Kernel and Moment based Prediction and Planning

Applications to Robotics and Natural Language Processing

CMU-RI-TR-18-13
February 19, 2018

Zita Marinho

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

Geoffrey J. Gordon, Chair
Siddhartha S. Srinivasa, CMU/University of Washington
André F. T. Martins, Unbabel/IT, Instituto Superior Técnico
João P. Costeira, Instituto Superior Técnico
Shay B. Cohen, University of Edinburgh
Mathew T. Mason, Carnegie Mellon University

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was supported by the Portuguese Foundation for Science and Technology under grant SFRH/BD/52015/2012.

Keywords— Sequence prediction, Method of Moments, Semi-supervised learning, Predictive State Representations, Reinforcement-Learning

Abstract

This thesis focuses on moment and kernel-based methods for applications in Robotics and Natural Language Processing. Kernel and moment-based learning leverage information about correlated data that allow the design of compact representations and efficient learning algorithms.

We explore kernel algorithms for planning by leveraging inherently continuous properties of reproducing kernel Hilbert spaces. We introduce a kernel based robot motion planner based on gradient optimization, in a space of smooth trajectories, a reproducing kernel Hilbert space. We further study a kernel-based approach in the context of prediction, for learning a generative model, and in the context of planning for learning to interact with a controlled process.

Our work on moment-based learning can be decomposed into two main branches: spectral techniques and anchor-based methods.

Spectral learning describes a more expressive model, which implicitly uses hidden state variables. We use it as a means to obtain a more expressive predictive model that we can use to learn to control an interactive agent, in the context of reinforcement learning. We propose a combination of predictive representations with deep reinforcement learning to produce a recurrent network that is able to learn continuous policies under partial observability. We introduce an efficient end-to-end learning algorithm that is able to maximize cumulative reward while minimizing prediction error. We apply this approach to several continuous observation and action environments.

Anchor learning, on the other hand, provides an explicit form of representing state variables, by relating states to unambiguous observations. We rely on anchor-based techniques to provide a form of explicitly recovering the model parameters, in particular when states have a discrete representation such as in many Natural Language Processing tasks. This family of methods provides an easier form of integrating supervised information during the learning process. We apply anchor-based algorithms on word labelling tasks in Natural Language Processing, namely semi-supervised part-of-speech tagging where annotations are learned from a large amount of raw text and a small annotated corpus.

Acknowledgements

I would like to thank the many people that encouraged me and contributed during the period of my PhD. First of all I would like to thank all of my three academic guiding stars: André Martins, Geoff Gordon and Siddhartha Srinivasa for their incredible support and patience teaching me throughout the PhD. Thanks to their support and guidance I was able to learn from a rich and diverse set of research subjects. André was a very attentive and incredible advisor and I am very grateful to him for always directing me towards the right direction. He was my NLP guiding star. I would like to thank Geoff, my Machine Learning guiding star, for always being present throughout these years and for his insightful advice on many research and practical problems. And lastly, I would like to thank Sidd, my Robotics guiding star, for being truly inspiring and for providing all the support i needed in the PhD, thanks to him I was able to be a part of a inspiring and thriving lab, the Personal Robotics Lab, I would like to thank Mike Koval and Anca Dragan for collaborating and working with me during the PhD. I was able to learn a lot from their advice and experience.

From the Geoff's side of the family, I am very grateful to Byron Boots for his guidance in the RKHS project and for teaching me about PSRs, and to Ahmed Hefny for collaborating with me on the RPSP paper, his contribution was invaluable to the completion of this work. I am also very thankful to all my other collaborators: Wen Sun, Arun Byravan, and in particular to Gilwoo Lee for her work on learning PWS hybrid systems.

Thank you also to Shay Cohen and Matt Mason for being part of my committee for their feedback and discussions on the thesis topics. I would like to thank my academic grandfather Noah Smith for welcoming me in his research group and taking the time to teach my independent study at CMU.

Thank you to Prof. João Paulo Costeira for his enormous support and for welcoming me in the SIPG lab, and also all my colleagues at ISR in Lisbon: Shanghang, Qiwei, João Carvalho, João Saúde, Jayakorn, Beatriz, Sérgio, Cláudia and Manuel for the great environment, which I already miss terribly, and also Sabina and Susana for their immense support and friendship. I would like to say a special thank you to Ada Zhang, Renato Negrinho, João Saúde and Alessandro Giordano, for being great friends and housemates. Thank you also to Priberam for hosting me in my first years of my PhD.

This thesis could not have been possible without the support of the Fundação para a Ciência e Tecnologia (FCT) and the Information and Communication Technology Institute (ICTI) through the CMU/Portugal program (SFRH/BD/ 52015/2012). This work has also been partially supported by the European Union under H2020 project SUMMA, grant 688139, and by FCT, through contracts UID/EEA/50008/2013, through the LearnBig project (PTDC/EEISII/7092/2014), and the GoLocal project (grant CMUPERI/TIC/0046/2014).

Personally I would like to thank all my friends and family for their incredible support through the best of times and through the worst of times. Andreia and Sofia you have always had a sympathetic ear. Thank you Gil for your support, and André Moita, Cristina and Leonardo for all your advice, and Wang Ling for your great encouragement.

I would like to dedicate this thesis to my family, for which I am deeply grateful. And lastly, thank you Ana for being my sister, without your help none of this would have been possible.

Contents

1
Introduction

In this chapter, we discus the core aspects of moment and kernel-based learning and summarize the main contributions made in this thesis.

Method of Moments

Many problems in machine learning attempt to find a compact model that is capable of explaining complex behaviours from observable quantities. We focus mostly on sequential observations, where time dependence plays a crucial role in the evolution of these events.

Latent variable learning provides a compact representation for learning these complex, high dimensional data in a structured form. However, learning latent variables is often difficult, in part because of the non-convexity of the likelihood function (Sun, 2014; Terwijn, 2002). Yet, the main challenge is associated with estimating the latent (unobserved) states, which need to be estimated indirectly by looking at correlations among observations (examples in Figure 1.2.

The method of moments (MoM) provides an alternative form for explaining high dimensional and complex data, based on classical statistics and probability theory. MoM dates back to Pearson's solution for curve fitting problems, *i.e.*, for finding parameters that fit a mixture of two Gaussian distributions (Pearson, 1894). MoM's estimation relies on the idea that empirical moments are "natural" estimators of population moments. In essence, learning a model via the MoM comes down to estimating model parameters that represent distributions whose moments are in agreement with sample moments observed in the data.

matching moments vs. maximum likelihood

Moment-based algorithms differ from likelihood-based methods in that they attempt to recover parameters based on moment matching instead of maximizing likelihood which leads to intractable optimization. MoMs are also faster to compute and usually require more data to build good empirical estimates. Recent work applied MoM to different latent variable models, and demonstrated better convergence

properties over the most commonly applied likelihood approach, Expectation Maximization (EM), both in terms of runtime and consistency (Hsu et al., 2009). MoM also provides asymptotically consistent estimates, in the sense that the method yields the correct model parameters as the size of training sequences goes to infinity. In many applications unlabeled data is readily available, which makes moment-based algorithms a preferable choice under unsupervised settings, or even when labeled data is scarce. These methods yield (asymptotically) statistically optimal solutions, and can be computed efficiently.

MoM's intuitive construction together with statistical consistency guarantees makes this class of estimators an increasingly attractive choice in a wide range of research areas including: System Identification (Van Overschee et al., 1996), Video Modeling (Blaschko et al., 2008), Robotics (Siddiqi et al., 2007), and Natural Language Processing (Cohen et al., 2012; Parikh et al., 2011; Balle et al., 2011). Since Pearson's work, MoM has been studied and adapted for a variety of problems. "Identifiability" was proven to be theoretically possible for mixture of Gaussian distributions—where two mixture of different Gaussians have two distinct probability distributions (Teicher, 1961). This property ensures the parameters estimated via MoM will be uniquely identified (Chang, 1996). Over the last decade, the machine learning community has contributed with new theoretical advances in MoM estimation: starting with Dasgupta et al. (2000) and Arora et al. (2001) and Vempala et al. (2004) who designed polynomial time algorithms based on moments estimation. Later Hsu et al. (2009) provided sample complexity guarantees for different hidden variable models, Anandkumar et al. (2012a) provided an algorithm to learn general hidden variable models and Anandkumar et al. (2012b) and Arora et al. (2013) contributed with computationally efficient learning methods. Some of these methods rely in their core on a spectral decomposition of the observed statistics—moment matrices or tensors—and are thus commonly denoted as "spectral methods". MoM or alternatively spectral learning has been widely used to learn challenging models including hidden Markov models (Hsu et al., 2009), latent trees (Parikh et al., 2014), latent junction trees (Parikh et al., 2012a), probabilistic context free grammars (Cohen et al., 2012; Parikh et al., 2011), mixture models (Anandkumar et al., 2014), finite state machines (Luque et al., 2012; Balle et al., 2011) and predictive state representations (Littman et al., 2002; Rosencrantz et al., 2004a).

spectral methods

A challenge specific to spectral methods involves estimating probabilities from spectral decompositions. MoMs do not restrict, in general, the parameters to be non-negative, the *non-negativity problem*, leading to violation of the probability constraint. For moment-based methods, an analogous problem arises, when we extract parameters that match

MoM challenges

non-negativity problem

empirical moments: one of the main difficulties is related to statistical noise that and resides in undoing the linear transformation, such that the recovered parameters are constrained to a feasible set, either their marginal polytope[1] or probability simplex. An alternative interpretation of this issue lies within the fact that MoMs rely on an infinite data assumption, *i.e.*, the knowledge of exact moments to correctly retrieve the model parameters. On top of that, in practice, even for large amounts of data, these methods suffer statistical noise in the empirical estimates of the moments, leading to suboptimal rates of convergence, the *statistical efficiency problem*. Another problem with moment-based methods is that in order to construct a set of moments that makes the parameter estimation problem well conditioned, we may need to move to a harder class of computational problems: for example, we may prefer to observe a large sparse matrix of second moments instead of a small dense one. In this case the matrix factorization problem becomes a matrix completion problem, which is computationally more difficult (Balle et al., 2012a; Bailly et al., 2013a). The challenges above stem in part from the fact that it is common for MoM estimators to introduce an extra transformation (often linear) on top of the traditional parameter representation, and it is difficult to undo this transformation.

[1] The marginal polytope defines the set of realizable marginal probabilities (**Wainwright03**)

statistical efficiency problem

From a practical point of view moment-based algorithms have not yet succeeded in many domains as a viable and more effective alternative to likelihood-based methods. Prior work has been done in the direction of finding heuristics to improve results. Cohen et al. (2013b) and Luque et al. (2012) provided improvements in terms of accuracy and runtime, but these have mostly focused on how to handle problems inherent to spectral learning such as negative probabilities (non-negativity problem), and limited data (breaking the infinite data assumption these methods require). Nevertheless, there is still room for experimental improvement, specially when compared with local search methods, such as gradient descent or EM, when the observations space is large. *In this thesis, we focus on the MoM for estimating parameters of sequential models such as hidden Markov models and predictive state representations. Next, we provide an overview of how the MoM can be applied to structured prediction tasks. We further focus on how the MoM and kernel approaches can be applied in the context of Natural Language Processing and Robotics.*

Structured Prediction with Method of Moments

Structured prediction is a general machine learning problem that attempts to uncover a hidden structure of the data. Structural elements can be found both in Natural Language Processing (NLP) and Robotics

in different forms, such as parse-trees, input/output sequence models in NLP or sequential hybrid models in Robotics.

Output prediction can be regarded as a labeling problem for the sequence of inputs. In Natural Language Processing there are several tasks that could be framed in this setting, such as part-of-speech tagging, named entity recognition, information extraction, and syntactic disambiguation (Manning et al., 1999). Conversely, in Robotics there is also a large number of tasks that could be posed as a input/output prediction problem, such as mode identification (Lauer et al., 2008; Paoletti et al., 2007) and grasp classification (Cutkosky et al., 1990).

In this thesis, we deal with two types of structured prediction: sequence completion and sequence labeling.

Sequence completion consists of predicting a sequence of events based on preceding elements. Here, the order of the elements is crucial to learning a correct model of the internal representation, and different algorithms exist for different types of problems. In this thesis, we consider both instances of discrete and continuous sequential data, *i.e.*, the domain of the elements of each sequence can be discrete elements in a finite set or discrete-time observations of a continuous valued process. In NLP, typically we deal with the former, and in Robotics we generally address the latter. In both cases, the prediction task can be cast as a more general learning problem of a high level model that is able to predict future elements of an output sequence of events, based on an input sequence, as depicted in Figure 1.2.

In standard sequence completion problems, the output sequence corresponds to the consecutive element or elements of the sequence, such as in most common sequence prediction problems: weather forecasting, stock market forecasting, language modeling, robot pose prediction in robotic manipulation. In the more general sequence labeling tasks, the output has a less restrictive interpretation.

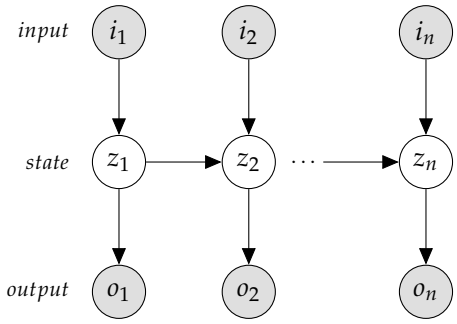


Figure 1.1: Sequence prediction task with input i_t/output o_t sequences (in gray). The process is modeled by a finite state machine with states z_t (in white).

Method of moments for Robotics and Natural Language Processing

The learning of predictive models can be done with existing input/output sequence pairs, in a supervised setting; relying only on sequences of input sequences, in an unsupervised setting or using a combination of input/output pairs with typically less expensive unlabeled input sequences, also referred to as semi-supervised learning. MoM or spectral learning algorithms are often associated with unsupervised learning, where no information about the input sequence is available. More specifically, when we are interested in augmenting the states as refinement layers, or as in latent-variable learning. The sequence in certain models, hidden Markov models (HMMs) or predictive state represen-

tations (PSRs), can be considered as a sequence of hidden state variables that serve as additional information about the input sequence, increasing the expressiveness of the model.

In many applications, it is not necessary to explicitly know this hidden space representation and tasks, such as filtering, and prediction can be done effectively without it. Many Robotics/NLP applications fall under this category, such as localization and observation prediction in a dynamical system, language modeling and model refinement. We group this latter form of learning together with spectral learning techniques. However, in many applications we are interested in an explicit representation of the hidden states. Whenever we grant hidden states with meaningful representations, we need to explicitly ascertain about their probability distribution. Many NLP tasks require this property, such as part-of-speech tagging, named entity recognition, and all structured prediction tasks where the states represent labels. We can encode this information by using different approaches, namely anchor techniques (Arora et al., 2013).

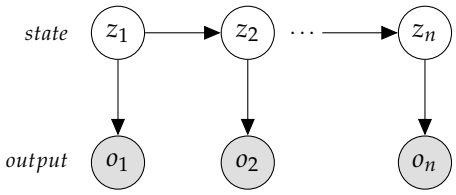


Figure 1.2: Sequence prediction task with output o_t sequences (in gray). The process is modeled by a finite state machine with states z_t (in white).

interpretable vs. hidden states

In this thesis, we will investigate both directions. In the first direction, we learn model parameters in an interpretable setting, and explicitly recover the hidden state distribution— *anchor learning*. In the second one, we ignore the hidden state interpretation and are only interested in predicting future observations conditioned on past observations, by means of a more expressive model— *spectral learning*.

Kernel-based learning

Kernel methods are a common approach in Machine Learning. They provide a compact representation of high dimensional, non-linear data by embedding data from an input set $\mathcal{X}$ into a feature space Φ. The kernel function provides the means to reason about distance between points in this space, via the use of feature inner-products $k(x, y) = \langle \phi(x), \phi(y) \rangle$. This fact brings forth the advantages of linear models, such as computationally efficiency without losing the expressivity of non-linear models (Scholkopf et al., 2001).

Additionally, a Gram matrix formulation of this mapping is often used to circumvent the need to represent high dimensional feature mappings. However, in many cases this formulation is computationally prohibitive, since one needs to compute a $N \times N$ Gram matrix of pairwise kernel evaluations for a dataset on N samples. This formulation deprives the applicability of kernel methods to large training datasets. On the other hand, randomized feature representations trade off accuracy of the kernel expansion for scalability, by mapping data into a low-dimensional randomized feature space, greatly reducing the computation of kernel evaluations and training (Rahimi et al.,

2008; Sinha et al., 2016).

More recently, efficient learning of deep architectures (Orr et al., 1998; Hinton et al., 2006) gave rise to an efficient and flexible family of algorithms that have proven to be very successful in many research communities. Carefully designed forms of regularization techniques to control the representational power of deep networks allowed the learning of compact representations (Prechelt, 1997; Srivastava et al., 2014; Ioffe et al., 2015). However, in contrast to kernel approaches, these deep learning techniques do not benefit from a well understood theoretical analysis (Bengio, 2009).

In this thesis, we focus on reproducing kernel Hilbert spaces (RKHSs), a space of kernels with additional structure of its feature space (Aronszajn, 1950; Wahba, 1999). RKHSs provide a good trade off between expressivity and tractability. They also benefit from an extensive theoretical analysis of convergence (Schölkopf et al., 2002). But the main advantage of RKHSs lies on the ability to evaluate a continuous function f at a given point x by considering the inner product of f with the reproducing kernel feature map at that point $k(\cdot, x)$. This is not true, in general, for any arbitrary Hilbert space (Riesz, 1928; Conway, 1985). RKHSs may equivalently be defined as a Hilbert space of functions (space endowed with inner product) whose evaluation of functions is a continuous and linear operation. The interpretation of function evaluation as inner products in the RKHS is the core property that grants rich enough structure to design compact and efficient algorithms. Optimization problems, such as taking the gradient of any function f in the RKHS with respect to f, may be translated into reproducing kernel variants in terms of the inner product of f in the RKHS $\langle f, k(\cdot, x)\rangle$, since the gradient assumes the form $k(\cdot, x)$. This will allow to describe any optimization algorithm of the form: $\mathbf{min}\ f_\theta(x)$ with respect to θ in RKHS terms.

RKHSs have been successfully applied in many fields such as Statistics (Parzen, 1962; Parzen et al., 1970) and Machine Learning (Manton et al., 2015; Scholkopf et al., 2001). In this thesis, we will focus on prediction techniques for sequential Robotic problems.

Thesis Statement

The core of this thesis consists of improving kernel and moment-based methods for learning sequence prediction and planning. We validate the proposed methods on a varying range of tasks, such as labeling problems in Natural Language Processing, and trajectory optimization and policy learning in Robotics.

We focus on planning algorithms for Robotic applications, with emphasis on optimization based planning. We propose a novel planning algorithm for optimizing robot trajectories in high dimensional spaces using kernel methods. Furthermore, we investigate planning

under uncertainty by combining MoM and reinforcement learning algorithms. We study variants of PSRs for continuous observations for more complex robotics tasks, such as robotic manipulation and planning in a partial observable setting. Additionally, we propose to study other moment matching techniques, that provide easier forms of interpreting hidden state variables, as well as subsuming supervised information, where we intend to design algorithms that are capable of handling large amounts of data in a weakly supervised setting, are flexible, and allow the inclusion of model constraints. In this regard we introduce sequential prediction for discrete representations with anchoring observations.

1.1 *Main contributions*

This thesis makes the following contributions:

Contribution 1: Interpretable representation of states

In many applications, it is often beneficial to build up models with meaningful representations, such as in sequence labeling tasks, where the hidden variables represent labels in a finite set. Many tasks in Natural Language Processing and Robotics fall under this category, in particular those pertaining to sequence labeling tasks, where the hidden states represent part-of-speech tags, named entities, or parsing constituents. It is, therefore, important to recover explicitly a mapping between hidden states and labels. This remains one of the main challenges in MoM estimation.Due to the nature of the problem, it is difficult to find an interpretable representation of the hidden structure.

Usually, spectral methods marginalize over the hidden variables, not revealing its state representations. There has been prior effort into finding the explicit model parameters, *e.g.*, via the tensor power method (Arora et al., 2013) and simultaneous diagonalization (Kuleshov et al., 2015), both of which address learning a similarity transform between hidden states and labels. However, these methods are very sensitive to model mismatch, and to statistical noise. We build upon an alternative form of learning that relies on anchor observations, *i.e.*, unambiguous observations that uniquely map to a state variable. This work was first derived for finding topic distributions in topic modeling (Arora et al., 2013).

We introduce an anchor-based algorithm for learning sequence labels using the MoM. By taking advantage of the anchor idea we introduce a novel algorithm that learns interpretable representations of state in a hidden Markov model, in Chapter 3.

Contribution 2: Integrate supervision with moment-based estimation

It is often desirable to restrict models to retain certain labeled information, either in the form of prior distributions or constraints over model parameters. For this purpose we require an explicit mapping into the hidden state representation. Finding model parameters that agree with given moment constraints and conform with supervised information is not a straightforward task, in particular in those methods that rely on spectral decompositions of observations. The main problem is that most moment-based methods describe probability of observations as a series of operator multiplications, without providing an explicit representation for hidden states, since this is a harder problem. They simply perform inference by propagating the effect through the hidden structure implicitly. This renders integration of prior knowledge hard. For instance, estimated operators, also denoted as predictive state representations (PSRs) (Littman et al., 2002; Singh et al., 2004b) or observable operator models (OOMs) (Jaeger, 2000) can be obtained by first expressing moments in a lower dimensional space that correlates well with past observations, and secondly by regressing the predicted future estimates from the past. This process does not explicitly involve the model parameters, being only derived from observed data.

Another line of moment methods addresses this challenge by modeling sequential tasks as input-output finite state machines, but still keeping the hidden state space unknown (Balle et al., 2011). Here inputs relate to observations and outputs to labels. However, this approach requires the knowledge of the labels during training, which makes the learning problem not suitable for weak supervision on large datasets.

We provide a learning algorithm based on moment estimation that is able to incorporate supervised information in different forms. First, we introduce a weakly-supervised learning approach that is able to learn from a small labeled and a large unlabeled dataset. Second, we introduce an additional source of supervision by means of supervised regularization in Section 3.6.

Contribution 3: Feature based anchor learning

Sequence models with more expressive representations have long been used in Machine Learning. These allow for more complete representations, such as low dimensional embeddings or features of observations. In particular in the Natural Language Processing community is it common to consider sequential models with log-linear emissions, or also known as exponential family of distributions. Smith et al. (2005), Lafferty et al. (2001), and McCallum et al. (2000) considered such feature-based representations of observations in a discriminative

setting and Berg-Kirkpatrick et al. (2010) generalized this notion for generative models.

We introduce a moment-based anchor learning algorithm that is able to handle continuous representations using HMMs with log-linear emissions. We apply this method in a Natural Language Processing task, part-of-speech tagging (§ 3.4).

Contribution 4: Combine kernel approaches for gradient based planning

Functional gradient algorithms are a popular choice for robot motion planning in complex high dimensional environments. Optimization based planning attempts to directly improve the trajectory of a robot within a space of continuous trajectories. The objective is to reach a goal position without colliding into obstacles and by maintaining geometric properties such as smoothness. In practice, standard planning implementations such as CHOMP (Zucker et al., 2013) and TrajOpt (Schulman et al., 2013) typically commit to a fixed, finite parametrization of trajectories, often as a sequence of waypoints.

We introduce a functional gradient descent trajectory optimization algorithm for robot motion planning in Reproducing Kernel Hilbert Spaces (RKHSs), in Chapter 4. We represent trajectories as functions in an RKHS (§ 4.2). Restricting trajectories to a space of reproducing kernels provides a seemly representation that naturally gives rise to a notion of smoothness in terms of the norm induced by the reproducing kernel. In consequence, depending on the selection of kernel, we can directly optimize in spaces of trajectories/functions that are inherently smooth in velocity, jerk, curvature, or any meaningful efficiency norm (§ 4.4).

Contribution 5: Design an efficient algorithm including kinematic and dynamic robot constraints

Common optimization based strategies work (in theory) by directly optimizing within a space of continuous trajectories to avoid obstacles while maintaining smoothness. However, in practice, standard planning implementations represent trajectories as a sequence of finely discretized waypoints. Such a parameterization can lose much of the benefit of reasoning in a continuous trajectory space: e.g., it can require taking an inconveniently small step size and large number of iterations to maintain smoothness.

Trajectory optimization in RKHSs generalizes functional gradient trajectory optimization by representing trajectories as linear combinations of kernel functions. As a result, we are able to take larger steps and achieve a locally optimal trajectory in just a few iterations. The selection of an appropriate kernel

can reduce the complexity to a low-dimensional, adaptively chosen parameterization. We propose an efficient gradient-based algorithm that is able to handle robot constraints, such as boundary conditions, joint limits and velocity limits (§ 4.3). Our experiments illustrate the effectiveness of the planner for different kernels, including Gaussian RBFs with independent and coupled interactions among robot joints, Laplacian RBFs, and B-splines (§ 4.7).

Contribution 6: Combine modeling and planning

Next we focus on how to integrate predictive models to improve existing planning algorithms. Once a model of the environment is learned, how can we act upon it to achieve a desired goal? We refer to planning as the mechanism to determine a sequence of actions that maximize a given objective function, typically the sum of expected return provided by a learning agent in an interactive environment. Little work has been done in combining predictive representations with planning, and is mostly restricted to the discrete or blind settings.

We introduce a new class of policies that is able to leverage the benefits of having a predictive model with a recurrent architecture that maps predictive states to actions. We introduce a novel architecture that is designed to learn this mapping in a partially observable environment. We denote it Recurrent Predictive State Policy (RPSP) network, in Chapter 5. RPSPs consist of a recursive filter, which keeps track of a belief about the state of the environment, and a reactive policy that directly maps beliefs to actions. The recursive filter uses predictive state representations (PSRs) (Rosencrantz et al., 2004b; Sun et al., 2016a) by modeling the state as a predictive state, i.e., *a prediction of the distribution of future observations conditioned on history and future actions. Therefore, the policy component of the RPSP-network can be purely reactive, simplifying training while still allowing optimal behaviour. A reactive policy mapping from predictive states to actions will have rich information for decision making, as it can reason about the entire history of the controlled process.*

Contribution 7: Filter initialization

Recurrent networks with memory models, such as GRUs and LSTMs, can leverage information about previous observations and actions to keep track over a belief of the state of the environment. Although RNNs may exploit heuristics for an improved initialization, such as noisy initial values (Zimmermann et al., 2012), however, such a strategy does not guarantee good performance.

We propose an initialization procedure for RPSP recurrent filters that provides good theoretical guarantees in the limit of infinite data, in Chapter 5. RPSP filters correspond to PSR models; as a result they can make use of history like LSTMs/GRUs, and track in the RKHS of distributions of future

observations based on past histories like PSRs. For this reason, RPSPs have a statistically driven form of initialization, that can be obtained using moment matching techniques, with statistically consistent algorithms (Hefny et al., 2017b; Boots et al., 2013a)

Contribution 8: End-to-end algorithm with prediction as regularization

RPSPs define computation graphs, where the parameters are optimized by leveraging the states of the system. Predictive states encode rich enough information of the partial observable environment, with the additional benefit of having a clear interpretation as a prediction of future observations trained based on that interpretation.

We make use of the PSR interpretation to formulate an end-to-end training algorithm, by incorporating prediction error in the loss function. The entire network (recursive filter and reactive policy) is differentiable and can be trained using gradient based methods, in Section 5.6. We optimize RPSPs using a combination of policy gradient based on rewards (Williams, 1992) and gradient descent based on prediction error. We show the efficacy of RPSP-networks on a set of robotic control tasks from OpenAI Gym. We empirically show that RPSP-networks perform well compared with memory-preserving networks such as GRUs, as well as finite memory models, being the overall best performing method (§ 5.8).

1.2 Previous Publications

During the course of this doctoral study, further work was developed in collaboration with other authors which do not count as first author contributions and are thus omitted from this thesis. This includes work in the field of sequence prediction with mode switching, where we combine Gaussian process regression with particle filters to improve prediction in the vicinity of mode transition (Lee et al., 2017); and work in the area of predictive planning where we introduce predictive representations to controlled processes as a filtering mechanism in partially observable environments (Hefny et al., 2017a).

1.3 Thesis organization

This thesis is organized as follows:

Part II: Background

This part summarizes existing work on the method of moments, providing the necessary background required to understand the main

contributions of this thesis. We provide an overview of the method of moments literature for prediction and planning in four chapters, as we describe below:

Section 2.1 provides an overview of the notation used throughout the thesis.

Section 2.3 describes the most common latent variable models used in moment estimation, in §2.3, and §2.4 describes learning techniques for finding model parameters by explicitly recovering the model parameters.

Section 2.5 In §2.5 we describe spectral learning models for sequence prediction, including HMMs, OOMs, FSTs and PSRs and their learning algorithms in §2.5.6.

Section 2.6 provides an overview of planning algorithms mostly applied to Robotics.

Section 2.7 provides an overview of planning under uncertainty/reinforcement learning algorithms.

Part III: Sequence Labeling with Method of Moments

The second part of this thesis provides contributions made in the field of sequence labeling using method of moments in Chapter 3. We provide a description in the following order:

Section 3.3 introduces a novel algorithm for sequence labeling using anchor-based learning, with an extended notion of anchors.

Section 3.4 introduces a feature-based extension of anchor learning for hidden Markov models (HMM).

Section 3.6 presents an empirical analysis of anchor-based learning for HMMs and log linear models.

Part IV: Planning with Kernel Methods

The second part of this thesis provides contributions made in the field of Robot Motion Planning using kernel approaches in Chapter 4. We provide a description in the following order:

Section 4.2 introduces an approach for representing trajectories as functions in a functional reproducing kernel Hilbert space (RKHS).

Section 4.3 introduces a constrained optimization algorithm based on gradient descent methods in RKHSs.

Section 4.7 presents an empirical analysis of trajectory optimization using distinct variants of RKHSs in simulated toy and more complex robotic environments.

Part V: Planning with Method of Moments

The third part of the thesis describe how we can combine a predictive models with existing planning algorithms based on reinforcement learning in Chapter 5.

- **Section 5.5** proposes a new class of policies combining moment-based predictive models with reactive policies.

  Section 5.6 combines controllable predictive state models with planning algorithms, in a reinforcement learning approach. This section presents an end-to-end algorithm for continuous actions based on predictive representations under partial observability.

- **Section 5.8** provides experimental analysis of learning RPSP networks through gradient descent in a joint and alternate approach, using OpenAI Gym simulated environments.

Part VI: Conclusions

This chapter highlight the main conclusions and findings of this thesis and provides possible directions for future work.

2
Background

In this chapter, we introduce related work and background required for a comprehensive understanding of this thesis. We provide an overview of sequential models in (§ 2.2), we introduce some latent variable models in (§ 2.3) and learning algorithms in (§ 2.4). We discuss sequential models for spectral learning in (§ 2.5). We review some background on planning in (§ 2.6), and in particular in reinforcement learning in (§ 2.7).

2.1 Notation

In this thesis, we will define vectors in bold notation as column vectors $\boldsymbol{v} \in \mathbb{R}^{n \times 1}$, where $[\boldsymbol{v}]_i$ denotes the entry indexed by the i-th component of the vector $\boldsymbol{v}$. We write matrices as capital letters M, with $M_{i,j}$ denoting the i-th row and j-th column of the matrix. We further denote $\mathbb{R}_+$ to be the field of real non-negative values and $\mathbb{R}_{++}$ the field of strictly positive values. We say $\boldsymbol{v}$ belongs to the probability simplex Δ^{d-1} such that:

$$\Delta^{d-1} = \{\boldsymbol{v} \in \mathbb{R}^d : v_i \in [0,1] \forall_i \in [d], \sum_{i=1}^{d} v_i = 1\} \tag{2.1}$$

We refer to real valued functions $f : \Xi \to \mathbb{R}$ as points in a function space $\mathcal{F}$ by keeping the argument without assignment $f(\cdot) \in \mathcal{F}$ represents a function, *i.e.,* an element in the space, while $f(0) \in \mathbb{R}$ refers to an evaluation of the function at point 0. We use $\otimes$ to denote the outer-product, for vectors $\boldsymbol{x} \in \mathbb{R}^X$ and $\boldsymbol{y} \in \mathbb{R}^Y$ is equivalent to $\boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{x}\boldsymbol{y}^\top \in \mathbb{R}^{X \times Y}$. $\times$ is used to denote the element-wise product of vectors and matrices of the same dimension. Furthermore, we define the tensor product of any arbitrary p-order tensor $T \in \mathbb{R}^{n_1 \times \ldots \times n_p}$ as a multilinear mapping, where $\times_i$ refers to the multiplication along mode i. Let the vector $v_j \in \mathbb{R}^n_j$ and the matrices $V_j \in \mathbb{R}^{n_j \times d_j}$, $\forall_{j \in [3]}$. The projection of the matrices on each mode of the tensor yields

$$T(V_1, \ldots, V_p) = T \times_1 V_1 \times_2 V_2 \ldots \times_p V_p \ \in \mathbb{R}^{d_1 \times d_2 \ldots \times d_p} \tag{2.2}$$

When the tensor is a second-order tensor, *i.e.*, a matrix, the matrix and vector multiplication reduces to the usual definition $T(V_1, V_2) = V_1^\top T V_2$ and $T(I, \boldsymbol{v}) = T\boldsymbol{v}$.

