# Probabilistic Models for Learning the Dynamics Model of Robots

Gsenger Othmar, BSc.
othmar@sbox.tugraz.at

Institute for Theoretical Computer Science (IGI)
Graz University of Technology
Inffeldgasse 16b
8010 Graz, Austria


Master Thesis

Supervisor: o.Univ.-Prof.Dr.Wolfgang Maass
Assessor: o.Univ.-Prof.Dr.Wolfgang Maass

May, 2013

STATUTORY DECLARATION

*I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.*

..... .....

(date) (signature)

EIDESSTATTLICHE ERKLÄRUNG

*Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.*

Graz, am ..... .....

(Unterschrift)

# Acknowledgements

I would like to thank my family for supporting me all the time during my studies. I want to thank my mother especially for her generous financial support, despite the fact that everything took much longer than expected.

I also want to thank DI Elmar Rückert and DI Gerhard Neumann for supervising me while writing this master thesis and o. Univ.-Prof. Dr. Wolfgang Maass for being my mentor.

# Abstract

In this thesis I investigate three probabilistic regression models from the literature and try to use them for learning the dynamics model of robots, i.e. the function that describes the transition between two states when executing a certain action. I add regularization terms to the error function where necessary to address the problem of overfitting and numerical instability and derive the corresponding learning rules. Then I evaluate these models on four simulated robotic tasks using my MATLAB implementation of the models. A simple one-dimensional toy task is used to analyze and visualize the characteristics of the approaches. In a second and third experiment I test these models on multi-link arms and analyze their robustness to noise. Finally, in the most complex experiment I use an existing simplified model of a humanoid robot in a balancing task. The learned dynamics model is used in combination with a well-known movement planning algorithm to solve a balancing problem, where the robot gets pushed with different forces. The results demonstrate that it is possible to predict and plan movement using these models. The Hierarchical Mixtures of Experts model shows the worst performance, against the expectation to work best on high dimensional data because it has the least number of model parameters. However, this can be improved using different initialization approaches. The Gated Linear Regression with Gaussian Noise Model and Conditioned Mixture of Gaussian show good performance on small datasets. Additionally, the Conditioned Mixture of Gaussian has the advantage of training both the kinematics and inverse kinematic model at the same time, without overhead compared to the other models.

**Keywords:** model learning, probabilistic models, movement, dynamics, robots

# Kurzfassung

In dieser Arbeit beschäftige ich mich mit drei bekannten wahrscheinlichkeitstheoretischen Regressionsmodellen und versuche Sie zum Lernen von Dynamik-Modellen von Robotern, z.B. der Funktion, die die Änderung durch Ausführen von Aktionen zwischen zwei Zuständen beschreibt, zu benutzen. Die Fehlerfunktion dieser Modelle ergänze ich um Regularisierungs-Ausdrücke, wo das durch Probleme mit Overfitting und numerischer Instabilität erforderlich ist. Daraus leite ich die zugehörigen Lernregeln für die Modelle ab. Ich vergleiche diese Modelle anhand von vier Aufgaben aus der Robotik mittels meiner MATLAB Implementierung der Modelle. Zunächst visualisiere ich die Modelle und analysieren ihr Grundverhalten anhand einer einfachen eindimensionalen Funktion. Im zweiten und dritten Experiment teste ich die Modelle an Armen mit mehreren Gelenken. Dabei untersuche ich auch das Verhalten bei Rauschen. Abschließend und als schwierigstes Experiment versuche ich mit einem vereinfachten Modell eines humanoiden Roboters das Gleichgewicht zu halten. Dazu nutze ich die gelernten Modelle in Kombination mit einem bekannten Algorithmus zur Bewegungsplanung. Ich zeige, dass mit Hilfe der gelernten Modelle Bewegungen in der Robotik geplant und durchgeführt werden können. Im Vergleich verschiedener Modelle schneidet das *Hierarchical Mixtures of Experts* Modell überraschend am schlechtesten ab. Ich erwartete hier einen Vorteil bei hochdimensionalen Problemen, da es am wenigsten Modellparameter benötigt. Der Grund für das schlechte Abschneiden liegt an der schwierigen Initialisierung dieses Modells, hier können durch andere Initialisierungsverfahren noch Verbesserungen erreicht werden. Die anderen beiden Modelle, *Gated Linear Regression with Gaussian Noise Model* und *Conditioned Mixture of Gaussian*, zeigten sehr gute Resultate schon bei kleinen Datensätzen. Das *Conditioned Mixture of Gaussian* Modell bietet darüber hinaus den Vorteil das Umkehrmodell ohne zusätzlichen Rechenaufwand mitzulernen.

**Stichwörter:** Modelllernen, Wahrscheinlichkeitsmodelle, Bewegung, Dynamic, Robotik

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

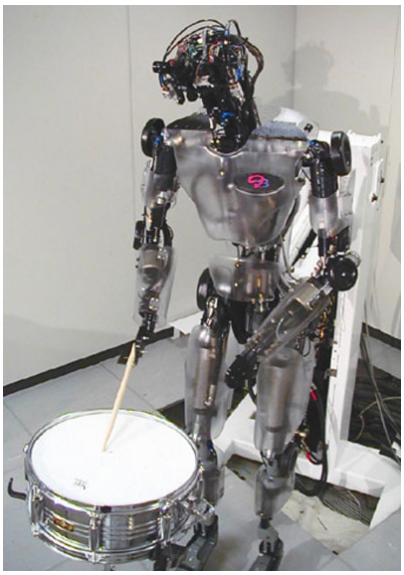


Figure 1.1: Picture of a humanoid robot [8].

Movement planning is a central topic in robotics. Recent work has introduced the idea that planning is accomplished through probabilistic inference [5]. Although relatively little work has been done to specify the computational principles involved in goal-directed decision making (planning), portraying it as probabilistic inference fits into a broad movement within both psychology and neuroscience [20]. Model predictive control also known as stochastic optimal control (SOC) has been shown to work in robotic tasks such as legged locomotion, hand manipulation and ball bouncing, not only in simulations, but also in real life applications [9].

Movement planning with SOC depends on a known model of the robot's dynamics. This model specifies a transition function from a current state and an applied action to the next state. Having a dynamics model comes with the following advantages:

- Only executable trajectories can be learned.
- Noise can be modeled.
- Constraints for control, joint, obstacles and other desired features can be added.

Therefore I want to investigate model based approaches. However, these approaches depend on having a dynamics model. As dynamics models are often very complex and

unknown, this thesis aims to find and evaluate models that can automatically learn such models. To achieve this, methods from the field of Machine Learning will be used to automatically learn such models based on observed data.

I want to evaluate standard methods and compare their performance on dynamic problems of different complexity. Mathematically inspired models are used, rather than biologically inspired ones. As observations and predictions always include some uncertainty, this brings us to the field of Probability Theory. Standard probabilistic models will be evaluated and slightly modified if necessary.

## 1.2 Dynamics Model of Robots

The dynamics model of a robot defines the mapping of a state and an action to a new state. The state is the position and velocity of the robot's elements and the actions might be forces applied by muscles or the input of a motor controller.

Humanoid robots as shown in Figure 1.1 often consist of many links and even more artificial muscles, leading to a high dimensional action and state space. In real live applications noise gets added to the signal that controls the action, as well as to the sensors that measure the state of the robot. As the state and action space are non-discrete, there exists an infinite number of possible movement trajectories. It is only possible to use a limited number of movement trajectories that have already been observed to learn the dynamics model. This leads to the problem that new, unobserved trajectories have to be predicted. Thus a model that shows a good generalization of the model function without overfitting the training data is needed.

## 1.3 Model Learning as Probabilistic Regression Problem

I observe data generated by a function

$$f : \mathbf{x} \rightarrow \mathbf{y},$$

with the input $\mathbf{x} \in \mathbb{R}^{D_x}$ and the output $\mathbf{y} \in \mathbb{R}^{D_y}$. The function $f$ is unknown and should be approximated by the used model. This is done by observing multiple samples $\mathbf{x}_n \rightarrow \mathbf{y}_n$, giving us the training dataset

$$X = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}, \quad Y = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_n \end{pmatrix}.$$

I try to model a probability function $p(\mathbf{y}|\mathbf{x})$ that I use for function approximation

$$\tilde{f}(\mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}),$$

and assume that the observation is produced by a Gaussian process.

All used models are based on the idea that the mean of the Gaussian process can be approximated by a piecewise linear regression consisting of $K$ pieces. It is also possible to replace the input $\mathbf{x}$ with a feature vector $\phi(\mathbf{x}) \in \mathbb{R}^{D_\phi}$ [4], but for easier notation I will skip this in the rest of the document.

## 1.4 Necessary work

In this thesis I investigate three probabilistic regression models from the literature and try to use them for learning the dynamics model of robots. The Gated Linear Regression with Gaussian Noise Model (GLR) and Conditioned Mixture of Gaussian (CMG) are well documented [4] and learning rules are known. Thus I implemented the models in MATLAB. As computation is limited by the floating point precision of the implementation, additional regularization terms had to be added. I show how the learning rules can be derived from a modified error function that addresses problems with numerical instability and overfitting. As for the Hierarchical Mixtures of Experts (HME) model, no learning rules or details are included in the publication [3]. Therefore I introduce my own error functions and derive learning rules, that differ from the original implementation of the author. I test these three models on four tasks. The toy task, the two and five link robot and all regression models are implemented by myself. For movement planning and the simulation of the simplified humanoid robot I use an existing implementation [17]. Finally I present all simulation results and interpret their meaning.

## 1.5 Notation

### 1.5.1 Matrix and vector notations

| | | |
|---|---|---|
| $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ | ... | a row vector |
| $\mathbf{x}^T$ | ... | transpose of vector $\mathbf{x}$ |
| $x_i$ | ... | i-th element of vector $\mathbf{x}$ |
| $\mathbf{X}$ | ... | a matrix |
| $\mathbf{x_i}$ | ... | i-th row of matrix $\mathbf{X}$ |
| $\mathbf{x_i}^T$ | ... | i-th row of matrix $\mathbf{X}$ as a column vector |
| $(\mathbf{x}^T)_j$ | ... | j-th column of matrix $\mathbf{X}$ as a row vector |
| $x_{i,j}$ | ... | j-th element of $\mathbf{x_i}$ |

### 1.5.2 Functions

$$\text{diag}(\mathbf{x}) = \begin{pmatrix} x_1 & 0 & \ldots & 0 \\ 0 & x_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & x_{d_x} \end{pmatrix}$$

$|\mathbf{\Sigma}|$ ⋯ determinant of $\mathbf{\Sigma}$

$$\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)^T\right)$$

### 1.5.3 Constants and recurring symbols

| | | | | |
|---|---|---|---|---|
| $N$ | | | ... | Number of samples, |
| | | | | Number of rows of the matrix |
| $n$ | $\in$ | $[1, \ldots, N]$ | ... | Index of the sample |
| $D$ | | | ... | Number of columns of the matrix |
| $D_x$ | | | ... | Dimension of samples, |
| | | | | Number of columns of the matrix $\mathbf{X}$ |
| $d_x$ | $\in$ | $[1, \ldots, D_x]$ | ... | Index of the dimension |
| $D_y$ | | | ... | Dimension of output, |
| | | | | Number of columns of the output matrix $\mathbf{Y}$ |
| $d_y$ | $\in$ | $[1, \ldots, D_y]$ | ... | Index of the output dimension |
| $K$ | | | ... | Number of Clusters, |
| | | | | Number of Experts |
| $k$ | $\in$ | $[1, \ldots, K]$ | ... | Index of the cluster |
| $G$ | | | ... | Number of gates, |
| | | | | Number of gating nodes |
| $g$ | $\in$ | $[1, \ldots, K]$ | ... | Index of the gate |

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \quad \ldots \text{ The unity matrix}$$

$$\mathbf{I}_0 = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \\ 0 & 0 & \ldots & 0 & 0 \end{pmatrix} \quad \ldots \text{ The unity matrix with } i_{D,D} \text{ being 0}$$

# Chapter 2

# Related work

Movement generation for high-dimensional stochastic robots is challenging. Movement Primitive (MP)s are introduced to reduce the dimensionality of the learning problem [19, 7, 1, 6, 13, 17]. MPs are elementary parts of a movement which can be sequenced or superimposed in time [11, 12, 17]. In robotics, model-free and model-based movement primitives are used. As model learning is challenging, most movement generation applications on real robots are model free [19, 1].

