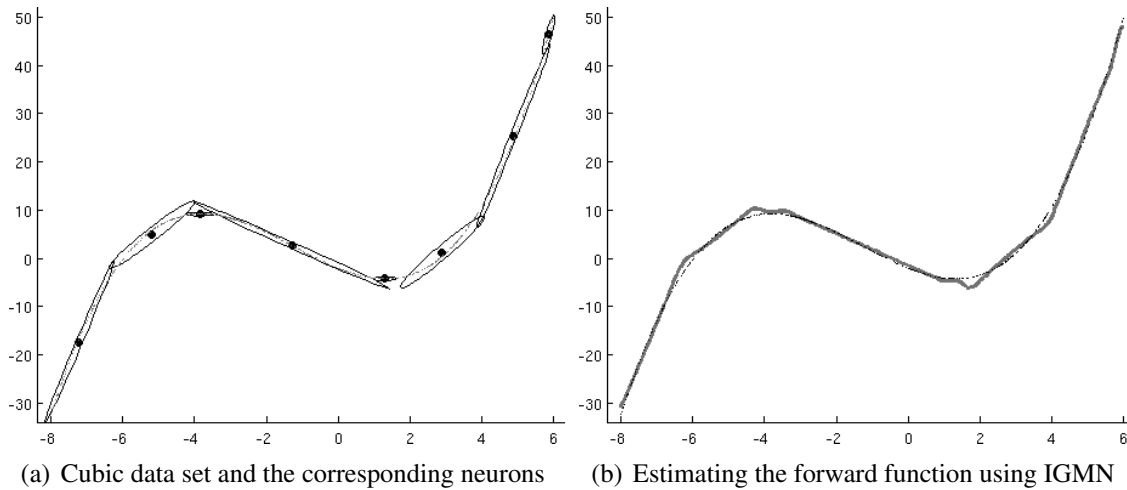


Figure 5.16: Experiments performed over the cubic data set

the interval  $[-4.2, 9.25]$ . If we use a connectionist approach such as MLP or GRNN to estimate the inverse function, the result will be similar to that presented in Figure 5.17(a), where the approximation is very poor in the interval  $[-4.2, 9.25]$ , i.e., the predictions are far from the target function in that region. Nevertheless, outside this interval the approximation of  $f(\cdot)^{-1}$  using a MLP is quite good, as can be seen in Figure 5.17(a).

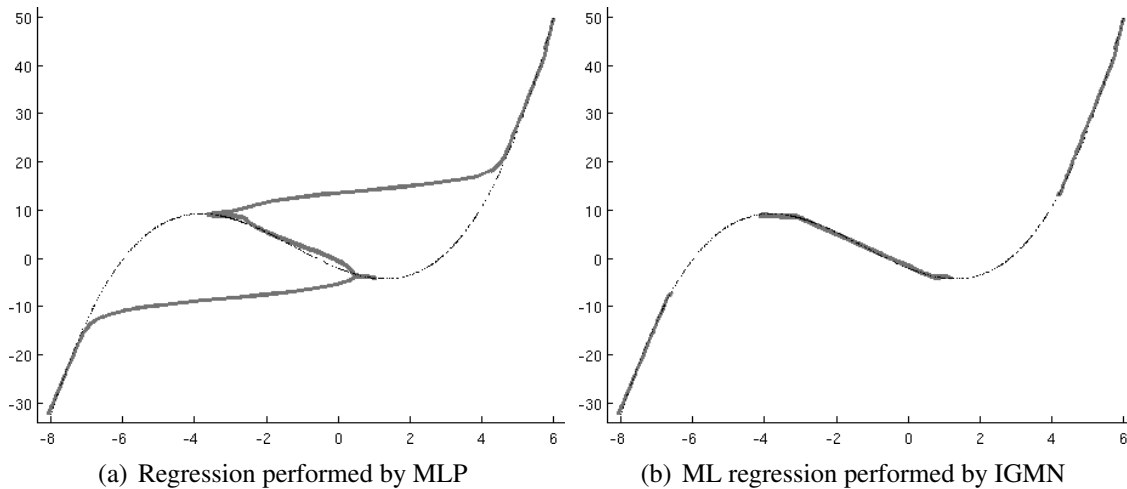


Figure 5.17: Estimating both  $\hat{a}$  from  $b$  in the cubic data set

Using a traditional ANN model such as MLP and GRNN this can be tackled only by restricting the domain in which the inverse function is evaluated, but using a more complex, high-dimensional data set, sometimes it is difficult to identify the regions where  $f(\cdot)^{-1}$  is a mathematical function. Using IGMN, on the other hand, the solution is straightforward: we use the approach described in Section 4.4 to identify this situation and tackle it appropriately. As described in Section 4.4, the solution for this problem can be: (i) to predict  $\hat{a}$  only in those regions where  $f(\cdot)^{-1}$  is a function; (ii) to return the most likely hypothesis in the conflict points and (iii) to return all possible hypotheses and the corresponding a posteriori probabilities. Figure 5.17(b) shows the approximation performed by IGMN using the second alternative, i.e., returning the most likely hypothesis of  $\hat{a}$ . According to Bishop (1995), the most likely hypothesis is a good solution in many

tasks such as control applications, where the forward function is easy to compute but the inverse function has multiple solutions.

It is important to notice that in IGMN there is no distinction between  $\mathbf{a}$  and  $\mathbf{b}$ , and therefore the solutions pointed out above can be used where either  $f(\cdot)$ ,  $f(\cdot)^{-1}$  or both are not mathematical functions. To illustrate this situation, Figure 5.18(a) shows a circular data set, where both  $f(\cdot)$  and  $f(\cdot)^{-1}$  lead to many solutions in the entire domain. This data set is composed by  $N = 500$  samples generated using the following equations:

$$\begin{aligned} a &= 50 \sin(2\pi n/N) + \epsilon \\ b &= 50 \cos(2\pi n/N) + \epsilon \end{aligned} \quad (5.5)$$

where  $n$  is uniformly taken in the interval  $[1, 500]$  and  $\epsilon$  is a independent Gaussian noise vector with mean  $\mu_\epsilon = 0$  and standard deviation  $\sigma_\epsilon = 0.25$ . Figure 5.18(b) shows the Gaussian units added by IGMN during learning (28 Gaussian units were added). We can notice that more units are added in those regions where the target function is almost parallel to a coordinate axis, because in those regions it is more difficult to predict the target function (they lead to infinite solutions).

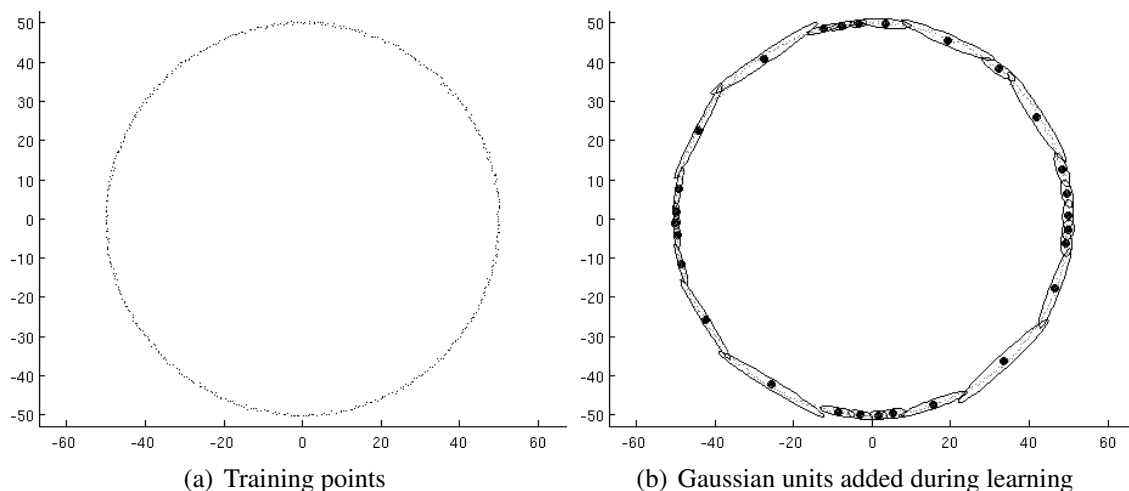


Figure 5.18: Circular data set used to evaluate IGMN using multiple hypothesis

Using the most likely hypothesis we can predict both  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  using IGMN, as show Figures 5.19(a) and 5.19(b), respectively. The IGMN configuration parameters used in this experiment are  $\varepsilon_{max} = 0.025$ ,  $\delta = 0.01$  and  $\Omega = 6$ . An interesting characteristic of this data set is that each training point has two possible solutions in both  $f(\cdot)$ ,  $f(\cdot)^{-1}$ , and consequently the a posteriori probabilities are almost the same ( $\approx 50\%$ ) for each hypothesis. Hence the ML hypothesis is chosen almost arbitrarily in this experiment (if they are equal the less recent hypothesis is chosen, but due to the numerical imprecisions a draw seldom occurs). However, if the training data set has different densities (number of training points) for each solution, then the hypothesis that represents more training points will be the most likely.

Another possibility pointed out above is to return all possible solutions, as show Figures 5.20(a) and 5.20(b), where the first solution is shown in dark tones and the second solution in light tones. It can be noticed in these figures that returning all possible solutions (just two in this case) practically all state space is covered. In fact, just those regions where the Gaussian distributions are almost parallel to the coordinate axis are not covered, which occurs because in those regions there are infinite solutions. When this occurs

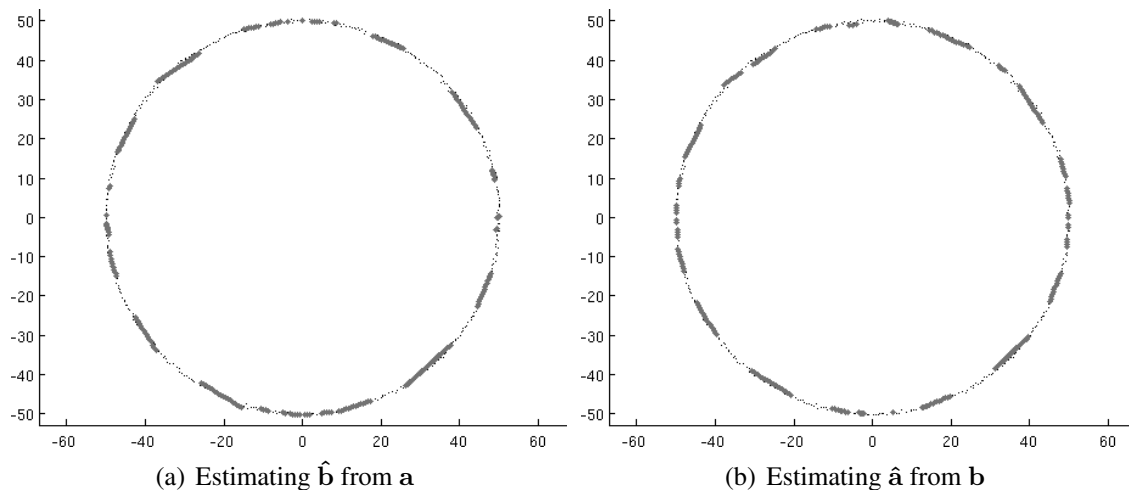


Figure 5.19: Estimating both  $\hat{\mathbf{a}}$  and  $\hat{\mathbf{b}}$  on the circular data set

IGMN returns the solution that is nearest to the center of the ML hypothesis, which is generally the best solution because it represent an “average answer”.

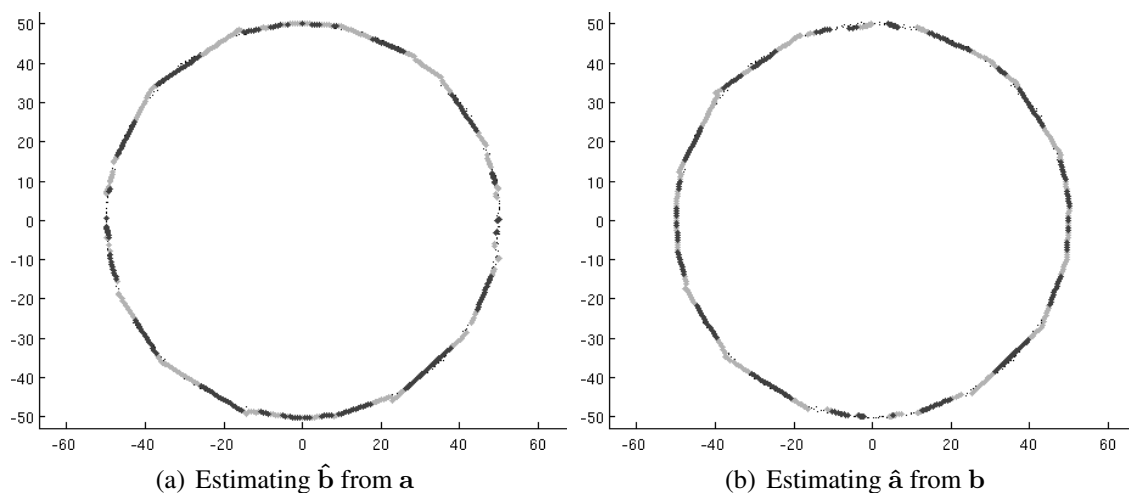


Figure 5.20: Returning multiple solutions for the circular data set

Of course if we need to take an action, for instance, we must decide what hypothesis is the most suitable for a given situation. As said before, in the circular data set both hypotheses are equiprobable, and hence the a posteriori probabilities do not help us to take a decision. But if we are using the data set shown in Figure 5.16(a), on the other hand, the ML hypothesis will be that associated with the highest “probability mass”, and therefore we will tend to choose the ML hypothesis because it represents the *most common* answer among the valid ones, i.e., the hypothesis that is more supported by the training data.

Based on these experiments we conclude that IGMN can be used to predict both the forward and the inverse functions even when they are not functions in the mathematical sense. Moreover, these predictions can be made using the same partially trained neural network, i.e., it is not necessary to have distinct neural networks for the forward and the inverse models. To the best of our knowledge IGMN is the first neural network model which has this capability, although in the context of Gaussian mixture models some possibili-

ties have been pointed out before by Bishop (1995) and Ghahramani and Jordan (1994a; 1994b) (but not using an incremental Gaussian mixture model).

## 5.4 Estimating the outputs of a nonlinear plant

The next experiment consists in identifying a nonlinear plant originally proposed by Narendra and Parthasarathy (1990) for the control of nonlinear dynamical systems using MLP neural networks. This plant is assumed to be of the form:

$$y_p(k+1) = f[y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)], \quad (5.6)$$

where  $y_p(k+1)$  is the next time sample of the plant output,  $y_p(k)$  is the current output,  $y_p(k-1)$  and  $y_p(k-2)$  are delayed time samples of the output,  $u(k)$  is the current input,  $u(k-1)$  is the previous input, and the unknown function  $f(\cdot)$  has the form:

$$f[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5] = \frac{\mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3 \mathbf{x}_5 (\mathbf{x}_3 - 1) + \mathbf{x}_4}{1 + \mathbf{x}_2^2 + \mathbf{x}_3^2}. \quad (5.7)$$

In Narendra and Parthasarathy (1990) this plant was identified using a MLP neural network composed by five neurons in the input layer, 20 neurons in the first hidden layer, 10 neurons on the second hidden layer and a single neuron in the output layer. This neural network was trained using the standard backpropagation algorithm for 100000 steps using a random input signal uniformly distributed in the interval  $[-1, 1]$  and a step size of  $\eta = 0.25$ . During the identification procedure (i.e., the learning process) a series-parallel model was used (Figure 5.21(b) (NARENDRA; PARTHASARATHY, 1990)), in which the output of the plant values (i.e., the *desired* answers) were used in the delay units  $y_p(k-1)$  and  $y_p(k-2)$ . After identification the performance of the model was assessed using a parallel model (Figure 5.21(a) (NARENDRA; PARTHASARATHY, 1990)), in which the actual neural network outputs are fed into the delay units. The identified plant (i.e., the trained neural network) was tested using the following function as input:

$$u(k) = \begin{cases} \sin(2\pi k/250) & \text{if } k \leq 500 \\ 0.8 \sin(2\pi k/250) + 0.2 \sin(2\pi k/25) & \text{if } k > 500 \end{cases} \quad (5.8)$$

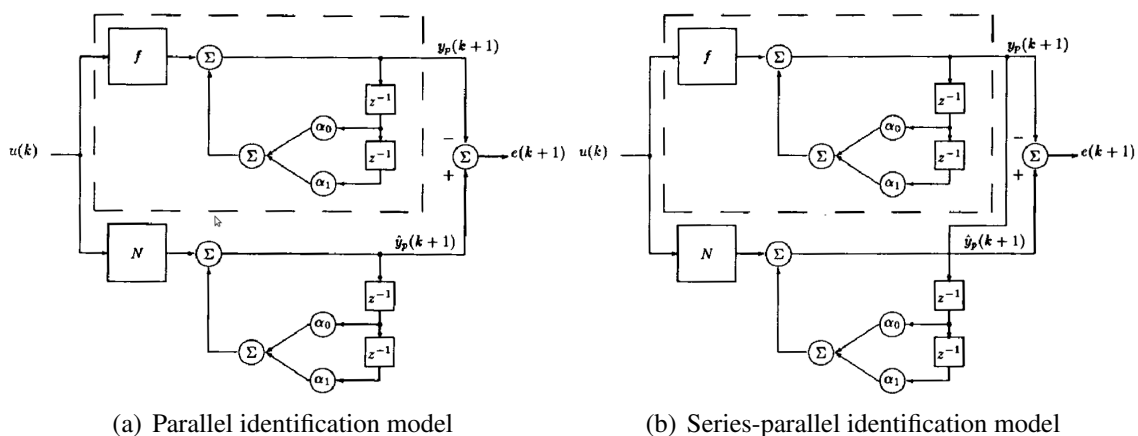


Figure 5.21: Identification models

Figure 5.22(a), reproduced from Narendra and Parthasarathy (1990), shows the results obtained in that experiment. It can be noticed that the Narendra's model was able to

identify this plant and to predict the next output  $y_p(k+1)$  even using a random input signal (rather than Equation 5.8) during learning, which shows that the neural network was able to learn the input/out behavior of the given plant after 100,000 learning steps.

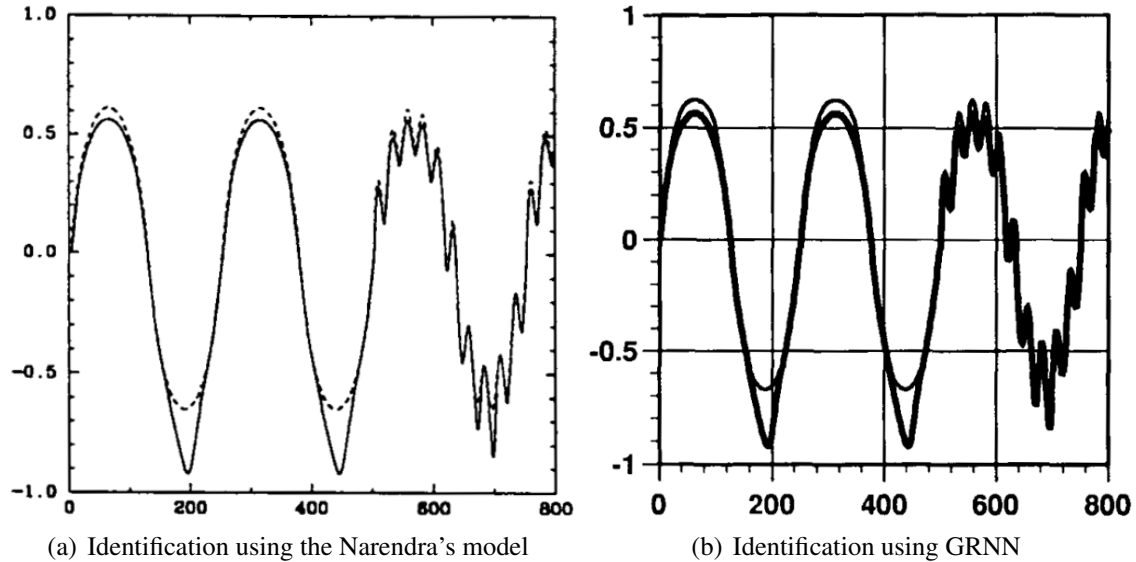


Figure 5.22: Identification of the nonlinear plant using some previous approaches

In Specht (1991) a GRNN network is used to approximate the function  $f(\cdot)$  above. In that experiment the identification procedure was carried out for 1000 time steps using a random input signal uniformly distributed in the interval  $[-1, 1]$ , and the spreading parameter was set to  $\sigma = 0.315$ . Figure 5.22(b), reproduced from Specht (1991), shows the results obtained in that experiment. Using GRNN 1000 pattern units were used to represent this plant (one for each random input signal used to identify the plant).

Unfortunately in Specht (1991) the approximation error computed using the testing function (5.8) is not informed, and thus we reproduced that experiment 10 times using the original conditions (a standard GRNN network with  $\sigma = 0.315$ ) and different random input signals in each replication. Figure 5.23(a) shows the best identification (i.e., the replication in which the NRMS error is the smaller) performed by GRNN. The averaged NRMS error computed over 10 replications was 0.053267, and 0.213 seconds were necessary to perform each replication.

To compare the performance of IGMN against the ANN models described above, we have repeated this experiment using the same conditions described above, i.e.:

- A series-parallel model (Figure 5.21(b)) for training and a parallel model (Figure 5.21(a)) for testing;
- The identification procedure was carried out for 1000 time steps using a random input signal uniformly distributed in the interval  $[-1, 1]$ ;
- The training data was presented in a random order (the training data set is shuffled);
- After learning the identified model was tested using inputs given by Equation 5.8;
- The whole experiment was repeated 10 times using different random input signals in each replication.

Figure 5.23(b) shows the identification performed by IGMN using default parameters, i.e.,  $\varepsilon_{max} = 0.05$  and  $\delta = 0.01$ . Observing Figure 5.23(b) we can notice that the identification performed by IGMN is superior than those performed by MLP and GRNN (Figures

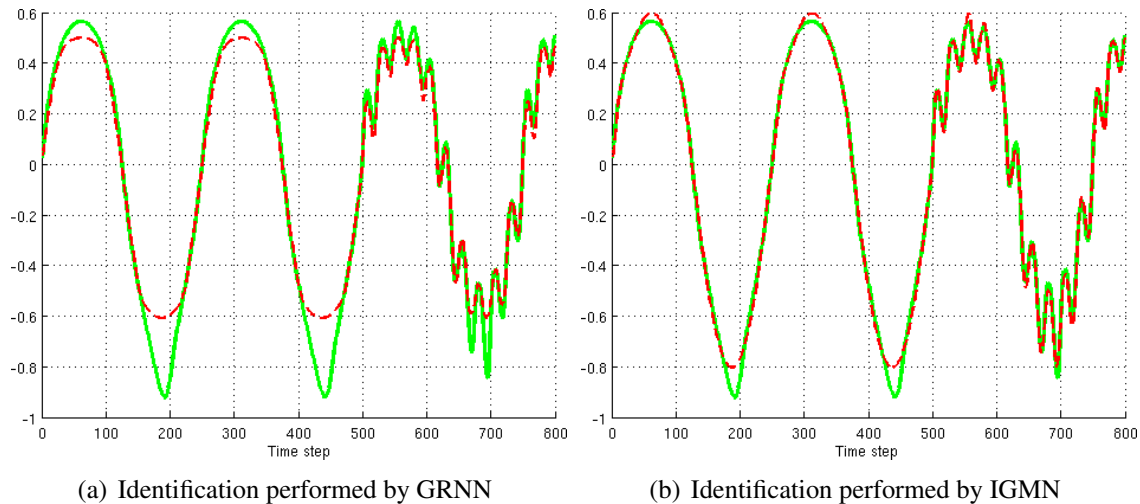


Figure 5.23: Identification of a nonlinear plant using 1000 training samples

5.22(a) and 5.22(b), respectively). The average NRMS error computed over 10 replications was 0.037, 19 neurons were added by IGMN during learning and 0.089 seconds were spent in each replication.

Table 5.6: Assessing the sensibility of IGMN to the  $\varepsilon_{max}$  parameter

Setting of $\varepsilon_{max}$	0.1	0.075	0.050	0.025	0.010
NRMS error	0.0474	0.0399	0.0370	0.0394	0.0304
Neurons ( $M$ )	8.30	12.60	19.10	38.10	80.40
Learning time	0.055s	0.089s	0.095s	0.125s	0.285s

To assess the sensibility of IGMN to the  $\varepsilon_{max}$  parameter, this experiment was repeated using  $\delta = 0.01$  and varying the value of the  $\varepsilon_{max}$  in the interval  $[0.01, 0.1]$ . Table 5.6 shows the results obtained in this experiment (averaged over 10 replications), and Figures 5.24(a) and 5.24(b) show the boxplot graphs of the NRMS error and the number of units added during learning, respectively. It can be noticed in these figures that  $\varepsilon_{max} = 0.05$  is a good choice, because it allows a good compromise between the model complexity (number of neurons) and the approximation level (NRMS error).

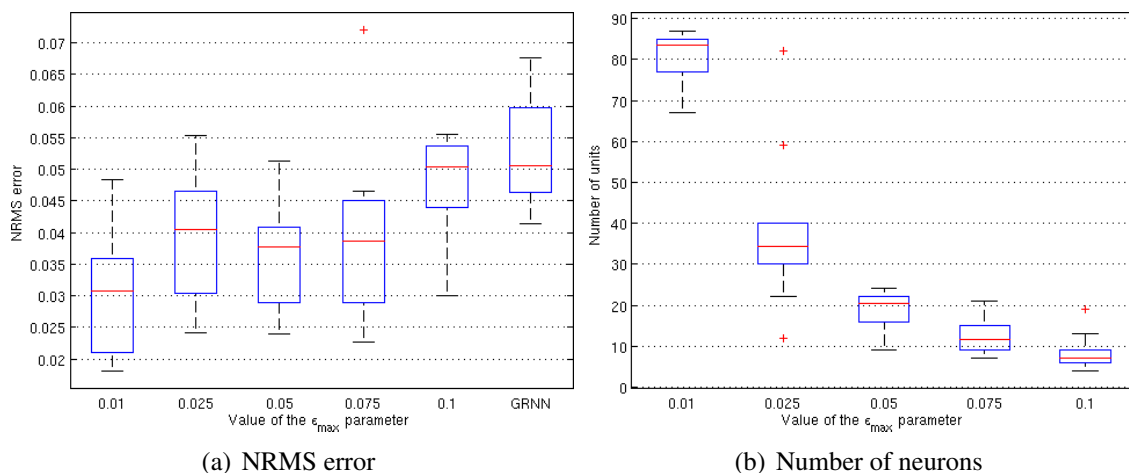


Figure 5.24: Assessing the sensibility of IGMN to the  $\varepsilon_{max}$  parameter

The next experiment aims to assess the sensibility of IGMN to  $\delta$  using this plant, where  $\varepsilon_{max}$  was kept fixed at 0.05 and  $\delta$  was varied in the interval  $[0.0025, 0.075]$ . Table 5.7 shows the average results obtained in this experiment, and Figures 5.25(a) and 5.25(b) show the boxplot graphs of the NRMS error and the number of units added during learning, respectively. It can be noticed that the IGMN performance is practically the same using any value of  $\delta$  in the interval  $[0.005, 0.05]$ , although using  $\delta = 0.025$  the NRMS error is slightly lower. Therefore we can affirm that IGMN is not sensible to the  $\delta$  parameter, and as we have noticed before (in Subsection 5.1.2) just extreme values of  $\delta$  degrade the IGMN performance and/or require many Gaussian units.