2.2 *Models for sequential systems*

In this chapter, we review some existing learning algorithms for sequential systems, where a sequence of observable events $o_i \in \Sigma$ is determined by an underlying stochastic process Z, see Figure 2.1. Depending on the application observations may correspond to continuous values, such as many Robotics problems, or discrete values, such as words in a dictionary in Natural Language Processing (NLP).

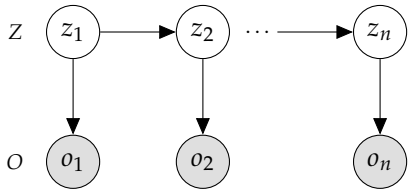


Figure 2.1: Observations o_i (bottom), generated from a hidden process z_j (top).

In the following sections we review sequential models for stochastic processes under different assumptions, described in Figure 2.2, using method of moments (MoM), see Section 2.4.2. An important characteristic of moment-based learning is that they provide consistent parameter estimates, under certain assumptions on the singular values of the model. They provide an alternative to a different class of learning methods, based on likelihood maximization estimation (MLE), in particular local methods such as the popular Expectation-Maximization (EM) algorithm (Dempster et al., 1977).

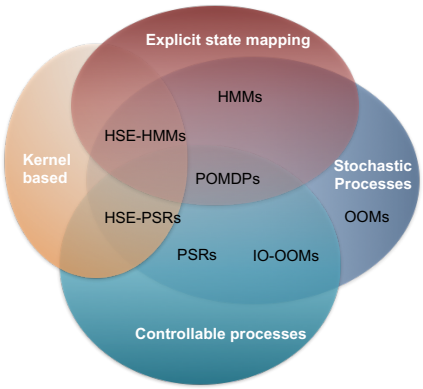


Figure 2.2: Sequential Models overview. Models with interpretable view—latent variable models (gray). Non-linear sequential models (orange). Linear sequential models (blue).

MLE provides a statistically efficient estimation paradigm but it is intractable to solve, due to the non-convexity of the likelihood function. Local methods, such as EM, turn this problem into a tractable one, but suffer from local optima and slow convergence (Redner et al., 1984), see Section 2.4.1. In fact for some models such as hidden Markov models (HMMs) (Rabiner, 1989), latent trees (Mossel et al., 2006) and topic models (Arora et al., 2012) MLE is NP-hard. MoM, on the other hand, does not suffer from local optima, even in situations where maximum likelihood does. MoM exploits algebraic properties of the model to build factorization of moments into model parameter estimates. Moment-based techniques can be divided into two groups. The first involves performing some spectral decomposition and is usually denoted as *spectral learning*, described in more detail in Section 2.5.6. The second refers to moment-matching techniques that explicitly model the hidden states, these are mostly known as *anchor methods*, see Section 2.4.4.

spectral learning vs. anchor learning

A large group of theoretical computer scientists studied the computational and sample complexity related to estimating certain latent variable models such as Gaussian mixture models (Dasgupta, 1999; Arora et al., 2001; Dasgupta et al., 2007; Vempala et al., 2004; Kannan et al., 2008; Chaudhuri et al., 2008; Brubaker et al., 2008; Belkin et al., 2010; Moitra et al., 2010) and HMMs (Hsu et al., 2012a; Chang, 1996; Mossel et al., 2006; Anandkumar et al., 2012a)

In Section 2.3, we formally describe the models where we describe data as a stochastic generative process and where we explicitly identify the hidden state variables. Furthermore, we provide an overview of learning algorithms based on MoM, in Section 2.4, by estimating the parameters through an explicit mapping of the hidden state variables, which we denote by *latent variable learning*. This explicit learning variant recovers model parameters— conditionals M and marginals Z, from moments of observations W. These parameters can be combined together in the form of some operators, also known as observable operators, that allow us to describe probabilities of sequences of observations, in Figure 2.3a.

A different line of work based on subspace identification (Van Overschee et al., 1996), observable operator models (OOMs) (Jaeger, 2000) and multiplicity automata (Schützenberger, 1961) have been proposed to latent variable models. In particular Hsu et al. (2009) and Rodu et al. (2013) applied spectral learning to HMMs with finite sample bounds. Further work developed a more general class of models to capture the evolution of sequential systems without explicitly recovering the hidden variables, such as reduced rank HMMs (Siddiqi et al., 2010b), kernel HMMs (Song et al., 2010b), predictive state representations (PSRs) (Littman et al., 2002; Singh et al., 2003), latent tree graphical models (Parikh et al., 2012a), weighted automata (Balle et al., 2012b; Bailly et al., 2013a) and probabilistic context-free grammars (PCFGs) (Cohen et al., 2013b; Dhillon et al., 2011). These methods rely on low order moments to estimate an operator that embeds the distribution of a sufficient statistic of the model. These operators can be used for density estimation and belief state updates, very commonly used in robotics and language generation.

In Table 2.1 we summarize the main literature on moment-based approaches for sequential systems, emphasizing the main characteristics of each approach.

In Section 2.5, we review learning approaches for sequential models, where there is no explicit mapping to the hidden state space, which we denote as *spectral learning*. In this variant, we are not interested in recovering the hidden structure, but instead model the generative process, while being able to perform prediction over sequences of observations.

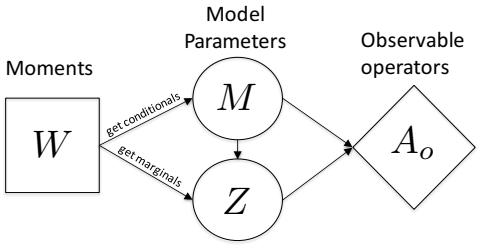
Models	Explicit state	Kernel based	Controllable systems	Time variant	Sec.	Refs.
						Hsu 09; Anandkumar 12; Anandkumar 14
HMMs	✓	-	-	-	2.3.1	Chaganty 14; Kuleshov 15; Siddiqi 10
anchorHMM	✓	-	-	-	3	Marinho 16
HSEHMMs	✓	✓	-	-	2.5.5	Song 10; Smola 07
POMDPS	✓	-	✓	-		Azizzadenesheli 16
OOMs	-	-	-	-		Jaeger 99
IO-OOMs	-	-	✓	-	2.5.3	Jaeger 00; Thon 15
WFA	-	-	-	-	2.5.2	Balle 12; Bailly 13
PCFGs	-	-	-	-		Cohen 13; Dhillon 11
PSRs	-	-	✓	-	2.5.4	Littman 01; Singh 03; Rosencrantz 04; Boots 11
HSEPSRs	-	✓	✓	-	2.5.6	Boots 13; Hefny 15
PSIMs	-	✓	-	✓		Sun 16

Table 2.1: Summary of moment-based approaches for prediction in sequential systems.

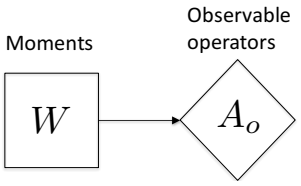
In spectral learning we directly estimate observable operators, from which we can describe the sequential process, in Figure 2.3b. We provide a general description of sequential models and bring together different models proposed for controlled and uncontrolled sequential systems. In Section 2.5.6, we consider learning of controlled systems where we need to account for the effect of actions that an agent can perform, either a robot or a state machine. The added challenge in learning controlled systems lies in the fact that the actions performed by the agent change the distribution of future observations.

The first class of methods expresses a simpler model but is able to extract the hidden state parameters, which can be important depending on the task at hand. Conversely, the second class of methods characterizes a more expressive model, with the caveat of not being able to reason about the hidden states explicitly. We present methods that can cope with dynamical systems with discrete observations (HMMS (§ 2.3.1), PSRs (§ 2.5.4)) and continuous observations (HSE-HMMs (§ 2.5.5)), later we describe work on controlled processes (controlled PSRs (§ 2.5.6)).

If we want to do sequence labeling or discover explicit parameters in the model, such as transitions and emissions in a hidden Markov model, the former family of estimators is more suitable. However if we want to predict sequences of observations conditioned on past observations and/or actions, or even generate observations from the model (language modeling), the former groups of methods are probably a better fit. In this thesis, we attend to both types of models.



(a) Latent variable learning, with explicit state mapping, requires first to estimate model parameters M, Z, from moments W.



(b) Spectral learning, where the mapping to the state variables remains unknown, directly finds observable operators.

Figure 2.3: Moment based learning.

2.3 *Latent Variable Models*

Latent space models define a general tool in machine learning to model data generated from i.i.d., sequential or more structured samples (Ghahramani et al., 1999; Jordan et al., 1999; Blei et al., 2003; Quattoni et al., 2005; Haghighi et al., 2006). Learning these models is hard in most cases, since the hidden/latent states are not directly observed. Instead, they need to be estimated from correlated observations. We consider in particular the case of sequential models.

Let $\mathcal{G}$ be a graphical model with observed variables $\mathbf{o} = \{o_1, \ldots, o_L\}$ where $o_i \in \Sigma$ is a discrete observation in the set of dimension $|\Sigma| = d$; and let $\mathbf{z} = \{z_1, \ldots, z_M\} \in [K]^M$ denote its discrete hidden variables with each $z_j \in [K] \forall_{j \in [M]}$.

For every graphical model, its parameters can be estimated via the MoMs, where we can exploit the underlying structure of the model. We decompose the estimation into two parts $p(O, Z) = \prod_{S \in \mathcal{C}} p(O^v \mid Z_S) p(Z_S)$, for any set of nodes forming a clique in the graph S. First, we estimate the leaves of the graph corresponding to observed variables conditioned on some conditionally independent hidden variables. We denote them conditional moments or *conditionals*

conditional moments

$$M_v = p(O_v \mid Z_S) \in \mathbb{R}^{d \times K}. \tag{2.3}$$

Second, we estimate the connection among the subset of hidden variables, we denote them *marginals*

$$p(Z_S) \in \mathbb{R}^K. \tag{2.4}$$

marginals

Correct estimation of conditional moments usually requires certain assumptions either in the form of independent views. In particular, for HMMs three independent views are required to uniquely identify the model O^v, $v = [1, 2, 3]$ need to exist for each hidden variable, or full rank for each conditional moment matrix M_v (Kruskal, 1977). The full-rank assumption of the conditional moments ensures that all hidden variables are linearly independent, guaranteeing the recovery of conditional moments. On the other hand, the marginal distribution of the hidden variables $p(Z_S)$ can be recovered for those hidden variables $S \in \mathcal{Z}$ using different approaches, which we will present in Section 2.4.5.

2.3.1 *Hidden Markov Models (HMMs)*

HMMs are one of the most fundamental and widely used statistical tools for modeling discrete time series (Rabiner, 1989). These models are characterized by a Markov chain of hidden states $z_1, z_2, \ldots, z_L \in \mathcal{Z}$ over $|\mathcal{Z}| = K$ possible states $[K]$ (components), with initial state distribution given by $\boldsymbol{\pi} = p(Z_1) \in \Delta^{K-1}$. Given the state at time n

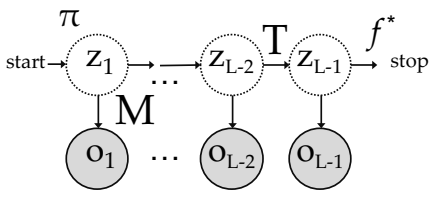


Figure 2.4: Hidden Markov Model, observations (o_n) hidden states (z_n).

$Z_n = z_j \in [K]$ the observations $O_n = o_i \in [d]$ are generated independently from other states and observations, as shown in Figure 2.4. We denote the conditional mean distribution of observations as $\mu_j = p(O_n \mid Z_n = z_j) \in \Delta^{d-1}$. We write the transition distribution between hidden states as $T \in \mathbb{R}^{K \times K}$, where $T_{j,m} = p(Z_n = z_j \mid Z_{n-1} = z_m)$. We further define a stopping distribution $\mathbf{f}^* \in \mathbb{R}^K$ and $f_j^* = p(\text{STOP} \mid Z_{L-1} = j), \forall_{j \in [K]}$, such that $\mathbb{1}_k^\top T + \mathbf{f}^* = \mathbb{1}_k$.

We consider the matrix of all conditional probabilities— also known as emission or observation matrix $M = [\boldsymbol{\mu}_1 \mid \boldsymbol{\mu}_2 \mid \cdots \mid \boldsymbol{\mu}_K]$.

HMMs are characterized by two conditional independence assumptions. The first one is the Markov assumption , which states that the current state is independent on all the previous history of events conditioned on just the previous state $p(Z_n \mid Z_{n-1}, Z_{n-1}, \ldots, Z_1) = p(Z_n \mid Z_{n-1})$. The second conditional independence assumption defines the generative property of the model's observations O_n are generated conditionally independently given the current state Z_n. Given these assumptions, $T, \boldsymbol{\pi}, \mathbf{f}^*$ and M fully characterize the HMM. Figure 2.4 shows the corresponding graphical model of an HMM.

Markov assumption

2.4 *Latent Variable learning*

In many classes of models latent states can be estimated from low order moments, typically second or third order moments. This contrasts with previous moment algorithms that relied on the estimation of high-order moments, which are usually hard to estimate accurately due to their large variances. These decomposition typically come with simple and efficient learning algorithms, some of which we discuss in this section.

2.4.1 *Maximum Likelihood*

The most common solution to learning latent variable models is based on the maximum likelihood principle or Bayesian inference, contrary to moment-based approaches. Here model parameters θ are obtained by maximizing the likelihood of the observed data $o \in \Sigma$ given the chosen parameters, also known as maximum likelihood estimation (MLE) (Fisher, 1912) w.r.t. a probability distribution $q(z)$ over the la-

tent variables.

$$\begin{aligned}\mathcal{L}(\theta) &= \log p(o \mid \theta) = \log \sum_z q(z) p(o, z \mid \theta) \\ &=^{1} \log \sum_z q(z) \frac{p(o \mid z, \theta) p(z \mid \theta)}{p(z \mid o, \theta)} \\ &= \log \sum_z q(z) \frac{p(o \mid z, \theta) p(z \mid \theta)}{q(z)} \frac{q(z)}{p(z \mid o, \theta)} \\ &\geq \sum_z q(z) \log \frac{p(o \mid z, \theta) p(z \mid \theta)}{q(z)} + \sum_z \log \frac{q(z)}{p(z \mid o, \theta)} \\ &= \mathcal{F}(q, \theta) + KL(q(z) || p(z \mid o, \theta))\end{aligned} \tag{2.5}$$

where $z \in Z$ denotes the hidden parameters of the model. $\mathcal{F}$ defines a lower bound on the likelihood function, and is referred as the *free energy*, and the second term refers to the Kullback-Leibler divergence between the latent distribution $q(z)$ and the model distribution $p(z \mid o, \theta)$, which is always non-negative. The inequality follows from Jensen's inequality over $q(z)$ (Jensen, 1906).

[1] From Bayes' rule $p(z|o,\theta) = \frac{p(o|z,\theta)p(z|\theta)}{p(o|\theta)} \Leftrightarrow p(o|\theta) = \frac{p(o|z,\theta)p(z|\theta)}{p(z|o,\theta)}$

This ensures that maximizing the lower bound is enough to guarantee an increase in the exact log likelihood. This bound can be further decomposed into two terms the *energy* and the *entropy* $\mathcal{H}(q) = -\int q(z) \log q(z) dz$, which is independent of θ:

$$\mathcal{F}(q, \theta) = \sum_z q(z) \log p(o, z \mid \theta) + \mathcal{H}(q) \tag{2.6}$$

MLE is statistically consistent but leads to intractable optimization problems due to the expectation over all possible latent distributions (Redner et al., 1984; Mossel et al., 2006). Local heuristics such as the expectation-maximization (EM) algorithm (Dempster et al., 1977), have proven to be very successful in many fields. They have been applied to a vast range of latent variable models, such as Gaussian mixture models (GMMs) (Titterington et al., 1985), topic models such as latent Dirichlet allocation (LDA) (Blei et al., 2003), hidden Markov models (HMMs) (Rabiner, 1989) and probabilistic context free grammars (PCFGs) (Matsuzaki et al., 2005). The EM algorithm consists of two alternating maximization steps w.r.t. q and θ — also regarded as coordinate ascent of the free energy:

- **E-step**: Fix θ^k and solve for the distribution over latent variables $q^{k+1} = \arg\max_{q \in \mathcal{P}} \mathcal{F}(q, \theta^k)$. Since the log-likelihood is independent of $q(z)$ maximizing the lower bound is equivalent to minimizing the KL divergence $KL(q(z)||p(z \mid o, \theta))$, which yields $q(^{t+1}) = p(z \mid o, \theta)$.
- **M-step**: Fix q^{k+1} and solve for the model parameters' $\theta^{k+1} = \arg\max_\theta \mathcal{F}(q^{k+1}, \theta)$. The second term of Eq. 2.6 is independent of θ so it is enough to maximize the energy term.

Specifically for HMMs (with parameters $\pi = p(z), T = p(z_n \mid z_{n-1}), M = p(o_n \mid z_n)$), the EM algorithm (Baum et al., 1970; Welch, 2003) estimates state and transition marginals in the E-step, using the forward-backward algorithm in Eq. 2.7. In the forward-backward algorithm, forward variables are update forward $\alpha_{n+1} = T\text{diag}M_{o_{n+1}}\alpha_n$ using $\alpha_1^i = M_{o_1}^\top \pi$. Backward variables get updated $\beta_n = \beta_{n+1}^\top T\text{diag}M_{o_{n+1}}$, where $\beta_\ell = 1$. Consider the parameters of the model $\theta^t = (T, M)$ in Section 2.3.1, and let $\boldsymbol{o}_{1:\ell}$ refer to the full sequence of observations of size ℓ. We define the forward variables to be $\alpha_n^j = p(\boldsymbol{o}_{1:n}, z_n = z_j \mid \theta)$, and we refer to the backward variables as $\beta_n^j = p(\boldsymbol{o}_{n+1:\ell} \mid z_n = z_j, \theta) \in \mathbb{R}$ such that $\beta_n, \alpha_n \in \mathbb{R}^K$.

$$p(Z_n = z_j \mid \boldsymbol{o}_{1:\ell}, \theta^t) = \frac{\alpha_n^j \beta_n^j}{\alpha_n^\top \beta_n} \tag{2.7}$$

$$p(Z_n = z_j, Z_{n+1} = z_i \mid \boldsymbol{o}_{1:\ell}, \theta^t) = \frac{\beta_t^i T_{i,j} M_{o_{t+1},i} \alpha_n^j}{\beta_n^\top T\text{diag}M_{o_{n+1}}\alpha_n} \tag{2.8}$$

In the M-Step we update the HMM parameters using the marginals

$$T_{i,j}^{t+1} = \frac{\sum_{n=1}^{\ell-1} p(Z_n = z_j, Z_{n+1} = z_i \mid \boldsymbol{o}_{1:\ell}, \theta^t)}{\sum_{n=1}^{\ell-1} p(Z_n = z_j \mid \boldsymbol{o}_{1:\ell}, \theta^t)} \tag{2.9}$$

$$M_{m,i}^{t+1} = \frac{\sum_{n=1}^{\ell} \mathbb{1}_{O_n=m} p(Z_n = z_j \mid \boldsymbol{o}_{1:\ell}, \theta^t)}{\sum_{n=1}^{\ell} p(Z_n = z_j \mid \boldsymbol{o}_{1:\ell}, \theta^t)} \tag{2.10}$$

This process is repeated until a local maximum of the free energy is reached. This is equivalent to maximizing the complete likelihood of the observations since the entropy does not depend on the parameters θ. This algorithm presents a tractable solution to maximum likelihood estimation but provide only local convergence guarantees, being sensitive to initialization. Furthermore, this method is known to suffer from slow convergence, and typically requires several inference passes over the data, which can be prohibitively expensive in large-scale datasets.

2.4.2 *Moment-based Learning*

Moment-based learning offers an alternative approach to learn latent variable models with better convergence guarantees. These methods yield (asymptotically) statistically optimal solutions, and can be computed efficiently. Consider the multi view model in Section 2.3, where we represent observations as vectors in $\boldsymbol{o_i} \in \mathbb{R}^V$ for every symbol in $o_i \in \Sigma$. We assume a generative perspective where observations are generated from conditionally independent hidden variables Z with values in $[K]$.[2] We define second and third order moments as uncentered covariances $W_2 \in \mathbb{R}^{V\times V}$ and $W_3 \in \mathbb{R}^{V\times V\times V}$ respectively. When the observation representation is encoded as indicator vectors,

[2] In topic models the observations are generated from the same hidden variable Z, however for arbitrary models we simply require the different views to be conditionally independent given the variable Z.

i.e., $\boldsymbol{o}_i = \boldsymbol{e}_i \in \mathbb{R}^V$ is zero everywhere except in the i-th coordinate, the moment matrices and tensors may be interpreted as probabilities, otherwise they appear as expectations.

$$\begin{aligned} W_2 &= \mathbb{E}[\boldsymbol{o}_1 \otimes \boldsymbol{o}_2] = \mathbb{E}[\mathbb{E}[\boldsymbol{o}_1 \otimes \boldsymbol{o}_2 \mid Z]], \\ &= \mathbb{E}[\mathbb{E}[\boldsymbol{o}_1 \mid Z] \otimes \mathbb{E}[\boldsymbol{o}_2 \mid Z]] \\ &= \mathbb{E}[Z \otimes Z](M_1, M_2) = Z_2(M_1, M_2) \end{aligned} \tag{2.11}$$

where $M_i = \mathbb{E}[\boldsymbol{o}_i \mid Z] \in \mathbb{R}^{V \times K}$ is the conditional moment w.r.t. view i, and $Z_2 \in \mathbb{R}^{K \times K}$ represents the marginals. Additionally we define third order moments as a 3-mode tensor $W_3 \in \mathbb{R}^{V \times V \times V}$, see Figure 2.5

$$\begin{aligned} W_3 &= \mathbb{E}[\boldsymbol{o}_1 \otimes \boldsymbol{o}_2 \otimes \boldsymbol{o}_3] \\ &= \mathbb{E}[Z \otimes Z \otimes Z](M_1, M_2, M_3) = Z_3(M_1, M_2, M_3) \end{aligned} \tag{2.12}$$

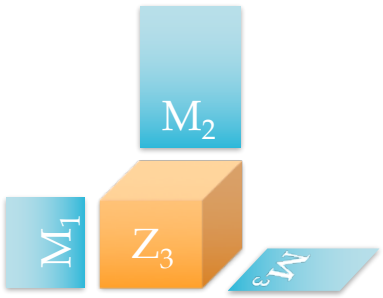
where $Z_3 \in \mathbb{R}^{K \times K \times K}$ is the inner tensor representing the marginals.

In this section we describe several approaches to learn conditional and marginal factors when we know the mapping to the hidden state variables— in the family of latent variable learning. The first is based on *tensor factorization*, and has been used in a wide range of applications, such as community detections (Anandkumar et al., 2012a), parsing (Cohen et al., 2013a), knowledge base completion (Chang et al., 2014; Singh et al., 2015), topic modeling (Anandkumar et al., 2015), mixture models (Anandkumar et al., 2012b), graphical models (Chaganty et al., 2014a). For many types of multi-view models it is sufficient to learn from low order moments typically third-order, hence the need for general tensor decomposition or factorization algorithms in Section 2.4.3.

The second approach relies on a non-negative factorization of moment matrices. This method requires certain separability conditions on the hidden states, and exploits the inherent structure of the latent model to obtain the different factors. It aims to find sets of observations to serve as surrogate representations for hidden variables. These are known as *anchor methods*, and have been mostly applied to topic modeling in Section 2.4.4.

Additionally we revisit different forms of obtaining the marginals in Eq. 2.11: using a pseudo-inverse and a likelihood based approach in Section 2.4.5.

(a) Second order moment factorization into conditional moments (blue), for two views $M_1 \in \mathbb{R}^{d_1 \times K}, M_2 \in \mathbb{R}^{d_2 \times K}$, and marginals (orange) represented by $K \times K$ matrix Z_2.



(b) Third order moment factorization into conditional moments (blue), for two views $M_v \in \mathbb{R}^{d_v \times K}$, $v = [1, 2, 3]$, and marginals (orange) represented by a $K \times K \times K$ tensor Z_3.

Figure 2.5: Moment decomposition. Tensor factorization approaches find a decomposition of moments of third order, anchor learning deals with factorization of second order moments.

2.4.3 *Tensor Methods*

Tensor factorization algorithms have been proposed as an alternative way of learning the mapping to the hidden state variables. This class of methods has the advantage of directly allowing us to estimate the conditionals from moment statistics. There are several tensor decomposition methods that attempt to find a low rank decomposition of a

tensor. Let $\otimes$ to denote the tensor product, and given a pth-order tensor $T^{\otimes p} = \mathbb{E}[o_1 \otimes \ldots \otimes o_p] \in \mathbb{R}^{n^p}$[3], we consider its decomposition into rank-one terms— *canonical polyadic decomposition* (Hitchcock, 1927) :

[3] also known as p-mode tensor.

$$T = \sum_{i=1}^{K} \gamma_i \boldsymbol{\mu}_1^i \otimes \boldsymbol{\mu}_2^i \otimes \cdots \otimes \boldsymbol{\mu}_p^i \tag{2.13}$$

where K is defined to be the rank of the tensor, and $\boldsymbol{\mu}_j^i = \mathbb{E}[o_j \mid z_i] \in \mathbb{R}^n \; \forall_{j\in[p],i\in[K]}$ its factors. When $\boldsymbol{\mu}_1^i = \ldots = \boldsymbol{\mu}_p^i = \boldsymbol{\mu}^i$ for all i the tensor is said to be *symmetric*, and if $\{\boldsymbol{\mu}_j^i\}_{i\in[K]} \; \forall_{j\in[p]}$ form an orthogonal set then we say the factorization is *orthogonal*. The rank of the tensor is defined as the smallest non-negative integer K such that Eq. 2.13 applies. The rank of a 2nd-order tensor reduces to the usual definition of matrix rank, and its decomposition is equivalent to the SVD. However, the concept of tensor rank is not as clearly defined as the case of matrix rank, for instance the rank of a symmetric tensor is not guaranteed to be finite (Comon et al., 2008). Tensor decomposition aims to find the conditional moments by decomposing the tensor into its factors in Eq. 2.13.

In the MoM, we build an empirical tensor in Eq. 2.14 from consecutive samples of the three view model in Figure 2.5b— by the tensor product of observations of the three views of the model $\{\boldsymbol{o}_1, \boldsymbol{o}_2, \boldsymbol{o}_3\}$, $o_i \in \mathbb{R}^V \; i \in [3]$.[4]

[4] when observation vectors are one-hot encodings the tensor corresponds to probabilities in $\mathbb{R}^{\otimes^3}$, but in general for continuous representations it represents an expectation.

$$\hat{T} = \frac{1}{t} \sum_{t=1}^{D} \boldsymbol{o}_1^t \otimes \boldsymbol{o}_2^t \otimes \boldsymbol{o}_3^t \tag{2.14}$$

The empirical tensor can be described in terms of additive noise $W \in \mathbb{R}^{V\times V\times V}$

$$\hat{T} = \sum_{i=1}^{K} \gamma_i \boldsymbol{\mu}_1^i \otimes \boldsymbol{\mu}_2^i \otimes \cdots \otimes \boldsymbol{\mu}_p^i + \epsilon W. \tag{2.15}$$

In the infinite data limit $t \to \infty$ then $W \to 0$ and the tensor factorization approaches the true CPD factorization Eq. 2.13, resulting in a consistent moment based estimation. The eigenfactors correspond to columns of the conditional moments $\boldsymbol{\mu}_j^i = [M_j]_i = \mathbb{E}[o_j \mid z_i]$.

In an HMM for example, the three views are equivalent to: past corresponds to o_1, current observations to o_2 and future observations to o_3, see Figure 2.4. When the observations in the present are indicator representation this yields the emission probabilities directly, however possible extensions may be considered for arbitrary features of observations.The observations matrix corresponds to the conditional moment w.r.t. the present view $O = M_2 = p(o_2 \mid z) = \boldsymbol{\mu}_2$.

tensor factorizations

Work of Anandkumar et al. (2012b) provides a robust algorithm based on a tensor power method (TPM) which provides global convergence guarantees for symmetric tensors with orthogonal factorization.

First, it reduces the symmetric tensor into an orthogonal decomposable form via a whitening step, and secondly it performs a power method algorithm to find its eigenvector/value pairs. The framework used in this approach is very restrictive, requiring further procedures to transform moments into a symmetric form. In most methods, the model is inherently non-symmetric. HMMs, for instance, are comprised of three independent views: past (o_1), future (o_3) and present (o_2). Performing symmetrizations, in these cases, may lead to information losses, specifically, because they are done through pseudo-inverses of moment matrices (Souloumiac, 2009). Another drawback of this approach is that, if the moments are not retrieved from the true model, the recovery process is not robust. Additionally, this method suffers from the non-negativity problem, where parameters need to be converted back into the probability simplex.
More recent work of Anandkumar et al. (2014) (ALS) describes a non-orthogonal tensor decomposition relying on an alternating least squares variant. This is a generalization to non-symmetric tensors. Finding eigenvectors in this setting is a non-convex problem. This work provides an alternating power iteration that deflates each mode of the tensor asymmetrically, and requires an extra clustering method at the end. Additional work of Shalit et al. (2014) explores an algorithm based on coordinate descent methods for orthogonal tensor factorization, which involves Givens rotations and obtains better empirical results. Simultaneous diagonalization by Lathauwer (2006) is applicable to general tensor factorizations, but requires solving $\mathcal{O}(V^4)$ linear system. Kuleshov et al. (2015) provides an additional tensor factorization approach that works for the non-orthogonal setting with arbitrary incoherence, but requires non-singular factors. Alternative tensor factorizations exist, such as Tucker decomposition that attempts to decompose a tensor into a set of matrices and a core low dimensional symmetric tensor (Anandkumar et al., 2015).

2.4.4 *Anchor Learning*

In this section, we describe an alternative approach for learning hidden variable models using moment techniques. The development of these methods originated in *non-negative matrix factorization*(NMF) algorithms for probabilistic topic modeling.

non-negative matrix factorization (NMF)

Consider a collection of D documents where each document n may be classified with a single topic $Z_n = z_j$ among a total of $[K]$ distinct topics. Let words be discrete observations $O_n = o_i$, in a vocabulary of V words, and assume N words are drawn independently for each topic according to a multinomial distribution. Let $[\boldsymbol{\mu}_j]_i = p(O_n = o_i \mid$

$Z_n = z_j$) denote the word-topic distribution, and the matrix $M = [\boldsymbol{\mu}_1 \mid \boldsymbol{\mu}_2 \mid \cdots \mid \boldsymbol{\mu}_K] \in \mathbb{R}^{d \times K}$ denote the matrix of conditional probabilities. Observations in this graphical model are generated as follows:

- a document's topic is drawn from the multinomial distribution defining the topic prior distribution $w_j = p(Z_n = z_j) \forall_{z_j \in [K]}$, where w_j is an element of the probability vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_K) \in \Delta^{K-1}$;
- given a topic Z_n the document's L words are drawn independently according to the word-topic distribution $\boldsymbol{\mu}_j \in \Delta^{d-1}$. We assume $L \geq 3$ words per document.

This graphical model can be represented by the structure depicted in Figure 2.6. To correctly estimate the conditional distributions we require that the each topic has non-zero probability $w_j > 0 \forall_{j \in [K]}$, and $\boldsymbol{O}$ has rank K, which ensures that any topic's word distribution is linearly independent of the other topics' word distributions. Specifically each document is assumed to be represented as a convex combination of the topic vectors $\boldsymbol{\mu}_j \; \forall j \in [K]$ (Papadimitriou et al., 1998; Hofmann, 1999). This combination may arise from different distributions: multinomial for topic models or Dirichlet in models like LDA (Blei et al., 2003). We consider the moment matrix $W \in \mathbb{R}^{V \times D}$ to represent the words frequencies of each document $C_n = c_n \in [D]$, as column vectors in W—*bag of words* representation. We represent words as indicator vectors in the vocabulary $\boldsymbol{o}_i = \boldsymbol{e}_i \in \mathbb{R}^V$ is zero everywhere except in the i-th coordinate for word $i \in [V]$. Using this representation for words and documents the moment matrix encodes conditional probabilities $W_{i,n} = \mathbb{E}[\boldsymbol{o}_i \mid c_n] = p(O = i \mid c_n)$. Considering this inherent structure of the problem we may decompose the moment matrix into a factorization of two non-negative matrices, its NMF, see Figure 2.7.

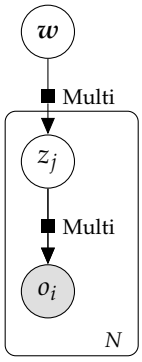


Figure 2.6: Single topic model in plate notation. Topic Z_j generates N observations O_i from with probability $\boldsymbol{\mu}_j$.

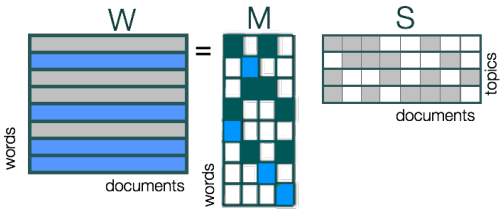


Figure 2.7: Non-negative matrix factorization of document matrix $W \in \mathbb{R}^{V \times D}$, into conditional moments $M \in \mathbb{R}^{V \times K}$ word-topic distribution, and topic-document distribution $S \in \mathbb{R}^{K \times D}$.

$$W = MS, \tag{2.16}$$

where $M = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k] \in \mathcal{R}^{V \times K}$ corresponds to a matrix of mean word-per-topic vectors or *conditionals* in Eq. 2.3, with each column $\boldsymbol{\mu}_j = p[O \mid Z = z_j] \in \Delta^{V-1}$, $\forall_{z_j \in [K]}$. $S_{j,n} = p(Z = z_j \mid c_n)$ is the matrix of topic-per-document distribution $S \in \mathbb{R}^{K \times D}$. The latter is specified by an unknown distribution, for instance a Dirichlet distribution for LDA models or logistic normal distribution for correlated topic models.

