Non-parametric methods in machine learning: applications in robotics and data analysis

Final scientific report

Lehel Csató

October 10, 2014

Abstract

This synthesis presents the scientific results related to the project for young research groups, code **PN-II-RU-TE-2011-3-0278** for the period *2011-2014*.In this document we detail the results grouped in categories according to the table of contents as below.

The web-address of the project is: `http://datamin.ubbcluj.ro/index.php?a=projects`; An important contribution of the project towards the education of young researchers was the scientific seminars can be found at the seminars section.

Contents

1 Introduction

The general topic of non-parametric methods and their applications in machine learning comprises of several important sub-domains, all of these domains in principle being capable of handling large – or vast – amounts of data. Our project dealt with studying automated machine learning algorithms, focusing to inferring non-parametric models and on applying the probabilistic Bayes method. We applied our algorithms to robotics and to the process of analysing data, with accent on analysing textual data.

The potential applicability – mainly due to their increased complexity – of these methods justifies the effort to create *automated* data processing algorithms – like those within the family of reinforcement learning – or ones that require minimal human intervention; we mention here the semi-supervised learning methods. The main advantage of non-parametric methods is their possibility to "automatically" set the complexity of the resulting model, this complexity adjustment is done via the addition or removal of some *latent* variables into/from the model. The main directions considered in this research plan were the following:

1. Applying – the non-parametric and probabilistic – Gaussian processes (GP) for the development of novel reinforcement learning algorithms, more precisely approximate reinforcement learning methods.
2. Application of Gaussian processes in the development of novel robotic algorithms.
3. Using non-parametric methods for improving input noise models.
4. Developing algorithms for improving fast search methods and applying non-parametric methods within semi-supervised methods.

2 Approximate algorithms for reinforcement learning

The field of *approximate reinforcement learning (ARL)* is one of the sub-domains of practically successful reinforcement learning algorithms. We mention once more that the importance of these methods is that – contrary to the classical inference methods – ARL can be applied relatively easily to real-world control and robot learning problems without the need to be specific – i.e. mathematically model – the dynamics or kinematics of the robotic unit. These types of applications are characterised as follows:

- Continuous and high-dimensional state and action spaces;
- Fast – usually *on-line* – algorithms are necessary for successful functioning;
- Diversity of control problems, leading to a difficulty in designing universal approximating methods;
- The number of available *trials* that can be used for training is limited;
- The computational costs also have to be cut – in order to reduce the energy consumption of the – potentially – mobile unit that embeds the controller.

The characteristics mentioned above led the research directions within this section. (1) We addressed the shortcomings of the classical RL algorithms based on the notion of a *value function* that enumerates the state–action Cartesian space and computes a value for each element. Once the spaces are large – or infinite – we have to resort to approximations; these approximations have to be adjusted to the application domain to be efficient, in a more concrete situation the complexity of the solution has to be set "manually", either by inspection or by repeated trials. The *non-parametric* and probabilistic Gaussian processes (GPs) on the other hand contain a complexity regulation mechanism, therefore we studied these methods. (2) A second advantage of GPs is their possibility to be employed *"on-line"*, i.e. the inferred model can be simultaneously exploited and further refined – explored –, both by continuing the interaction with the environment. Given the specifics of the Bayesian and non-probabilistic Gaussian processes, the continuous exploration of the model might lead to an ever-growing parameter size, the change in the latent objective function – i.e. non-stationary behaviour –, and the highly correlated measurements that degrade the performance of the learning algorithm. Within the project we addressed some of the problems mentioned above, most importantly the possibility to reduce the computational costs using an elimination algorithm inspired the original sparse on-line algorithm [Csató and Opper, 2002].

2.1 Improving the approximations to the value functions

In year 2011 we studied the possibility to improve on the approximations to value functions and to the state-action Q functions. The theoretical background is given by the *"Markov Decision Processes"* – or MDPs – [Puterman, 1994], defined as a quadruple of $M\,(S, A, P, R)$, with S being the set of states, A the set of actions, $P(s'|a, s)$ *describes the world* via the conditional probabilities that specify – within a non-deterministic set-up – the result on an action a when our machine is in state s, and $R(s)$ is the reward function associated to each and every state

that we are considering. Solving an MDP means the computation of an *optimal strategy* that gives – after running the algorithm a number of times – the largest *expected accumulated reward*, and this function is defined as the following conditional probability: $\pi(a|s)$, being interpreted as the conditional probability of choosing an *action* a when in state s. Classically the optimal strategy is defined with the help of the value – $V(s)$ – and Q- – $Q(s,a)$ – functions [Sutton and Barto, 1998]. The definition domains of the value and the Q-functions are $V_\pi : S \to \mathbb{R}$ and $Q_\pi : S \times A \to \mathbb{R}$ respectively, and the corresponding definitions are:

$$V_\pi(s) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right], \qquad Q_\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s, a_0 = a\right]$$

In the equations above $s \in S$ is a state;$a \in A$ is an action, and $\pi : S \times A \to [0,1]$ is a *strategy* based on which an action in the given state is chosen.

Applying RL methods for real-life problems requires approximating the value functions and the Q-functions. Additionally to that, the strategy function often needs to be approximated also. The usage of Gaussian processes for approximating value functions has been studied before [Rasmussen and Williams, 2006, Ghavamzadeh and Engel, 2007, Deisenroth et al., 2009], one of the main drawback in those methods was the unavailability of quick on-line process inference. We tackled the efficient inference using GP's by re-using both the training data – new data generation could be difficult – and the re-use of the model [Jakab and Csató, 2011]. We modified the on-line sparsification method that was based on the Kullback-Leibler distance of two GPs.

2.2 Reinforcement learning using Gaussian processes and directed search

Approximating functions using a probabilistic set-up can be exploited within the family of *policy gradient* (PG) methods. Here we exploit the extra information provided by the *mean* of the posterior GP in order to reduce the variance of the PG gradient estimates. A second direction is the influencing the search directions of the RL algorithms, again using the variance information. In both cases we had to find – to infer – an a-posteriori GP, for which we used *pairs* of states-actions $x_t \stackrel{\text{def}}{=} (s_t, a_t)$ and as outputs we used the "discounted" rewards: $\sum_{i=0}^{H-t} \gamma^i R(s_{t+i}, a_{t+i})$, that are based on a momentary value of the policy function (therefore the output value for a given input might change whilst using the algorithm). To train the Gaussian process we used the sparse on-line methods developed in [Csató and Opper, 2002].

2.2.1 Modifying search directions in reinforcement learning problems

Policy gradient (PG) methods require an explicit representation of the strategy function by using a parametric form, with θ the *set of parameters* to the policy function, therefore the notation $\pi_\theta(s,a) \stackrel{\text{def}}{=} \pi(a|s)$.

In [Jakab and Csató, 2012] we introduced two algorithmic variations to improve the exploration process. The first one is the change of the amplitude of the added noise – therefore the exploration radius – to the strategy function as follows:

$$\begin{aligned} \pi_\theta &= c(s,\theta_c) + \mathcal{N}(0, \sigma^2_{GP} I) \\ \sigma^2_{GP} &= \lambda\left(k_q(x^*,x^*) - \mathbf{k}^* \mathbf{C}_n \mathbf{k}^{*T}\right), \quad \text{where } x^* = \{s, c(s,\theta_c)\} \end{aligned} \tag{1}$$

In the equation above $\mathbf{k}^* = [k_q(x^*,x_1), \ldots, k_q(x^*,x_n)]$ is a vector that contains the covariances of the function values at x^* with the rest of the training data $\mathcal{D}$, σ^2_{GP} is the variance of the GP evaluated at $x^* \stackrel{\text{def}}{=} (s, c(s|\theta_c))$, the function $c(\cdot)$ is the parametrized controller.

To influence both the size and the direction of the exploration step, in [Jakab and Csató, 2012] we introduced an algorithm that uses the mean function of the GP evaluated in a proximity of the current state (a Gibbs-like policy function):

$$\pi(a|s) = \frac{e^{\beta E(s,a)}}{Z(\beta)}, \text{ where } Z(\beta) = \int da\, e^{\beta E(s,a)} \text{ is the normalising function of the probability.} \tag{2}$$

The above formalism allows us to define a *stochastic policy function* whose fluctuation can be controlled by the parameter β defined as an "inverse temperature". The choice of the "optimal strategy" a in a state s, given the strategy $\pi_\theta(a|s)$, is achieved with the energy function $E(s,a)$. This function includes – as a multiplicative term – a penalty for selecting actions that are far from $c_\theta(s)$:

$$E(s,a) \quad = \quad \hat{Q}_{GP}(s,a)\ \exp\left[-\frac{\parallel a - c_\theta(s) \parallel^2}{2\sigma_e^2}\right]$$

2.2.2 Reducing the variance of the policy gradient method

A second difficulty of the PG methods is the large variance when estimating the gradient of the parameter; we tackled the variance reduction in [Jakab and Csató, 2012], which we define next.