Table 5.7: Assessing the sensibility of IGMN to the  $\delta$  parameter

Setting of $\delta$	0.0025	0.005	0.0075	0.01	0.025	0.05	0.075
NRMS error	0.0415	0.0372	0.0373	0.0370	0.0320	0.0399	0.0373
Neurons ( $M$ )	18.60	20.20	19.20	19.10	24.70	24.40	38.80
Learning time	0.088s	0.091s	0.090s	0.089s	0.106s	0.105s	0.125s

In Specht (1991) it is also pointed out that results similar to those presented in Figure 5.22(b) can be obtained using just 100 training samples. To validate this affirmation, we repeated this experiment using  $N = 100$  training samples randomly chosen in the interval  $[-1, 1]$ . Using GRNN the NRMS error averaged over 10 replications was 0.090904, and using IGMN it was just 0.067239. Moreover, using IGMN the average number of neurons added during learning was just 3.6, whilst GRNN had used 100 pattern units in each replication. Based on these experiments we conclude that IGMN is a good tool for approximating nonlinear plants, because it allows a good generalization level using few Gaussian units and a single scan over the training data. Next section describes how IGMN can be used to predict future values in a time series.

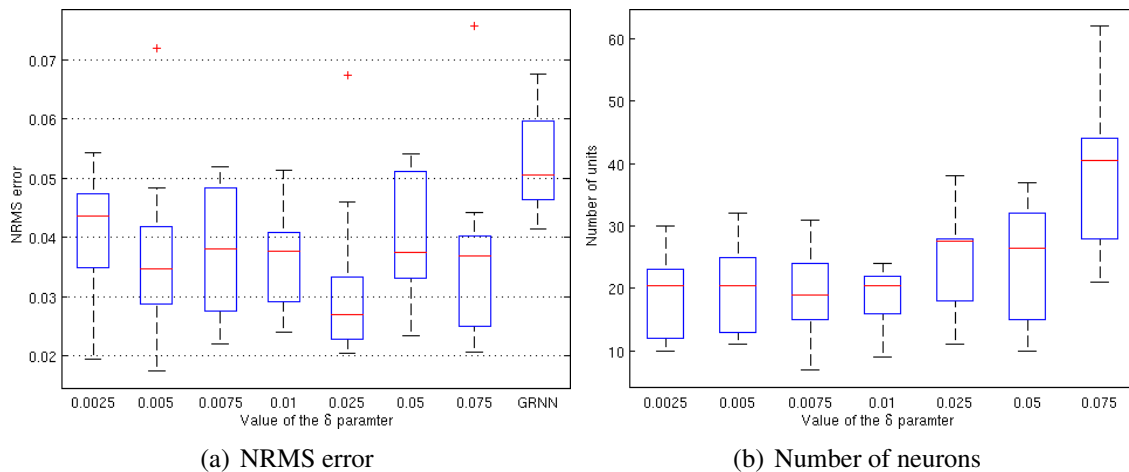


Figure 5.25: Assessing the sensibility of IGMN to the  $\delta$  parameter

## 5.5 Predicting future values of a time series

This section describes an experiment that consists in predicting the future values of a cyclic time series. More specifically, this experiment was performed using a known database of the number of airline passengers in the USA, originally published in Box



et al. (1976). The data consist of the monthly quantity of passengers measured over 12 consecutive years, between 1949 and 1960, summing up 144 samples. The goal of this experiment is the prediction of the number of passengers for the subsequent four years, i.e., the next 48 data records. This database has been focus of a competition for time series prediction in the 25th International Symposium on Forecasting<sup>2</sup>, held in 2005.

A critical decision in this experiment is the choice of the number of entries,  $D$ , to be used in the neural network. This decision is critical due to the reduced number of samples in the sample data set. As  $D$  increases, the number of training/testing samples decreases. The relationship between the number of samples and entries is given as follows:

$$Q = N - D,$$

where  $Q$  is the number of samples available for training. If  $D = 12$ , for instance, the records of the previous year,  $\mathcal{X} = \{x_{n-D+1}, x_{n-D+2}, \dots, x_{n-1}, x_n\}$ , are used to predict the next record  $x_{n+1}$  and  $Q = 132$  samples are available for learning and testing. In Camargo and Engel (2005) it is pointed out that using a MLP network the best possible configuration is given by using two hidden neurons and  $D = 48$  entries (i.e., the prediction is based on the previous four years), which results in 96 samples available for learning and testing. We have also performed several preliminary experiments using different configurations in a MLP network, and these experiments confirmed that the configuration pointed out above is the best possible for a MLP neural network trained using the Levenberg-Marquardt (LM) algorithm.

Using the best configuration pointed out above, i.e., a MLP network with 48 inputs, two hidden neurons and a single output, we have repeated this experiment 10 times using different random initializations, and the best replication is shown in Figure 5.26(a). It can be noticed in this figure that the MLP neural network does not learn correctly the growing tendency in the time series. Moreover, this is the only really good result obtained in the 10 replications. The most common results are similar to that shown in Figure 5.26(b), where the prediction of the next 48 values is poor. Besides, in some replications the results are similar to that present in Figure 5.26(c), where the neural network did not learn even the training data set. The time required for learning is about 2 seconds using the MATLAB NN Toolbox.

This experiment points out two well-known problems of the backpropagation algorithm and its variants (e.g., RPROP and LM): (i) the obtained results depend on the initialization of the random weights; and (ii) the learning algorithm is susceptible to local minima. Using less entries in the neural network, e.g.,  $D = 18$ , the results are even worse, as can be seen in Figure 5.26(d), that is the best replication using this configuration.

In Camargo (2010) a progressive enhancement neural model is used to select the most relevant features and to retrain the neural network using just these features. Figure 5.27, reproduced from Camargo (2010), shows the results obtained by Camargo using this technique. The results shown in Figure 5.27 were sent to the competition for time series prediction in the 5th International Symposium on Forecasting, and was awarded among the three best works in this competition (CAMARGO; ENGEL, 2005). Unfortunately the Camargo's technique is not on-line nor incremental: it needs to retrain the neural network many times using a batch-mode algorithm.

In the next experiment IGMN was used to predict the subsequent 48 values of this time series. In this experiment we have used the number of entries pointed out above ( $D = 48$ ),

<sup>2</sup>25th IFS, San Antonio, USA – [http://www.upcomillas.es/presim/documentos/recu\\_09.pdf](http://www.upcomillas.es/presim/documentos/recu_09.pdf)

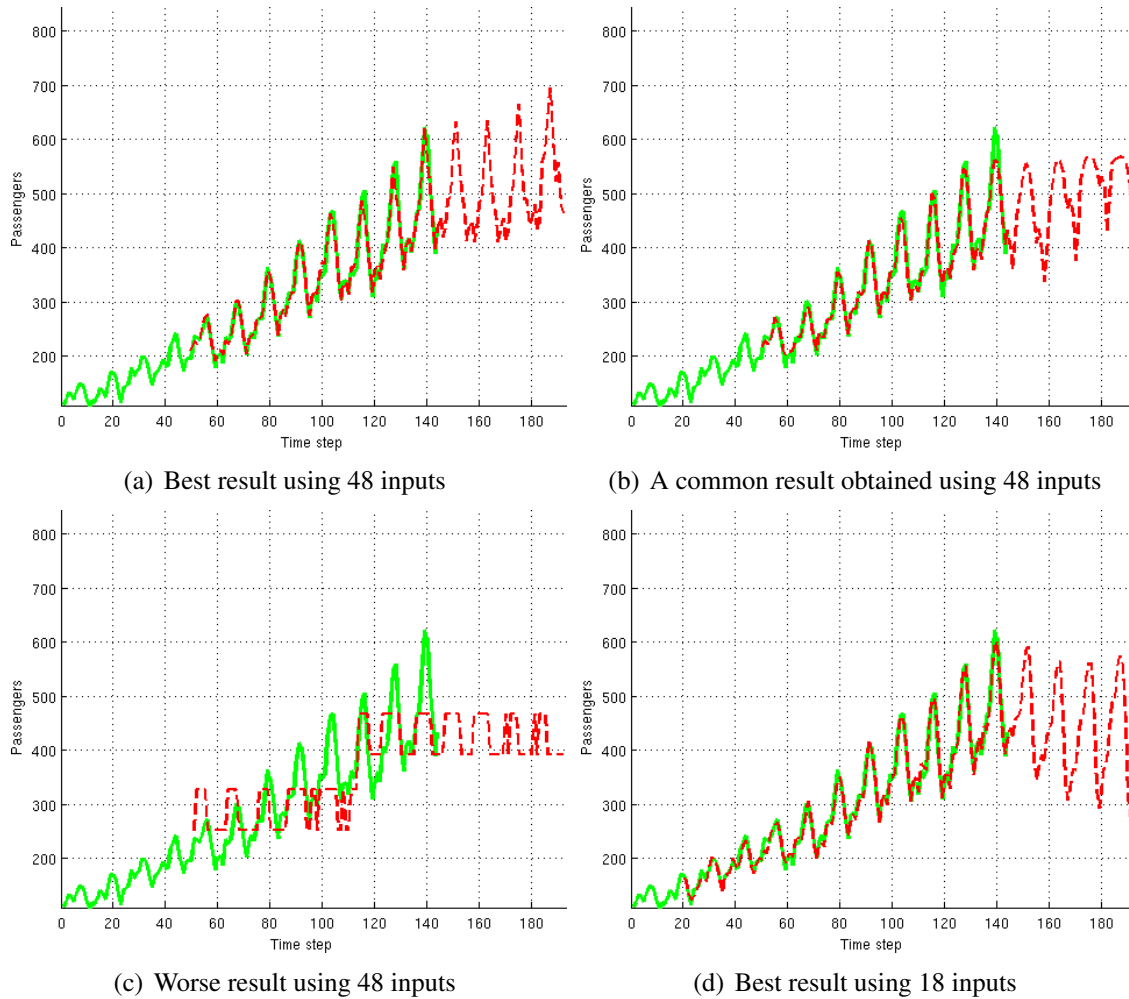


Figure 5.26: Predicting a time series using MLP

and hence the cortical region  $\mathcal{N}^A$  has  $D^A = 48$  entries and  $\mathcal{N}^B$  has just one entry. The IGMN configuration parameters were set to  $\varepsilon_{max} = 0.1$ ,  $\delta = 0.01$ . Figure 5.28(a) shows the results obtained in this experiment. The time required for learning is 0.05 seconds and two neurons were added in each region. If we use a lower setting on  $\varepsilon_{max}$ , the predictions will be almost the same presented in Figure 5.28(a), but more neurons will be added to each region during learning. Using  $\varepsilon_{max} = 0.01$ , for instance, 23 neurons are added in each region by IGMN.

Although it is not possible to say what is the best prediction (the number of passengers for the subsequent four years is not available), we can notice in Figure 5.28(a) that IGMN has learned correctly the growing tendency of the time series. Moreover, IGMN learns incrementally using a single scan over the training data, does not require a fine-tuning of its configuration parameters, does not depend on the initial conditions and is not susceptible to local minima. In fact, the results produced by IGMN are always the same using the configuration pointed out above, which is a strong advantage over MLP networks.

In the next experiment we have changed the number of entries in the neural network to verify if IGMN is able to predict the time series using less entries, i.e., based on a short period of time. This experiment indicated that using just 18 entries IGMN is able to make good predictions, as shows Figure 5.28(b). In this experiment just one multidimensional neuron was added to each region, and the learning time was just 0.01 seconds. Observing

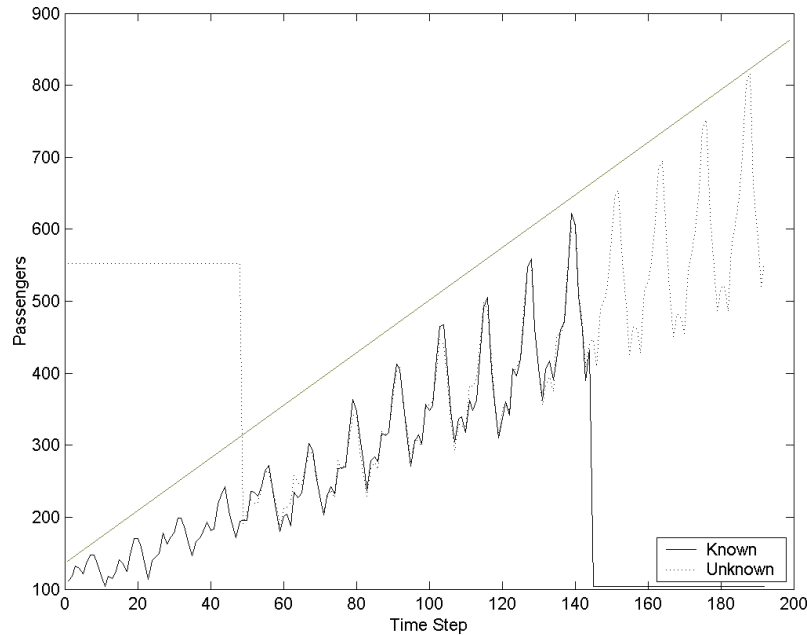


Figure 5.27: Predicting the time series using Camargo's model

Figure 5.28(b) we can notice that IGMN correctly predicts the time series based on the last 18 months with an accuracy higher than that provided by MLP (Figure 5.26(d)).

After this we have used GRNN to learn this time series and to compare its performance against IGMN. After an exhaustive search for the best adjusting of the spreading parameter  $\sigma$ , we have found out that the best setting is given using  $\sigma = 0.075$ . The training data were also normalized in the interval  $[-1, 1]$ , which is an requirement of GRNN because the Parzen kernels have the same width in each direction (SPECHT, 1991). Figures 5.29(a) and 5.29(b) shows the results obtained using a GRNN net with 48 and 18 entries, respectively.

We notice in Figure 5.29 that even using the best possible configuration GRNN cannot learn the growing tendency of the time series, and using other configurations and/or without normalizing the training data the results are even worse. This occurs because according to Specht (1991) in GRNN “the estimate is bounded by the minimum and max-

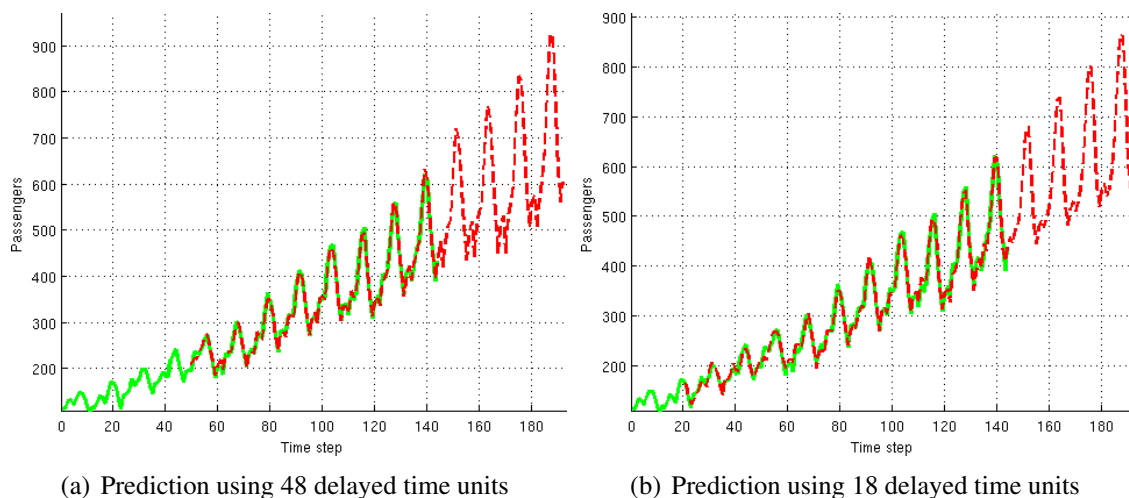


Figure 5.28: Predicting the time series using IGMN

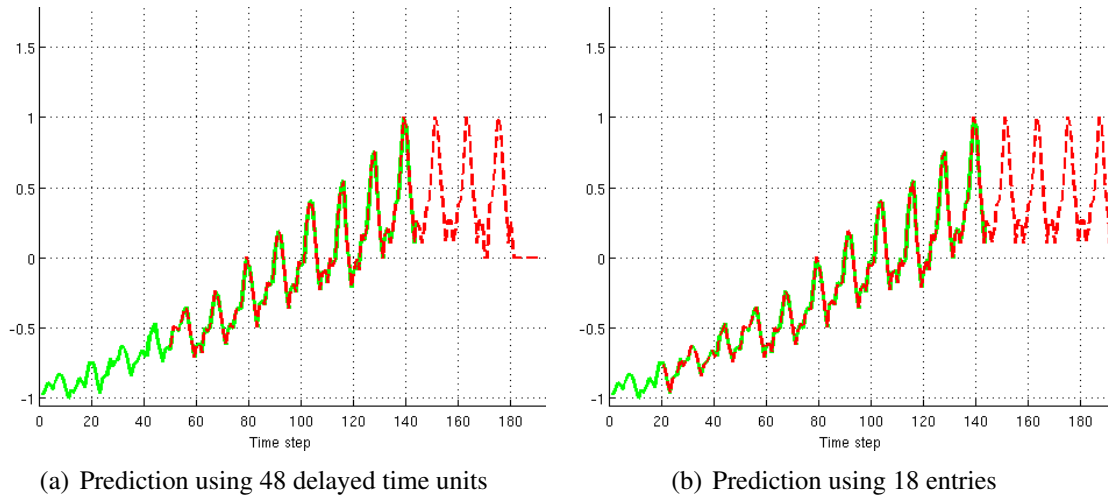


Figure 5.29: Predicting the time series using GRNN ( $\sigma = 0.075$ )

imum of the observations”, which prevents it of learning the growing tendency in the time series. Similarly, if we use IGMN with the standard algorithm, a result similar to that presented in Figure 5.29(a) is obtained, no matter how long time we spend trying to adjust the configuration parameters. This experiment points out that only using multivariate Gaussian units we can predict correctly this time series. Therefore, we conclude that the representational power of the multivariate algorithm really provides a real benefit to the neural network, and as we have pointed out before in Section 5.1 the higher computational cost of the multivariate representation is worthwhile because it allow us to predict a cyclic time series with growing tendency better than other ANN models.

Another important aspect about predicting time series is that sometimes the prediction itself is not enough. In fact, as the behavior of many systems is chaotic, it is very useful to compute the confidence limits of the forecasts, because they may allow us to make a better decision based on the confidence intervals, for instance. As described before, predicting the confidence limits using IGMN is straightforward – we just need to compute the variance at  $\hat{b}$  using (4.12) and use this variance to compute other estimators such as the confidence intervals. Figures 5.30(a) and 5.30(b) show the 95% confidence intervals computed by IGMN over the time series described above using 48 and 18 entries, respectively. We can notice that both experiments lead to similar results, but using 48 entries the confidence interval is narrow, what indicates that the confidence is higher when more information is used (i.e., the past experience is more deeply exploited by the neural network) to predict the time series.

Based on these experiments we can conclude that IGMN is a very good tool for learning time series, because it allow us to predict a cyclic time series in a robust and efficient manner. Moreover, it captures correctly the growing tendency of the time series and is not susceptible to local minima. Next section presents some final conclusions about the experiments described in this chapter.

## 5.6 Final remarks

This chapter has presented several experiments performed using the proposed neural network model in function approximation and prediction tasks. These experiments have exemplified the main characteristics of IGMN, hence showing that it is a very suitable ma-

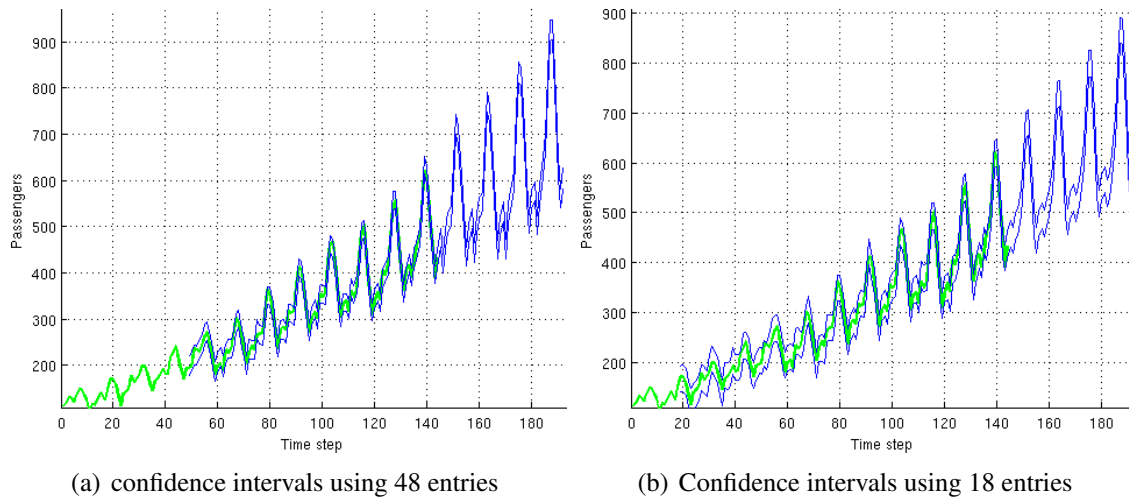


Figure 5.30: 95% confidence intervals computed by IGMN

chine learning tool for on-line prediction and regression tasks. In fact, the results obtained in these experiments have demonstrated that: (i) IGMN learns very fast using a single scan over training data; (ii) it does not depend on the initial conditions nor is susceptible to local minima (the results are always identical for the same data set and configuration); (iii) the neural network architecture is incrementally defined; (iv) it is not necessary to fine-tune the configuration parameters to obtain good results; (v) the multivariate representation is robust to the order or presentation of data; (vi) the predictions are not bounded by the minimum and maximum values of the training data set; (vii) IGMN can provide the confidence levels of its estimates; (viii) the same neural network can be used to estimate both the forward and the inverse functions; (ix) IGMN can provide good estimates even when either  $f(\cdot)$  or  $f(\cdot)^{-1}$  are not mathematical functions; (x) its performance in the experiments described above is equivalent or even superior than those presented by other connectionist solutions such as MLP and GRNN; (xi) the learning algorithm is relatively robust to the overfitting problem; and (xii) IGMN can be successfully used in many problems such as estimating the outputs of a nonlinear plant or predicting the future values of a time series with growing tendency. Next chapter presents some experiments in which IGMN is used in more complex machine learning tasks such as reinforcement learning and robotic control.



## 6 ROBOTICS AN OTHER RELATED TASKS

Last chapter has described many experiments, using artificial (synthetic) datasets, that demonstrated the representational power and learning performance of IGMN. This chapter describes the second contribution of this thesis, which is the use of IGMN in practical applications such as reinforcement learning and robotics. The main goal of these experiments is to validate the proposed model in real applications and/or using real data and also to demonstrate its suitability in many potential applications that require incremental learning and real time performance. This chapter is structured as follows. Section 6.1 discusses the use of IGMN for incremental concept formation (ENGEL; HEINEN, 2010a,b; HEINEN; ENGEL, 2010e,f), which is an important task in machine learning and robotics. Section 6.2 shows how IGMN can be used to compute the control actions for a mobile robot performing a wall following behavior (HEINEN; ENGEL, 2010a,b,d). Section 6.3 demonstrates the suitability of IGMN for computing the inverse kinematics in a gait control task using a simulated legged robot (HEINEN; OSÓRIO, 2009, 2008, 2007b). Section 6.4 presents the use of IGMN as a function approximator in reinforcement learning (RL) algorithms (HEINEN; ENGEL, 2010g,a, 2009a,c). Finally, Section 6.5 describes a new feature-based mapping algorithm (HEINEN; ENGEL, 2011, 2010c,f), based on IGMN, which represents the environment using multivariate GMMs rather than grid cells or line segments. Great part of these experiments has been published in the corresponding papers cited above using a previous (i.e., the standard) IGMN version, but in this chapter we have performed these experiments again using the most recent (i.e, the multivariate) IGMN algorithm, which leads to better results than those presented in the papers mentioned above.

### 6.1 Incremental concept formation

One of our primary motivations in developing IGMN was to tackle problems like those encountered in autonomous robotics. To be more specific, let us consider the so called perceptual learning, which allows an embodied agent to understand the world (BURFOOT; LUNGARELLA; KUNIYOSHI, 2008). Here an important task is the detection of concepts such as “corners”, “walls” and “corridors” from the sequence of noisy sensor readings (e.g., sonar data) of a mobile robot. The detection of these regularities in data flow allows the robot to localize its position and to detect changes in the environment (THRUN; BURGARD; FOX, 2006).

Although concept formation has a long tradition in machine learning literature, in the field of unsupervised learning, most methods assume some restrictions in the probabilistic modeling (GENNARI; LANGLEY; FISHER, 1989) which prevent their use in on-line tasks. The well known *k-means* algorithm (MACQUEEN, 1967; TAN; STEINBACH;

KUMAR, 2006) (Section 2.2), for instance, represents a concept as a mean of a subset or cluster of data. In this case, each data point must deterministically belong to one concept. The membership of a data point to a concept is decided by the minimum distance to the means of the concepts. To compute the means, all data points belonging to every concept are averaged using a fixed number of concepts along all the learning process. For learning probabilistic models, a very used approach is the batch-mode EM algorithm (DEMPSTER; LAIRD; RUBIN, 1977) (Section 2.3), which follows a mixture distribution approach for probabilistic modeling. Like *k-means*, this algorithm requires that the number of concepts be fixed and known at the start of the learning process. Moreover, the parameters of each distribution are computed through the usual statistical point estimators, a batch-mode approach which considers that the complete training set is previously known and fixed (TAN; STEINBACH; KUMAR, 2006).

These restrictions make the *k-means* and EM algorithms not suitable for on-line concept formation, because in this kind of task usually each data point is just instantaneously available, i.e., the learning system needs to build a model, seen as a set of concepts of the environment, incrementally from data flows. The neural network model proposed in this thesis, on the other hand, is able to learn from data flows in an incremental (new concepts can be added by demand) and on-line (it does not require that the complete training set be previously known and fixed) way, which makes it a good solution for concept formation in on-line robotic tasks. Moreover, unlike the traditional neural network models (e.g., MLP and GRNN), the IGMN hidden neurons are not “black boxes”, and thus the Gaussian units can be interpreted as representations of the input space, i.e., high level concepts (HEINEN; ENGEL, 2010e). The remaining of this section is organized as follows: Subsection 6.1.1 presents some related work about concept formation, and Subsection 6.1.2 describes how IGMN can be used to build high-level concepts incrementally from data flows.

### 6.1.1 Related work

In the past different approaches were presented to create high level concepts from sonar data in robotic tasks. As a typical example of these approaches, Nolfi and Tani (1999) proposed a hierarchical architecture to extract regularities from time series, in which higher layers are trained to predict the internal state of lower layers when such states significantly change. In this approach, the segmentation was cast as a traditional error minimization problem (HAYKIN, 2008), which favors the most frequent inputs, filtering out less frequent input patterns as being “noise”. The result is that the system recognizes slightly differing walls, that represent frequent input patterns, as distinct concepts, but is unable to detect corridors or corners that are occasionally (infrequently) encountered. Moreover, this algorithm has scarce means to handle the stability-plasticity dilemma and to appropriately model the data.