Alternative approaches based on spectral decompositions are commonly used to estimate M, such as latent semantic indexing (LSI) (Deerwester et al., 1990a), see Section 2.4. This approach attempts to find a low dimensional projection of maximum variance onto a sub-space of dimension K, the number of topics. This solution is given in terms of the singular vector recomposition of $\hat{W}_K = \text{SVD}(W) = \sum_{j=1}^{K} \boldsymbol{\mu}_j \sigma_j \boldsymbol{v}_j^\top$,

its best rank-K approximation, from its truncated singular value decomposition. The vectors $\boldsymbol{\mu}_j$, $\forall_{j\in[K]}$ form an orthonormal basis, where each vector corresponds to a single topic. Since this basis is orthogonal we only consider a single topic per document, *pure documents* (Papadimitriou et al., 1998). These spectral based methods decompose the matrix into positive and negative values, since it involves factors $\boldsymbol{\mu}_j, \boldsymbol{v}_j$ corresponding to the singular vectors, which could be a disadvantage when we require a probabilistic interpretation of the factorization. There exists additional work on non-negative matrix factorization that uses the singular vectors of W to recover span of the column space of M, and use it only as a means to evaluate document similarity.

More recent work of Arora et al. (2013) attempts to recover the non-negative factors via a distinct approach, in which they require instead a separability between topics and always maintain non-negative factors. This approach allows mixtures of topics per document and learns the non-negative factors of W in $\mathcal{O}(\log VK^6)$. We describe it in more detail below.

The objective in topic model learning is to estimate the matrix M in Eq. 2.16 by looking at co-occurrence of events or moments of second order: $\bar{Q} = \mathbb{E}[WW^\top] = p(O_1, O_2) \in \mathbb{R}^{V\times V}$ which we denote *word co-occurrence* matrix, or *moment matrix*. The true matrix, *i.e.*, the co-occurrence of infinitely many samples, can be decomposed into two factors according to its conditional independence structure $O_1 \perp O_2 \mid Z_1$: the conditionals $M = p(O_v \mid Z_v), v = \{1, 2\}$ and the marginals $Z = p(Z_1, Z_2) \in \mathbb{R}^{K\times K}$ from $S = p(Z \mid C) \in \mathbb{R}^{K\times D}$:

$$
\begin{aligned}
p(O_1, O_2) = \bar{Q} = [WW^\top] = M_1\mathbb{E}[SS^\top]M_2^\top = MZM^\top \\
= \sum_{z_k=1}^{K} p(O_1 \mid Z_1 = z_k)p(Z_1, Z_2)p(O_2 \mid Z_1 = z_k)
\end{aligned}
\tag{2.17}
$$

See Figure 2.7. In practice, we aim to estimate conditionals and marginals (M, Z) from empirical counts, *i.e.*, finite samples, which we denote as $\bar{Q}$. The most common approach to solve this type of problem resorts to likelihood estimation, by finding the matrix M that generates $W = MS$ with highest probability (Blei et al., 2003). MLE for topic modeling is NP-hard (Vavasis, 2009) and approximate solutions suffer from the problems discussed in Section 2.4.1. Sanjeev Arora (2012) proposed an algorithm that learns both M, Z from unsupervised data without any assumption on the nature of the distribution that generates topics for each document. This approach considers instead the decomposition of the conditional moment $Q_{m,n} = p(O_2 = o_m \mid O_1 = o_n)$ where

$Q \in \mathbb{R}^{d \times d}$ is obtained by column normalization of Q:

$$Q = M\,\Gamma = \sum_{z_j=1}^{K} p(O_2 \mid Z_1 = z_j)p(Z_1 = z_j \mid O_1), \qquad (2.18)$$
$$\text{s.t. } \mathbb{1}^\top Q = \mathbb{1}$$

and $\Gamma = p(Z_1 \mid O_1) \in \mathbb{R}^{K \times V}$ is a stochastic matrix of coefficients. In this work we assume the word-topic matrix M is *p-separable* (Donoho et al., 2003). A word-topic matrix is *α-separable* if for each topic z_s, there exists a word $a_s \in \Sigma$ such that $p(a_s \mid z_s) \geq \alpha$, and $p(a_s \mid z_j) = 0$ for $j \neq s$, $\alpha > 0$. Then we say this word in an *anchor word* for topic z_s. *anchor word*
Anchor words are unambiguous indicators of the existence of the topic in the document, since there is no other topic that generates this word. Hence we may use them as surrogate topic representations, for a given anchor word for state/topic z_s $a_s \in \Sigma$, we may write $p(Z_1 = z_s \mid O_1 = a_s) = 1$, such that the conditional moment factorization becomes:

$$\begin{aligned} Q_{m,a_s} &= \sum_{z_j=1}^{K} p(O_2 = m \mid Z_1 = z_j)p(Z_1 = z_j \mid O_1 = a_s) \qquad (2.19)\\ &= p(O_2 = m \mid Z_1 = z_s)p(Z_1 = z_s \mid O_1 = a_s) \\ &= p(O_2 = m \mid Z_1 = z_s) \end{aligned}$$

The rows of the conditional moment matrix may be regarded as a convex combinations of the columns corresponding to anchor words:

$$\begin{aligned} Q_{m,n} &= \sum_{j=1}^{K} p(O_2 = m \mid Z_1 = z_j)p(Z_1 = z_j \mid O_1 = n) \qquad (2.20)\\ &= \sum_{j=1}^{K} Q_{m,a_j} p(Z_1 = z_j \mid O_1 = n) = \sum_{j=1}^{K} Q_{m,a_j}\Gamma_{j,n} \\ &= R_m \gamma_n \end{aligned}$$

where $R_m = [Q_{m,a_1}, \dots Q_{m,a_K}]$, $R \in \mathbb{R}^{V \times K}$ corresponds to all the columns of anchor words for each topic, and $\gamma_n = p(Z_1 \mid O_1 = n) \in \mathbb{R}^K$ is the convex coefficient associated with word n and corresponds to each column in Γ. We may turn Eq. 2.20 into an algorithm by minimizing a loss between each row Q_m and the factorization $R_m\Gamma$, such that $\mathbb{1}^\top \gamma_n = \mathbb{1}$. Algorithm 1 shows the algorithm proposed by Sanjeev Arora (2012) to recover M, Z, using a KL divergence loss $\text{loss}(Q_m, R_m\Gamma) = \text{KL}(Q_m \| R_m\Gamma)$ or a quadratic loss $\text{loss}(Q_m, R_m\Gamma) = \|Q_m - R_m\Gamma)\|_2^2$. The conditionals are estimated using Bayes' rule, where $\boldsymbol{q} = p(O) = \sum_n p(O_1 = m, O_2 = n)$:

$$M_{m,j} = \frac{p(Z_1 = z_j \mid O_1 = m)p(O_1 = m)}{\sum_{n=1}^{V} p(Z_1 = z_j \mid O_1 = n)p(O_1 = n)} = \frac{\Gamma_{j,m}^\top \boldsymbol{q}_m}{\Gamma_{j,\cdot}^\top \boldsymbol{q}}, \qquad (2.21)$$

Algorithm 1: NMF: Non-Negative Matrix Factorization: find conditionals M and marginals Z.

Input: Anchor set $\mathcal{A}$, moment matrix Q

1: Column normalization of $Q = \bar{Q}/\bar{Q}\mathbb{1}$.
2: **for** each word $m = 1, \ldots, V$ **do**
3: Find the coefficients $\gamma_m^* = \arg\min_{\gamma_m} \text{loss}(Q_m, R_m\Gamma)$,
4: s.t. $\mathbb{1}^\top \gamma_m = \mathbb{1}$, and $\Gamma_{m,j} \geq 0$, $\forall_{j \in [K]}$.
5: Estimate conditionals $M = \text{diag}(\boldsymbol{q})\Gamma$ and normalize columns to 1.
6: Estimate marginals $Z = M^\dagger Q M^{\dagger\top}$ via §2.4.5.
7: **Return:** M, Z.

Algorithm 2: Find Anchors

Input: N points in V dimensions $\mathcal{D} = \{d_1, \ldots, d_V\}$.

1: Project each point $d_i \in \mathcal{D}$ onto a $4logV/\epsilon$ dimensional subspace.
2: Add the furthest point $d_j \in \mathcal{D}$ from the origin $\mathcal{A} \leftarrow \{d_j\}$
3: **for** each vertex $j = 1, \ldots, K-1$ **do**
4: Find the furthest point $d_i \in \mathcal{D}$ to the span$(\mathcal{A})$,
5: Add the point to the anchor set $\mathcal{A} \leftarrow \mathcal{A} \cup \{d_i\}$.
6: $\mathcal{A} = \{v_1, \ldots, v_K\}$
7: **for** $j = 1, \ldots, K$ **do**
8: Find the furthest point $d_i \in \mathcal{D}$ to the span$(\mathcal{A} \setminus \{v_j\})$,
9: Replace the vertex of the anchor set $\mathcal{A} \leftarrow \mathcal{A} \setminus \{v_j\} \cup \{d_i\}$.
10: **Return:** $\mathcal{A}$

while the marginals Z may be recovered from a pseudo-inverse approach, in Section 2.4.5

We describe an unsupervised algorithm that extract anchor words from unlabeled samples in Algorithm 3. The core idea of this method lies on the fact that anchor words are extreme points of the vocabulary—in a geometrical sense. In the infinite data limit the convex hull of the columns of Q correspond to the simplex of probabilities, whose vertices are associated with anchor words. In practice, however, we get empirical estimates of the simplex hence we only observe noisy perturbations of these vertices. The p-separability condition ensures a minimum distance between the vertices. Algorithm 3 provides a description of the approach, where it first iteratively finds points that are furthest apart from the subspace spanned by the current set of anchors, and it terminates when it finds the K-most-separated words.[5] This approach is robust to noise and behaves better as the distance of each vertex to the affine hull of the remaining vertices increases. Algorithm 3 provides the algorithm proposed by Sanjeev Arora (2012) to recover anchor words unsupervisedly in Eq. 2.22.

[5] The distance refers to the norm of the projection of each vertex v_j onto the orthogonal complement of the $span(\mathcal{A})$

Finally, after estimating the set of anchor words we can recover the convex coefficients for each symbol in the alphabet $\gamma_n = p(Z_1 \mid O_1 = n)$, $\forall_{n \in \Sigma}$, in Eq. 2.20. Then, the conditional moments will be deter-

mined via Bayes' rule, in Algorithm 1.

2.4.5 *Recovering Marginals*

Following the estimation of the conditional moments $M_v = \mathbb{E}[\boldsymbol{o}_v \mid Z] \in \mathbb{R}^{V \times K}$ for each conditionally independent view $\forall_{v \in [1,2]}$, we are left with estimating the state marginals Z, in Figure 2.3a. The conditionals may be estimated via anchor learning for topic modeling/ mixture models, in Section 2.4.4 or via tensor factorization for general graphical models, in Section 2.4.3. For most graphical models, and in specifically for sequential models, is it enough to consider hidden marginals of second order $Z_2 = \mathbb{E}[Z_1 \otimes Z_2]$ to correctly estimate the model— transition probabilities for HMMs. Consider the moment decomposition in Eq. 2.11 of W_2 in terms of the hidden marginals Z_2 where

$$W_2 = Z_2(M_1, M_2) = M_1 Z_2 M_2^\top \in \mathbb{R}^{V \times V}.$$

We can solve for Z_2 by matrix inversion where $M_v^\dagger \in \mathbb{R}^{K \times V}$ denotes the Moore-Penrose pseudo-inverse $\forall_{v \in [1,2]}$

$$Z_2 = W_2(M_1^\dagger, M_2^\dagger) = M_1^\dagger W_2 M_2^{\dagger\top} \tag{2.22}$$

and the analogous for third order moments

$$Z_3 = W_3(M_1^\dagger, M_2^\dagger, M_3^\dagger) \tag{2.23}$$

Eq. 2.22 recovers the hidden marginals of a set of hidden variables that constitute the joint probability of consecutive state variables in the latent variable model. In many models further computations will be required to retrieve the model parameters. In HMMs we recover the transition matrix by an additional step of normalization $[T]_{k,k'} = [Z_2]_{k,k'} / \sum_j [Z_2]_{k,j} \in \mathbb{R}^{K \times K}$.

Chaganty et al. (2014a) provides an alternative approach that builds upon a likelihood based approach (Lindsay, 1988). Making use of the already estimated conditionals M_v and the observed moments W_S they recover the marginals $Z_S = p(S)$ over a set of nodes $S = \{Z_1, \dots Z_m\}$ with certain conditional independent properties. Both pseudo-inverse approach and the convex composite likelihood retrieve the joint probability over states or marginals. In some models, such as HMMs we further require to transform marginals into model parameters, the conditional transition matrix $T = p(Z_1, Z_2) / \sum_{j=1} p(Z_1, Z_2 = j)$— generally in directed models this step comes down to normalizing the probability for each state with the parents states in the set S.

2.5 *Spectral learning of sequential systems*

Consider the existence of two sample spaces a state space $\mathcal{Z}$ and an observation space Σ and a many-to-one mapping from $\mathcal{Z}$ to Σ. Observed data $o \in \Sigma$ are realizations from Σ, while z are realizations from $\mathcal{Z}$, which are not directly observed. The former space consists of output observable quantities in a dynamical system, while the latter models the evolution of the observed process Σ implicitly, via its internal state representation, as shown in Figure 2.8.

In this section, we focus on a distinct line of work that focuses on modeling a dynamical system or a stochastic process without recovering the actual parameters of the model, in particular the conditional and hidden marginals, see Figure 2.3b. These methods provide alternative forms for predicting and generating sequences of observations by means of observable representations alone (Jaeger, 2000; Balle et al., 2012a; Bailly et al., 2013a). They are however less amenable to combine with likelihood-based approaches. Parikh et al. (2012a) provided an approach to estimate these observable operator representations for general graphical models which are bottlenecked using latent junction tree representations.

In this section, we introduce sequential models under the perspective of dynamical system learning (Thon et al., 2015). We define a common representation for sequential models in Section 2.5.1 and provide a description and connection of closely related algebraic formalisms for different communities: observable operator models for stochastic processes in Section 2.5.3, predictive state representations for controlled systems in Section 2.5.4, and weighted finite automata in formal language theory for stochastic languages in Section 2.5.2. we discuss how these different representations are connected, their differences and commonalities.

2.5.1 *Sequential models*

Consider observation in a sequential system can be approximated by some finite dimensional random vector $O = (O_i)_{i \in [d]}$, where O_i is stationary with values in a finite alphabet $\Sigma = \{o_1, \ldots, o_d\}$ of d possible symbols. Let $\Sigma^* = \{*\} \cup \Sigma \cup \Sigma^2 \cup \ldots$ be the Kleene closure of Σ, *i.e.* the set of all finite sequences with elements in Σ, plus the empty sequence $*$. We define a sequence as consecutive realizations of symbols in the alphabet $\boldsymbol{o}_{1:n} = o_1 o_2 \ldots o_n \in \Sigma^*$. Consider a more general description of sequential systems, as defined in Thon et al. (2015).

Characterization Functions

Consider functions $f : \Sigma^* \to \mathbb{R}^K$ that map sequences of observations to

the field of real values $\mathbb{R}^K$. These functions can characterize distributions over sequences in Σ^*, if we consider a mapping to the simplex of probabilities Δ^{K-1}, used to describe probabilistic languages (Weighted Automata), or simply probabilities of observations in the context of stochastic processes, if we consider a mapping over the interval $[0, 1]$ (which is the case of OOMs and PSRs).

For any given function we define $f_{\boldsymbol{h}} : \Sigma^* \to \mathbb{R}$ to be a function in a Banach space of functions $\mathcal{F}$ where the sequence $\boldsymbol{h}$ is a sequence of observations of arbitrary size. We further distinguish between the sequences of already observed events at time n, which we call *histories* $\boldsymbol{h} \equiv h_n = \{o_n, o_{n-1}, \ldots, o_1\} \in \Sigma^*$ and sequences of future observations, which we call *tests* $\boldsymbol{t} \equiv t_n = \{o_{n+\ell}, \ldots, o_{n+2}, o_{n+1}\} \in \Sigma^*$, as depicted in Figure 2.8. The state is considered to be sufficient to model the evolution of the dynamical system and is closely related to the functions $f_{\boldsymbol{h}} \in \mathcal{F}$, which evaluate sequences of observations. Let the concatenation of two sequences be defined as $\boldsymbol{ht}$, evaluating the concatenated sequence results in $f_{\boldsymbol{h}}(\boldsymbol{t}) = f(\boldsymbol{ht})$. The functions $f_{\boldsymbol{h}} : \Sigma^* \to \mathbb{R}$ fully characterize the evolution of the system. Let the functional Banach space $\mathcal{F}$ be defined by the closure of the linear span of future prediction functions $\mathcal{F} = \text{span}\{f_{\boldsymbol{h}} \mid \boldsymbol{h} \in \Sigma^*\}$. We assume this space has finite dimension and we say the *system dimension* is $\dim(\mathcal{F}) = d \leq \infty$. If for a given function $f : \Sigma^* \to \mathbb{R}$ the finite set of functions spans $\mathcal{F}$:

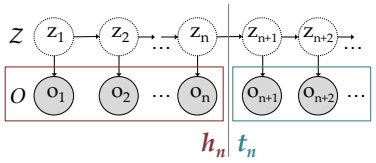


Figure 2.8: Observations $\mathcal{O}$ (bottom), generated from a hidden process $\mathcal{Z}$ (upper); histories $\boldsymbol{h}_i$ (left) and tests $\boldsymbol{t}_i$ (right).

h *history*

t *test*

$$\mathcal{F} = \text{span}\{f_{\boldsymbol{h}_i} \forall_i \in N\}, \tag{2.24}$$

then we say these set forms a *basis* of $\mathcal{F}$. If the number of functions is $N = d = \text{rank}(\mathcal{F})$, then we say they form a *minimal* basis. In the planning literature, $\dim(\mathcal{F})$ is also known as *linear dimension* of a dynamical system (Singh et al., 2004b). The dimension corresponds to the number of elements in the basis for finite dimensional spaces.

linear dimension

Linear Transition Operators

Next, we define linear operators that transform these characterizing functions. These observable operators linearly weight the characterizing functions for each observation, and do not depend on the choice of basis for $\mathcal{F}$. We define a linear operator $A_o : \mathcal{F} \to \mathcal{F}$ that encodes information about the evolution of the system due to an incoming observation $o \in \Sigma$. We denote this operator an *observable operator* .

A_o observable operator

$$A_o(f_{\boldsymbol{h}}) = f_{o\boldsymbol{h}} \in \mathcal{F}, \text{ and} \tag{2.25}$$

$$A_{\boldsymbol{h}} = A_{o_n}(A_{o_{n-1}}(\ldots A(o_1))) = A_{o_n} \circ A_{o_{n-1}} \circ \ldots \circ A_{o_1} \tag{2.26}$$

where $o\boldsymbol{h} \in \Sigma^*$ represents the concatenation of the sequence $\boldsymbol{h} \in \Sigma^*$ with the consecutive observation $o \in \Sigma$. We can then denote the probability of a sequence in terms of these operators. Consider now a *linear*

functional $\alpha_\infty : \mathcal{F} \to \mathbb{R}$ that operates on the characterizing functions:

$$\alpha_\infty(f_{\boldsymbol{h}}) = f(\boldsymbol{h}) \qquad \forall \boldsymbol{h} \in \Sigma^*, \tag{2.27}$$

in particular for the empty symbol $*$ and the function generating the empty symbol f_* we have $\alpha_\infty(f_*) = f(*)$, and $A_{\boldsymbol{h}}(f_*) = f_{\boldsymbol{h}}$. We also define a *normalization function* f_Σ such that:

$$f_\Sigma(\boldsymbol{h}) = \sum_{o \in \Sigma} f(o\boldsymbol{h}). \tag{2.28}$$

Sequential Models

sequential model

We define a *Sequential Model* $\boldsymbol{A}$ of dimension K as the tuple $\langle \mathbb{R}^K, \alpha_\infty, \{A_o\}_{o\in\Sigma}, \alpha_0 \rangle$, where, $\alpha_0 \in \mathbb{R}^K$ denotes the *initial state* vector, and $A_o \in \mathbb{R}^{K\times K}$, $\forall_{o\in\Sigma}$ represents *state transition matrices*. The *state* of the process at time n $\alpha_n \in \mathbb{R}^K$ is given by:

state of system

$$\alpha_n = A_{\boldsymbol{h}}\alpha_0 = A_{o_n}A_{o_{n-1}}A_1\alpha_0. \tag{2.29}$$

Evaluating the state of the process can be done via the evaluation functional $\alpha_\infty \in \mathbb{R}^K$, such that its function $f_{\boldsymbol{A}} : \Sigma^* \to \mathbb{R}$ is given by[6]:

$$f_{\boldsymbol{A}}(\boldsymbol{h}) = \alpha_\infty^\top A_{\boldsymbol{h}}\alpha_0 \tag{2.30}$$

[6] This relation provides the connection between characterizing functions and the observable operators. Note that function evaluation $\in \mathbb{R}$, while observable operators define how the state changes according to an incoming observation $\in \mathbb{R}^{K\times K}$

rank of the system

We define the *rank* of the model to be $\text{rank}(f_{\boldsymbol{A}}) = \text{rank}(\boldsymbol{A})$.

Minimal Model

Let an arbitrary function $f : \Sigma^* \to \mathbb{R}$ of rank $d = \text{rank}(f) \leq \infty$, then there exists a d-dimensional model $\boldsymbol{A}$ such that $f = f_{\boldsymbol{A}}$. This refers to the existence of a model whose states are coordinate representations of characterizing functions in some basis of dimension d, such that $f(\boldsymbol{h}) = \alpha_\infty^\top A_{\boldsymbol{h}}\alpha_0$. When this basis is minimal then we say the model that originated from this basis is also *minimal*. For this case alone it is possible to establish an equivalence relation between models.

minimal model

model equivalence

Model Equivalence

Two sequential models $\boldsymbol{A}$ and $\boldsymbol{B}$ are said to be *equivalent* if they define the same function f, such that $f = f_{\boldsymbol{A}} = f_{\boldsymbol{B}}$. Consider two models $\boldsymbol{A} = \langle \mathbb{R}^m, \alpha_\infty, \{A_o\}, \alpha_0 \rangle$ and $\boldsymbol{B} = \langle \mathbb{R}^n, b_\infty, \{B_o\}, b_0 \rangle$ such that they are related by a bijective linear transformations $S \in \mathbb{R}^{n\times m}$, and $S' \in \mathbb{R}^{m\times n}$ such that $SAS' = \boldsymbol{B} = \langle \mathbb{R}^n, S'^\top \alpha_\infty, \{SA_oS'\}, S'\alpha_0 \rangle$. If the linear transformation is non-singular then $m = n$ and $S' = S^{-1}$ exists, and the two models are equivalent $f_A = f_B$, and

$$\boldsymbol{B} = S\boldsymbol{A}S^{-1} \tag{2.31}$$

is denoted a *conjugate equivalent* model.[7] If the sequential model is minimal then the linear transformation S corresponds to a change of

[7] In different communities this property is also known as conjugate invariance, model equivalence under conjugation, or simply equivalent models

basis of the space $\mathcal{F}$. Therefore, the conditions in Eq. 2.30 allow us to define a valid sequential model, but not uniquely, since we can find a different but equivalent model that describes the same function via conjugate equivalence, in Eq. 2.31. This additional degree of freedom, lets us represent sequential models with increasing representational power. We can interpret state as an arbitrary sufficient statistic of future observations. We will make use of this more general interpretation below, when we define spectral algorithms in Section 2.5.6. Since S is a bijective linear map, then S must be full column rank, and the terms SS^{-1} cancel out, such that:

model equivalence

$$\begin{aligned} b_\infty &= S^{-1}\alpha_\infty, \quad b_0 = S\alpha_0, \quad B_o = SA_oS^{-1}, \\ f(*) &= b_\infty^\top b_0 = (S^{-1}\alpha_\infty)^\top S\alpha_0 = \alpha_\infty^\top \alpha_0, \\ f(\boldsymbol{h}) &= b_\infty^\top B_{\boldsymbol{h}} b_0 \\ &= (S^{-1}\alpha_\infty)^\top (SA_{o_n}S^{-1}) \dots (SA_{o_1}S^{-1})(SA_{o_1}S^{-1})S\alpha_0 \\ &= \alpha_\infty^\top A_{\boldsymbol{h}} \alpha_0. \end{aligned} \tag{2.32}$$

We can see that sequential models are defined up to a similarity transform S, in the case of finite spaces $\mathbb{R}^m$.

Interpretable Model

Let $\boldsymbol{h}_1, \dots, \boldsymbol{h}_m$ be realizations of each set $T_j \;\; \forall_{j\in[m]} \subset \Sigma^*$. Consider the operators defined on each subset $T_j : A_{T_j} = \sum_{\boldsymbol{t}\in T_j} A_{\boldsymbol{t}}$, such that $f(\boldsymbol{h}T_j) = \sum_{\boldsymbol{t}\in T_j} f_{\boldsymbol{ht}} = \alpha_\infty(A_{\boldsymbol{t}}(f(\boldsymbol{h})))$. We say the occurrences of each subset $T_i \;\; \forall_{i\in[m]}$ are considered to be *characteristic events*, if the linear transformation $S \in \mathbb{R}^{m\times m}$

characteristic events

$$S = [\alpha_\infty^\top A_{T_1}, \dots, \alpha_\infty^\top A_{T_m}], \tag{2.33}$$

is non-singular. We consider an *interpretable* model w.r.t. subsets $T_1, \dots, T_m \subset \Sigma^*$ if the state of the model $\alpha_{\boldsymbol{h}}$ has an interpretable representation, *i.e*, corresponds to evaluations of the function on the subsets

$$\alpha_{\boldsymbol{h}} = [f(\boldsymbol{h}T_1), \dots, f(\boldsymbol{h}T_m)]^\top \in \mathbb{R}^m. \tag{2.34}$$

Then for any minimal model such that S is non-singular it is possible to find an *equivalent interpretable* model such that $\boldsymbol{B} = S\boldsymbol{A}S^{-1}$ with states $\alpha' = S\alpha = [\alpha_\infty^\top A_{T_1} A_{\boldsymbol{h}}\alpha_0, \dots, \alpha_\infty^\top A_{T_d} A_{\boldsymbol{h}}\alpha_0]^\top$.

interpretable model

Interpretability only requires mutually exclusive sets if the evaluation functional $\alpha_\infty^\top = [1, 1, \dots, 1]$ as in older interpretations of sequential models, then the sets $T_j, j \in [m]$ need to partition Σ^ℓ for sequences of length ℓ. Then the characteristic events correspond to a minimum number of sequences of observations that do not overlap with each other, but still uniquely describe the system. In this particular setting, the finite dimensional state vectors $b_{\boldsymbol{h}} \in \mathbb{R}^K$ can be interpreted as distributions over the simplex of probabilities Δ^{K-1}. Since characteristic

events are mutually exclusive they form a basis to represent future predictions:

$$b_\infty = \mathbb{1}_K$$
$$b_0 = p(\boldsymbol{t}_i|\epsilon)\forall_{\boldsymbol{t}_i \in \mathcal{T}_c} \tag{2.35}$$
$$b_{\boldsymbol{h}} = p(\boldsymbol{t}_i|\boldsymbol{h}) \in \mathbb{R}^K, \forall_{\boldsymbol{h}\in\Sigma^*, \boldsymbol{t}_i \in \mathcal{T}_c}\} \tag{2.36}$$
$$B_{o_n} = p(o_n|\boldsymbol{h})p(\boldsymbol{t}_i|\boldsymbol{h})^\top = \frac{p(o, \boldsymbol{h})}{p(\boldsymbol{h})} \in \mathbb{R}^{K\times K}, \forall_{o\in\Sigma, \boldsymbol{t}_i \in \mathcal{T}_c} \tag{2.37}$$

In this case, the set of future predictions/state estimates are aligned with the characteristic events and we say the basis is interpretable, but in general interpretability only requires Eq. 2.34.

Hankel Matrix

Hankel matrix duality
Hankel matrix

Next, we define Hankel matrices (Carlyle et al., 1971; Fliess, 1974) and their duality to minimal sequential models. The *Hankel matrix* is defined as a bi-infinite matrix $H \in \mathbb{R}^{\Sigma^*\times\Sigma^*}$, whose rows and columns are indexed by sequences in Σ^*, such that for all sequences $\boldsymbol{t}, \boldsymbol{t}', \boldsymbol{h}, \boldsymbol{h}' \in \Sigma^*$ with $\boldsymbol{ht} = \boldsymbol{h}'\boldsymbol{t}'$, then $H(\boldsymbol{h}, \boldsymbol{t}) = H(\boldsymbol{h}'\boldsymbol{t}')$. The Hankel matrix encodes a characterizing function $f : \Sigma^* \to \mathbb{R}$. Hankel matrices and characterizing functions are algebraically related, such that there exists a duality between them. On the one hand, the Hankel matrix H_f encodes f, such that its entries are given by $H_f(\boldsymbol{h}, \boldsymbol{t}) = f(\boldsymbol{ht})$. On the other hand, given a bi-infinite matrix H with the Hankel property, then there exists a unique function $f : \Sigma^* \to \mathbb{R}$, such that $H_f = H$ (Carlyle et al., 1971). A function $f : \Sigma^* \to \mathbb{R}$ can be realized by a sequential model if and only if $\text{rank}(H_f)$ is finite, and in that case $\text{rank}(H_f) = \text{rank}(f)$ is the minimal number of states of any model $\boldsymbol{A}$, also denoted as its linear dimension (Carlyle et al., 1971; Fliess, 1974). In the dynamical systems community the Hankel matrix is also known as the system-dynamics matrix. Later in Section 2.5.6 we will discuss learning algorithms for sequential models that make use of this duality by performing a spectral decomposition of the Hankel matrix, giving rise to the "spectral learning" algorithms.

2.5.2 *Weighted Automata*

Finite automata define a more general class of sequential models. They fall under the class of *finite state machines* (FSMs). These have been mostly developed in the context of formal language theory to model the generation of strings. Also from the dynamical systems perspective, FSM have been used to model the behaviour of a system that takes as input a sequence of observations and produces a sequence of outputs, by following a set of rules determined by the internal state of the system. The state admits only one out of a finite set of possible

Finite-State Machines (FSMs)

states, hence the name *finite state machines*. *Finite-state transducers* are the most general sequential model in FSMs where both an input and output label are associated with each transition. *Weighted finite-state transducers (WFSTs)* are finite-state transducers where each transition carries some weight in addition to the labels (Salomaa et al., 1978; Berstel et al., 1979). *Weighted finite automata* (WFA) can be regarded as a special case of *weighted finite-state transducers*, where the output label is omitted. WFSTs defined over the reals are closely related to IO-OOMs and PSRs (§ 2.5.4), while WFA are closely related to OOMs (§ 2.5.3). A lot of the analysis of WFA and WFSTs can be described by sequential systems characterization (Balle et al., 2011). As specific cases of WFA, we have the family of probability distributions generated by HMMs and probabilistic non-deterministic finite automata (PNFA), and other non-probabilistic distributions. WFA can also be defined more generally over an arbitrary semi-ring instead of the field of real numbers, giving rise to the more general case of *multiplicity automata* (Beimel et al., 2000). Sequential models can be though of as an instance of finite automata applied to sequential systems. In particular, the distributions over Σ^* realized by WFA, also denoted as *rational stochastic languages*, are a type of sequential model where $f(o) \geq 0, \sum_{o \in \Sigma^*} f(o) = 1$. This includes probabilistic automata with stopping probabilities (Dupont et al., 2005), hidden Markov models with absorbing states (Rabiner, 1989) and predictive state representations (Singh et al., 2004b). More details on spectral learning of WFA and FSMs are presented in Mohri (2009) and Balle (2013). A unified perspective of multiplicity automata, observable operator models and predictive state representations is provided by Thon et al. (2015).

Weighted finite-state transducers (WFSTs)

Weighted Automata (WA)

2.5.3 *Observable Operator Models (OOMs)*

Observable Operator Models (OOMs) were introduced by Jaeger (2000) as an algebraic tool to model stochastic processes (Heller, 1965; Ito, 1992; Upper, 1997). In these models characterizing functions represent future prediction functions, meaning they define probabilities over sequences of observations $f_{\boldsymbol{t}} = p(o_{n+\ell:n}) \forall_{\boldsymbol{t}} \in \Sigma^*$. Here, $f_{\boldsymbol{h}} : \Sigma^* \to \mathbb{R}$ defines a *future prediction function*, that describes the distribution of O after a previous realization of a history $\boldsymbol{h}$.

