Non-linear regression algorithms for motor skill acquisition: a comparison

Thibaut Munzer, Freek Stulp, Olivier Sigaud

Non-linear regression algorithms for motor skill acquisition: a comparison

Thibaut Munzer[1], Freek Stulp[1,2] and Olivier Sigaud[3]

[1] FLOWERS Research Team, INRIA Bordeaux Sud-Ouest
351, Cours de la Libération 33405 Talence. thibaut.munzer@inria.fr

[2] Robotics and Computer Vision, École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech),
32, Boulevard Victor 75015 Paris freek.stulp@ensta-paristech.fr

[3] Université Pierre et Marie Curie, Institut des Systèmes Intelligents et de Robotique - CNRS UMR 7222
Pyramide Tour 55 - Boîte Courrier 173, 4 Place Jussieu, 75252 Paris CEDEX 5, France
olivier.sigaud@isir.upmc.fr

Abstract : Endowing robots with the capability to learn is an important goal for the robotics research community. One part of this research is focused on learning skills, where usually two learning paradigms are used sequentially. First, a robot learns a motor primitive by demonstration (or imitation). Then, it improves this motor primitive with respect to some externally defined criterion. In this paper, we study how the representation used in the demonstration learning step can influence the performance of the policy improvement step. We provide a conceptual survey of different demonstration learning algorithms and perform an empirical comparison of their performance when combined with a subsequent policy improvement step.

1 Introduction

Autonomous skill acquisition is a process endowing a robot with a capability to learn new tasks and to improve the corresponding behaviours through interactions with its environment. A common approach to skill learning is the following:

Representation Choose a parametric control policy representation for the skill, e.g. based on dynamical systems (Khansari-Zadeh and Billard, 2011) or Dynamical Movement Primitives (DMPs) (Ijspeert et al., 2013).

Imitation Initialize the policy from demonstration(s) with imitation learning, based on regression (Hersch et al., 2008). Regression approximates a *latent function* by setting the open parameters of the policy so that it convincingly reproduces the demonstrated trajectories.

Optimization Since imitation alone often does not suffice to solve the task (Kober and Peters, 2008), a further step is optimize the policy parameters with respect to a utility function with Reinforcement Learning (RL) (Kober and Peters, 2008; Theodorou et al., 2010) or Black-Box Optimization (BBO) (Rubinstein, 1999; Hansen et al., 2003).

So far, research on motor skill acquisition has focused on 1) developing and comparing policy representations that capture relevant aspects of the demonstration(s) during imitation, whilst also enabling generalization on the robot. 2) developing and comparing policy improvement algorithms for the optimization step. On the other hand, comparisons of regression methods have been more confined to learning mechanical models of robots (e.g. (Sigaud et al., 2011; Droniou et al., 2012)) or learning from demonstration *per se* (e.g. (Khansari-Zadeh and Billard, 2011)). Although these research lines have made progress mostly in isolation, in principle there should be a strong interdependency between the regression algorithm used during the imitation stage, the initial parametric policy representation obtained as a result, and the capability of an RL or BBO algorithm to improve the performance of this initial policy.

Intuitively, this interdependency is mainly characterized by two factors. First, and most obviously, the number of parameters used to represent the policy matters a great deal: more parameters to optimize result in a slower improvement, but too few parameters constrain the improvement capability. Second, whether the policy representation is local[1] or not should matter too, even if the impact of this property is more difficult to anticipate.

The main contribution of this paper consists in empirically investigating the above interdependency. Thus, we consider the effect of different latent function representations and meta-parameters setting within the entire *Representation-Imitation-Optimization* process. In particular, we compare the performance after optimization when using different regression algorithms for imitation learning. The second main contribution of this paper is to propose one general, unified latent function representation for a variety of regression algorithms (RBFNs, iRFRLS, LWR, LWPR, GMR).

The paper is organized as follows. In the next section, we describe the policy *Representation* used throughout this paper. In Section 3, we review the regression algorithms used for the regression in the *Imitation* step. The *Optimization* phase, i.e. policy improvement, is presented in Section 4. The experimental evaluation, including results and discussion, is presented in Section 5. A conclusion summarizes the achievements and suggests directions for future work.

2 *Representation*: Dynamical Movement Primitives

In RL, a policy π maps states to actions. An optimal policy π^* chooses the action that optimizes the cumulative discounted reward over time. In RL problems with continuous state and action spaces, a policy cannot be represented by enumerating all actions, so parametric policy representations $\pi_{\boldsymbol{\theta}}$ are required, where $\boldsymbol{\theta} \in \Theta$ is a vector of parameters. Thus, finding the optimal policy π^* corresponds to finding optimal policy parameters $\boldsymbol{\theta}^*$. This process is known as *direct policy search* or *policy improvement*. Because policy improvement algorithms are almost always local methods, they require a good initialization, i.e. $\boldsymbol{\theta}^{\text{init}} \approx \boldsymbol{\theta}^*$. This initialization thus corresponds to the *Imitation* phase, and the subsequent policy improvement to the *Optimization* phase.

For parameterized policy representations, one must distinguish between approaches where a feedback controller is represented directly as a parametric dynamical system (Khansari-Zadeh and Billard, 2011) and approaches based on Dynamical Movement Primitives (DMPs) (Ijspeert et al., 2002).

The latter representation, which we use in this paper, combines a non-parametric closed-loop spring-damper system that drives the system to the goal, and a parametric open-loop term which allows arbitrary (smooth) movements to be represented (Ijspeert et al., 2013).

A DMP is a learnable controller that combines a fixed spring-damper system with known convergence and stability properties (as well as generalization w.r.t. the goal) and a learnable part that can be optimized through trial-and-error. A DMP is represented by the following dynamical system:

$$\tau\ddot{x}(t) = \underbrace{\alpha(\beta(x(t) - g(t)) - \dot{x}(t))}_{\text{spring-damper system}} + \underbrace{h(t) f_{\boldsymbol{\theta}}(\phi(t))}_{\text{forcing term}}. \tag{1}$$

By (numerically) integrating this dynamical system over time, it generates a trajectory $[x(t), \dot{x}(t), \ddot{x}(t)]_{t=0}^{T}$. When several of such systems are coupled, the output trajectories can be used to represent, for instance, joint angles or end-effector positions of a robotic arm.

The symbols α and β are the spring-damper system coefficients, they are commonly set such that the system is critically damped ($\beta = \frac{\alpha}{4}$). The g system represents the goal of the spring-damper system, i.e. the value to which x will converge. The h function, called the gating system, must be close to zero at the end of the movement and monotonically tend to zero, so that, after some time, the system tends to a simple spring-damper system which is known to converge. In the original framework, the same system is used for the gating and the phase ($\phi = h$), so h is computed by integrating $\dot{h} = -\alpha_p h$ (Ijspeert et al., 2002).