Figure 6.1(a) shows the environment used by Nolfi and Tani (1999) to validate their approach, where it is considered the case of a mobile robot that performs a wall following behavior in an environment composed by two rooms joined by a short corridor. In this figure the straight lines represent the walls and the full circle represents a cylindrical object. The circle on the left-bottom side represents the robot and the trace on the terrain represents the trajectory of the robot during a few laps in the environment. The robotic platform used in this experiment is a simulated Khepera robot. This robot has a circular shape with a diameter of 55mm, a height of 30mm, and a weight of 70g. It is supported by two wheels and two small Teflon balls. The wheels are controlled by two DC motors



with an incremental encoder. The robot is provided with eight infrared proximity sensors that can detect obstacles within a range of about 3cm.

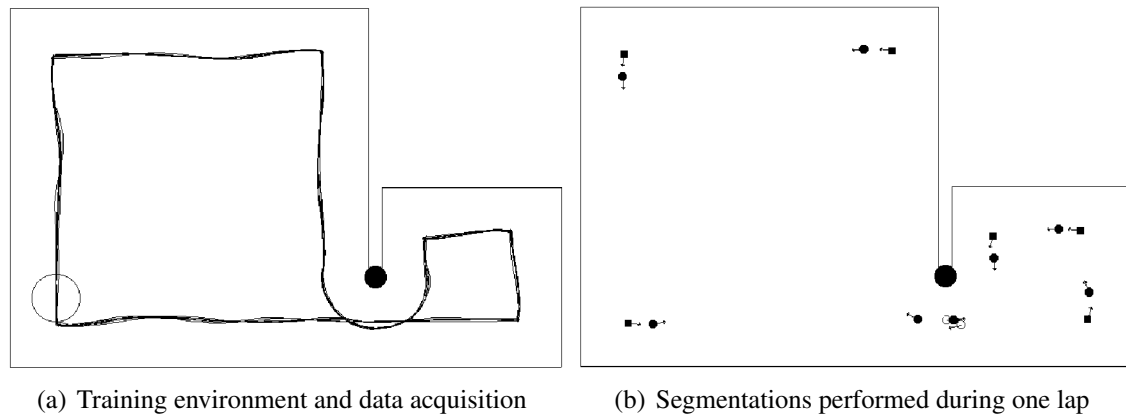


Figure 6.1: Results obtained using the Nolfi and Tani's model in an environment composed by two rooms joined by a short corridor

Figure 6.1(a), reproduced from Nolfi and Tani (1999), shows the segmentation performed by the Nolfi and Tani's model during one lap in the environment. The full circle, empty circle, and full square indicate, respectively, which of the three segmentation units is active in a given cycle. The state of the segmentation units is displayed only when a new segmentation occurs (i.e. when the state changes). The arrows indicate the direction of the robot when the corresponding segmentation occurs. This experiment was performed using an Elman's network (ELMAN, 1990) with 10 input units (which encode the state of the eight infrared sensors of the robot and the speed of the two motors), 3 hidden units (which detect the states of the environment), and 8 output units (which encode the state of the infrared sensors at time  $t+1$ ). The main drawbacks of this approach are: (i) it requires that the number of concepts (i.e., hidden units) be fixed and known at the start of the learning process; and (ii) the learning algorithm is off-line and requires several scans over the training data to converge (in fact the neural network using in this experiment was trained for 100000 epochs).

Focusing in change detection, Linåker and Niklasson (2000a; 2000b) proposed an adaptive resource allocating vector quantization (ARAVQ) network, which stores moving averages of segments of the data sequence as vectors allocated to output nodes of the network. New model vectors are incorporated to the model if a mismatch between the moving average of the input signal and the existing model vectors is greater than a specified threshold, and a minimum stability criterion for the input signal is fulfilled. Figure 6.2, reproduced from Linåker (2003), shows the results obtained using the ARAVQ network and the Khepera robot simulator (MICHEL, 1996), in the same environment shown in Figure 6.1(a). The main advantage of this approach over the Nolfi and Tani's model is that the ARAVQ network requires a single scan over the training data to converge. Moreover, it can add hidden neurons (i.e., to create new concepts) incrementally from data flows.

Although the ARAVQ network has been used in some robotic tasks (LINÅKER; JACOBSSON, 2001; BAKKER; LINÅKER; SCHMIDHUBER, 2002; BAKKER et al., 2003; LINÅKER, 2003; HOLLAND; GOODMAN, 2003; STENING; JACOBSSON; ZIEMKE, 2005), like other distance-based clustering algorithms its induced model is equivalent to a set of equiprobable spherical distributions sharing the same variance, what

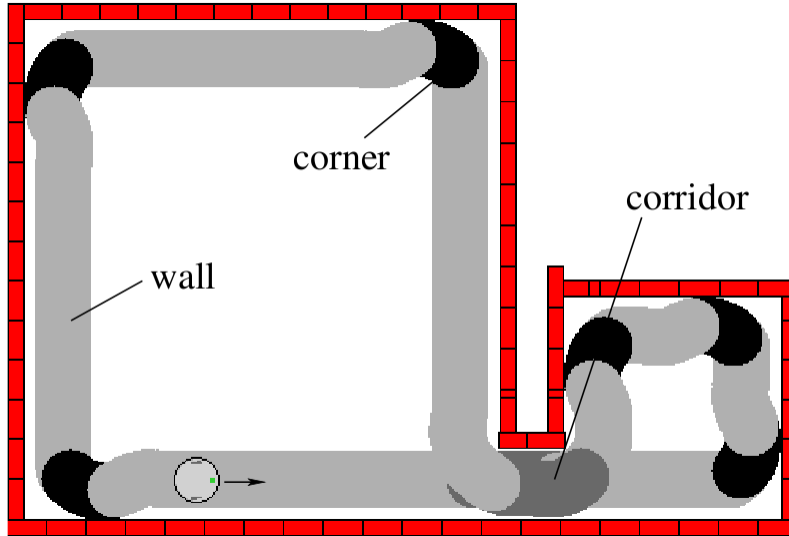


Figure 6.2: Results obtained using ARAVQ in an environment composed by two rooms joined by a short corridor

badly fits to a data flow with temporal correlation, better described by elongated elliptical distributions. Next subsection describes some experiments in which IGMN is used to learn high-level concepts in an incremental and efficient way.

### 6.1.2 Concept formation experiments

This section describes some experiments in concept formation tasks. In these experiments the data consist of 10 continuous values provided by the Pioneer 3-DX simulator software ARCOS (Advanced Robot Control & Operations Software). A Pioneer 3-DX robot has 8 sonar sensors, disposed in front of the robot at regular intervals, and a two-wheel differential, reversible drive system with a rear caster for balance. Figure 6.3 shows a Pioneer 3-DX robot and the disposition of its sonar sensors.

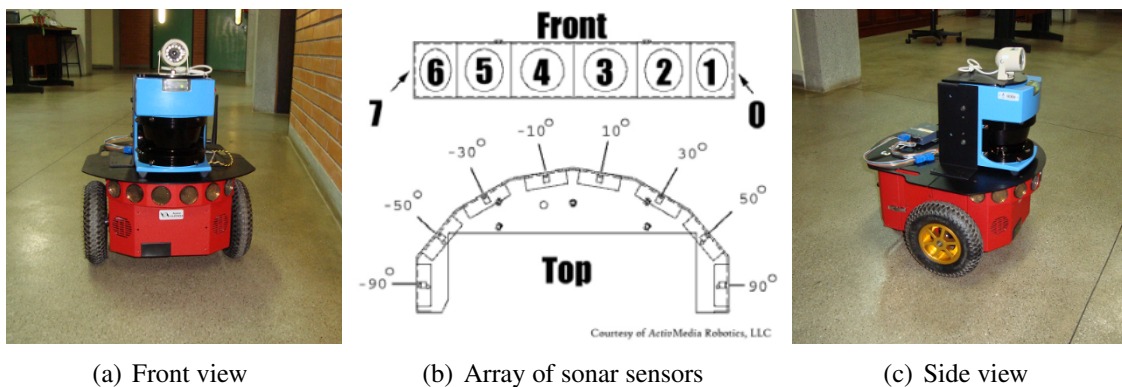


Figure 6.3: Pioneer 3-DX robot with eight sonars and a differential steering drive

The IGMN network used in these experiments has two cortical regions,  $\mathcal{N}^s$  and  $\mathcal{N}^v$ . The cortical region  $\mathcal{N}^s$  tackles the values of the sonar readings, i.e.,  $\mathbf{s} = \{s_1, s_2, \dots, s_8\}$ , and the cortical region  $\mathcal{N}^v$  receives the speeds applied at the robot wheels at time  $t$ , i.e.,  $\mathbf{v} = \{v_1, v_2\}$ . To decide what is the most active concept at time  $t$ , the maximum likelihood (ML) hypothesis  $\ell = \arg \max_j [p(j|\mathbf{z})]$ , where  $\mathbf{z} = \{\mathbf{s}, \mathbf{v}\}$ , is used. It is important to note that IGMN computes and maintains the a posteriori probabilities of all concepts at

each time, and hence it can be used in applications such as the so called multi-hypothesis tracking problem in robotic localization domains (FILLIAT; MEYER, 2003; THRUN; BURGARD; FOX, 2006). The configuration parameters used in the following experiments are  $\delta = 0.1$  and  $\tau_{nov} = 0.01$  (as the main goal in concept formation is to identify natural groupings in the input space, we have decided to use in this experiment the  $\tau_{nov}$  parameter to adjust the model complexity rather than the  $\varepsilon_{max}$  parameter). It is important to say that no exhaustive search was performed to optimize the configuration parameters.

The first experiment was accomplished in an environment composed of six corridors (four external and two internal), and the robot performed a complete cycle in the external corridors. Figure 6.4 shows the segmentation of the trajectory obtained by IGMN when the robot follows the corridors of this environment. IGMN created four probabilistic units, corresponding to the concepts “corridor” (1: plus sign), “wall at right” (2: circle), “corridor / obstacle front” (3: asterisk) and “curve at left” (4: cross). The symbols in the trajectory of Figure 6.4 represent the ML hypothesis in each robot position, and the black arrow represents the robot starting position and direction. More details about this experiment can be found at Engel and Heinen (2010a).

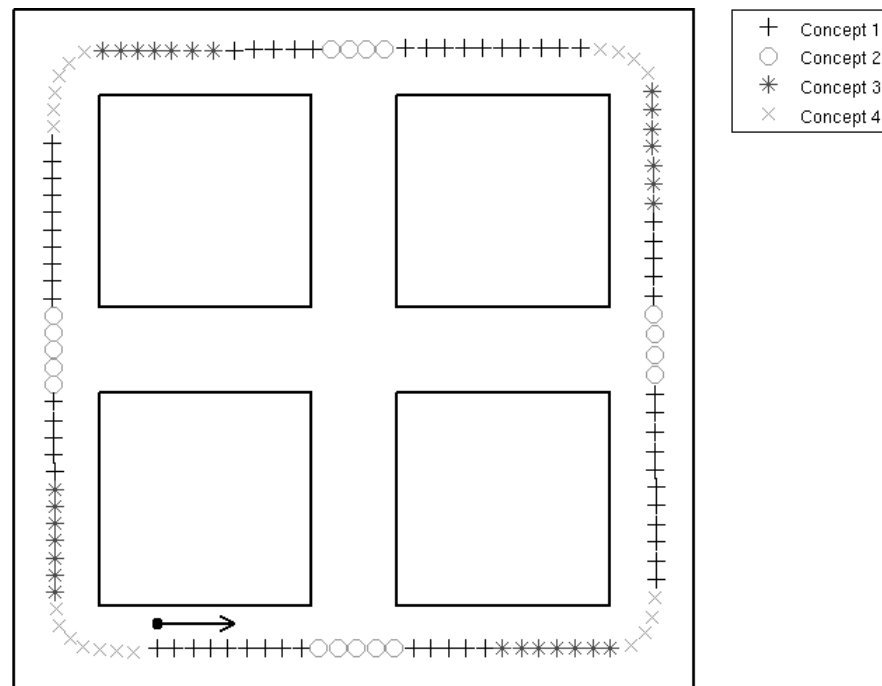


Figure 6.4: Segmentation obtained by IGMN in six corridors experiment

The next experiment was performed in a more complex environment, composed by two different sized rooms connected by a short corridor. This environment was inspired in those used by Linåker and Niklasson (2000a; 2000b) (Figure 6.1(a) above). Figure 6.5 shows the segmentation performed by IGMN in this experiment. IGMN has created seven clusters, corresponding to the concepts “wall at right” (1: plus sign), “corridor” (2: circle), “wall at right / obstacle front” (3: asterisk), “curve at left” (4: cross), “bifurcation / obstacle front” (5: square), “bifurcation / curve at right” (6: five-pointed star) and “wall at left / curve at right” (7: hexagram).

Comparing these experiments, it can be noticed that some similar concepts, like “curve at left” and “obstacle front”, were discovered in both experiments, although these environments are different (the environment shown in Figure 6.4 has many corridors whilst

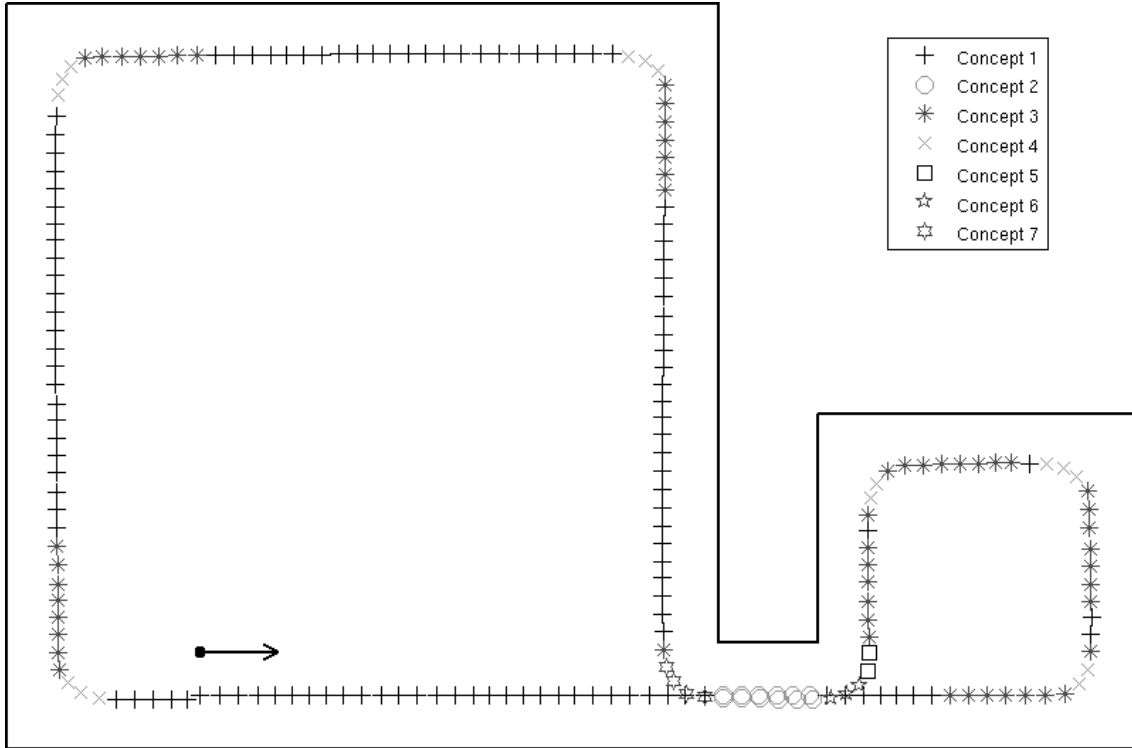


Figure 6.5: Segmentation obtained by IGMN in the two rooms experiment

that one shown in Figure 6.5 has two large rooms and just one short corridor). This points out that concepts extracted from a data flow corresponding to a specific sensed environment are not restricted to this environment, but they form an alphabet that can be reused in other contexts. This is a useful aspect, that can improve the learning process in more complex environments.

As described above, the main goal in concept formation is to identify natural groupings in the input space, which are represented by IGMN using its Gaussian units. Therefore, in this kind of task the estimation/prediction capabilities of IGMN are not necessary (in fact, this experiment can be performed using IGMM as well). Nevertheless, these estimation/prediction capabilities can be used to compute the motor actions for a robot performing a wall following behavior, for instance, as is shown in the next subsection.

## 6.2 Estimating the desired speeds in a mobile robotics application

In the experiments described in the previous section the main goal was to identify natural groupings in the input space, which were represented by the IGMN Gaussian units, and therefore the IGMN estimation/prediction capabilities were not used. In this section we will use these estimation/prediction capabilities to compute the desired actions (i.e., the wheel speeds) for a mobile robot performing a wall following behavior in a simulated environment. These experiments are relevant because in robotic control tasks usually it is not possible to predict all situations that occur in the real world, and hence the robot needs to learn from experience while interacting with the environment.

In a previous experiment, presented in Heinen and Engel (2010a), Equation 4.9 was used to estimate the desired speeds in the same environment described above (Figure 6.5) and using a IGMN neural network with two cortical regions,  $\mathcal{N}^S$  and  $\mathcal{N}^V$  and seven

Gaussian units added by the minimum likelihood criterion. Figure 6.6(a) shows the results obtained in this previous experiment, where the  $x$  axis corresponds to the time index of the sensor readings (the training database is composed by 2070 data samples) and the  $y$  axis corresponds to the difference between the right and left motor speeds, i.e.,  $y_d(t) = v_1 - v_2$ . A positive value in  $y_d(t)$  corresponds to a left turn in the robot trajectory and a negative value corresponds to a right turn. The solid gray line in Figure 6.6(a) represents the desired  $y_d(t)$  values and the dashed black line represents the difference between the actual IGMN outcomes, i.e.:  $y_o(t) = \hat{v}_1 - \hat{v}_2$ . It is important to say that the region  $\mathcal{N}^{\mathcal{V}}$  has size  $D^{\mathcal{V}} = 2$ , i.e., the difference  $y_o(t)$  was computed just to improve the visualization in Figure 6.6(a).

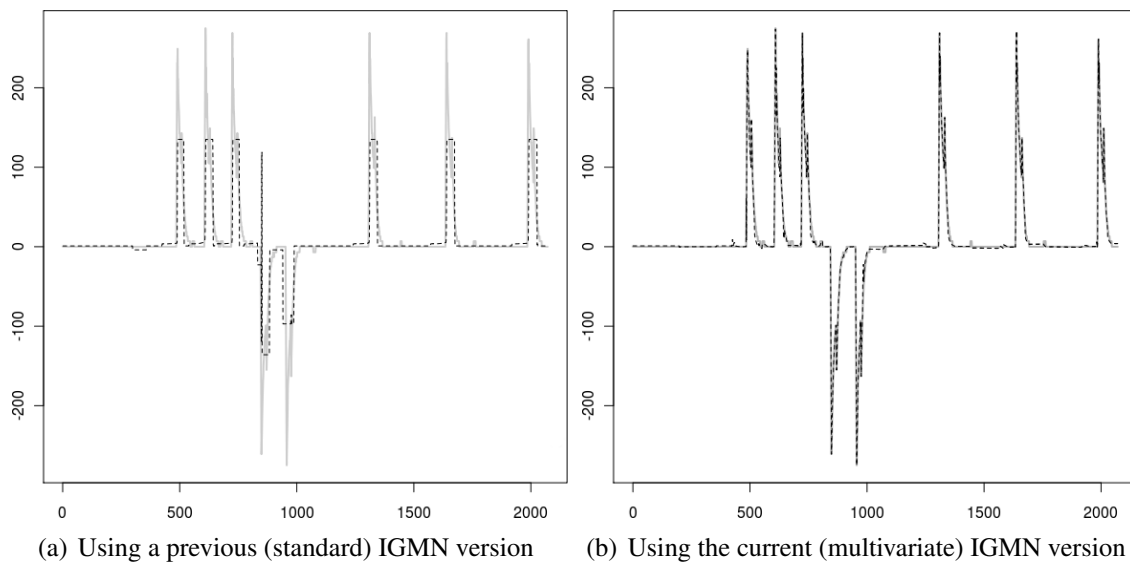


Figure 6.6: Difference between the speeds of the right and left motors of the robot while it is following the trajectory shown in Figure 6.5.

Observing Figure 6.6(a) we can notice that the standard IGMN version was able to approximate the desired output with a reasonable accuracy (the NRMS error was 0.050373), using few Gaussian units (just seven) and no memory of past perceptions and actions (e.g., no recurrent connections nor delay units). The main differences between  $y_d(t)$  and  $y_o(t)$  are in the extremes, i.e., the approximation performed by IGMN is smoother than the target function. Although these results are reasonable, if we use this neural network to control the robot it will hit the wall in the first curve of the environment, i.e., it cannot control the robot correctly. This occurs because in this experiment the minimum likelihood criterion was used to add new Gaussian units, which does not necessarily leads to a good generalization level. In fact, to obtain a better approximation the error-driven mechanism, introduced in Section 4.3, is more suitable because it is directly related to the approximation error rather than the model complexity.

Therefore, this experiment was repeated using the multivariate IGMN version presented in Chapter 4, and the results obtained in this experiment are show in Figure 6.6(b). The configuration parameters used in this experiment are  $\delta = 0.01$ ,  $\varepsilon_{max} = 0.01$  and  $\Omega = 6$ . We can notice in this figure that using the current IGMN version the approximation error is much lower than using the previous IGMN version: the NRMS error is just 0.034598. The time required for learning was 0.351 seconds, and 40 Gaussian units were added in each region. Moreover, if we use this trained network to control the robot, it

will follow the trajectory shown in Figure 6.7, where we can notice that the robot follows the “target trajectory” very well. These results are still more impressive because the robot does not receive any information about its position (just the readings of the sonar sensors are provided at each 100 milliseconds) and the neural network has no memory of past perceptions and actions, i.e., an action must be taken using just the current perception.

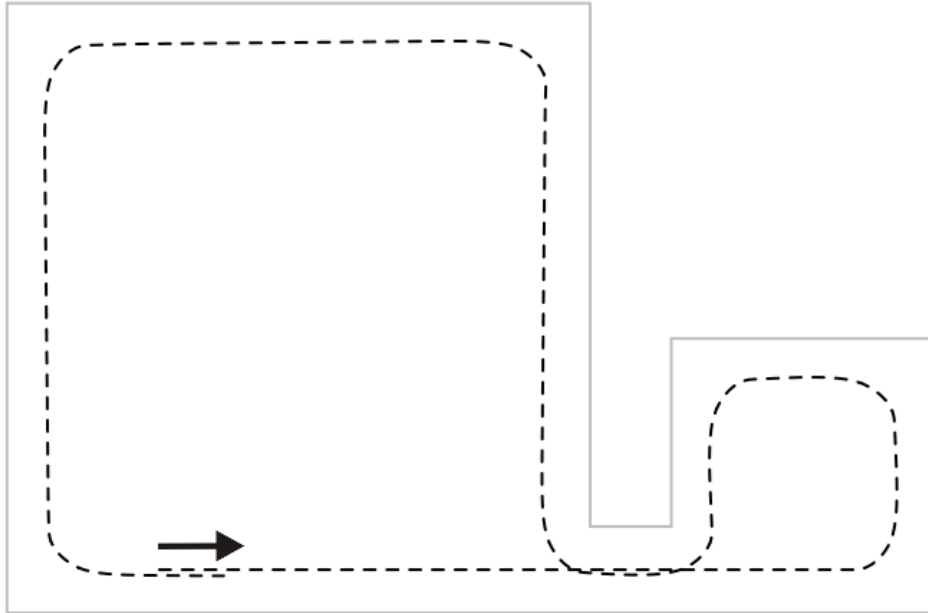


Figure 6.7: Trajectory obtained using IGMN to choose the robot actions

Table 6.1 shows a comparison among the results obtained using IGMN and other neural approaches. The first column presents the ANN model. The following columns show, respectively, the number of hidden/pattern units, the NRMS error computed using the 10-fold cross validation procedure, the average number of training epochs and the time required to perform each replication, i.e. 1/10 of the time required by the entire 10-fold cross validation procedure. To facilitate our comparison, the last row on Table 6.1 shows the results obtained using IGMN. Figure 6.8 shows a boxplot graph comparing the NRMS errors in these experiments. To allow a fair comparison, an exhaustive search was performed to find out the best configuration parameters of each neural network model.

Table 6.1: Comparative among ANNs in follow the trajectory of Figure 6.5

ANN model	Units	NRMS	Epochs	Time
MLP – RPROP	10	0.043192	236.4	5.367s
MLP – LM	10	0.034778	53.7	4.897s
GRNN ( $\sigma = 90$ )	2070	0.033172	1.0	0.498s
IGMN – Multivariate	40	0.034598	1.0	0.351s

We can notice in Table 6.1 that the IGMN performance is comparable to other ANN models. In fact, just the GRNN model has a better performance, but observing the boxplot graphs in Figure 6.8 we can notice that the differences are not statistically significant, i.e., the confidence intervals overlap. Moreover, IGMN can learn the target function very fast using a single scan over the training data and does not require an exhaustive search to find out the best configuration parameters (actually just the  $\varepsilon_{max}$  parameter must be reduced until the desired approximation level is achieved).

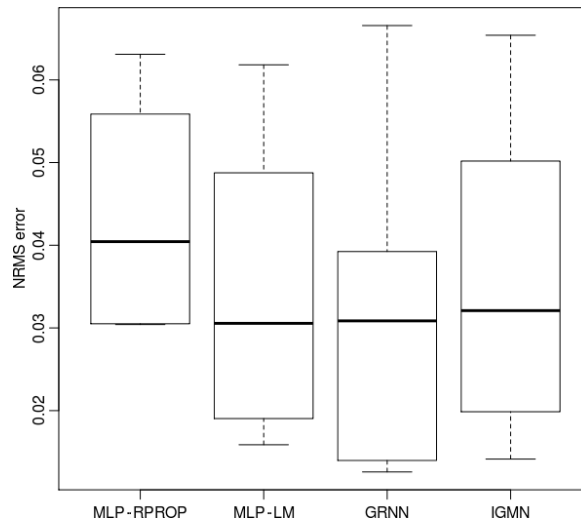


Figure 6.8: Comparing the approximation errors in the trajectory of Figure 6.5

The next experiment was performed in a more complex and irregular environment, shown in Figure 6.9, where the robot was preprogrammed to follow the external walls of the simulated environment. IGMN was trained using the data corresponding to one lap in the environment (1631 samples), and was tested using another independent lap (1551 samples). This experiment is more difficult than the previous one (Figure 6.5) because the control algorithm is more complex. In fact, in the previous experiment the robot was manually controlled to perform a fixed trajectory in the regular environment, whilst in this experiment the robot is automatically controlled using the noisy sonar data to perform a wall following behavior, and therefore the robot's trajectory changes slightly at each lap according to the noisy readings received at each instant.