$\tau_{\boldsymbol{h}}(\boldsymbol{t})$ *future prediction function*

In order to predict future observations conditioned on past events $p(\boldsymbol{t}|\boldsymbol{h}) = p(o_{n+\ell}, \ldots, o_{n+1}|o_n, \ldots, o_1)$ we define normalized functions that correspond to conditional probabilities $f_{\boldsymbol{h}}(\boldsymbol{t})/f(\boldsymbol{h}) = p(\boldsymbol{ht})/p(\boldsymbol{h}) = p(\boldsymbol{t} \mid \boldsymbol{h}) \forall \boldsymbol{h}, \boldsymbol{t} \in \Sigma^*$. Joint distributions can be considered as conditional probabilities conditioned on the empty symbol $p(\boldsymbol{h}) \equiv p(\boldsymbol{h}|*)$. Therefore we can represent the system in terms of the set of all *conditional continuation probabilities* of the form $p(\boldsymbol{t}|\boldsymbol{h}), \forall_{\boldsymbol{t},\boldsymbol{h} \in \Sigma^*}$, see Figure 2.8.

$p(\boldsymbol{t}|\boldsymbol{h})$ *conditional continuation probabilities*

OOMs are typically defined over these conditional continuation probabilities, so for convenience we define OOMs as sequential models with normalized states. We define *Observable Operator Models* (OOMs) $\boldsymbol{A}$ of dimension K as sequential models for stochastic processes. We say $f_{\boldsymbol{A}}$ is a stochastic process if:

stochastic process

$$f_{\boldsymbol{A}}(*) = 1, \tag{2.38}$$

$$f_{\boldsymbol{A}}(\boldsymbol{h}) = \sum_{o \in \Sigma} f(o\boldsymbol{h}) \quad \forall \boldsymbol{h} \in \Sigma^*, \tag{2.39}$$

OOMs define a sequential model with normalized states:

$$\alpha_n = \frac{\alpha_{\boldsymbol{h}}}{\alpha_\infty^\top \alpha_{\boldsymbol{h}}} = \frac{A_{\boldsymbol{h}} \alpha_0}{\alpha_\infty^\top A_{\boldsymbol{h}} \alpha_0}. \tag{2.40}$$

In this sense, OOMs are a subset of sequential models applied to stochastic systems (Eqs.2.38-2.39), consequently they comply with all the properties derived for sequential models, in particular, interpretability. Interpretable OOMs' state space dimensions can be interpreted as probabilities of future predictions. Next, we discuss the connection of OOMs and HMMs.

Connection to Hidden Markov Models

from HMMs to OOMs

Any HMM can be described by a tuple $\langle T, O, \boldsymbol{\pi}, \mathbf{f}^* \rangle$, in Section 2.3.1, where $T \in \mathbb{R}^{K \times K} : T_{m,j} = p(Z_n = z_m \mid Z_{n-1} = z_j)$, $\forall_{z_m, z_j \in [K]}$ denotes its state transition matrix, $\boldsymbol{\pi} \in \mathbb{R}^K : \pi_j = p(Z_0 = z_j)$ its initial state distribution, $\boldsymbol{f^*} \in \mathbb{R}^K : f_j^* = p(\text{STOP} \mid Z_\ell = z_j)$ its final state distribution, and $O \in \mathbb{R}^{d \times K} : O_{i,j} = p(O_n = o_i \mid Z_n = z_j)$ its observation matrix giving the probability of state $z_j \in [K]$ generating an observation $o_i \in \Sigma$. We further write $O_i = p(O_n = o_i \mid Z_n) \in \mathbb{R}^K$ as rows of the full observation matrix.

HMMs can as well be described as a sequential models such that the state vector describes the belief (probability distribution) of the underlying hidden process. The sequential model representation of HMMs is equivalent to a an interpretable OOM such that we have:

$$HMM \equiv OOM = \langle \boldsymbol{\alpha}_\infty = \boldsymbol{f^*}, \{A_{o_i} = T \text{diag}(O_i)\}_{o_i \in \Sigma}, \boldsymbol{\alpha}_0 = \boldsymbol{\pi} \rangle \tag{2.41}$$

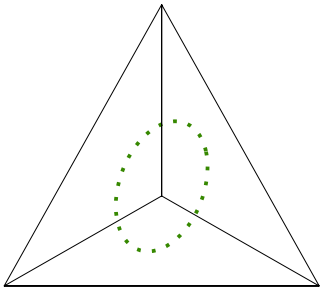


Figure 2.9: Probability clock of a 3 dimensional OOM. Green dots represent states in each linear operator transforms one point to the consecutive by some rotation A_o. HMMs would require an infinite amount of states to model the same example exactly.

One of the main differences between HMMs and OOMs is the state representation: in the former, the state describes the probability distribution of the process being in a given hidden state, so state dimensions must be non negative and sum to one together with the stopping symbol, whereas in the latter case, the state is simply a representation of future predictions in a given basis, where states and observable operators may be negative, meaning they need only obey to the constraints in Eqs.(2.38-2.39). This fact makes OOMs a more general class of models. There exists finite dimensional OOMs that cannot be described

by any finite dimensional HMM, such as the probability clock example (Jaeger, 2000). Figure 2.9 shows how the OOM operators that correspond to rotations in the ABC-plane cannot be modeled by any finite dimensional HMM. Rotation operators need negative entries, which contradicts the definition of HMM stochastic matrices. Conversion of an OOM into the closest HMM is still an open research problem. Nevertheless Hsu et al. (2009) and Anandkumar et al. (2012a) provide alternative approaches to recover the parameters of an HMM $\langle T, O, \boldsymbol{\pi}, \boldsymbol{f}^* \rangle$ from an equivalent OOM. We focus into these problems in detail in Section 2.5.6 and Section 2.4.3.

2.5.4 *Predictive State Representations and Controlled Processes*

Several sequential models have been proposed to handle dynamical systems that can be controlled from external input at each time step, which we denote *action*. Jaeger (1999) proposed *input-output OOMs* as an generalization of linear operator models, where each transformation is indexed by a pair of finite observations and actions $ao \equiv (a, o)$, $\forall_{o \in \Sigma_O, a \in \Sigma_I}$, such that $\Sigma = \Sigma_I \oplus \Sigma_O$, and sequences now refer to sequences of pairs of observations and executed actions $\boldsymbol{h} = (ao_n, ao_{n-1}, \ldots, ao_1) \forall_{\boldsymbol{h} \in \Sigma^*}$. Littman et al. (2001), Rosencrantz et al. (2004a), and Boots et al. (2011a) have also proposed models in a dynamical system perspective that model controlled processes. They denoted this family of models as *predictive state representations* (PSRs). We model future prediction functions through a K-dimensional state vector of an implicit process (Z_n).

Predictive State Representations (PSRs)

The tuple $\langle \mathbb{R}^K, \mathcal{T}, \tau_0, \tau_\infty, \{A_{oa}\}_{o \in \Sigma_O, a \in \Sigma_I} \rangle$ defines a *input-output OOM* (PSR) of the process (O_n). IO-OOMs are sequential models for a stochastic processes such that for any given function $f : \Sigma^* \to [0, 1]$:

$$f(*) = 1, \tag{2.42}$$

$$f(\boldsymbol{h}) = \sum_{o \in \Sigma_O} f(\boldsymbol{h}ao), \ \forall a \in \Sigma_A, \tag{2.43}$$

$$f(\boldsymbol{t} \mid \boldsymbol{h}) = f(\boldsymbol{h}\boldsymbol{t}) / f(\boldsymbol{h}), \text{ for } f(\boldsymbol{h}) \neq 0 \tag{2.44}$$

Here, f may describe a joint probability function $p(o_{n:1}, a_{n:1})$ or conditional probabilities of observations conditioned on actions $p(o_{n:1} \mid a_{n:1})$. PSRs describe the same process an IO-OOM describes, but in terms of an implicit state representation, which could be more representative than probability of tests conditioned on histories. PSRs can describe expected values of arbitrary statistics of the future. This representation provides a generalization to partial observable Markov decision processes, in the same way OOMs provide a generalization to HMMs.

PSRs and POMDPs

Let $\mathcal{T} = \Sigma^*$ be the set of all possible sequence of future observations/actions, and $\mathcal{H} = \Sigma^*$ the set of all possible past observations/actions.

We denote the bi-infinite dimensional *system-dynamics* matrix $P_{\mathcal{T},\mathcal{H}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{H}|}$. This matrix contains all possible probabilities of all sequences in the vocabulary Σ^*, such that $[P_{\mathcal{T},\mathcal{H}}]_{\boldsymbol{t},\boldsymbol{h}} = f(\boldsymbol{th})$. If we restrict to a finite representation of tests and histories such that $\mathcal{T}, \mathcal{H} \subset \Sigma^*$, and $\text{rank}(P_{\mathcal{T},\mathcal{H}}) = \dim(O_n) = \text{rank}(\mathcal{F})$, then the set of tests and histories are called *core*. In particular when the number of tests and histories in the sets $|\mathcal{H}| = |\mathcal{T}| = \dim(O_n)$, then these sets are *minimal core* sets. Given a set of core tests, we may define a finite dimensional sequential model, also known as PSRs (Singh et al., 2004b; Littman et al., 2001).

$P_{\mathcal{T},\mathcal{H}}$ system-dynamics matrix

This matrix uniquely characterizes the stochastic process, since it contains all possible sequence probabilities. The rank of system-dynamics matrix corresponds to the dimension of the process.

core tests and histories

PSRs define a sequential model $\langle \mathbb{R}^K, \alpha_\infty, \{A_{ao}\}_{o\in\Sigma_O, a\in\Sigma_I}, \alpha_0 \rangle$, such that for any history $\boldsymbol{h} \in \Sigma^*$:

$$f(\boldsymbol{t} \mid \boldsymbol{h}) = \frac{\alpha_\infty^\top A_{\boldsymbol{t}} A_{\boldsymbol{h}} \alpha_0}{\alpha_\infty^\top A_{\boldsymbol{h}} \alpha_0} = m_{\boldsymbol{t}} \alpha_{\boldsymbol{h}}, \tag{2.45}$$

$$m_{\boldsymbol{t}} = \alpha_\infty^\top A_{\boldsymbol{t}} \tag{2.46}$$

$$\alpha_{\boldsymbol{h}} = A_{\boldsymbol{h}} \alpha_0 / \alpha_\infty^\top A_{\boldsymbol{h}} \alpha_0 \tag{2.47}$$

where $m_{\boldsymbol{t}}$ defines a *projection function*, and $\alpha_{\boldsymbol{h}}$ defines a *predictive state*. Originally, PSRs were first derived in the interpretable setting w.r.t. a fixed set of *core tests* $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_K\}$, $q_i \in \Sigma^*$ (Littman et al., 2001). In this case, the state of the PSR are predictions of normalized core tests:

$$f(\boldsymbol{t} \mid \boldsymbol{h}) = m_{\boldsymbol{t}}(\alpha_{\boldsymbol{h}}), \tag{2.48}$$

$$\alpha_{\boldsymbol{h}} = [f(\boldsymbol{q_1} \mid \boldsymbol{h}), f(\boldsymbol{q}_1 \mid \boldsymbol{h}), f(\boldsymbol{q}_2 \mid \boldsymbol{h}), \ldots, f(\boldsymbol{q}_K \mid \boldsymbol{h})]^\top \quad \forall_{\boldsymbol{h}\in\Sigma^*} \tag{2.49}$$

The state $\alpha_{\boldsymbol{h}}$ is the state of the equivalent normalized sequential model such that it represents state in an interpretable basis w.r.t. core sets. $m_{\boldsymbol{t}}$ defines *projection functions* that linearly project any arbitrary test into a linear combination of the core tests, *i.e.*, they describe projection functions onto the interpretable basis of core sets, which is equivalent to setting:

$$A_{ao} = [m_{o\boldsymbol{q}_1}^\top, \ldots, m_{o\boldsymbol{q}_K}^\top]^\top \quad \forall_{o\in\Sigma_O}, \tag{2.50}$$

$$\alpha_\infty = \sum_{o\in\Sigma_O} m_{ao} \quad \forall_{a\in\Sigma_A}, \tag{2.51}$$

$$m_{\boldsymbol{t}} = \alpha_\infty^\top A_{\boldsymbol{t}} \tag{2.52}$$

Finding sets of minimal core sets is in many cases a difficult task, also known as the *discovery problem*. Work of Rosencrantz et al. (2004a) provides an alternative form of obtaining a low dimensional representation of the predictive state and projection functions. This process gives rise to the so called *transformed PSRs*, and it involves computing a spectral decomposition of moments of a large number of tests

discovery problem

and histories, such that an equivalent transformation is obtained in Eq. 2.31.

Transformed PSRs

2.5.5 *Hilbert Space Predictive State Representations*

Additional work on PSRs extended sequential models to kernel domains with continuous observations, using Hilbert space embeddings to define distributions over observations and latent states (Smola et al., 2007a). These methods were initially proposed to perform inference in an HMM by updating points in a Hilbert space, using covariance operators (Fukumizu et al., 2013a) and conditional operators (Song et al., 2009). Next, we describe the main concepts behind this approach.

We assume that future prediction functions are elements in a Hilbert space of functions $\mathcal{T}$, endowed with an inner product. We consider $\mathcal{H}$ to be the analogous space of past prediction functions. Now let us define a feature mappings of past $\phi : \mathcal{H} \to \mathbb{R}^H$ as a function that represents histories $\boldsymbol{h}$ as H-dimensional vectors. The same for features of future or tests $\boldsymbol{t}$ in F dimensions: $\psi : \mathcal{T} \to \mathbb{R}^F$, and features of skipped-future observations $\xi : \mathcal{T} \to \mathbb{R}^F$, where we skip the current observation, in Figure 2.10.

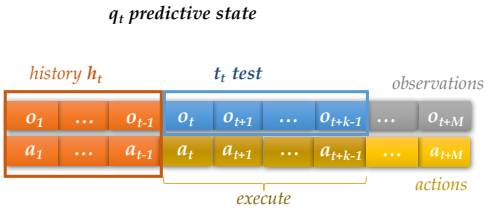


Figure 2.10: Test and histories defining a PSR.

Feature histories form a basis of an Hilbert space, where we assume $\mathcal{R}_H = \text{span}\{\phi(f_{\boldsymbol{h}_i})\}_{i=1}^N$, as in Eq. 2.24. We consider $\mathcal{R}_H$ to denote the space of past, $\mathcal{R}_F$ future and $\mathcal{R}_{SF}$ skipped-future functions respectively.

The inner product in this space is given in terms of a kernel of the form $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{R}_H}$. We additionally assume certain continuity constraints: a point in $\mathcal{R}_H$ corresponds to a function f, and evaluations can be translated into inner products $f(x) = \langle f(\cdot), \phi(x) \rangle_{\mathcal{R}_H}$ with the functions ϕ that span the space.[8] This special property makes this Hilbert Space into a *Reproducing Kernel Hilbert Space* (RKHS) (Aronszajn, 1950; Shawe-Taylor et al., 2004).. Then $\mathcal{R}_H$ defines the RKHS of past, $\mathcal{R}_F$ the RKHS of future and $\mathcal{R}_{SF}$ the RKHS of skipped-future functions. In this setting, probability distributions of histories $p(\boldsymbol{h}) \forall \boldsymbol{h} \in \mathcal{H}$ can be embedded in this space as points describing expected feature values, also called *mean maps*.

reproducing property

[8] The functions that generate this space are also commonly defined as $k_{\mathcal{R}_H}(x, \cdot)$.

Reproducing Kernel Hilbert Spaces

$$\boldsymbol{\mu}_{\mathcal{H}}(x) = \langle p(\cdot), \phi(x \cdot) \rangle_{\mathcal{R}_H} = \mathbb{E}_{\boldsymbol{h} \sim p}[\phi(x)] \in \mathbb{R}^H \tag{2.53}$$

$\mu_{\mathcal{H}}$ mean maps

By the reproducing property for any function in $\mathcal{R}_H$, we may write its expectation in terms of an inner-product with the mean map function.In turn, the inner product of any function with the mean map yields the expected value of its function evaluation:

$$\mathbb{E}_{\boldsymbol{h} \sim p}[f(\boldsymbol{h})] = \langle f(\cdot), \boldsymbol{\mu}_{\mathcal{T}}(\cdot \boldsymbol{h}) \rangle_{\mathcal{R}_H} \in \mathbb{R}^H. \tag{2.54}$$

The same is true for joint distributions, where the analogous mapping is done via covariance operators (Fukumizu et al., 2013b). Let

$\mathcal{R}_H \otimes \mathcal{R}_F$[9] denote the RKHS whose feature map is defined through the tensor product $\varphi(\boldsymbol{t}, \boldsymbol{h}) = \psi(\boldsymbol{t}) \otimes \phi(\boldsymbol{h})$. This space is an RKHS in itself, where points in the space embed joint probability distributions via *covariance operators* $\mathcal{C}_{\mathcal{T},\mathcal{H}} : \mathcal{R}_F \times \mathcal{R}_H \to \mathbb{R}^{F \times H}$.

[9] $\boldsymbol{x} \otimes \boldsymbol{y}$ denotes the outer product, for vectors becomes $\boldsymbol{x}\boldsymbol{y}^\top$

$\mathcal{C}_{\mathcal{T},\mathcal{H}}$ covariance operators

$$\mathcal{C}_{\mathcal{T},\mathcal{H}} = \langle p_{\mathcal{T},\mathcal{H}}(\cdot), \varphi \rangle_{\mathcal{R}_F \times \mathcal{R}_H} = \mathbb{E}_{\mathcal{T},\mathcal{H}}[\psi(\mathcal{T}) \otimes \phi(\mathcal{H})] \tag{2.55}$$

In the case of delta kernels under discrete events, also known as the "tabular setting", $\mathcal{C}_{\mathcal{H},\mathcal{T}}$ coincides with the joint probability table $p(\mathcal{H}, \mathcal{T})$.

Conceptually conditional distributions $p(\mathcal{T} \mid \boldsymbol{h})$ can be embedded in an RKHS in a similar form, using a mean map function $\mu_{\mathcal{T}|\boldsymbol{h}} = \mathbb{E}_{\mathcal{T}|\boldsymbol{h}}[\psi(\cdot) \mid \boldsymbol{h}]$. This defines a family of conditional embeddings, considering all possible values of $\boldsymbol{h} \in \mathcal{H}$. In the tabular setting, *i.e.*, for delta kernels under discrete events, $\boldsymbol{\mu}_{\mathcal{T}|\boldsymbol{h}}$ coincides with the conditional probability table $p(\mathcal{T} \mid \mathcal{H} = \boldsymbol{h})$. Consider now a mapping between different RKHSs, defined in terms of a *conditional operator* $\mathcal{C}_{\mathcal{T}|\mathcal{H}} : \mathcal{R}_F \to \mathcal{R}_H$, such that

$\mathcal{C}_{\mathcal{T}|\mathcal{H}}$ conditional operators

Kernel Bayes' rule

$$\mathcal{C}_{\mathcal{T}|\mathcal{H}} = \mathbb{E}[\psi(\mathcal{T}) \mid \mathcal{H} = \cdot] \in \mathcal{R}_H \tag{2.56}$$

$$\mu_{\mathcal{T}|\boldsymbol{h}} = \mathcal{C}_{\mathcal{T}|\mathcal{H}}\phi(\boldsymbol{h}) = \mathcal{C}_{\mathcal{T},\mathcal{H}}\mathcal{C}^{-1}_{\mathcal{H},\mathcal{H}}\phi(\boldsymbol{h}) \tag{2.57}$$

This establishes the connection between the two RKHSs.[10] Intuitively, conditional operators map probability distributions embeddings into other probability distribution embeddings. Using this notion, we may establish the relation between mean maps:

[10] This estimator is also commonly used in the regularized Tikhonov variant: $\mu_{\mathcal{T}|\boldsymbol{h}} = \mathcal{C}_{\mathcal{T},\mathcal{H}} \left(\mathcal{C}^2_{\mathcal{H},\mathcal{H}} + \lambda_d \mathbb{I}\right)^{-1} \mathcal{C}_{\mathcal{H},\mathcal{H}}\phi(\boldsymbol{h})$

$$\mu_{\mathcal{T}} = \mathcal{C}_{\mathcal{T}|\mathcal{H}}\mu_{\mathcal{H}} \text{ and} \tag{2.58}$$

$$\mathcal{C}_{\mathcal{T},\mathcal{H}} = \mathcal{C}_{\mathcal{T}|\mathcal{H}}\mathcal{C}_{\mathcal{H},\mathcal{H}} \tag{2.59}$$

In fact, this is what observable operators do in the original spaces of distributions. Mean maps, on the other hand, are equivalent to distributions, in particular future prediction functions. Hence we can define PSRs embedded in RKHSs using solely conditional operators and mean maps.

Consider a PSR with predictive states in $\mathcal{Z}$. Let $\mathcal{R}_Z$ be the RKHS of predictive states with kernel $l(z, z') = \langle \zeta(z), \zeta(z') \rangle_{\mathcal{R}_Z} \forall_{z \in \mathcal{Z}}$, and $\mu_{\mathcal{T}|\boldsymbol{h}}$ be the mean map for the probability density of future predictions $p(\mathcal{T} \mid \boldsymbol{h})$. The observable operator $A_o : \mathcal{R}_Z \times \mathcal{R}_F \to \mathcal{R}_Z \times \mathcal{R}_F$ transforms future prediction functions into other prediction functions:[11]

[11] The mean map for future predictions with RKHS for state space $\mathcal{Z} is$:
$\mu_{\mathcal{T}_n|\boldsymbol{h}} = \mathbb{E}_{\mathcal{T}_n|\boldsymbol{h}}[\psi(\mathcal{T}_n)]$
$= \mathbb{E}_{\mathcal{Z}_n|\boldsymbol{h}}[\mathbb{E}_{\mathcal{T}_n|\mathcal{Z}_n}[\psi(\mathcal{T}_n)]] = \mathbb{E}_{\mathcal{Z}_n|\boldsymbol{h}}[\mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}\zeta(\mathcal{Z}_n)]$
$= \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}\mathbb{E}_{\mathcal{Z}_n|\boldsymbol{h}_n}[\zeta(\mathcal{Z}_n)] = \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}\mu_{\mathcal{Z}_n|\boldsymbol{h}_n}$
$= \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}A_{o_{n-1}}\mu_{\mathcal{Z}_{n-1}|\boldsymbol{h}_{n-1}} = \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}A_{o_{n-1:1}}\mu_{\mathcal{H}_0}$
$= \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}A_{\boldsymbol{h}_n}\mu_{\mathcal{H}_0}$

$$\mu_{\mathcal{Z}_{n+1}|\boldsymbol{h}_{n+1}} = A_{o_n}\mu_{\mathcal{Z}_n|\boldsymbol{h}_n}. \tag{2.60}$$

We define a PSRs in RKHSs following Eqs.(2.45-2.47):

$$\mu_{\mathcal{T}_n|\boldsymbol{h}_n} = \mathcal{C}_{\mathcal{T}_n|\mathcal{Z}_n}\mu_{\mathcal{Z}_n|\boldsymbol{h}_n} = A_{\boldsymbol{t}}A_{\boldsymbol{h}}\alpha_0 \tag{2.61}$$

$$\alpha_0 = \mathcal{C}_{\mathcal{Z}_0|\mathcal{H}_0}\mu_{\mathcal{H}_0} \qquad \text{with } \alpha_0 : \mathcal{R}_Z \to \mathbb{R}^H \tag{2.62}$$

$$\alpha_n = \mu_{\mathcal{Z}_n|\boldsymbol{h}_n} = A_o\mu_{\mathcal{Z}_{n-1}|\boldsymbol{h}_{n-1}} = A_{\boldsymbol{h}}\alpha_0 \qquad \text{with } \alpha_n : \mathcal{R}_Z \to \mathcal{R}_Z \tag{2.63}$$

$$\alpha_\infty = \langle f A_{\boldsymbol{t}}, \alpha_n \rangle_{\mathcal{R}_Z} \;\; \forall f \in \mathcal{R}_F \qquad \text{with } \alpha_\infty : \mathcal{R}_Z \to \mathbb{R}^F \tag{2.64}$$

Evaluations in the RKHS may be done by computing expectations of future predictions, via mean map inner products in the RKHS. For any $f \in \mathcal{R}_F$ [12]

$$\mathbb{E}[f(\boldsymbol{t}_n \mid \boldsymbol{h}_n)] = \langle f, \mu_{\mathcal{T}_n|\boldsymbol{h}_n} \rangle_{\mathcal{R}_F} = \langle f A_{\boldsymbol{t}}, \alpha_n \rangle_{\mathcal{R}_Z} \in \mathbb{R}^F \tag{2.65}$$

mean map embedding

[12] Instead of the standard form of evaluation of sequential models $f(\boldsymbol{t} \mid \boldsymbol{h}) = \alpha_\infty^\top A_{\boldsymbol{t}} A_{\boldsymbol{h}} \alpha_0$.

2.5.6 *Spectral learning of PSRs*

Moment-based approaches rely in their core on a spectral decomposition of observed statistics, and are commonly denoted as *spectral learning*. Spectral learning has been used to estimate different latent variable models, such as HMMs, (Hsu et al., 2012a) GMMs, (Anandkumar et al., 2012a) PCFGs, (Cohen et al., 2012) additive trees, (Parikh et al., 2014) and finite state transducers (Bailly et al., 2013a). Singular Value Decomposition (SVD) or Canonical Correlation Analysis (CCA), Reduced Rank Regression (RRR) are common spectral decompositions for matrices. They are used to retrieve a latent state representation from correlations of past and future observations. Higher order statistics use tensor decomposition techniques to achieve the same purpose (§ 2.4.3). The first learning method used Predictive State Representations (PSRs) proposed by Littman et al. (2001). Consider the first order moment $W_1 = \mathbb{E}[\phi(\boldsymbol{h})] \in \mathbb{R}^{|\mathcal{H}|}$ of histories $\forall_{\boldsymbol{h} \in \mathcal{H}}$ and some feature mapping for histories $\phi : \mathcal{H} \to \mathbb{R}^H$. The second order moment matrix can be defined for conditional expectation of features of tests $\psi : \mathcal{T} \to \mathbb{R}^F$ conditioned on histories $\boldsymbol{h} \in \mathcal{H}$ as $W_2 = \mathbb{E}[\psi(\boldsymbol{t}) \mid \phi(\boldsymbol{h})] \in \mathbb{R}^{F \times H}$, for a large non minimal number of tests $|\mathcal{T}|$ and histories $|\mathcal{H}|$.[13] Let the third order moment of tests, current observation/action pairs and histories $W_3 = \mathbb{E}[\psi(\boldsymbol{t}) \otimes \varphi(o) \mid \phi(\boldsymbol{h}); \varphi(a)]$ be written as matrix slices $\mathbb{R}^{F \times H}$ for each possible observation/action pairs $\forall_{o \in \Sigma, a \in A}$. Let $\cdot$ denote the indices of all elements and $*$ refer to the index of the empty string/history. Hsu et al. (2009) and Boots et al. (2011a) show that there is a conjugate equivalent PSR whose parameters can be obtained from these observable quantities using

[13] This conditional moment refers to a conditional embedding operator defined by Kernel Bayes Rule, see (Fukumizu et al., 2013c; Song et al., 2009) for details.

$$\alpha_0 = U^\top [W_2]_{\cdot,*} \tag{2.66}$$

$$\alpha_\infty^\top = W_1^\top \left(U^\top W_2 \right)^\dagger \tag{2.67}$$

$$A_{a,o} = U^\top [W_3]_{\cdot,ao,\cdot} \left(U^\top W_2 \right)^\dagger \tag{2.68}$$

where U is the projection matrix defining the learned subspace, typically obtained from a spectral decomposition of the second moment matrix $W_2 = U \Sigma V^\top$.[14] Here $M^\dagger$ denotes the Moore-Penrose pseudo-inverse of M. The solution in Eq. 2.66 can be obtained from solving a least squares optimization of the form:

[14] For core identification methods U is the identity matrix and the moments specify probabilities over core tests instead.

$$\arg\min_{A_{a,o}} \| U^\top [W_3]_{\cdot,ao,\cdot} - U^\top W_2 A_{a,o} \|_F^2 \tag{2.69}$$

If the feature mapping corresponds to indicator functions then these moments correspond to conditional probability tables and the PSR parameters are defined in the tabular case. However, Eq. 2.66 provides an extended description that is able to handle continuous observations and actions as embeddings in an RKHS. Conditional moments in this continuous setting are referred to as conditional embedding operators for a detailed description, as shown by Song et al. (2009), Fukumizu et al. (2013c), and Boots et al. (2013b).

The proposed method is a consistent learning method, in the sense that the PSR parameters learned from data will recover the process exactly under an infinite data assumption. This method has been superseded by better methods such as the ones we will describe later. Work of Singh et al. (2004b) extended the learning process to *transformed PSRs*. They presented a method to learn lower dimensional PSRs. Whenever the set of core tests is not minimal, the dimension of the PSR is higher than the linear dimension of the system. This work allowed dealing with larger core sets of tests and histories, by finding a transformed PSR of lower dimension, that still generates the process. Later, Siddiqi et al. (2010b) proposed learning a reduced rank HMMs (RR-HMMs). Some HMMs require a large number of hidden states to smoothly model the evolution of the process. RR-HMMs provide a way to model the same process by projecting into a lower dimensional state space, which makes the learning process easier. For a particular choice of core test we will have a different, but equivalent PSR. However, finding sets of core tests can be a difficult task, the so called *discovery problem* (Singh et al., 2003). Kulesza et al. (2015) provide a form of selecting these sets, such that the learning algorithm recovers a better conditioned PSR.

transformed PSRs (TPSRs)

Reduced-rank HMMs (RR-HMMs)

discovery problem

Spectral learning of Hidden Markov Models

The most common algorithms for learning HMMs resort to search heuristics to maximize the likelihood of the observed data, such as EM (Baum et al., 1970; Dempster et al., 1977) in Section 2.4.1. These methods, however, suffer from local optima and require several inference passes over the data, which makes them slower than spectral methods.

The first work for learning PSRs for Hidden Markov Models (HMMs) was proposed by Hsu et al. (2009). They derived a learning algorithm that learns PSRs from a spectral decomposition. They provided PAC-bound guarantees for learning the observable operators from data. The idea comes from *subspace identification* methods in control theory, that use spectral approaches to discover the relationship between hidden states and the observations (Ljung, 1999; Van Overschee et al., 1996).

subspace identification

The algorithm involves projecting observations into a lower dimensional space that correlates past and future observations, using a Singular Value Decomposition (SVD) $\text{SVD}(P_{\mathcal{T},\mathcal{H}}) = U\Lambda V^\top$, and keeping the first k left singular values $U \equiv U_k \in \mathbb{R}^{|\mathcal{T}|\times k}$.

$$\begin{aligned} \boldsymbol{b}_0 &= U^\top P_{\mathcal{T},\epsilon} \quad \in \mathbb{R}^k \\ \boldsymbol{b}_\infty^\top &= P_{\mathcal{H},\epsilon}^\top (U^\top P_{\mathcal{T},\mathcal{H}}^\dagger) \quad \in \mathbb{R}^k \\ B_o &= (U^\top P_{\mathcal{T},o\mathcal{H}})(U^\top P_{\mathcal{T},\mathcal{H}}^\dagger) \quad \in \mathbb{R}^{k\times k} \end{aligned} \tag{2.70}$$

The matrix U needs only to be in the same range as $\text{range}(U) = \text{range}(P_{\mathcal{T},\mathcal{H}}) = \text{range}(O)$, such that $(U^\top O)$ is invertible. All the quantities can be computed from data and they relate to the parameters of the HMM through a similarity transform $S = (U^\top O) \in \mathbb{R}^{k\times k}$. The equations above define an equivalent sequential model with observable operators $A_o = T\text{diag}O_x$, where $\text{diag}(O_x) \in \mathbb{R}^{k\times k}$ denotes the diagonal matrix containing the *x-th* row of the emission matrix, *i.e.* all emission probabilities for observation x. The results in Eq. 2.70 correspond to the solution of an equivalent regression problem (Eq. 2.69), using a lower projection onto the column space of U:

$$\arg\min_{B_o} \|U^\top P_{\mathcal{T},o\mathcal{H}} - U^\top P_{\mathcal{T},\mathcal{H}} B_o\|_F^2 \tag{2.71}$$

This method provided PAC-bound guarantees, under certain spectral separation assumptions of the model. It is also more sample efficient than previous work (Mossel et al., 2006), since it does not attempt to explicitly retrieve the emissions and transitions. Regardless, Hsu et al. (2009) provide a possible way to retrieve the parameters T, O. The process consists of first estimating the emission matrix, then the transition making use of pseudo-inverses. This recovery process is, however unstable in its core, since it makes extensive use of pseudo-inverses and inverses. This makes the algorithm very sensitive to noise, which, in the case of insufficient statistics or inexact moments, can be a fatal problem. Other spectral techniques have been employed in order to find a lower dimensional state space (using U). Canonical Correlation Analysis (CCA) has been used to find directions of maximum correlation, between two views (past and future) (Cohen et al., 2013b).

Song et al. (2009) presents a kernel version for learning observable operators that do not require the explicit computation of features, which could have a potentially infinite dimensional basis.[15] Instead they consider the dual problem, and derive equations in terms of *Gram matrices*.