In the classical "policy gradient" methods, in each learning step the gradient of an *objective function* – $J^\pi = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right]$ is computed; the gradient being w.r.to the parameters of the policy function $\pi(a|s,\theta)$, θ. The usual computation mechanism is the Monte-Carlo (MC) approximation:

$$\nabla_\theta J = E_\tau\left[\sum_{t=0}^{H-1}\nabla_\theta \log\pi(a_t|s_t)\sum_{i=0}^{H-t}\gamma^i R(s_{t+i},a_{t+i})\right] \tag{3}$$

where τ represents a "roll-out" – sequence of outputs resulting from running the controller H steps, and $E[\cdot]_\tau$ is the average w.r.to the "distribution" of the "roll-outs" τ. The above formulation is nice in the sense that it does not depend on the particular parametrisation of the strategy function but due to the *high variance* of $\sum_{i=0}^{H-t}\gamma^i R(s_{t+i},a_{t+i})$, the calculated gradient and thus the MC-approximation can be slow.

In [Jakab and Csató, 2012] we demonstrated that – by using a GP for approximating the value function and by replacing the MC-average in the expression of the gradient with the mean of the posterior GP, the variance of the estimation can be significantly reduced:

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{H-1}\nabla_\theta \log\pi(a_t|s_t)\hat{Q}_{GP}(s_t,a_t)\right], \tag{4}$$

where $\hat{Q}_{GP}(\cdot,\cdot)$ is the GP-approximated value function.

2.3 Manifold-based non-parametric approximations

Approximating value functions is difficult also due to the discontinuities inherently present in the value function (or Q-function); these discontinuities make the function inference difficult.

We tried to represent the manifolds on which the value (or Q-function) varies smoothly using a *data-dependent kernel function*, inferred during the exploration phase. The kernel function has the form:

$$k_{sp}(x,x') = A\exp\left(-\frac{\mathcal{SP}(x,x')^2}{2\sigma_{sp}}\right) \tag{5}$$

where A and σ_{sp} are positive hyper-parameters, and $\mathcal{SP}$ is the *shortest-path* kernel.

The shortest-path kernel is based on the idea of nearest neighbours that are stored and represented as a graph that is built in an on-line manner:

$$E_{x_t,x_i} = \begin{cases}\|x_i - x_t\|^2 & \text{if } \|x_i - x_t\| < \varepsilon \quad \varepsilon > 0\\ 0 & \text{otherwise}\end{cases} \tag{6}$$

The algorithm above builds the ε-neighbourhood graph and effectively limits the number of neighbours, making the algorithm efficient.

Using the neighbourhood graph, we calculate the *shortest-path* along the manifold defined by the model. The model is built only with the available data using temporal relations. The function used is the following – required for discretisation of the input space –, where x^* is a new point and x_j is a node in the graph:

$$\begin{aligned}\mathcal{SP}(x^*,x_j) &\overset{(1)}{=} \|x^* - x_i\|^2 + \mathbf{P}_{i,j}, \text{ with } x_i = \underset{x_\ell\in BV}{\operatorname{argmin}}\|x^* - x_\ell\|^2\\ \mathcal{SP}(x^*,x_j) &\overset{(2)}{=} \mathbf{k}_{x*}^T\mathbf{P}\mathbf{e}_j = \sum_{i=1}^{n}k(x^*,x_i)\mathbf{P}_{i,j}\end{aligned}$$

where the matrix $\mathbf{P}$ encodes the shortest paths between x_i and x_j, and $\mathbf{e}_j$is the j-th unit vector of length n, the details of the method can be found in [Jakab and Csató, 2012].

2.4 Intelligent models for modelling robot behaviour

In March 2012 Hunor Jakab successfully finished his PhD with the thesis entitled *Intelligent Models for Robotic Behavior, Decision Making and Environment Interaction* [Jakab, 2012].

The thesis details the research undertaken in the period 2009–2012, a significant portion being done within the present project. The thesis contains 7 chapters and deals with the improvement possibilities of methods for robot learning using reinforcement learning and non-parametric methods, with emphasis on their applicability to real-world problems.

2.5 Sparse models in reinforcement learning algorithms

In 2013 we studied the methods to obtain models with sparse – i.e. reduced but efficient – representations that can be used in reinforcement algorithms in non-parametric set-up.

We considered an alternative method for approximating value functions, the *least squares value approximation* (LSVA) method, introduced by Bradtke et al. [1996]. The method is based on the *temporal difference learning* (TD) – a well-known method in classical reinforcement learning scenario [Sutton and Barto, 1998]. The adaptation of this algorithm to problems with high dimensionality is done – again – via the "kernelization" of the input space, leading to a model of the following form:

$$\tilde{V}(s_t) = \sum_{i=0}^{n} \mathbf{w}_i^* k(s_i, s_t) \quad s \in S, \tag{7}$$

where – after a series of transformations – $\mathbf{w}_i^*$ are scalars that parametrise the value function. An important result is the *iterative* re-calculation of the weights using data structures based on the explored data:

$$\mathbf{w}^* = \tilde{A}^{-1}\tilde{b}, \text{ where } \tilde{A} = \frac{1}{n}\sum_{t=0}^{n} \mathbf{k}(s_t)\left[\mathbf{k}^T(s_t) - \gamma \mathbf{k}^T(s_{t+1})\right], \quad \tilde{b} = \frac{1}{n}\sum_{t=0}^{n} \mathbf{k}(s_t) R_t. \tag{8}$$

where – as in the previous sections – $k(\cdot,\cdot)$ is a kernel function, $\mathbf{k}(s_t) = [k(s_t, s_1), \ldots, k(s_t, s_n)]^T$ is the concatenation of the kernel function values evaluated at the new point s_t and the training points $\mathcal{D} = \{s_i \,|i \in [1, n]\}$

The above iterative update permits the extension of the method to the following *approximative* method: each time a new training input is presented, we can decide whether to keep it and build into the model, or to "throw" it and keep only some information about the datum. The procedure of above – depending on the amount and definition of the information – is called *sparsification* and is necessary since the inversion of matrix $\tilde{A}$ is cubic [Csató and Opper, 2002].

2.5.1 Data sparsification

Following the above research directions, in year 2013 we studied the *approximate linear dependencies*, a method frequently used in realizing sparse algorithms. The sparse linear dependency method keeps those inputs that cannot be expressed – or cannot be expressed without significant loss – using the already existing inputs. In [Jakab and Csató, 2013] we introduced a sparsification method based on an on-line graph built in an on-line and iterative manner on a subset of input points. It uses the *Laplace* operator to formulate a criterion function $\mu(\cdot)$ that is used as a decision function when deciding the inclusion or dropping of a new training datum: the importance of a node $s_i \in \mathcal{V}$ is a weighted sum of squared differences between neighbouring function values, the weighting is given by the adjacency matrix: $\mu(s_i \in \mathcal{V}) = \sum_{v_j \in \text{neig}(s_i)} A_{ij} \left[f(s_i) - f(s_j)\right]^2$. The fitness - quality – of the set $\mathcal{V}$ is expressed using the same Laplacian operator as:

$$\sum_{s_i \in \mathcal{V}} \mu(s_i) = \sum_{i,j=1}^{n} A_{ij} \left[f(s_i) - f(s_j)\right]^2 = \mathbf{f^T L f}. \tag{9}$$

For building the graph $G(\mathcal{E}, \mathcal{V}), \quad \mathcal{V} \subset S$ we built two algorithms: a first one based on the *k-nearest neighbour* (KNN) and a second one based on the *extended sphere of influence* (eSIG). Details of the algorithms can be found in [Jakab and Csató, 2013]. The experimental results show that the sparsification method helps in distributing the elements of $\mathcal{V}$ in "important" regions of the input space: the new functions are more stable and more precise when compared to the results of the classical methods.

2.5.2 Improving on sparsification methods

In 2014 we improved on the sparse LSVA method – results being published in [Jakab and Csató, 2014]. One of the important results is the simplification of the one-line update rules for the matrix $\tilde{A}$ and the vector $\tilde{b}$ from (8). Note that this reduces the one-step computation from cubic time – as defined previously – to a quadratic time, reducing the computational cost, therefore permitting the application of the algorithm to large data-sets. The method is based on the incremental computation of the inverse $C = \tilde{A}^{-1}$ without keeping the original matrix $\tilde{A}$. The update equations are the following:

$$C_{t+1} = \tilde{A}_{t+1}^{-1} = \frac{t+1}{t}\begin{bmatrix}\tilde{A}_t + \mathbf{u}\mathbf{v}^T & v^*\mathbf{u} \\ u^*\mathbf{v}^T & u^*v^*\end{bmatrix}^{-1} = \begin{bmatrix}C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^* v^*}\end{bmatrix} \tag{10}$$

where – to keep a cleaner notation – we used the following notations:

$$\mathbf{u} = [k(s_t, s_1), \ldots, k(s_t, s_t)]^T \qquad u^* = k(s_t, s_{t+1})$$
$$\mathbf{v} = [k(s_t, s_1) - \gamma k(s_{t+1}, s_1), \ldots, k(s_t, s_t) - \gamma k(s_{t+1}, s_t)]^T \qquad v^* = k(s_t, s_{t+1} - \gamma k(s_{t+1}, s_{t+1})$$