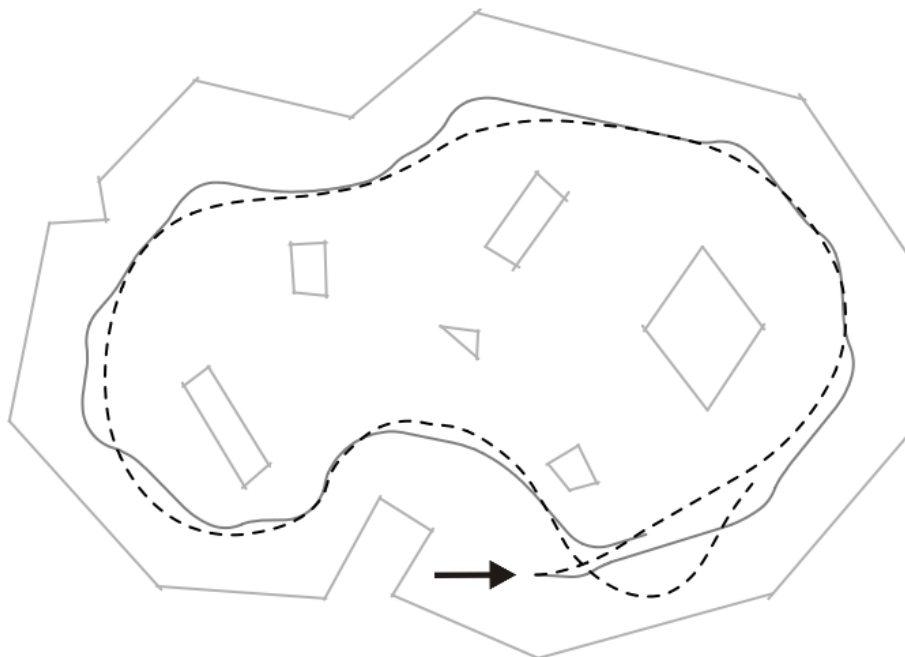


Figure 6.9: Wall following behavior in a more complex environment. The solid gray line shows the trajectory followed by the robot in the learning mode and the dashed black line shows the trajectory followed by the robot using IGMN to control its actions.

The solid gray line in Figure 6.9 shows the trajectory followed by the robot during the learning phase, and the dashed black line shows the trajectory followed by the robot using the trained IGMN network to control its actions. We can notice in this figure that the robot have not simply repeated the “target trajectory” after training. On the contrary, it follows a softer trajectory through the environment, which demonstrates that the neural network has really learned the wall-following behavior rather than just reproducing the target trajectory.

The configuration parameters used in this experiment are  $\delta = 0.01$ ,  $\varepsilon_{max} = 0.75$  and  $\Omega = 6$ , and just two Gaussian units were added during learning. Actually using lower values in  $\varepsilon_{max}$  (e.g.,  $\varepsilon_{max} = 0.1$ ) the robot’s trajectory will be more similar to the target one (and more neurons will be added to each region, of course), but from a practical point of view this is not interesting. As a matter of fact, it is much better learning a control behavior (a wall following behavior, in this case) than just reproducing a target trajectory, because the control behavior is much more robust against changes in the environment.

Table 6.2 shows a comparison among the results obtained in this experiment using IGMN and other ANN approaches. We can notice in this table that the IGMN performance is comparable to other models even using just two Gaussian units and without fine-tuning its configuration parameters.

Table 6.2: Comparative among ANNs in learning the wall-following behavior

ANN model	Units	NRMS	Epochs	Time
MLP – RPROP	6	0.168697	78.2	4.224s
MLP – LM	6	0.157162	27.9	7.146s
GRNN ( $\sigma = 210$ )	1631	0.142121	1.0	0.229s
IGMN – Multivariate	2	0.140875	1.0	0.045s

These experiments show that IGMN is a very suitable tool for robotic control tasks, because it can learn a control behavior very fast, incrementally and using few Gaussian units. Next section describes how IGMN can solve the inverse kinematics problem.

### 6.3 Computing the inverse kinematics in a legged robot task

The experiments described above were performed using a “wheeled” robot, i.e. a simulated robot which uses wheels for locomotion. The next experiment, on the other hand is performed using a simulated legged robot, and the role of IGMN in this experiment is to compute the inverse kinematics of the robot legs. This experiment was performed using the LegGen simulator (HEINEN; OSÓRIO, 2009, 2008, 2007a,b, 2006a,b,c), which is devised to evolve gait control strategies using a physically realistic simulation environment.

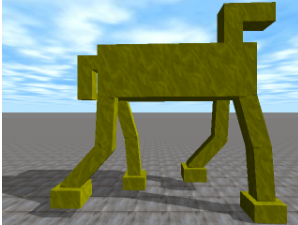
According to Osório et al. (2006), for a simulation of legged robots be realistic, it is necessary to model several elements of the real world, such as gravity and friction, in order that the simulated bodies to behave in a way similar to their equivalents in the real world (BEKEY, 2005). In the LegGen simulator this is achieved by using Open Dynamics Engine (ODE<sup>1</sup>), which is a C++ software library that allows the simulation of rigid bodies with great physical realism. LegGen also uses genetic algorithms (GA) (GOLDBERG, 1989), implemented using the GALib<sup>2</sup> software library, to evolve the control strategies.

<sup>1</sup>Open Dynamics Engine (ODE) – <http://openode.sourceforge.net>

<sup>2</sup>GALib software library – <http://www.lancet.mit.edu/ga/>



Figure 6.10 shows the simulated robot used in this experiment. Its dimensions are approximately those of a medium sized dog. The joint restrictions used in the simulated robot are similar to its biological equivalent, with the following values: Hip=[-60°;15°]; Knee=[0°;120°]; Ankle=[-90°;30°]. To simplify the control strategy all robot legs have the same joint restrictions. More details about LegGen and the simulated robot are found at Heinen and Osório (2009; 2008; 2007a; 2007b).



Dimensions			
Part	x	y	z
Body	45.0cm	15.0cm	25.0cm
Thigh	5.0cm	15.0cm	5.0cm
Shin	5.0cm	15.0cm	5.0cm
Paw	8.0cm	5.0cm	9.0cm

Figure 6.10: Modeled robot

In some previous experiments, presented in Heinen (2007) and Heinen and Osório (2007a; 2006a), the inverse kinematics was computed using the Powell's direction set method (BRENT, 1973; PRESS et al., 1992), which works well but requires an initial guess of the joint angles to approximate a valid answer. Moreover, the Powell's method requires several iterations to converge, which can prevent its use in real time, and can become trapped into local minima. In this section we propose to use IGMN for computing the inverse kinematics of the robot legs, hence preventing these restrictions of the Powell's method described above. More specifically, IGMN was used to learn the forward kinematic model and afterwards to compute the inverse model in real time. This is only possible because the same partially trained IGMN network can approximate both  $f(\cdot)$  and  $f(\cdot)^{-1}$  even in regions where the target function is multivalued. The training dataset used in this experiment is composed by 1000 data samples generated using the forward model given by:

$$\begin{aligned} \phi &= \sum_{i=1}^N \alpha_i \\ x &= \sum_{i=1}^N \left( l_i \sum_{j=1}^i \cos(\alpha_j) \right) \\ y &= \sum_{i=1}^N \left( l_i \sum_{j=1}^i \sin(\alpha_j) \right) \end{aligned} \quad (6.1)$$

where  $l_i$  is the length of  $i$ th leg part,  $\alpha_i$  is the angle of the  $i$  leg joint,  $N$  is the number of leg parts in each robot's leg (three in this case) and  $(x, y, \phi)$  is the pose of the actuator (the robot's paw). For generating this training dataset the joint angles  $\alpha_i$  were randomly chosen using a uniform distribution in the interval of the maximum and minimum angles (i.e., the joint restrictions) of each joint in the robot legs. The neural network used to learn this dataset has two cortical regions,  $\mathcal{N}^A$  and  $\mathcal{N}^B$ , where the region  $\mathcal{N}^A$  receives the angles of each joint at each instant  $t$ , i.e.:  $\mathbf{a}^t = \{\alpha_1, \alpha_2, \alpha_3\}$ , and the cortical region  $\mathcal{N}^B$  receives the corresponding pose of the robot's paw computed by (6.1), i.e.,  $\mathbf{b}^t = \{x, y, \phi\}$ . The IGMN configuration parameters were set to  $\delta = 0.01$ ,  $\varepsilon_{max} = 0.1$  and  $\Omega = 6$ .

IGMN has learned this training dataset very well: the NRMS error was 0.006414, the time required for learning was just 0.18 seconds and 45 Gaussian units were added during learning. After training, this neural network was used to compute the inverse kinematics of the robot legs, i.e., it receives a desired paw position and computes the joint angles to achieve that position. Figure 6.11 shows the robot's walking produced by LegGen after the evolving of the control strategy. More details about the evolution of the control strategies using LegGen can be found at Heinen and Osório (2009; 2008) (here we will

not deep into details of LegGen because the evolution of the legged robot gaits is outside the scope of this thesis – our interest here is just computing the inverse kinematics in a robust and efficient way).

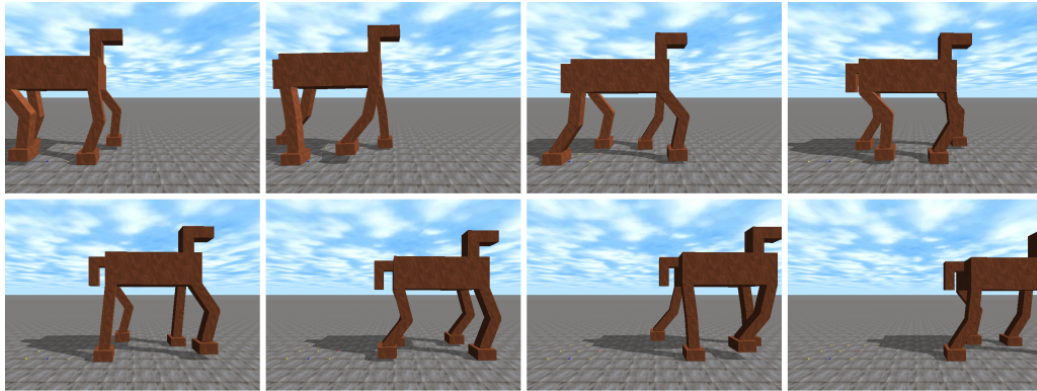


Figure 6.11: Robot walking using IGMN to compute the inverse kinematics

Actually the results shown in Figure 6.11 are similar to those presented in Heinen and Osório (2007a), because both the Powell’s method and the IGMN model can compute the inverse kinematics in a very precise way. The main advantage of IGMN in this task is that it does not require an initial guess of the joint angles to compute a valid solution nor become trapped into local minima (the produced solution is always a valid one). Moreover, after training the neural network computes the inverse kinematics very fast – it requires less than  $10^{-4}$  seconds – thus allowing the use of IGMN in real time. The main drawback of IGMN is that in some cases it may outcome a valid answer that is not the best one in a given moment (e.g., IGMN can chose a combination of joint angles that is far from the current angles but that still leads to the desired robot’s paw pose), but in this experiment this problem was not observed. Actually this situation can be prevented by using a temporal context in the neural network (e.g., the current position of each joint) in order to induce a solution near from the current joints position, for instance.

This experiment demonstrates that IGMN is a good tool for solving the inverse kinematics problem even in real time robotic control applications. Other ANN models such as MLP, RBF and GRNN cannot be used to solve this problem, because as described in Section 4.4, these models use a sum-of-squares error function and hence may not produce valid answers in regions of the state space where the target function is multivalued (BISHOP, 1995). Next section describes how IGMN can be used as a function approximator in reinforcement learning tasks.

## 6.4 Reinforcement Learning using IGMN

This section presents a couple of experiments, published in Heinen and Engel (2010a; 2009a), in which IGMN is used as a function approximator in reinforcement learning (RL) algorithms. Traditional reinforcement learning techniques (e.g., Q-learning and Sarsa) (SUTTON; BARTO, 1998) generally assume that states and actions are discrete, which seldom occurs in real mobile robot applications. To allow continuous states and actions directly in RL (i.e., without discretization) it is necessary to use function approximators like MLP (UTSUNOMIYA; SHIBATA, 2009) or RBF (DOYA, 2000; BASSO; ENGEL, 2009) neural networks. According to Smart (2002), for a function approximator be successfully used in reinforcement learning tasks (i.e., for converging to a good solution) it

must be: (i) incremental (it should not have to wait until a large batch of data points arrives to start the learning process); (ii) aggressive (it should be capable of producing reasonable predictions based on just a few training points); (iii) non-destructive (it should not be subject to destructive interference or “forgetting” past values); and (iv) must provide confidence estimates of its own predictions. Thus, according to these principles IGMN is very suitable for reinforcement tasks, i.e., it satisfies all the requirements described above. The remainder of this section is organized as follows. Subsection 6.4.1 presents some related work in the field of reinforcement learning using continuous states and actions. Subsection 6.4.2 describes how IGMN can be used as a function approximator in a RL algorithm. Finally, Subsections 6.4.3 and 6.4.4 describe some experiments performed to evaluate the proposed model in reinforcement learning tasks.

### 6.4.1 Related work

In the past several approaches were proposed to allow continuous states and actions in RL algorithms. As an example of these approaches, in Doya (1996; 2000) a continuous formulation of the temporal difference  $TD(\lambda)$  algorithm is presented (SUTTON, 1988). This formulation uses normalized radial basis function (RBF) networks to approximate the continuous state values and to learn the continuous actions. According to Doya (2000), RBF networks are more suitable for reinforcement learning tasks than MLP (RUMELHART; HINTON; WILLIAMS, 1986; HAYKIN, 2008) because they perform a local encoding of the input receptive fields, which avoids the catastrophic interference, i.e., the knowledge acquired in a region of the input space does not destroy the knowledge acquired previously in another region of the input space (BASSO; ENGEL, 2009). However, in the algorithm described in Doya (1996; 2000) the radial basis functions are simply uniformly distributed among the input space and kept fixed during all the learning process, i.e., just the (linear) output layer is adjusted by the learning algorithm. Therefore, this algorithm does not adjust the network parameters of the hidden layers, which is a complex and nonlinear task. Moreover, it requires a priory knowledge to setup the neural units and in general wastes computational resources in unimportant regions of the input space.

Another interesting approach to allow continuous states and actions in RL is the Locally Weighted Regression (LWR) algorithm proposed by Smart and Kaelbling (2000). Although at a first glance LWR seems very promising, it has a strong drawback: it requires that all data points received so far be stored and analyzed at each decision making (i.e., it is a “lazy learning” algorithm). Thus, this algorithm is not suitable for on-line robotic tasks, because in this kind of task the sensory data are very abundant, which makes the algorithm very slow and requires large amount of memory to store all previous data points. The neural network model proposed in this thesis, on the other hand, does not require that any previous data be stored or revisited, i.e., each training data can be immediately used and discarded. This makes the proposed model more suitable to be used in on-line robotic tasks, specially when the learning process must occur perpetually (i.e., when there are no separate phases for learning and use). Next subsection describes how IGMN can be used to select continuous actions in a RL algorithm.

### 6.4.2 Selecting the robot actions using IGMN

Implementing a reinforcement learning algorithm using IGMN can be straightforward – we just need to use three cortical regions,  $\mathcal{N}^s$ ,  $\mathcal{N}^a$  and  $\mathcal{N}^Q$ , to represent the states,  $s$ , actions,  $a$ , and the  $Q(s, a)$  values, respectively. If the actions are discrete, then it is very

easy to select the best action at each time – we just need to propagate the current state and all possible actions in the corresponding cortical regions and select the action which has the highest  $Q$  value, i.e.:

$$\mathbf{a}^* = \arg \max_{\mathbf{a}} [Q(s, a)]. \quad (6.2)$$

Moreover, the exploration  $\times$  exploitation dilemma can be tackled using an action selection mechanism such as *softmax* and  $\epsilon$ -greedy (SUTTON; BARTO, 1998). On the other hand, if the actions are continuous, the action selection process becomes a general optimization problem far from trivial (SMART, 2002).

In this thesis we propose a new strategy for selecting continuous actions in reinforcement learning algorithms. This strategy consists in first propagating through the IGMN network the current state,  $s$ , and the maximum value,  $Q_{max}$ , currently stored in the corresponding Gaussian units, i.e.:

$$Q_{max} = \max_{j \in M} (\mu_j^Q). \quad (6.3)$$

Then the  $Q_{max}$  value is propagated through the cortical region  $\mathcal{N}^Q$ , the associative region  $\mathcal{P}$  is activated and the *greedy* action  $\hat{\mathbf{a}}$  is computed in the cortical region  $\mathcal{N}^A$  using (4.10).

To tackle the exploration  $\times$  exploitation dilemma, instead of simply choosing the greedy action  $\hat{\mathbf{a}}$  at each moment we can randomly select the actions using the estimated covariance matrix  $\hat{\mathbf{C}}^A$ , i.e., the actions can be randomly selected using a Gaussian distribution of mean  $\hat{\mathbf{a}}$  and covariance matrix  $\hat{\mathbf{C}}^A$ . In the beginning of the learning process, when  $M = 0$ , the initial action can be randomly chosen. The main advantage of this action selection mechanism is that it enables high exploration rates in the beginning of the learning process, when the Gaussian distributions are larger, and this exploration is reduced as the confidence estimates become stronger. Moreover this mechanism does not require any optimization technique (just the IGMN itself), which makes the proposed RL algorithm very fast. Hence, this mechanism allows an exploration strategy based on statistical principles which does not require *ad-hoc* parameters.

The following subsections describe two experiments performed to evaluate the proposed model in reinforcement learning tasks using continuous states and actions: a pendulum with limited torque and a robot soccer task in a simulated environment. The configuration parameters used in these experiments are  $\delta = 0.01$  and  $\varepsilon_{max} = 0.1$ . More details about these experiments are found at Heinen and Engel (2010a; 2009a; 2009c).

### 6.4.3 Pendulum with limited torque

This experiment, also performed by Doya (1996; 2000) to evaluate the Doya’s continuous actor-critic, consists in learning the control policy of a pendulum with limited torque using reinforcement learning (Figure 6.12). The dynamics of the pendulum are given by  $\dot{\theta} = \omega$  and  $ml^2\dot{\omega} = -\mu\omega + mgl \sin \theta + \mu$  (DOYA, 2000), where  $\theta$  is the pendulum angle and  $\dot{\theta}$  is the angular velocity.

The physical parameters are mass  $m = 1$ , pendulum length  $l = 1$ , gravity constant  $g = 9.81$ , time step  $\Delta t = 0.02$  and maximum torque  $T_{max} = 5.0$ . The reward is given by the height of the tip of the pendulum,  $R(\mathbf{x}) = \cos \theta$ , and the discount factor is  $\gamma = 0.9$ . Each episode starts from an initial state  $x(0) = (\theta(0), 0)$ , where  $\theta(0)$  is selected randomly in  $[-\pi, \pi]$ . An episode lasted for 20 seconds unless the pendulum is over-rotated ( $|\theta| > 5\pi$ ). These parameters are the same used by Doya (2000) in the continuous actor-critic. Due to the stochastic nature of RL, this experiment was repeated 50 times using different random seeds, and the average of the obtained results is shown in Figure 6.13(a).

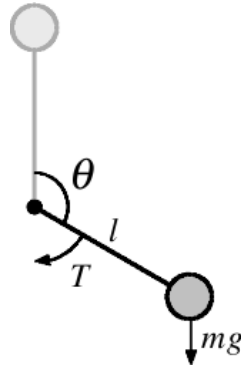


Figure 6.12: Pendulum swing up task

In Figure 6.13(a) the  $x$  axis represents the learning episode, and the  $y$  axis represents the time in which the pendulum stayed up ( $t_{up}$ ) i.e., when  $|\theta| < \pi/4$  (this is the same evaluation criteria used by Doya (2000)). The thick line in Figure 6.13(a) represents the mean and the thin lines represent the 95% confidence interval of the obtained results. Comparing these results with those presented in Doya (2000), reproduced here in Figure 6.13(b), we can notice that the proposed model has a superior performance compared to the Doya's continuous actor-critic (specially in the first episodes), is more stable and does not require any previous configuration of the Gaussian units. The average number of probabilistic units added during learning is IGMN was 109.41.

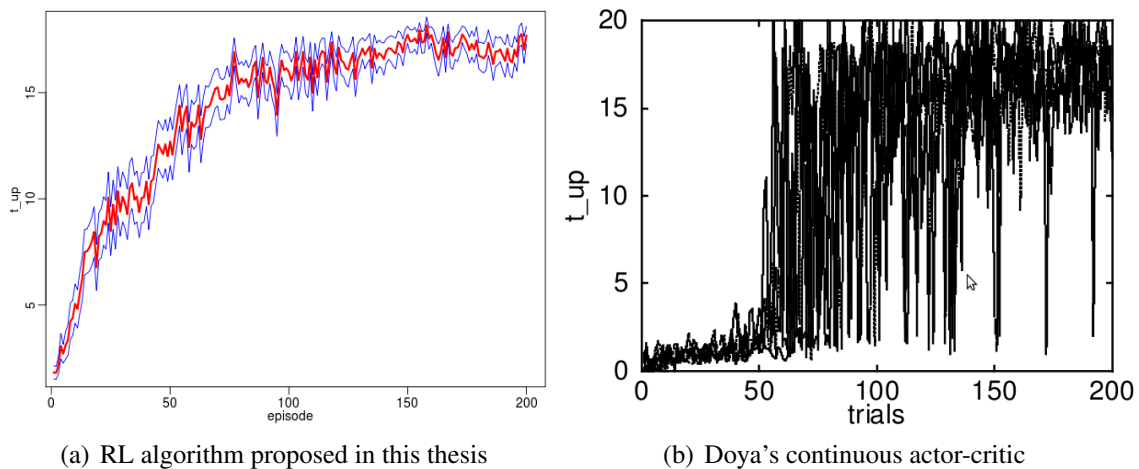


Figure 6.13: Results obtained by IGMN in the pendulum swing up task

#### 6.4.4 Robot soccer task

The next experiment, originally proposed in Asada et al. (1996; 2003), consists in learning to shoot a ball into the goal of a simulated robot soccer environment. To perform this experiment a robot soccer simulator was developed using the Open Dynamics Engine (ODE)<sup>3</sup> physics simulation library. A previous version of this simulator, described in Heinen and Osório (2007a; 2006b; 2006c), was used to evolve gaits of legged robots. The simulated environment follows the rules of the RoboCup<sup>4</sup> Soccer Middle Size League. The soccer field has 18 meters of length by 12 meters of width, the goal has 1 meter

<sup>3</sup>ODE – <http://www.ode.org>

<sup>4</sup>RoboCup – <http://www.robocup.org/>

of height and 2 meters of width, the goal posts have 12.5cm of diameter, and the ball has 70cm of circumference and 450 grams of weight. Moreover, walls of 1 meter of height were installed 1 meter apart from the field limits allowing the robot to perceive the environment using sonar sensors.

The simulated robot is similar to the Pioneer 3-DX robot used in the previous experiments. It has a box shape of 44.5cm of length, 39.3cm of width and 23.7cm of height. Its weight is 9kg and it has two wheels with 19.53cm of diameter and differential kinematics. The time interval  $\Delta t$  used in the simulations is 0.05 seconds. More details about this simulated environment can be found at Heinen and Engel (2009a; 2009c). The IGMN network used in this experiment has two cortical regions,  $\mathcal{N}^s$  and  $\mathcal{N}^v$ . The cortical region  $\mathcal{N}^s$  receives the values of the sonar readings, i.e.,  $\mathbf{s} = \{s_1, s_2, \dots, s_8\}$ , and the cortical region  $\mathcal{N}^v$  receives the speeds applied at the robot wheels at time  $t$ , i.e.,  $\mathbf{v} = \{v_1, v_2\}$ . The reward function  $r(t)$  used in this experiment is:

$$\begin{aligned} r(t) &= a(-d_{rb}(t)) + b(-d_{bg}(t)) && \text{if } d_{bg}(t) > 0 \\ r(t) &= 10 && \text{if } d_{bg}(t) \leq 0 \\ r(t) &= -10 && \text{if the ball exits the field} \end{aligned} \quad (6.4)$$

where  $d_{rb}(t)$  is the distance from the robot to the ball, and  $d_{bg}(t)$  is the distance from the ball to the goal in the time  $t$ . The parameters  $a = 1/4L$  and  $b = 2/L$  (where  $L$  is the field length) are used to modulate the influence of the terms in the reward function. If the ball hits the goal the episode ends with a reward  $r(t) = 10$  for a second, and if the ball exits the field the episode ends with a reward  $r(t) = -10$  for a second. Moreover, if the simulation time exceeds  $t_{max} = 300$  seconds the episode ends without any reward.

The learning process occurs in 1000 episodes. The robot starts an episode always in the same position, but the ball is randomly positioned (but in the range view of the sonar sensors). Thus, to obtain success in this task the robot needs: (i) to identify the ball using just sensory information; (ii) to move in the direction of the ball; and (iii) to “shoot” (or to lead) the ball into the goal without losing it. To evaluate the results two estimators were used: (i) the distance of the ball to the goal at the end of the episode (zero when the ball hits the goal) and (ii) the time required to the ball hit the goal ( $t_{max}$  in case of failing). Due to the stochastic nature of the task the whole experiment was repeated 30 times using different random numbers. Figure 6.14 shows the mean of the results obtained in these 30 replications of the experiment.

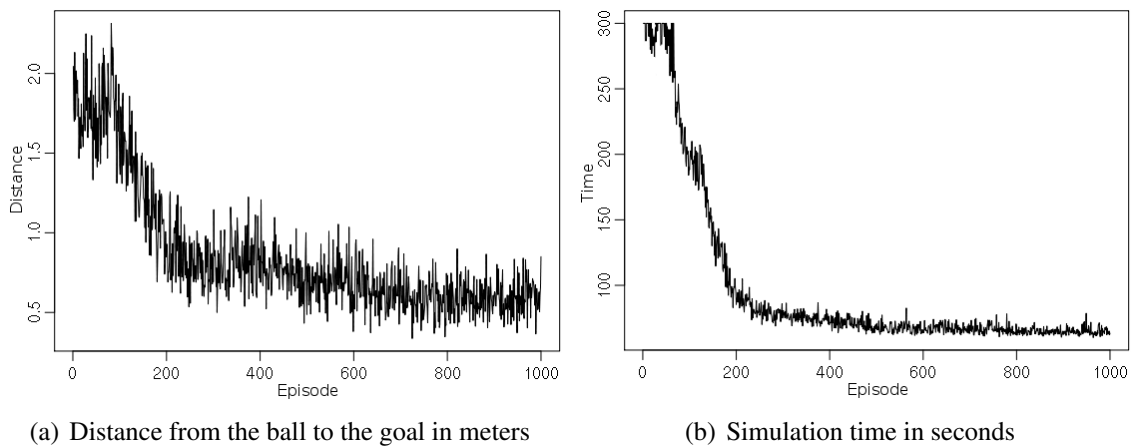


Figure 6.14: Results obtained in the robot soccer experiment

Observing the graph of Figure 6.14(a) it can be noticed that in the first episodes the robot was not able to reach the ball (the variations are due to the random initial ball position), but after the 100th episode the distances are strongly reduced. After the 750th episode the mean distances had stabilized near 0.6 meters, which indicates that the robot was able to lead the ball into the goal in great part of the episodes.

The graph of Figure 6.14(b), on the other hand, shows that the simulation time was practically constant (near  $t_{max}$ ) until the 80th episode, where it starts to reduce strongly until the 200th episode. Beyond this point the time reduces more slowly and stabilizes near 65 seconds after the 600th episode. These results show that the robot was able to accomplish the task at the end of the training process, because 60 seconds is the minimum time required to perform this task (i.e., to shoot a ball into the goal) using the simulation conditions described above.

Figure 6.15 shows an example of robot trajectory during the task (a circle marks the ball position in the first image of this figure). The number of probabilistic neurons added by IGMN during the learning process was 138.32 in average, and the time required to execute each experiment (i.e., to perform 1000 episodes) was approximately 2.5 hours.