In this paper, we use a slighly modified version of DMPs (Kulvicius et al., 2012), where:

$$\dot{\phi}(t) = \begin{cases} \frac{1}{T} & \text{if } t \leq T \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

[1] We say that a representation is local if the represented space is partitioned into a set of local models and if the output to a point in that space is computed mostly from the corresponding local model.

$$\dot{h}(t) = -\frac{\alpha_h e^{\frac{\alpha_h}{\Delta_t}(T-t)}}{(1+e^{\frac{\alpha_h}{\Delta_t}(T-t)})^2} \tag{3}$$

with, $h(0) = 1$ and $\phi(0) = 0$ (Δ_t is the number of time step). This modified representation leads to more regularized input/target data for $f_{\boldsymbol{\theta}}$ (Kulvicius et al., 2012).

The ϕ system, called the phase system, is used to decouple the produced movement from time. For all practical purposes, it may be considered as an alternative representation of time. Since this time signal is the only input to $f_{\boldsymbol{\theta}}(\phi(t))$, called the forcing term, is open-loop.

The $f_{\boldsymbol{\theta}}$ function deforms the spring-damper trajectory to allow the system to generate arbitrary (smooth) trajectories. If DMPs are used as a parameterized policies $\pi_{\boldsymbol{\theta}}$, their only open parameters $\boldsymbol{\theta}$ are the parameters of the latent function $f_{\boldsymbol{\theta}}$, so searching for a good policy $\pi_{\boldsymbol{\theta}}$ boils down to searching for a good latent function $f_{\boldsymbol{\theta}}$ by exploring the space Θ.

During the *Imitation* phase, the input/target data for the latent function is acquired by taking a given demonstration trajectory $[x(t), \dot{x}(t), \ddot{x}(t)]_{t=0}^{T}$, and solving (1) for f with regression:

$$f_{\boldsymbol{\theta}}(\phi(t)) = \frac{\ddot{x}(t) - \alpha(\beta(x(t) - g(t)) - \dot{x}(t))}{h(t)}. \tag{4}$$

Thus, the input data for the latent function $f_{\boldsymbol{\theta}}$ is the 1D phase signal $\phi(t)$ (acquired by integrating the phase system over time), and the target is the right-hand term in (4). Since both input and targets are continuous, approximating the latent function $f_{\boldsymbol{\theta}}$ corresponds to a regression. This concludes our discussion of the *Representation* of the parameterized policy. The next section presents several algorithms which can be used to perform the regression to acquire $\boldsymbol{\theta}$. Section 4 then shows how these parameters are then used as an initialization of the *Optimization* phase, whose goal it is to find the optimal policy parameters $\boldsymbol{\theta}^*$.

3 *Imitation*: Regression methods

Given a set of continuous input/output datapoints, the purpose of function approximation, also called regression, is to determine from a family of model functions a latent function $f_{\boldsymbol{\theta}}$ that best matches this set of points.

To define notations, we assume that the data consists of N examples. Each example i is composed of an input vector $\boldsymbol{x}_i$ of dimension m and an output scalar y_i. The collection of all input can be seen as a $m \times N$ matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N)$ and the collection of output as a vector $\boldsymbol{y} = (y_1, y_2, ..., y_N)$.

In this section, we describe two families of function approximation representations (Sigaud et al., 2011). In the first family, the function is represented as a weighted sum of basis functions, e.g. Radial Basis Function Networks (RBFNs) (Park and Sandberg, 1993) and iRFRLS (Gijsberts and Metta, 2011; Gijsberts and Metta, 2012). In the second, the function is represented as a Gaussian mixture of linear models, i.e. a weighted sum of local linear models where the weights depend on the input space through Gaussian basis functions, e.g. Locally Weighted Regression (LWR) (Atkeson and Schaal, 1995), Locally Weighted Projection Regression (LWPR) (Vijayakumar and Schaal, 2000) and Gaussian Mixture Regression (GMR) (Hersch et al., 2008; Calinon, 2009). We conclude that both the output model of both these families can be unified into a single more general latent function representation.

3.1 Least Square regression of the weights of basis functions

In the first family of regression methods, the latent function is represented as a weighted sum of basis functions. We first explain the basics about the Least Square and Regularized Least Square methods as well as their incremental variants and then explain how they are used to approximate through a weighted sum of basis functions.

3.1.1 Least Square and Regularized Least Square

In the case of linear regression, the family of model functions is linear, which means that the latent function f is such that $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$, where $\boldsymbol{w}$ is a vector of weights. A popular method for choosing $\boldsymbol{w}$ is the Least

Squares algorithm (LS). With this algorithm, the chosen $\boldsymbol{w}$ is the one that minimizes the squared errors, i.e.

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{w}\|^2. \tag{5}$$

Minimizing the Least Square errors is a continuously differentiable unconstrained optimization problem that can be solved analytically. Its solution is

$$\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{6}$$

However, potential singularities in $\boldsymbol{X}^T\boldsymbol{X}$ may make it difficult to invert and can result in very large weigths $\boldsymbol{w}$. A solution to this issue consists in including a penalization for large weigths in the optimization criterion, resulting in Regularized Least Square (RGLS) also called Ridge Regression (RR) (see e.g. (Schmidt, 2005) for a detailed presentation).

So (5) becomes

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{w}\|^2, \tag{7}$$

where λ is a trade-off parameter balancing between large weights and error. This minimization problem has an analytical solution

$$\boldsymbol{w}^* = (\lambda\boldsymbol{I} + \boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{8}$$

Computing $\boldsymbol{w}^*$ requires the inversion of the $(\lambda\boldsymbol{I} + \boldsymbol{X}^T\boldsymbol{X})$ matrix, which is in $O(N^3)$. This complexity can be reduced to $O(N^2)$ by using the Sherman-Morrisson formula, giving rise to an incremental update of the inverse, but this method is sensitive to rounding errors. A numerically more stable option consists in updating the Cholesky factor of the matrix using the QR algorithm (see (Gijsberts and Metta, 2012) for details).

3.1.2 Kernel Regularized Least Square

Least Square and RGLS methods can be used to approximate a linear function. When the latent function is non-linear, one approach consists in paving the input space with non-linear basis functions and representing the latent function as a weigthed sum of these basis functions. This transforms non-linear regression into linear regression in a different space, using the basis functions to project from one space to the other. The basis functions used in this approach are generally Radial Basis Functions (RBFs), called "kernels" in this context. These functions are such that their output is maximal for a given input and decreases with the distance to this input, which provides a locality property. Gaussian functions are the most often used kernels.

Kernel Regularized Least-Squares (KRGLS) is the extension of RGLS to non-linear functions using this approach, which is known as the kernel trick in classification (Saunders et al., 1998) and has been popularized in SVMs (Rifkin et al., 2003).

The standard method consists in adding for each datapoint x_i a kernel function $k(\boldsymbol{x}, \boldsymbol{x}_i)$. The latent function is then defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i.k(\boldsymbol{x}, \boldsymbol{x}_i). \tag{9}$$

Defining the $m \times N$ kernel matrix as $\boldsymbol{K} = [k(\boldsymbol{x}, \boldsymbol{x}_i)]$, the optimal vector of coefficients is given by

$$\boldsymbol{w}^* = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}. \tag{10}$$

Unfortunately, the size of this matrix increases with the number of samples, giving rise to the kernel expansion problem. Indeed, like RGLS, the cost of KRGLS is cubic in the number of samples. Here again, the complexity can be reduced using the QR algorithm.