The above results are obtainable using the Sherman-Woodbury equations, similarly to [Csató and Opper, 2002]. The final result is of a surprisingly simple form:

$$\begin{aligned}\mathbf{w}_{t+1} &= C_{t+1}\tilde{\mathbf{b}}_{t+1} = \begin{bmatrix}C_t & -C_t\mathbf{u}/u^* \\ -\mathbf{v}^T C_t/v^* & \frac{1+\mathbf{v}^T C_t \mathbf{u}}{u^* v^*}\end{bmatrix}\left(\begin{bmatrix}\tilde{\mathbf{b}}_t \\ 0\end{bmatrix} + \begin{bmatrix}\mathbf{u} \\ u^*\end{bmatrix} R_t\right) \\ &= \begin{bmatrix}\mathbf{w}_t \\ \left(R_t - \mathbf{v}^T\mathbf{w}_t\right)/v^*\end{bmatrix} = \begin{bmatrix}\mathbf{w}_t \\ \left(R_t - (\tilde{V}(s_t) - \gamma\tilde{V}(s_{t+1}))\right)/v^*\end{bmatrix}\end{aligned} \tag{11}$$

and the update for $\tilde{\mathbf{b}}_t$ is:

$$\tilde{\mathbf{b}}_{t+1} = \frac{t}{t+1}\left(\begin{bmatrix}\tilde{\mathbf{b}}_t \\ 0\end{bmatrix} + \begin{bmatrix}\mathbf{u} \\ u^*\end{bmatrix} R_t\right)$$

We highlight once more that the computational costs are reduced from an original $\mathcal{O}(n^3)$ to a much lower $O(nk^2)$, where n is the number of processed data and k is the number of data kept as "parameters". Note that n is the most significant, since in principle this can be a continuous stream of inputs. Finally we mention that in [Bócsi et al., 2014] we investigated new possibilities of building proximity graphs to improve on the models coming from sparsified algorithms.

3 Model learning for tracking control

Robotic control is in generally based on the mathematical model of the robot, defined using the physical parameters of the robot. Based on the physical parameters, the equations for desired joint configurations – known as kinematic model – or the desired forces – the dynamic model – have all analytical forms. The analytic specification of the robot model is beneficial for rigid robot architecture. When either the robot architecture or the interacting environment are changing, the analytic models encounter a series of drawbacks, resulting either from an under-specified mathematical model, or from an overly complex model that is difficult to handle.

In this section we propose an adaptive treatment to modelling the robotic problem. Our aim was not to define a control model based on the physical characteristics of the robot, but to replace it with a function that is inferred from sensory inputs and the control signals. Being an adaptive method, we arrive to an *efficient and fast approximation to the model*.

3.1 Robotic models for tracking control

Tracking control problems are formulated usually in the "task-space" – *i.e.* in the space of the end-effector of the robot – the goal being to follow a pre-defined trajectory. The solution to the tracking control is typically *not unique*: for redundant robots for – almost – every position of the end-effector there are multiple joint configurations leading to the same state; usually these solutions form a non-convex set. In [Bócsi et al., 2011b,a, 2012, 2014a] we described algorithms – based on machine learning methods – to control non-rigid robots in situations where standard solutions and standard algorithms perform poorly. The tracking control algorithm comprises of three

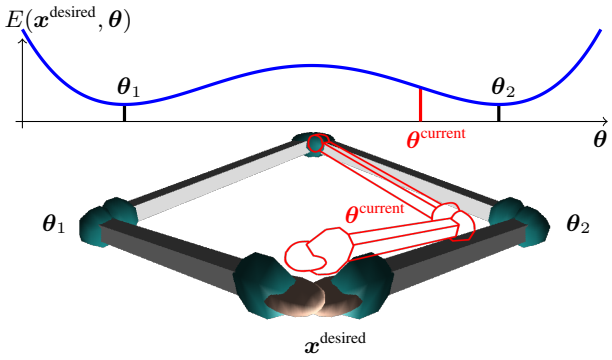


Figure 1: Illustration of prediction using the inverse kinematics.

parts: (1.) approximate a common model for the effector space and for the joint space by using machine learning methods. (2.) perform local optimisation for obtaining the inverse kinematics, denoted as f^{-1}; (3.) based on the inverse kinematics, we use a tracking controller to all joints, with full degree of freedom (DoF) of the robot.

3.2 Indirect model learning

The principal observation is that the model minimising $E(\mathbf{x}, \theta)$ – a mapping from input data to outputs – is well defined and the predictive output is obtained by minimising the model *at the current input*:

$$f^{-1}(\mathbf{x}) \quad \stackrel{\circ}{=} \quad \operatorname*{argmin}_{\theta \in \theta} E(\mathbf{x}, \theta), \tag{12}$$

where $\mathbf{x}$ is the position of the effector and θ is the coordinates in the joint configuration space. An interesting question is related to whether the eq. (12), has a unique solution or not, and more importantly: if an end-effector state $\mathbf{x}^{\text{desired}}$ can be reached from multiple joint configurations, e.g. θ_1 and θ_2 in Fig. 1, then how to proceed? We designed an algorithm such that the predictions are the closest to the existing configuration, such that the hard moves – quick ones that might damage the unit – are avoided. In the case of Fig. 1 the prediction is θ_2, that leads to a smooth operation; this can be achieved by starting to minimise the energy from the current joint configuration.

We observe that the model $E(\cdot, \cdot)$ is important in being able to solve the equations – in the manner specified above: i.e. adaptively finding the closest solution to the current joint configuration. We proposed three solutions, using the methods (1) *joint kernel support estimation*, (2) *structured output Gaussian processes*, and (3) a method based on the kinematic model of the robot. For the *"joint kernel support estimation"* (JKSE) [Bócsi et al., 2011b], the energy function is the negated data log-probability of the data $(\mathbf{x}, \theta)$:

$$E(\mathbf{x}, \theta) \quad \stackrel{\circ}{=} \quad -p(\mathbf{x}, \theta). \tag{13}$$

The method JKSE models the joint distribution of the input data as a log-linear model in a feature space. Using thus the joint log-probabilities, the inverse kinematics model is:

$$f^{-1}(\mathbf{x}) \quad = \quad \operatorname*{argmax}_{\theta \in \theta} \ \mathbf{w}^\top \phi(\mathbf{x}, \theta),$$

where $\mathbf{w}$ is the vector of parameters to the distribution that explains the data $\mathcal{D} = \{(\mathbf{x}_i, \theta_i)\}_{i=1}^m$. The model uses the one-class support vector machines to compute the values for $\mathbf{w}$.

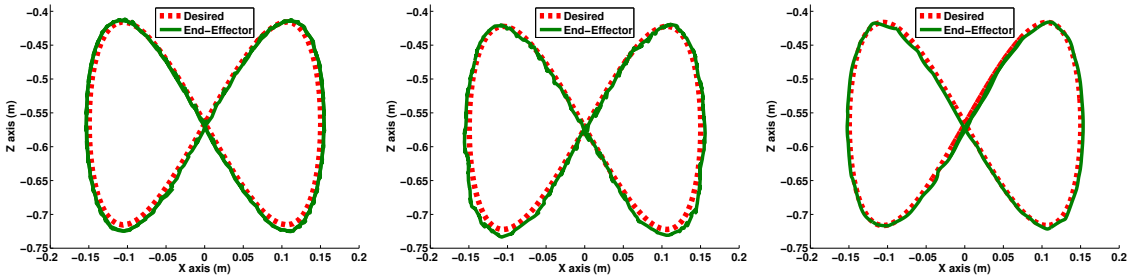
For the *"structured output Gaussian processes"* (SOGP) [Bócsi et al., 2011a], the energy function $E(\cdot, \cdot)$ is the a-posteriori mean function of a GP, i.e.

$$E(\mathbf{x}, \theta) \quad \stackrel{\circ}{=} \quad -\mu_{(\mathbf{x},\theta)}. \tag{14}$$

where the inputs are the same input data as before, and the outputs are $\mathbf{1}$ for all data. The posterior mean is:

$$\mu_{(\mathbf{x},\theta)} \quad = \quad \boldsymbol{k}_{(\mathbf{x},\theta)}^\top (\boldsymbol{K} + \sigma_0^2 \boldsymbol{I_m})^{-1} \mathbf{1},$$

where $\boldsymbol{K} \in \mathbb{R}^{m \times m}$ with $\boldsymbol{K}^{ij} = k((\mathbf{x}_i, \theta_i), (\mathbf{x}_j, \theta_j))$, $\boldsymbol{k}_{(\mathbf{x},\theta)} \in \mathbb{R}^{m \times 1}$ with $\boldsymbol{k}^i_{(\mathbf{x},\theta)} = k((\mathbf{x}_i, \theta_i), (\mathbf{x}, \theta))$, $k_{(\mathbf{x},\theta)(\mathbf{x},\theta)} = k((\mathbf{x}, \theta), (\mathbf{x}, \theta))$, $\boldsymbol{I_m}$ is the identity matrix, σ_0^2 is the noise variance, and $\mathbf{1}$ is the vector of ones of size m.



(a) JKSE. (b) SOGP. (c) FWGP.

Figure 2: Result of the tracking of the figure eight with the off-line method. We see god performance for all models.