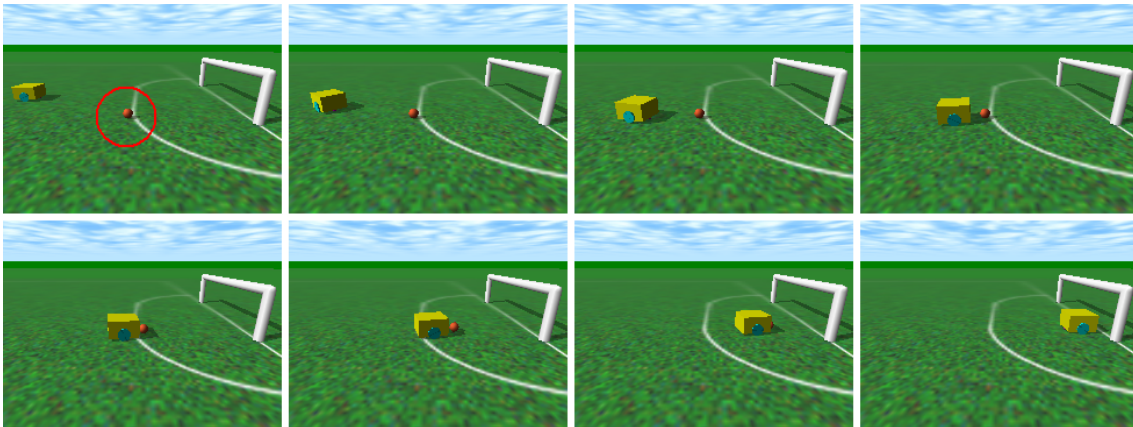


Figure 6.15: Example of robot trajectory during the task

## 6.5 Feature-based mapping using IGMN

Map building is a fundamental problem in mobile robotics, in which a robot must memorize the perceived objects and features, merging corresponding objects in consecutive scans of the local environment (THRUN, 2002). There are several approaches to solve the map building problem. Among those are occupancy grid and feature-based maps. The occupancy grid maps are generated from stochastic estimates of the occupancy state of an object in a given grid cell (THRUN; BURGARD; FOX, 2006). They are relatively easy to construct and maintain, but in large environments the discretization errors, storage space and time requirements become matters of concern. Feature-based maps, on the other hand, model the environment by a set of geometric primitives such as lines, points and arcs (MEYER; FILLIAT, 2003). Segment-based maps, which are the most common type of feature-based maps, have been advocated as a way to reduce the dimensions of the data structures storing the representation of the environment (AMIGONI; FONTANA; GARIGIOLA, 2006). Its main advantage over occupancy grid maps is that line segments can be represented with few variables, thus requiring less storage space. Moreover, line segments are also easy to extract automatically from range data.

However, segment-based maps are not able to give closed and connected regions like occupancy grid maps because some objects do not provide line segments. Moreover, the number of extracted line segments is very high if the environment is irregular (not composed only by straight walls) and/or the range data is quite noisy. Another disadvantage of segment-based maps is the absence of probabilistic information in the generated map (although some researchers (GASÓS; MARTÍN, 1997; IP et al., 2002) have used fuzzy sets to deal with uncertainty in the mapping process). In fact, according to Thrun (2006), probabilistic approaches are typically more robust in face of sensor limitations, sensor noise, environment dynamics, and so on. Other localization and mapping techniques, such as particle filters and potential fields, generally use grid maps to represent the environment, and therefore have the same restrictions pointed out above.

This section presents a new feature-based mapping algorithm, published in Heinen and Engel (2011; 2010c), which uses the IGMN probabilistic units to represent the features perceived in the environment. This kind of representation, which is inherently probabilistic, is more effective than segment-based maps because it has an arbitrary accuracy (it does not require discretization) and can even model objects that do not provide line segments. Moreover, the proposed mapping algorithm does not require an exclusive kind of sensor (it can be used either with laser scanners or sonar sensors), requires low storage space and is very fast, which allows it to be used in real time. The remainder of this section is organized as follows: Subsection 6.5.1 describes some previous feature-based mapping techniques; Subsection 6.5.2 describes how IGMN can be used to create feature-based maps; and Subsection 6.5.3 describes some experiments performed to evaluate the proposed mapping algorithm using sonar and laser range data.

### 6.5.1 Related work

In the last decade several feature-based mapping algorithms have been proposed to solve the map building problem. In Zhang and Ghosh (2000) a segment-based mapping algorithm is proposed that describes a line segment using the center of gravity of its points and the direction  $\theta$  of its supporting line. This algorithm groups laser points in clusters, and for each cluster, a line segment is generated. In Lee et al. (2005) a feature based mapping algorithm is presented which uses an association model to extract lines, points and arc features from sparse sonar data. In Puente et al. (2009) a mapping algorithm is presented which uses a segmentation algorithm derived from computer vision techniques to extract geometrical features from laser range data. In Latecki et al. (2004) a mapping algorithm is proposed which represents the environment by polygonal curves (polylines).

In Amigoni, Fontana and Garigiola (2006) a method derived from the Lu and Milos' algorithm (LU; MILIOS, 1998) is presented for building segment-based maps that contain a small number of line segments. In Amigoni, Gasparini and Gini (2006) an algorithm is proposed to build a global geometric map by integrating scans collected by laser range scanners. This method, which considers scans as collections of line segments, works without any knowledge about the robot pose. In Luo et al. (2008) an indoor localization method based on segment-based maps is proposed. It works in four steps: clustering scan data; feature extraction from laser data; line-based matching; and pose prediction. But this method assumes that the environment map already exists, i.e., it neither creates nor updates the map.

In Lorenzo et al. (2004) a method is proposed to solve the SLAM (Simultaneous Localization and Map Building) problem based on segments extracted from local occupancy maps, in which line segments are categorized as new obstacle boundaries of a simultane-



ously built global segment-based map or as prolongations of previously extracted boundaries. In Meyer-Delius and Burgard (2010) a point-based representation is described, in which the data points gathered by the robot are directly used as a non-parametric representation. To reduce the large memory requirements necessary to store all data points, an off-line algorithm based on the fuzzy  $k$ -means is used to select the maximum-likelihood subsets of data points.

As described above, the main limitation of all these feature-based mapping techniques is the absence of probabilistic information in the generated map. To avoid this limitation, Gasós et al. use fuzzy-segments to represent uncertainty in the feature positions (GASÓS; MARTÍN, 1997; GASÓS; ROSSETI, 1999). This model extracts segments from points provided by sonar sensors, which are modeled on the map using fuzzy-sets. In the model proposed by Ip et al. (2002), on the other hand, an adaptive fuzzy clustering algorithm is used to extract and classify line segments in order to build a complete map for an unknown environment.

Although these mapping techniques are able to create good representations in simple environments composed by straight walls, most of them are not able to build the map in real-time while the robot navigates in the environment (i.e., they are off-line solutions). The mapping algorithm proposed in this subsection, on the other hand, is able to build environment representations in real time and incrementally. Moreover, it is inherently probabilistic and does not assume a specific environment structure. The next subsection describes how IGMN can be used in a feature-based mapping algorithm.

### 6.5.2 Incremental feature-based mapping using IGMN

This subsection describes a geometric-based mapping algorithm which uses the IGMN units (also called mixture model components) to represent the features (objects, walls, etc) of the environment. Figure 6.16 shows the general architecture of the mapping algorithm. It consists mainly of two IGMN's, the local model, which represents the local perception of the robot, and the global model, which represents the global environment map. Each IGMN has two cortical regions,  $\mathcal{N}^A$  and  $\mathcal{N}^B$ . The cortical region  $\mathcal{N}^A$  tackles the values of the sensor readings and the cortical region  $\mathcal{N}^B$  receives the odometric information provided by the Pioneer 3-DX simulator.

Initially both models are empty. When a sensor reading arrives (which can be a laser scan or a sonar reading) it is transformed into object locations on a global coordinate system based on the robot position estimated by the dead reckoning system. These object locations are grouped in clusters using the IGMN algorithm, thus composing the local model. When a specified number  $N$  of sensor readings arrives (e.g., 10 laser scans or 100 sonar readings), the local model is matched against the global model. If the global model is still empty, all local units are added to it and deleted from the local model. Otherwise the robot pose is adjusted to minimize the differences between the local and global models using the component matching process described in the next subsection. Both IGMN models are then merged into the global model and the local model is emptied. When a new sensor reading arrives, all these steps are repeated, and so the global model is updated at each  $N$  readings.

All this process occurs in real time at normal sensor arriving speeds (e.g., one laser scan at each 100 milliseconds) even with more than 500 Gaussian units in the IGMN models. In fact, the prototype was able to perform all these operations (including the matching and merging processes) in less than 30 milliseconds on the same typical computer described before. In relation to memory requirements, the proposed mapping algo-

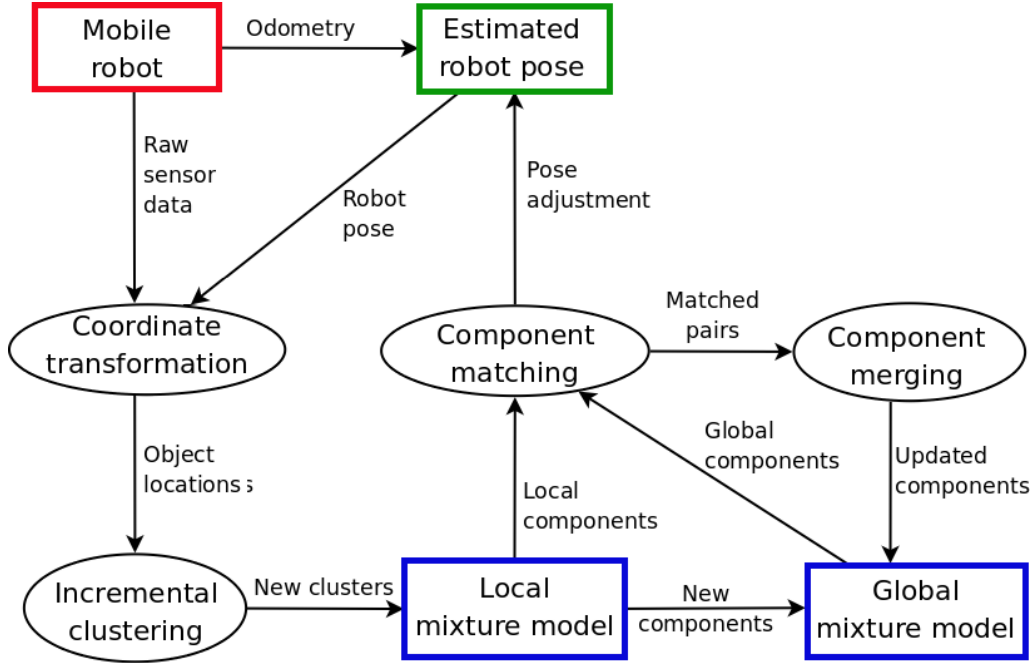


Figure 6.16: General architecture of the proposed mapping algorithm

rithm is very parsimonious, requiring just eight floating point numbers to store each two-dimensional Gaussian distribution ( $D^2 + D + 2$  floating-point variables, where  $D = 2$  is the dimension of the map).

#### 6.5.2.1 Component matching

In the proposed mapping algorithm, the robot maintains knowledge about its pose using data coming from odometry, which is an unreliable source of information (THRUN; BURGARD; FOX, 2006). In order to refine the estimated robot poses, the local model, which represents the robot current view, is matched against the global model, and the differences are used to adjust the estimated pose of the robot and the local units.

The matching process used here is inspired in the Lu and Milos' algorithm (LU; MILIOS, 1998) and works in the following way. Initially, all spurious components (i.e., those units which represent few data points and/or noise) of the local model are excluded. A unit  $j$  is considered spurious if its density is lower than a minimum value  $\rho_{min}$ . The density  $\rho_j$  of a cluster  $j$  is computed using the following equation adapted from Gath and Geva (1989):

$$\rho_j = \frac{sp_j}{(2\pi)^{D/2} \sqrt{|\mathbf{C}_j|}} \quad (6.5)$$

where  $D$  is the dimensionality of the data points (2 for a two-dimensional map),  $\mathbf{C}$  is the variance/covariance matrix of the  $j$  unit, and  $sp_j$  is the accumulator of the a posteriori probabilities.

A low cluster density  $\rho_j$  indicates that the cluster  $j$  represents few data points compared to its size, which in our case may represent that  $j$  was generated by few and sparse (possibly noisy) sensor readings. Moreover, if a cluster is almost parallel to the sensor beam, i.e., the angles of the cluster main axis (computed using its eigenvectors) and the sensor beam (the line segment linking the sensor to the center of the cluster) differ by less than a user-specified parameter  $\alpha_{min}$ , this cluster is also considered spurious because the readings are not reliable.

The next step in the proposed algorithm is to find equivalent components in both models, i.e., those clusters corresponding to the same environment features. Two clusters are considered equivalent if: (i) the difference between the cluster orientations is lower than a user-specified parameter  $\alpha_{max}$ ; (ii) the Mahalanobis distance between the center  $\boldsymbol{\mu}$  of each unit (computed using the covariance matrix  $\mathbf{C}$  of the global model) is lower than a user-specified parameter  $\eta_{max}$ . Those global units invisible from the current robot viewpoint (i.e., the orientation of the segment(s) linking the sensor(s) to the cluster is out of range) are not considered here. The output of this process is a list containing the pairs of equivalent Gaussian components in both models.

The new robot pose is computed using the Powell's direction set method (POWELL, 1964; PRESS et al., 1992), in order to minimize the discrepancies  $E_{lg}$  among the equivalent global  $g$  and local  $l$  units, i.e.:

$$E_{lg} = \frac{1}{M} \sum_{i=1}^M (\alpha_{gi} - \alpha_{li})^2 + \mathcal{D}_{\mathcal{M}}(\boldsymbol{\mu}_{gi}, \boldsymbol{\mu}_{li}) \quad (6.6)$$

where  $M$  is the number of matching pairs and  $\mathcal{D}_{\mathcal{M}}(\cdot)$  is the Mahalanobis distance computed using the covariance matrix of the global unit  $\mathbf{C}_g$ . The robot pose derived from the odometry is used as an initial guess in the minimization process. In general the Powell's method converges fast if this initial guess is reasonable, which is the case here because odometric information is reliable in short distances.

### 6.5.2.2 Component merging

In this procedure, the local components which are not spurious are inserted into the global model, thus expanding it to reflect the new perceived environment features. This insertion procedure occurs in two different steps. Initially the global clusters which have equivalents in the local model (i.e., those that are present in the list of matching pairs described previously) are updated using the following equations (DECLERCQ; PIATER, 2008):

$$\boldsymbol{\mu}_g^* = \frac{sp_g \boldsymbol{\mu}_g + sp_l \boldsymbol{\mu}_l}{sp_g + sp_l} \quad (6.7)$$

$$\mathbf{C}_g^* = \frac{sp_g \mathbf{C}_g + sp_l \mathbf{C}_l}{sp_g + sp_l} + \frac{sp_g \boldsymbol{\mu}_g \boldsymbol{\mu}_g^T + sp_l \boldsymbol{\mu}_l \boldsymbol{\mu}_l^T}{sp_g + sp_l} - \boldsymbol{\mu}_g^* \boldsymbol{\mu}_g^{*T} \quad (6.8)$$

$$sp_g^* = sp_g + sp_l \quad (6.9)$$

where  $\boldsymbol{\mu}_g$  is the old mean vector of the global unit  $g$  and  $\boldsymbol{\mu}_g^*$  is the new mean vector computed by (6.7) at time  $t$ . The local clusters that have no counterparts in the global model (i.e., those that represent new environment features) are simply inserted into the global model without modification. This also occurs in the beginning of the mapping process (when the global model is empty).

After merging, all local units are removed from the local model and the prior probabilities  $p$  of all global units are updated by:

$$p_g^* = \frac{sp_g}{\sum_{j=1}^M sp_j} \quad (6.10)$$

where  $M$  is the current number of global units. This global model represents the environment using occupancy probabilities, i.e., for each  $(x, y)$  coordinate it is possible to compute the probability that an obstacle (e.g., wall) is present in that location. Next subsection describes several experiments performed to evaluate the performance of the proposed mapping algorithm in robot mapping tasks.

### 6.5.3 Experiments

This subsection describes some experiments performed to evaluate the proposed mapping algorithm using two kinds of sensory information: (i) data provided by a simulated laser scanner; and (ii) data provided by sonar sensors. The robot used in these experiments is a Pioneer 3-DX, shown in Figure 6.3 above. This robot has a Sick LMS-200 laser scanner installed on it, which in ideal conditions is capable of measuring out to 80m over a  $180^\circ$  arc. Figure 6.17 shows the real environment used in the simulation. It is composed by two long corridors of  $2.3 \times 30$  meters linked by two short corridors of  $2.3 \times 10$  meters, as shown in the schematic map presented in Figure 6.17(c). This environment has several irregularities (e.g. doors, saliences and printers) which difficult the mapping process.

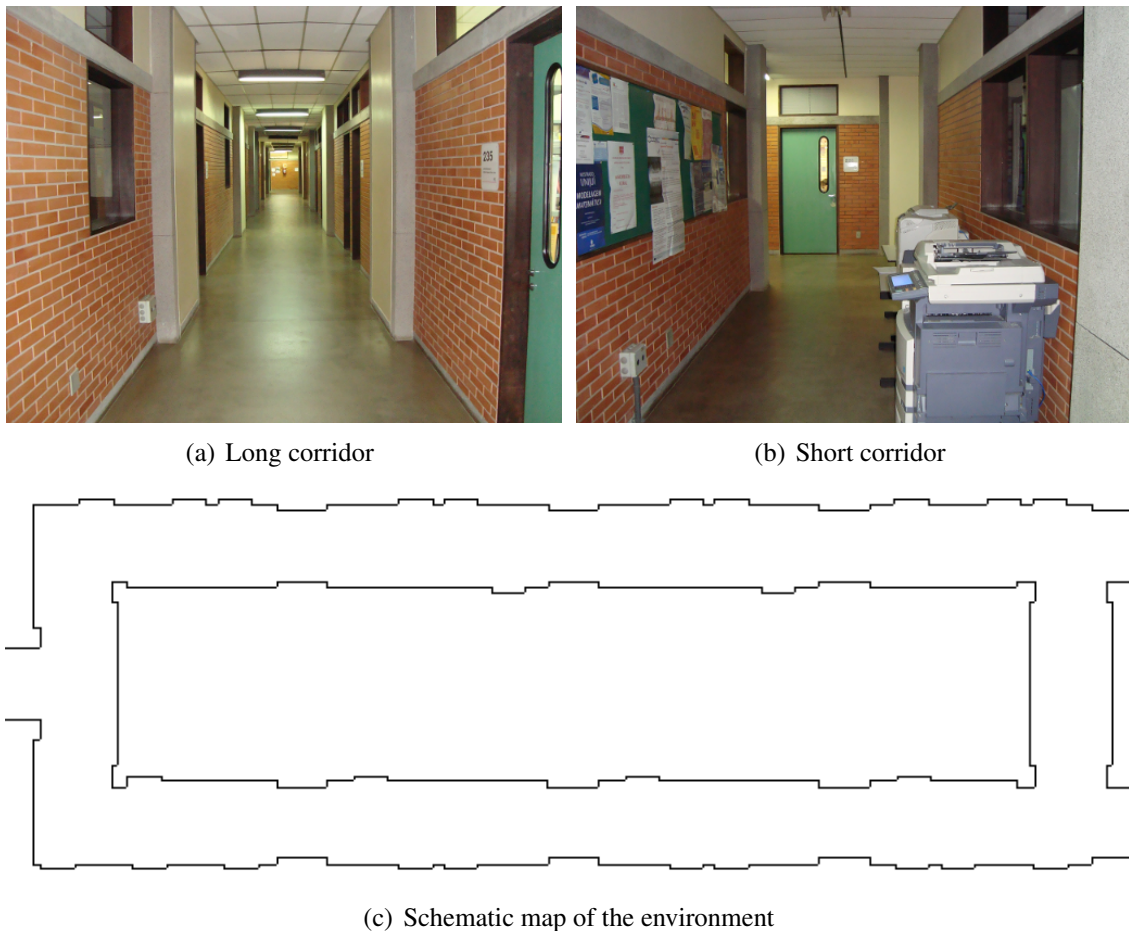


Figure 6.17: Environment used in the feature-based mapping experiments

The configuration parameters used in all experiments are:  $\delta = 0.1$ ;  $\rho_{min} = 10^{-5}$ ;  $\alpha_{min} = 10^\circ$ ;  $\alpha_{max} = 10^\circ$  and  $\eta_{max} = 2$  (the proposed algorithm is not sensible to these parameters). The only configuration parameter that needs to be adjusted to each experiment is the  $\tau_{nov} \in (0, 1]$  parameter, which in this case defines the granularity of the model, i.e., small values ( $\ll 0.01$ ) produce few larger clusters, and larger values ( $\geq 0.01$ ) produce several little clusters. As in Section 6.1, here we decided to use the  $\tau_{nov}$  parameter instead of  $\varepsilon_{max}$  because in these experiments the goal is to find natural groupings (i.e., clusters) rather than function approximation. In the next subsections we describe experiments performed using data provided by a simulated laser scanner and a sonar range data.

### 6.5.3.1 Experiments using simulated laser scanner data

This subsection describes two experiments performed using sensor data provided by a simulated laser scanner that is equivalent to the Sick LMS-200 installed on the real Pioneer 3-DX robot. In these experiments, the robot was manually controlled to perform one loop in the simulated environment shown in Figure 6.17(c). A complete laser scan is received at each 100 milliseconds, and the mapping process is performed at each second (i.e., using 10 complete scans). The first experiment was conducted using  $\tau_{nov} = 10^{-8}$ , and this small value produced 9 large clusters, as can be seen in Figure 6.18.

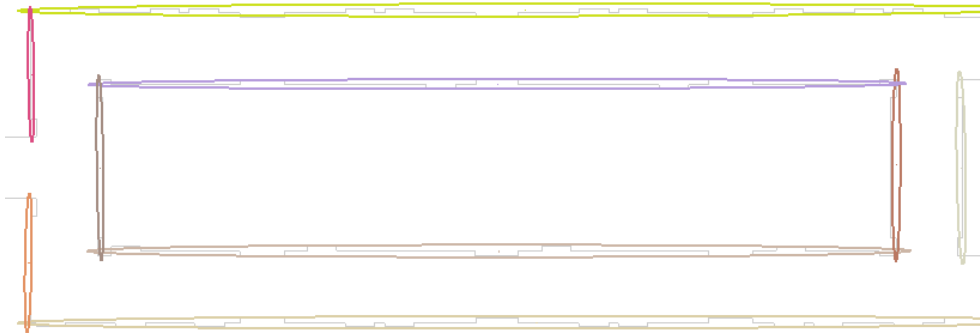


Figure 6.18: Gaussian distributions generated using laser data ( $\tau_{nov} = 10^{-8}$ )

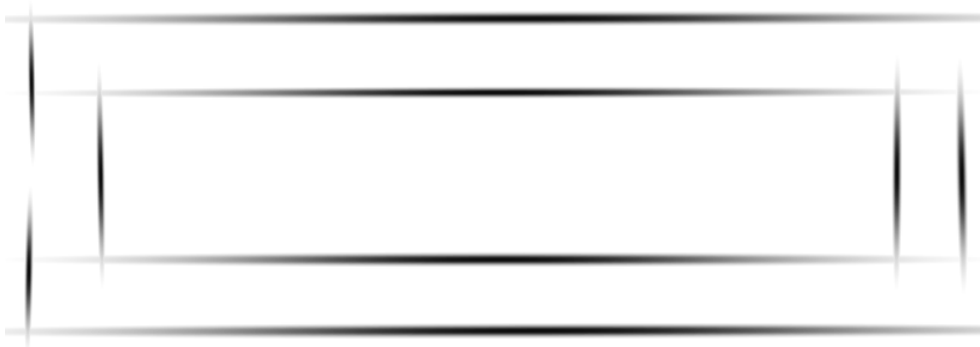


Figure 6.19: Occupancy probabilities of Figure 6.18 model

In this figure, each cluster is represented by an ellipse whose width is equivalent to a Mahalanobis distance of two. The occupancy probabilities of this map are graphically shown in Figure 6.19, where darker regions represent higher occupancy probabilities (close to 1) and lighter regions correspond to probabilities close to 0. It is important to highlight that the proposed mapping algorithm does not have any random initialization and/or decision, and thus the obtained results are always identical for the same dataset and configuration parameters.

The next experiment was performed using the same conditions described above, but using a larger  $\tau_{nov} = 10^{-2}$  value which makes the system more sensible to small variations in the laser data. The results obtained in this experiment are shown in Figures 6.20 and 6.21. It can be noticed from Figure 6.20 that much more clusters were generated in this experiment (76 Gaussian components were generated). Nevertheless, these clusters fit very well the environment features, existing almost one cluster for each feature (doors entrances, saliences in the walls, etc.). Moreover, each wall is modeled by a thin, long cluster which closely represents the center of the wall.

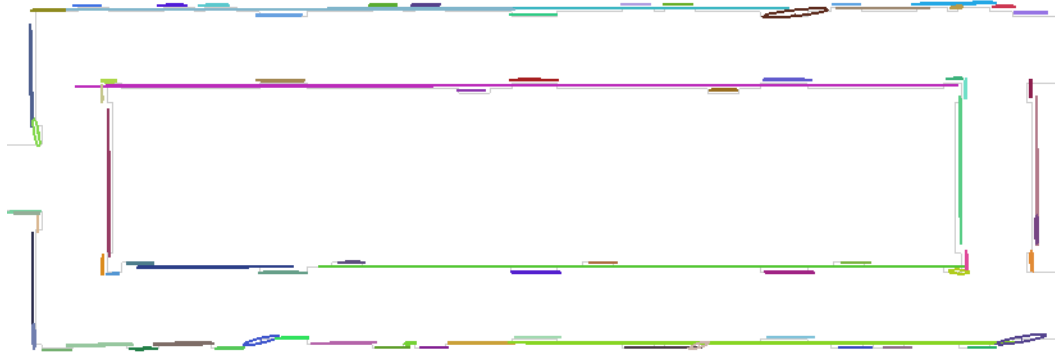


Figure 6.20: Gaussian distributions generated using laser data ( $\tau_{nov} = 10^{-2}$ )



Figure 6.21: Occupancy probabilities of Figure 6.20 model

These experiments show that the proposed mapping algorithm is able to create useful representations of the environment, and these representations can be coarse (Figure 6.18) or fine (Figure 6.20) depending on the  $\tau_{nov}$  configuration value. Although these experiments were performed using high quality simulated laser range data, the proposed mapping algorithm is not restricted to this kind of sensory data. In the next subsection we describe an experiment performed using this same environment, but with data provided by real sonar sensors, which are less accurate and noisier than laser scanners.

### 6.5.3.2 Experiments using real sonar data

In these experiments, the proposed mapping algorithm is evaluated using data provided by the sonar sensor array of the real Pioneer 3-DX robot. The time interval of each scan is 100 milliseconds, and the local model is generated using 100 complete scans (i.e., at each 10 seconds). Figure 6.22 shows the global model generated after one loop in the environment, and Figure 6.23 illustrates the occupancy probabilities of this map, where darker regions represents higher probabilities. The configuration parameters used in this experiment are the same of the previous one (i.e.,  $\tau_{nov} = 10^{-2}$ ).