Several additional methods can be used to reduce this complexity, giving rise to a suboptimal estimation of the latent function. One can limit the number of considered samples (Dekel et al., 2008; Engel et al., 2002), or replace the analytical computation of $\boldsymbol{w}^*$ by a gradient-like method. Finally, one can also replace the expanding set of kernel functions by a fixed set of such functions. This is the approach taken in the next sections.

3.1.3 Radial Basis Function Networks

A widely used algorithm in this context is Radial Basis Function Networks. It consists in using KRGLS with a fixed set of D radial basis functions. If Gaussians are used as basis functions, the d^{th} Gaussian function g_d is defined for any state $\boldsymbol{x}$ as

$$g_d(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_d)^T \boldsymbol{\Sigma_d^{-1}} (\boldsymbol{x}-\boldsymbol{x}_d)} \tag{11}$$

where $\boldsymbol{x}_d$ is the center of the Gaussian and the $\boldsymbol{\Sigma_d}$ covariance matrix defines an ellipsoid related to the variance of the Gaussian function in all dimensions.

The latent function is then computed as

$$f(\boldsymbol{x}) = \sum_{d=1}^{D} w_d . g_d(\boldsymbol{x}). \tag{12}$$

These functions can be either regularly placed to pave the input space or drawn randomly. The first option is the most often used, though the algorithm presented in the next section suggests that random placement might be more efficient.

3.1.4 IRFRLS

In (Rahimi and Recht, 2008), the authors propose to approximate the kernel matrix $\boldsymbol{K}$ with a set of random features, giving rise to *Random Features Regularized Least Squares* (RFRLS), an algorithm that converges to KRGLS in $\mathcal{O}\left(\frac{1}{\sqrt{D}}\right)$, where D is the number of features on which the kernel is projected. The computation time for prediction being linearly dependent on the number of training samples, random features are used to approximate the kernel and keep the prediction time constant.

In (Gijsberts and Metta, 2011), the authors present an incremental version of RFRLS (named iRFRLS hereafter) that introduces the incremental estimation method of RGLS into RFRLS. With respect to RBFNs, the key idea behind iRFRLS is that any function can be approximated arbitrarily well as a weighted sum of cosine functions, as outlined by the theory underlying the Fourier Transform. Thus, instead of using Gaussian kernels as in RBFNs, the latent function is approximated as a set of D cosine functions using

$$\widetilde{\boldsymbol{y}} = f(\boldsymbol{x}) = \sum_{d=1}^{D} w_d . cos(\boldsymbol{\omega}^T \boldsymbol{x} + \phi). \tag{13}$$

The basis functions are defined by randomly drawing their multidimensional period $1/\boldsymbol{\omega}$ and phase b using $\boldsymbol{\omega} \sim \mathcal{N}(0, 2\gamma I)$ and $\phi \sim \mathcal{U}(0, 2\pi)$.

As in RBFNs, the weights w_d of these basis functions are tuned so as to approximate the latent function f by the incremental version of the KRGLS algorithm described above.

The accuracy versus time complexity trade-off can be easily tuned by changing the number of features D. Furthermore, the complexity of this method is in $\mathcal{O}\left(D^2\right)$, thus it is independent of the number of samples. The other two meta-parameters of iRFRLS are λ, the regularization parameter and γ, the variance over the period of the random cosine features. Experiments on several databases confirm that RFRLS and iRFRLS can be set close to KRGLS with a small enough processing time. Indeed, iRFRLS shows interesting practical performances (Droniou et al., 2012) and its computation time is independent of the learning data.

3.1.5 Summary and Intermediate Discussion

The same regression algorithm, namely the incremental version of RLS, can be applied to the RBFN representation and to the iRFRLS representation. Thus these approaches only differ in the basis functions they use (local Gaussian kernels in RBFNs versus global cosine functions in iRFRLS) and in the way these basis functions are organized (regularly distributed in RBFNs versus drawn randomly in iRFRLS). As a consequence, any difference in performance between these algorithms can be explained by a difference in the corresponding representations. An open question is whether replacing the local kernels used in RBFNs by non-local basis functions in iRFRLS makes the global model harder to optimize or not.

3.2 Locally Weighted Regression methods

Locally Weighted Regression methods are a family of methods that all rely on the same model for function approximation. The way a function is approximated with such a model is explained in Figure 1.

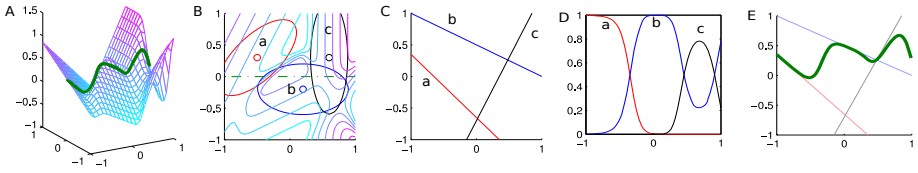


Figure 1: Function approximation in Locally Weighted Regression models. A: The latent function to be approximated. For the sake of clarity, we focus on approximating it in one dimension along the green line. B: A view from above, showing 3 basis functions (a, b and c) along the green line. C: the linear model along the green line corresponding to each basis function. D: The relative importance of each basis function along the green line. E: The model output is obtained by weighting each linear model with the relative importance of the corresponding basis function. It matches the green line shown in A.

More formally, each basis function d defines for any state $\boldsymbol{x}$ a weight $w_d(\boldsymbol{x})$ that is represented as a Gaussian function $g_d(\boldsymbol{x})$ as defined in (11).

A local linear model $\Psi_d(\boldsymbol{x}) = \boldsymbol{A}_d\boldsymbol{x} + b_d$ is associated to this basis function. The approximated output $\widetilde{\boldsymbol{y}}$ is computed as a weigthed sum of all linear models where the weight is given by the relative importance of the corresponding basis function, that is

$$\widetilde{\boldsymbol{y}} = \frac{\sum_{d=1}^{D} w_d(\boldsymbol{x})\Psi_d(\boldsymbol{x})}{\sum_{d=1}^{D} w_d(\boldsymbol{x})}. \tag{14}$$

This Gaussian mixture of linear models is used in diverse algorithms listed below.

3.2.1 The Locally Weighted Regression algorithm

The Locally Weighted Regression algorithm (LWR) (Atkeson and Schaal, 1995) is the ancestor of the whole family, but is still a competitive algorithm. In LWR, the number of basis functions, their position and variance is left to the user and is not adapted over training sessions. The linear models are adapted with an LS algorithm or its incremental variant, Recursive Least Square (RLS). In the case of RLS, the corresponding algorithm is called RFWR, see (Sigaud et al., 2011) for more information.

3.2.2 Locally Weighted Projection Regression

The Locally Weighted Projection Regression algorithm (LWPR) (Vijayakumar and Schaal, 2000) has been often used to learn mechanical models of robots (see (Sigaud et al., 2011) for a survey). LWPR is an improvement over LWR mainly in four ways: the algorithm is incremental; new basis functions are added automatically by the algorithm when needed; the shape of basis functions is adapted online; and low dimensional projection is realized before fitting the linear models (using an incremental version of Partial Least Squares called NIPALS (Geladi and Kowalski, 1986)). LWPR suffers from its many non intuitive meta-parameters. In this paper we only tune **w_gen**, which is responsible for adding new basis functions online and **init_d**, their initial shape.