The third approach, entitled "*forward Gaussian process modelling*" (FWGP) [Bócsi et al., 2012], is based on the observation that the direct kinematic model – denoted as $f(\cdot)$ – is easier to model as its inverse. We therefore build an energy function such that it depends explicitly on the direct kinematics. Using this knowledge, the energy function is defined using the Euclidean distance between the anticipated and the end-effector position, *i.e.*,

$$E(\mathbf{x}, \theta) \quad \stackrel{\circ}{=} \quad \|\mathbf{x} - f(\theta)\|^2. \tag{15}$$

We model the direct kinematics function with a GP. Given the inputs $\mathcal{D} = \{(\theta_i, \mathbf{x}_i)\}_{i=1}^m$ with inputs θ_i and outputs $\mathbf{x}_i$, the prediction for a new θ is a r.v. with a Gaussian distribution, having mean μ_θ:

$$\mu_\theta \quad = \quad \sum_{i=1}^{m} \alpha^i k(\theta, \theta_i) = \boldsymbol{k}_\theta^\top \boldsymbol{\alpha}, \tag{16}$$

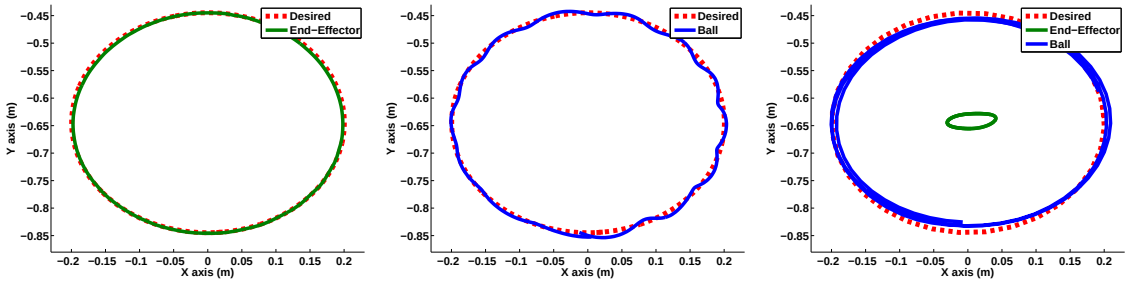
where $k_{\theta\theta} = k(\theta, \theta)$, $\boldsymbol{k}_\theta \in \mathbb{R}^{m\times 1}$ is a vector with elements $\boldsymbol{k}_\theta^i = k(\theta, \theta_i)$ şi $\boldsymbol{K} \in \mathbb{R}^{m\times m}$ is a matrix with elements $\boldsymbol{K}^{ij} = k(\theta_i, \theta_j)$. The function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a valid kernel function and $\boldsymbol{\alpha} \in \mathbb{R}^{m\times 1}$, where $\boldsymbol{\alpha} = (\boldsymbol{K} + \boldsymbol{I}_m \sigma_0^2)^{-1}\mathbf{x}$ are the parameters of the posterior GP. We used the posterior mean to predict the direct kinematic model from eq. (16), *i.e.*, $f(\theta) = \mu_\theta$. The gradients for all of the energy functions mentioned above are analytical, therefore the gradient search for parameters can be efficiently made.

3.3 Experiments

We evaluated and compared empirically the presented methods in the context of tracking control: we applied our algorithm for inferring the inverse kinematics of the Barrett WAM robot [Bócsi et al., 2011b] and for tracking the figure eight. The results of the experiments are presented in Fig. 2, where we can see that the precision of the algorithms is satisfactory. The experiments also show that the FWGP algorithm leads to a slightly better solution when compared to the JKSE or SOGP that ultimately will lead to the precision of the analytical controller. This is a slightly surprising result, considering the amount of data needed to be processed by each algorithm: JKSE required 8456 points, SOGP 200, whilst FWGP was based only on 31 points.

In a second experiment we increased the complexity of the problem: with a rope (of length 20 cm) we attached a ball to the original end-effector of the robot and we aimed to track with this ball. The oscillations of the ball resulted in a non-linear control problem. Using the settings above we repeated the tracking experiments, again the goal was to track a horizontal 20 cm circle, as in Fig. 3a. We repeated the experiments in two settings: (1) with slowly moving target (a complete rotation was made in 24 secs) and (2) with fast moving target (a complete rotation made in 0.62 secs).

In the first – slow motion – case, with the FWGP model we learned to move the ball on the circle, whilst the end-effector of the robot was also moving on the circle. We mention that the results from Figs 3a and 3b required four minutes of interaction time and the inferred GP model contained $20 - -25$ points. For the second – fast motion – case, the end-effector and the ball both moved on circular trajectories, with the ball on the tracked trajectory, as in Fig. 3c. In this case, after 20 minutes of running time, the GP model was using $13 - 15$ points. We emphasise that all experiments were using the same parameter setting, therefore we can conclude that the adaptive behaviour of the model is weakly dependent on the GP hyper-parameters. The fact that the extra DoF



(a) End-effector trajectory for slow target. (b) Ball trajectory for slow target. (c) Ball and end-effector trajectories for fast target.

Figure 3: Robot tracking control using a simulated Barrett WAM unit with ball attached to the end-effector.

was incorporated in the Gaussian process is a success. The visualisation of the experiments is available at the following links:

- http://www.youtube.com/watch?v=o-ib-XNJVaM,
- https://www.youtube.com/watch?v=q4D7rQTbikE,
- http://www.youtube.com/watch?v=nRyWLX8GXq0

3.4 Learning models using transfer learning

The motivation to using transfer learning is the human behaviour [Bócsi et al., 2013], a fundamental difference being that – opposite to human learning – in robotic learning there is no a-priori knowledge about the world. We can say thus that – despite being a completely new task – the human actors can cope better with completely novel tasks since they can "transfer" their past experiences accumulated from performing different but similar tasks in the past.

3.4.1 Transfer learning methods in robotics

The aim of this research was to improve – possibly speed up – the learning process of a robotic model when we assume the existence of information encoded from past experiments. We envisage that these data can be used as additional information in learning the new task. Therefore we define a *source task* – the information from the old experiment – and a *target task* – the new problem that is difficult to solve. We assume that the source task was trained using a set of data $\mathcal{D}^s = \{(\theta_i^s, \mathbf{x}_i^s)\}_{i=1}^N$, where N is the size of the data-set, and that the target task uses the dataset $\mathcal{D}^t = \{(\theta_i^t, \mathbf{x}_i^t)\}_{i=1}^K$, where K is the size of this new set.

As a first step, using the *"principal component analysis"* (PCA), we reduce the dimensionality of both sets to the same common – latent – dimension. First we centre the data, *i.e.* we subtract the mean value from each data-set, then divide by the respective variances, arriving to:

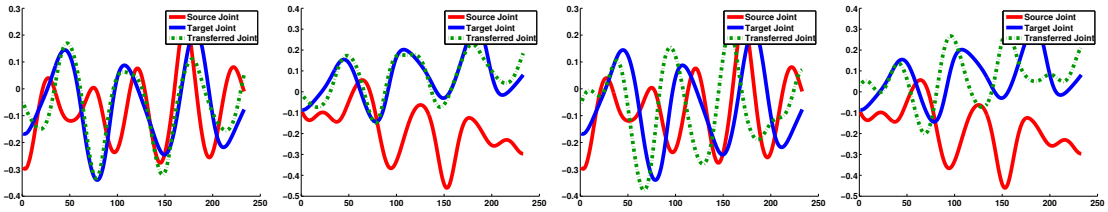
$$\begin{aligned} \mathbf{s} &= \mathbf{B}_s(\mathbf{d}^s - \mu_s) \\ \mathbf{t} &= \mathbf{B}_t(\mathbf{d}^t - \mu_t), \end{aligned}$$

where $\mathbf{d}^s \in \mathcal{D}^s$ and $\mathbf{d}^t \in \mathcal{D}^t$ are the data from the target and source tasks respectively, and $\mathbf{t} \in \mathbf{M}^t$ and $\mathbf{s} \in \mathbf{M}^s$ are the points in the respective reduced spaces, and the vectors $\mu_s = \mathbf{E}\{\mathcal{D}^s\}$ and $\mu_t = \mathbf{E}\{\mathcal{D}^t\}$ are the averages of the original data-sets. The matrices $\mathbf{B}_s$ and $\mathbf{B}_t$ are transformation matrices such that the transformed variance to the respective subspaces is maximal (for details, please consult [Bócsi et al., 2012]). We note that the dimension for both subspaces is the same, we denote it with J.

In the second step we model the optimal transformation from the two varieties as a linear function

$$f : \mathbf{M}^s \rightarrow \mathbf{M}^t, \quad f(\mathbf{s}) = \mathbf{A}\mathbf{s}, \tag{17}$$

where $\mathbf{A} \in \mathbb{R}^{J \times J}$ is the transformation matrix. In what follows we define two alignment methods: (1) we assume that there exists a *direct* correspondence between the data-points, *i.e.* that the data sets $\mathcal{D}^s$ and $\mathcal{D}^t$ are of the same size and that the points are paired – this set-up is for the case when the same task has been performed in two



(a) Joint 1. (b) Joint 3. (c) Joint 1. (d) Joint 3.