[15] Such as RBFs or exponential kernels

Gram matrix

Spectral learning of HSE-PSRs

Previously we have defined linear operators and learning algorithms in terms of discrete action/observation pairs. This formulation can be-

come infeasible for large scale systems where it is difficult to enumerate all possible observations and actions, such as in robotics domains where observations and actions are continuous. Boots et al. (2013b) extended spectral learning of embedded distributions in RKHSs to predictive representations of controlled models, also known as Hilbert space embedding PSRs (HSE-PSRs). Let $\mathcal{T}^O$ denote the set of tests of observations, $\mathcal{T}^A$ the set of test actions, $\mathcal{O}$ the set of observations, and $\mathcal{A}$ the set of actions. We further define feature mappings shown in Figure 2.11, for immediate, future and extended future actions and observations:

- $\phi_t^o(\mathbf{o}_t) : \mathcal{O} \to \mathbb{R}^O$ of immediate observations,
- $\phi_t^a(\mathbf{a}_t) : \mathcal{A} \to \mathbb{R}^A$ of immediate actions,
- $\boldsymbol{\psi}_t^o(\mathbf{o}_{t:t+k-1}) : \mathcal{O}^k \to \mathbb{R}^{F^O}$ of future observations,
- $\psi_t^a(\mathbf{a}_{t:t+k-1}) : \mathcal{A}^k \to \mathbb{R}^{F^A}$ of future actions,
- $\xi_t^o(\mathbf{o}_{t:t+k}) \equiv [\boldsymbol{\psi}_{t+1}^o \otimes \boldsymbol{\phi}_t^o, \boldsymbol{\phi}_t^o \otimes \boldsymbol{\phi}_t^o] : \mathcal{O}^{k+1} \to \mathbb{R}^{(F^O+O)\times 2O}$ of extended observations,
- $\xi_t^a(\mathbf{a}_{t:t+k}) \equiv \boldsymbol{\psi}_{t+1}^a \otimes \boldsymbol{\phi}_t^a : \mathcal{A}^{k+1} \to \mathbb{R}^{F^A \times A}$ of extended actions.

HSE-PSRs the predictive state $\boldsymbol{q}_t$ is updated using covariance operators

$$\begin{aligned} q_{t+1} &= \mathcal{C}_{\mathcal{T}^O|\mathcal{T}^A \boldsymbol{h}_t, a_t, o_t} \\ &= \mathcal{C}_{\mathcal{T}^O, O|\mathcal{T}^A \boldsymbol{h}_t, a_t} \otimes_O \mathcal{C}^{-1}_{\mathcal{O}, \mathcal{O}|\boldsymbol{h}_t, a_t} \phi(o_t) \\ &= W^{\xi}_{\text{ext}} \boldsymbol{q}_t \otimes_A \phi^A(a_t) \otimes_O \left(W^o_{\text{ext}} \boldsymbol{q}_t \otimes_A \phi^A(a_t) \right)^{-1} \phi^O(o_t) \end{aligned} \tag{2.72}$$

where W^{ξ}_{ext} is a linear transformation from futures to skipped futures and W_{ext^o} a linear transformation from futures to the current observation, in Figure 2.11.

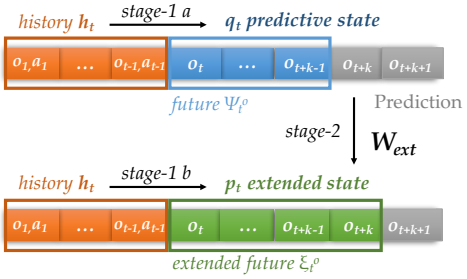


Figure 2.11: Illustration of the PSR update step.

Hilbert Space Embeddings PSRs (HSE-PSRs)

Two-stage Regression

In this section, we revisit a continuous representation for filtering and prediction equations based on a two-stage regression approach proposed by Hefny et al. (2017b). This method interprets learning of PSRS as an instrumental variable regression algorithm, and in particular learning of HSE-PSRs. They assume future expectations $\bar{\mathbf{p}}_t = \mathbb{E}[\boldsymbol{\xi}_t^o(\mathbf{o}_{t:t+k}); \psi_t^a(\mathbf{a}_{t:t+k})]$ are a linear transformation of extended future expectations
$\bar{\mathbf{q}}_t = \mathbb{E}[\boldsymbol{\psi}_t^o(\mathbf{o}_{t:t+k-1}); \psi_t^a(\mathbf{a}_{t:t+k-1})]$, in Figure 2.11.

Due to temporal correlation between these two expectations $\bar{\mathbf{p}}_t$ and $\bar{\mathbf{q}}_t$, we cannot directly learn the mapping W_{ext} over empirical estimates of $\bar{\mathbf{p}}_t$ and $\bar{\mathbf{q}}_t$, using linear regression, since their noise is also correlated. Alternatively, they turn to an instrumental variable regression where history is used as instrument, since it is correlated with the observables

$\boldsymbol{\psi}_t$ and $\boldsymbol{\xi}_t$, but not with the noise. In this case, we can go from predictive to extended states by first computing a possibly non-linear regression from histories, to both predictive (Eq. 2.73) and extended statistics (Eq. 2.74) (stage-1 regression a and b). When we consider features of a Hilbert space Embedding PSR (HSE-PSR) this non-linear regression can be computed by Kernel Bayes Rule (KBR) (Fukumizu et al., 2013b), but could be any arbitrary non-linear transformation (regression trees, logistic regression,$\cdots$). Predictive states can be regarded as evaluations of conditional mean maps $m_{\mathcal{T}|\mathcal{H}}$ where $\mathbf{p}_t = \langle \boldsymbol{\xi}_{t+1}, m_{\mathcal{T}|\mathcal{H}} \rangle_{\mathcal{R}_F}$ in the RKHS of future sequences $\mathcal{R}_F$, and the analogous for future actions and extended states.

$$\mathbf{p}_t \equiv \mathbb{E}\left[\boldsymbol{\xi}^o_{t+1} \mid \mathbf{h}^\infty_t; \boldsymbol{\xi}^a_{t+1}\right] \quad \text{stage-1}a \tag{2.73}$$

$$\mathbf{q}_t \equiv \mathbb{E}\left[\boldsymbol{\psi}^o_t \mid \mathbf{h}^\infty_t; \boldsymbol{\psi}^a_t\right] \quad \text{stage-1}b \tag{2.74}$$

Subsequently, this approach linearly regresses the denoised extended state $\mathbf{p}_t$ from the denoised predictive state $\mathbf{q}_t$, using a least squares approach (stage-2 regression), in Eq. 2.75. For ease of explanation, let us further partition extended states in two parts $\mathbf{p}_t \equiv [\mathbf{p}^\xi_t, \mathbf{p}^o_t]$ and $W_{\text{ext}} \equiv [W^\xi_{\text{ext}}, W^o_{\text{ext}}]$, derived from the skipped future $\boldsymbol{\psi}^o_{t+1}$ and present $\boldsymbol{\phi}^o_t$ observations, see Figure 2.11.

prediction

$$\begin{aligned}
\mathbf{p}^\xi_t &\equiv \mathbb{E}\left[\boldsymbol{\psi}^o_{t+1} \otimes \boldsymbol{\phi}^o_t \mid \mathbf{h}^\infty_t ; \boldsymbol{\psi}^a_{t+1} \otimes \boldsymbol{\phi}^a_t\right] \\
\mathbf{p}^o_t &\equiv \mathbb{E}\left[\boldsymbol{\phi}^o_t \otimes \boldsymbol{\phi}^o_t \mid \mathbf{h}^\infty_t ; \boldsymbol{\phi}^a_t\right] \\
\mathbf{p}_t &= [\mathbf{p}^\xi_t, \mathbf{p}^o_t]^\top = [W^\xi_{ext}, W^o_{\text{ext}}]^\top \mathbf{q}_t \quad \text{stage-2}
\end{aligned} \tag{2.75}$$

For RFFPSRs stage-1 regression can be derived from either a joint regression over action/observation pairs, using KBR or by solving a regularized least squares problem, for a full derivation refer to Hefny et al. (2017b).

The second step in Eq. 2.76 is provided by a *filtering function* f_t, that tracks predictive states over time:

filtering

$$\mathbf{q}_{t+1} = f_{\text{cond}}(W_{\text{ext}}\mathbf{q}_t, \mathbf{o}_t, \mathbf{a}_t) \equiv f_t(\mathbf{q}_t, \mathbf{o}_t, \mathbf{a}_t) \tag{2.76}$$

Namely, for HSE-PSRs filtering corresponds to conditioning on the current observation via KBR, in Eq. 2.77. In contrast with predictive state inference machines where they learn directly a filtering function by minimizing prediction loss (Sun et al., 2016b), here we assume a generative approach and learn the filtering function parameters subject to the generative HSE-PSR framework.

We consider $\mathbf{p}^\xi_t$ as a 4-mode tensor $(\boldsymbol{\psi}^o_{t+1}, \boldsymbol{\phi}^o_t, \boldsymbol{\psi}^a_{t+1}, \boldsymbol{\phi}^a_t)$, where $\times_{\phi^o}$ defines the multiplication along the ϕ^o-mode.

predictive filter in HSE-PSRs

$$
\begin{aligned}
\mathbf{q}_{t+1} &= \mathbb{E}[\boldsymbol{\psi}^o_{t+1} \mid \mathbf{h}^\infty_{t+1}; \boldsymbol{\psi}^a_{t+1}] \\
&= \mathbb{E}[\boldsymbol{\psi}^o_{t+1} \mid \boldsymbol{\phi}^o_t, \mathbf{h}^\infty_t; \boldsymbol{\psi}^a_{t+1}, \boldsymbol{\phi}^a_t] \\
&= \mathbb{E}[\boldsymbol{\psi}^o_{t+1} \otimes \boldsymbol{\phi}^o_t \mid \mathbf{h}^\infty_t; \boldsymbol{\psi}^a_{t+1} \otimes \boldsymbol{\phi}^a_t] \times_{\boldsymbol{\phi}^o} \boldsymbol{\phi}^o_t \times_{\boldsymbol{\phi}^a} \boldsymbol{\phi}^a_t \\
&\qquad \left[\mathbb{E}[\boldsymbol{\phi}^o_t \otimes \boldsymbol{\phi}^o_t | \boldsymbol{\phi}^a_t; \mathbf{h}^\infty_t] \boldsymbol{\phi}^{a\top}_t + \lambda I\right]^{-1} \\
&= \mathbf{p}^\xi_t \times_{\boldsymbol{\phi}^o} \boldsymbol{\phi}^o_t \times_{\boldsymbol{\phi}^a} \boldsymbol{\phi}^a_t \left[\mathbf{p}^o_t \boldsymbol{\phi}^{a\top}_t + \lambda I\right]^{-1} \\
&= W^\xi_{\text{ext}} \mathbf{q}_t \times_{\boldsymbol{\phi}^o} \boldsymbol{\phi}^o_t \times_{\boldsymbol{\phi}^a} \boldsymbol{\phi}^a_t \left[W^o_{\text{ext}} \mathbf{q}_t \boldsymbol{\phi}^{a\top}_t + \lambda I\right]^{-1} \\
&\equiv f_t(\mathbf{q}_t, \mathbf{a}_t, \mathbf{o}_t)
\end{aligned}
\tag{2.77}
$$

This equation explicitly defines the full predictive state update equation or filtering in Eq. 2.76, when considering HSE-PSRs as a representation of $\mathbf{p}_t$ and $\mathbf{q}_t$.

2.6 *Planning approaches*

Planning is a general term with different connotations according to different research communities. Common to all remains the central idea that planning consists on devising a plan for an agent to execute in order to complete a certain task. The planning taxonomy can be described in terms of a *plan*, the strategy or behaviour that could be a sequence of actions from a *start* state the needs to be executed to achieve a given *goal*; a *state*, which describes the current position of the robot within its environment; *time* that translates how far we are in the execution of the plan; and feasibility and optimality of a given plan. Feasibility pertains to the existence of a solution while optimality consists on improving the quality of the plan.

In machine learning, planning is most commonly addressed based on decision theoretic methods to model uncertainties, adversarial scenarios and optimization. Once learning is achieved, what decisions should be made to achieve a desired goal. Most commonly, this question is posed in the discrete setting where we assume the agent may choose from a discrete set of actions and acts upon a discrete state space. Within Natural Language Processing, we may regard planning in this setting, where the agent (generally a state machine), may perform a given discrete set of decisions or actions defining a plan, *i.e.,* the sequence of actions that lead to the desired goal. These have been applied in a vast range of natural language tasks such as parsing (Luque et al., 2012; Goldberg et al., 2012; Goodman et al., 2016), part-of-speech-tagging (Chang et al., 2015), handwriting recognition (Ross et al., 2010), information extraction (Branavan et al., 2011; Narasimhan et al., 2016), machine translation (Grissom II et al., 2014; He et al., 2016), summarization (Paulus et al., 2017) and question & answering (Choi et al., 2017).

Within the field of Robotics planning concerns with finding algorithms that map high-level specifications of tasks from humans into low-level motion controls/actions of a robot. Two concepts are mostly connected with robotic planning: motion planning and trajectory optimization.

The former describes rotations and translations required to find a solution for moving a body within a constrained space, usually ignoring body dynamics and other differential constraints. This problem is also known as the *Piano Movers' Problem*, where knowing the map of a building we are required to place a piano from a start position into a goal location without hitting any objects within the building.

The second problem, trajectory optimization, usually refers to finding the motion of the robot according to a solution provided by a motion planner and considering the mechanical limitations of the robot.

Trajectory optimization is considered complementary to motion planning in that it usually requires an initial guess, in many cases provided by a motion planning algorithm. In motion planning, the state space defines the set of possible transformations that could be applied to the robot, its *configuration space* (Lozano-Perez, 1981; Lozano-Pérez et al., 1979; Latombe, 1991). $\mathcal{C}$. We define the *work space* $\mathcal{W}$ the space where the robots and the obstacles live. Motion planning algorithms can be decomposed into several classes: geometric approaches, combinatorial methods, sampling-based approaches and optimization methods.

Motion planning

configuration space $\mathcal{C}$

Geometric problems require understanding and manipulating complex geometric representations that transform a single body or chain of bodies. This requires defining geometric primitives which will enable the robot to move in space. Most approaches require a description of configuration space obstacles, which can be prohibitive to create in the context of high-dimensional planning problems.

Combinatorial methods build paths in continuous configuration space by building discrete representations without loss of information. These methods allow complete solutions, meaning that they will find a solution if it exists (Agarwal et al., 1997; BaezaYates et al., 1993; Arya et al., 1994).

Sampling-based approaches have been widely used in complex problems (Aronov et al., 1998; Lavalle et al., 2000; Kavraki et al., 1994; Chiang et al., 2007) and also high dimensional scenarios (Xanthidis et al., 2016). These consist mostly on search algorithms that sample the configuration space in order to find a feasible solution. Planning is defined as a continuous problem, which can be applied to problem when the number of geometric primitives is very large. They usually require a collision detection module to determine feasibility and generally they provide weaker notions of completeness.

Optimization-based planning offers an alternative portion of motion planning algorithms. This family of methods attempts to directly search over the space of trajectories and combines optimization techniques to improve the quality of the final trajectory. The optimization occurs in the space of possible trajectories Ξ, where each trajectory $\xi : [0,1] \rightarrow \mathcal{C}$ is considered as a function of time $t \in [0,1]$. In consequence calculus of variations will be a crucial tool to characterize extrema of the problem. The quality of each trajectory is measured as a functional cost $\mathcal{U} : \Xi \rightarrow \mathbb{R}$ that we wish to minimize. In order to obtain trajectories that are collision free and smooth the objective functional needs to account for object proximity in the form of $\mathcal{U}_{obs}$ as well as dynamical quantities respectively, such as velocities and accelerations represented as $\mathcal{U}_{smooth}$.

$$\mathcal{U} = \mathcal{U}_{obs} + \beta \mathcal{U}_{smooth} \tag{2.78}$$

The boundary conditions, such as initial $\xi(0)$ and final configuration $\xi(1)$ can be considered as constraints in the optimization problem. Further dynamical or mechanical boundaries may be specified as an additional set of constraints that must be satisfied during optimization. This process can be regarded as incrementally searching for trajectories that minimize the gradient of the cost functional in Eq. 2.78, while constrained to the tangent space of the constraints. Hence, much of the problems in motion planning under trajectory optimization may be cast as an application of Newton's method or gradient descent. The differential equations describing the motion of the agent typically used in dynamic programming or the minimum principle are difficult to solve analytically (Rimon et al., 1992), therefore, in most cases, numerical techniques are preferred (Quinlan et al., 1993; Brock, 2002).

Gradient-based trajectory optimization

Gradient methods can provide efficient algorithms since they leverage gradient information while avoiding the explicit computation of second-order dynamics. In Calculus of Variations, gradients of the cost functional $\nabla\mathcal{U}$ correspond to perturbations $\phi : [0,1] \to \mathcal{C}$ of the current trajectory ξ that maximize $\nabla\mathcal{U}[\xi + \epsilon\phi]$ as $\phi \to 0$. Consider the linear functional to be of the form $\mathcal{U}[\xi] = \int v(\xi(t), \dot{\xi}(t))dt$, and the consecutive configuration $\bar{\xi} = \xi + \epsilon\phi$. The extrema of the cost functional can be characterized by the Euler-Lagrange equation:

$$\nabla\mathcal{U}[\xi] = \frac{\partial v}{\partial \xi} - \frac{d}{dt}\frac{\partial v}{\partial \dot{\xi}} = 0. \tag{2.79}$$

Therefore, we may optimize trajectories by simply taking iterative steps in the direction of the negative functional gradient defined by

$$\xi_{t+1} = \xi_t - \lambda\nabla\mathcal{U}[\xi]. \tag{2.80}$$

Next, we briefly describe a computationally efficient algorithm that performs covariant[16] updates of the functional cost, also known as CHOMP (Covariant-Hamiltonian Optimization for Motion Planning) (Zucker et al., 2013). CHOMP defines an obstacle cost functional in terms of a pre-computed distance field to obstacles in the workspace environment $c : \mathcal{W} \to \mathbb{R}$, where $\mathcal{B}$ defines the set of body points of the agent/robot and $x : \mathcal{C} \times \mathcal{B} \to \mathcal{W}$ its forward kinematics function, mapping a particular point in the robot $u \in \mathcal{B}$ for a given body configuration $\xi \in \mathcal{C}$ to a workspace location.

[16] independent of reparametrization.

$$\mathcal{U}_{obs}[\xi] = \int_0^1 \int_{\mathcal{B}} c(x(\xi(t), u)) \left\| \frac{d}{dt} x(\xi(t), u) \right\| du dt \tag{2.81}$$

$$\mathcal{U}_{smooth}[\xi] = \frac{1}{2} \int_0^1 \left\| \frac{d}{dt} \xi(t) \right\|^2 dt \tag{2.82}$$

This algorithm performs gradient descent in trajectory space with respect to the natural norm of the functional, which is equivalent to the

inverse of the metric times its Euclidean gradient $\nabla_A \mathcal{U}[\xi] = A^{-1}\nabla\mathcal{U}[\xi]$. This improved update transformation makes the algorithm invariant to reparametrization, *i.e.*, independent of the metric space A.

2.7 *Planning under uncertainty*

An alternative variant of planning algorithms branches out to decision theoretic contexts, where uncertainty plays an important role. This gives rise to a family of algorithms also known as *reinforcement learning* (RL), where good actions provide positive feedback in the form of a reward. This view describes planning as an optimization approach where instead of minimizing a cost functional we maximize a reward functional, also denoted reward-to-go or cost-to-go.

reinforcement learning

In reinforcement learning, an agent tries to maximize the accumulated reward over each trajectory. In an episodic setting, meaning the task is restarted at the end of each trajectory or episode, the objective is to maximize the total reward per episode, while if the task in never ending a discounted factor is introduced to discriminate between the importance of long vs. short term rewards. In decision theoretic planning or reinforcement learning the agent or robot interacts with its environment by performing actions $a \in A$ in a continuous or discrete set. This process is characterized by a state $s \in S$ which can also have a continuous or discrete representation. The state contains all the necessary information to predict future states, for instance the current position and velocity of a swing up pendulum. Actions are used to control or change the state of the system, in the same example actions correspond to torques applied to the pendulum joints. For each interaction the agent receives a scalar valued reward $r \in \mathbb{R}$ and could be a function of state or state and action, in the swing-up task it could be defined as the distance to the top position. The objective of the reinforcement learning agent is thus to find a mapping from state to actions $\pi : S \to A$, denoted *policy*, that maximizes cumulative expected reward $J = \mathbb{E}_\pi[\sum_t \gamma^{t-1} r_t]$. This mapping can either be deterministic yielding the exact same action for every realization of the same state $a = \pi(s)$, or probabilistic if it defines a distribution over actions $a \sim \pi(s, a) = p(a \mid s)$

RL taxonomy

Classical reinforcement learning approaches are based on discrete theoretic decision algorithms such as Markov decision processes (MDPs) defined in terms of a set of rewards R, states S, actions A and state transition probabilities $T : S \to S$ capturing the dynamics of the system $T(s', a, s) = p(s' \mid a, s)$. Transition probabilities account for uncertainty in the system dynamics even if the state is fully observable. MDPs are further characterized by the Markov property that requires that the future state and reward depend only on the current state (Sut-

ton et al., 1998). Meaning that the state of the system is a sufficient statistic for future state predictions.

Value vs. Policy optimization

The value function is a quantity that the reflects the quality of the policy in a particular state $V(s)$ or state/action pair $Q(s,a)$. This quantity can be used as an objective in the optimization process. However value function approaches struggle with many Robotics RL challenges, such as the high dimensionality of the state/action space. Value function algorithms are also very sensitive to approximation errors, leading to drastic changes in policy which could result in damage of the robot. For all of these reasons, value-based RL is often discouraged in Robotics applications. On the other hand, policy search methods, directly optimize over policy parameters $\theta : \pi(\theta)$ avoiding the need to specify a value function, which leads to smoother changes of policy and consequently safer behaviour for robots. This advances RL into high dimensional continuous domains with stability and robustness guarantees in some cases (Agarwal et al., 1999) Policy search methods generally require fewer parameters than valued-based counterparts. Local search within policy parameter landscape already provides good results (Kirk, 2012), while addition of constraints can be easily incorporated, for instance by regularizing the change in path distribution. Overall, policy search provide a more scalable and robust variant of RL algorithms

Local optimization around existing policies π parametrized by a set of policy parameter θ occurs by incrementally change parameters to improve the expected return

$$\theta_{t+1} = \theta_t + \Delta\theta \tag{2.83}$$

The nature of the parameter update $\Delta\theta$ brings about a wide range of methods, ranging from pairwise comparisons (Strens et al., 2001; Ng et al., 2004), gradient estimation from finite differences (Geng et al., 2006; Kohl et al., 2004; Mitsunaga et al., 2006; Roberts et al., 2010; Sato et al., 2002; Tedrake et al., 2005), to stochastic optimization methods (Bagnell et al., 2001), cross-entropy (Rubinstein et al., 2004), differential dynamic programming (Atkeson, 1998), shooting approaches (T. Betts, 2001). These methods can be categorized into three classes: policy gradient (PG) methods, expectation-maximization (EM) or information theoretic updates (Kappen, 2005).

Another useful distinction classifies RL approaches into model based vs. model free methods. The former requires learning a model of the state dynamics and use it to improve the policy, while the latter directly learns a policy based on sampled trajectories of the environment. Model free learning explores all three variants of policy parameter updates while model base focuses mainly on policy gradient methods. Next, we describe policy gradient methods, since they provide some

of the most widely use approaches in Robotics.

2.7.1 *Policy gradient methods*

Policy gradient approaches may be described as updates on policy parameters following the direction of the gradient of the expected return $J(\pi) = \frac{1}{T}\sum_T \mathbb{E}[\gamma^{t-1} r_t \mid \pi_\theta]$

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi). \tag{2.84}$$

where α is a stepsize that dictates how far we improve along each gradient direction. Different forms of estimating the gradient bring about different techniques: some are based on likelihood ratio estimation (Sutton et al., 1999), others arise from Expectation-Maximization approaches (Toussaint et al., 2011; Dayan et al., 1997; Vlassis et al., 2009; Kober et al., 2010), path integral methods (Kappen, 2005), or more recently natural gradient approaches (Peters et al., 2008; Wang et al., 2016; Schulman et al., 2015)

Reinforcement learning within Robotics differs from many other learning domains since they often require high dimensional continuous state and action spaces. Contrary to standard optimal control paradigm in Robotics it is generally unrealistic to assume the true state is completely observable and noise free. Uncertainty plays a crucial role in modeling under these settings. Uncertainty generally involves two main independent sources: predictability associated with model errors, leading to uncertainty in predicting future states, and sensing in which model noise and uncertainty from initial conditions, sensors and memory of previously applied actions can make different states look similar. For this reason models in Robotics need to be robust to undermodeling and deal with model uncertainty.

Furthermore, real robot experience is costly and difficult to reproduce. Usually it cannot be replaced by learning from simulations alone since even small modeling errors can accumulate to substantially different outcomes, specially in highly dynamic tasks.

Reward shaping poses another challenge in Robotics (Laud, 2004), in that the design of appropriate reward functions requires some domain knowledge.

Over and all, many complex tasks in Robotics leverage the information provided by models to counteract these difficulties (Atkeson et al., 1997; Abbeel et al., 2008; Deisenroth et al., 2011), and typically policy search approaches are preferred over value-based methods, since they provide more robust estimates of policies (Gullapalli et al., 1994; Miyamoto et al., 1996; Bagnell et al., 2001; Kohl et al., 2004; Tedrake et al., 2005; Peters et al., 2008; Kober et al., 2009).

In this section we have summarized the main literature in sequence

prediction and planning using moment-based approaches and kernel algorithms. Next, we will present the work developed under this thesis regarding both prediction Chapter 3, planning Chapter 4 and a combination of both under a reinforcement learning setting Chapter 5 respectively.

3
Sequence Labeling with Method of Moments

In this chapter, we introduce a sequence prediction algorithm based on moment-matching techniques that can be applied to discrete domains, such as sequence labeling problems. We assume the existence of certain observations that correlate with each label exactly. We provide an efficient algorithm that is able to perform label prediction in large datasets, using little supervised information. We rely only on moment-based approaches to recover the state marginals and consecutively the most probable label for each observation in the sequence. However, this algorithm is amenable to post-refinement using likelihood-based approaches or gradient descent, similarly to what is done in the following chapters.

We apply the proposed algorithm to the domain of Natural Language Processing in a part-of-speech (POS) tagging problem. We start by describing two generative sequence models in Section 3.2: hidden Markov models (HMMs) §3.2.1, and their generalization with emission features (§3.2.2). Later, we propose a weakly-supervised method for estimating these models' parameters (§3.3–§3.4) based only on observed statistics of words and contexts. Section 3.5–3.6 describes some practical advice and empirical analysis of the moment-based approach in a POS tagging problem for twitter and for a low resource language.

3.1 Motivation

Statistical learning of NLP models is often limited by the scarcity of annotated data. Weakly supervised methods have been proposed as an alternative to laborious manual annotation, combining large amounts of unlabeled data with limited resources, such as tag dictionaries or small annotated datasets (Merialdo, 1994; Smith et al., 2005; Garrette et al., 2013). Unfortunately, most semi-supervised learning algorithms for the structured problems found in NLP are computationally expensive, requiring multiple decoding passes through the unlabeled data, or expensive similarity graphs. More scalable learning algorithms are

necessary if we wish to take advantage of very large corpora. Certain semi-supervised learning methods like label propagation do not require decoding the unlabeled data, but they often require building a huge, expensive graph.

In this chapter, we propose a moment-matching method for semi-supervised learning of sequence models, as stated in the background §2.3.1. Spectral learning and moment-matching approaches have recently proved a viable alternative to expectation-maximization (EM) for unsupervised learning (Hsu et al., 2012b; Balle et al., 2012c; Bailly et al., 2013b), supervised learning with latent variables (Cohen et al., 2014; Quattoni et al., 2014; Stratos et al., 2013) and topic modeling (Arora et al., 2013; Nguyen et al., 2015). These methods have learnability guarantees, do not suffer from local optima, and are computationally less demanding. Current moment-matching approaches are, however, less flexible—for example, it is not obvious how to adapt existing methods to semi-supervised learning, where the latent variables do not represent arbitrary states, but rather denote labels, observed for some data points and unobserved for others. One drawback of spectral methods is that they do not provide an explicit representation for states, although there has been some work that attempts to recover this transformation (Hsu et al., 2012b), this method does not provide good results in practice. Observations can be predicted as products of operators, without requiring to explicitly identify the hidden states, by marginalizing over all possible states. They have been used as forms of doing prediction over observations without taking into account the explicit hidden state representation (Boots et al., 2011a). They have been used also as decorations in a structured problem, enriching labels with more refined information (Stratos et al., 2013; Cohen et al., 2012). Different approaches based on optimization techniques have been proposed, where a more generic perspective of the problem is provided, allowing integration with other parameter constraints (Balle et al., 2012a; Chaganty et al., 2014a).

Existing spectral learning approaches estimate operators invariant to similarity transforms, as in §2.5 (Jaeger, 2000). In order for states to represent well-defined labels, it would be necessary to break this invariance, which would spoil the current estimation procedures based on matrix and tensor decomposition, in §2.4.3. Next, we describe an alternative method that enables the explicit recovery of model parameters. This chapter does not make use of predictive state representations. Instead, it uncovers the relation between observations and states by “anchoring" specific observations to hidden states, in §2.4.4. This approach learns the model parameters directly. Although it does not have the same expressive power as a PSR, it finds parameters that match moments of observations while allowing to add labeled infor-

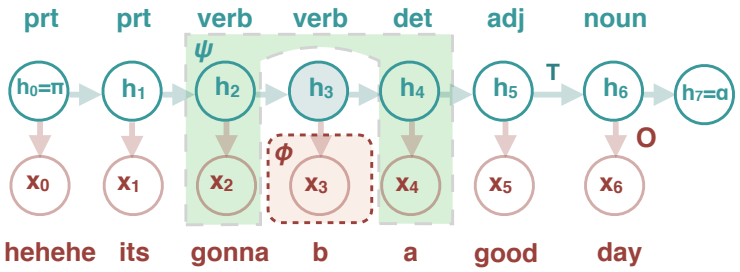


Figure 3.1: HMM, context (green) conditionally independent of present (red) x_ℓ given state h_ℓ.

mation easily. This is mostly relevant if we want to achieve good performance, compared with other semi-supervised methods such as EM. Unlike spectral methods, ours does not require an orthogonal decomposition of any matrix or tensor. Instead, it considers a more restricted form of supervision: words that have unambiguous annotations, so-called **anchor words** (Arora et al., 2013). Rather than identifying anchor words from unlabeled data (Stratos et al., 2016), we extract them from a small labeled dataset or from a dictionary. Given the anchor words, the estimation of the model parameters can be made efficient by collecting moment statistics from unlabeled data, then solving a small quadratic program for each word.

Our contributions are as follows:

- We adapt anchor methods to semi-supervised learning of generative sequence models.
- We show how our method can also handle log-linear feature-based emissions.
- We apply this model to POS tagging. Our experiments on the Twitter dataset introduced by Gimpel et al. (2011) and on the dataset introduced by Garrette et al. (2013) for Malagasy, a low-resource language, show that our method does particularly well with very little labeled data, outperforming semi-supervised EM and self-training.

3.2 *Sequence Labeling*

In this chapter, we address the problem of sequence labeling. Let $\boldsymbol{x}_{1:L} = \langle x_1, \ldots, x_L \rangle$ be a sequence of L input observations (for example, words in a sentence). The goal is to predict a sequence of labels $\boldsymbol{h}_{1:L} = \langle h_1, \ldots, h_L \rangle$, where each h_i is a label for the observation x_i (for example, the word's POS tag).

3.2.1 *Hidden Markov Models*

We define random variables $\boldsymbol{X} := \langle X_1, \ldots, X_L \rangle$ and $\boldsymbol{H} := \langle H_1, \ldots, H_L \rangle$, corresponding to observations and labels, respectively. Each X_i is a

random variable over a set $\mathcal{X}$ (the vocabulary), and each H_i ranges over $\mathcal{H}$ (a finite set of "states" or "labels"). We denote the vocabulary size by $V = |\mathcal{X}|$, and the number of labels by $K = |\mathcal{H}|$. A first-order HMM has the following generative scheme:

$$p(\boldsymbol{X} = \boldsymbol{x}_{1:L}, \boldsymbol{H} = \boldsymbol{h}_{1:L}) := \tag{3.1}$$
$$\prod_{\ell=1}^{L} p(X_\ell{=}x_\ell \mid H_\ell{=}h_\ell) \prod_{\ell=0}^{L} p(H_{\ell+1}{=}h_{\ell+1} \mid H_\ell{=}h_\ell),$$

where we have defined $h_0 = \text{START}$ and $h_{L+1} = \text{STOP}$. We adopt the following notation for the parameters of the HMM in Section 2.3.1, the emission matrix $O \in \mathbb{R}^{V \times K}$, defined as $O_{x,h} := p(X_\ell = x \mid H_\ell = h)$, $\forall h \in \mathcal{H}, x \in \mathcal{X}$, and the transition matrix $T \in \mathbb{R}^{(K+2)\times(K+2)}$, defined as $T_{h,h'} := p(H_{\ell+1} = h \mid H_\ell = h')$, for every $h, h' \in \mathcal{H} \cup \{\text{START}, \text{STOP}\}$.[1] This matrix satisfies $T^\top \mathbf{1} = \mathbf{1}$.[2]

[1] Where $T_{\text{START,STOP}} = 1$ and $T_{\text{START},h'} = 0$ for $h' \neq \text{STOP}$

[2] That is, it satisfies $\sum_{h=1}^{K} p(H_{\ell+1} = h \mid H_\ell = h') + p(H_{\ell+1} = \text{STOP} \mid H_\ell = h') = 1$; and also $\sum_{h=1}^{K} p(H_1 = h \mid H_0 = \text{START}) = 1$.