3.2.3 Gaussian Mixture Regression

Gaussian Mixture Regression (GMR) is a batch method based on the unsupervised Expectation-Maximization (EM) (Dempster et al., 1977) algorithm. It learns a Gaussian Mixture Model (GMM) of the latent function f such that $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ where ϵ_i is a Gaussian noise. Given the set of input $\boldsymbol{x}_i$ and output y_i, the GMM builds a model of the density of the vectors $Z_i = [\boldsymbol{x}_i^T y_i]^T$ using a weigthed sum of D Gaussian functions

$$p(Z_i) = \sum_{d=1}^{D} \pi_d \mathcal{N}(Z_i; \mu_d, \Sigma_d), \text{ with } \sum_{d=1}^{D} \pi_d = 1. \tag{15}$$

This model is in the input×output space, as illustrated in Figure 2.

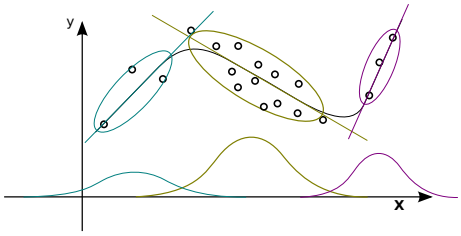


Figure 2: Gaussian Mixture Regression. Upper part: the density of datapoints is approximated as a Gaussian Mixture Model with Gaussian functions in the input × output space. In order to estimate the expectation of the output given an input, these Gaussian functions are projected in the input space, as shown in the lower part of the figure. The Gaussian Mixture Model can then be seen as performing a weighted combination of local linear models exactly as LWR methods do.

The EM algorithm adjusts the priors π_d and the parameters μ_d and Σ_d of the Gaussian functions that define this model (see (Ghahramani and Jordan, 1994) for details).

As noted in (Ghahramani and Jordan, 1994), the learned density can be exploited in several ways, since we can estimate $\widetilde{y} = f(\boldsymbol{x})$, $\widetilde{\boldsymbol{x}} = f^{-1}(y)$, or any other relation between two subsets of the elements of the concatenated vector $(\boldsymbol{x}^T y)$. As a Bayesian method, an advantage of GMR over other regression methods is that it can easily be used in an active learning framework. However, GMR is not an iterative algorithm, so it is less applicable in a robotic context where on-line learning is more appealing.

Here, we are interested in evaluating $\widetilde{y} = E(y|\boldsymbol{x})$, the expectation of y given $\boldsymbol{x}$. To do so, μ_d and Σ_d can be separated in input and output components using

$$\mu_d = [\mu_{d,X}^T, \mu_{d,Y}^T]^T \text{ and } \Sigma_d = \begin{pmatrix} \Sigma_{d,X} & \Sigma_{d,XY} \\ \Sigma_{d,YX} & \Sigma_{d,Y} \end{pmatrix}, \tag{16}$$

and then compute the expected output y given an input $\boldsymbol{x}$ using

$$\widetilde{y} = \sum_{d=1}^{D} h_d(\boldsymbol{x})(\mu_{d,Y} + \Sigma_{d,YX}\Sigma_{d,Y}^{-1}(\boldsymbol{x} - \mu_{d,X})), \tag{17}$$

with:

$$h_d(\boldsymbol{x}) = \frac{\pi_d \mathcal{N}(\boldsymbol{x}; \mu_{d,X}, \Sigma_{d,X})}{\sum_{l=0}^{D} \pi_l \mathcal{N}(\boldsymbol{x}; \mu_{l,X}, \Sigma_{l,X})}. \tag{18}$$

Furthermore, we can reformulate (17) as a Gaussian mixture of linear models

$$\widetilde{y} = \sum_{d=1}^{D} h_d(\boldsymbol{x})(\boldsymbol{A}_d \boldsymbol{x} + b_d), \tag{19}$$

with $\boldsymbol{A}_d = \Sigma_{d,YX}\Sigma_{d,Y}^{-1}$ and $b_d = \mu_{d,Y} - \Sigma_{d,YX}\Sigma_{d,Y}^{-1}\mu_{d,X}$.

By equating $h_d(\boldsymbol{x})$ to $\frac{w_d(\boldsymbol{x})}{\sum_{d=1}^{D} w_d(\boldsymbol{x})}$ in (14), one can recognize that this model is strictly equivalent to the one of LWR methods. The only meta-parameter of GMR is the number D of Gaussian features.

3.2.4 Summary and Intermediate Discussion

A Gaussian mixture of linear models is used in LWR, LWPR and GMR. Thus these algorithms only differ in the way they tune the basis functions and the linear models, and in the meta-parameters that are used for this tuning. At one extreme, in LWR, all the parameters of the basis functions are predetermined by the user and the local linear models are learned with a batch Least Square method. GMR is also a batch method whose only meta-parameter is the number D of basis functions. Being batch, it can perform accurate regression with fewer basis functions than incremental methods. At the other extreme, in LWPR the number of basis functions is adaptive but constrained through meta-parameters. Other algorithms like XCSF (Butz et al., 2004; Butz and Herbort, 2008) or ILO-GMR (Cederborg et al., 2010) are not treated in this paper, but the representation is essentially the same.

3.3 A unified representation for non-linear regression

In the previous section, we have presented two families of models: weighted combination of basis functions and Gaussian mixture of linear models. Actually, as outlined in Table 1, a weighted combination of Gaussian features is a particular case of Gaussian mixture of linear models where the linear models are all constant.

Algorithm	basis function	local model
LWR	$\frac{g_d(\boldsymbol{x})}{\sum_{k=1}^{D} g_k(\boldsymbol{x})}$	$\boldsymbol{A_d x} + \quad b_d$
LWPR	$\frac{g_d(\boldsymbol{x})}{\sum_{k=1}^{D} g_k(\boldsymbol{x})}$	$\boldsymbol{A_d x} + \quad b_d$
GMR	$\frac{g_d(\boldsymbol{x})}{\sum_{k=1}^{D} g_k(\boldsymbol{x})}$	$\boldsymbol{A_d x} + \quad b_d$
RBFN	$g_d(\boldsymbol{x})$	$b_d \ (= w_d)$
iRFRLS	$\cos(\boldsymbol{\omega}^T \boldsymbol{x} + \phi)$	$b_d \ (= w_d)$

Table 1: Representational elements for all algorithms (with $g_d(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_d)^T \boldsymbol{\Sigma_d^{-1}} (\boldsymbol{x}-\boldsymbol{x}_d)}$)

So the whole list of non-linear regression algorithms studied in this paper can be seen as working on the same representation, feature-based mixture of linear models, where most of the used features are Gaussian functions, the only exception being iRFRLS which uses cosine features, and some representations are constrained to $\boldsymbol{A} = \boldsymbol{0}$.