Another approach is model-predictive control, which uses the model's dynamic in the process of planning. With known model dynamics these approaches can be used to learn a movement policy [10]. There are well performing methods based on optimal control theory using a local linearization of the dynamics model and a quadratic cost function like Iterative Linear Quadratic Regulator (ILQR) [10] or Iterative Linear-Quadratic-Gaussian (ILQG) [22]. Stochastic Dynamic Programming is another approach from the area of SOC theory, but it uses a second order approximation of the dynamics model [21]. Approximate inference control (AICO) is an extension to ILQG through a model for which the maximum likelihood (ML) trajectory coincides with the optimal trajectory and which reproduces the classical SOC solution in the Linear-Quadratic-Gaussian (LQG) case. The algorithm then utilizes approximate inference methods (similar to expectation propagation), that efficiently generalize to non-LQG systems [23]. It is also possible to combine the idea of MP with methods from SOC leading to Planning Movement Primitive (PMP) [18].

However, all these models rely on a known dynamics model. In this work I use three simple models to formulate a dynamics model. An example for successful work in this area are Locally Weighted Regression (LWR) [2], its enhancement Locally Weighted Projection Regression (LWPR) [24] and Gaussian process regression [16]. It has been shown that all these methods provide usable models for robot dynamics [15, 14]. These methods basically differ in the choice of the kernel function $\phi(\mathbf{x})$.

In contrast to these model learning methods, this work concentrates on a probabilistic regression approach, leaving the question for an adequate kernel function out of the scope. I decided to evaluate the Hierarchical Mixtures of Experts (HME) [3] model as it has

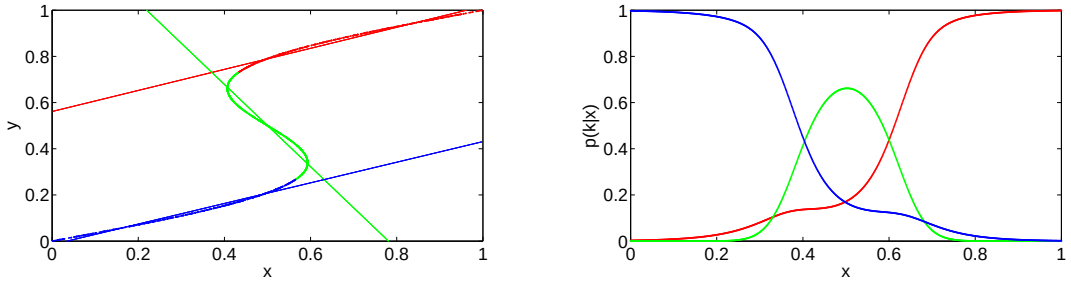
been proposed to fit high dimensional data well. It will be compared to the Gated Linear Regression with Gaussian Noise Model (GLR), that is a modified version of the HME with a simpler gating network and the Conditioned Mixture of Gaussian (CMG) [4] model, that is a reformulation of the GLR with a different learning approach.

# Chapter 3

# Gated Linear Regression with Gaussian Noise Model

The Gated Linear Regression with Gaussian Noise Model is a mixture of linear regression models. Mixtures are used to represent an arbitrary multi-dimensional function $f : \mathbf{x} \rightarrow \mathbf{y}$. This is done by partitioning the input space into separate regions and applying an independent linear regression model to each region. The partitioning of the input space is performed by the *gating distributions* and the regression is done by so-called *experts* [3]. Figure 3.1 shows an one dimensional function and the linear regression model of three experts. I discuss model details in subsection 3.1.

## 3.1 Model




(a) Training data gets assigned to an expert and the experts learn the linear model.

(b) Probability $p(\mathbf{z}_n = \mathbf{c}_k | x_n)$ of each expert being active. This is determined by the gates.

Figure 3.1: Gated Linear Regression with Gaussian Noise Model performing linear regression with three experts: The function is colored in the color of the expert $k$.

The Gaussian Gating separates the input into several clusters. The number of clusters is given by the parameter $K$. The latent variable $\mathbf{Z} \in \mathbb{R}^{K \times N}$ determines the cluster that is active for a given sample. $\mathbf{Z}$ is a binary latent variable. Each line corresponds to a

training sample given by the index $n$ and each column corresponds to a cluster $k$. In every row exactly one value is one and all others are zero, viz only one cluster can be active at a given time.

For each cluster $k$ two Gaussian distributions are used, one to model the output for the given cluster and one to determine the probability of the cluster being active given the input. Figure 3.2 shows the graphical representation of this model.

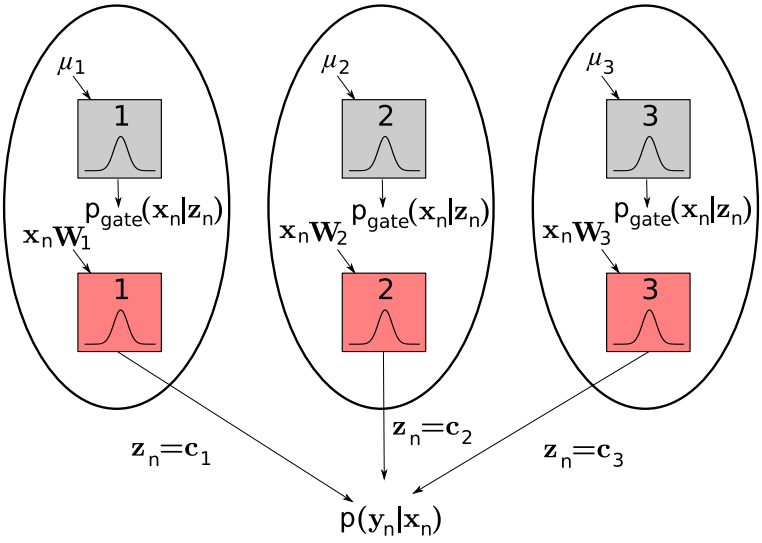



Figure 3.2: Schematic representation of the Gated Linear Regression with Gaussian Noise Model. Grey boxes are the gating distributions $\mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)$ and the red boxes are the output distributions $\mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k, \mathrm{diag}(\mathbf{s}_k))$.

To simplify the notation, I introduce the matrix $\mathbf{C} = \mathbf{I}$ being the unity matrix of size $K \times K$. The $k$ 'th row vector is given by $\mathbf{c}_k$. This row vector is all zero except for the $k$'th value, which means I can use the notation $\mathbf{z}_n = \mathbf{c}_k$ to describe that for the $n$'th sample the $k$'th cluster is being active.

The probability of a gating node producing the observed input sample $\mathbf{x}_n$ is given by

$$p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k) = \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k).$$

For the prior probability of a cluster being active I use a constant

$$p_{\text{gate}}(\mathbf{z}_n = \mathbf{c}_k) = \frac{1}{K}.$$

This leads to the mixing coefficient

$$\begin{aligned}
\pi_k(\mathbf{x}_n) &= p_{\text{gate}}(\mathbf{z}_n = \mathbf{c}_k|\mathbf{x}_n), \\
\pi_k(\mathbf{x}_n) &= \frac{p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)p_{\text{gate}}(\mathbf{z}_n = \mathbf{c}_k)}{p(\mathbf{x}_n)}, \\
\pi_k(\mathbf{x}_n) &= \frac{p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)}{Kp(\mathbf{x}_n)}, \\
\pi_k(\mathbf{x}_n) &= \frac{p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)}{\sum_{k=1}^{K} p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})}.
\end{aligned}$$

An expert produces an output $\mathbf{y}_n$ with the probability given by

$$p_{\text{expert}}(\mathbf{y}_n|\mathbf{z}_n = \mathbf{c}_k, \mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k, \text{diag}(\mathbf{s}_k)),$$

where $\mathbf{x}\mathbf{W}_k$ represents the linear model. This results in a piecewise linear regression with Gaussian noise represented by $\text{diag}(\mathbf{s}_k)$.

The complete model is given by

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{n=1}^{N}\sum_{k=1}^{K} p_{\text{expert}}(\mathbf{y}_n|\mathbf{z}_n = \mathbf{c}_k, \mathbf{x}_n)\pi_k(\mathbf{x}_n).$$

The joined probability of gate and expert producing both $\mathbf{x}_n$ and $\mathbf{y}_n$ is

$$p(\mathbf{x}_n, \mathbf{y}_n|\mathbf{z}_n = \mathbf{c}_k) = p_{\text{expert}}(\mathbf{y}_n, \mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k).$$

## 3.2 Learning

I want to learn the model parameters

$$\theta = \left\{ \begin{array}{lcl} \mathbf{\Sigma} & \in & \mathbb{R}^{K\times D_x\times D_x} \\ \mu & \in & \mathbb{R}^{K\times D_x} \\ \mathbf{S} & \in & \mathbb{R}^{K\times D_y} \\ \mathbf{W} & \in & \mathbb{R}^{K\times D_x\times D_y} \end{array} \right\}.$$

Unfortunately, there exists no closed form solution for this problem. However I can use an iterative approach such as the Expectation Maximisation (EM) algorithm. There are also alternative ways to solve this problem, like variational inference or Markov-Chain-Monte-Carlo (MCMC). The EM algorithm consists of the following two steps.

### 3.2.1 E-Step

The first step, also known as Expectation (E)-step, is used to determine the probabilities for the latent variable $\mathbf{Z}$. The responsibility is given by the probability of a cluster being responsible for producing the sample

$$p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{x}_n, \mathbf{y}_n) = \frac{p_{\text{expert}}(\mathbf{y}_n, \mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k)}{\sum_{\tilde{k}=1}^{K} p_{\text{expert}}(\mathbf{y}_n, \mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p(\mathbf{z}_n = \mathbf{c}_{\tilde{k}})}.$$

The prior probability $p(\mathbf{z}_n = \mathbf{c}_k)$ is independent of $k$ and constant for the model, so the responsibility can be reduced to

$$\gamma_{n,k} = p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{x}_n, \mathbf{y}_n) = \frac{p_{\text{expert}}(\mathbf{y}_n, \mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_k)}{\sum_{\tilde{k}=1}^{K} p_{\text{expert}}(\mathbf{y}_n, \mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p_{\text{gate}}(\mathbf{x}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})}.$$

I also introduce the expectation for the number of samples generated by the cluster $k$

$$N_k = \mathop{\mathbb{E}}_{\mathbf{Z}}\left\{\sum_{n=1}^{N} \mathbf{z}_{n,k}\right\} = \sum_{n=1}^{N} \gamma_{n,k}.$$

### 3.2.2 M-Step

The second step, called Maximisation (M)-step, is applied to optimize the model parameters, while keeping the probabilities of the latent variable $\mathbf{Z}$ constant.

The M-step maximizes the log-likelihood. The log-likelihood of the model is calculated by marginalizing the latent variable $\mathbf{z}$ out and joining the probabilities over all samples $n$. This can be written as

$$\ln p(\mathbf{Y}|\theta) = \ln \prod_{n=1}^{N} \sum_{k=1}^{K} p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k),$$

$$\ln p(\mathbf{Y}|\theta) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n = \mathbf{c}_k)\pi_k(\mathbf{x}_n).$$

The complete log-likelihood including the latent variable $\mathbf{Z}$ takes the form

$$\ln p(\mathbf{Y}, \mathbf{Z}|\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \ln \left(p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n = \mathbf{c}_k)\pi_k(\mathbf{x}_n)\right).$$

Now I can use the expectation of the latent variable as determined in the E-step

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{Z}}\{\ln p(\mathbf{Y},\mathbf{Z}|\theta)\} &= \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{n,k}\ln\left(p(\mathbf{y}_n|\mathbf{x}_n,\mathbf{z}_n=\mathbf{c}_k)\pi_k(\mathbf{x}_n)\right),\\ \mathop{\mathbb{E}}_{\mathbf{Z}}\{\ln p(\mathbf{Y},\mathbf{Z}|\theta)\} &= \sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{n,k}\ln\left(\mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k,\mathrm{diag}(\mathbf{s}_k))\pi_k(\mathbf{x}_n)\right).\end{aligned}$$

Differentiating the log-likelihood by a model parameter and setting the result to zero leads to an equation for the corresponding model parameter that maximizes the likelihood function

$$\begin{aligned}0 &= \nabla_{\mathbf{W}_k}\sum_{n=1}^{N}\sum_{k=1}^{K}\gamma_{n,k}\ln\left(\mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k,\mathrm{diag}(\mathbf{s}_k))\mathcal{N}(\mathbf{x}_n|\mu_k,\mathbf{\Sigma}_k)\frac{1}{Kp(\mathbf{x}_n)}\right),\\ 0 &= \nabla_{\mathbf{W}_k}\sum_{n=1}^{N}\gamma_{n,k}\ln\Big(\mathcal{N}\big(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k,\mathrm{diag}(\mathbf{s}_k)\big)\Big),\\ 0 &= \nabla_{\mathbf{W}_k}\sum_{n=1}^{N}\gamma_{n,k}\ln\left(\frac{1}{(2\pi)^{\frac{D_y}{2}}|\,\mathrm{diag}(\mathbf{s}_k)|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{y}_n-\mathbf{x}_n\mathbf{W}_k)\,\mathrm{diag}(\mathbf{s}_k)^{-1}(\mathbf{y}_n-\mathbf{x}_n\mathbf{W}_k)^T\right)\right),\\ 0 &= \nabla_{\mathbf{W}_k}\sum_{n=1}^{N}\gamma_{n,k}\left(-\frac{1}{2}(\mathbf{y}_n-\mathbf{x}_n\mathbf{W}_k)\,\mathrm{diag}(\mathbf{s}_k)^{-1}(\mathbf{y}_n-\mathbf{x}_n\mathbf{W}_k)^T\right).\end{aligned}$$