Figure 4: Results for direct and rough alignment

different environments. In the second scenario (2) we assume that there is no correspondence between the data points and that the sets have different cardinalities.

Results:

- In the first case – named **direct alignment** –, we minimise the quadratic loss of the transformation error: we wish to find the optimum linear transformation matrix $\mathbf{A}$ from eq. (17) such that on average the transformation to lead to the smallest quadratic error, *i.e.*,

$$\mathbf{A}^* = \underset{\mathbf{A}}{\operatorname{argmin}} \, \mathbf{E}\left\{(\mathbf{t} - \mathbf{A}\mathbf{s})^\top (\mathbf{t} - \mathbf{A}\mathbf{s})\right\}, \tag{18}$$

  and this system has the solution $\mathbf{A}^* = \mathbf{\Sigma}_{ss}^{-1}\mathbf{\Sigma}_{ts}$, with $\mathbf{\Sigma}_{ss}$, $\mathbf{\Sigma}_{tt}$, and $\mathbf{\Sigma}_{ts}$ being the (co-)variance matrices of the respective quantities.
- In the second case – called **rough alignment** –, we minimise the *distance* between the distributions $\mathbf{M}^s$ and $\mathbf{M}^t$, where the sets are considered a *representation of the respective distributions*. In what follows we assume that the data distribution is Gaussian and therefore we are minimising distance between two Gaussian distributions, $p(\mathbf{M}^s)$ and $p(\mathbf{M}^t)$ respectively. Using the Kullback-Leibler "distance function", the distance of two Gaussians has an analytic form. The result of the minimisation is a matrix $\mathbf{A}$ solving the following equation:

$$\mathbf{\Sigma}_{tt} \quad = \quad \mathbf{A}\mathbf{\Sigma}_{ss}\mathbf{A}^\top.$$

  The above equation is *quadratic in* $\mathbf{A}$ and therefore the solution is not unique. A solution is found by using the eigenvalue decomposition of the covariance matrices [Trefethen and Bau, 1997]:

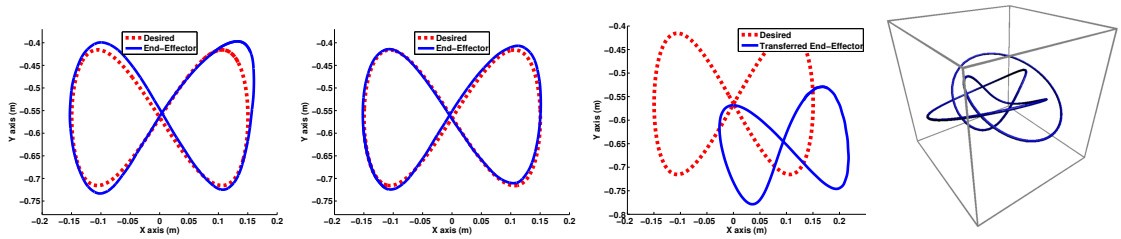
$$\mathbf{A} \quad = \quad \mathbf{U}_t\mathbf{\Lambda}_t^{1/2}\mathbf{\Lambda}_s^{-1/2}\mathbf{U}_s^\top,$$

  where $\mathbf{U}_s$ and $\mathbf{U}_t$ are the eigenvectors of $\mathbf{\Sigma}_{ss}$ and $\mathbf{\Sigma}_{tt}$, and $\mathbf{\Lambda}_s$ and $\mathbf{\Lambda}_t$ are diagonal matrices with the positive eigenvalues on the diagonal.

3.4.2 Experiments

We performed experiments with different robots, tot highlight the following two important characteristics of our method: (1) the information loss induced by the dimensionality reduction is not significant (2) the expressive power of the linear function is sufficient to obtain an efficient alignment.

The experiments were performed on a simulated Sarcos Master robotic arm with eight degrees of freedom. We also used a Barret WAM robotic arm with seven degrees of freedom, also in a simulated environment. In both control problems the goal was to learn to track the trefoil knot, as in Fig. 5d, with the end effector of the robotic arm. To collect training data we used the analytic model of the robot arm, and both the source and the target tasks were to track a trefoil knot with the robotic arm's end effector. We used the analytical controllers of the robots to collect data for both tasks, and running each task with the same speed for one minute, we had data points with direct correspondence. We applied the direct correspondence method on this data-set. Figure 4a and Figure 4b show that after estimating $\mathbf{A}$ with Equation (18), we could transfer the data points from the source data-set (red) to the target data-set (blue) with good efficiency (dashed green). To see the performance without direct correspondence but only distribution minimization, we applied the rough alignment method to this data-set. Results are presented in Figure 4c and we see that the transformation is not accurate, but nevertheless it captures the correct mean and variance of the transferred data, as it was expected.



(a) After data transfer *without* further learning.

(b) After data transfer *with* further learning.

(c) Transfer from the Master arm to the Barrett arm.

(d) Trefoil knot.

Figure 5: Tracking control results after three seconds of on-line learning, followed by using the transfer learning algorithm

In a subsequent experiment we directly transferred a task from the Master arm to the Barret arm using the transition matrix obtained from the previous data-sets. We draw a figure eight with the Master arm (using the analytical controller) that was placed inside the space defined by the trefoil knot. After transforming the joint-space trajectories and following the transformed trajectories with the Barrett arm, the figure eight presented in 5c has been obtained. Note that showing a desired figure eight in Fig. 5c may be misleading since there is no ground truth of task transformation. We defined the desired figure eight as the figure tracked by the analytical controller of the Barrett arm with the same initial posture as the trefoil knot. This intuitive definition is based on the principle that the data transfer must incorporate only translation, rotation and scaling, and no deformation of the tracked figures.

In the last experiment, we used the same robot two architecture for the source task and for the target task as in the previous experiment. We used the source data-set from the previous experiment (the trefoil knot). The target task was to speed-up the forward kinematics model learning of the Barret arm.

The forward kinematics model has been approximated with sparse on-line Gaussian processes and used to perform task-space control. Without transfer learning it takes from 20 seconds to four minutes to learn this model on-line.

After performing quasi-random movements for three seconds with the Barrett arm, we applied the distribution alignment approach. We stopped the learning process after the three seconds of burn-in period and used the transferred points to further train the forward kinematics model. Figure 5a shows the figure eight as a result. The shape eight is not perfect, however, we needed only three seconds of learning and the transfer algorithm. We repeated the experiment but now the learning process was not stopped after three seconds, only the Master arm data-set has been used to gain additional training samples. Figure 5b shows that if learning is not stopped an accurate figure eight tracking is achievable after three seconds of learning and the transfer algorithm.

On November 23 2012 Botond Bócsi successfully defended his PhD thesis entitled *"Model Learning for Robot Control"*. The thesis was largely based on the above research which was performed within the present project.

4 Input noise models in regression

Conventional modelling only assumes output noise and input noise is neglected and we analysed regression in the presence of the input noise and we choose as regressors the non-parametric Gaussian processes (GPs). Given the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, *i.e.*,

$$y = \tilde{y} + \epsilon_y \qquad\qquad \mathbf{x} = \tilde{\mathbf{x}} + \epsilon_x,$$

where y is the observed noisy label, $\tilde{y}$ is the true label, $\epsilon_y \sim \mathcal{N}(0, \sigma_y^2)$ is the output noise process , $\mathbf{x}$ is the observed input, $\tilde{\mathbf{x}}$ is the true input, and $\epsilon_x \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ is the input noise process.

4.1 Hessian corrected input noise models

As a first approach in [Bócsi and Csató, 2013] we used the Taylor series expansion of the regression function. The noise process is additive and zero mean Gaussian, the correct regression function can be expressed in the

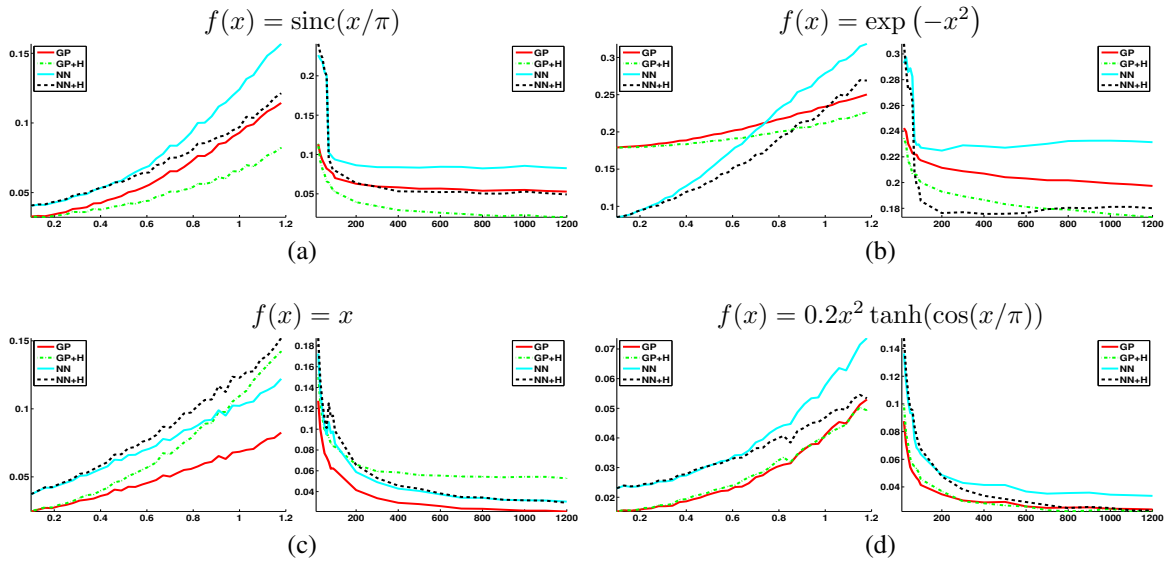


Figure 6: The performance of the SIMEX method evaluated on artificial data

Data-set name	GP	GP+H	NN	NN+H	Noise (σ)	Dim.	Set size
Boston housing ($1000s)	2.2271	2.2271	3.4819	3.4819	0.1	13	506
Concrete (MPa)	4.1281	4.1280	6.1865	6.1864	1	8	1030
Barrett WAM 1 (mm)	2.9272	2.9242	2.8726	2.8488	0.01	4	1000
Barrett WAM 2 (mm)	13.707	13.731	12.439	9.3524	0.1	4	1000
CPU performance	17.762	17.763	16.846	16.846	5	7	209
Auto MPG (mpg)	4.0301	4.0322	2.2990	2.2988	3	7	301