The optimization criterion of non-linear regression algorithms is a function of the distance[2] to the demonstrated trajectories. By contrast, in the stochastic optimization stage, there is no constraint on the nature of the optimization criterion. Thus non-linear regression and stochastic optimization are complementary: non-linear regression cannot address all criteria and stochastic optimization needs to be bootstrapped from a good enough starting point. Furthermore, if the optimized controller remains close enough to the demonstrated trajectories, its behavior may satisfy some task-related criteria that are not explicitely embedded in the cost function but that the demonstrator may enforce when providing such trajectories.

In Section 5 we compare several function approximator implementations, when used for the function $f_{\boldsymbol{\theta}}$ in the context of regression for *Imitation* when using DMPs as a parameterized policy *Representation*.

4 *Optimization*: Policy Improvement with PI$^{\text{BB}}$

After the *Imitation* step, which in DMPs corresponds to determining $\boldsymbol{\theta}^{\text{init}}$ with regression as in (4), the *Optimization* step optimizes $\boldsymbol{\theta}$ with respect to a utility (reward or cost) function. The basic idea behind policy improvement is that a robot executes the parameterized policy with slight variations of the policy parameters ($\boldsymbol{\theta} + \epsilon$) and observes the rewards/costs. One such execution is known as a *rollout*. Given such rollouts, the parameters are updated so as to yield lower future costs.

Since the parameters $\boldsymbol{\theta}$ of DMPs correspond only to the open-loop term of the dynamical system, the costs/rewards accumulated during a *rollout* are not relevant to updating the parameters (Stulp and Sigaud, 2013). Therefore, policy improvement with DMPs can be treated as a black-box optimization problem, to which a variety of general purpose optimization algorithms are applicable (Stulp and Sigaud, 2012; Stulp and Sigaud, 2013). In this work we use the PI$^{\text{BB}}$algorithm – Policy Improvement through Black Box optimization – described in detail in (Stulp and Sigaud, 2013).

The PI$^{\text{BB}}$evolution strategy is explained and visualized in Figure 3, where the cost function is simply $J(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||$, i.e. the distance to the origin. The algorithm alternates between an exploration phase and a parameter update phase. *Exploration:* K policy parameter vectors $\boldsymbol{\theta}_{k=1...K}$ are sampled from a normal distribution with mean $\boldsymbol{\theta}_\mu$ and covariance matrix $\boldsymbol{\Sigma}$ (20). The cost function J is then evaluated for each of these vectors; in policy improvement this corresponds to executing the policy, and computing the return by summing over the costs at each time step (21). The K resulting rollouts are together called an *epoch*.

[2]Generally a quadratic function for algorithms based on Least-Square methods, i.e. using an L2-norm, as is the case in this paper, but one may also use L1-norm methods, see e.g. (Schmidt, 2005).

$$\boldsymbol{\theta}_{k=1\ldots K} \sim \mathcal{N}(\boldsymbol{\theta}_\mu, \boldsymbol{\Sigma}) \qquad \text{sample} \qquad (20)$$

$$\forall k \; J_k = J(\boldsymbol{\theta}_k) \qquad \text{cost function} \qquad (21)$$

$$\forall k \; P_k = e^{\left(\frac{-h(J_k - \min(J_k))}{\max(J_k) - \min(J_k)}\right)} \qquad \text{cost-to-weight} \qquad (22)$$

("lower cost ⇒ higher weight")

$$\boldsymbol{\theta}_\mu^{new} = \sum_{k=1}^{K} P_k \boldsymbol{\theta}_k \qquad \text{weighted averaging} \qquad (23)$$

$$\boldsymbol{\Sigma}^{new} = \sum_{k=1}^{K} P_k (\boldsymbol{\theta}_k - \boldsymbol{\theta}_\mu)(\boldsymbol{\theta}_k - \boldsymbol{\theta}_\mu)^\intercal \qquad \text{weighted averaging} \qquad (24)$$

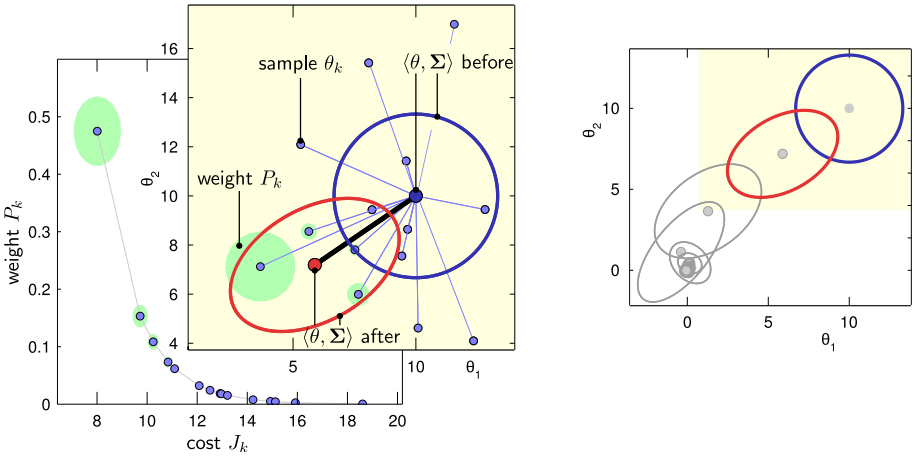


Figure 3: Visualization of PI^{BB}. Left: distributions in parameter space before (blue) and after (red) updating. The image in the background shows the mapping from costs J_k to weights P_k. Right: Iterative updating towards the minimum $\boldsymbol{\theta}^*$, which lies at the origin. The covariance matrix shrinks, once the mean of the distribution ($\boldsymbol{\theta}_\mu$) is at the optimum ($\boldsymbol{\theta}^*$).

Parameter Update: Given the scalar cost of each rollout, the costs are then converted into weights with an exponential mapping, which assigns higher weights to samples with lower costs (this function is visualized in the lower left corner of Figure 3). Then, the mean is computed by taking the weighted average over the samples (23). Because of the mapping from cost to weights, low-cost samples contribute more to the new mean then high-cost samples, and the the mean $\boldsymbol{\theta}_\mu$ (on average) moves closer to $\boldsymbol{\theta}^*$. The same is done for the covariance matrix (24). Figure 3 (left) visualizes such an update of the distribution. Updating the distribution is iterated until the costs converge, or a fixed number of iterations has been completed, as visualized in Figure 3 (right). The most important open parameters of PI^{BB}is the initial mean of the distribution $\boldsymbol{\theta}^{\text{init}}$ which is determined in the *Imitation* phase, and the initial covariance matrix $\boldsymbol{\Sigma}^{\text{init}}$. We study the effect of choosing $\boldsymbol{\Sigma}^{\text{init}}$ in Section 5.

5 Experimental Evaluation

We now investigate *Imitation* and *Optimization* phases in skill learning within one coherent experimental framework. Our experiments are designed explicitely to determine potential interdependencies between *Imitation* and *Optimization*, in terms of convergence speed and sensitivity to changes in algorithmic meta-parameters.

We study the influence of four independent variables, which are summarized in Table 2.