This leads to the update rule for the weights for the linear regression of the output

$$\mathbf{W}_k=\sum_{n=1}^{N}(\mathbf{x}_n^T\gamma_{n,k}\mathbf{x}_n)^{-1}\mathbf{x}_n^T\gamma_{n,k}\mathbf{y}_n.$$

By differentiating the log-likelihood by the other parameters as presented above, I can derive the update rules. The noise on the output:

$$\mathbf{s}_k=\frac{1}{N_k}\sum_{n=1}^{N}(\gamma_{n,k}(\mathbf{y}_n-\mathbf{x}_n\mathbf{W}_k)^2)^T.$$

The center of an input cluster cluster:

$$\mu_k=\frac{1}{N_k}\sum_{n=1}^{N}\gamma_{n,k}\mathbf{x}_n.$$

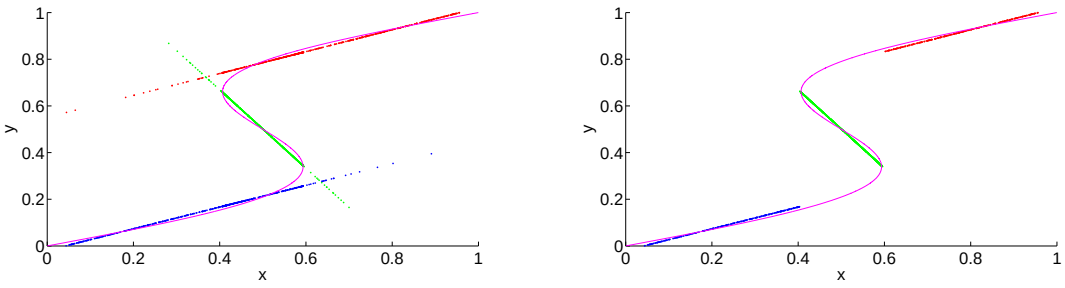
To calculate the covariance matrix $\mathbf{\Sigma}_k$ of an input cluster it is necessary to first subtract

the mean given by

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n,$$

from the original input. This gives the mean-free input $\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$. Finally, the covariance can be determined by the weighted summed squares of $\tilde{\mathbf{x}}_n$

$$\mathbf{\Sigma}_k = \frac{1}{N_k}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\gamma_{n,k}\tilde{\mathbf{x}}_n^T.$$




(a) Random solution

(b) Most probable solution.

Figure 3.3: This shows the output of the trained Gated Linear Regression with Gaussian Noise Model working on a test set. The data points are colored in the color of the expert generating the point. The original function is shown in magenta.

### 3.2.3 Regularization

To avoid numerical problems with probabilities very close to zero it is necessary to keep the weights $\mathbf{W}_k$ small and the covariance $\mathbf{\Sigma}_k$ greater than zero.

Keeping the weights small was done by adding a regularization term that contributes to the error function

$$E = \mathop{\mathbb{E}}_{\mathbf{Z}}\{\ln p(\mathbf{Y}, \mathbf{Z}|\theta)\} + \sum_k \alpha^2\mathbf{W}_k^2.$$

Optimizing this error function instead of the log-likelihood results in the new update rule

$$\mathbf{W}_k = \sum_{n=1}^{N}(\mathbf{x}_n^T\gamma_{n,k}\mathbf{x}_n + \alpha^2\mathbf{I})^{-1}\mathbf{x}_n^T\,\mathrm{diag}(\gamma_n)\mathbf{y}_n.$$

Keeping $\mathbf{\Sigma}_k$ greater than zero is done by adding $\alpha_{gate}$ samples with a covariance of $\sigma_{gate}\mathbf{I}$ to each cluster. This results in the modified update rule

$$\mathbf{\Sigma}_k = \frac{1}{N_k + \alpha_{gate}}(\alpha_{gate}\sigma_{gate}\mathbf{I} + \frac{1}{N_k}\sum_{n=1}^{N}\tilde{\mathbf{x}}_n\gamma_{n,k}\tilde{\mathbf{x}}_n^T).$$

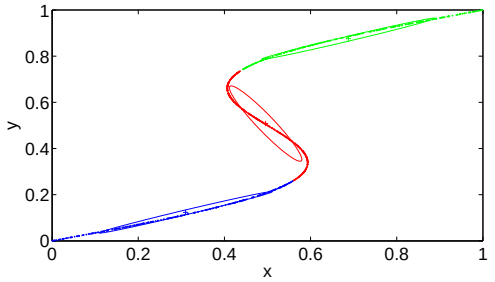
### 3.2.4 Initialization

Initialization is done by splitting the data into $K$ clusters using K-means. For each cluster an expert gets trained using standard linear regression on that data slice. The gates are initialised with the mean and variance of the data assigned to the cluster.

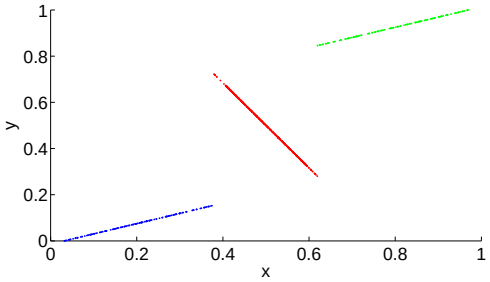
# Chapter 4

# Conditioned Mixture of Gaussian

The Conditioned Mixture of Gaussian model is based on a simple Mixture of Gaussian model [4]. Both the input data $\mathbf{X}$ and the output $\mathbf{Y}$ are concatenated into one data matrix $\mathbf{U}$. Then a Mixture of Gaussian model is trained based on this data. After learning, the model can be transformed into a gated linear regression model by conditioning onto arbitrary dimensions of the data matrix, e.g. conditioning on the original input data $\mathbf{X}$. Therefore it is possible to determine which dimensions are treated as input and which are treated as output after learning has finished. This has the advantage that there is no need to train different models for different input/output combinations. For example, an inverse dynamics model can easily be determined by conditioning on the output instead of the input. Figure 4.1 shows a simple example for the mixture of three Gaussian distributions.




(a) Training data and the trained Gaussian distributions are shown. Ellipses represent the standard deviation.




(b) Simulation of the model, using $\mathbf{x}$ as input.

Figure 4.1: Gaussian mixture with three experts. The function's color corresponds to the data point's expert $k$ . Simulation is done by choosing the most probable expert first and then calculating the expected output $\mathbf{y}$ of that expert.

## 4.1 Model

The model does not treat input and output differently. I define a new matrix $\mathbf{U}$ containing both the input and the output

$$\mathbf{U} = (\mathbf{X}, \mathbf{Y}).$$

The model is based on a Gaussian mixture model. Like in the previous model I introduce a binary latent variable $\mathbf{Z}$. The overall probability for observing a data point $\mathbf{u}_n$ is given as a sum over multiple Gaussian distributions, also called clusters

$$p(\mathbf{u}_n) = \sum_{k=1}^{K} p(\mathbf{z}_n = \mathbf{c}_k)\mathcal{N}(\mathbf{u}_n|\mu_k, \mathbf{\Sigma}_k).$$

The mixing coefficients $\pi_k$ represent the prior probabilities for the latent variable $p(\mathbf{z}_n = \mathbf{c}_k) = \pi_k$. Those coefficients have to fulfill the condition $\sum_{k=1}^{K} \pi_k = 1$ . Note that the sum of all probabilities has to be equal to one, which can be seen by marginalizing out $\mathbf{u}_n$

$$\int_{\mathbf{u}_n \in \mathbb{R}^{D_u}} p(\mathbf{u}_n) d\mathbf{u}_n = \sum_{k=1}^{K} p(\mathbf{z}_n = \mathbf{c}_k) = 1.$$

Combining the above, the complete model reads

$$p(\mathbf{U}) = \prod_{n=1}^{N} p(\mathbf{u}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{u}_n|\mu_k, \mathbf{\Sigma}_k).$$

## 4.2 Learning

I want to learn the model parameters

$$\theta = \left\{ \begin{array}{lcl} \mathbf{\Sigma} & \in & \mathbb{R}^{K \times D_u \times D_u} \\ \mu & \in & \mathbb{R}^{K \times D_u} \\ \pi & \in & \mathbb{R}^{K} \end{array} \right\}.$$

Again there exists no closed form solution for this problem, so the EM algorithm is used.

### 4.2.1 E-Step

The responsibility is given by the probability of a cluster being responsible for producing the sample

$$\gamma_{n,k} = p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{u}_n) = \frac{p(\mathbf{u}_n|\mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k)}{\sum_{\tilde{k}=1}^{K} p(\mathbf{u}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p(\mathbf{z}_n = \mathbf{c}_{\tilde{k}})} = \frac{\mathcal{N}(\mathbf{u}_n|\mu_k, \mathbf{\Sigma}_k)\pi_k}{\sum_{\tilde{k}=1}^{K} \mathcal{N}(\mathbf{u}_n|\mu_{\tilde{k}}, \mathbf{\Sigma}_{\tilde{k}})\pi_{\tilde{k}}}.$$

To simplify the notation, I introduce the expectation of the number of samples generated by the cluster $k$

$$N_k = \mathop{\mathbb{E}}_{\mathbf{Z}} \left\{ \sum_{n=1}^{N} \mathbf{z}_{n,k} \right\} = \sum_{n=1}^{N} \gamma_{n,k}.$$

### 4.2.2 M-Step

The maximum likelihood solution for a Gaussian cluster $k$ can now be calculated independently from the other clusters by weighting the samples with the cluster's responsibility. This results in an update rule for the center $\mu_k$ as weighted sum over all samples

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{n,k} \mathbf{u}_n.$$

To calculate the covariance matrix it is necessary to first subtract the mean

$$\bar{\mathbf{u}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{u}_n$$

from all samples, yielding the new data vector $\tilde{\mathbf{u}}_n = \mathbf{u}_n - \bar{\mathbf{u}}$. The covariance again is calculated as a weighted sum over the mean free samples

$$\mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{n,k} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_n^T.$$

The prior probability of a cluster is the fraction of the number of samples expected to be produced by the cluster over the overall number of samples

$$\pi_k = \frac{N_k}{N}.$$

### 4.2.3 Regularization

Very small variance can lead to clusters specializing on few samples, which can result in overfitting. To reduce this effect I add $\alpha$ samples with a variance of $\sigma_0$ to each cluster. This results in the altered update rule

$$\mathbf{\Sigma}_k = \frac{1}{N_k + \alpha} \left( \alpha \sigma_0 \mathbf{I} + \sum_{n=1}^{N} \gamma_{n,k} \tilde{\mathbf{u}}_n \tilde{\mathbf{u}}_n^T \right).$$

### 4.2.4 Initialisation

Initialisation is done by using K-means to split the data into multiple clusters and calculating the distribution of each cluster independently. All mixing coefficients are initialized

to the same value

$$\pi_k = \frac{1}{K}.$$

## 4.3 Conditioning

For learning it was not necessary to distinguish between input and output data. I used the combined matrix $\mathbf{U}$. When simulating the network it is often more interesting to specify some dimensions of the data as input, that is they are known a priori. To represent the fact that these inputs do not have to be equal to the original inputs $\mathbf{X}$ I will denote them as $\mathbf{B}$. The corresponding outputs will be denoted as $\mathbf{A}$.