Throughout the rest of this chapter we will adopt $X \equiv X_\ell$ and $H \equiv H_\ell$ to simplify notation, whenever the index ℓ is clear from the context. Under this generative process, predicting the most probable label sequence $\boldsymbol{h}_{1:L}$ given observations $\boldsymbol{x}_{1:L}$ is accomplished with the Viterbi algorithm in $O(LK^2)$ time.

If labeled data are available, the model parameters O and T can be estimated with the maximum likelihood principle, which boils down to a simple counting of events and normalization. If we only have unlabeled data, the traditional approach is the expectation-maximization (EM) algorithm in §2.4.1, which alternately decodes the unlabeled examples and updates the model parameters, requiring multiple passes over the data. The same algorithm can be used in semi-supervised learning when labeled and unlabeled data are combined, by initializing the model parameters with the supervised estimates and interpolating the estimates in the M-step.

3.2.2 *Feature-Based Hidden Markov Models*

Sequence models with log-linear emissions have been considered by Smith et al. (2005), in a discriminative setting, and by Berg-Kirkpatrick et al. (2010), as generative models for POS induction. Feature-based HMMs (FHMMs) define a feature function for words, $\boldsymbol{\phi}(X) \in \mathbb{R}^W$, which can be discrete or continuous. This allows, for example, to indicate whether an observation, corresponding to a word, starts with an uppercase letter, contains digits or has specific affixes. More generally, it helps with the treatment of out-of-vocabulary words. The emission probabilities are modeled as K conditional distributions parametrized by a log-linear model, where the $\boldsymbol{\theta}_h \in \mathbb{R}^W$ represent feature weights:

$$p(X = x \mid H = h) := \exp(\boldsymbol{\theta}_h^\top \boldsymbol{\phi}(x)) / Z(\boldsymbol{\theta}_h). \tag{3.2}$$

Algorithm 3: Semi-Supervised Learning of HMMs with Moments

Input: Labeled dataset $\mathcal{D}_L$, unlabeled dataset $\mathcal{D}_U$
Output: Estimates of emissions O and transitions T

1: Estimate context-word moments $\widehat{Q}$ from $\mathcal{D}_U$ (Eq. 3.5)
2: **for** each label $h \in \mathcal{H}$ **do**
3: Extract set of anchor words $\mathcal{A}(h)$ from $\mathcal{D}_L$ (§3.3.2)
4: Estimate context-label moments $\widehat{R}$ from anchors and $\mathcal{D}_U$ (Eq. 3.12)
5: **for** each word $w \in [V]$ **do**
6: Solve the QP in Eq. 3.14 to obtain γ_w from $\widehat{Q}, \widehat{R}$
7: Estimate emissions O from Γ via Eq. 3.15
8: Estimate transitions T (§ 3.3.5)
9: **Return:** Emission and transition matrices (O, T)

Above, $Z(\boldsymbol{\theta}_h) := \sum_{x' \in \mathcal{X}} \exp(\boldsymbol{\theta}_h^\top \boldsymbol{\phi}(x'))$ is a normalization factor. We will show in §3.4 how our moment-based semi-supervised method can also be used to learn the feature weights $\boldsymbol{\theta}_h$.

3.3 *Semi-Supervised Learning via Moments*

We now describe our moment-based semi-supervised learning method for HMMs. Throughout, we assume the availability of a small labeled dataset $\mathcal{D}_L$ and a large unlabeled dataset $\mathcal{D}_U$.

The full roadmap of our method is shown as Algorithm 3. Key to our method is the decomposition of a **context-word moment matrix** $Q \in \mathbb{R}^{C \times V}$, which counts co-occurrences of words and contexts, and will be formally defined in §3.3.1. Such co-occurrence matrices are often collected in NLP, for various problems, ranging from dimensionality reduction of documents using latent semantic indexing (Deerwester et al., 1990b; Landauer et al., 1998), distributional semantics (Schütze, 1998; Levy et al., 2015) and word embedding generation (Dhillon et al., 2015b; Osborne et al., 2016). We can build such a moment matrix entirely from the unlabeled data $\mathcal{D}_U$. The same unlabeled data is used to build an estimate of a **context-label moment matrix** $R \in \mathbb{R}^{C \times K}$, as explained in §3.3.3. This is done by first identifying words that are unambiguously associated with each label h, called **anchor words**, with the aid of a few labeled data; this is outlined in §3.3.2. Finally, given empirical estimates of Q and R, we estimate the emission matrix O by solving a small optimization problem independently per word (§3.3.4). The transition matrix T is obtained directly from the labeled dataset $\mathcal{D}_L$ by maximizing the likelihood.

3.3.1 *Moments of Contexts and Words*

To formalize the notion of "context," we introduce the shorthand $\boldsymbol{Z}_\ell := \langle \boldsymbol{X}_{1:(\ell-1)}, \boldsymbol{X}_{(\ell+1):L} \rangle$. Importantly, the HMM in Eq. 3.1 entails the fol-

lowing conditional independence assumption: X_ℓ is conditionally independent of the surrounding context Z_ℓ given the hidden state H_ℓ. This is illustrated in Figure 3.1, using POS tagging as an example task.

We introduce a vector of **context features** $\boldsymbol{\psi}(\boldsymbol{Z}_\ell) \in \mathbb{R}^C$, which may look arbitrarily within the context $\boldsymbol{Z}_\ell$ (left or right), but not at X_ℓ itself. These features could be "one-hot" representations or other reduced-dimensionality embeddings (as described later in §3.5). Consider the word $w \in \mathcal{X}$ an instance of $X \equiv X_\ell$. A pivotal matrix in our formulation is the matrix $Q \in \mathbb{R}^{V \times C}$, defined as:

$$Q_{w,c} := \mathbb{E}[\psi_c(\boldsymbol{Z}) \mid X = w]. \tag{3.3}$$

Expectations here are taken with respect to the probabilistic model in Eq. 3.1 that generates the data. The following marginal probabilities will also be necessary:

$$q_c := \mathbb{E}[\psi_c(\boldsymbol{Z})], \quad p_w := p(X = w). \tag{3.4}$$

Since all the variables in Eqs. 3.3–3.4 are observed, we can easily obtain empirical estimates by taking expectations over the unlabeled data:

$$\widehat{Q}_{w,c} = \frac{\sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \psi_c(\boldsymbol{z}) \mathbb{1}(x = w)}{\sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \mathbb{1}(x = w)}, \tag{3.5}$$

$$\widehat{q}_c = \sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \psi_c(\boldsymbol{z}) / |\mathcal{D}_U|, \tag{3.6}$$

$$\widehat{p}_w = \sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \mathbb{1}(x = w) / |\mathcal{D}_U|. \tag{3.7}$$

where we take $\mathbb{1}(x = w)$ to be the indicator for word w. Note that, under our modeling assumptions, Q decomposes in terms of its hidden states:

$$\mathbb{E}[\psi_c(\boldsymbol{Z}) \mid X = w] = \sum_{h \in \mathcal{H}} p(H = h \mid X = w) \mathbb{E}[\psi_c(\boldsymbol{Z}) \mid H = h] \tag{3.8}$$

The reason why this holds is that, as stated above, $\boldsymbol{Z}$ and X are conditionally independent given H.

3.3.2 *Anchor Words*

Following Arora et al. (2013) and Cohen et al. (2014), we identify **anchor words** whose hidden state is assumed to be deterministic, regardless of context. In this thesis, we generalize this notion to more than one anchor word per label, for improved context estimates. This allows for more flexible forms of anchors with weak supervision. For each state $h \in \mathcal{H}$, let its set of anchor words be

$$\begin{aligned} \mathcal{A}(h) &= \{w \in \mathcal{X} : p(H = h \mid X = w) = 1\} \\ &= \left\{w \in \mathcal{X} : O_{w,h} > 0 \wedge O_{w,h'} = 0, \forall h' \neq h\right\}. \end{aligned} \tag{3.9}$$

That is, $\mathcal{A}(h)$ is the set of unambiguous words that always take the label h. This can be estimated from the labeled dataset $\mathcal{D}_L$ by collecting the most frequent unambiguous words for each label.

Algorithms for identifying $\mathcal{A}(h)$ from unlabeled data alone were proposed by Arora et al. (2013) and Zhou et al. (2014), with application to topic models. Our work differs in which we do not aim to discover anchor words from pure unlabeled data, but rather exploit the fact that small amounts of labeled data are commonly available in many NLP tasks—better anchors can be extracted easily from such small labeled datasets. There is however a trade off between the amount of labeled information and the quality of the anchors. In the proposed approach, we assume the anchors are pure unambiguous observations, however these anchors may be imperfectly identified, for example, if the labeled dataset is too small. In practice we verified that it is enough to select frequent but impure anchors for each label. In §3.5 we give a more detailed description of the selection process.

3.3.3 *Moments of Contexts and Labels*

We define the matrix $R \in \mathbb{R}^{K \times C}$ as follows:

$$R_{h,c} := \mathbb{E}[\psi_c(\boldsymbol{Z}) \mid H = h]. \tag{3.10}$$

Since the expectation in Eq. 3.10 is conditioned on the (unobserved) label h, we cannot directly estimate it using moments of observed variables, as we do for Q. However, if we have identified sets of anchor words for each label $h \in \mathcal{H}$, we have:

$$\begin{aligned}
&\mathbb{E}[\psi_c(\boldsymbol{Z}) \mid X \in \mathcal{A}(h)] = \\
&\quad = \sum_{h'} \mathbb{E}[\psi_c(\boldsymbol{Z}) \mid H = h'] \underbrace{p(H = h' \mid X \in \mathcal{A}(h))}_{=\mathbb{1}(h'=h)} \\
&\quad = R_{h,c}.
\end{aligned} \tag{3.11}$$

Therefore, given the set of anchor words $\mathcal{A}(h)$, the hth row of R can be estimated in a single pass over the unlabeled data, as follows:

$$\widehat{R}_{h,c} = \frac{\sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \psi_c(\boldsymbol{z}) \mathbb{1}(x \in \mathcal{A}(h))}{\sum_{x,\boldsymbol{z} \in \mathcal{D}_U} \mathbb{1}(x \in \mathcal{A}(h))} \tag{3.12}$$

3.3.4 *Emission Distributions*

We can now put all the ingredients above together to estimate the emission probability matrix O. The procedure we propose here is computationally very efficient, since only one pass is required over the unlabeled data, to collect the co-occurrence statistics $\widehat{Q}$ and $\widehat{R}$. The emissions will be estimated from these moments by solving a small

problem independently for each word. Unlike EM and self-training, no decoding is necessary, only counting and normalizing; and unlike label propagation methods, there is no requirement to build a graph with the unlabeled data.

The crux of our method is the decomposition in Eq. 3.8, which is combined with the one-to-one correspondence between labels h and anchor words $\mathcal{A}(h)$ in Figure 3.3.4. We can rewrite Eq. 3.8 as:

$$Q_{w,c} = \sum_h p(H = h \mid X = w) R_{h,c}. \tag{3.13}$$

In matrix notation, we have $Q = \Gamma R$, where $\Gamma \in \mathbb{R}^{V \times K}$ is defined as $\Gamma_{w,h} := p(H = h \mid X = w)$.

If we had infinite unlabeled data, our moment estimates $\widehat{Q}$ and $\widehat{R}$ would be perfect and we could solve the system of equations in Eq. 3.13 to obtain Γ exactly. Since we have finite data, we resort to a least squares solution. This corresponds to solving a simple quadratic program (QP) per word, independent from all the other words, as follows. Denote by $\mathbf{q}_w := \mathbb{E}[\boldsymbol{\psi}(\boldsymbol{Z}) \mid X = w] \in \mathbb{R}^C$ and by $\gamma_w := p(H = \cdot \mid X = w) \in \mathbb{R}^K$ the wth rows of Q and Γ, respectively. We estimate the latter distribution following Arora et al. (2013):

$$\begin{aligned} \widehat{\gamma}_w \;=\; & \arg\min_{\gamma_w} \quad \left\| \mathbf{q}_w - \gamma_w^\top R \right\|_2^2 \\ & \text{s.t.} \qquad \mathbf{1}^\top \gamma_w = 1,\ \gamma_w \geq \mathbf{0}. \end{aligned} \tag{3.14}$$

Note that this QP is very small—it has only K variables—however the vocabulary size is typically in the order of $10^4 \sim 10^5$ word types, which we can solve efficiently (1.7 ms on average for each word w, in Gurobi, with $K = 12$).

Given the probability tables for $p(H = h \mid X = w)$, we can estimate the emission probabilities O by direct application of Bayes' rule:

$$\widehat{O}_{w,h} = \frac{p(H = h \mid X = w) \times p(X = w)}{p(H = h)} \tag{3.15}$$

$$= \frac{\widehat{\gamma}_{w,c} \times \overbrace{\widehat{p}_w}^{\text{Eq. 3.7}}}{\sum_{w'} \widehat{\gamma}_{w',c} \times \widehat{p}_{w'}}. \tag{3.16}$$

These parameters are guaranteed to lie in the probability simplex, avoiding the need of heuristics for dealing with "negative" and "unnormalized" probabilities required by prior work in spectral learning (Cohen et al., 2013c).

3.3.5 *Transition Distributions*

It remains to estimate the transition matrix T. We propose two methods for estimating the transition matrix T of the model. The first uses

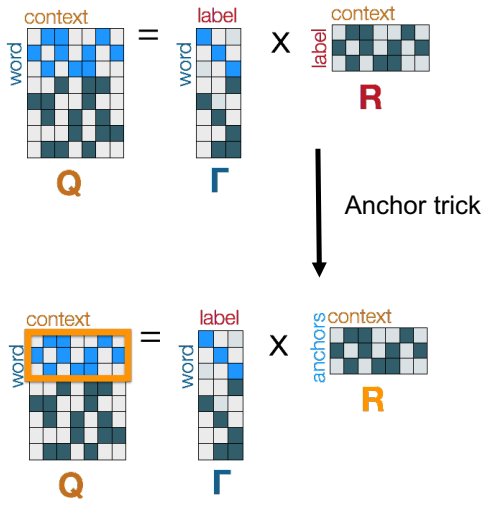


Figure 3.2: Anchor trick to solve QP from observable moments, replacing state by anchor surrogates (top light blue rows in Q) in Eq. 3.11.

relative frequencies directly estimated from annotated data, ignoring the unannotated data.

For the problems tackled in this chapter, the number of labels K is small, compared to the vocabulary size V. The transition matrix has only $O(K^2)$ degrees of freedom, and we found it effective to estimate it using the labeled sequences in $\mathcal{D}_L$ alone, without any refinement. This was done by smoothed maximum likelihood estimation on the labeled data, which boils down to counting occurrences of consecutive labels, applying add-one smoothing to avoid zero probabilities for unobserved transitions, and normalizing. Let $M_{h',h} = p(H_\ell = h', H_{\ell-1} = h), \forall_{h,h' \in \mathcal{H}}$, so that $M \in \mathbb{R}^{(K+1)\times(K+1)}$, contains an extra row and column for start/stop probabilities, respectively. The conditional transition probability matrix $T \in \mathbb{R}^{K\times K}$ can be directly derived from M through proper normalization (first method):

$$T_{h,h'} = p(H_\ell = h | H_{\ell-1} = h') = \frac{M_{h,h'}}{\sum_{h''=1}^{K} M_{h'',h'}}$$

The start and stop probabilities can also be read off of the appropriate row and column ($\boldsymbol{\pi}_h = M_{h,K+1}$ and $\mathbf{f}^*_h = M_{K+1,h}$).

For problems with numerous labels, a second alternative is the composite likelihood method (Chaganty et al., 2014b).[3] Given an emission matrix $\widehat{O}$, the maximization of the composite log-likelihood function leads to a convex optimization problem that can be efficiently optimized with an EM algorithm. A similar procedure was carried out by Cohen et al. (2014).

[3] In the experimental section, the compositional likelihood method was not competitive with estimating the transition matrices directly from the labeled data, on the datasets described in Figure 3.3. However, this may be a viable alternative if there is no labeled data and the anchors are extracted from gazetteers or a dictionary.

This approach involves a maximization of a pairwise composite likelihood over pairs of words $\mathcal{L}_{CL}(X_{\ell-1}, X_\ell)$=$p(X_{\ell-1}, X_\ell)$ in Eq. 3.17 (Liang et al., 2003; Lindsay, 1988). Composite likelihood, or also known as pseudo-likelihood (Molenberghs et al., 2006), provide a popular alternative to model the likelihood function, in cases where the full likelihood is unavailable or difficult to model. In this model, we look into pairwise composite likelihoods to account for the transition parameters. Composite likelihoods may be regarded as misspecified likelihoods derived from the model's independence assumptions.[4] Under the model, the pairwise word probability decomposes into $p(X_{\ell-1}{=}x', X_\ell{=}x) = \sum_{h,h'} M_{h,h'} O_{x',h'} O_{x,h}$.

[4] The misspecification refers to the conditional independence assumption that observation are conditionally independent given the state and each state only depends on the previous state, giving rise to a factorization into pairwise terms.

$$\arg\max_{M} \sum_{\ell} \log \mathcal{L}_{CL}(X_{\ell-1}, X_\ell) \qquad (3.17)$$
$$\text{s. t. } \mathbf{1}^\top M \mathbf{1} = 1, M \geq \mathbf{0}$$

Even if the model is misspecified, the maximum composite likelihood parameter $\theta = M, O$ is consistent and asymptotically normal distributed, with variance equal to the inverse of the Godambe distribution $G(\theta) =$

Algorithm 4: Get Transitions

Input: $O, \tilde{p}(X_\ell, X_{\ell-1}), M_0$

Run for each iteration:

1: **for** $i, j = 1, \ldots, V$ **do**

E-Step - Compute state posteriors:

2: $p(h, h', x_i, x_j) = M_{h,h'} O_{x_i,h'} O_{x_j,h}$

3: $p(h, h'|x_i, x_j) = \dfrac{p(h, h', x_i, x_j)}{\sum_{h,h'} p(h, h', x_i, x_j)}$

Update composite likelihood $\mathcal{L}_{cl}$ iteratively for pairs of words

4: $\mathcal{L}_{cl} \mathrel{+}= \tilde{p}(x_i, x_j) \log \left[\sum_{h,h'} p(h, h', x_i, x_j)\right]$

M-Step - Update parameters

5: $M_{h,h'} = \tilde{p}(x_i, x_j) p(h, h'|x_i, x_j)$

6: Normalize to $\mathbf{1}^T M \mathbf{1} = 1$.

7: Get initial $\boldsymbol{\pi}$, final $\boldsymbol{f}$, and transition probabilities T from M.

8: **Return:** $\langle \boldsymbol{T}, \boldsymbol{\pi}, \boldsymbol{f}^* \rangle$.

$H(\theta)J(\theta)^{-1}H(\theta)$ (Godambe, 1960), where $H(\theta) = \mathbb{E}[-\nabla^2_\theta \log \mathcal{L}_{CL}]$ refers to the sensitivity matrix, and $J(\theta) = \text{var}_\theta[\nabla_\theta \log(\mathcal{L}_{CL})]$ its variability matrix. When the composite likelihood is the likelihood function, then $H = J$ and the Godambe information is the same as the Fisher Information matrix $I(\theta) = \text{var}_\theta(\nabla_\theta \log p(X))$ of the full log-likelihood function, and the maximum composite likelihood is said to be perfectly efficient $G = H = I$, however this is not true in the most general case (Varin et al., 2005; Varin et al., 2011). ℓ ranges over the positions in the data where a word (or stop symbol $*$) is preceded by another word. This process can be performed over large amounts of unlabeled data. The optimization in Eq. 3.17 is convex in M; Algorithm 4 presents an alternating minimization approach (EM) that finds a global optimum, in just a few iterations (typically 5-10) (Cohen et al., 2014; Chaganty et al., 2014b).

3.4 *Feature-Based Emissions*

Next, we extend our method to estimate the parameters of the FHMM in §3.2.2. Other than contextual features $\boldsymbol{\psi}(\boldsymbol{Z}) \in \mathbb{R}^C$, we also assume a feature encoding function for words, $\boldsymbol{\phi}(X) \in \mathbb{R}^W$. Our framework, illustrated in Algorithm 5, allows for both discrete and continuous word and context features. Lines 2–4 are the same as in Algorithm 3, replacing word occurrences with expected values of word features (we redefine Q and Γ to cope with features instead of words). The main difference with respect to Algorithm 3 is that we do not estimate emission probabilities; rather, we first estimate the **mean parameters** (feature expectations $\mathbb{E}[\boldsymbol{\phi}(X) \mid H = h]$), by solving one QP for each emission feature; and then we solve a convex optimization problem, for each

Algorithm 5: Semi-Supervised Learning of Feature-Based HMMs with Moments
Input: Labeled dataset $\mathcal{D}_L$, unlabeled dataset $\mathcal{D}_U$
Output: Emission log-linear parameters Θ and transitions T
Estimate context-word moments $\widehat{Q}$ from $\mathcal{D}_U$ (Eq. 3.21)
for each label $h \in \mathcal{H}$ **do**
    Extract set of anchor words $\mathcal{A}(h)$ from $\mathcal{D}_L$ (§3.3.2)
Estimate context-label moments $\widehat{R}$ from the anchors and $\mathcal{D}_U$ (Eq. 3.12)
for each word feature $j \in [W]$ **do**
    Solve the QP in Eq. 3.23 to obtain γ_j from $\widehat{Q}, \widehat{R}$
for each label $h \in \mathcal{H}$ **do**
    Estimate the mean parameters $\boldsymbol{\mu}_h$ from Γ (Eq. 3.25)
    Estimate the canonical parameters $\boldsymbol{\theta}_h$ from $\boldsymbol{\mu}_h$ in Algorithm 6
Estimate transitions T from $\mathcal{D}_L$
Return: Canonical parameters and transition matrix (Θ, T)

label h, to recover the log-linear weights over emission features (called **canonical parameters**).

3.4.1 *Estimation of Mean Parameters*

First of all, we replace word probabilities by expectations over word features. We redefine the matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{W \times K}$ as follows:

$$\Gamma_{j,h} := \frac{p(H = h) \times \mathbb{E}[\phi_j(X) \mid H = h]}{\mathbb{E}[\phi_j(X)]}. \tag{3.18}$$

Note that, with one-hot word features, we have $\mathbb{E}[\phi_w(X) \mid H = h] = P(X = w \mid H = h)$, $\mathbb{E}[\phi_w(X)] = p(X = w)$, and therefore $\Gamma_{w,h} = p(H = h \mid X = w)$, so this can be regarded as a generalization of the framework in §3.3.4. Second, we redefine the context-word moment matrix Q as the following matrix in $\mathbb{R}^{C \times W}$:

$$Q_{j,c} = \mathbb{E}\left[\psi_c(\boldsymbol{Z}) \times \phi_j(X)\right] / \mathbb{E}[\phi_j(X)]. \tag{3.19}$$

Again, note that we recover the previous Q if we use one-hot word features. We then have the following generalization of Eq. 3.13:

$$\begin{aligned} &\mathbb{E}\left[\psi_c(\boldsymbol{Z}) \times \phi_j(X)\right] / \mathbb{E}[\phi_j(X)] = \\ &\quad \textstyle\sum_h \frac{P(H=h)\mathbb{E}[\phi_j(X) \mid H=h]}{\mathbb{E}[\phi_j(X)]} \mathbb{E}\left[\psi_c(\boldsymbol{Z}) \mid H = h\right], \end{aligned} \tag{3.20}$$

or, in matrix notation, $Q = \Gamma R$.

Again, matrices Q and R can be estimated from data by collecting empirical feature expectations over unlabeled sequences of observations. For R use Eq. 3.12 with no change; for Q replace Eq. 3.5 by

$$\widehat{Q}_{j,c} = \textstyle\sum_{x,z \in \mathcal{D}_U} \psi_c(z)\phi_j(x) / \sum_{x \in \mathcal{D}_U} \phi_j(x). \tag{3.21}$$

Let $\mathbf{q}_j \in \mathbb{R}^C$ and $\gamma_j \in \mathbb{R}^K$ be rows of $\widehat{Q}$ and $\widehat{\Gamma}$, respectively. Note that we must have

$$\mathbf{1}^\top \gamma_j = \sum_h \frac{P(H=h)\mathbb{E}\big[\phi_j(X)|H=h\big]}{\mathbb{E}\big[\phi_j(X)\big]} = 1, \tag{3.22}$$

since $\mathbb{E}[\phi_j(X)]=\sum_h P(H=h)\mathbb{E}\left[\phi_j(X) \mid H=h\right]$. We rewrite the QP to minimize the squared difference for each dimension j independently:

$$\widehat{\gamma}_j \;=\; \arg\min_{\gamma_j} \quad \left\|\mathbf{q}_j - \gamma_j^\top R\right\|_2^2 \;\text{ s.t. }\; \mathbf{1}^\top \gamma_j = 1. \tag{3.23}$$

Note that, if $\boldsymbol{\phi}(x) \geq \mathbf{0}$ for all $x \in \mathcal{X}$, then we must have $\gamma_j \geq \mathbf{0}$, and therefore we may impose this inequality as an additional constraint.[5]

Let $\bar{\gamma} \in \mathbb{R}^K$ be the vector of state probabilities, with entries $\bar{\gamma}_h := p(H = h)$ for $h \in \mathcal{H}$. This vector can also be recovered from the unlabeled dataset and the set of anchors, by solving another QP that aggregates information for all words:

$$\bar{\gamma} \;=\; \arg\min_{\bar{\gamma}} \quad \left\|\bar{\boldsymbol{q}} - \bar{\gamma}^\top R\right\|_2^2 \;\text{ s.t. }\; \mathbf{1}^\top \bar{\gamma} = 1,\; \bar{\gamma} \geq \mathbf{0}. \tag{3.24}$$

where $\bar{\boldsymbol{q}} := \widehat{\mathbb{E}}[\boldsymbol{\psi}(\boldsymbol{Z})] \in \mathbb{R}^C$ is the vector whose entries are defined in Eq. 3.6.

Let $\boldsymbol{\mu}_h := \mathbb{E}[\boldsymbol{\phi}(X) \mid H = h] \in \mathbb{R}^W$ be the mean parameters of the distribution for each state h. These parameters are computed by solving W independent QPs (Eq. 3.23), yielding the matrix Γ defined in Eq. 3.18, and then applying the formula:

$$\mu_{j,h} = \Gamma_{j,h} \times \mathbb{E}[\phi_j(X)] \Big/ \bar{\gamma}_h, \tag{3.25}$$

with $\bar{\gamma}_h = p(H = h)$ estimated as in Eq. 3.24.

3.4.2 *Estimation of Canonical Parameters*

To compute a mapping from mean parameters $\boldsymbol{\mu}_h$ to canonical parameters $\boldsymbol{\theta}_h$, we use the well-known Fenchel-Legendre duality between the entropy and the log-partition function (Wainwright et al., 2008). Namely, we need to solve the following convex optimization problem:

$$\widehat{\boldsymbol{\theta}}_h = \arg\max_{\boldsymbol{\theta}_h} \; \boldsymbol{\theta}_h^\top \boldsymbol{\mu}_h - \log Z(\boldsymbol{\theta}_h) + \epsilon\|\boldsymbol{\theta}_h\|, \tag{3.26}$$

where ϵ is a regularization constant.[6] In practice, this regularization is important, since it prevents $\boldsymbol{\theta}_h$ from growing unbounded whenever $\boldsymbol{\mu}_h$ falls outside the marginal polytope of possible mean parameters. We solve Eq. 3.26 with the limited-memory BFGS algorithm (Liu et al., 1989). Algorithm 6 summarizes the procedure.

[5] If we ensure strictly positive constraints, then $\mathbb{E}[\phi_j]$ will be always non-zero. However when this constraint is not specified if $q_{j,i} = 0$, then the inequality in Eq. 3.22 may be regarded as $P(H = h_i)\mathbb{E}\left[\phi_j(X) \mid H = h_i\right] = \mathbb{E}\left[\phi_j(X)\right]$, where $\mathbb{E}\left[\phi_j(X)\right] = 0$ and so $\mathbb{E}\left[\phi_j(X) \mid h_i\right] = 0$.

[6] As shown by Zhu et al. (1999) and Altun et al. (2006), this regularization is equivalent, in the dual, to a "soft" constraint $\|\mathbb{E}_{\boldsymbol{\theta}_h}[\boldsymbol{\phi}(X) \mid H = h] - \boldsymbol{\mu}_h\|_2 \leq \epsilon$, as opposed to a strict equality.

Algorithm 6: Get Canonical Parameters

Input: mean paramaters $\boldsymbol{\mu}_h$

Compute sufficient statistics matrix for all words $S \in \mathbb{R}^{W \times V}$

while $\|\nabla f(\theta_h)\|_2 > tol$ **do**

  Compute the log-partition function

$$Z_h = \log \sum_{x \in \mathcal{X}} \exp\left(\boldsymbol{\theta}_h^\top \boldsymbol{\phi}(x)\right)$$

  Objective function:

$$f(\boldsymbol{\theta}_h) = \boldsymbol{\theta}_h^\top \boldsymbol{\mu}_h - Z_h + \epsilon \|\boldsymbol{\theta}_h\|_2$$

  Compute:

$$p_{\boldsymbol{\theta}_h}(x|h) = \exp\left(\boldsymbol{\theta}_h^\top \boldsymbol{\phi}(x) - Z_h\right)$$

$$E_{\sim \boldsymbol{\theta}_h}[\boldsymbol{\phi}(x)|h] = \sum_{x \in \mathcal{X}} p_{\boldsymbol{\theta}_h}(x) \boldsymbol{\phi}(x)$$

  Gradient:

$$\nabla f(\boldsymbol{\theta}_h) \leftarrow \boldsymbol{\mu}_h - \mathbb{E}_{\sim \boldsymbol{\theta}_h}[\boldsymbol{\phi}(x)|h] + \epsilon \boldsymbol{\theta}_h \|\boldsymbol{\theta} + h\|_2^{-1}$$

  Update $\boldsymbol{\theta}_h$ using L-BFGS.

Return: $\boldsymbol{\theta}_h^*$.

3.5 *Method Improvements*

In this section we detail three improvements to our moment-based method that had a practical impact.

Supervised Regularization. We add a supervised penalty term to Eq. 3.23 to keep the label posteriors γ_j close to the label posteriors estimated in the labeled set, γ'_j, for every feature $j \in [W]$. The regularized least-squares problem becomes:

$$\begin{aligned} &\min_{\gamma_j} (1-\lambda)\|\mathbf{q}_j - \gamma_j^\top R\|^2 + \lambda \|\gamma_j - \gamma'_j\|^2 \\ &\text{s.t. } \mathbf{1}^\top \gamma_j = 1. \end{aligned} \tag{3.27}$$

CCA Projections. A one-hot feature representation of words and contexts has the disadvantage that it grows with the vocabulary size, making the moment matrix Q too sparse. The number of contextual features and words can grow rapidly on large text corpora. Similarly to Cohen et al. (2014) and Dhillon et al. (2015b), we use canonical correlation analysis (CCA) to reduce the dimension of these vectors. We use CCA to form low-dimensional projection matrices for features of words $P_W \in \mathbb{R}^{W \times D}$ and features of contexts $P_C \in \mathbb{R}^{C \times D}$, with $D \ll \min\{W, C\}$. We use these projections on the original feature vectors and replace the these vectors with their projections.

Selecting Anchors. Algorithms for identifying $\mathcal{A}(h)$ from unlabeled data alone were proposed by Arora et al. (2013) and Zhou et al. (2014). We propose instead to recover anchors from a small annotated corpus.

We collect counts of each word-label pair, and select up to 500 anchors with high conditional probability on the anchoring state $\widehat{p}(h \mid w)$. We tuned the probability threshold to select the anchors on the validation set, using steps of 0.1 in the unit interval, and making sure that all tags have at least one anchor. We also considered a frequency threshold, constraining anchors to occur more than 500 times in the unlabeled corpus, and four times in the labeled corpus. Note that past work used a single anchor word per state (i.e., $|\mathcal{A}(h)| = 1$). The anchor assumption is quite strict, and is usually not satisfied in its most strict sense. We found that much better results are obtained when $|\mathcal{A}(h)| \gg 1$, as choosing more anchors increases the number of samples used to estimate the context-label moment matrix $\widehat{R}$, reducing noise. However, increasing the number of anchors makes it harder to find pure anchors $p(h \mid \mathcal{A}(h)) = 1$. In the following experimental sections, we observe that this trade-off helps empirically.

3.6 *Experiments*

We evaluated our method on two tasks: POS tagging of Twitter text (in English), and POS tagging for a low-resource language (Malagasy). For all the experiments, we used the universal POS tagset (Petrov et al., 2011), which consists of $K = 12$ tags. We compared our method against supervised baselines (HMM and FHMM), which use the labeled data only, and two semi-supervised baselines that exploit the unlabeled data: self-training and EM. For the Twitter experiments, we also evaluated a stacked architecture in which we derived features from our model's predictions to improve a state-of-the-art POS tagger (MEMM).[7]

[7] http://www.ark.cs.cmu.edu/TweetNLP/

3.6.1 *Twitter POS Tagging*

For the Twitter experiment, we used the *Oct27* dataset of Gimpel et al. (2011), with the provided partitions (1,000 tweets for training and 328 for validation), and tested on the *Daily547* dataset (547 tweets). Anchor words were selected from the training partition as described in §3.5. We used 2.7M unlabeled tweets (O'Connor et al., 2010) to train the semi-supervised methods, filtering the English tweets as in Lui et al. (2012), tokenizing them as in Owoputi et al. (2013), and normalizing at-mentions, URLs, and emoticons.