Table 1: Results of experiments on real world data-sets

following form:

$$f(\mathbf{x}) \quad = \quad g(\mathbf{x}) - \frac{1}{2}\sigma^\top H_g(\mathbf{x})\sigma, \tag{19}$$

where σ is the variance of the input noise $g()$ is an arbitrary regression function, $H_g()$ is the Hessian of the function and $f()$ is the true regression function.

We performed experiments on one and multi-dimensional functions. As a regression function we used Gaussian processes (GP), neural networks (NN) and their Hessian corrected versions (+H). Figure 6 shows the results of our experiments on the artificial data. It can be seen that the Hessian corrected versions are almost always better than the standard GP. Moreover the improvements are more significant when we increase the variance of the input noise. Table 1 summarizes the results on different multidimensional datasets. The improvements induced by our method are not as significant as in the previous case. A possible explanation would be that in high dimensions we prefer rather linear models to avoid over-fitting. The true Hessian of a linear model is zero thus, the addition of the approximated Hessian does not improve the prediction. This is generally true for multidimensional data.

4.2 Simulation extrapolation for Gaussian Processes

The base idea of our second approach, presented in [Bócsi et al., 2014b] is to *generate* a number of input data sets with controlled input noise variance.

$$\varphi_\lambda(x) \sim f(x + \sqrt{1+\lambda}\,\epsilon_x), \tag{20}$$

where $\varphi_\lambda(x)$ is the regression function based on input data where input noise with λ variance was added. $\varphi_0(x)$ is the original function where no additional input noise was added. This way we can measure the effect of the added input noise with different variances and extrapolate to a location where no input noise is present, $\varphi_{-1}(x)$:

$$\varphi_3, \varphi_2, \varphi_1, \varphi_0 \quad \rightarrow \quad \varphi_{-1}. \tag{21}$$

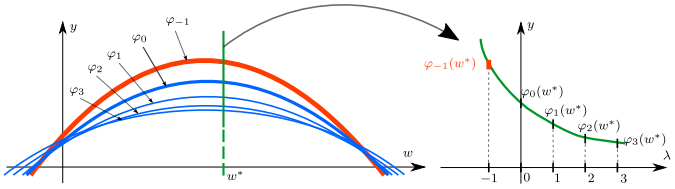


Figure 7: Illustration of probabilistic simulation extrapolation. (left) φ_λ for different $\lambda = \{0, 1, 2, 3\}$ values drawn with thin blue lines; φ_{-1} is the point-wise extrapolated posterior mean drawn with red. (right) the individual extrapolation $\varphi_\lambda(w^*)$ – as a function of λ – for a test point w^*.

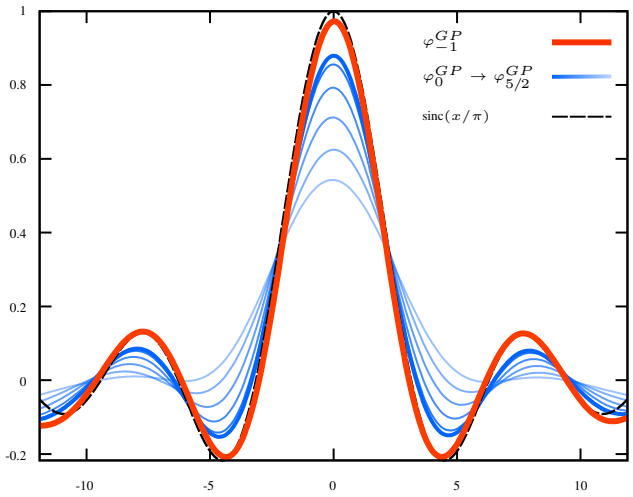


Figure 8: Results for the function $\text{sinc}(x/\pi)$ with input noise with variance $\Sigma = 0.8$. The black curve represents the real function $\text{sinc}(\cdot)$. The blue lines $\varphi_0^{GP} \rightarrow \varphi_{5/2}^{GP}$ represent the regression functions when input noise with different variances was added. The red line is the extrapolated prediction φ_{-1}^{GP}.

Figure 7 illustrates the basic principle of the simulation extrapolation method.

In all our experiments we used Gaussian processes for the regression function. As a consequence of this choice, we can employ the probabilistic nature of GPs and obtain a distribution over regression functions in the end. Figure 8 shows the improvements made by our method.

5 Hashing methods for fast nearest neighbour search

In recent years several algorithms were proposed for fast approximate nearest-neighbour search, providing sub-linear search times for a query. Introduced by Cover and Hart [1967], the k-nearest neighbour algorithm is one of the most successful and most investigated methods in the machine learning community. However, for large datasets it has a serious drawback: it must go through all the *training* data to find the nearest neighbours. This implies a linear time complexity for a single point. The method of k-nearest neighbours can be used in machine learning to label an unknown point based on the majority label among the neighbours. Considering the nearest neighbours of a point, however, is not used only in machine learning; the properties of the neighbours can be beneficially utilized to infer supposedly valid facts about the point in question. Without pretension to completeness, we can mention important applications in information retrieval, computer vision, coding theory, recommendation systems, computational geometry; for details see *e.g.* [Rajaraman and Ullman, 2012]. Therefore, speeding up the search for nearest neighbours we consider a task of great importance.

Learned binary embeddings for large datasets, where nearest-neighbours of a given point needed to be found, are efficient and powerful tools for indexing these sets. These embeddings are designed to approximately preserve similarities in the embedding Hamming space. The beneficial properties of the codewords make possible to use Hamming distance computations for nearest-neighbour search, by which the searching process can be substantially sped up: to compute the Hamming distance of two vectors, a XOR operation has to be performed between the

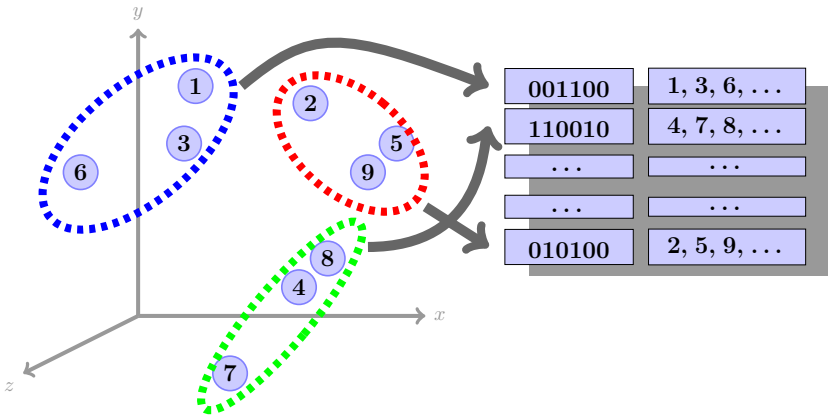


Figure 9: Schematic representation of hashing. In optimal case nearby points, defined by a metric, are mapped to the same hash bucket or nearby buckets having a small Hamming distance.

vectors and the resulting set bits have to be counted. We differentiate between two parts of fast nearest-neighbour search with binary embeddings. The first part consists of generating the binary codes, and the second part is the actual searching process. One can also distinguish between unsupervised, supervised and semi-supervised codeword generation methods, based on the information they use to obtain the embedding [Grauman and Fergus, 2013]. Unsupervised methods use only the information carried by the points themselves. Additional information can be given to supervised methods in form of label information as in a supervised machine learning setting, as well as giving the neighbourhood list for a subset of points, or as paired constraints – defining the points which ought or ought not cluster together. Finally, semi-supervised methods exploit the supervised information, besides which other clustering or regularization approaches are used.

Locality-sensitive hashing with hyperplanes [Charikar, 2002] is perhaps the most popular unsupervised hashing method, generating binary codewords by taking random vectors as normal vectors of separating hyperplanes. The hash function is the sign of the dot product between the data point and the random vectors. In [Kulis and Grauman, 2009] it was extended to support arbitrary kernels by using a clever method to generate random vectors in the feature space. In Figure 9. the schematic representation of hashing is shown.