- The regression algorithm (LWR, LWPR, GMR, RBFN, iRFRLS) used for *Imitation*
- The meta-parameters of the different algorithms (e.g. D for LWR) used for *Imitation*

- The subset of model-parameters that are optimized (explained below) during *Optimization*
- The initial exploration magnitude ($\mathbf{\Sigma}^{\text{init}}$) for PI$^{\text{BB}}$at the beginning of *Optimization*, labelled `var`.

Imitation		*Optimization*	
1. FA used	2. algorithm meta-parameters	3. boolean parameter subset selectors	4. optimization meta-parameter
LWR	$D = 1 \dots 23$	`offsets, slopes, positions, shapes`	$\mathbf{\Sigma}^{\text{init}} = 10^{-\{2,3,4\}}$
LWPR	$w_{gen} = \{0.2, 0.5, 0.75\}$ $init_d = \{200, 500, 1000, 1200\}$	`offsets, slopes, positions, shapes`	$\mathbf{\Sigma}^{\text{init}} = 10^{-\{2,3,4\}}$
GMR	$D = 1 \dots 15$	`offsets, slopes, positions, shapes, priors`	$\mathbf{\Sigma}^{\text{init}} = 10^{-\{2,3,4\}}$
iRFRLS	$D = \{10, 15, 20\}$ $\gamma = \{2, 5, 10\}$ $\lambda = \{0.01, 0.05, 0.10\}$	`offsets, positions, shapes`	$\mathbf{\Sigma}^{\text{init}} = 10^{-\{2,3,4\}}$

Table 2: Independent variables used in the experiments

A grid search over all the possible meta-parameter values is performed for all regression algorithms. The specific values used in the grid search are also listed in Table 2 (we actually explored a wider scope but decided not to report them for figure clarity issues). Each *Imitation* and *Optimization* process is run four times for each combination of parameters, and we compute the average over the results.

Further optimization meta-parameters that are set to constant values are that 15 rollouts/update are used, and the eliteness parameter h is set to 10.

Optimization of different model-parameter subsets

Policy improvement consists of optimizing the parameters of a policy $\boldsymbol{\theta}$. As explained in Section 2, when a DMP is used as the policy representation, $\boldsymbol{\theta}$ corresponds to the parameters of a latent function. When using LWR, one might let $\boldsymbol{\theta}$ correspond to the slopes and offsets of the linear model, i.e. $\boldsymbol{\theta} = \langle \boldsymbol{A_d}, b_d \rangle$. This choice is quite arbitrary, as for LWR, one might also optimize the centers of the Gaussian basis functions, i.e. $\boldsymbol{\theta} = \langle \boldsymbol{A_d}, b_d, \boldsymbol{x}_d \rangle$, or just the centers $\boldsymbol{\theta} = \langle \boldsymbol{x}_d \rangle$. Our aim is to determine which subsets of parameters of different representations lead to the best optimization result. Whether a particular parameter type is included in the parameter vector $\boldsymbol{\theta}$ is determined by a set of boolean variable (which we denote with `this font`). For our unified representation, this leads to the following boolean options:

- `slopes`: whether the slopes of the linear models are included in $\boldsymbol{\theta}$
- `offsets`: whether the offsets $\boldsymbol{b}$ of the linear models are included in $\boldsymbol{\theta}$. $\boldsymbol{b}$ is the value of the line segment at the center of the basis function, not at the origin.
- `positions`: whether the positions of the basis functions are included in $\boldsymbol{\theta}$. For Gaussian basis functions these positions correspond to the centers. For iRFRLS they correspond to the phase ϕ of the cosine functions.
- `shapes`: whether the shapes of the basis functions are included in $\boldsymbol{\theta}$. For Gaussian basis functions the shapes correspond to the values in the covariance matrices. For iRFRLS it corresponds to the frequency ω of the cosine functions.
- `priors`: whether the priors are included in $\boldsymbol{\theta}$. For GMR only.

In our experiments, each combination of model-parameters subsets is optimized during the grid search.

Task

Similarly to (Theodorou et al., 2010), we use a viapoint task as a basis for our comparisons. The DMP generates a trajectory in 2D space. The initial state is $(0, 0.1)$ and the goal state is $g = (1, 0.9)$. The DMP is numerically integrated for $1s$ with a integration step of $dt = 0.01$. The aim of the task is to pass through a viapoint at $v = (0.3, 0.7)$, as visualized in Figure 4. The cost function penalizes the distance to the viapoint, the squared acceleration and the distance to the goal after T (see (25)).

$$J(\tau) = 100 \times \min_i(\|X_i - v\|) + 0.0001 \times \sum_{i=0}^{\infty} \|\ddot{X}_i\|^2 + \sum_{i=T\times dt}^{\infty} \|X_i - g\|^2 \quad (25)$$

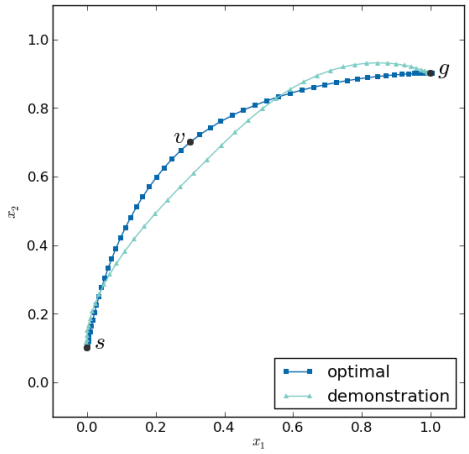


Figure 4: Viapoint task. The demonstration trajectory differs significantly from the one that is optimal with respect to the criterion given in (25)

Demonstrations are provided as cubic polynomials and of duration 0.5s points. These demonstrations are not optimal, i.e. see Figure 4 for a comparison of a demonstration and an optimal trajectory. If the demonstration would be optimal, we would not be able to measure as well the contribution that the *Optimization* phase makes to finding the optimal movement. In real robotics problems, the demonstration is often suboptimal also (Kober and Peters, 2008).

For each regression algorithm, many *Imitation +Optimization* learning sessions are run. For LWR for instance, we ran 4140=23×15×3×4 learning sessions. 23 for $D = 1 \ldots 23$, 15 for all true/false combinations of the 4 parameter subset selectors (with at least one being true), 3 for $\mathbf{\Sigma}^{\text{init}} = 10^{-\{2,3,4\}}$, and 4 learning sessions for each of these combinations.

5.1 Results

In this study we consider three questions:

1. does using a non-linear regression algorithm rather than another make a difference in terms of the optimization performance?
2. how sensitive are the different non-linear regression algorithms to meta-parameter setting?
3. are there combinations of function approximation meta-parameters, boolean parameters and optimization meta-parameters that result in more efficient searches than others?

5.1.1 Learning curves

Figure 5 shows the learning curves ($\mu \pm \sigma$) for the best 10% of these experiments. For instance, for LWR these are the 41 learning curves with the best performance. The single best experiment parameters for each algorithm are listed in Table 3.