We used as word features $\boldsymbol{\phi}(X)$ the word iself, as well as binary features for capitalization, titles, and digits (Berg-Kirkpatrick et al., 2010), the word shape, and the Unicode class of each character. Similarly to Owoputi et al. (2013), we also used suffixes and prefixes (up to length 3), and Twitter-specific features: whether the word starts with @, #,

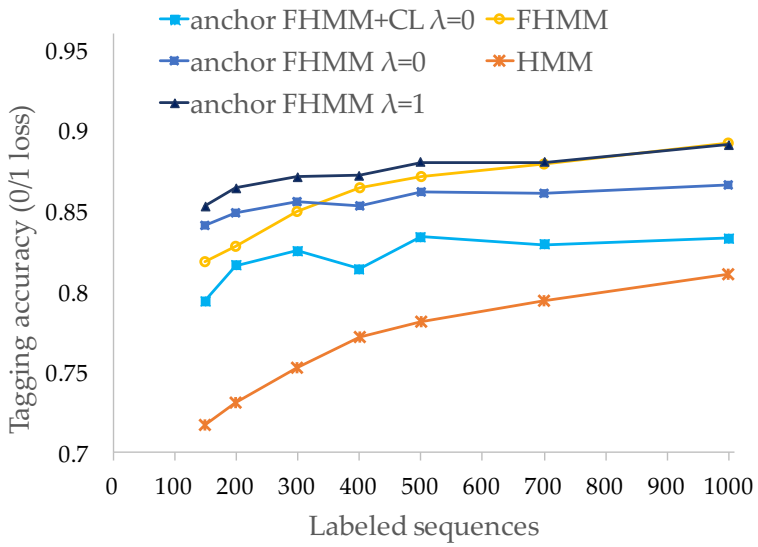


Figure 3.3: POS tagging accuracy in the Twitter data versus the number of labeled training sequences.

or *http://*. As contextual features $\boldsymbol{\psi}(Z)$, we derive analogous features for the preceding and following words, before reducing dimensionality with CCA. We collect feature expectations for words and contexts that occur more than 20 times in the unlabeled corpus. We tuned hyperparameters on the development set: the supervised interpolation coefficient in Eq. 3.27, $\lambda \in \{0, 0.1, \ldots, \underline{1.0}\}$, and, for all systems, the regularization coefficient $\epsilon \in \{\underline{0.0001}, 0.001, 0.01, 0.1, 1, 10\}$. [8] The former controls how much we rely on the supervised vs. unsupervised estimates. For $\lambda = 1.0$ we used supervised estimates only for words that occur in the labeled corpus; all the remaining words rely solely on unsupervised estimates.

[8] Underlines indicate selected values.

Varying supervision. Figure 3.3 compares the learning curves of our anchor-word method for the FHMM with the supervised baselines. We show the performance of the anchor methods without interpolation ($\lambda = 0$), and with supervised interpolation coefficient ($\lambda = 1$). When the amount of supervision is small, our method with and without interpolation outperforms all the supervised baselines. This improvement is gradually attenuated when more labeled sequences are used, with the supervised FHMM catching up when the full labeled dataset is used. The best model $\lambda = 1.0$ relies on supervised estimates for words that occur in the labeled corpus, and on anchor estimates for words that occur only in the unlabeled corpus. The unregularized model $\lambda = 0.0$ relies solely on unsupervised estimates given the set of anchors.

Semi-supervised comparison. Next, we compare our method to two other semi-supervised baselines, using both HMMs and FHMMs: EM and

	HMM		FHMM	
Models / #sequences	150	1000	150	1000
Supervised baseline				
HMM	71.7	81.1	81.8	89.1
Semi-supervised baselines				
EM	77.2	83.1	81.8	89.1
self-training	78.2	86.1	83.4	**89.4**
Anchor Models				
anchors, $\lambda = 0.0$	83.0	85.5	84.1	86.7
anchors, $\lambda = 1.0$	**84.3**	**88.0**	**85.3**	89.1

Table 3.1: Tagging accuracies on Twitter. Shown are the supervised and semi-supervised baselines, and our moment-based method, trained with 150 training labeled sequences, and the full labeled corpus (1000 sequences).

self-training. EM requires decoding and counting in multiple passes over the full unlabeled corpus. We initialized the parameters with the supervised estimates, and selected the iteration with the best accuracy on the development set.The FHMM with EM did not perform better than the supervised baseline, so we consider the initial value as the best accuracy under this model. The self-training baseline uses the supervised system to tag the unlabeled data, and then retrains on all the data.

Results are shown in Table 3.1. According to a word-level paired Kolmogorov-Smirnov test, for the FHMM with 1,000 tweets, the self-training method outperforms the other methods with statistical significance at $p < 0.01$; and for the FHMM with 150 tweets the anchor-based and self-training methods outperform the other baselines with the same p-value. Our best HMM outperforms the other baselines at a significance level of $p < 0.01$ for 150 and 1000 sequences. We observe that, for small amounts of labeled data (150 tweets), our method outperforms all the supervised and semi-supervised baselines, yielding accuracies 6.1 points above the best semi-supervised baseline for a simple HMM, and 1.9 points above for the FHMM. With more labeled data (1,000 instances), our method outperforms all the baselines for the HMM, but not with the more sophisticated FHMM, in which our accuracies are 0.3 points below the self-training method.These results suggest that our method is more effective when the amount of labeled data is small. We further study the effect of different learning algorithms for the transition model: estimated from a composite likelihood (CL) and a maximum likelihood (ML) approach in Table 3.2. We report results on the best emission model (FHMM) with and without supervised interpolation ($\lambda = 0, \lambda = 1$).

Models / #sequences	150	1000
ML ($\lambda = 1$)	85.3	89.1
CL ($\lambda = 1$)	79.3	85.5
CL ($\lambda = 0$)[9]	77.6	84.0

Table 3.2: Anchor learning with transition model estimated from two methods ML and CL in (§ 3.3.5) with the best FHMM anchor emissions.

[9] This approach is completely unsupervised assuming we know the set of anchor words. Anchors can also be recovered from an unsupervised approach in (Arora et al., 2013).

Models / #sequences	150	1000
MEMM (same+clusters)	89.57	**93.36**
MEMM (same+clusters+posteriors)	**91.14**	93.18
MEMM (all+clusters)	91.55	**94.17**
MEMM (all+clusters+posteriors)	**92.06**	94.11

Table 3.3: Tagging accuracy for the MEMM POS tagger of Owoputi et al. (2013) with additional features from our model's posteriors.

Stacking features. We also evaluated a stacked architecture in which we use our model's predictions as an additional feature to improve the state-of-the-art Twitter POS tagger of Owoputi et al. (2013). This system is based on a semi-supervised discriminative model with Brown cluster features (Brown et al., 1992). We provide results using their full set of features (*all*), and using the same set of features in our anchor model (*same*). We compare tagging accuracy on a model with these features plus Brown clusters (+*clusters*) against a model that also incorporates the posteriors from the anchor method as an additional feature in the MEMM (+*clusters+posteriors*). The results in Table 3.3 show that using our model's posteriors are beneficial in the small labeled case, but not if the entire labeled data is used.

Runtime comparison. The training time of anchor FHMM is 3.8h (hours) and 4h with composite likelihood transitions, for self-training HMM 10.3h, for EM HMM 14.9h and for Twitter MEMM (all+clusters) 42h. As such, the anchor method is much more efficient than all the baselines because it requires a single pass over the corpus to collect the moment statistics, followed by the QPs, without the need to decode the unlabeled data. EM and the Brown clustering method (the latter used to extract features for the Twitter MEMM) require several passes over the data; and the self-training method involves decoding the full unlabeled corpus, which is expensive when the corpus is large. Our analysis adds to previous evidence that spectral methods are more scalable than learning algorithms that require inference (Parikh et al., 2012b; Cohen et al., 2013c).

3.6.2 *Malagasy POS Tagging*

In this section, we provide results using a low resource language, Malagasy, for which there is a small amount of annotations. For the Malagasy experiment, we used the small labeled dataset from Garrette et al. (2013), which consists of 176 sentences and 4,230 tokens. We also make use of their tag dictionaries with 2,773 types and 23 tags, and their unlabeled data (43.6K sequences, 777K tokens). We converted all the original POS tags to universal tags using the mapping proposed in Garrette et al. (2013).

Table 3.4 compares our method with semi-supervised EM and self-

Models	Accuracies
supervised FHMM	90.5
EM FHMM	90.5
self-training FHMM	88.7
anchors FHMM (token), λ=1.0	89.4
anchors FHMM (type+token), λ=1.0	**90.9**

Table 3.4: Tagging accuracies for the Malagasy dataset.

training, for the FHMM. Note that the accuracies are not directly comparable to Garrette et al. (2013), who use a different tag set. However, our supervised baseline trained on those tags is already superior to the best semi-supervised system in Garrette et al. (2013), as we get 86.9% against the 81.2% reported in Garrette et al. (2013) using their tagset. We tested two supervision settings: token only, and type+token annotations, analogous to Garrette et al. (2013). The anchor method outperformed the baselines when both type and token annotations were used to build the set of anchor words.In this task our method took approximately 5 minutes to train.

3.7 *Conclusions*

In this chapter, we introduced an efficient semi-supervised sequence labeling method using a generative log-linear model. We use contextual information from a set of *anchor* observations to disambiguate state, and build a weakly supervised method from this set. The proposed method outperforms other supervised and semi-supervised methods, with small supervision in POS-tagging for Malagasy, a scarcely annotated language, and for Twitter. The anchor method is most competitive for learning with large amounts of unlabeled data, under weak supervision, while training an order of magnitude faster than any of the baselines.

xs

4
Planning with Kernel Methods

In this chapter, we combine optimization based planning with kernel methods. We introduce a functional gradient descent trajectory optimization algorithm for robot motion planning in Reproducing Kernel Hilbert Spaces (RKHSs). This work generalizes functional gradient trajectory optimization by formulating it as minimization of a cost functional in an RKHS. This generalization lets us represent trajectories as linear combinations of kernel functions, without any need for waypoints. As a result, we are able to take larger steps and achieve a locally optimal trajectory in just a few iterations.

In Section 4.2, we define trajectories in RKHSs, Section 4.3 describes the optimization algorithm, Section 4.4 demonstrates the benefits of optimizing under RKHSs norm as an inherently efficient space, Section 4.6 describes different forms of representing the obstacle avoidance functional and Section 4.7 provides an empirical analysis of the effectiveness of the planner for different kernels, including Gaussian RBFs, Laplacian RBFs, and B-splines, as compared to the standard discretized waypoint representation.

4.1 Motivation

Motion planning is an important component of robotics: it ensures that robots are able to safely move from a start to a goal configuration without colliding with obstacles. *Trajectory optimizers* for motion planning focus on finding feasible configuration-space trajectories that are also efficient—e.g., approximately locally optimal for some cost function. Many trajectory optimizers have demonstrated great success in a number of high-dimensional real-world problems (Quinlan et al., 1993; Schulman et al., 2013; Todorov et al., 2005; Berg et al., 2011). Often, they work by defining a cost functional over an infinite-dimensional Hilbert space of trajectories, then taking steps down the functional gradient of cost to search for smooth, collision-free trajectories (Zucker et al., 2013; Ratliff et al., 2009).

In this work, we exploit the same functional gradient approach, but with a novel approach to trajectory representation. While previous algorithms are derived for trajectories in Hilbert spaces in theory, *in practice* they commit to a finite parametrization of trajectories in order to instantiate a gradient update (Zucker et al., 2013; Park et al., 2012; Kalakrishnan et al., 2011)—typically a large but finite list of discretized waypoints. The number of waypoints is a parameter that trades off between computational complexity and trajectory expressiveness. Our work frees the optimizer from a discrete parametrization, enabling it to perform gradient descent on a much more general trajectory parametrization: reproducing-kernel Hilbert spaces (RKHSs) (Scholkopf et al., 2001; Kimeldorf et al., 1971; Aronszajn, 1950), of which waypoint parametrizations are merely one instance. RKHSs impose just enough structure on generic Hilbert spaces to enable a concrete and implementable gradient update rule, while leaving the choice of parametrization flexible: different kernels lead to different geometries (§ 4.2).

Our contribution is two-fold. Our *theoretical* contribution is the formulation of functional gradient descent motion planning in RKHSs, as the minimization of a cost functional regularized by the RKHS norm (§ 4.3). Regularizing by the RKHS norm is a common way to ensure smoothness in function approximation (Hofmann et al., 2008), and we apply the same idea to trajectory parametrization. By choosing the RKHS appropriately, the trajectory norm can quantify different forms of smoothness or efficiency, such as preferring small values for any n-th order derivative (Yuan et al., 2010). So, RKHS norm regularization can be tuned to prefer trajectories that are smooth with, for example, low velocity, acceleration, or jerk (§ 4.4) (Rawlik et al., 2013).

Our *practical* contribution is an algorithm for very efficient motion planning in inherently smooth trajectory space with low-dimensional parametrizations. Unlike discretized parametrizations, which require many waypoints to produce smooth trajectories, our algorithm can represent and search for smooth trajectories with only a few point evaluations. The inherent smoothness of our trajectory space also increases efficiency; our parametrization allows the optimizer to take large steps at every iteration without violating trajectory smoothness, therefore converging to a collision-free and high-quality trajectory faster than competing approaches.

Our experiments demonstrate the effectiveness of planning in RKHSs using synthetic 2D environment, with a 3-DOF planar arm, and using more complex scenarios, with a 7-DOF robotic arm. We show how different choices of kernels yield different preferences over trajectories. We further introduce reproducing kernels that represent interactions among joints. Sections 4.6 and 4.7 illustrate these advantages of

RKHSs, and compare different choices of kernels.

4.2 *Trajectories in Reproducing Kernel Hilbert Spaces*

A trajectory is a function $\boldsymbol{\xi} \in \mathcal{H}$ such that $\mathcal{H} \subseteq \{\xi : [0,1] \to \mathcal{C}\}$ defines a space of functions that map time $t \in [0,1]$ to robot configurations $\boldsymbol{\xi}(t) \in \mathcal{C} \equiv \mathbb{R}^D$. Note that $\boldsymbol{\xi} \in \mathcal{H}$ is a trajectory, a vector valued function in the RKHS, while $\boldsymbol{\xi}(t) \in \mathcal{C}$ is a trajectory evaluation at a single time point corresponding to a robot configuration. We can treat a set of trajectories as a Hilbert space by defining vector-space operations such as addition and scalar multiplication of trajectories (Kreyszig, 1978).

In this work, we restrict to trajectories in *Reproducing Kernel Hilbert Spaces*. We can upgrade our Hilbert space to an RKHS $\mathcal{H}$ by assuming additional structure: for any $\boldsymbol{y} \in \mathcal{C}$ and $t \in [0,1]$, the functional $\boldsymbol{\xi} \mapsto \boldsymbol{y}^\top \boldsymbol{\xi}(t)$ must be continuous (Scholkopf et al., 2001; Wahba, 1999; Ratliff et al., 2007). Note that, since the configuration space is typically multidimensional ($D > 1$), our trajectories form an RKHS of *vector-valued* functions (Micchelli et al., 2005), defined by the above property. Vector-valued RKHSs describe a direct approach to model different views, or dimensions of the same input data. In contrast, real-valued RKHS could describe the same input information using different independent real-valued functions. However, it is often useful to consider the interactions of these different views/dimensions when learning the mapping from input to output data, and one of these cases is the problem studied in this chapter, of robot pose modeling.

Depending on whether the chosen RKHS describes real or vector-valued functions, we may select a corresponding vector or matrix valued kernel, depending on whether we are interested in modeling a single dimension or their dependencies, respectively. For instance, in a robot setting we are interested in reproducing kernels with continuous properties such as C^∞ spaces whose velocities, acceleration and higher order derivatives are also continuous, such as the Gaussian RBF kernel.

The *reproducing kernel* associated with a vector valued RKHS becomes a matrix valued kernel $K : [0,1] \times [0,1] \to \mathcal{C} \times \mathcal{C}$. Eq. 4.1 represents the kernel matrix for two different time instances:

$$K(t,t') = \begin{bmatrix} k_{1,1}(t,t') & k_{1,2}(t,t') & \dots & k_{1,D}(t,t') \\ k_{2,1}(t,t') & k_{2,2}(t,t') & \dots & k_{2,D}(t,t') \\ \vdots & \ddots & & \vdots \\ k_{D,1}(t,t') & k_{D,2}(t,t') & \dots & k_{D,D}(t,t') \end{bmatrix} \tag{4.1}$$

This matrix has a very intuitive physical interpretation. Each element in Eq. 4.1, $k_{d,d'}(t,t')$ tells us how joint $[\xi(t)]_d$ at time t affects the motion of joint $[\xi(t')]_{d'}$ at t', *i.e.* its degree of correlation or similarity between

the two (joint,time) pairs. In practice, off-diagonal terms of Eq. 4.1 will not be zero, hence perturbations of a given joint d propagate through time, as well as through the rest of the joints. The norm and inner product defined in a vector-valued RKHS can be written in terms of the kernel matrix, via the reproducing property: trajectory evaluation can be represented as an inner product of the vector valued functions in the RKHS, as described in Eq. 4.2 below. For any configuration $\boldsymbol{y} \in \mathcal{C}$, and time $t \in [0,1]$, we get the inner product of $\boldsymbol{y}$ with the trajectory in the vector-valued RKHS evaluated at time t:

$$\boldsymbol{y}^\top \boldsymbol{\xi}(t) = \langle \boldsymbol{\xi}, K(t,\cdot)\boldsymbol{y} \rangle_{\mathcal{H}}, \quad \forall \boldsymbol{y} \in \mathcal{C} \tag{4.2}$$

In our planning algorithm we will represent a trajectory in the RKHS in terms of some finite support $\{t_i\}_{i=1}^N \in \mathcal{T}$. This set grows adaptively as we pick more points to represent the final trajectory. At each step our trajectory will be a linear combination of functions in the RKHS, each indexed by a time-point in $\mathcal{T}$, see Figure 4.1:

$$\boldsymbol{y}^\top \boldsymbol{\xi}(t) = \sum_{t_i \in \mathcal{T}} \boldsymbol{y}^\top K(t,t_i)\boldsymbol{a_i} \tag{4.3}$$

for $t, t_i \in [0,1]$, and $\boldsymbol{a_i} \in \mathcal{C}$. If we consider the configuration vector $\boldsymbol{y} \equiv \boldsymbol{e}_d$ to be the indicator of joint d, then we can capture its evolution over time as: $[\xi(t)]_d = \sum_i \boldsymbol{e}_d^\top K(t,t_i)\boldsymbol{a_i}$, taking into account the effect of all other joints. The inner product in $\mathcal{H}$ of functions $\boldsymbol{y}^\top \boldsymbol{\xi}^1(t) = \sum_i \boldsymbol{y}^\top K(t,t_i)\boldsymbol{a}_i$ and $\boldsymbol{y}^\top \boldsymbol{\xi}^2(t) = \sum_j \boldsymbol{y}^\top K(t,t_j)\boldsymbol{b}_j$, for $\boldsymbol{y}, \boldsymbol{a}_i, \boldsymbol{b}_j \in \mathcal{C}$ is defined as:

$$\langle \boldsymbol{\xi}^1, \boldsymbol{\xi}^2 \rangle_{\mathcal{H}} = \sum_{i,j} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{b}_j \tag{4.4}$$

$$\|\boldsymbol{\xi}\|_{\mathcal{H}}^2 = \langle \boldsymbol{\xi}, \boldsymbol{\xi} \rangle = \sum_{i,j} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{a}_j \tag{4.5}$$

For example, in the Gaussian RBF RKHS (with kernel $k_{d,d'}(t,t') = \exp(\|t-t'\|^2/2\sigma^2)$, when $d' = d$ and 0 otherwise), a trajectory is a weighted sum of radial basis functions:

$$\boldsymbol{\xi}(t) = \sum_{d,i} a_{i,d} \exp\left(\frac{\|t-t_i\|^2}{2\sigma^2}\right) \boldsymbol{e}_d, \quad a_{i,d} \in \mathbb{R} \tag{4.6}$$

The coefficients $a_{i,d}$ assess how important a particular joint d is to the overall trajectory, at time t_i. They can be interpreted as weights of local perturbations to the motions of different joints centered at different times. Interactions among joints, as described in Eq. 4.1, can be represented for instance using a kernel matrix of the form :

$$K(t,t') = \exp\left(\frac{\|t-t'\|^2}{2\sigma^2}\right) \boldsymbol{J}^\top \boldsymbol{J} \in \mathbb{R}^{D \times D} \tag{4.7}$$

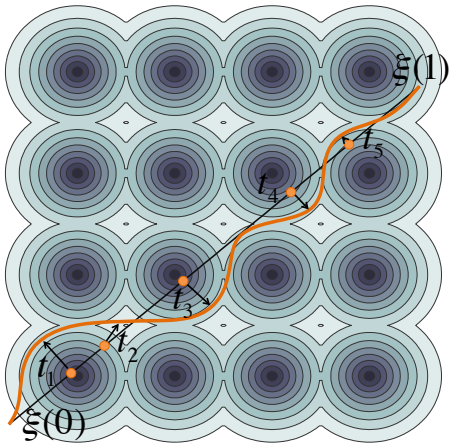


Figure 4.1: Trajectory in RKHS as linear combination of kernel functions. At each iteration, the optimizer takes the current trajectory (black) and identifies the point of maximum obstacle cost t_i (orange points). It then updates the trajectory by a point evaluation function centered around t_i. Grey regions depict isocontours of the obstacle cost field (darker means closest to obstacles, higher cost). Black arrows show reduce operation with 5 max-cost points.

where $\boldsymbol{J} \in \mathbb{R}^{3\times D}$ represents the workspace Jacobian matrix at a fixed configuration. This strategy changes the RKHS metric in configuration space according to the robot Jacobian in the workspace. This norm can be interpreted as an approximation of the velocity of the robot in workspace (Ratliff et al., 2015). The trajectory norm measures the size of the perturbations, and the correlation among them, quantifying how complex the trajectory is in the RKHS, see Section 4.4. Different norms can be considered for representing the RKHS; this can leverage more problem-specific information, which could reduce the number of iterations required to find a low cost trajectory.

4.3 *Motion Planning in an RKHS*

In this section, we describe how trajectory optimization can be achieved by functional gradient descent in an RKHS of trajectories.

4.3.1 *Cost Functional*

We introduce a cost functional $\mathcal{U} : \mathcal{H} \to \mathbb{R}$ that maps each trajectory to a scalar cost. This functional quantifies the quality of a given a trajectory (function in the RKHS). $\mathcal{U}$ trades off between a regularization term that measures the efficiency of the trajectory, and an obstacle term that measures its proximity to obstacles:

$$\mathcal{U}[\boldsymbol{\xi}] = \mathcal{U}_{obs}[\boldsymbol{\xi}] + \frac{\beta}{2}\|\boldsymbol{\xi}\|^2_{\mathcal{H}} \tag{4.8}$$

As described in Section 4.4, we choose our RKHS so that the regularization term encodes our desired notion of smoothness or trajectory efficiency—*e.g.*, minimum length, velocity, acceleration, jerk. The obstacle cost functional is defined on trajectories in configuration space, but obstacles are defined in the robot's workspace $\mathcal{W} \equiv \mathbb{R}^3$. So, we connect configuration space to workspace via a *forward kinematics* map x: if $\mathcal{B}$ is the set of body points of the robot, then $x : \mathcal{C} \times \mathcal{B} \to \mathcal{W}$ tells us the workspace coordinates of a given body point when the robot is in a given configuration in Figure 4.2. We can then decompose the obstacle cost functional as:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] \equiv \underset{t,u}{\text{reduce}}\, c\left(x(\boldsymbol{\xi}(t), u)\right) \tag{4.9}$$

where reduce is an operator that aggregates costs over the entire trajectory and robot body—e.g., a maximum or an integral, see Section 4.6. We assume that the reduce operator takes (at least approximately) the form of a sum over some finite set of (time, body point) pairs $\mathcal{T}(\boldsymbol{\xi})$:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] \approx \sum_{(t,u)\in\mathcal{T}(\boldsymbol{\xi})} c\left(x(\boldsymbol{\xi}(t), u)\right) \tag{4.10}$$

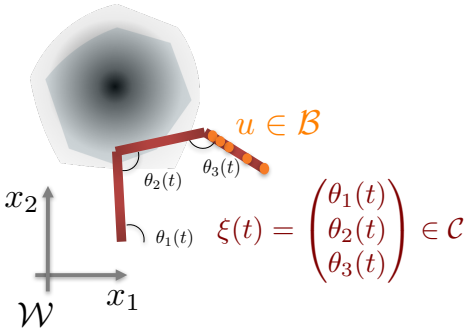


Figure 4.2: 3-link arm robot in 2D workspace. Robot configuration in red $\mathcal{C}$, robot points in orange $\mathcal{B}$. Grey regions depict the obstacle cost field in workspace $\mathcal{W}$ (darker means closest to obstacles, higher cost).

For example, the maximum operator takes this form: if (t, u) achieves the maximum, then $\mathcal{T}(\xi)$ is the singleton set $\{(t, u)\}$, in Figure 4.1. Integral operators do not take this form, but they can be well approximated in this form using quadrature rules, see Section 4.6.2.

4.3.2 *Optimization*

We can derive the functional gradient update by minimizing a local linear approximation of $\mathcal{U}$ in Eq. 4.8:

$$\xi^{n+1} = \arg\min_{\xi} \ \langle \xi - \xi^n, \nabla \mathcal{U}[\xi^n] \rangle_{\mathcal{H}} + \frac{\lambda}{2} \|\xi - \xi^n\|^2_{\mathcal{H}} \tag{4.11}$$

The quadratic term is based on the RKHS norm, meaning that we prefer "smooth" updates, analogous to Zucker et al. (2013).

This minimization admits a solution in closed form:

$$\xi^{n+1}(\cdot) = \left(1 - \frac{\beta}{\lambda}\right) \xi^n(\cdot) - \frac{1}{\lambda} \nabla \mathcal{U}_{obs}[\xi^n](\cdot) \tag{4.12}$$

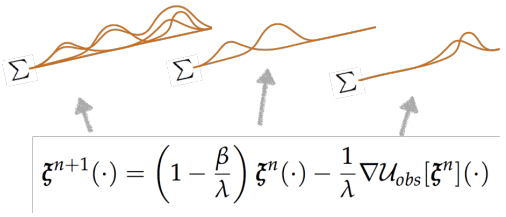


Figure 4.3: Iterative trajectory update with gradient descent. Each term is a sum of kernel functions in the RKHS.

Since we have assumed that the cost functional $\mathcal{U}_{obs}[\xi]$ depends only on a finite set of points $\mathcal{T}(\xi)$ (Eq. 4.10), it is straightforward to show that the functional gradient update has a finite representation (so that the overall trajectory, which is a sum of such updates, also has a finite representation, in Figure 4.3). In particular, assume the workspace cost field c and the forward kinematics function x are differentiable; then we can obtain the cost functional gradient by the chain rule (Ratliff et al., 2007; Scholkopf et al., 2001):

$$\nabla \mathcal{U}_{obs}(\cdot) = \sum_{(t,u) \in \mathcal{T}} K(\cdot, t) \, \mathbf{J}^\top(t, u) \nabla c(x(\xi(t), u)) \tag{4.13}$$

where $\mathbf{J}(t, u) = \frac{\partial}{\partial \xi(t)} x(\xi(t), u) \in \mathbb{R}^{3 \times D}$ is the workspace Jacobian matrix at time t for body point u, and the kernel function $K(\cdot, t)$ is the gradient of $\xi(t)$ with respect to ξ. The kernel matrix is defined in Eq. 4.1.

This solution is a generic form of functional gradient optimization with a *directly instantiable* obstacle gradient that does not depend on a predetermined set of waypoints, offering a more expressive representation with fewer parameters. We derive a constrained optimization update rule, by solving the KKT conditions for a vector of Lagrange multipliers, see Section 4.3.3. The full method is summarized as Algorithm 7.

4.3.3 *Constrained optimization*

Consider equality constraints (fixed start and goal configurations) and inequality constraints (joint limits) on the trajectory $h(\xi(t)) = 0, g(\xi(t)) \leq$

Algorithm 7: Trajectory optimization in RKHSs

Input: $\left(N, c, \nabla c, \xi^{(n)}(0), \xi^{(n)}(1)\right)$

1: **for all** each joint angle $d \in D$ **do**
2: Initialize to a straight line trajectory $\xi_d^0(t) = \xi_d(0) + (\xi_d(1) - \xi_d(0))t$.
3: **while** ($\mathcal{U}[\xi^n] > \epsilon$ and $n < N_{\text{MAX}}$) **do**
4: Compute $\mathcal{U}_{obs}[\xi^n]$ Eq. 4.13.
5: Find the support of time/body points $\mathcal{T}(\xi) = \{t_i, u_i\}, i = 1, \ldots, N$ Eq. 4.10.
6: **for all** $(t_i, u_i)_{i=1}^N \in \mathcal{T}(\xi)$ **do**
7: Evaluate the cost gradient $\nabla c(\xi(t_i), u_i)$ and Jacobian $\mathbf{J}(t_i, u_i)$
8: Update trajectory: $\xi^{n+1} = (1 - \frac{\beta}{\lambda})\xi^n - \frac{1}{\lambda} \sum_{(t,u)\in\mathcal{T}} K(\cdot, t)\, \mathbf{J}^\top(t, u) \nabla c(x(\xi(t), u))$
9: If constraints are present, project onto constraint set Eq. 4.16.
10: **Return:** Final trajectory ξ^* and costs $\|\xi^*\|^2_{\mathcal{H}}, \mathcal{U}_{obs}[\xi^*]$.

0, respectively. We write them as inner products with kernel functions in the RKHS Eq. 4.14. For any configuration $\boldsymbol{y} \in \mathcal{C}$, $\boldsymbol{q_o}, \boldsymbol{q_p} \in \mathcal{C}$ and $t_o = \{0, 1\}$, $t_p = [0, 1]$

$$h(\cdot)^\top \boldsymbol{y} = \langle \xi, K(t_o, \cdot)\boldsymbol{y} \rangle_{\mathcal{H}} - \boldsymbol{q_o}^\top \boldsymbol{y} = 0, \tag{4.14}$$

$$g(\cdot)^\top \boldsymbol{y} = \langle \xi, K(t_p, \cdot)\boldsymbol{y} \rangle_{\mathcal{H}} - \boldsymbol{q_p}^\top \boldsymbol{y} \leq 0. \tag{4.15}$$

Let, γ^o, $\mu^p \in \mathbb{R}^D$ be the concatenation of all equality (γ^o) and inequality (μ^p) Lagrange multipliers. We rewrite the objective function in Eq. 4.11 including joint constraints:

$$\xi^{n+1}(\cdot) = \arg\min_{\xi} \; \langle \xi - \xi^n, \nabla \mathcal{U}[\xi^n] \rangle_{\mathcal{H}} + \frac{\lambda}{2} \|\xi - \xi^n\|^2_{\mathcal{H}} + \gamma^{o\top} h[\xi] + \mu^{p\top} g[\xi] \tag{4.16}$$

Solving the KKT system for the stationary point of Eq. 4.16 (ξ, γ^o, μ^p) with $\mu^p \geq 0$, we obtain the constrained solution in Eq. 4.17. Let $\mathrm{d}c_j \equiv \boldsymbol{J}^\top(t_j, u_j) \nabla c \left(x(\xi^n(t_j), u_j)\right) \in \mathbb{R}^D$. The full update rule becomes:

$$\xi^*(\cdot) = \left(1 - \frac{\beta}{\lambda}\right) \xi^n(\cdot) - \frac{1}{\lambda} \left(\sum_{t_j \in \mathcal{T}} K(\cdot, t_j) \mathrm{d}c_j + K(\cdot, t_o)\gamma o + K(\cdot, t_p)\boldsymbol{\mu}^p \right) \tag{4.17}$$

This constrained solution ends up augmenting the finite support set ($\mathcal{T}$) with points that violate the constraints, weighted by the respective Lagrange multipliers. Each of the multipliers can be interpreted as a quantification of how much the points t_o or t_p affect the overall trajectory over time and joint space.