We can notice that even with many noise sources present in the environment, the proposed mapping algorithm was able to create a reasonable map of the environment using these very noisy sensory data. The results obtained in Figure 6.22 show that the proposed mapping algorithm does not require just laser scanner sensors, but of course the quality of the maps is superior when high quality sensor data is used.

Although in this experiment more Gaussian distributions were generated (216), this is yet a small number compared to the number of cells necessary to map this environment

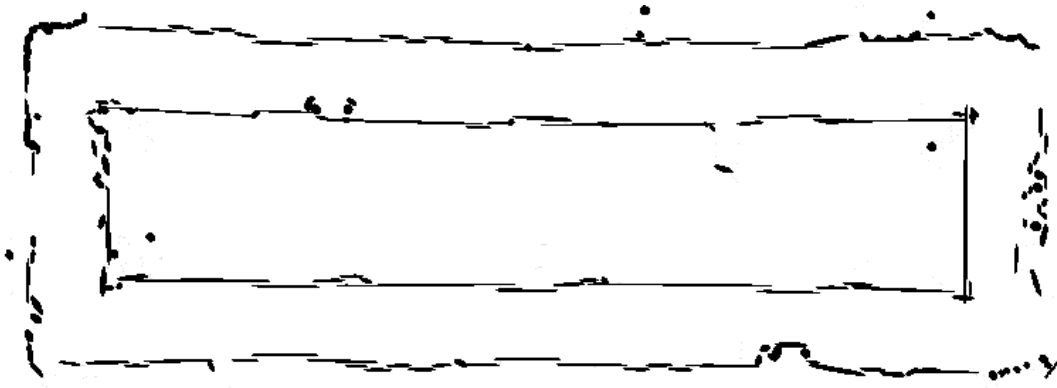


Figure 6.22: Distributions generated using sonar sensors ( $\tau_{nov} = 10^{-2}$ )

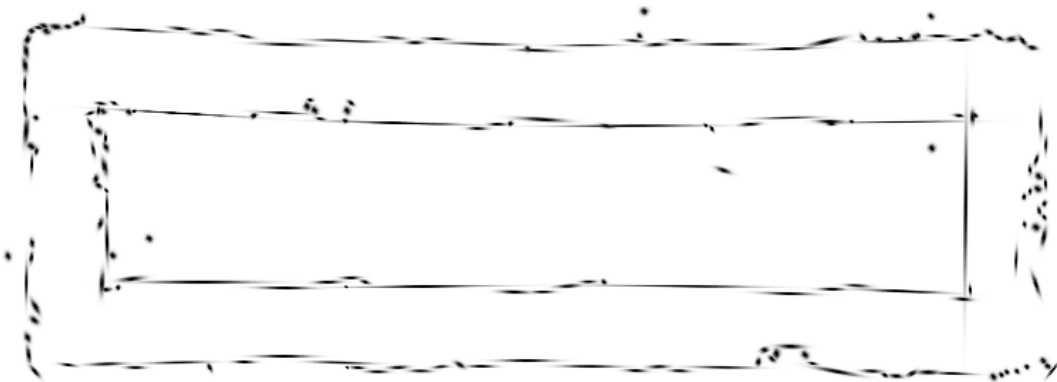


Figure 6.23: Occupancy probabilities of Figure 6.22 model

using grid-based mapping techniques. In fact, to obtain a similar performance using a occupancy grid maps it would be necessary a large number of cells. To demonstrate this, Figure 6.24 shows a grid map generated using the same real data that generated the map presented in Figure 6.23, i.e., the real sensor readings adjusted using the matching technique proposed in this paper. Each cell in Figure 6.24 represent an area of  $20 \times 20$  centimeters, and  $200 \times 80 = 16000$  were necessary to represent this map. It can be noticed in Figure 6.24 that even using 16000 cells the map resolution is inferior than that presented in Figure 6.23. Moreover, the number and size of the grid cells must be previously informed and kept fixed.

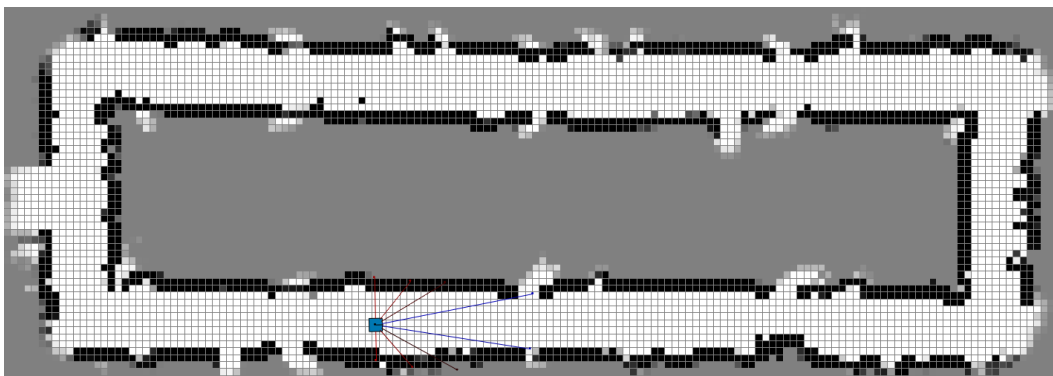


Figure 6.24: Grid map generated using the same data of Figure 6.24

A interesting characteristic of the mapping algorithm proposed here is that the occupancy probabilities of the unvisited regions are near zero, whilst using a grid-based map these regions will have occupancy probabilities near to 50%. This characteristic is a natural consequence of representing just the occupied regions in the environment map, and is also shared by other kinds of feature-based maps such as segment-based maps. However, this characteristic is very useful because it encourages the exploration of unvisited regions. In fact, if a region has an occupancy probability near zero the robot will try to navigate through this region, and then it finds out the actual structure of this region and the map will be improved and/or expanded. Hence, this characteristic allows an exploration strategy that follows the Autotelic principle (STEELS, 2004), which states that an agent must be self-motivated for learning and improving its skills. Moreover, this strategy is not based on *ad-hoc* choices nor configuration parameters.

### 6.5.3.3 Complex simulated environments

The next experiment was performed using the Pioneer 3-DX simulator software AR-COS (Advanced Robot Control & Operations Software) and the “AMROffice.map” environment (Figure 6.25), a complex map generated using real data and distributed with the robot simulator. The time interval of each scan is 100 milliseconds, and the local mixture model is generated using 100 complete scans.

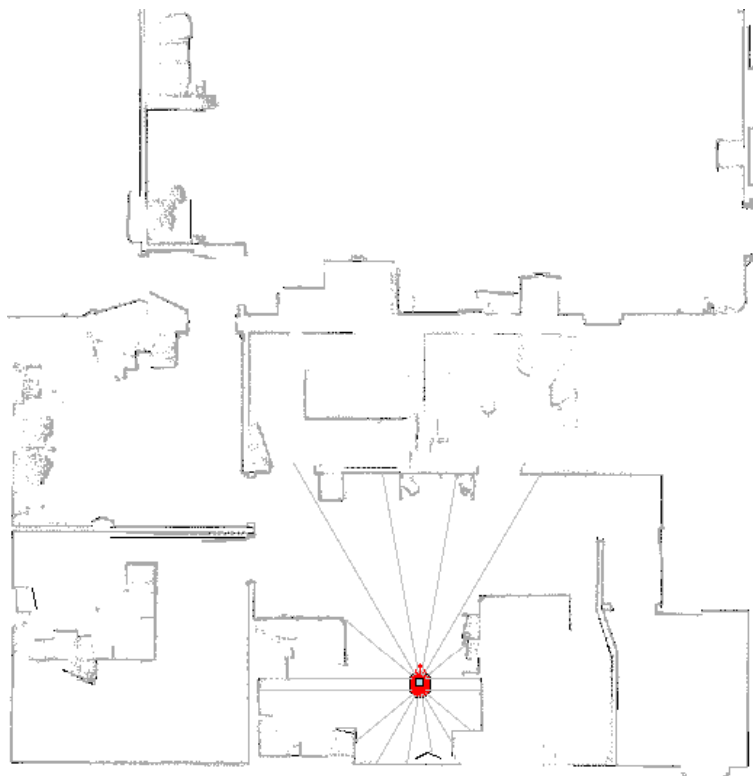


Figure 6.25: AMROffice environment

Figure 6.26(a) shows the map generated in the first experiment. The number of Gaussian mixtures generated in the global model was 364, which is a small value compared to the number of cells necessary to map this environment using grid-based mapping techniques. The occupancy probabilities of this map are graphically shown in Figure 6.26(b), where darker regions represent higher occupancy probabilities (close to 1) and lighter regions correspond to probabilities close to 0.



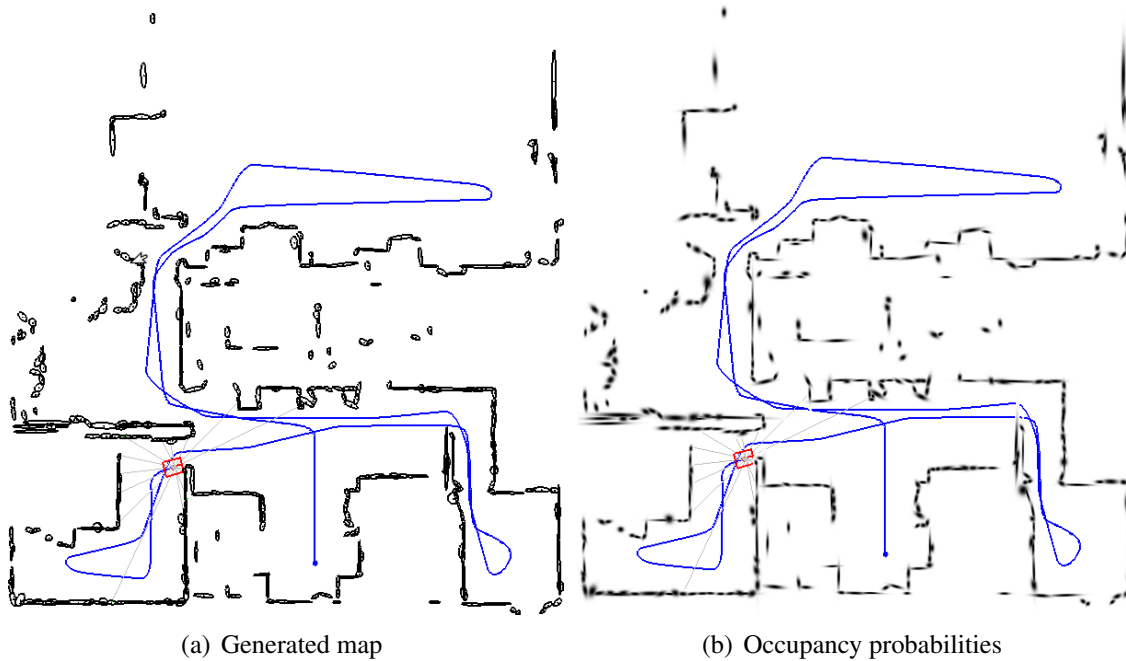


Figure 6.26: Results obtained in the AMROffice environment

The proposed model has also obtained good results in the irregular environment presented in Figure 6.9, as shows Figure 6.27. We can notice in Figure 6.27 that the Gaussian distributions are perfectly aligned with the external wall of the environment. In this figure we have represented the map using rectangles rather than ellipses to better visualize the alignment of the Gaussian units with the modeled structures. Moreover, we can see in

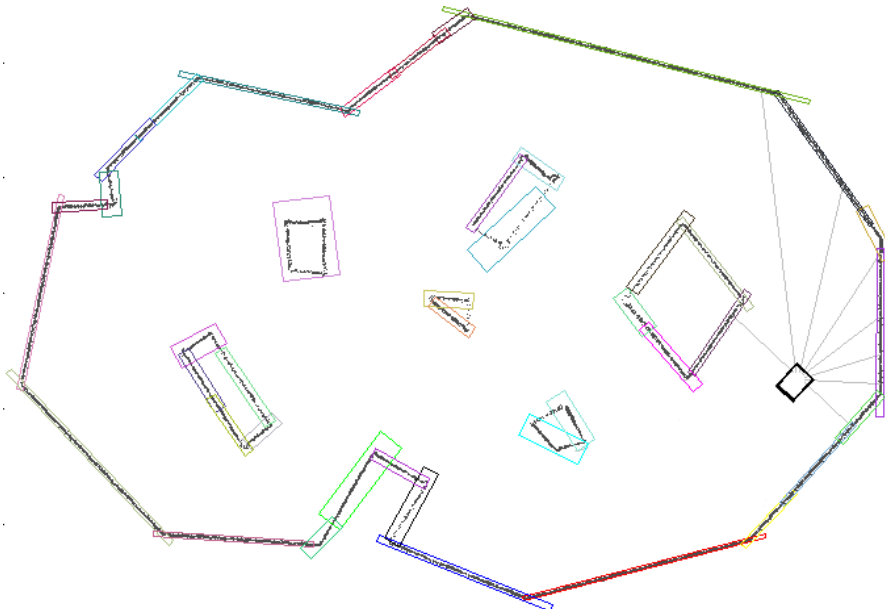


Figure 6.27: Map generated in an irregular environment

Figure 6.28, which shows the occupancy probabilities by ellipses, that the internal objects were modeled using larger distributions. This occurs because the perception of the robot is occluded in some parts of the environment, which prevents the mapping algorithm to accurately represent the internal faces of some objects (the trajectory followed by the



## 7 CONCLUSION AND FUTURE WORK

This monograph has presented IGMN, a new connectionist approach for incremental function approximation and on-line prediction, that is the main contribution of this thesis. IGMN is inspired on recent theories about the brain, specially the Memory-Prediction Framework (MPF) (HAWKINS, 2005) and the constructivist artificial intelligence (DRESCHER, 1991), which endows it with some unique features that are not present in other ANN models such as MLP, RBF and GRNN. More specifically in IGMN we don't use the words *input* and *output* to represent the data features of a training sample such as  $\mathbf{z} = \{\mathbf{a}, \mathbf{b}\}$ , for instance. Instead, we consider that the data vectors  $\mathbf{a}$  and  $\mathbf{b}$  are different sensory and/or motor modalities with distinct domains, and one modality (e.g.  $\mathbf{a}$ ) can be used to estimate another (e.g.  $\hat{\mathbf{b}}$ ). Moreover, IGMN is based on strong statistical principles (Gaussian mixture models) and asymptotically converges to the optimal regression surface as more training data arrive.

To validate the proposed model, several experiments were performed using synthetic data (e.g., sinusoid datasets) and these experiments have demonstrated that: (i) IGMN learns incrementally using a single scan over the training data; (ii) It does not require to fine tune its configuration parameters (in fact just a single and easy to adjust configuration parameter must be set); (iii) IGMN is relatively robust to the order of presentation of data; (iv) the proposed model can provide the confidence levels of its estimates; and (v) the IGMN performance is comparable to those of other ANN models (e.g., MLP, RBF, GRNN) but without requiring that the training data set be complete and available at the beginning of the learning process.

IGMN was also tested in practical applications such as: (i) the identification of a nonlinear plant; (ii) time series prediction; (iii) concept formation; (iv) reinforcement learning; and (iv) robotic mapping and control, and these experiments have shown that IGMN can be used successfully in applications that require incremental learning and/or real time performance. More specifically, the performed experiments have demonstrated that IGMN has the following advantages over the existing connectionist approaches:

**The IGMN learning algorithm is very aggressive.** As said before, IGMN learns incrementally using a single scan over the training data, i.e., each training pattern can be immediately used and discarded. Moreover, IGMN can produce reasonable estimates based on few training data, and these estimates are improved as new training data arrive.

**The learning process can proceed perpetually.** Unlike other ANN models, IGMN does not require separate phases for learning and recalling, i.e., the learning process can proceed perpetually without suffering from catastrophic interference. Hence, the proposed model can always improve its performance as new data arrive and consequently the neural network can adapt itself to changes in the environment.

**IGMN is relatively robust to overfitting.** As occurs in GRNN, the equations used to compute the estimates are relatively robust to overfitting. In fact, even if a Gaussian unit is added for each training pattern the computed approximation will not be significantly affected by the noise (specially if we consider a white noise), because the estimate will be a soft interpolation weighted by the a posteriori probabilities. Moreover the estimates are not affected by infrequent extreme values because these values have low a posteriori probabilities.

**It is not required to fine-tune the configuration parameters.** In IGMN just the maximum instantaneous approximation error  $\varepsilon_{max}$  must be adjusted for each training data set, but this configuration parameter is easy to set because in general we know the maximum error allowed in a given task, specially if this error is normalized.

**The proposed model does not depend on the initial conditions.** Unlike other ANN models such as MLP and SOM, IGMN does not use any random initialization and/or decision, and hence the obtained results are always identical for the same data set and configuration. Thus, using IGMN we do not repeat an experiment several times using different random initializations.

**IGMN is not is susceptible to local minima.** The IGMN learning algorithm is not based on error minimization nor gradient descent, and thus it does not seek for a minimum in the error surface. Instead, the IGMN learning algorithm finds out correlations among different stimuli and uses these correlations to compute the estimate of a sensory/motor stimulus.

**The neural network topology is defined automatically and incrementally.** IGMN starts with a single neuron in each region and more neurons are added whenever necessary based on an error driven mechanism. Thus, in the proposed model the user does not have to manually configure the neural network topology and this topology does not have to be fixed.

**IGMN can handle the stability-plasticity dilemma.** As described before, in IGMN an error driven mechanism is used to decide if a new unit must be added to accommodate a new information or if we just need to adjust the existing units. Therefore IGMN is stable because it preserves the acquired knowledge in the existing units and is plastic because it creates new units to accommodate new information without forgetting the previous experience.

**It does not suffer from catastrophic interference.** IGMN does not suffer from catastrophic interference because the acquired knowledge is local, i.e., the changes in the ANN parameters are restricted to a specific region of the state-space.

**The proposed neural network model is robust to the order of presentation of data.** As demonstrated in Section 5.1, IGMN can produce good estimates if the training data is presented both in order or shuffled. Thus, the proposed model can be used in on-line systems, where the training data is received in a specific order, or in systems where the data is received completely shuffled (e.g., off-line systems) as well.

**IGMN uses a probabilistic framework.** IGMN is based on a probabilistic framework (Gaussian mixture models), and as occurs in GRNN, it approximates the optimal Bayesian decision using the available training data. In other words, IGMN produces the maximum a posteriori hypothesis computed over the data received to the moment, and thus its estimates can be consider as statistically optima.

**The proposed model creates useful internal representations.** The representations created in the cortical regions of IGMN correspond to natural groupings (i.e. clusters) of the state space that can be interpreted by a human specialist. Hence, IGMN is not a *black box*, i.e., we can interpret and justify the decisions taken by the neural network by analyzing the posterior probabilities computed at each node of the neural network.

**IGMN can provide the confidence level of its estimates.** IGMN can inform not just a punctual estimate (i.e., the *expected* value) of a sensory/motor stimulus but also the variance/covariance of this estimate. Therefore IGMN can inform the confidence levels of its estimates, thus allowing us to take better decisions based on the confidence intervals rather than on a single punctual estimate.

**The proposed model can be used in supervised, unsupervised or reinforcement learning tasks.** As demonstrated through several experiments, IGMN can be used in both supervised and unsupervised learning tasks. Moreover, IGMN can also be used as a function approximator in reinforcement learning tasks.

**Bidirectional information flow.** Unlike most ANN models, in IGMN the information flow is not unidirectional from input to output, i.e., the same partially trained neural network (as IGMN can learn continuously we never consider that the training process has finished) can be used to estimate the values of a function  $f(\cdot)$  and its corresponding inverse  $f(\cdot)^{-1}$ .

**IGMN can solve forward and inverse problems.** As said in Section 4.4, in many potential applications of neural networks there is a well-defined *forward* problem which is characterized by a *functional* (i.e. single-valued) mapping, but the corresponding *inverse* problem is multi-valued (BISHOP, 1995). Although traditional ANN models (e.g., MLP, RBF and GRNN) cannot be used in multi-valued problems, IGMN can provide valid answers even in regions of the state-space where the target data are multi-valued.

**The proposed model has good computational performance.** Although IGMN requires inverting many covariance matrices at each time, we can reduce the computational complexity of the neural network by using separate cortical regions for each sensory/motor modality. Thus IGMN can have good computational performance, which allows its use in real time and critical control applications.

Based on the advantages described above we can note that the main objectives of this thesis were achieved, i.e., we were able to develop a new connectionist approach for incremental function approximation that learns from data flows. In fact the proposed model can be used successfully in many potential applications such as incremental regression, on-line prediction, system identification, time series prediction, concept formation and robotic tasks. Moreover, the performed experiments have shown that the efficiency of IGMN and its representational power can expand the set of potential tasks in which the neural networks can be applied, thus opening new research directions in which important contributions can be made, as will be described in the next section.

## 7.1 Future work

There are many research directions in which this work can be continued and/or used to improve the results obtained by other connectionist approaches. Some of these research directions in which it is possible to obtain important contributions are:

- *Hierarchic representation*: The proposed model can be expanded to create hierarchies of Gaussian units in a similar way to those proposed in the memory prediction framework (MPF) (HAWKINS, 2005; HAWKINS; DILEEP, 2006; PINTO, 2009);
- *Hierarchical reinforcement learning*: The RL algorithm proposed in Section 6.4 can be modified into a hierarchical RL algorithm (BARTO; MAHADEVAN, 2003; MOERMAN, 2009), thus improving the learning performance by dividing the complete state space in subregions;
- *Time series prediction*: IGMN can be used to predict more complex, chaotic and cyclic time series;
- *The inverse kinematics problem*: to use IGMN for computing the inverse kinematics of more complex robotic arms composed by many joints and with six degrees of freedom;
- *Simultaneous localization and mapping (SLAM)*: The feature-based mapping algorithm presented in Section 6.5 can be expanded to a SLAM solution by using IGMN to predict the robot's position as it navigates through the environment;
- *Multi-hypothesis tracking*: IGMN can implement a global localization mechanism (FILLIAT; MEYER, 2003) that allows the robot to recover its position even from the *kidnapped robot problem* (THRUN; BURGARD; FOX, 2006).
- *Autotelic exploration*: to implement an exploration algorithm based on the Autotelic principle (STEELS, 2004) that self-motivates the robot to explore new regions and/or to develop new skills;
- *Attentional mode neural network*: IGMN can be combined with the attentional mode neural network (AMNN) (ENGEL, 1996) in order to improve both models and to obtain more stable results in control tasks (currently AMNN learns faster than RL (SUTTON; BARTO, 1998), but it suffers from catastrophic interference);
- *Classification*: although in this thesis we are interested just in regression, IGMN can be easily adapted to be used in supervised classification tasks.

Hence we can notice that there are many improvements that can be made over the work presented in this thesis, and hence we can affirm that IGMN is a very useful machine learning tool that expands the horizon of tasks in which the artificial neural networks can be used successfully and efficiently.

## REFERENCES

AMIGONI, F.; FONTANA, G.; GARIGIOLA, F. **Distributed Autonomous Robotic Systems (DARS)**. Berlin, Germany: Springer-Verlag, 2006. v.7, p.11–20.

AMIGONI, F.; GASPARINI, S.; GINI, M. Building Segment-Based Maps Without Pose Information. **Proceedings of the IEEE**, Los Alamitos, CA, USA, v.94, n.7, p.1340–1359, July 2006.

ARANDJELOVIC, O.; CIPOLLA, R. Incremental Learning of Temporally-Coherent Gaussian Mixture Models. In: BRITISH MACHINE VISION CONFERENCE (BMVC), 16., Oxford, UK. **Proceedings...** British Machine Vision Association (BMVA), 2005. p.759–768.

ASADA, M. et al. Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. **Machine Learning**, Boston, MA, USA, v.23, p.279–303, 1996.

ASADA, M. et al. A Humanoid Approaches to the Goal - Reinforcement Learning Based on Rhythmic Walking Parameters. In: INTERNATIONAL ROBOCUP SYMPOSIUM, 7., Padua, Italy. **Proceedings...** Springer-Verlag, 2003. p.344–354. (Lecture Notes in Computer Science, v.3020).

ASHBY, W. R. **An Introduction to Cybernetics**. London, UK: Chapman & Hall, 1956. 308p.

ATTIAS, H. A Variational Bayesian Framework for Graphical Models. **Neural Information Processing Systems (NIPS)**, Cambridge, MA, USA, v.12, p.209–215, 2000.

BAKKER, B. et al. A Robot that Reinforcement-Learns to Identify and Memorize Important Previous Observations. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), Las Vegas, NV, USA. **Proceedings...** IEEE Press, 2003. v.1, p.430–435.

BAKKER, B.; LINÅKER, F.; SCHMIDHUBER, J. Reinforcement Learning in Partially Observable Mobile Robot Domains Using Unsupervised Event Extraction. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), EPFL, Lausanne, Switzerland. **Proceedings...** IEEE Press, 2002. v.1, p.938–943.

BARRON, A. R.; BARRON, R. L. Statistical Learning Networks: A unifying view. In: SYMPOSIUM ON THE INTERFACE: STATISTICS AND COMPUTING SCIENCE, Virginia, USA. **Proceedings...** Reston, 1988.

BARTO, A. G.; MAHADEVAN, S. Recent Advances in Hierarchical Reinforcement Learning. **Discrete Event Dynamic Systems**, Netherlands, v.13, n.4, p.341–379, 2003.

BASSO, E. W.; ENGEL, P. M. Reinforcement Learning in Non-Stationary Continuous Time and Space Scenarios. In: ARTIFICIAL INTELLIGENCE NATIONAL MEETING (ENIA), 7., Bento Gonçalves, RS, Brazil. **Proceedings...** SBC Editora, 2009.

BASTOS, E. N. F. **Uma Rede Neural Auto-Organizável Construtiva para Aprendizado Perpétuo de Padrões Espaço-Temporais**. 2007. Master's Thesis — Univeridade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil. Text in Portuguese.

BEKEY, G. A. **Autonomous Robots: from biological inspiration to implementation and control**. Cambridge, MA, USA: The MIT Press, 2005. 577p.

BERGER, J. O. **Statistical Decision Theory and Bayesian Analysis**. 2.ed. New York, NY, USA: Springer-Verlag, 1980. 627p.

BERTHOLD, M. R.; DIAMOND, J. Constructive Training of Probabilistic Neural Networks. **Neurocomputing**, New York, NY, USA, v.19, p.167–183, 1998.

BHATTACHARYYA, N. et al. Incremental PNN Classifier for a Versatile Electronic Nose. In: INTERNATIONAL CONFERENCE ON SENSING TECHNOLOGY, 3., Tainan, Taiwan. **Proceedings...** IEEE Press, 2008. p.242–247.

BISHOP, C. M. **Neural Networks for Pattern Recognition**. New York, USA: Oxford University Press, 1995. 482p.

BLEI, D. M.; JORDAN, M. I. Variational Inference for Dirichlet Process Mixtures. **Journal of Bayesian Analysis**, Columbus, OH, USA, v.1, n.1, p.121–144, 2005.

BOX, G. E. P.; JENKINS, G. M.; REINSEL, G. C. **Time Series Analysis: forecasting and control**. 3.ed. San Francisco, CA, USA: Holden Day, 1976. 575p.

BRENT, R. P. **Algorithms for Minimization without Derivatives**. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.