Now I can split all samples $\mathbf{U}$ into input samples $\mathbf{U}_b$ and output samples $\mathbf{U}_a$

$$\mathbf{U} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \end{pmatrix}.$$

Next I have to split the learned distribution parameters $\mu$ and $\mathbf{\Sigma}$ as well. The mean $\mu$ is easily split into the corresponding dimensions of the sample data

$$\mu_k = \begin{pmatrix} \mu_{A,k} & \mu_{B,k} \end{pmatrix}.$$

When splitting the covariance matrix I not only get the covariance matrices of the input and output data, but also the cross-covariance between input and output

$$\mathbf{\Sigma}_k = \begin{pmatrix} \mathbf{\Sigma}_{AA,k} & \mathbf{\Sigma}_{AB,k} \\ \mathbf{\Sigma}_{BA,k} & \mathbf{\Sigma}_{BB,k} \end{pmatrix}.$$

To calculate the output I am interested in the output probability conditioned on the input $p(\mathbf{a}_n|\mathbf{b}_n)$. This probability is composed of the probability of a cluster being active and the probability of the cluster producing the output

$$p(\mathbf{a}_n|\mathbf{b}_n) = \sum_{k=1}^{K} p(\mathbf{a}_n|\mathbf{b}_n, \mathbf{z}_n = \mathbf{c}_k) p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n).$$

Let's have a look at the probability of the cluster producing the output $p(\mathbf{a}_n|\mathbf{b}_n, \mathbf{z}_n = \mathbf{c}_k)$ first. It can be calculated by the conditional Gaussian distribution as in [4]

$$p(\mathbf{a}_n|\mathbf{b}_n, \mathbf{z}_n = \mathbf{c}_k) = \mathcal{N}(\mathbf{a}_n|\mu_{A|B,k}, \mathbf{\Sigma}_{A|B,k}).$$

The covariance matrix of the conditional distribution is calculated from the parts of the original covariance matrix

$$\mathbf{\Sigma}_{A|B,k} = \mathbf{\Sigma}_{AB,k} \mathbf{\Sigma}_{BB,k}^{-1} \mathbf{\Sigma}_{BA,k}.$$

The mean is a linear function of the input parameter $\mathbf{z}_b$

$$\mu_{A|B,k} = \mu_{A,k} + \left(\mathbf{\Sigma}_{AB,k}\mathbf{\Sigma}_{BB,k}^{-1}(\mathbf{b}_n - \mu_{B,k})^T\right)^T.$$

Therefore the conditional distribution of one cluster takes the form of a linear regression with a Gaussian noise model.

Now let's look at the probability of the cluster being active $p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n)$ .

$$p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n) = \frac{p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k)}{p(\mathbf{b}_n)} = \frac{p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k)}{\sum_{\tilde{k}=1}^{K} p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p(\mathbf{z}_n = \mathbf{c}_{\tilde{k}})}.$$

The probability $p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_k)$ is a marginal distribution of $p(\mathbf{b}_n, \mathbf{a}_n|\mathbf{z}_n = \mathbf{c}_k)$. It can be calculated as shown in [4] by

$$p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_k) = \mathcal{N}(\mathbf{b}_n|\mu_{B,k}, \mathbf{\Sigma}_{BB,k}).$$

The posterior is derived using Bayes's theorem

$$p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n) = \frac{p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k)}{\sum_{\tilde{k}=1}^{K} p(\mathbf{b}_n|\mathbf{z}_n = \mathbf{c}_{\tilde{k}})p(\mathbf{z}_n = \mathbf{c}_{\tilde{k}})} = \frac{\mathcal{N}(\mathbf{b}_n|\mu_{B,k}, \mathbf{\Sigma}_{BB,k})\pi_k}{\sum_{\tilde{k}=1}^{K} \mathcal{N}(\mathbf{b}_n|\mu_{B,\tilde{k}}, \mathbf{\Sigma}_{BB,\tilde{k}})\pi_{\tilde{k}}}.$$

Finally the conditional distribution I am interested in reads

$$p(\mathbf{a}_n|\mathbf{b}_n) = \sum_{k=1}^{K} p(\mathbf{a}_n|\mathbf{b}_n, \mathbf{z}_n = \mathbf{c}_k)p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n),$$

$$p(\mathbf{a}_n|\mathbf{b}_n) = \sum_{k=1}^{K} \frac{\mathcal{N}(\mathbf{a}_n|\mu_{A|B,k}, \mathbf{\Sigma}_{A|B,k})\mathcal{N}(\mathbf{b}_n|\mu_{B,k}, \mathbf{\Sigma}_{BB,k})\pi_k}{\sum_{\tilde{k}=1}^{K} \mathcal{N}(\mathbf{b}_n|\mu_{B,\tilde{k}}, \mathbf{\Sigma}_{BB,\tilde{k}})\pi_{\tilde{k}}}.$$

Please note that inputs and outputs can be chosen arbitrary and no relearning is required for different input/output combinations. This is extremely convenient when both a forward and an inverse kinematic have to be learned. A simulation using such a conditioned distribution is shown in Figure 4.1.

## 4.4 Comparison to the previous model

The conditioned model can be interpreted as a mixture of experts model, with $p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n)$ representing the gates and $p(\mathbf{z}_n = \mathbf{c}_k|\mathbf{b}_n)$ representing the experts. This gives a result very similar to the previous model. Table 4.1 shows a comparison between GLR and CMG using $\mathbf{X}$ as conditioned input $\mathbf{B}$.

| | GLR | CMG |
|---|---|---|
| $p_{\text{gate}}(\mathbf{z}_n = \mathbf{c}_k \mid \mathbf{x}_n) :$ | $\frac{\frac{1}{K}\mathcal{N}(\mathbf{x}_n \mid \mu_k, \mathbf{\Sigma}_k)}{\sum_{\tilde{k}=1}^{K} \frac{1}{K}\mathcal{N}(\mathbf{x}_n \mid \mu_{\tilde{k}}, \mathbf{\Sigma}_{ildek})}$ | $\frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_{X,k}, \mathbf{\Sigma}_{XX,k})}{\sum_{\tilde{k}=1}^{K} \pi_{\tilde{k}} \mathcal{N}(\mathbf{x}_n \mid \mu_{X,\tilde{k}}, \mathbf{\Sigma}_{XX,\tilde{k}})}$ |
| $p_{\text{expert}}(\mathbf{y}_n \mid \mathbf{z}_n = \mathbf{c}_k, \mathbf{x}_n) :$ | $\mathcal{N}(\mathbf{y}_n \mid \mathbf{x}_n \mathbf{W}_k, \text{diag}(\mathbf{s}_k))$ | $\mathcal{N}(\mathbf{x}_n \mid \mu_{Y\mid X,k}, \mathbf{\Sigma}_{Y\mid X,k})$ |

Table 4.1: Comparison between GLR and CMG using $\mathbf{X}$ as conditioned input $\mathbf{B}$

The difference between the two models are only the additional mixing parameter $\pi_k$ that has to be learned, and that the covariance matrix $\mathbf{\Sigma}_{XX,k}$ may have a non-diagonal form compared to $\text{diag}(\mathbf{s}_k)$ . This shows that the advantage of being able to determine input and output dimensions after learning comes with the disadvantage of additional model parameters.

# Chapter 5

# Hierarchical Mixtures of Experts

The HME model [3] consists of a sigmoid gating network and multiple experts performing linear regression. Figure 5.1 shows the gating network (gray boxes) and the experts for linear regression with a Gaussian noise model (red boxes). The input $\mathbf{X}$ is used for all gating nodes or *gates*. These gates are organized in form of a tree. A sigmoid function on $\mathbf{X}$ determines the probability of the gate to choose the left or right subtree. All leaves of the tree have to be experts. The single expert chosen by the gating network produces the output $\mathbf{Y}$.

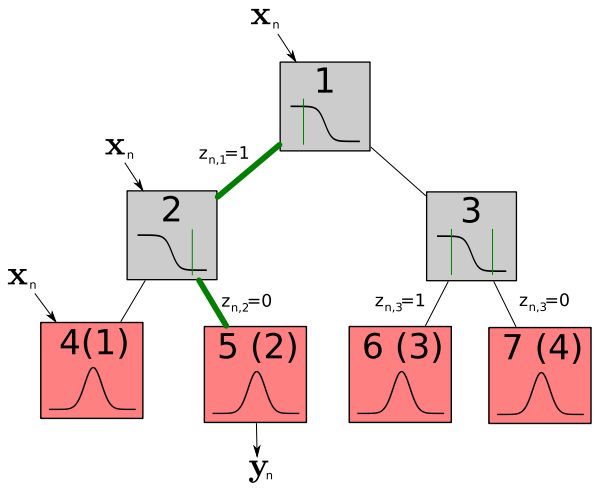



Figure 5.1: Illustration of Hierarchical Mixtures of Experts in form of a binary tree.

## 5.1 Model

The model is given by an input-dependant mixing coefficient and a regression model

$$p(\mathbf{y}_n|\mathbf{x}_n) = \sum_k p(\mathbf{z}_n \in \mathbf{P}_k|\mathbf{x}_n)p(\mathbf{y}_n|\mathbf{z}_n \in \mathbf{P}_k, \mathbf{x}_n).$$

I introduce the binary latent variable $\mathbf{Z} \in \{0,1\}^{G\times N}$ . For all samples $n$ the vector $\mathbf{z}_n$ contains the state of all $G$ gates. A value of one means that the left path is chosen, and a value of zero activates the right path. Following the active path from the root of the tree to its leaves, one arrives at the active expert $k$ as seen in Figure 5.1. The set $\mathbf{P}_k$ contains all possible combinations for $\mathbf{z}_n$ that result in an active expert $k$. For my example of a binary tree with depth two, the set $\mathbf{P}_2$ contains all possible paths to expert two (green in figure 5.1), expressed by the values of $\mathbf{z}_n$ ,

$$\mathbf{P}_2 = \{(1,0,0),(1,0,1)\}\,.$$

I use the notation

$$\mathbf{z}_n \in \mathbf{P}_k$$

to state the fact that the path to the expert $k$ is active, i.e. that expert is responsible for creating the output. I specify the linear regression model for an expert as

$$p(\mathbf{y}_n|\mathbf{z}_n \in \mathbf{P}_k, \mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_k, \mathrm{diag}(\mathbf{s}_k)).$$

The mixing coefficient is determined by the gating network. For each gate I define its probability to chose a certain path $z_{n,g}$ as

$$p(z_{n,g}|\mathbf{x}_n) = \sigma(\mathbf{x}_n\mathbf{v}_g^T)^{z_{n,g}}[1-\sigma(\mathbf{x}_n\mathbf{v}_g^T)^{1-z_{n,g}}],$$

with

$$\sigma(a) = \frac{1}{1+\exp(-a)},$$

being the logsig function.

For easier notation I introduce the set $path(k)$ containing the indexes of all gates on the path from the root of the tree to the expert $k$. The function

$$\tilde{z_g}(k) = \begin{cases} 1 & \text{if } k \text{ is in the left sub-tree of g} \\ 0 & \text{otherwise} \end{cases}$$

is used to succinctly describe the states of all gates along the path to $k$ to activate that

path. Now I can calculate the mixing coefficient

$$\pi_k(\mathbf{x}_n) = p(\mathbf{z}_n \in \mathbf{P}_k|\mathbf{x}_n) = \prod_{s \in path(k)} p(z_{n,g} = \tilde{z_g}(k)|\mathbf{x}_n).$$

## 5.2 Learning

I want to learn the model parameters

$$\theta = \left\{ \begin{array}{lcl} \mathbf{V} & \in & \mathbb{R}^{K \times D_x} \\ \mathbf{S} & \in & \mathbb{R}^{K \times D_y} \\ \mathbf{W} & \in & \mathbb{R}^{K \times D_x \times D_y} \end{array} \right\}.$$

Unfortunately there exists no closed form solution for this problem. However I can use an iterative approach such as the EM algorithm. There are also alternative ways to solve this problem like variational inference or MCMC. The EM algorithm consists of the following two steps.

### 5.2.1 E-Step

The log-likelihood reads

$$L = \sum_{n=1}^{N} \ln(p(\mathbf{y}_n|\mathbf{x}_n, \theta)),$$

and the responsibility reads

$$\begin{aligned} \gamma_{n,k} &= p(\mathbf{z}_n \in \mathbf{P}_k|\mathbf{x}_n, \mathbf{y}_n), \\ \gamma_{n,k} &= \frac{p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n \in \mathbf{P}_k)p(\mathbf{z}_n \in \mathbf{P}_k|\mathbf{x}_n)}{p(\mathbf{y}_n|\mathbf{x}_n)}, \\ \gamma_{n,k} &= = \frac{p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n \in \mathbf{P}_k)p(\mathbf{z}_n \in \mathbf{P}_k|\mathbf{x}_n)}{\sum_{\tilde{k}} p(\mathbf{y}_n|\mathbf{x}_n, \mathbf{z}_n \in \mathbf{P}_{\tilde{k}})p(\mathbf{z}_n \in \mathbf{P}_{\tilde{k}}|\mathbf{x}_n)}. \end{aligned}$$

For easier notation I also introduce the expectation for the number of samples generated by the cluster $k$

$$N_k = \sum_{n=1}^{N} \gamma_{n,k}.$$

$G$ denotes the total number of gates. I introduce a new index variable $\hat{g} \in \{1, \cdots, G+C\}$ referring to the gate $g = \hat{g}$ for values up to $G$ and referring to the expert $k = \hat{g} - G$ for values above. Therefore the gate's responsibility can be written as

$$\hat{\gamma}_{n,\hat{g}} = \left\{ \begin{array}{ll} \hat{\gamma}_{n,\text{left}(\hat{g})} + \hat{\gamma}_{n,\text{right}(\hat{g})} & \text{if } 1 \leq \hat{g} \leq G \\ \gamma_{n,\hat{g}-G} & \text{if } G < \hat{g} \leq G + C \end{array} \right.$$

With left($\hat{g}$) being the index of the left child of $\hat{g}$ and right($\hat{g}$) being the index of the child leave of $\hat{g}$. Indexes $1 \leq \hat{g} \leq G$ are reserved for gates and $G < \hat{g} \leq G + C$ are leaves. An example for a full binary tree with gate depth $u \in \mathbb{N}^+$ and one layer of $C = 2^u$ expert leaves result in the functions

$$\text{left}(\hat{g}) = 2\hat{g},$$

$$\text{right}(\hat{g}) = 2\hat{g} + 1,$$

with the total number of gates reading

$$G = C - 1.$$

In Figure 5.1 such a binary tree of gate depth two is shown. The grey squares represent the sigmoid gates. For a sample $n$ the gate $g$ is chosen to take the left path if $z_{n,g} = 1$, or the right path otherwise ($z_{n,g} = 0$). The red squares represent the Gaussian clusters or experts with the higher number being the index in the tree $\hat{g}$ and the lower number in brackets being the real index of the cluster $k = \hat{g} - G$. The green lines show the path to the currently active expert. This corresponds to the state $z_{n,1} = 1, z_{n,2} = 0$. The output is taken from expert two, meaning $\mathbf{y}_n = \mathcal{N}(\mathbf{y}_n|\mathbf{x}_n\mathbf{W}_2, \text{diag}(\mathbf{s}_2))$ or $\mathbb{E}\{\mathbf{y}_n\} = \mathbf{x}_n\mathbf{W}_2$.