From these results, we conclude that GMR achieves the best performance, that LWPR is competitive for a small number of optimization iterations and that iRFRLS has a larger variance than other algorithms. All algorithms seem to converge at a similar rate and LWR, LWPR and iRFRLS converge to a similar performance, whereas GMR does significantly better.

The fact that GMR outperforms LWR is not surprising, since it generates latent functions where the basis function placement has been optimized whereas they are fixed in LWR. This performance gain comes at the price of a higher computational cost.

Imitation 1. FA used	2. function approx. meta-parameters	*Optimization* 3. boolean parameter subset selectors `slopes`	`offsets`	`positions`	`shapes`	`priors`	4. optimization meta-parameter `var`	5. final cost
LWR	$D = 22$	false	true	true	false	N/A	$\mathbf{\Sigma}^{\text{init}} = 10^{-2}$	0.0416
LWPR	$w_{gen} = 0.2, init_d = 1200$	true	false	false	true	N/A	$\mathbf{\Sigma}^{\text{init}} = 10^{-3}$	0.0460
GMR	$D = 13$	true	true	false	false	true	$\mathbf{\Sigma}^{\text{init}} = 10^{-4}$	0.0413
iRFRLS	$D = 15, \gamma = 10, \lambda = 0.01$	N/A	false	true	false	N/A	$\mathbf{\Sigma}^{\text{init}} = 10^{-3}$	0.0448

Table 3: Values of experiment parameters with the best performance for each algorithm.

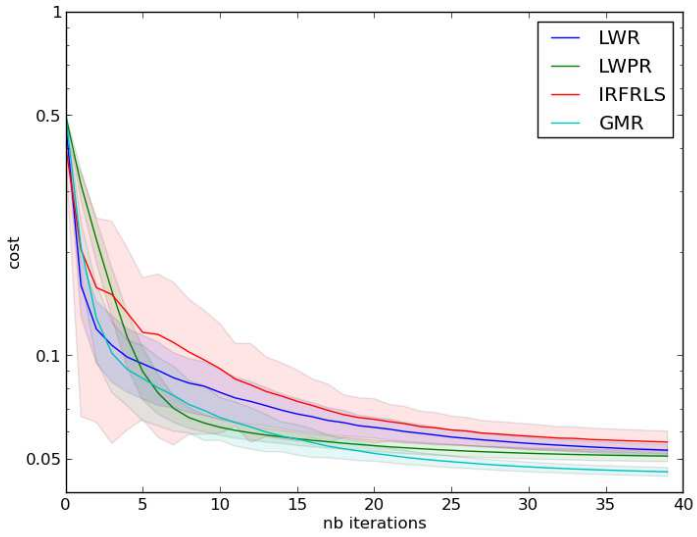


Figure 5: Optimization performance through iterations: for each algorithm, the best 10 percent of the used meta-parameters were selected and their mean and standard deviation are shown. The y axis uses a logarithmic scale.

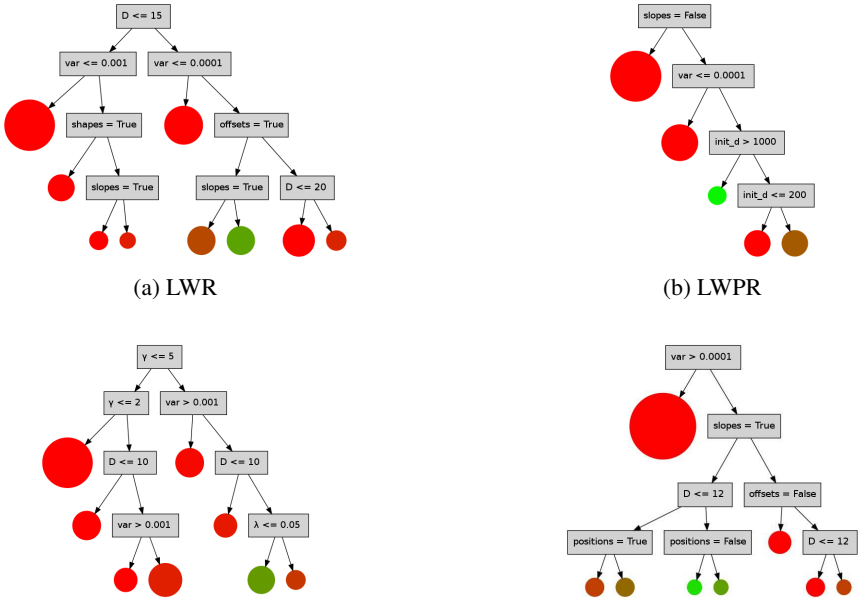
5.1.2 Influence of optimizing different model-parameter subsets

The boolean parameter subset selectors determine which types of model-parameters are included in the parameter space Θ that is used for optimization. For LWR for instance, $\boldsymbol{\theta}$ may contain only the line slopes, or the line slopes and offset, or just the basis function centers, etc.

To investigate which parameter subsets leads to the best optimization performance for which algorithms, we split the learning sessions of each algorithm into two classes: the best 10% (as above), and the rest, i.e. 90% 'underperformers'. We then predict which class a learning session belongs with the C4.5 decision tree learning algorithm, where the input features are the experiment parameters in Table 2. The results for the four regression algorithms are shown in Figure 6.

These trees nicely condense the wealth of information contained in the 1000s of learning sessions. They provide recommendations about how to set meta-parameters of the function approximators, the initial exploration magnitude $\mathbf{\Sigma}$ for optimization, as well as which model-parameter subsets should be used for optimization.

For instance, although LWR and LWPR belong to the same family, we see here that best optimization results are obtained for LWR when both offsets and slopes are optimized, whereas for LWPR it is better not to include the slopes in the optimization. Another interesting observation is that the number of basis functions D appears in all trees (except LWPR, which does not have this meta-parameter), and that all decisions nodes specify *upper* bounds for D, e.g. $D \leq 10$, or $D \leq 12$. Apparently, for optimization it is better to have fewer basis functions, which corresponds to our intuition that smaller search spaces lead on average to faster optimization.



(a) LWR (b) LWPR

(c) IRFRLS (d) GMR

Figure 6: Decision trees for meta-parameter selection. For each algorithm, a dataset was created where the positive class contains the meta-parameters that give rise to the best 10% of the optimization performance. Then decision tree were grown with c4.5 and truncated at a depth of four decision nodes. Left branch means that the test is true, right that it is false. The color shows how many instances in the leafs belong to the best 10% of the used meta-parameters, green means all positive, red means all negative. The size is related to the number of instances in each leaf.