BURFOOT, D.; LUNGARELLA, M.; KUNIYOSHI, Y. Toward a Theory of Embodied Statistical Learning. In: INTERNATIONAL CONFERENCE ON SIMULATION OF ADAPTIVE BEHAVIOR (SAB 2008), 10., Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2008. p.270–279. (Lecture Notes in Artificial Intelligence, v.5040).

CACOULOS, T. Estimation of a Multivariate Density. **Annals of the Institute of Statistical Mathematics**, Tokyo, Japan, v.18, p.179–189, Sept. 1966.

CAMARGO, S. S. **Um Modelo Neural de Aprimoramento Progressivo para Redução de Dimensionalidade**. 2010. Ph.D. thesis — Univeridade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil. Text in Portuguese.

CAMARGO, S. S.; ENGEL, P. M. Time Series Prediction with Focused Time Lagged Feed-Forward Networks. In: INTERNATIONAL SYMPOSIUM ON FORECASTING (ISF), 25., San Antonio, TX, USA. **Proceedings...** International Institute of Forecasting, 2005. p.123.



CAPPÉ, O.; MOULINES, E. Online EM Algorithm for Latent Data Models. **Journal of the Royal Statistical Society**, London, UK, Sept. 2008.

CARPENTER, G. A. et al. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.3, n.5, p.698–713, Sept. 1992.

CARPENTER, G. A.; GROSSBERG, S. A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. **Computer Vision, Graphics and Image Processing**, New York, NY, USA, v.37, p.54–115, 1987.

CARPENTER, G. A.; GROSSBERG, S. Adaptive Resonance Theory. In: ARBIB, M. A. (Ed.). **The Handbook of Brain Theory and Neural Networks**. 2.ed. Cambridge, MA, USA: The MIT Press, 2003. p.87–90.

CARPENTER, G. A.; GROSSBERG, S.; REYNOLDS, J. ARTMAP: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. **Neural Networks**, Oxford, UK, v.4, p.565–588, 1991.

CARPENTER, G. A.; GROSSBERG, S.; ROSEN, D. B. **Fuzzy ART**: fast stable learning and categorization of analog patterns by an adaptive resonance system. Boston, MA, USA: Boston University, 1991. Neural Networks – Technical Report. (CAS/CNS-TR-91-015).

CHANG, R. K. Y.; LOO, C. K.; RAO, M. V. C. Enhanced Probabilistic Neural Network with Data Imputation Capabilities for Machine-Fault Classification. **Neural Computing and Applications**, London, UK, v.18, p.791–800, 2009.

CHAPUT, H. H. **The Constructivist Learning Architecture**: A model of cognitive development for robust autonomous robots. 2004. Ph.D. Thesis — University of Texas, Austin, TX, USA.

CHEN, S.; COWAN, C. F. N.; GRANT, P. M. Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.2, p.302–309, Mar. 1991.

CHOI, S. P. M.; YAN-YEUNG, D.; ZHANG, N. L. Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making. **Sequence Learning – Paradigms, Algorithms, and Applications**, London, UK, p.264–287, 2001.

COLLINS, M. **The EM Algorithm**. Burnaby, BC, Canada: School of Computing Science, Simon Fraser University (SFU), 1997. Technical Report. (CMPT-825).

COPPERSMITH DON; WINOGRAD, S. Matrix Multiplication Via Arithmetic Progressions. **SIAM News**, Philadelphia, PA, USA, v.9, n.3, p.251–280, 1990.

COX, D. R.; HINKLEY, D. V. **Theoretical statistics**. London, UK: Chapman and Hall, 1974. 511p.

CRICK, F. **The Astonishing Hypothesis**: the scientific search for the soul. London, UK: Simon & Schuster, 1994. 317p.

CSÁJI, B. C. **Approximation with Artificial Neural Networks**. 2001. M.Sc. Thesis — Eindhoven University of Technology, The Netherlands.

ĆWIK, J.; KORONACKI, J. Probability Density Estimation using a Gaussian Clustering Algorithm. **Neural Computing and Applications**, London, UK, v.4, p.149–160, 1996.

DAMASIO, A. R. **Descartes' Error: emotion, reason, and the human brain**. New York, NY, USA: HarperCollins, 1994. 316p.

DECLERCQ, A.; PIATER, J. H. Online Learning of Gaussian Mixture Models: A two-level approach. In: **THIRD INTERNATIONAL CONFERENCE ON COMPUTER VISION THEORY AND APPLICATIONS (VISAPP)**, Funchal, Madeira, Portugal. **Proceedings...** Springer-Verlag, 2008. v.1.

DELGOSHA, F.; MENHAJ, M. B. Fuzzy Probabilistic Neural Networks: A practical approach to the implementation of bayesian classifier. In: REUSCH, B. (Ed.). **Computational Intelligence: theory and applications**. Berlin, Germany: Springer Berlin / Heidelberg, 2001. p.76–85. (Lecture Notes in Computer Science, v.2206).

DEMPSTER, A. P.; LAIRD, N. M.; RUBIN, D. B. Maximum Likelihood from Incomplete Data Via the EM Algorithm. **Journal of the Royal Statistical Society**, London, UK, v.39, n.1, p.1–38, 1977.

DENNETT, D. C. **Kinds of Minds: towards an understanding of consciousness**. New York, NY, USA: Basic Books, 1996. 192p.

DOMINGOS, P.; HULTEN, G. H. A General Framework for Mining Massive Data Streams. **Journal of Computational and Graphical Statistics**, Alexandria, VA, USA, v.12, 2003.

DOYA, K. Temporal Difference Learning in Continuous Time and Space. **Advances in Neural Information Processing Systems**, Cambridge, MA, USA, v.8, p.1073–1079, 1996.

DOYA, K. Reinforcement Learning in Continuous Time and Space. **Neural Computation**, Cambridge, MA, USA, v.12, n.1, p.219–245, 2000.

DRESCHER, G. L. **Made-up Minds: A constructivist approach to artificial intelligence**. Cambridge, MA, USA: The MIT Press, 1991. 220p.

DUDEK, G.; JENKIN, M. **Computational Principles of Mobile Robotics**. Cambridge, UK: Cambridge University Press, 2000.

EDELMAN, G. M. Group Selection and Phasic Reentrant Signalling: A theory of higher brain function. In: EDELMAN, G. M.; MOUNTCASTLE, V. B. (Ed.). **The Mindful Brain: cortical organization and the group-selective theory of higher brain function**. Cambridge, MA, USA: The MIT Press, 1978. p.51–100.

EDELMAN, G. M. **Neural Darwinism: the theory of neuronal group selection**. New York, NY, USA: Basic Books, 1987. 371p.

ELMAN, J. L. Finding Structure in Time. **Cognitive Science**, Hoboken, NJ, USA, v.14, n.2, p.179–211, 1990.

ENGEL, P. M. Attentional Mode Neural Network: A new approach for real-time self-learning. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS(ISCAS'96). **Proceedings...** IEEE Press, 1996. p.45–48.

ENGEL, P. M. **INBC**: An incremental algorithm for dataflow segmentation based on a probabilistic approach. Porto Alegre, RS, Brazil: UFRGS, 2009. Technical Report. (RP-360).

ENGEL, P. M.; HEINEN, M. R. Concept Formation using Incremental Gaussian Mixture Models. In: IBEROAMERICAN CONGRESS ON PATTERN RECOGNITION (CIARP), 15., São Paulo, SP, Brazil. **Proceedings...** Springer-Verlag, 2010. p.128–135. (Lecture Notes in Computer Science, v.6419).

ENGEL, P. M.; HEINEN, M. R. Incremental Learning of Multivariate Gaussian Mixture Models. In: BRAZILIAN SYMPOSIUM ON AI (SBIA): ADVANCES IN ARTIFICIAL INTELLIGENCE, 20., São Bernardo do Campo, SP, Brazil. **Proceedings...** Springer-Verlag, 2010. p.82–91. (Lecture Notes in Artificial Intelligence, v.6404).

EVES, H. **Foundations and Fundamental Concepts of Mathematics**. 3.ed. Mineola, NY, USA: Dover Publications, Inc., 1990.

FAHLMAN, S. E.; LEBIERE, C. **The Cascade-Correlation Learning Architecture**. Pittsburgh, PA, USA: Carnegie-Mellon University (CMU), 1990. Technical Report. (CMU-CS-90-100).

FERGUSON, T. S. A Bayesian Analysis of Some Nonparametric Problems. **The Annals of Statistics**, Philadelphia, PA, USA, 1973.

FIGUEIREDO, M. A. T.; JAIN, A. K. Unsupervised Learning of Finite Mixture Models. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Los Alamitos, CA, USA, v.24, n.3, p.381–396, Mar. 2002.

FILLIAT, D.; MEYER, J.-A. Map-Based Navigation in Mobile Robots: I. A review of localization strategies. **Cognitive Systems Research**, New York, NY, USA, v.4, n.4, p.243–282, 2003.

FISHER, D. H. Knowledge Acquisition via Incremental Conceptual Learning. **Machine Learning**, Netherlands, v.2, p.139–172, 1987.

FREEMAN, J. A.; SKAPURA, D. M. **Neural Networks, Algorithms, Applications, and Programming Techniques**. Boston, MA, USA: Addison-Wesley, 1991.

FRENCH, R. M. Catastrophic Forgetting in Connectionist Networks. **Encyclopedia of Cognitive Science**, London, UK, v.1, p.431–435, 2003.

FUKUNAGA, K. **Introduction to Statistical Pattern Recognition**. 2.ed. New York, NY, USA: Academic Press, 1990.

GASÓS, J.; MARTÍN, A. Mobile Robot Localization Using Fuzzy Maps. In: WORKSHOP ON FUZZY LOGIC IN ARTIFICIAL INTELLIGENCE: TOWARDS INTELLIGENT SYSTEMS, London, UK. **Proceedings...** Springer-Verlag, 1997. p.207–224. (Lecture Notes in Artificial Intelligence, v.1188).

GASÓS, J.; ROSSETI, A. Uncertainty Representation for Mobile Robots: perception, modeling and navigation in unknown environments. **Fuzzy Sets and Systems**, New York, NY, USA, v.107, n.1, p.1–24, Oct. 1999.

GATH, I.; GEVA, A. B. Unsupervised Optimal Fuzzy Clustering. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Los Alamitos, CA, USA, v.11, n.7, p.773–781, July 1989.

GENNARI, J. H.; LANGLEY, P.; FISHER, D. H. Models of Incremental Concept Formation. **Artificial Intelligence**, New York, NY, USA, v.40, n.1-3, p.11–61, 1989.

GHAHRAMANI, Z.; JORDAN, M. I. **Learning from Incomplete Data**. Cambridge, MA, USA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT) and Center for Biological and Computational Learning, Department of Brain and Cognitive Sciences, 1994. Technical Report. (A.I. Memo No. 1509, C.B.C.L. Paper No. 108).

GHAHRAMANI, Z.; JORDAN, M. I. Supervised Learning from Incomplete Data Via an EM Approach. **Advances in Neural Information Processing Systems**, San Francisco, CA, USA, v.6, 1994.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Reading, MA, USA: Addison-Wesley, 1989.

GOMES, R.; WELLING, M.; PERONA, P. Incremental Learning of Nonparametric Bayesian Mixture Models. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR), Anchorage, AK, USA. **Proceedings...** IEEE Press, 2008.

GRANGER, E.; CONNOLLY, J.-F.; SABOURIN, R. A Comparison of Fuzzy ARTMAP and Gaussian ARTMAP Neural Networks for Incremental Learning. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN). **Proceedings...** IEEE Press, 2008. p.3305–3312.

GROSSBERG, S. Adaptive Pattern Classification and Universal Recoding, I: Parallel development and coding of neural feature detectors. **Biological Cybernetics**, Berlin, Germany, v.23, p.121–134, 1976.

GROSSBERG, S. Adaptive Pattern Classification and Universal Recoding, II: Feedback, expectation, olfaction, and illusions. **Biological Cybernetics**, Berlin, Germany, v.23, p.187–202, 1976.

GROSSBERG, S. **Studies of Mind and Brain**: neural principles of learning, perception, development, cognition and motor control. Boston, MA, USA: Reidel Press, 1982. 660p.

GROSSBERG, S. **Adaptive Resonance Theory**. Boston, MA, USA: Boston University, 2000. Technical Report. (CAS/CNS-2000-024).

GROSSBERG, S. **How Does the Cerebral Cortex Work? Development, Learning, Attention, and 3D Vision by Laminar Circuits of Visual Cortex**. Boston, MA, USA: Boston University, 2003. Technical Report – Invited Article for Behavioral and Cognitive Neuroscience Reviews. (CAS/CNS TR-2003-005).

HAGAN, M. T.; MENHAJ, M. B. Training Feed Forward Network With the Marquardt Algorithm. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.6, n.5, p.989–993, 1994.

HAMKER, F. H. Life-Long Learning Cell Structures – Continuously Learning Without Catastrophic Interference. **Neural Networks**, Oxford, UK, v.14, p.551–573, 2001.

HAWKINS, J. **On Intelligence**. New York, NY, USA: Owl Books, 2005.

HAWKINS, J.; DILEEP, G. **Hierarchical Temporal Memory: concepts, theory, and terminology**. Redwood City, CA, USA: Numenta Inc., 2006. Whitepaper.

HAYKIN, S. **Neural Networks and Learning Machines**. 3.ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2008.

HEINEN, M. R. **Controle Inteligente do Caminhar de Robôs Móveis Simulados**. 2007. 121p. Master's Thesis - Applied Computing — Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS, Brazil. Date of Conclusion: 20/02/2007, Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. Aprendizado Autônomo de Robôs Móveis Simulados em Ambientes Contínuos. In: XXXV LATIN AMERICAN INFORMATICS CONFERENCE (CLEI), Pelotas, RS, Brazil. **Proceedings...** Springer-Verlag, 2009. p.10. Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. Aprendizado e Controle de Robôs Móveis Autônomos Utilizando Atenção Visual. In: I SIMPÓSIO DE COMPUTAÇÃO APLICADA (SCA 2009), Passo Fundo, RS, Brazil. **Anais...** UPF Editora, 2009. Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. Aprendizado de Robôs Móveis Autônomos em Ambientes Simulados Contínuos. In: IX CONGRESSO BRASILEIRO DE REDES NEURAIAS / INTELIGÊNCIA COMPUTACIONAL (CBRN), Ouro Preto, MG, Brazil. **Anais...** Sociedade Brasileira de Redes Neurais (SBRN), 2009. p.5. Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. NLOOK: A computational attention model for robot vision. **Journal of the Brazilian Computer Society (JBACS)**, Porto Alegre, RS, Brazil, p.3–17, Sept. 2009.

HEINEN, M. R.; ENGEL, P. M. Evaluation of Visual Attention Models under 2D Similarity Transformations. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC 2009) – SPECIAL TRACK ON INTELLIGENT ROBOTIC SYSTEMS, 24., Honolulu, Hawaii, USA. **Proceedings...** ACM press, 2009. p.1156–1160.

HEINEN, M. R.; ENGEL, P. M. An Incremental Probabilistic Neural Network for Regression and Reinforcement Learning Tasks. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETWORKS (ICANN 2010), 20., Thessaloniki, Greece. **Proceedings...** Springer-Verlag, 2010. p.170–179. (Lecture Notes in Computer Science, v.6353).

HEINEN, M. R.; ENGEL, P. M. IPNN: An incremental probabilistic neural network for function approximation and regression tasks. In: INTERNATIONAL JOINT CONFERENCE 2010, 11TH BRAZILIAN NEURAL NETWORKS SYMPOSIUM (SBRN), São Bernardo do Campo, SP, Brazil. **Proceedings...** IEEE Press, 2010. p.39–44.

HEINEN, M. R.; ENGEL, P. M. Feature-Based Mapping using Incremental Gaussian Mixture Models. In: VII LATIN AMERICAN ROBOTICS SYMPOSIUM AND VI INTELLIGENT ROBOTIC MEETING (LARS 2010), São Bernardo do Campo, SP, Brazil. **Proceedings...** IEEE Press, 2010. p.67–72.

HEINEN, M. R.; ENGEL, P. M. IGMN: An incremental neural network model for on-line tasks. In: V WORKSHOP ON MSC DISSERTATION AND PHD THESIS IN ARTIFICIAL INTELLIGENCE (WTDIA 2010), São Bernardo do Campo, SP, Brazil. **Proceedings...** SBC Editora, 2010. p.732–741.

HEINEN, M. R.; ENGEL, P. M. Aprendizado incremental de conceitos hierárquicos utilizando modelos de mistura gaussianos. In: XVIII CONGRESSO BRASILEIRO DE AUTOMÁTICA (CBA 2010), Bonito, MS, Brazil. **Anais...** Sociedade Brasileira de Autômática, 2010. Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. Incremental Probabilistic Neural Networks for Concept Formation, Robotic Localization and Mapping. In: DYNAMICS DAYS SOUTH AMERICA 2010 – INTERNATIONAL CONFERENCE ON CHAOS AND NONLINEAR DYNAMICS, São José dos Campos, SP, Brazil. **Proceedings...** Institute of Physics (IOP) Publishing, 2010.

HEINEN, M. R.; ENGEL, P. M. Aprendizado e Controle de Robôs Móveis Autônomos Utilizando Atenção Visual. **Revista de Informática Teórica e Aplicada (RITA)**, Porto Alegre, RS, Brazil, v.17, n.3, p.349–363, 2010. Text in Portuguese.

HEINEN, M. R.; ENGEL, P. M. Incremental Feature-Based Mapping from Sonar Data using Gaussian Mixture Models. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC 2011) – SPECIAL TRACK ON INTELLIGENT ROBOTIC SYSTEMS, 26., TaiChung, Taiwan. **Proceedings...** ACM press, 2011. p.1370–1375.

HEINEN, M. R.; OSÓRIO, F. S. Neural Networks Applied to Gait Control of Physically Based Simulated Robots. In: INTERNATIONAL JOINT CONFERENCE 2006, 9TH BRAZILIAN NEURAL NETWORKS SYMPOSIUM (SBRN), Ribeirão Preto, SP, Brazil. **Proceedings...** IEEE Press, 2006.

HEINEN, M. R.; OSÓRIO, F. S. Applying Genetic Algorithms to Control Gait of Physically Based Simulated Robots. In: IEEE CONGRESS ON EVOLUTIONARY COMPUTATION (CEC), Vancouver, Canada. **Proceedings...** IEEE Press, 2006.

HEINEN, M. R.; OSÓRIO, F. S. Gait Control Generation for Physically Based Simulated Robots using Genetic Algorithms. In: INTERNATIONAL JOINT CONFERENCE 2006, 10TH IBERO-AMERICAN CONFERENCE ON AI (IBERAMIA), 18TH BRAZILIAN SYMPOSIUM ON AI (SBIA), Ribeirão Preto - SP, Brazil. **Proceedings...** Springer-Verlag, 2006. (Lecture Notes in Computer Science).

HEINEN, M. R.; OSÓRIO, F. S. Evolving Gait Control of Physically Based Simulated Robots. **Journal of Theoretical and Applied Informatics (RITA)**, Porto Alegre, RS, Brazil, v.14, n.1, p.119–134, 2007.

HEINEN, M. R.; OSÓRIO, F. S. Applying Genetic Algorithms to Control Gait of Simulated Robots. In: IEEE ELECTRONICS, ROBOTICS AND AUTOMOTIVE MECHANICS CONFERENCE (CERMA) 2007, Cuernavaca, Morelos, Mexico. **Proceedings...** IEEE Press, 2007. p.500–505.

HEINEN, M. R.; OSÓRIO, F. S. Applying Neural Networks to Control Gait of Simulated Robots. In: INTERNATIONAL JOINT CONFERENCE 2008, 10TH BRAZILIAN NEURAL NETWORKS SYMPOSIUM (SBRN), Salvador, BH, Brazil. **Proceedings...** IEEE Press, 2008.

HEINEN, M. R.; OSÓRIO, F. S. Evolving Morphologies and Gaits of Physically Realistic Simulated Robots. In: ACM SYMPOSIUM ON APPLIED COMPUTING (SAC 2009) – SPECIAL TRACK ON INTELLIGENT ROBOTIC SYSTEMS, 24., Honolulu, Hawaii, USA. **Proceedings...** ACM press, 2009. p.1161–1165.

HOLLAND, O.; GOODMAN, R. Robots with Internal Models: A route to machine consciousness? **Journal of Consciousness Studies**, Exeter, UK, v.10, p.77–109, 2003.

HORNIK, K. Approximation Capabilities of Multilayer Feedforward Networks. **Neural Networks**, New York, NY, USA, v.4, 1991.

IP, Y. L. et al. Segment-Based Map Building Using Enhanced Adaptive Fuzzy Clustering Algorithm for Mobile Robot Applications. **Journal of Intelligent and Robotic Systems**, The Netherlands, v.35, n.3, p.221–245, 2002.

JAIN, A. K.; DUIN, R. P.; MAO, J. Statistical Pattern Recognition: A review. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Los Alamitos, CA, USA, v.22, n.1, p.4–37, Jan. 2000.

JEPSON, A. D.; FLEET, D. J.; EL-MARAGHI, T. F. Robust Online Appearance Models for Visual Tracking. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Los Alamitos, CA, USA, v.25, n.10, p.1296–1311, Oct. 2003.

JORDAN, M. I. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine. In: EIGHTH ANNUAL CONFERENCE OF THE COGNITIVE SCIENCE SOCIETY, Englewood Cliffs, NJ, USA. **Proceedings...** Erlbaum Associates, 1986. p.531–546.

KEEHN, D. G. A Note on Learning for Gaussian Proprieties. **IEEE Transactions on Information Theory**, Los Alamitos, CA, USA, v.11, p.126–132, 1965.

KOHONEN, T. The Self-Organizing Map. **Proceedings of the IEEE**, Los Alamitos, CA, USA, v.78, n.9, p.1464–1480, Sept. 1990.

KOHONEN, T. **Self-Organizing Maps**. 3.ed. Berlin, Germany: Springer-Verlag, 2001.

KRICHMAR, J. L.; EDELMAN, G. M. Machine Psychology: Autonomous behavior, perceptual categorization and conditioning in a brain-based device. **Cerebral Cortex**, New York, USA, v.12, n.8, p.818–830, Aug. 2002.

KRICHMAR, J. L.; EDELMAN, G. M. Brain-Based Devices: intelligent systems based on principles of the nervous system. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), Las Vegas, NV, USA. **Proceedings...** IEEE Press, 2003. v.1, p.940–945.

KRICHMAR, J. L.; EDELMAN, G. M. Brain-Based Devices for the Study of Nervous Systems and the Development of Intelligent Machines. **Artificial Life**, Cambridge, MA, USA, v.11, n.1-2, p.63–77, 2005.

KRISTAN, M.; SKOCAJ, D.; LEONARDIS, A. Incremental Learning with Gaussian Mixture Models. In: COMPUTER VISION WINTER WORKSHOP 2008, Moravske Toplice, Slovenia. **Proceedings...** Slovenian Pattern Recognition Society, 2008. p.25–32.

LATECKI, L. J. et al. Building Polygonal Maps from Laser Range Data. In: INTERNATIONAL COGNITIVE ROBOTICS WORKSHOP (CogRob), 4., Valencia, Spain. **Proceedings...** European Coordinating Committee for Artificial Intelligence, 2004.

LEE, S. J. et al. Feature Based Map Building Using Sparse Sonar Data. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), Alberta, Canada. **Proceedings...** IEEE Press, 2005. p.492–496.

LIN, S. H.; KUNG, S. Y.; LIN, L. J. Face Recognition/Detection by Probabilistic Decision-Based Neural Network. **IEEE Transactions on Neural Networks: Special Issue on Biometric Identification**, Los Alamitos, CA, USA, v.8, n.1, p.114–132, 1997.

LINÅKER, F. **Unsupervised On-line Data Reduction for Memorisation and Learning in Mobile Robotics**. 2003. 112p. Ph.D. Thesis — Computer Science, University of Sheffield, Sheffield, UK.

LINÅKER, F.; JACOBSSON, H. Mobile Robot Learning of Delayed Response Tasks through Event Extraction: A solution to the road sign problem and beyond. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI'01), 17., San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc., 2001. p.777–782.

LINÅKER, F.; NIKLASSON, L. Time Series Segmentation Using an Adaptive Resource Allocating Vector Quantization Network Based on Change Detection. In: IEEE-INNS-ENNS INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN 2000), Los Alamitos, CA. **Proceedings...** IEEE Press, 2000. p.323–328.

LINÅKER, F.; NIKLASSON, L. Sensory Flow Segmentation Using a Resource Allocating Vector Quantizer. In: JOINT IAPR INTERNATIONAL WORKSHOPS ON ADVANCES IN PATTERN RECOGNITION, London, UK. **Proceedings...** Springer-Verlag, 2000. p.853–862.

LORENZO, J. M. P. et al. A Hough-based Method for Concurrent Mapping and Localization in Indoor Environments. In: IEEE CONFERENCE ON ROBOTICS, AUTOMATION AND MECHATRONICS (RAM), Singapore. **Proceedings...** IEEE Press, 2004. v.2, p.840–845.

LU, F.; MILIOS, E. E. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. **Journal of Intelligent and Robotic Systems**, The Netherlands, v.18, n.3, p.249–275, 1998.

LUO, R. C.; LI, J.-X.; CHEN, C.-T. Indoor Localization Using Line Based Map for Autonomous Mobile Robot. In: IEEE WORKSHOP ON ADVANCED ROBOTICS AND ITS SOCIAL IMPACTS, Taipei, Taiwan. **Proceedings...** IEEE Press, 2008. p.1–6.



- MACQUEEN, J. B. Some Methods for Classification and Analysis of Multivariate Observations. In: BERKELEY SYMPOSIUM ON MATHEMATICAL STATISTICS AND PROBABILITY, 5., Berkeley, CA, USA. **Proceedings...** University of California Press, 1967. v.1, p.281–297.
- MAO, K. Z.; TAN, K.-C.; SER, W. Probabilistic Neural-Network Structure Determination for Pattern Classification. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.11, n.4, p.1009–1016, July 2000.
- MASSART, D. L. et al. Practical Data Handling: visual presentation of data by means of box plots. **LC–GC Europe**, Santa Monica, CA, USA, v.18, n.4, p.215–218, Apr. 2005.
- MCCULLOCH, W. S.; PITTS, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. **Bulletin of Mathematical Biophysics**, Boulder, CO, USA, v.5, p.115–133, 1943.
- MCLACHLAN, G.; BASFORD, K. **Mixture Models: inference and application to clustering**. New York, NY, USA: Marcel Dekker, 1988.
- MCLACHLAN, G.; KRISHNAN, T. **The EM Algorithm and Extensions**. New York, NY, USA: John Wiley & Sons, 1997.
- MCLACHLAN, G.; PEEL, D. **Finite Mixture Models**. New York, NY, USA: John Wiley & Sons, 2000.
- MENEGAZ, M.; ENGEL, P. M. Using The GTSOM Network for Mobile Robot Navigation With Reinforcement Learning. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), Atlanta, GA, USA. **Proceedings...** IEEE Press, 2009.
- MEYER-DELIUS, D.; BURGARD, W. Maximum-likelihood sample-based maps for mobile robots. **Robotics and Autonomous Systems**, New York, NY, USA, v.58, p.133–139, 2010.
- MEYER, J.-A.; FILLIAT, D. Map-Based Navigation in Mobile Robots: II. A review of map-learning and path-planning strategies. **Cognitive Systems Research**, New York, NY, USA, v.4, n.4, p.283–317, 2003.
- MICHEL, O. **Khepera Simulator version 2.0 - User Manual**. 1996.
- MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge, MA, USA: The MIT Press, 1996.
- MITCHELL, T. **Machine Learning**. New York, USA: McGrall-Hill, 1997.
- MOERMAN, W. **Hierarchical Reinforcement Learning: Assignment of behaviours to subpolicies by self-organization**. 2009. 122p. Dissertação (Mestrado em Ciência da Computação) — Cognitive Artificial Intelligence, Utrecht University.
- MONTANA, D. A Weighted Probabilistic Neural Network. **Advances in Neural Information Processing Systems**, San Francisco, CA, USA, v.4, p.1110–1117, 1992.
- MOODY, J.; DARKEN, C. Fast Learning in Networks of Locally-Tuned Processing Units. **Neural Computation**, Cambridge, MA, USA, v.1, p.281–294, 1989.