### 5.2.2 M-Step

In this step I maximize the likelihood of the model while keeping the probabilities of the hidden variable $p(\mathbf{Z})$ constant.

#### Experts

Again the update rule of the expert takes the form of a weighted linear regression (compare with GLR)

$$\mathbf{W}_k = \sum_{n=1}^{N} (\mathbf{x}_n^T \gamma_{n,k} \mathbf{x}_n)^{-1} \mathbf{x}_n^T \mathbf{y}_n,$$

$$\mathbf{s}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{n,k} (\mathbf{y}_n - \mathbf{x}_n \mathbf{W}_k)^2.$$

#### Gates

The gate target $t_{n,g}$ reads

$$t_{n,g} = \frac{\hat{\gamma}_{n,\text{left}(s)}}{\hat{\gamma}_{n,g}}.$$

Gate learning is done with the iteratively reweighted least squares (IRLS) algorithm: The log-likelihood reads

$$p((t^T)_s|\mathbf{v}_g) = \prod_{n=1}^{N} (\hat{y}_{n,g})^{t_{n,g}} (1-\hat{y}_{n,g})^{1-t_{n,g}},$$

as in [4] with

$$\hat{y}_{n,g} = p(z_{n,g} = 1|\mathbf{x}_n) = \sigma(\mathbf{x}_n \mathbf{v}_g^T).$$

I define the error as the weighted negative log-likelihood. This results in the weighted cross-entropy error $E$

$$E(\mathbf{v}_g) = -\ln p((\mathbf{t}^T)_s|\mathbf{v}_n) = -\sum_{n=1}^{N} \hat{\gamma}_{n,g} \left\{ \hat{y}_{n,g} t_{n,g} + (1-\hat{y}_{n,g})(1-t_{n,g}) \right\},$$

As this function is concave, I am allowed to use the Newton-Raphson update by calculating the gradient

$$\nabla_{\mathbf{v}_g} E(\mathbf{v}_g) = \sum_{n=1}^{N} \hat{\gamma}_{n,g} (\hat{y}_{n,g} - t_{n,g}) \mathbf{x}_n^T,$$

and the Hessian matrix

$$\mathbf{H} = \nabla_{\mathbf{v}_g} \nabla_{\mathbf{v}_g} E(\mathbf{v}_g) = \hat{\gamma}_{n,g} \hat{y}_{n,g} (1-\hat{y}_{n,g}) \mathbf{x}_n \mathbf{x}_n^T,$$

and using them in a local quadratic approximation

$$\mathbf{v}_g^{(new)} = \mathbf{v}_g^{(old)} - \mathbf{H}^{-1} \nabla_{\mathbf{v}_g} E(\mathbf{v}_g).$$

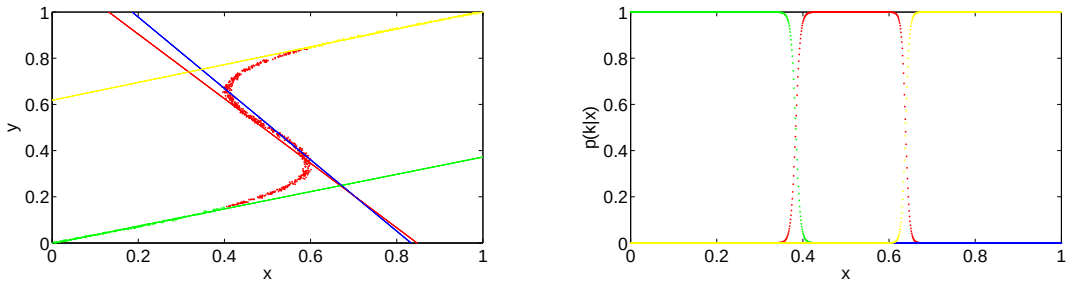
### 5.2.3 Regularization of gates

The HME model learns a gating network with high values for $\mathbf{v}_g$, viz it separates the data very sharply. To avoid this effect, as it is considered overfitting, I want to regularize the weights of the gates $\mathbf{v}_g$. Figure 5.2 shows the probability of an expert being active for the regularized gating network. Optimizing only the log-likelihood [4] of a gate

$$p((t^T)_s|\mathbf{v}_g) = \prod_{n=1}^{N} (\hat{y}_{n,g})^{t_{n,g}} (1-\hat{y}_{n,g})^{1-t_{n,g}},$$

results in overfitting. Therefore I define the error as the weighted negative log-likelihood plus the weighted norm of the weight vector $\mathbf{v}_g$. This will result in a trade-off solution, between the best possible solution and the solution with the smallest weights. This trade-off can be adjusted with the $\alpha$ parameter.

I introduce a vector $\tilde{\mathbf{v}}_g$ being equal to $\mathbf{v}_g$ for all elements, but the element corresponding to the bias term being zero. Thus I have no regularization on the bias weight.




(a) Training data and the trained experts are shown.

(b) Probability of an expert being active in simulation for given input.

Figure 5.2: HME with four experts (one inactive). The function is colored in the color of the expert $k$ corresponding to the data point.

The error function reads

$$E(\mathbf{v}_g) = \alpha\tilde{\mathbf{v}}_g\tilde{\mathbf{v}}_g^T - \ln p((t^T)_s|\mathbf{v}_g) = \alpha\tilde{\mathbf{v}}_g\tilde{\mathbf{v}}_g^T - \sum_{n=1}^{N}\hat{\gamma}_{n,g}\left\{\hat{y}_{n,g}t_{n,g} + (1-\hat{y}_{n,g})(1-t_{n,g})\right\}.$$

The optimization can be done in the same way as without the regularization using gradient descent. I calculate the gradient

$$\nabla_{\mathbf{v}_g}E(\mathbf{v}_g) = \alpha\tilde{\mathbf{v}}_s^T\sum_{n=1}^{N}\hat{\gamma}_{n,g}(\hat{y}_{n,g} - t_{n,g})\mathbf{x}_n^T,$$

and use the Hessian matrix

$$\mathbf{H} = \nabla_{\mathbf{v}_g}\nabla_{\mathbf{v}_g}E(\mathbf{v}_g) = \alpha\mathbf{I}_0 + \hat{\gamma}_{n,g}\hat{y}_{n,g}(1-\hat{y}_{n,g})\mathbf{x}_n\mathbf{x}_n^T,$$

as learn rate, resulting in the local quadratic approximation

$$v^{(new)} = v^{(old)} - \mathbf{H}^{-1}\nabla_{\mathbf{v}_g}E(\mathbf{v}_g).$$

### 5.2.4 Initialization

K-means is used recursively to split the data into multiple clusters. At the beginning the data gets separated into two clusters using K-means. These two clusters are used to train the first gate. Each of these two clusters is split again, using K-means with two clusters on that slice of the dataset. Those two new clusters are used to train the gates at depth two. These steps are repeated until the desired depth of the tree is reached. The final clusters are used to initialize the experts using standard linear regression for the weights.

The initialization of the gating network is essential for the model performance, as learning will only find a locally optimal solution. Different initialization techniques for HME are outside the scope of this work, but might significantly improve the overall model performance.

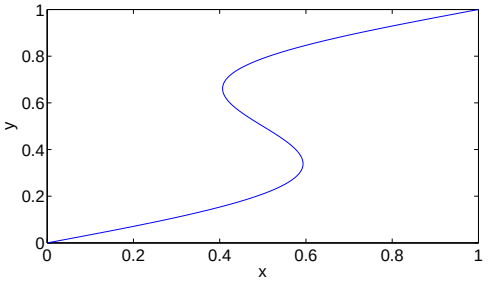
# Chapter 6

# Experiments

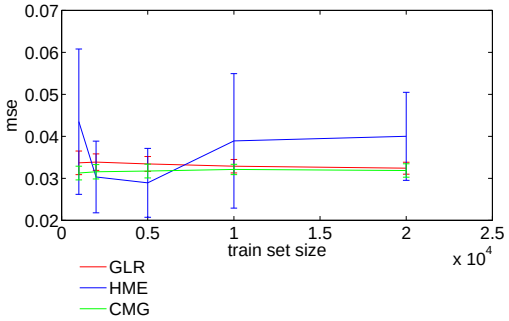
I evaluate three probabilistic methods for model learning on four tasks. In all tasks I contrast their performance. The tasks have different complexity. With each task I add more dimensions to the training data. In the third task I also check for robustness to noise and in the final task I test the models in combination with robotic movement planning.

## 6.1 Toy data set

This is one of the simplest possible tasks. It has only one input and one output dimension. I used this task to see the basic input/output behavior of the evaluated models. The one dimensional data gives us the possibility to have a look at the internal probabilities of the models using two dimensional plots.




(a) Toy dataset: The figure shows the task without noise.




(b) The figure compares the mean squared error on the toy task for the evaluated models. Simulations were done ten times. The bars represent the standard deviation on the results.

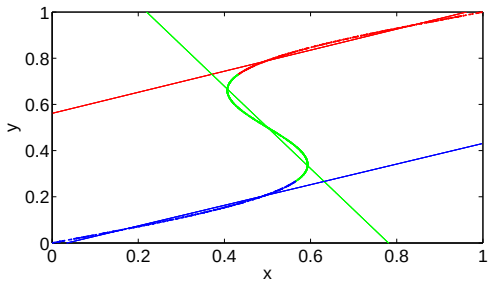
Figure 6.1: toy data task and model performance

The toy data set is given by

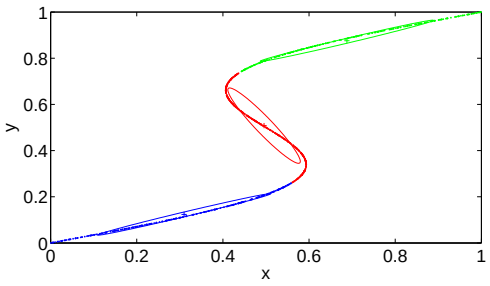
$$x = y + 0.3\sin(2\pi y) + noise$$

as in [3]. A minimal Gaussian noise with standard deviation 0.001 is used. Figure 6.1(b) shows the mean squared error (MSE). Figure 6.1(a) shows the data set. Please note that there is no unique solution for $y(x)$, which means that such a solution for the problem cannot be learned. However, it is possible to learn the probability function $p(y|x)$ which can account for the fact that one value of $x$ might lead to one of multiple possible values of $y$. This means $p(y|x)$ is a multi modal distribution ( i.e. for $x = 0.5$ ).

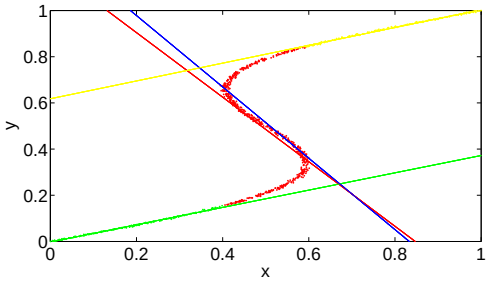
Using standard linear regression would lead to a solution like $x = y$. That solution would show a MSE of 0.09 on the used test set.