5.1 Kernel locality-sensitive hashing using pre-images

In [Bodó and Csató, 2012] we proposed a method for improving the codeword generation algorithm in kernelised *locality-sensitive hashing* (kLSH) introduced in [Kulis and Grauman, 2009]. In order a function $h : \mathcal{X} \to \mathcal{V}$ to be called a locality-sensitive hash function, it must fulfil the following conditions:

1. The probability of collision for nearby points is $P(h(\mathbf{x}) = h(\mathbf{z})) \geq p_1$, for $\|\mathbf{x} - \mathbf{z}\| \leq R_1$;
2. The probability of collision for distant points is $P(h(\mathbf{x}) = h(\mathbf{z})) \leq p_2$, for $\|\mathbf{x} - \mathbf{z}\| \geq R_2 = (1 + \epsilon)R_1$;
3. $p_1 > p_2$.

One such hash function is the dot-product based function of Charikar [2002], which uses a randomly generated hyperplane with normal vector $\mathbf{r}$. The hash keys are binary vectors of length L, where each bit assigned to a random hyperplane is generated by the function:

$$h_{\mathbf{r}}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{r}'\mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} .$$

A hash key for a data point is composed of a sequence of L such random hash functions, $[h_{\mathbf{r}_1}(\mathbf{x}), \ldots, h_{\mathbf{r}_L}(\mathbf{x})]$. In [Kulis and Grauman, 2009] the authors solved the problem of generating random hyperplanes in the feature space induced by the kernel using the central limit theorem, the spectral theorem and the "kernel trick". The problem with this formulation is that for a new (test) point several kernel calculations are needed. We reduced this number to one by computing the pre-images of the random feature space Gaussian vectors, using a method that does not

Algorithm 1 Linear Spectral Hashing

Input: Data $\mathbf{X}$, $\mathbf{D} = \mathrm{diag}(\mathbf{X}'\mathbf{X}\mathbf{1})$, eigenvectors $\mathbf{V}$ and $\mathbf{U}$, and a point $\mathbf{x}$

Output: Code $f(\mathbf{x})$

1: Computing the eigenvalues of $\mathbf{D}^{-1/2}\mathbf{X}'\mathbf{X}\mathbf{D}^{-1/2}$ or $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$, starting with the second largest one:

$$[\mathbf{v}_2,\ \mathbf{v}_3,\ \ldots,\ \mathbf{v}_{r+1}] \quad \text{or}$$
$$[\mathbf{u}_2,\ \mathbf{u}_3,\ \ldots,\ \mathbf{u}_{r+1}]$$

2: Define the function:

$$f_i(\mathbf{x}) = \mathbf{x}'\mathbf{u}_i = \mathbf{x}'\mathbf{X}\mathbf{D}^{-1/2}\mathbf{v}_i e_i^{-1/2}, \quad i = 2, \ldots, r+1$$

where e_i, $i = 2, \ldots, r+1$ denotes the eigenvectors of $\mathbf{X}\mathbf{D}^{-1}\mathbf{X}'$.

3: The code for $\mathbf{x}$ is as follows:

$$f(\mathbf{x}) = [f_2(\mathbf{x}),\ f_3(\mathbf{x}),\ \ldots,\ f_{r+1}(\mathbf{x})]'$$

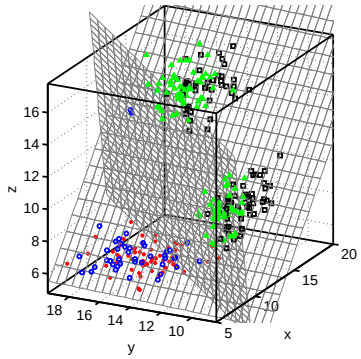


Figure 10: Points sampled from three multivariate Gaussians and their hashing to two dimensions performed using linear spectral hashing.

require *additional* computations. Our formula for calculating the jth component ($j = 1, 2, \ldots, d$) of the pre-image of the ith random feature-space vector ($i = 1, 2, \ldots, L$) becomes:

$$\frac{1}{\sqrt{t}}\mathbf{X}_{j\cdot}\mathbf{K}^{-\frac{3}{2}}\sum_{k\in\mathcal{S}_i}\mathbf{k}_{\mathbf{x}_k}$$

where p is the size of the sample and t is the smaller sample size, sampled from the p points. There will be L such samples – that is of size t – the indices of which are stored in $\mathcal{S}_i$, $i = 1, \ldots, L$. Generating the codeword of a new point is done simply by calculating the kernel values (for each bit) of the unseen point with the generated vectors. This work was presented at the IJCNN 2012 conference and the paper appeared in the conference proceedings.

5.2 Linear spectral hashing

In [Bodó and Csató, 2013] and [Bodó and Csató, 2014a] we have proposed a linear method for spectral hashing – a popular and successful method in approximate k-nearest neighbour search. In spectral hashing generating the codewords for the training points is simple, however, generalization for unseen points becomes difficult. For this, we proposed a linear method by which the codeword generation consists of taking the dot product of the new point with a set of generated vectors.

Based on spectral hashing [Weiss et al., 2008] and the work of [Rahimi and Recht, 2004] we traced the problem back to spectral clustering, that is clustering with maximum-margin hyperplanes. We also stated and proved the following:

Statement 1 *Linear spectral hashing finds a set of orthogonal vectors that are normal vectors of maximum-margin separating hyperplanes.*

$$\mathbf{M} = \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

Figure 11: Two-versus-rest scheme (for $k = 4$ classes)

Connections to spectral clustering and Laplacian eigenmaps were also discussed in the paper. The algorithm is presented in Algorithm 1 and the resulting separating hyperplanes for some points sampled from multivariate Gaussians are shown in Figure 10. This work was presented at the ESANN 2013 conference and the short version of the paper appeared in the conference proceedings, after which we were invited to publish our work in the Neurocomputing journal. The article appeared in October 2014.

6 Semi-supervised learning

Semi-supervised learning (SSL) is a special case of classification; it is halfway between classification and clustering. In semi-supervised learning the training data is augmented by a set of unlabelled data samples, that is we have $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \mathbf{x}_{\ell+1}, \mathbf{x}_{\ell+2}, \ldots, \mathbf{x}_{\ell+u}\}$, where usually there are far less labelled data than unlabelled ones, *i.e.* $\ell \ll u$. We denote by $N = \ell + u$ the size of the entire data set. Semi-supervised learning is the problem of assigning labels to the unlabelled samples of the data set using the information provided by both the labelled and the unlabelled data. SSL techniques can also be used in the conventional classification setting, when simply the labels of unknown data points are needed without any extra knowledge. In these situations the test points play the role of the unlabelled points, assumed they are drawn from the same distribution as the training data.

In the semi-supervised case the inputs $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are separated from their corresponding labels, and we can say that – in mathematical terms – the unlabelled set improves on the estimation of the density function of the inputs. With additional assumptions about the nature of the whole data set, one can improve the performance of a specific algorithm by incorporating this extra knowledge into the database.

6.1 Augmented hashing for semi-supervised scenarios

In [Bodó and Csató, 2014b] we have proposed a general framework for augmenting hash codewords obtained by unsupervised techniques. We assumed that we are given some class labels for the training data, thus creating a semi-supervised learning scenario. We proposed to extend the codewords using error correcting output coding (ECOC) [Dietterich and Bakiri, 1995] with semi-supervised classifiers. The outline of the method is the following:

- form a *data-dependent* error correcting output coding matrix
- train *semi-supervised* classifiers that will generate the second part of the hash code

A coding matrix is a $k \times s$ matrix defined over the set $\{-1, 1\}$, where k denotes the number of classes and s is the codeword length. For each column of the coding matrix a binary classifier is trained, splitting the training data into two sets – of positive and negative examples – based on the actual column.

Error correcting output coding is used in machine learning to perform multi-class classification using binary classifiers. At prediction each of the s classifiers outputs a sign, the closest codeword to the resulting vector is looked up in the coding matrix, and the resulting class is output as the decision. Figure 11 shows the two-vs-rest matrix for 4 classes.

A *good* error correcting code means *good* row and column separation. If sufficient labelled data is present, one can define an optimization problem for finding such codes $(\mathbf{c}_i, i = 1, \ldots, k)$ [Zhang et al., 2009]:

$$\min_{\mathbf{C} \in \mathbb{R}^{k \times s}} \quad \sum_{i,j=1}^{k} a_{ij} \|\mathbf{c}_i - \mathbf{c}_j\|^2 \, (= \operatorname{tr}(\mathbf{C}'\mathbf{L}\mathbf{C}))$$
$$\text{s.t. } \mathbf{C}'\mathbf{1} = \mathbf{0}, \quad \mathbf{C}'\mathbf{C} = \mathbf{I}$$

An important question is how to calculate the $\mathbf{A} = (a_{ij})_{i,j=1,\ldots,k}$ *class similarities*. We used two approaches in the experiments:

Algorithm 2 Label propagation

1: **repeat**
2: $\mathbf{Y} = \mathbf{TY}$
3: (Row-normalize $\mathbf{Y}$.)
4: Clamp the labeled data.
5: **until** convergence

1. compute class centres and calculate similarities using dot products of the centres;
2. train $k(k-1)/2$ SVMs (for each class pair) and use the inverse of the margin of the separating hyperplane as similarity measure – this will reflect how well-separated the classes are:

$$\|\mathbf{w}\|^2 = \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

The base codeword generation method and the semi-supervised learning algorithm can be chosen arbitrarily. In our experiments we used linear spectral hashing for unsupervised codeword generation, and Laplacian regularized least squares [Belkin et al., 2006] and semi-supervised support vector machines [Sindhwani and Keerthi, 2006] for learning. This work was presented at the ESANN 2014 conference (and the proceedings).