MOUNTCASTLE, V. B. An Organizing Principle for Cerebral Function: the unit model and the distributed system. In: EDELMAN, G. M.; MOUNTCASTLE, V. B. (Ed.). **The Mindful Brain: cortical organization and the group-selective theory of higher brain function**. Cambridge, MA, USA: The MIT Press, 1978. p.7–50.

NADARAYA, E. A. On Estimating Regression. **Theory of Probability and its Applications**, Philadelphia, PA, USA, v.9, n.1, p.141–142, Jan. 1964.

NARENDRA, K. S.; PARTHASARATHY, K. Identification and Control of Dynamical Systems using Neural Networks. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.1, p.4–27, Mar. 1990.

NEAL, R. M.; HINTON, G. E. A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. **Learning in Graphical Models**, Norwell, MA, USA, p.355–368, 1998.

NOLFI, S.; FLOREANO, D. **Evolutionary Robotics: the biology, intelligence, and technology of self-organizing machines**. Cambridge, MA, USA: The MIT Press, 2000. 320p.

NOLFI, S.; PARISI, D. Exploiting the Power of Sensory-Motor Coordination. In: EUROPEAN CONFERENCE ON ADVANCES IN ARTIFICIAL LIFE, 5., London, UK. **Proceedings...** Springer-Verlag, 1999. p.173–182. (Advances in Artificial Life).

NOLFI, S.; TANI, J. Extracting Regularities in Space and Time Through a Cascade of Prediction Networks: the case of a mobile robot navigating in a structured environment. **Connection Science**, London, UK, v.11, n.2, p.125–148, 1999.

OMOHUNDRO, S. Efficient Algorithms with Neural Network Behaviour. **Complex Systems**, Champaign, IL, USA, v.1, p.273, 1987.

OSÓRIO, F. S. et al. Increasing Reality in Virtual Reality Applications through Physical and Behavioural Simulation. In: FISCHER, X. (Ed.). **Research in Interactive Design – Proceedings of the Virtual Concept Conference 2006**. Berlin, Germany: Springer-Verlag, 2006. v.2, p.1–45.

PARZEN, E. On the Estimation of a Probability Density. **The Annals of Mathematical Statistics**, Beachwood, OH, USA, v.3, p.1065–1076, 1962.

PEREIRA, B. d. B.; RAO, C. R. **Data Mining with Neural Networks: A guide for statisticians**. <http://textbookrevolution.org/index.php/Book:Lists/Subjects/Mathematics>: Textbook Revolution, 2009.

PEROTTO, F. S. **CALM - Constructivist Agent Learning Mechanism**. 2010. Ph.D. Thesis — Univeridade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil.

PEROTTO, F. S.; ÁLVARES, L. O. Learning Regularities with a Constructivist Agent. In: INTERNATIONAL JOINT CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS (AAMAS'06), 5., Hakodate, Hokkaido, Japan. **Proceedings...** ACM Press, 2006. p.807–809.

PEROTTO, F. S.; ÁLVARES, L. O. Constructivist Anticipatory Learning Mechanism (CALM) - Dealing with Partially Deterministic and Partially Observable Environments.

In: INTERNATIONAL CONFERENCE ON EPIGENETIC ROBOTICS: MODELING COGNITIVE DEVELOPMENT IN ROBOTIC SYSTEMS, 7. **Proceedings...** Simulation of Adaptive Behavior, 2007. p.110–120. (Lund University Cognitive Studies, v.135).

PFEIFER, R.; IIDA, F.; BONGARD, J. New Robotics: design principles for intelligent systems. **Artificial Life**, Cambridge, MA, USA, v.11, n.1-2, p.99–120, Jan. 2005.

PFEIFER, R.; LUNGARELLA, M.; IIDA, F. Self-Organization, Embodiment, and Biologically Inspired Robotics. **Science**, Washington, DC, USA, v.318, p.1088–1093, Nov. 2007.

PFEIFER, R.; SCHEIER, C. From Perception to Action: the right direction? In: FROM PERCEPTION TO ACTION CONFERENCE, Los Alamitos, CA, USA. **Proceedings...** IEEE Press, 1994. p.1–11.

PFEIFER, R.; SCHEIER, C. Sensory-motor coordination: the metaphor and beyond. **Robotics and Autonomous Systems**, New York, NY, USA, v.20, n.2-4, p.157–178, June 1997.

PFEIFER, R.; SCHEIER, C. **Understanding Intelligence**. Cambridge, MA, USA: The MIT Press, 1999. 697p.

PIAGET, J. **The Origins of Intelligence in Children**. Madison, CT, USA: International Universities Press, 1952.

PIAGET, J. **The construction of Reality in the Child**. New York, NY, USA: Basic Books, 1954.

PINKER, S. **The Language Instinct**: how the mind works. New York, NY, USA: W. W. Norton, 1997. 662p.

PINTO, R. C. **A Neocortex Inspired Hierarchical Spatio-Temporal Pattern Recognition System**. 2009. Final Undergraduate Dissertation — Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, RS, Brazil.

PLATT, J. A Resource-Allocating Network for Function Interpolation. **Neural Computation**, Cambridge, MA, USA, v.3, n.2, p.213–225, 1991.

POGGIO, T.; GIROSI, F. **A Theory of Networks for Approximation and Learning**. Cambridge, MA, USA: Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT) and Center for Biological Information, Whitaker College, 1989. A.I. Memo No. 1140 – C.B.I.P. Paper No. 31.

POLAT, O.; YILDIRIM, T. FPGA Implementation of a General Regression Neural Network: An embedded pattern classification system. **Digital Signal Processing**, New York, NY, USA, v.20, p.881–886, 2010.

POWELL, M. J. D. An efficient method for Finding the Minimum for a function of several variables without calculating derivatives. **The Computer Journal**, New York, USA, v.7, p.155–162, 1964.

POWELL, M. J. D. Radial Basis Function for Multi-Variable Interpolation: A review. In: IMA CONFERENCE ON ALGORITHMS FOR THE APPROXIMATION OF FUNCTIONS AND DATA, RMCS, Shrivenham, UK. **Proceedings...** Institute of Mathematics and its Applications (IMA), 1985. p.143–167.

POWELL, M. J. D. **Radial Basis Function for Multivariate Interpolation, a Review**. Cambridge, MA, USA: Clarendon Press, 1987.

POWELL, M. J. D. Radial Basis Function Approximation to Polynomials. In: DUNDEE BIENNIAL NUMERICAL ANALYSIS CONFERENCE. **Proceedings...** Dundee University Press, 1987. p.223–241.

PRESS, W. H. et al. **Numerical Recipes in C: the art of scientific computing**. Cambridge, MA, USA: Cambridge University Press, 1992.

PUENTE, P. de la et al. 3D Feature Based Mapping Towards Mobile Robots' Enhanced Performance in Rescue Missions. In: IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS), St. Louis, MO, USA. **Proceedings...** IEEE Press, 2009. p.1138–1143.

QUINLAN, J. R. **C4.5- Programs for Machine Learning**. San Mateo, CA, USA: Morgan Kaufman Publishers, 1993.

RAMACHANDRAN, V. S. **The Emerging Mind**. Cambridge, MA, USA: BBC in association with Profile Books, 2003. (Reith Lectures).

RIEDMILLER, M.; BRAUN, H. A Direct Adaptive Method for Faster Backpropagation Learning: the RPROP algorithm. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS (ICNN), San Francisco, CA, USA. **Proceedings...** IEEE Press, 1993. p.586–591.

RISSANEN, J. **Stochastic Complexity in Statistical Inquiry**. Singapore: World Scientific, 1989.

ROBBINS, H.; MONRO, S. A Stochastic Approximation Method. **Annals of Mathematical Statistics**, Beachwood, OH, USA, v.22, p.400–407, 1951.

ROBINSON, S. Toward an Optimal Algorithm for Matrix Multiplication. **SIAM News**, Philadelphia, PA, USA, v.38, n.9, p.127–132, 2005.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning Internal Representations by Error Propagation**. Cambridge, MA, USA: The MIT Press, 1986.

RUTKOWSKI, L. Adaptive Probabilistic Neural Networks for Pattern Classification in Time-Varying Environment. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.15, n.4, p.811–827, July 2004.

RUTKOWSKI, L. Generalized Regression Neural Networks in Time-Varying Environment. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.15, n.3, p.576–596, May 2004.

SARLE, W. S. **Why Statisticians Should Not FART**. Cary, NC, USA: SAS Institute, 1995. Technical Report.

SATO, M.-A.; ISHII, S. On-line EM Algorithm for the Normalized Gaussian Network. **Neural Computation**, Cambridge, MA, USA, v.12, n.2, p.407–432, 2000.

SCIAVICCO, L.; SICILIANO, B. **Modeling and Control of Robot Manipulator**. New York, USA: McGraw-Hill, 1996.

SETHURAMAN, J. A Constructive Definition of Dirichlet Priors. **Statistica Sinica**, Taipei, Taiwan, v.4, p.639–650, 1994.

SHAHSVAND, A. An Optimal Radial Basis Function (RBF) Neural Network for Hyper-Surface Reconstruction. **Transactions C: Chemistry and Chemical Engineering**, Tehran, Iran, v.16, n.1, p.41–53, June 2009.

SILVA, B. C. d. et al. Dealing with Non-Stationary Environments using Context Detection. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING - (ICML 2006), 23., Pittsburgh, CA, USA. **Proceedings...** International Machine Learning Society (IMLS), 2006. p.217–224.

SMART, W. D. **Making Reinforcement Learning Work on Real Robots**. 2002. Ph.D. Thesis — Brown University, Providence, Rhode Island, USA.

SMART, W. D.; KAEHLING, L. P. Practical Reinforcement Learning in Continuous Spaces. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING (ICML'2000), 17., San Francisco, CA, USA. **Proceedings...** Morgan Kaufmann Publishers Inc., 2000. p.903–910.

SPECHT, D. F. Probabilistic Neural Networks for Classification, Mapping, or Associative Memory. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS, San Diego, CA, USA. **Proceedings...** IEEE Press, 1988. v.1, p.525–532.

SPECHT, D. F. Probabilistic Neural Networks. **Neural Networks**, New York, NY, USA, v.3, p.109–118, 1990.

SPECHT, D. F. A General Regression Neural Network. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.2, n.6, p.568–576, Nov. 1991.

SPECHT, D. F. Enhancements to the Probabilistic Neural Networks. In: IEEE INTERNATIONAL JOINT CONFERENCE ON NEURAL NETWORKS (IJCNN), Baltimore, MD, USA. **Proceedings...** IEEE Press, 1992. v.1, p.761–768.

STAUFFER, C.; GRIMSON, W. E. L. Adaptive Background Mixture Models for Real-time Tracking. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'99), Ft. Collins, CO, USA. **Proceedings...** IEEE Press, 1999. v.2, p.246–252.

STEELS, L. The Autotelic Principle. In: IIDA, F. et al. (Ed.). **Embodied Artificial Intelligence**. Berlin, Germany: Springer Berlin / Heidelberg, 2004. p.629–629. (Lecture Notes in Computer Science, v.3139).

STENING, J.; JACOBSSON, H.; ZIEMKE, T. Imagination and Abstraction of Sensorimotor Flow: towards a robot model. In: AISB SYMPOSIUM ON NEXT GENERATION APPROACHES TO MACHINE, Hatfield, UK. **Proceedings...** Society for the Study of Artificial Intelligence and the Simulation of Behavior (AISB), 2005. p.50–58.

STRASSEN, V. Gaussian Elimination is not Optimal. **Numerische Mathematik**, Berlin, Germany, v.13, n.3, p.354–356, 1969.

STREIT, R. L.; LUGINBUHL, T. E. Maximum Likelihood Training of Probabilistic Neural Networks. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.5, n.5, p.764–783, 1994.

SUTTON, R. S. Learning to Predict by the Methods of Temporal Differences. **Machine Learning**, Hingham, MA, USA, v.3, p.9–44, 1988.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An introduction**. Cambridge, MA, USA: The MIT Press, 1998.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to Data Mining**. Boston, MA, USA: Addison-Wesley, 2006. 769p.

THRUN, S. Robotic Mapping: A survey. **Exploring Artificial Intelligence in the New Millenium**, San Francisco, CA, USA, 2002.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. Cambridge, MA, USA: The MIT Press, 2006. 647p. (Intelligent Robotics and Autonomous Agents).

TITTERINGTON, D. M. Recursive Parameter Estimation Using Incomplete Data. **Journal of the Royal Statistical Society**, London, UK, v.46, n.2, p.257–267, 1984.

TITTERINGTON, D. M.; SMITH, A. F. M.; MAKOV, U. E. **Statistical Analysis of Finite Mixture Distributions**. New York, NY, USA: John Wiley, 1985.

TOU, J. T.; GONZALEZ, R. C. **Pattern Recognition Principles**. Reading, MA, USA: Addison-Wesley, 1974.

TRÁVÉN, H. G. H. A Neural Network Approach to Statistical Pattern Classification by “Semiparametric” Estimation of Probability Density Functions. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.2, n.3, p.366–377, May 1991.

TSUJI, T. et al. A Recurrent Log-Linearized Gaussian Mixture Network. **IEEE Transactions on Neural Networks**, Los Alamitos, CA, USA, v.2, n.14, p.304–316, Mar. 2003.

UTSUNOMIYA, H.; SHIBATA, K. Learning with a Recurrent Neural Network in a Continuous State and Action Space Task. In: INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING OF THE ASIA-PACIFIC NEURAL NETWORK ASSEMBLY (ICONIP’08): ADVANCES IN NEURO-INFORMATION PROCESSING, PART II, 15., Auckland, New Zealand. **Proceedings...** Springer-Verlag Berlin Heidelberg, 2009. p.970–978. (Lecture Notes in Computer Science, v.5507).

VALPOLA, H. **Bayesian Ensemble Learning for Nonlinear Factor Analysis**. 2000. 54p. Ph.D. Thesis — Acta Polytechnica Scandinavica, Espoo, Finland.

WANG, S.; ZHAO, Y. Almost Sure Convergence of Titterington’s Recursive Estimator for Mixture Models. **Statistics & Probability Letters**, New York, NY, USA, v.76, p.2001–2006, 2006.

WATSON, G. S. Smooth Regression Analysis. **Sankhyā: The Indian Journal of Statistics**, Kolkata, India, v.26, p.359–372, 1964.

WIDROW, B.; HOFF, M. E. Adaptive Switching Circuits. **1960 IRE WESCON Convention Record**, Cambridge, MA, USA, p.96–104, 1960. Reprinted in *Neurocomputing* (MIT Press, 1988).

WILLIAMSON, J. R. A Constructive, Incremental-Learning Neural Network for Mixture Modeling and Classification. **Neural Computation**, Cambridge, MA, USA, v.9, n.7, p.1517–1543, 1997.

WOLFE, J. H. Pattern Clustering by Multivariate Mixture Analysis. **Multivariate Behavioral Research**, Mahwah, NJ, USA, v.5, p.329–350, 1970.

XU, L.; JORDAN, M. On Convergence Properties of the EM Algorithm for Gaussian Mixtures. **Neural Computation**, Cambridge, MA, USA, v.8, p.129–151, 1996.

ZADEH, L. A. Fuzzy Sets. **Information and Control**, New York, NY, USA, v.8, n.3, p.338–353, 1965.

ZHANG, L.; GHOSH, B. K. Line Segment Based Map Building and Localization Using 2D Laser Rangefinder. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA), San Francisco, CA, USA. **Proceedings...** IEEE Press, 2000. p.2538–2543.





## APPENDIX – PRINCIPAIS CONTRIBUIÇÕES DA TESE

Esta monografia apresenta o IGMN, uma nova abordagem conexionista para aproximação incremental de funções e previsão em tempo real. O IGMN é inspirado em teorias recentes do cérebro, especialmente o MPF (do inglês *Memory-Prediction Framework*) (HAWKINS, 2005) e a Inteligência Artificial Construtivista (DRESCHER, 1991), e estas teorias conferem ao IGMN algumas características especiais que não estão presentes na maioria dos modelos de redes neurais existentes. Além disso, o IGMN é baseado em fortes princípios estatísticos (modelos de mistura gaussianos) e assintoticamente converge para a superfície de regressão ótima à medida que os dados de treinamento chegam. Através de diversos experimentos, realizados utilizando dados reais e sintéticos, é demonstrado nesta tese que o IGMN pode ser utilizado com sucesso em diversas aplicações potenciais que requerem aprendizado incremental e/ou desempenho em tempo real. Além disso, esses experimentos demonstraram que o IGMN possui diversas vantagens em relação à maioria das abordagens conexionistas existentes, como por exemplo:

**O IGMN aprende de forma bastante agressiva.** A rede neural proposta nesta tese aprende incrementalmente utilizando apenas uma passada (época) sobre os dados de treinamento, ou seja, cada dado pode ser imediatamente utilizado e descartado. Além disso, O IGMN produz estimativas razoáveis baseadas em poucos dados de treinamento, e estas estimativas são aprimoradas à medida que novos dados chegam.

**O processo de aprendizagem prossegue continuamente.** Diferentemente de outros modelos neurais, o IGMN não requer fases distintas para o aprendizado e a utilização, ou seja, o aprendizado pode prosseguir de forma contínua e perpétua. Em outras palavras, o IGMN pode sempre melhorar seu desempenho à medida que novos dados chegam, o que permite à rede neural se adaptar às mudanças do ambiente.

**O modelo proposto é relativamente robusto ao *overfitting*.** Diferentemente do que ocorre à maioria dos modelos de redes neurais, no IGMN as equações utilizadas no cálculo das estimativas são relativamente robustas ao problema de *overfitting*. De fato, mesmo se um neurônio for adicionado para cada dado de treinamento (como ocorre no GRNN (SPECHT, 1991)) a aproximação da função alvo não será afetada de forma significativa pelo ruído presente nos dados (principalmente se o ruído for *branco*) ou por valores extremos e infrequentes. Isto ocorre porque as estimativas calculadas pelo IGMN correspondem a uma *interpolação suave*, isto é, a resposta produzida pela rede neural é equivalente à média das ativações de cada um dos neurônios ponderada pelas probabilidades a posteriori, o que faz com que o ruído *branco* (que possui média zero) e os valores raros e infrequentes (que possuem baixas probabilidades a posteriori) não afetem os resultados de forma significativa.

**Não é necessário ajustar os parâmetros de configuração de forma fina.** Apesar do IGMN ter diversos parâmetros de configuração, apenas um único parâmetro, o erro máximo de aproximação normalizado ( $\varepsilon_{max}$ ), precisa ser realmente ajustado de acordo com as características do dados e do problema em questão. Além disso, este parâmetro é relativamente fácil de configurar porque em geral o erro máximo permitido em uma dada tarefa é geralmente conhecido, principalmente se este erro for normalizado.

**O modelo proposto não depende das condições de inicialização.** Diferentemente de outros modelos neurais, o IGMN não possui nenhuma inicialização e/ou decisão aleatória, e conseqüentemente os resultados produzidos serão sempre idênticos para uma mesma mesma base de dados e configuração. Assim para se obter resultados estatisticamente válidos não é necessário que o mesmo experimento seja repetido diversas vezes utilizando diferentes números aleatórios.

**O IGMN não é suscetível a mínimos locais.** O algoritmo de aprendizado utilizado pelo IGMN não é baseado na minimização iterativa do erro de predição nem na descida do gradiente, e assim este algoritmo não realiza uma busca por um mínimo na superfície de erro. Ao invés disso, o IGMN aprende instantaneamente as correlações entre diferentes estímulos sensoriais e motores, e estas correlações são utilizadas para a estimação de um determinado estímulo baseado nos demais estímulos.

**A topologia da rede é definida de forma automática e incremental.** No início do aprendizado o IGMN possui apenas um neurônio em cada região cortical e associativa, e outros neurônios são adicionados quando necessário de acordo com o erro instantâneo máximo de predição. Portanto o usuário não precisa configurar manualmente a topologia do IGMN. Além disso, esta topologia não precisa ser mantida fixa durante processo de aprendizado.

**O IGMN resolve o dilema da estabilidade-plasticidade.** O IGMN possui um mecanismo, inspirado na teoria da Inteligência Artificial Construtivista (DRESCHER, 1991), que decide se novos neurônios precisam ser adicionados para acomodar uma nova informação ou se apenas uma pequena mudança nos neurônios existentes é suficiente para que a rede neural assimile uma nova informação. Assim o IGMN é estável porque consegue preservar os conhecimentos adquiridos nas unidades existentes, e é plástico porque aloca sempre que necessário novas unidades para acomodar as informações que não podem ser explicadas de forma satisfatória pelos esquemas existentes.

**O modelo proposto não sofre de interferência catastrófica.** No IGMN o conhecimento é adquirido de forma local, e assim novas informações afetam somente poucos neurônios, ou seja, apenas as unidades responsáveis pelo conhecimento de uma determinada região do espaço de estados são afetadas pelos dados correntes. Além disso, a assimilação dos dados correntes é limitada porque se estes diferirem de forma significativa dos esquemas existentes novas unidades serão criadas para acomodar um novo “contexto”.

**A ordem de apresentação dos dados não afeta de forma significativa os resultados.** Através de diversos experimentos realizados utilizando o modelo proposto (Seção 5.1) é demonstrado que o IGMN produz estimativas razoáveis da função alvo independente da ordem em que os dados são apresentados. Ou seja, o IGMN pode ser utilizado tanto em sistemas *on-line*, onde os dados geralmente chegam de forma ordenada, quanto em sistemas *off-line*, onde os dados podem chegar de forma não ordenada e randômica.

**O modelo proposto é inerentemente probabilístico.** Como foi dito anteriormente, o

IGMN é baseado em um *framework* probabilístico (modelos de mistura gaussianos) e aproxima assintoticamente a superfície de decisão bayesiana ótima. Em outras palavras, o IGMN fornece a hipótese de máxima verossimilhança em função dos dados recebidos até o momento, e essa hipótese é melhorada à medida que novos dados chegam. Assim as estimativas fornecidas pelo IGMN podem ser consideradas ótimas do ponto de vista estatístico, pois refletem a melhor “decisão” que se pode tomar baseada nos dados disponíveis.

**As representações internas criadas pelo IGMN são bastante úteis.** Diferente do que acontece com outros modelos neurais (especialmente as redes MLP), as representações internas, criadas nas regiões associativa e corticais, correspondem a agrupamentos naturais (*clusters*) do espaço de estados que podem ser facilmente interpretadas por um especialista da área. Assim o IGMN não é uma “caixa preta”, ou seja, as respostas produzidas pela rede neural podem ser interpretadas e justificadas através da análise das probabilidades calculadas em cada nodo da rede neural.

**O IGMN fornece o nível de confiança de suas estimativas.** O IGMN fornece não apenas uma estimativa pontual (o valor esperado) de um determinado estímulo sensorio-motor, mas também a distribuição probabilística (variância e covariância) desta estimativa. Em outras palavras, o IGMN fornece o nível de confiança de suas estimativas, o que permite que melhores decisões baseadas no nível de confiança dessas estimativas, sejam tomadas.

**O modelo proposto pode ser utilizado em tarefas de aprendizado supervisionado, não-supervisionado e por reforço.** Através de diversos experimentos, descritos nos Capítulos 5 e 6, é demonstrado que o IGMN é bastante útil em tarefas de aprendizado supervisionado (aproximação de funções, identificação de sistemas e predição de séries temporais) e não supervisionado (categorização, formação incremental de conceitos e mapeamento robótico). Além disso, o IGMN também pode ser utilizado como um aproximador de funções em tarefas de aprendizado por reforço.

**O fluxo de informações é bidirecional.** Diferentemente da maioria dos modelos neurais existentes, no IGMN o fluxo de informações não é unidirecional da camada de entrada até a camada de saída. Ao invés disso o IGMN possui diversas regiões corticais, e o estímulo recebido em determinadas regiões corticais pode ser utilizado para estimar os valores dos demais estímulos. Desta forma uma mesma rede neural *parcialmente* treinada (como o IGMN aprende continuamente nunca se considera que a rede neural está completamente treinada) pode ser utilizada para estimar os valores de uma determinada função alvo,  $f(\cdot)$ , bem como os valores de sua respectiva inversa,  $f(\cdot)^{-1}$ .

**O modelo proposto pode ser utilizado para resolver problemas diretos e inversos.** De acordo com Bishop (1995), em muitas aplicações potenciais de redes neurais existe um problema direto (*forward*) bem definido que é caracterizado por um mapeamento funcional (uni-valorado), mas o problema inverso correspondente possui múltiplas soluções válidas. Embora a maioria das redes neurais existentes (MLP, RBF, GRNN e ARTMAP, por exemplo) não forneça boas estimativas em regiões do espaço de estados onde a *função* alvo possui múltiplas soluções (ou seja, a *função* alvo não é uma função), o IGMN é capaz de produzir respostas válidas mesmo nas regiões do espaço de estados que possuem múltiplas soluções. Neste caso as estimativas fornecidas pelo IGMN correspondem à hipótese de máxima verossimilhança, ou seja, as respostas fornecidas se encontram no ramo (*branch*) associado com a maior probabilidade de massa.

**O IGMN possui uma boa performance computacional.** Embora o cálculo da função de densidade de probabilidade (pdf) requeira a inversão de diversas matrizes de covariância a cada interação, a performance computacional do IGMN é melhor do que a de um modelo de mistura gaussiano equivalente porque diferentes estímulos sensorio/motores são processados em regiões corticais distintas, e assim as matrizes de covariância  $C$  são *quebradas* em matrizes de covariância menores associadas a cada estímulo sensorio/motor. Além disso, em problemas de grande dimensionalidade o número de regiões corticais pode ser aumentado de forma a reduzir ainda mais o tamanho das respectivas matrizes, o que torna possível estabelecer um compromisso entre precisão e performance computacional, o que é bastante útil em aplicações de tempo real.

Baseado nos resultados descritos acima é possível afirmar que os objetivos desta tese foram atingidos, ou seja, foi possível propor, implementar e validar uma nova abordagem conexionista para a aproximação incremental de funções e predição em tempo real. De fato, o modelo proposto pode ser utilizado com sucesso em diversas aplicações potenciais, como por exemplo regressão *on-line*, identificação de sistemas, predição de séries temporais, formação de conceitos, controle inteligente e mapeamento robótico. Além disso, os experimentos realizados demonstram que o poder de representação do IGMN, aliado ao seu algoritmo de aprendizado eficiente, permite que se expanda o horizonte de aplicações nas quais as redes neurais podem ser utilizadas com sucesso, abrindo assim novas direções de pesquisa nas quais importantes contribuições do estado da arte podem vir a ser feitas.