(a) Gated Linear Regression with Gaussian Noise Model



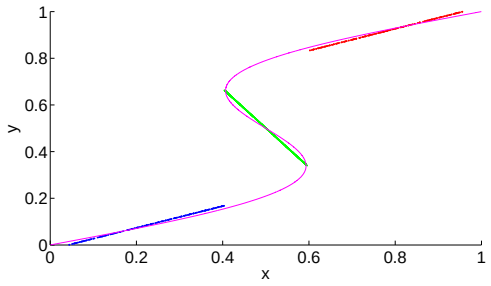

(b) Conditioned Mixture of Gaussian



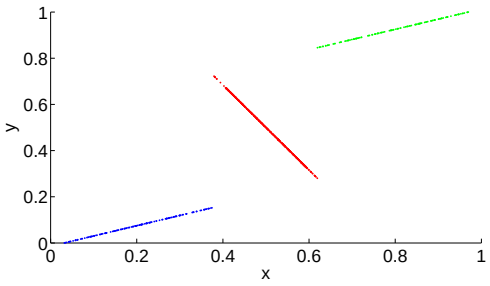

(c) Hierarchical Mixtures of Experts

Figure 6.2: This figure shows the training data and a representation of the models' experts. The training data is colored in the color of the expert $k$ with the highest probability $p(\mathbf{z} = \mathbf{c}_k|x)$ being responsible for producing the data point. As the experts perform a weighted linear repression, their output is mainly dependent on the training data of same color. The straight lines show the linear regressions representing the experts. For the CMG the Gaussian mixture before conditioning is shown. After conditioning this will also result in a linear regression in the direction of the covariance.

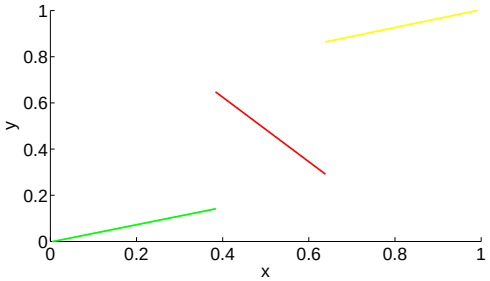
I tried to learn the toy dataset with the all three models. For this dataset I decided to use only three experts. Figure 6.2 shows how the training data is split into three clusters. Each cluster is represented by a different color. Please note that I used an HME with four clusters, because I only implemented full binary trees for gating. However, Figure 6.4 shows that one cluster of the HME becomes inactive, so I am allowed to compare it with the other three cluster simulations. The straight lines show the linear regression function for a cluster's data. The CMG is shown before conditioning. So the covariance matrix is visible. After conditioning, the CMG will also perform a linear regression (compare with simulation results shown in Figure 6.3).




(a) Gated Linear Regression with Gaussian Noise Model




(b) Conditioned Mixture of Gaussian

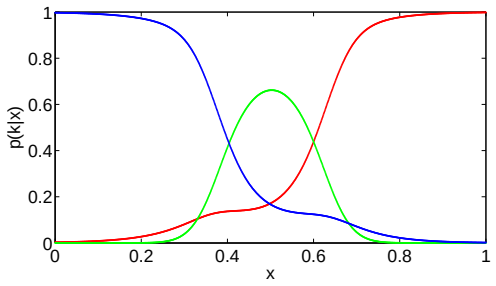



(c) Hierarchical Mixtures of Experts

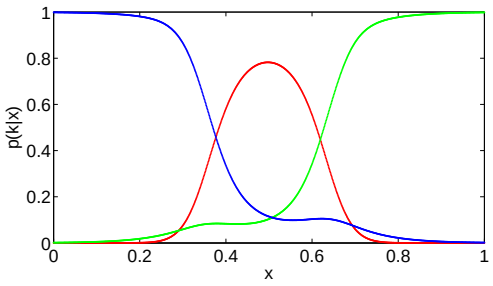
Figure 6.3: The output of a simulation using the trained models is shown. $x \in [0, 1]$ is given as input to the models and $y$ is the output of the models. The simulation was done using the most probable solution and therefore is fully deterministic. This means I chose the solution with the highest probability $p(y|x)$ for the output $y$. The results's color correspond to the expert which yielded the data point.

I generated different sizes of test and training sets and ran ten simulations. Figure 6.1(b) shows the mean squared error (MSE). The mean of the MSE and its standard deviation are plotted. As expected the MSE gets better and its standard deviation gets lower for higher number of training samples. The HME shows signs of overfitting as the MSE rises again for large training sets.

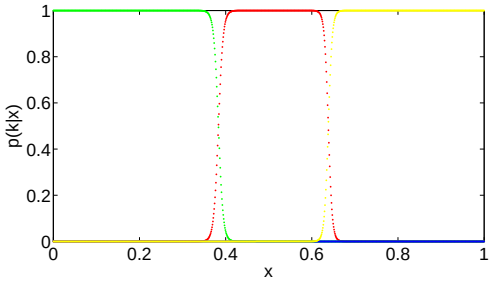
The calculation of the mean squared error is done by comparing the training and test data to the output of the simulated model. The simulation was done completely deterministic, i.e. I always used the solution with the highest probability. Figure 6.3 shows the output of the models with the highest probability.




(a) Gated Linear Regression with Gaussian Noise Model




(b) Conditioned Mixture of Gaussian




(c) Hierarchical Mixtures of Experts: The blue expert has a constant probability of zero, that is to say this model only uses three experts.

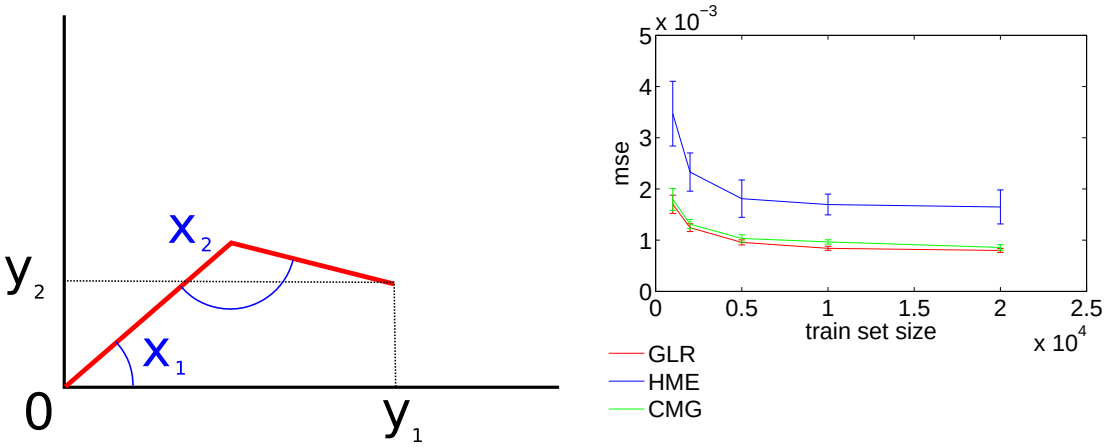
Figure 6.4: The probability of an expert being active is shown. The active expert produces the output of the simulation. The probabilities are colored in the color of the corresponding expert.

The output with highest probability was determined by choosing the most probable cluster and calculating the linear regression for the cluster. For simulations, I maximize the probability of the hidden variable $Z$ dependant on the input $p(\mathbf{z}_n = \mathbf{c}_k|x_n)$ Figure 6.4 shows the probability $p(\mathbf{z}_n = \mathbf{c}_k|x_n)$ for all clusters $k$ of being active for an arbitrary input $\mathbf{x}_n$.

As you can see, the Conditioned Mixture of Gaussian acts very similar to the Gated Linear Regression with Gaussian Noise Model. When learning the same covariance and mixing coefficient both models would behave identically. However, the Conditioned Mixture of Gaussian has the important advantage that learning does not distinguish between input and output dimensions, offering more flexibility.

## 6.2 Two link arm

To evaluate the discussed methods on a multi-dimensional task I analyze their performance on a two link kinematic arm. Using a two dimensional input enables use to have a closer look on the input clustering of the models using two dimensional plots.




(a) The geometry of two dimensional arm with two links. This task uses the angles $\mathbf{x}$ as input and the Cartesian coordinates $\mathbf{y}$ as output.

(b) The figure compares the mean squared error on the two link arm task for the evaluated models. Simulations were done ten times. The bars represent the standard deviation on the results.

Figure 6.5: Two link arm task: geometry and model performance

The arm can only move in two dimensions, and I ignore the dynamics. I am interested in the angles between the links and the position of the tip of the arm. The vector $\mathbf{x}_s$ contains all the samples of the s-th angle and the vector $\mathbf{x}_d$ contains the samples for the tip's position in the d-th dimension. See Figure 6.5(a) for model details.

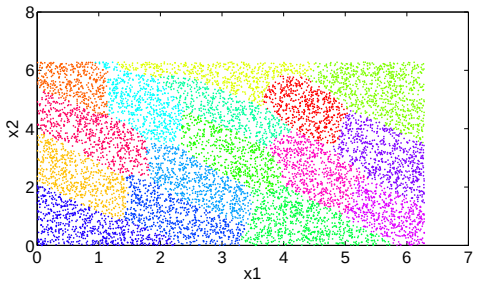
Also considering the length $l_n$ of the two link elements the position of the tip is given by the equations

$$\mathbf{y}_1 = l_1 \cdot \cos(\mathbf{x}_1) - l_2 \cdot \cos(\mathbf{x}_1 + \mathbf{x}_2),$$

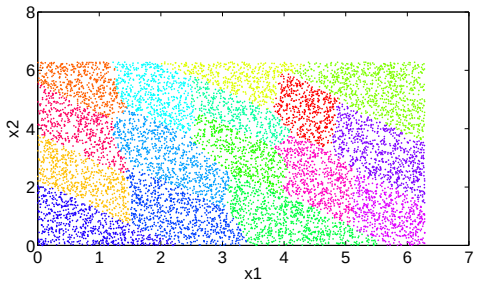
$$\mathbf{y}_2 = l_1 \cdot \sin(\mathbf{x}_1) - l_2 \cdot \sin(\mathbf{x}_1 + \mathbf{x}_2).$$

For the experiments I use identical lengths for all arm segments $l_1 = l_2 = 1$ . I learn the function $p(\mathbf{y}|\mathbf{x})$ .

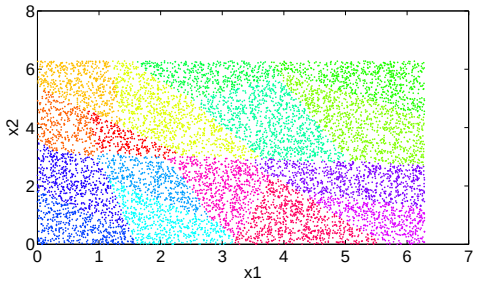
I evaluated all three models using 64 experts. Figure 6.5(b) shows the MSE for the test set. The GLR performs best. As expected the CMG shows a similar performance to the GLR. All three models' performance improves with the size of the training set. The improvement diminishes with larger training sets, and I don't expect a significantly reduced MSE for training sets larger than 20000 samples.




(a) Gated Linear Regression with Gaussian Noise Model




(b) Conditioned Mixture of Gaussian



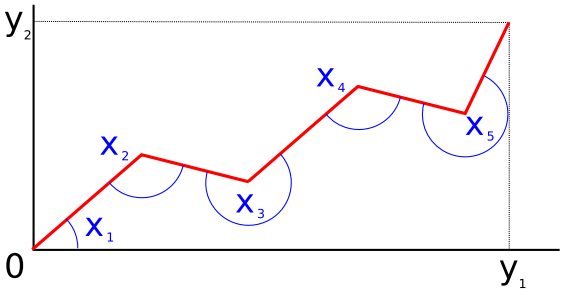

(c) Hierarchical Mixtures of Experts

Figure 6.6: The clustering of the input is shown. The input's color corresponds to the expert that it activates. This means presenting the input $\mathbf{x}$ leads to the expert $k$ being active and therefore creating the output $\mathbf{y}$. For this figure I only trained 16 experts to keep the plot more readable.

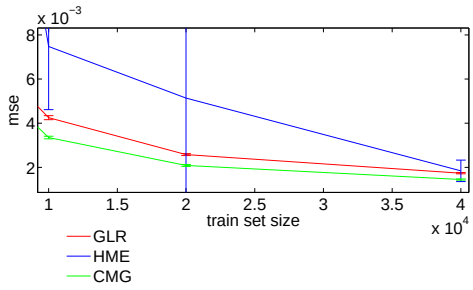
Figure 6.6 shows how training data gets fragmented into multiple clusters. For this plot I used models with only 16 experts to make the plots more readable. As the input is represented by multiple Gaussian clusters, the GLR and CMG tend to split the input space into rhombi. The HME uses the sigmoid function to split the input space, this separates the input space by a combination of straight lines (for higher dimensions hyperplanes). So it seems the HME should theoretically allow for a better split of the input space, and due to the similar experts the HME result in better overall performance. However, in the simulation it shows worse performance than the other approaches. This might be caused by poor initialization. Different initialization techniques might result in better performance, which is part of further studies.