6.2 Label propagation for semi-supervised learning

Label propagation is a *transductive graph-based* method for *semi-supervised* classification. It is transductive because the algorithm can predict the labels of the points included in the unlabelled learning dataset, it does not output an inductive classifier applicable for a new point.

In [Bodó and Csató, 2014c] we analyse two variants of the algorithm shown in Algorithm 2. The matrix $\mathbf{T}$ is an $N \times N$ *transition* matrix realizing the propagation of labels, and the two variants differ only in the selection of $\mathbf{T}$. We denote the similarity (or proximity) matrix by $\mathbf{W}$, $\mathbf{D} = \mathrm{diag}(\mathbf{W1})$ represents the diagonal degree matrix and $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$ is the transition probability matrix. In the first variant of the algorithm [Zhu and Ghahramani, 2002] $\mathbf{T} := \mathbf{P}'$, while the second variant [Zhu, 2005] chooses $\mathbf{T} := \mathbf{P}$. However the difference seems subtle, the outputs of the method – considering the formulae – differ:

$$\begin{aligned} \mathbf{Y}_U &= \mathbf{A}\mathbf{D}_L^{-1}\mathbf{Y}_L \\ \mathbf{Y}_U &= \mathbf{A}\mathbf{Y}_L \end{aligned}$$

This paper presents and compares the two algorithms – both theoretically and experimentally – and also tries to make a recommendation which variant to use. This work is currently sent for publication to Studia Universitatis Babeş–Bolyai, Series Informatica.

7 Conclusions and further research

To summarise, we appreciate that non-parametric methods can be used both in robotics and also in analysing data. Although the main drawback of these methods – the bad scaling with input data – is not eliminated, we can say that by allocating more resources – if one can do – then in general the performance of the non-parametric methods is better when compared to classical algorithms.

For reinforcement learning algorithms an important added characteristic is the *on-line* processing capability of the algorithm, therefore the results on Section 2.5.2 are important. We want to validate the fast on-line updates from eq. (11), for larger real-life robotic systems and to publish the results in scientific journals and conferences.

A second interesting new direction is the study of input noise models: we implemented input noise models based fully on Gaussian processes and studied *empirically* the properties of the method. We think it would be an interesting research direction to study the consistency of the method, establish theoretical limits to it, and later to apply simplifying assumptions that will lead to speed-ups in the computational time of the method.

Acknowledgements: we acknowledge the financial support from the Romanian agency UEFISCDI through grant *PN-II-RU-TE-2011-3-0278*.

References

B. Bócsi and L. Csató. Hessian corrected input noise models. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, volume 8131 of *LNCS*, pages 1–8, Sofia, Bulgaria, 2013. Springer. ISBN 978-3-642-40727-7. doi: 10.1007/978-3-642-40728-4_1.

B. Bócsi, L. Csató, and J. Peters. Structured output gaussian processes. Technical report, Babes-Bolyai University, 2011a. URL http://www.cs.ubbcluj.ro/~bboti/pubs/sogp_2011.pdf.

B. Bócsi, D. Nguyen-Tuong, L. Csató, B. Schoelkopf, and J. Peters. Learning inverse kinematics with structured prediction. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 698–703, San Francisco, USA, September 2011b. ISBN 978-1-61284-454-1. doi: 10.1109/IROS.2011.6094666.

B. Bócsi, P. Hennig, L. Csató, and J. Peters. Learning tracking control with forward models. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 259–264, St. Paul, MN, USA, 2012. ISBN 978-1-4673-1403-9. doi: 10.1109/ICRA.2012.6224831.

B. Bócsi, L. Csató, and J. Peters. Alignment-based transfer learning for robot models. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, Dallas, USA, 2013. doi: 10.1109/IJCNN.2013.6706721.

B. Bócsi, L. Csató, and J. Peters. Indirect robot model learning for tracking control. *Advanced Robotics*, 28(9): 589–599, 2014a. doi: 10.1080/01691864.2014.888371.

B. Bócsi, H. Jakab, and L. Csató. Simulation-extrapolation gaussian processes for input noise modeling. In *Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Romania, September 2014b.

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

B. Bócsi. *Model Learning for Robot Control*. PhD thesis, Babeş-Bolyai University of Cluj-Napoca, Faculty of Mathematics and Computer Science, 2012.

B. A. Bócsi, H. S. Jakab, and L. Csató. Simulation extrapolation gaussian processes for input noise modeling. In *16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014, Timisoara, Romania, September 21-25, 2014*, pages to–be–published, 2014.

Z. Bodó and L. Csató. Improving kernel locality-sensitive hashing using pre-images and bounds. In *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 2710–2717, 2012.

Z. Bodó and L. Csató. Linear spectral hashing. In *Proceedings of the 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 303–308, 2013.

Z. Bodó and L. Csató. Linear spectral hashing. *Neurocomputing*, 141:117–123, 2014a.

Z. Bodó and L. Csató. Augmented hashing for semi-supervised scenarios. In *Proceedings of the 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 53–58, 2014b.

Z. Bodó and L. Csató. A note on label propagation for semi-supervised learning. *Studia Universitatis Babeş–Bolyai, Series Informatica*, 2014c. (submitted).

S. J. Bradtke, A. G. Barto, and P. Kaelbling. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996.

M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13, 1967.

L. Csató and M. Opper. Sparse on-line Gaussian Processes. *Neural Computation*, 14(3):641–669, 2002.

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72 (7-9):1508–1524, 2009. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2008.12.019.

T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. of Artificial Intelligence Research*, 2:263–286, 1995.

M. Ghavamzadeh and Y. Engel. Bayesian policy gradient algorithms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *NIPS '07: Advances in Neural Information Processing Systems 19*, pages 457–464, Cambridge, MA, 2007. MIT Press.

K. Grauman and R. Fergus. Learning binary hash codes for large-scale image search. *Machine Learning for Computer Vision*, 411:49–87, 2013.

H. Jakab and L. Csató. Improving Gaussian process value function approximation in policy gradient algorithms. In T. Honkela, W. Duch, M. Girolami, and S. Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, volume 6792 of *Lecture Notes in Computer Science*, pages 221–228. Springer, 2011. ISBN 978-3-642-21737-1.

H. Jakab and L. Csató. Reinforcement learning with guided policy search using Gaussian processes. In *The 2012 International Joint Conference on Neural Networks (IJCNN),Brisbane, Australia, June 10-15, 2012*. IEEE, 2012. ISBN 978-1-4673-1488-6.

H. Jakab and L. Csató. Manifold-based non-parametric learning of action-value functions. In M. Verleysen, editor, *European Symposium on Artificial Neural Networks (ESANN)*, pages 579–585, Bruges, Belgium., 2012. UCL, KULeuven, (c)Ciaco. ISBN 978-2-87419-047-6.

H. S. Jakab. *Intelligent Models for Robotic Behavior, Decision Making and Environment Interaction*. PhD thesis, Joint PhD degree between: the Faculty of Computer Science, Eötvös Loránd University of Budapest, Hungary and Faculty of Mathematics and Computer Science, Babeş-Bolyai University of Cluj-Napoca, Romania, 2012.

H. S. Jakab and L. Csató. Novel feature selection and kernel-based value approximation method for reinforcement learning. In *Artificial Neural Networks and Machine Learning – ICANN*, volume 8131 of *Lecture Notes in Computer Science*, pages 170–177. Springer, 2013. ISBN 978-3-642-407277.

H. S. Jakab and L. Csató. Sparse approximations to value functions in reinforcement learning. In P. Koprinkova-Hristova, editor, *Springer Series in Bio-/Neuroinformatics, Artificial Neural Networks*, volume 4, pages 295–315. Springer International Publishing Switzerland, 2014. doi: 10.1007/978-3-319-09903-3_14.

B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137. IEEE, 2009.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

A. Rahimi and B. Recht. Clustering with normalized cuts is clustering with a hyperplane. In *Statistical Learning in Computer Vision*, 2004.

A. Rajaraman and J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2012.

C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear SVMs. In *SIGIR*, pages 477–484. ACM, 2006.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

L. N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. ISBN 0-89871-361-7.

Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760. MIT Press, 2008.

X. Zhang, L. Liang, and H.-Y. Shum. Spectral error correcting output codes for efficient multiclass recognition. In *ICCV*, pages 1111–1118. IEEE, 2009.

X. Zhu. *Semi-supervised learning with graphs*. PhD thesis, Pittsburgh, PA, USA, 2005.

X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002. URL citeseer.ist.psu.edu/zhu02learning.html.