5.1.3 Sensitivity to meta-parameters tuning

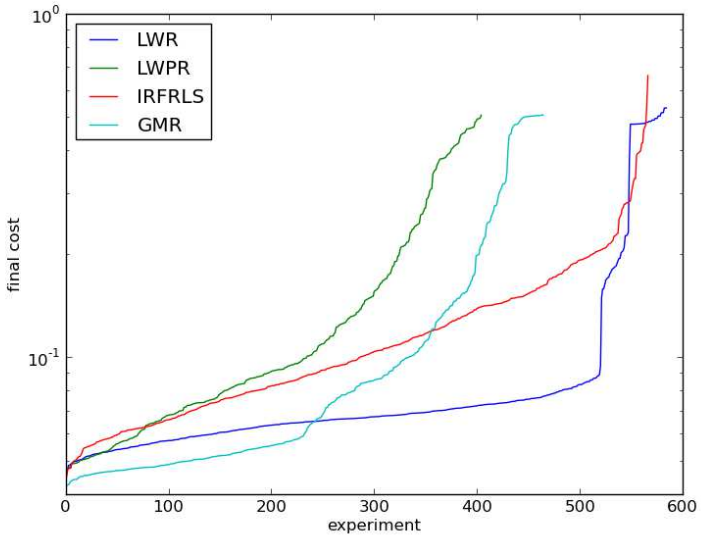


Figure 7: Sensitivity of each algorithm to meta-parameter tuning: for each algorithm, for N experiments, the average cost of the N best instances is plotted using a logarithmic scale. Thus, we can see how fast the performance degrades as more sub-optimal meta-parameter sets are added.

Figure 7 shows the sensitivity of these non-linear regression algorithms to meta-parameter tuning. A curve

that grows quicker means a greater sensitivity. One can see that LWR is the most robust to meta-parameter tuning, whereas iRFRLS is very sensitive, this may explain the greater standard deviation in Figure 5. The fact that iRFRLS is the only non-local algorithm may explain this higher sensitivity.

6 Conclusion and Future work

Motor skill acquisition is a central concern for robot learning. This problem requires efficient methods and representations because it involves a costly optimization step. In this work, in the search for the most efficient *Representation*, we have studied the relation between the *Imitation* stage and the *Optimization* stage when trying to acquire new motor skills using Dynamical Movement Primitives as policy representation.

The general finding is that the simplest algorithms like GMR and LWR are good enough for a task as simple as learning the forcing term of a DMP parametrized with time, which is a one dimensional problem. More sophisticated methods like LWPR and iRFRLS do not make a positive difference in this context, for iRFRLS because of the sensitivity resulting from non-local approximation and for LWPR because it is designed to perform non-linear regression in high-dimensional domains which is not the case when learning DMPs parametrized through just time.

The choice between GMR and LWR should mostly be driven by potential constraints on incrementality or computational complexity. If there are such constraints, LWR should be preferred to GMR, otherwise GMR should generally provide a better performance.

One may argue that the task studied here is too simple for this comparison. But the same applies to all cases where DMPs are parametrized through time, which is the case in most applications of these techniques to robotics.

Actually, there are many other applications of these optimization techniques where the chosen non-linear regression algorithm may make a difference. In the near future, we will perform a similar study in a context where the training trajectories are recorded from noisy human movements. On a longer term, we should also study the impact of the non-linear regression algorithm on learning *contextual DMPs*[3] (Stulp et al., 2013) which require to search a potentially much larger space. So it is unclear whether the results we obtained on standard DMPs should also hold for the case of contextual DMPs and controllers based on dynamical systems.

Another concern is that using *Imitation* before *Optimization* is based on the insight that, if the demonstrated trajectories are close enough to optimal trajectories, *Imitation* should provide a starting point for *Optimization* leading to a good local optimum, hence a good controller. However, if demonstrated trajectories are too far away from the optimal ones, the benefit resulting from using them in the first stage as a bootstrap may vanish. In such a context, one may compare all the performances in our work to performances obtained from using random representations as initial controllers for the stochastic search process. Actually, in this study we did not vary the distance between the demonstration trajectories and the optimal trajectories with respect to the optimization criterion. This is left for future work.

Acknowledgments

This work was supported by the MACSi Project (ANR-BLAN-0216), more at http://macsi.isir.upmc.fr and the 3rdHand Project (FP7-ICT-2013-10-610878).

References

Atkeson, C. G. and Schaal, S. (1995). Memory-based neural networks for robot learning. *Neurocomputing*, 9(3):243–269.

Butz, M. V. and Herbort, O. (2008). Context-dependent predictions and cognitive arm control with XCSF. In *Conference on Genetic and Evolutionary Computation*, pages 1357–1364. ACM.

Butz, M. V., Kovacs, T., Lanzi, P. L., and Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46.

[3]They are called "parametrized DMPs" in (Stulp et al., 2013) but this name is unfortunate.

Calinon, S. (2009). *Robot programming by demonstration*. EPFL/CRC Press.

Cederborg, T., Li, M., Baranes, A., and Oudeyer, P.-Y. (2010). Incremental local online Gaussian mixture regression for imitation learning of multiple tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 267–274.

Dekel, O., Shalev-Shwartz, S., and Singer, Y. (2008). The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38.

Droniou, A., Ivaldi, S., Padois, V., and Sigaud, O. (2012). Autonomous online learning of velocity kinematics on the icub: A comparative study. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3577–3582.

Engel, Y., Mannor, S., and Meir, R. (2002). Sparse online greedy support vector regression. In *ECML 2002*, pages 84–96. Springer.

Geladi, P. and Kowalski, B. (1986). Partial least squares regression:a tutorial. *Analytica Chimica Acta*, 185:1–17.

Ghahramani, Z. and Jordan, M. I. (1994). Supervised learning from incomplete data via an em approach. In *Advances in Neural Information Processing Systems 6*.

Gijsberts, A. and Metta, G. (2011). Incremental learning of robot dynamics using random features. In *IEEE International Conference on Robotics and Automation*, pages 951–956.

Gijsberts, A. and Metta, G. (2012). Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*.

Hansen, N., Muller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

Hersch, M., Guenter, F., Calinon, S., and Billard, A. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467.

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

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable non-linear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*.

Kober, J. and Peters, J. (2008). Policy search for motor primitives in robotics. In *NIPS*, pages 1–8.

Kulvicius, T., Ning, K., Tamosiunaite, M., and Worgötter, F. (2012). Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157.

Park, J. and Sandberg, I. W. (1993). Approximation and radial-basis-function networks. *Neural computation*, 5(2):305–316.

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184.

Rifkin, R., Yeo, G., and Poggio, T. (2003). Regularized least-squares classification. *Nato Science Series Sub Series III Computer and Systems Sciences*, 190:131–154.

Rubinstein, R. Y. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190.

Saunders, C., Gammerman, A., and Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In *(ICML-1998) Proceedings of the 15th International Conference on Machine Learning*, pages 515–521. Morgan Kaufmann.

Schmidt, M. (2005). Least squares optimization with l1-norm regularization. Technical report, CS542B Project Report.

Sigaud, O., Salaün, C., and Padois, V. (2011). On-line regression algorithms for learning mechanical models of robots: a survey. *Robotics and Autonomous Systems*, 51:1117–1125.

Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013). Learning compact parameterized skills with expanded function approximators. In *Proceedings of the IEEE International Conference on Humanoids Robotics*, pages 1–7.

Stulp, F. and Sigaud, O. (2012). Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML'2012)*, pages 1–8, Edinburgh, Scotland.

Stulp, F. and Sigaud, O. (2013). Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn Journal of Behavioral Robotics*, 4(1):49–61.

Theodorou, E., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: a path integral approach. In *International Conference on Robotics and Automation*, pages 2397–2403. IEEE.

Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, pages 288–293.