## 6.3 Five link arm

This task is an extended version of the previous task. I add three more segments to the arm. In this task I also test for the influence of higher noise. A Gaussian shaped noise of 0.1 is added to **y**-dimension of the training set. Figure 6.7 shows a graphical representation of the model.




(a) Picture of a five link arm: The model consists of five links moving in a two dimensional pane.




(b) The plots show the MSE on the test set for the evaluated models. The simulation was done ten times. The bars represent the standard deviation of the results.

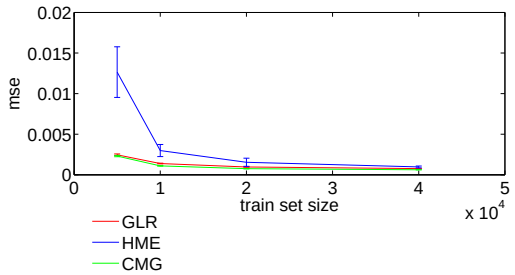
Figure 6.7: Five link arm: model and MSE

The Cartesian coordinates of the arm's tip can be calculated as sum over the projections of the arm's segments. A segment's projection can easily be determined by the cosine or sine function of its angle to the axis of the coordinate system. The angle is given by the sum of the angle of the previous segment and the angle between the segment and the previous segment (assuming all link lengths being one). Therefore the angle is the cumulated sum over all previous angles. Adding the projections of all segments leads to the position of the arm's tip

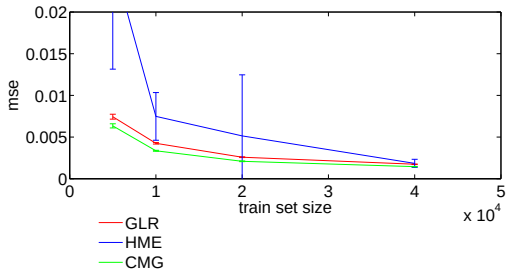
$$y_{1,n} = \sum_{i=1}^{5} \cos\left(\sum_{j=1}^{i} x_{j,n}\right) + \text{noise}, \quad y_{2,n} = \sum_{i=1}^{5} \sin\left(\sum_{j=1}^{i} x_{j,n}\right) + \text{noise}.$$

Figure 6.7(a) shows a graphical representation of the model. In Figure 6.7(b) I contrast the models' performance.

To investigate the robustness to noise I compare two scenarios. Figure 6.8(a) shows the MSE on the test set, where no noise was added. As expected the MSE gets lower with raising number of training samples. GLR and CMG show a similar performance. The HME reaches nearly the same performance for large training sets as the other two models. This is surprising, because the HME has less model parameters and therefore is expected to require less training data. In Figure 6.8(b) I applied Gaussian noise with a standard deviation of 0.1 . All models show worse performance compared to the noise-free task. With higher number of training data the influence of noise gets reduced. The HME suffers the most from the noise for small training sets.




(a) MSE without noise




(b) MSE with Gaussian noise using 0.1 as standard deviation

Figure 6.8: The plots show the MSE on the test set for the evaluated models. The simulation was done ten times. The bars represent the standard deviation of the results. Simulations without noise show better results. The influence of noise shrinks with rising number of training samples.

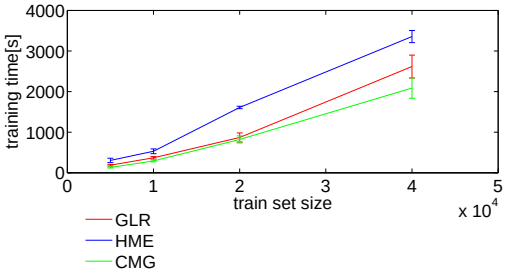
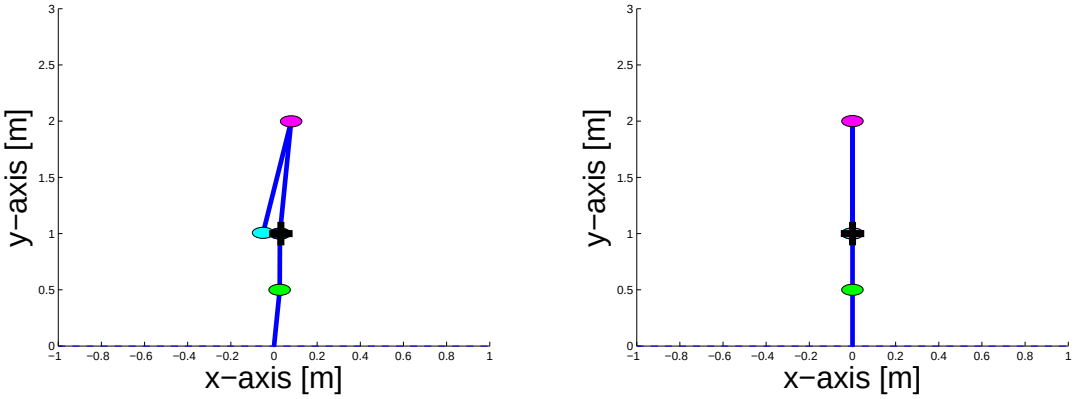



Figure 6.9: This plot shows the necessary time for training of the five link task. The simulation was done ten times. The bars represent the standard deviation of the results.

Figure 6.9 shows the computation time in seconds for training on my test computer (12x3.3Ghz, 94GB Ram). These values are highly dependent on the used hardware and implementation techniques. Implementation was done mainly by using the model formulas in a MATLAB simulation. The CMG is the fastest, as it has the simplest training algorithm. Everything is learned by a mixture of Gaussian. There is no extra gating network to train. Next comes the GLR with one gating layer. Finally, the HME has the highest training time, because of its multiple gating layers where each require a separate training step.

## 6.4 Four-Link Dynamic with Movement Planning

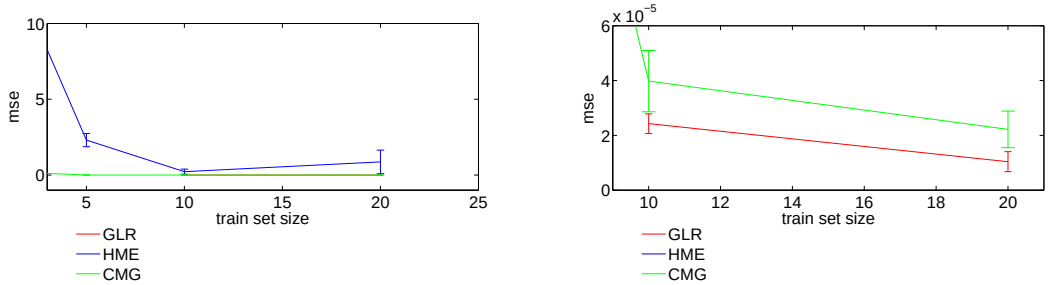



(a) Intermediate state while movement.

(b) Our target position is a upright stand with hand down.

Figure 6.10: In this experiment I use a model of a simplified humanoid robot. It consists of four link connected by knee, hip and shoulder. The robot is 2m tall and has a weight of 70kg. It can only move in a two dimensional pane. The Figures show the robot in two different positions.

In this experiment I use the models to approximate the dynamic model (state transition model, when applying an action) of a humanoid robot. The task consists of a four link robot that can move in a two dimensional plane. The angles represent knee, hip and shoulders of the humanoid robot. My goal is to reach the stable position as shown in Figure 6.10(b) starting from different initial states. As initial states I use the same upright position, but I apply a force to the robot's shoulder that tries to knock it over. The robot has to keep its balance and not to fall over.



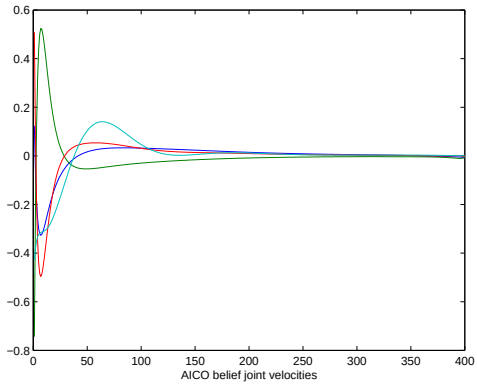

(a) MSE

(b) MSE zoomed

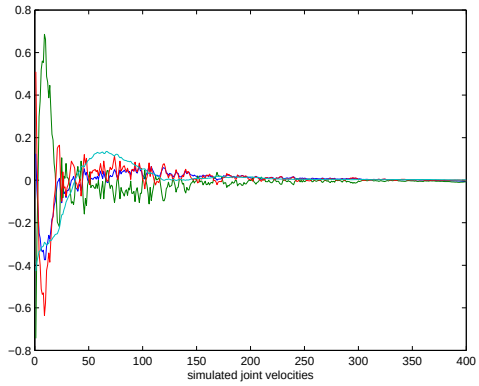
Figure 6.11: The plots show the MSE on the test set for the evaluated models. The simulation was done ten times. The bars represent the standard deviation of the results. The HME shows bad performance and overfitting for large training sets. The other two models perform well, with CMG and GLR showing similar performance.

To achieve this I use AICO [23, 17] for movement planning combined with the learned CMG model for the robot's dynamics. Figure 6.10(a) shows an intermediate position (after 200ms) that was reached while performing the movement planned by AICO. This task can also be considered a reverse pendulum task.

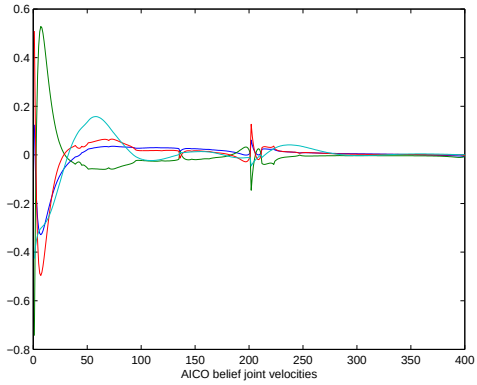
As I use a simulated robot, the analytic function of the dynamic is known. However, for a real robot this function might not be known. Thus learning it is necessary. I try to approximate the function with all three models by training them on data generated by the analytic function.



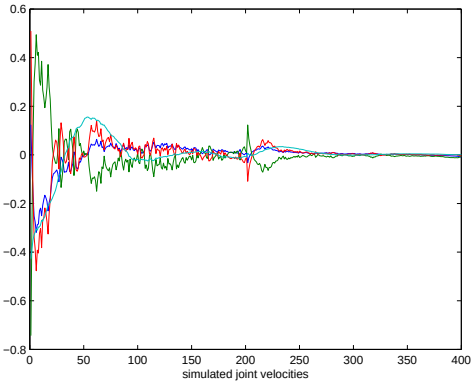

(a) AICO belief joint velocities with analytic model




(b) Simulated joint velocities with analytic model




(c) AICO belief joint velocities with CMG



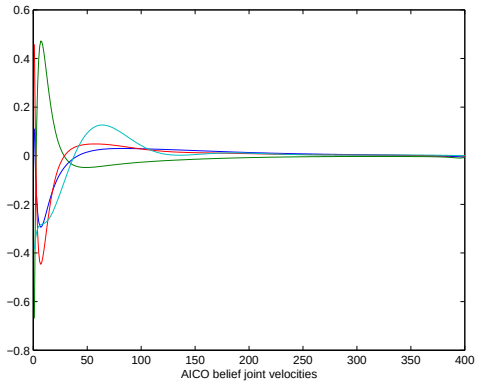

(d) Simulated joint velocities with CMG

Figure 6.12: Comparison of velocities using AICO with the analytic and CMG model: The robot starts with an upright stand and an impulse of 10Ns as an initial state. The trajectories show the link's velocities used to keep balance and end in an upright stand. The successful trajectories generated by AICO using the analytic model were used as training data for the CMG. Although the trajectories using the analytic model and the learned CMG differ, the number of time steps necessary to solve the task is similar.

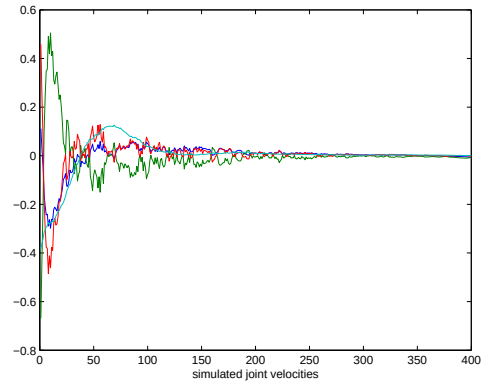
In this experiment a trajectory set is a list of $400 \times 3 = 1200$ samples [1]. Three

[1] Training samples are tuples of a current state, an action and the next state when applying that action.

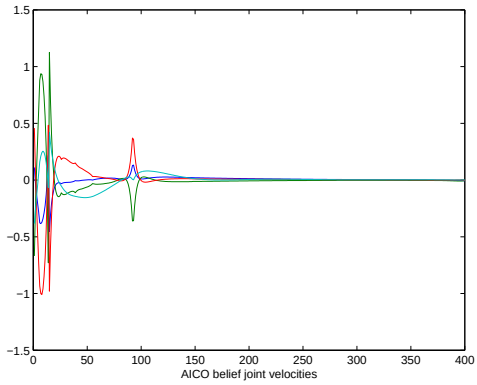
trajectories for the impules $F = \{8, 10, 12\}Ns$ with 400 discrete steps a generated using the analytic model with added noise. Figure 6.11(a) shows the MSE on a test set of the evaluated models. To better compare the performance of GLR and CMG, a zoomed version is shown in Figure 6.11(b). As one can see the HME has a far worse performance compared to the other two models. The GLR and CMG show similar performance. Both models are mathematically similar, so it is to be expected for them to show nearly identical performance on the data. The HME is expected to need less training data, because it has less model parameters. However, it performs worse. The reason probably lies in the difficult initialization of the HME that is relevant for model performance.



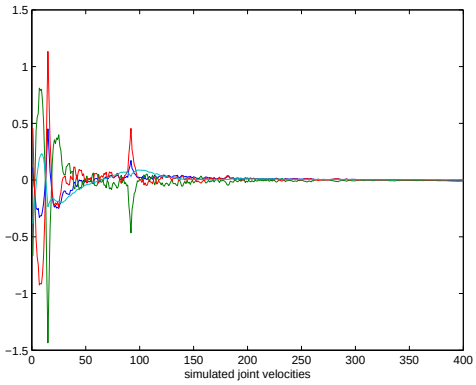

(a) AICO belief joint velocities with analytic model




(b) Simulated joint velocities with analytic model




(c) AICO belief joint velocities with CMG

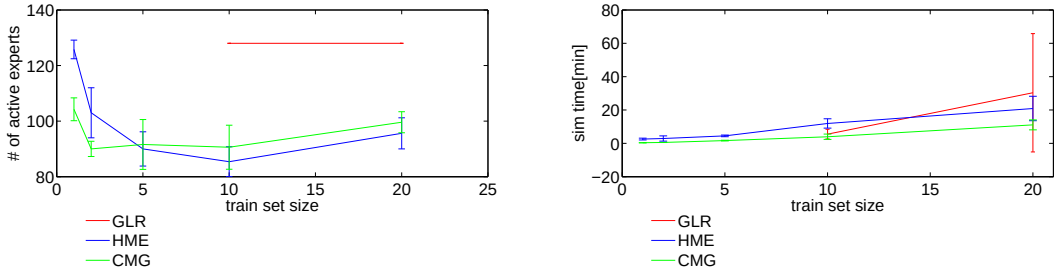



(d) Simulated joint velocities with CMG

Figure 6.13: Comparison of velocities using AICO with the analytic and CMG model on an unseen trajectory with initial impulse of 9Ns: The robot starts with an upright stand and an impulse of 9Ns as an initial state. The trajectories show the link's velocities used to keep balance and end in an upright stand. The successful trajectories generated by AICO using the analytic model with initial forces of 8,10 and 12Ns were used as training data for the CMG. Although the trajectories using the analytic model and the learned CMG differ, the number of time steps necessary to solve the task is similar.

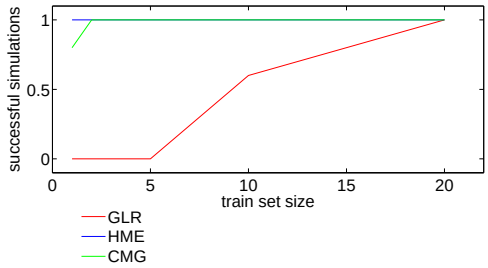
Figure 6.12 shows the necessary velocities of the robot links. Here I compare two runs

of AICO. First I test AICO with the known analytic function. Figure 6.12(a) shows the links' velocities believed to keep balance by AICO using the analytic model. As AICO internally works with a linear approximation to model dynamics, the simulation results using the true analytic dynamic function differs from AICO's belief in Figure 6.12(b). The velocity profiles have a complex curved form indicating that this is a hard task to master. I repeat this experiment using the learned dynamics function as given by the CMG model. Although the believe shown in Figure 6.12(c) and the actual movement in Figure 6.12(d) differ from the analytic example, the CMG model is able to solve the task within approximately the same number of simulation time steps. Thus the robot manages to keep balance. In the example shown in Figure 6.12 I trained the CMG with successful movement trajectories generated by AICO using the analytic model. To generate the training data I used impulses of 8, 10 and 12Ns as initial states. The model was tested on an impulse of 10Ns. Now I test the outcome of using a unseen impulse of 9Ns on the robot with the same model. Figure 6.13 shows that the CMG can predict the unseen trajectory with sufficient small error.




(a) This plot contrasts the number of active clusters or experts for the evaluated models. Empty clusters get dropped during training. All models use most of their experts.

(b) This plot shows the needed computational time for learning the model using the training set.




(c) This plot shows the ratio of mathematically successful simulations, i.e. where I did not run into numerical problems while training the models.

Figure 6.14: four link training statistics

In Figure 6.14(a) I illustrate the number of active experts. The computational time is contrasted in In Figure 6.14(b), which also scales linearly with the number of samples

as in the previous five link task. Figure 6.14(c) shows the ratio of successful simulations, i.e. where I did not run into numerical problems while training the models. Numerical operations failed due to covariance matrices being close to singular. These problems could most likely be solved by including additional regularization terms or adding prior distributions for model parameters. The HME is more robust because it has less covariance matrices. However I encountered problems with the gating network specializing on too few samples and clusters getting empty. I solved this by adding some regularization terms. With a similar approach it should be possible to increase the robustness of CMG and GLR, which is part of future research.

# Chapter 7

# Conclusions

I addressed the problem of learning highly nonlinear dynamic models of robot movement by using probabilistic regression methods. Three different models were implemented: the Gated Linear Regression with Gaussian Noise Model (GLR), the Conditioned Mixture of Gaussian (CMG) and the Hierarchical Mixtures of Experts (HME). I tested these models on multiple tasks, which ranged from a simple one dimensional problem to a simulated humanoid robot performing a reverse pendulum task. The CMG and GLR where able to solve all tasks. The HME always had the worst performance, being more sensitive to overfitting with an increasing number of training samples. However, as the HME has far less model parameters than the other two models, it remains very interesting for modeling high dimensional data. The question for optimal initializations of the HME leaves room for further research. I believe that the gap between HME and the other models could be closed by improving the initialization.

The CMG and GLR always showed very similar results, with each of them being slightly better than the other for half of the tasks. Thus I could not see a significant difference of performance of those two models. However the CMG has one big advantage over the GLR. It does not need to know which dimensions are input or output for the training process. The conditioning on one or multiple dimensions of the model happens after learning is done. Therefore the CMG can learn the dynamics and inverse dynamics model at the same time, where GLR would need an additional training cycle.

The CMG showes more robustness on small training sets. The GLR catches up with a rising number of training samples. Problems with robustness are mainly in cases of too small training sets, where the models are not capable of finding a good approximation of the dynamics function. The HME showed no robustness problems at all. However I invested much time to address numerical problems in the training process. I believe that the other two models' robustness could be improved in the same way. All of the tested models use most of their experts in solving the tasks. This means that only a small percentage of experts get inactive respectively only a few clusters get empty. Also the models show similar computational time on all the tasks.

Overall the CMG model is the mathematically simplest model. It is easy to use and shows good performance. The HME leaves the most room for improvements and further research and might be important for even higher dimensional problems.

In robotic tasks the models might benefit from the usage of a different feature vector $\phi(\mathbf{x})$, where I only investigated the identity, i.e. $\phi(\mathbf{x}) = \mathbf{x}$. For instance, adding a squared version of the data would lead to a second order regression. Finally the tested models not only generate a prediction for the output, but also a measure of uncertainty. Using this additional information in the planning algorithm might also be interesting for further work. In this manuscript I focused on batch learning, however it is straight forward to extend the used methods to online learning.

# Appendix A

# Abbreviations

**HME** Hierarchical Mixtures of Experts

**GLR** Gated Linear Regression with Gaussian Noise Model

**CMG** Conditioned Mixture of Gaussian

**MSE** mean squared error

**MCMC** Markov-Chain-Monte-Carlo

**EM** Expectation Maximisation

**E** Expectation

**M** Maximisation

**IRLS** iteratively reweighted least squares

**ILQR** Iterative Linear Quadratic Regulator

**LQG** Linear-Quadratic-Gaussian

**ILQG** Iterative Linear-Quadratic-Gaussian

**AICO** Approximate inference control

**SOC** stochastic optimal control

**PMP** Planning Movement Primitive

**MP** Movement Primitive

**LWPR** Locally Weighted Projection Regression

**LWR** Locally Weighted Regression

# Bibliography


[1] M. Ajallooeian, S. Pouya, A. Sproewitz, and A. Ijspeert. Central pattern generators augmented with virtual model control for quadruped rough terrain locomotion. 2013.

[2] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally Weighted Learning for Control. *Artificial Intelligence Review*, 11:75–113, 1997.

[3] C. M. Bishop. Bayesian Hierarchical Mixture of Experts. In *Uncwertainty in Artificial Intelligence: Proceesings of the Nineteenth Conference*, 2003.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning* . Springer Science + Business Media LCC, 2006.

[5] M. Botvinick and M. Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 2012.

[6] A. d'Avella, P. Saltiel, and E. Bizzi. Combinations of Muscle Synergies in the Construction of a Natural Motor Behavior. *Nature*, 6(3):300–308, March 2003.

[7] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[8] S. Kotosaka and S. Schaal. synchronized robot drumming by neural oscillator. (1):116–123, 2001.

[9] P. Kulchenko and E. Todorov. First-exit model predictive control of fast discontinuous dynamics: Application to ball bouncing. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2144–2151. IEEE, 2011.

[10] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the 1st International Conference on Informatics in Control, Automation and Robotics*, (ICINCO 2004), pages 222–229, Setúbal, Portugal, 2004.

[11] F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (IROS 2011), pages 3407–3412, San Francisco, CA, USA, 2011.

[12] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[13] G. Neumann, W. Maass, and J. Peters. Learning Complex Motions by Sequencing Simpler Motion Templates. In *Proceedings of the 26th International Conference on Machine Learning*, (ICML 2009), pages 753–760, Montreal, Canada, 2009.

[14] D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340, 2011.

[15] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf. Learning Inverse Dynamics: A Comparison. In *16th European Symposium on Artificial Neural Networks*, (ESANN 2008), pages 13–18, Bruges, Belgium, 2008.

[16] D. Nguyen-Tuong, M. Seeger, and J. Peters. Local Gaussian Process Regression for Real Time Online Model Learning and Control. In *Proceedings of 22nd Annual Conference on Neural Information Processing Systems*, (NIPS 2008), pages 1193–1200, Vancouver, BC, Canada, 2008.

[17] E. Rückert, G. Neumann, M. Toussaint, and W. Maass. Learned graphical Models for probabilistic Planning provide a new Class of Movement Primitives. In .

[18] E. A. Rückert, G. Neumann, M. Toussaint, and W. Maass. Learned graphical models for probabilistic planning provide a new class of movement primitives. *Frontiers in Computational Neuroscience*, 6(97), 2013.

[19] S. Schaal, J. Peters, J. Nakanishi, and A. J. Ijspeert. Learning Movement Primitives. In *International Symposium on Robotics Research*, (ISRR 2003), pages 561–572, Lucerne, Switzerland, 2003.

[20] A. Solway and M. M. Botvinick. Goal-directed decision making as probabilistic inference: a computational framework and potential neural correlates. *Psychological review*, 119(1):120, 2012.

[21] E. Theodorou, Y. Tassa, and E. Todorov. Stochastic Differential Dynamic Programming. In *Proceedings of the 29th American Control Conference*, (ACC 2010), Baltimore, Maryland, USA, 2010.

[22] E. Todorov and W. Li. A generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems. In *Proceedings of the 24th American Control Conference*, volume 1 of *(ACC 2005)*, pages 300 – 306, Portland, Oregon, USA, 2005.

[23] M. Toussaint. Robot Trajectory Optimization using Approximate Inference. In *Proceedings of the 26th International Conference on Machine Learning*, (ICML 2009), pages 1049–1056, Montreal, Canada, 2009.

[24] S. Vijayakumar, A. D'Souza, and S. Schaal. Incremental Online Learning in High Dimensions. *Neural Computation*, 17(12):2602–2634, dec 2005.