Efficient Constraint Update

We consider the sets of max points in the support set $\mathcal{T}$, equality constraint violation points (endpoints) T^o and inequality violation points

(joint limits) T^p. We recover the Lagrange multipliers in Eq. 4.17 by solving Eq. 4.18.

$$\begin{bmatrix} K(T^o,T^o) & K(T^o,T^p) \\ K(T^o,T^p)^\top & K(T^p,T^p) \end{bmatrix} \begin{bmatrix} \boldsymbol{\gamma}^o \\ \boldsymbol{\mu}^p \end{bmatrix} \tag{4.18}$$
$$= \begin{bmatrix} (\lambda-\beta)\,\boldsymbol{\xi}^n(T^o) + \mathrm{d}c_j^\top K(\mathcal{T},T^o) - \boldsymbol{q_o} \\ (\lambda-\beta)\,\boldsymbol{\xi}^n(T^p) + \mathrm{d}c_j^\top K(\mathcal{T},T^p) - \boldsymbol{q_p} \end{bmatrix} \equiv \begin{bmatrix} h(T^o) \\ g(T^p) \end{bmatrix}$$

To solve the system efficiently we make use of the reproducing property $K(T^o,T^o) = K(T^o,T^p)K(T^p,T^p)^{-1}K(T^o,T^p)^\top$, and the fact that we use use a separable kernel $K(t_i,t_j) = k(t_i,t_j)\otimes J^\top J$.

$$\begin{aligned} K(T^p,T^p)^{-1} &= k(T^p,T^p)^{-1}\otimes(J^\top J)^{-1} \\ \boldsymbol{\mu^p} &= K(T^p,T^p)^{-1}g(T^p) \\ \boldsymbol{\gamma^o} &= (T^o,T^o)^{-1}\left(h(T^o) - K(T^o,T^p)\boldsymbol{\mu}^p\right) \end{aligned} \tag{4.19}$$

4.4 *Trajectory Efficiency as Norm Encoding in RKHS*

In different applications, it is useful to consider different notions of trajectory efficiency or smoothness. We can do so by choosing appropriate kernel functions that have the desired property, and consequently desired induced norm/metric. In Figure 4.4 we empirically show how the kernel choice can impact on the resulting trajectory, we demonstrate how after just one update different kernels yield different trajectories. Later we study this effect on more complex scenarios. For instance, we can choose a kernel according to the topology of the obstacle field, or we can learn a kernel from demonstrations or user input, bringing problem specific information into the planning problem. This can help improve efficiency of the planner. Another possibility is to tune the resolution of the kernel via its width. We could build a kernel with adaptive width according to the environment, *i.e.*, higher sensitivity (lower width) in cluttered scenarios.

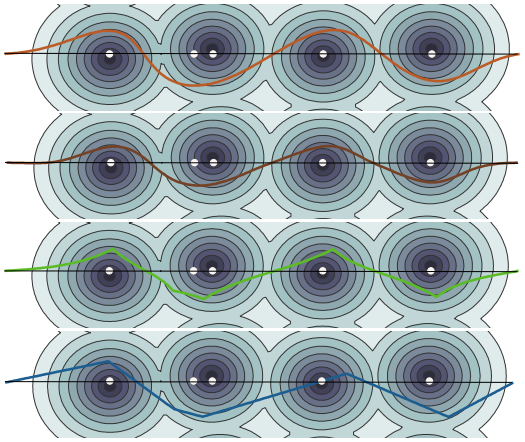


Figure 4.4: 2D trajectory with large steps (1 it. 5 max-cost points in white) Trajectory profile using different kernels in order (top-down): Gaussian RBF, B-splines, Laplacian RBF kernels, and waypoints.

Additionally, it is often desirable to penalize the velocity, acceleration, jerk, or other derivatives of a trajectory instead of (or in addition to) its magnitude. To do so, we can take advantage of a *derivative reproducing property*: let $\mathcal{H}_1$ be one of the coordinate RKHSs from our trajectory representation, with kernel k. If k has sufficiently many continuous derivatives, then for each partial derivative operator D^α, there exist representers $(D^\alpha k)_t \in \mathcal{H}_1$ such that, for all $f\in\mathcal{H}_1$, $(D^\alpha f)(t) = \langle (D^\alpha k)_t, f\rangle$ (Zhou, 2008, Theorem 1). (Here α is a multi-set of indices, indicating which partial derivative we are referring to.) We can, therefore, define a new RKHS with a norm that penalizes the partial derivative D^α: the kernel is $k^\alpha(t,t') = \langle (D^\alpha k)_t, (D^\alpha k)_{t'}\rangle$. If we use this RKHS norm as the smoothness penalty for the corresponding coordinate of

our trajectories, then our optimizer will automatically seek out trajectories with low velocity, acceleration, or jerk in this coordinate.

For example, consider an RBF kernel with a reproducing first order derivative: $D^1k(t,t_i) = D^1k_{t_i}[t] = \frac{(t-t_i)}{2\sigma^2}k(t,t_i)$ is the reproducing kernel for the velocity profile of a trajectory defined in an RBF kernel space $k(t,t_i) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\|t-t_i\|^2/2\sigma^2)$. The velocity profile of a trajectory $\boldsymbol{\xi}(t) = \sum_i \beta_i k(t,t_i)$ can be written as $D^1\boldsymbol{\xi}(t) = \sum_i \beta_i D^1k(t,t_i)$. The trajectory can be found by integrating $D^1\boldsymbol{\xi}(t)$ once and using the constraint $\boldsymbol{\xi}(0) = \boldsymbol{q}_i$.

$$\begin{aligned}\boldsymbol{\xi}(T) &= \int_0^T D^1\boldsymbol{\xi}(t)dt = \sum_i \beta_i \int_0^T \frac{(t-t_i)}{2\sigma^2}k(t,t_i)dt \\ &= \sum_i \beta_i \left[k(T,t_i) - k(0,t_i)\right] + \boldsymbol{q}_i\end{aligned} \tag{4.20}$$

The initial condition is verified automatically. The endpoint condition can be written as $\boldsymbol{q}_f = \sum_i \beta_i \left[k(1,t_i) - k(0,t_i)\right] + \boldsymbol{q}_i$; this imposes additional information over the coefficients $\beta_i \in \mathcal{C}$, which we can enforce during optimization.

Here, we explicitly consider only a space of first derivatives, but extensions to higher order derivatives can be derived similarly integrating p times to obtain the trajectory profile. Constraints over higher derivatives (up to order n), can be enforced using any constraint projection method, inside our gradient descent iteration. The update rule in this setting can be derived using the natural gradient in the space, where the new obstacle gradient becomes:

$$\nabla\mathcal{U}_{obs}(\cdot) = \sum_\alpha^n \sum_{(t,u)\in\mathcal{T}} D^\alpha K(\cdot,t)\, \mathbf{J}^\top(t,u)\nabla c(\, x(\boldsymbol{\xi}(t),u)\,) \tag{4.21}$$

Although Gaussian RBFs are a default choice of kernel in many kernel methods, RKHSs can also easily represent other types of kernel functions. For example, B-splines are a popular parametrization of smooth functions (Zhang et al., 1995; Pan et al., 1995; Blake et al., 1998), that are able to express smooth trajectories while avoiding obstacles, even though they are finite-dimensional kernels.

Regularization schemes in different RKHSs may encode different forms of trajectory efficiency. Here we proposed a different norm using derivative penalization. The choice of kernel should be application driven, and any reproducing kernel can easily be considered under the optimization framework presented here.

Other RKHS norms may be defined as sums, products, tensor product of kernels, or any closed kernel operation.

4.4.1 *Finite approximation of Path Integral Cost*

Trajectory optimization in RKHSs can be derived for different types of obstacle cost functionals, provided that trajectories have a finite representation.

Previous work defines an obstacle cost in terms of the arc-length integral of the trajectory (Zucker et al., 2013). We approximate the path integral cost functional with a finite representation using integral approximation methods, such as quadrature methods (Press et al., 1992). Consider a set of finite time points $t_i \in \mathcal{T}$ to be the abscissas of an integral approximation method. We use a Gauss-Legendre quadrature method, and represent t_i as roots of the Legendre polynomial $P_n(t)$ of degree n. Let w_i be the respective weights on each cost sample:

$$\mathcal{U}_{obs}[\boldsymbol{\xi}] = \int_0^1 c\left[\boldsymbol{\xi}(t)\right] \left\| D^1\boldsymbol{\xi}(t) \right\| dt \approx \sum_{t_i \in \mathcal{T}} \omega_i \, c\left[\boldsymbol{\xi}(t_i)\right] \left\| D^1\boldsymbol{\xi}(t_i) \right\| \tag{4.22}$$

with coefficients, and the Legendre polynomials obtained recursively from Rodrigues' Formula (Press et al., 1992):

$$P_n = 2^n \sum_{j=0}^{n} t^j \binom{n}{j} \binom{\frac{n+j-1}{2}}{n}$$

$$w_i = \frac{2}{\left(1 - t_i^2\right)\left[D^1 P_n(t_i)\right]^2}$$

We denote $D^1 \equiv \frac{d}{dt}$ the first order time derivative. Using this notation, we are able to work with integral functionals, using still a finite set of time points to represent the full trajectory.

4.5 *Kernel Metric in RKHS*

The norm provides a form of quantifying how complex a trajectory is in the space associated with the RKHS kernel metric K. The kernel metric is determined by the kernel functions we choose for the RKHS, as we have seen before (§ 4.4). Likewise, the set of time points $\mathcal{T}$ that support the trajectory contribute to the design of the kernel metric:

$$\begin{aligned} \|\boldsymbol{\xi}\|_{\mathcal{H}}^2 &= \sum_{d,d'} \sum_{t_i,t_j \in \mathcal{T}} a_{d,i} k_{d,d'}(t_i,t_j) a_{d',j} \\ &= \sum_{t_i,t_j \in \mathcal{T}} \boldsymbol{a}_i^\top K(t_i,t_j)\boldsymbol{a}_{j'}, \quad \boldsymbol{a}_i, \boldsymbol{a}_j \in \mathbb{R}^D \\ &= \boldsymbol{a}^\top \boldsymbol{K}(\mathcal{T},\mathcal{T})\boldsymbol{a}, \boldsymbol{a} \in \mathbb{R}^{DN} \end{aligned} \tag{4.23}$$

Here, $\boldsymbol{a}$ is the concatenation of all coefficients $\boldsymbol{a}_i$ over $\mathcal{T}$, $|\mathcal{T}| = N$. $\boldsymbol{K}(\mathcal{T},\mathcal{T}) \in \mathbb{R}^{DN \times DN}$ is the *Gram matrix* for all time points in the support of $\boldsymbol{\xi}$, and all joint angles of the robot. This matrix expresses the

degree of correlation or similarity among different joints throughout the time points in $\mathcal{T}$. It can be interpreted, alternatively, as a tensor metric in a Riemannian manifold (Amari, 1998; Ratliff et al., 2015). Its inverse is the key element that bridges the gradient of functional cost $\nabla\mathcal{U}$ (gradient in the RKHS, Eq.4.13), and its conventional gradient (Euclidean gradient).This makes the process covariant (invariant to reparametrization).

$$\nabla\mathcal{U} = \boldsymbol{K}^{-1}(\mathcal{T},\mathcal{T})\nabla_E\,\mathcal{U} \tag{4.24}$$

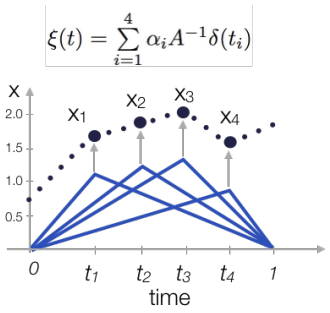
The minimizer of the full functional cost $\mathcal{U}$ has a closed form solution in Eq. 4.12, where the gradient $\nabla\mathcal{U}$ is the natural gradient in the RKHS. This can be seen as a warped version of the obstacle cost gradient according to the RKHS metric.

4.5.1 *Waypoint Parametrization as an Instance of RKHS*

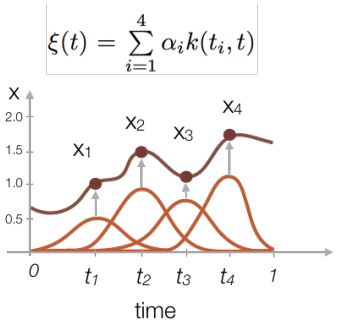
Consider a general Hilbert space of trajectories $\xi \in \Xi$, (not necessarily an RKHS) equipped with an inner product $\langle\xi_1,\xi_2\rangle_\Xi = \xi_1^T A\xi_2$. In the waypoint representation (Zucker et al., 2013), A is typically the Hessian matrix over points in the trajectory, which makes the norm in ξ penalize unsmooth and inefficient trajectories, in the sense of high acceleration trajectories.

The minimization under this norm $\|\xi\|_A = \sqrt{\xi^T A\xi}$ performs a line search over the negative gradient direction, where A dictates the shape of the manifold over trajectories.

This work generalizes the waypoint parameterization: we can represent waypoints by representing the trajectory in terms of delta Dirac basis functions $\langle\xi,\delta(t,\cdot)\rangle = \xi(t)$ with an additional smoothness metric A. Without A, each individual point is allowed to change without affecting points in the vicinity. Previous work overcame this caveat by introducing a new metric that propagates changes of a single point in the trajectory to all the other points. Kernel evaluations in this case become $k(t_i,\cdot) = A^{-1}\delta(t_i,\cdot)$, where $\xi(t) = \sum_i a_i A^{-1}\delta(t_i,\cdot)$. The inner product of two functions is defined as $\langle\xi_1,\xi_2\rangle_A = \sum_{i,j} a_i b_i A^{-1}\delta(t_i,t_i)$. Here, $\delta(t_i,\cdot)$ represents the finite dimensional delta function which is one for point t_i and zero for all the other points. A trajectory in the waypoint representation becomes a linear combination of the columns of A^{-1}. Columns of A^{-1} dictate how the corresponding point will affect the full trajectory, in Figure 4.5a. For an arbitrary kernel representation the behaviour of these points over the full trajectory are associated with the kernel functions associated with the space. For radial basis functions the trajectory is represented as Gaussians centered at a set of chosen time points (fewer in practice) instead of the full trajectory waypoints in Figure 4.5b. In this sense, we have a more



(a) Trajectory as linear combination of $A^{-1}\delta$ functions.



(b) Trajectory as linear combination of four Gaussian RBF functions.

Figure 4.5: Trajectory as linear combination of kernel functions for 1 DoF.

compact trajectory representation using RKHSs.

4.6 Cost Functional Analysis

Next, we analyze how the cost functional (different forms of the reduce operation in Section 4.3.1) affects obstacle avoidance performance and the resulting trajectory (§ 4.6).

We adopt a maximum cost version (§ 4.6.1), and an approximate integral cost version of the obstacle cost functional (§ 4.6.2). Other variants could be considered, providing the trajectory support remains finite, but we leave this as future work. Additionally, we compare the two forms against a more commonly used cost functional, the path integral cost (Ratliff et al., 2009), and we show our formulations perform comparably, while being faster to compute (§ 4.6.3). Based on these experiments, in the remaining sections we consider only the max-cost formulation, which we believe represents a good tradeoff between speed and performance.

4.6.1 Max-Cost Formulation

The maximum obstacle cost penalizes body points that pass too close to obstacles, *i.e.* high cost regions in workspace (regions inside/near obstacles). This maximum cost version of the reduce operation, considered in Eq. 4.9, can be described as picking time and body points (sampling) deepest inside or closest to obstacles, see Figure 4.1.

The sampling strategy for picking time points to represent the trajectory cost can be chosen arbitrarily, and further improved for time efficiency. We consider a simple version, where we sample points uniformly along sections of the trajectory, and choose N maximum violating points, one per section. This max-cost strategy allows us to represent trajectories in terms of a few points, rather then a set of finely discretized waypoints. This is a simplified version of the obstacle cost functional that yields a more compact representation (Ratliff et al., 2009; Park et al., 2012; Kalakrishnan et al., 2011).

4.6.2 Integral Cost Formulation

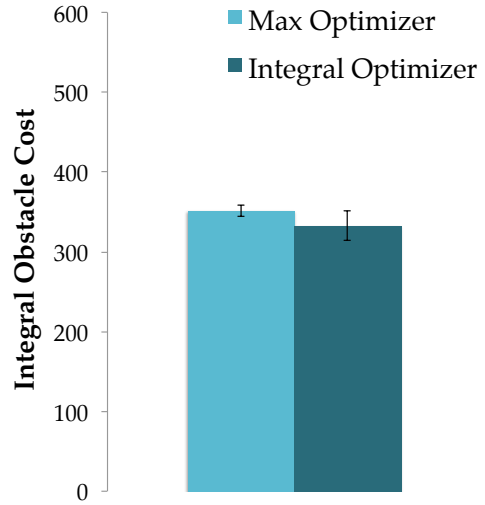
Instead of scoring a trajectory by the maximum of obstacle cost over time and body points, it is common to integrate cost over the entire trajectory and body, with the trajectory integral weighted by arc length to avoid velocity dependence (Zucker et al., 2013). While this path integral depends on all time and body points, we can approximate it to high accuracy from a finite number of point evaluations using numerical quadrature (Press et al., 1992). $\mathcal{T}(\xi)$ then becomes the set of abscissas of the quadrature method, which can be adaptively chosen

on each time step (e.g., to bracket the top few local optima of obstacle cost), see Section 4.4.1. In our experiments, we have observed good results with Gauss-Legendre quadrature.

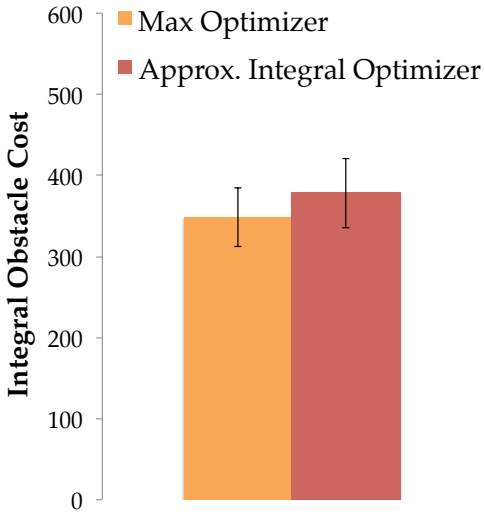
4.6.3 *Integral vs. Max-Cost Formulation*

We show that the max-cost does not hinder the optimization— that it leads to practically equivalent results as an integral over time and body points (Zucker et al., 2013). To do so, we manipulate the cost functional formulation, and measure the resulting trajectories' cost in terms of the integral formulation. Figure 4.6a shows the comparison: the integral cost decreased by only 5% when optimizing for the max.

Additionally we tested the max-cost formulation against the approximate integral cost using a Gauss-Legendre quadrature method. We performed tests over 100 randomly sampled scenarios and measured the final obstacle cost after 10 iterations. We used 20 points to represent the trajectory in both cases. Figure 4.6b shows the approximate integral cost formulation is only 8% above the max approach.



(a) $\mathcal{U}_{obs}$, Integral vs Max-cost.



(b) $\mathcal{U}_{obs}$, Approx integral vs Max-cost.

Figure 4.6: a) The integral costs after 5 large steps comparing using Gaussian RBG kernels vs. using the integral formulation (with waypoints). b) Gaussian RBF kernel integral cost using our max formulation vs. the approximate quadrature cost (20 points, 10 iterations).

4.7 *Experimental Results*

In what follows, we compare the performance of RKHS trajectory optimization vs. a discretized version (CHOMP) on a set of motion planning problems in a 2D world for a 3 DOF link planar arm as in Figure 4.7, and how different kernels with different norms affect the performance of the algorithm (§ 4.7.1). We then introduce a series of experiments that illustrate why RKHSs improve optimization (§ 4.7.2).

4.7.1 *RKHS with various Kernels vs. Waypoints*

For our main experiment, we systematically evaluate different parametrizations across a series of planning problems. We manipulate the parametrization (waypoints vs different kernels) as well as the number of iterations (which we use as a covariate in the analysis). Qualitatively, we optimize the stepsize for all methods over 10 iterations. We also select a stepsize that best performs in 10 iterations and we keep this parameter fixed and constant between methods $\sigma = 0.9$. To control for the cost functional as a confound, we use the max formulation for both parameterizations. We use iterations as our covariate because they are a natural unit in optimization, and because the amount of time per iteration is similar: the computational bottleneck in all methods is computing the maximum penetration points. We measure the obstacle and smoothness cost of the resulting trajectories. For the smoothness cost, we use the norm in the waypoint parametrization as opposed to

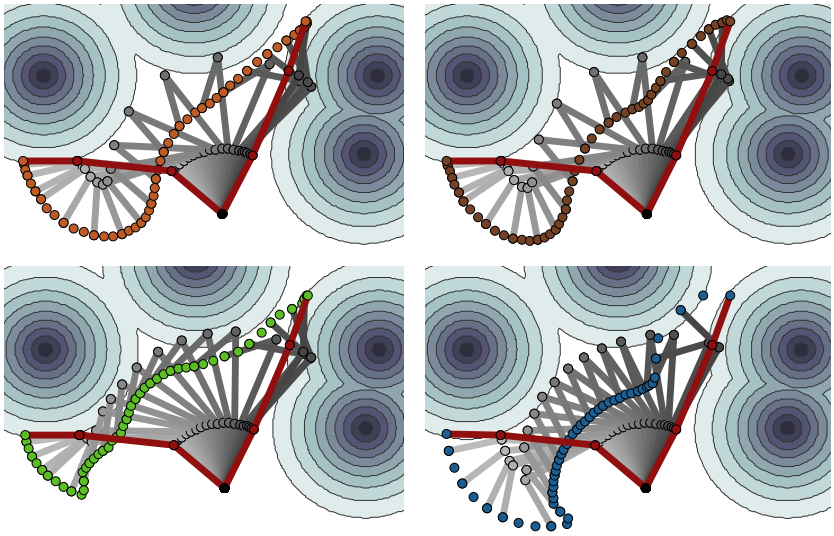


Figure 4.7: Robot 3DoF. Start and end configuration of 3-link arm in red, intermediate configurations in grey, end-effector colored in orange for Gaussian RBF kernel (top-left), in brown for B-splines kernel (top-right), in green for Laplacian RBF kernel (bottom-left), in blue for waypoints (bottom-right).Trajectory after 10 iterations. Isocontours of obstacle cost field shaded in grey (darker for higher cost).

the norm in the RKHS as the common metric, to avoid favoring our new methods.

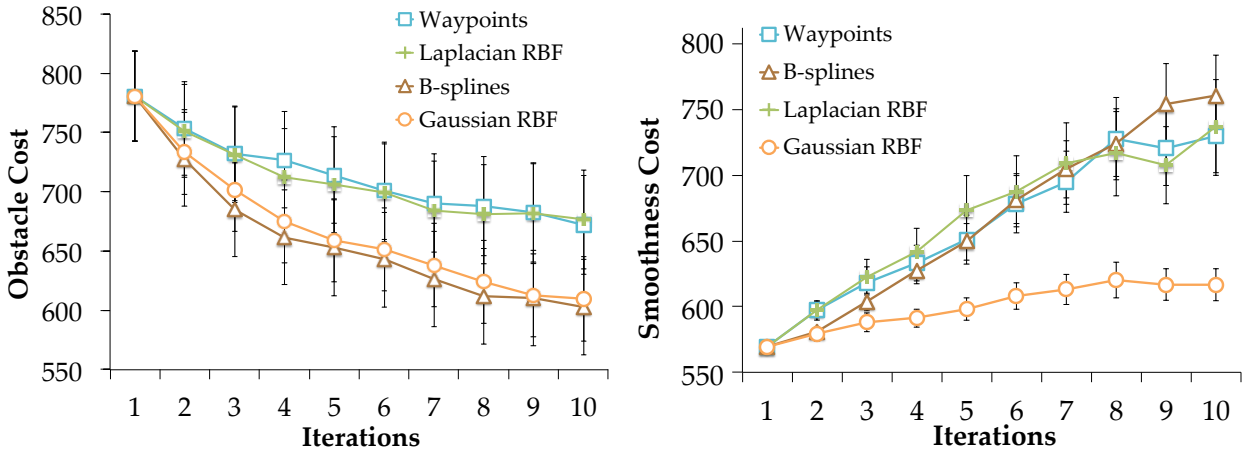
We use 100 different random obstacle placements and keep the start and goal configurations fixed. We compare the effectiveness of obstacle avoidance over 10 iterations, in 100 trials, of 12 randomly placed obstacles in a 2D environment, see Figure 4.7.

The trajectory is represented with 4 maximum-violation points over time and robot body points at each iteration. The RKHS parametrization results in comparable obstacle cost and lower smoothness cost for the same number of iterations. We performed a t-test using the last iteration samples, and showed that the Gaussian RBF RKHS representation resulted in significantly lower obstacle cost ($t(99) = -2.63, p < .01$) and smoothness cost ($t(99) = -3.53, p < .001$). We expect this to be true because with the Gaussian RBF parametrization, the algorithm can take larger steps without breaking smoothness, see Section 4.7.2.

We observe that waypoints and Laplacian RBF kernels (with large widths) have similar behaviour, while Gaussian RBF and B-spline kernels provide a smooth parametrization that allows the algorithm to take larger steps at each iteration. These kernels provide the additional benefit of controlling the motion amplitude, which is an important consideration for an adaptive motion planner. We compare the effectiveness of obstacle avoidance over 10 iterations, in 100 trials, of 12 randomly placed obstacles in a 2D environment, see Figure 4.8.

4.7.2 *RKHSs Allow Larger Steps than Waypoints*

One practical advantage of using an RKHS instead of the waypoint parametrization is the ability to take large steps during the optimization. Figure 4.9 compares the two approaches, while taking large steps:



(a) Obstacle cost vs. iterations for different kernels

(b) Smoothness cost vs. iterations for different kernels

Figure 4.8: (a,b): Cost over iterations for a 3DoF robot in 2D. Error bars show the standard error over 100 samples.

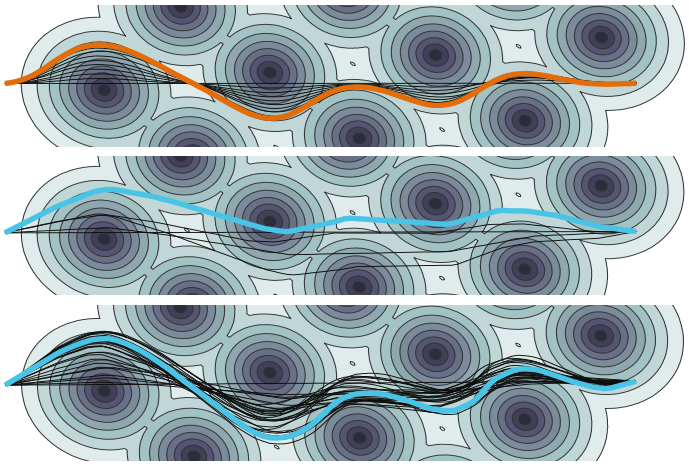


Figure 4.9: 1DOF 2D trajectory in a maze environment (obstacles shaded in grey). top: Gaussian RBF, large steps (5 it.); middle: waypoints, large steps (5 it.); bottom: waypoints, small steps (25 it.)

it takes 5 Gaussian RBF iterations to solve the problem, but would take 28 iterations with smaller steps for the waypoint parametrization — otherwise, large steps cause oscillation and break smoothness. The resulting obstacle cost is always lower with Gaussian RBFs ($t(99) = 5.32, p < .0001$). The smoothness cost is lower ($t(99) = 8.86, p < .0001$), as we saw in the previous experiment as well. Qualitatively, however, as seen before in Section 4.4, the Gaussian RBF trajectories appear smoother: even after just one iteration, as they do not break differential continuity.

We represented the discretized trajectory with 100 waypoints, but only used 5 kernel evaluation sets of points for the RKHS. We also tested the waypoint parametrization with only 5 waypoints, in order to evaluate an equivalent low dimensional representation, but this resulted in a much poorer trajectory with regard to smoothness.

4.7.3 *Experiments on a 7-DOF Manipulator*

This section describes a comparison of the waypoint parametrization (CHOMP) and the RKHS Gaussian RBF (GRBF) on a 7-DOF simple manipulation task (Figure 4.10). We qualitatively optimized the obstacle cost weight λ and smoothness weight β for all methods after 25 iterations (including the Waypoint method). The kernel width is kept constant over all 7-DOF experiments (σ=0.9 same as previous experiments). Figure 4.10a and Figure 4.10b illustrate both methods after 10 and 25 iterations, respectively.

Figure 4.10a shows the end-effector traces after 10 iterations. The path for CHOMP (blue) is very non-smooth and collides with the cabinet, while the Gaussian RBF optimization is able to find a smoother path (orange) that is not in collision. Note that we only use a single max-point for the RKHS version, which leads to much less computation per iteration as compared to CHOMP. Figure 4.10b shows the results from both methods after 25 iterations of optimization. CHOMP is now able to find a collision-free path, but the path is still not very smooth as compared to the RKHS-optimized path. These results echo our findings from the robot simulation and planar arm experiments.

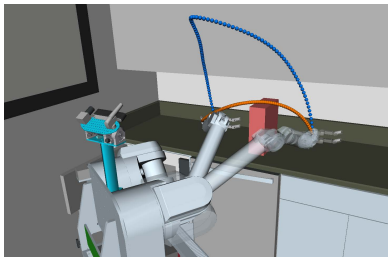
4.7.4 *Optimization under Different RKHS Norms*

In this section we experiment with different RKHS norms, where each one expresses a distinct notion of trajectory efficiency or smoothness. Here we have optimized obstacle cost weight (λ) after 5 iterations.

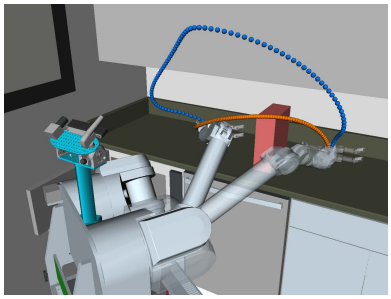
Figure 4.10c shows three different Gaussian RBF kernel based trajectories. We show the end-effector traces after 50 iterations for a Gaussian RBF kernel with independent joints Eq. 4.1 (red) , and a Gaussian RBF derivative with independent joints (yellow). We also consider interactions among joints (orange), with a vector-valued RKHS with a kernel composed of a 1-dim. time-kernel, combined with a $D \times D$ kernel matrix that models interactions across joints. We build this matrix based on the robot Jacobian at the start configuration (see Eq. 4.7).

The Gaussian derivative kernel takes the form of a sum over a Gaussian RBF kernel and its first order derivative. This kernel can be interpreted as an infinitely differentiable function with increased emphasis on its first derivative (penalizes trajectory velocity more). We can observe that the derivative kernel achieves a smoother path, when compared with the ordinary Gaussian RBF RKHS. The Gaussian RBF kernel with joint interactions yields trajectories that are also smoother than the independent Gaussian RBF path.

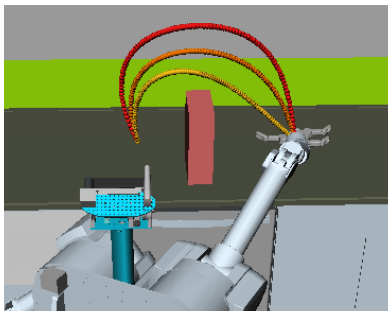
Figure 4.10d shows the end-effector traces of the GRBF derivative and the joint interaction kernel, together with a CHOMP trajectory with waypoints (blue). This result shows that Gaussian RBF RKHSs



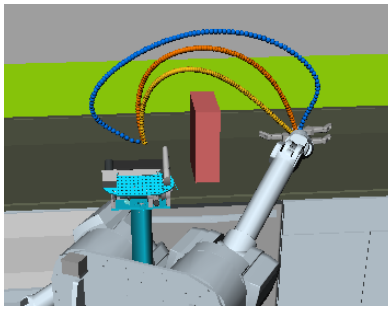
(a) Gaussian RBF (orange) with 1 max point (10 it., λ=20,β=0.5) vs. Waypoints (blue) with (10 it., λ=200).



(b) Gaussian RBF (orange) with 1 max point (25 it., λ=20,β=0.5) vs. Waypoints (blue) with (25 it., λ=200).



(c) Coupled vs. Indep. RKHSs, 50 it. GRBF-JJ with joint interactions (orange), GRBF-der derivative with joint interactions (yellow) (1 max point,λ=8,β=1.0), GRBF with independent joints (red), (1 max point, λ=16,β=1.0).



(d) RKHS vs. waypoints, 50 it. GRBF-JJ with joint interactions (orange), GRBF-der derivative with joint interactions (yellow) (1 max point,λ=8,β=1.0), waypoints in blue, (λ=40).

Figure 4.10: 7-dof experiment, plotting end-effector position from start to goal.

achieve a smoother, low-cost trajectory faster.

Figure 4.11 shows a time comparison between the Gaussian RBF with join interactions and CHOMP (waypoints) method. This shows that the kernel method is less time consuming than CHOMP (t(49), p<.001) over 10 iterations.

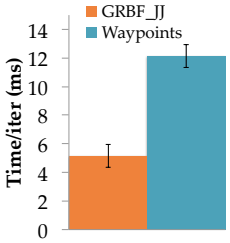


Figure 4.11: (a) Avg. time per iter. for 50 iter. GRBF RKHS with 1 max point ($\lambda = 8, \beta = 1$) vs. waypoints ($\lambda = 40$).

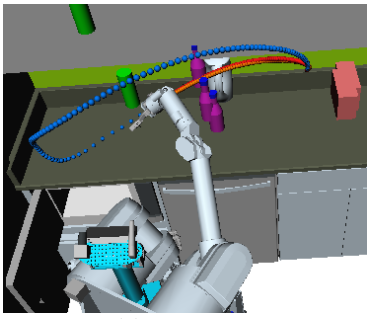
4.7.5 7-DOF Experiments in a Cluttered Environment

Next, we test our method in more cluttered scenarios. We create random environments, populated with different shaped objects to increase planning complexity (see Figure 4.12a). We place 8 objects (2 boxes, 3 bottles, 1 kettle, 2 cylinders) in the area above the kitchen table randomly in x, y, z positions. We plan with random collision-free initial and final configurations.

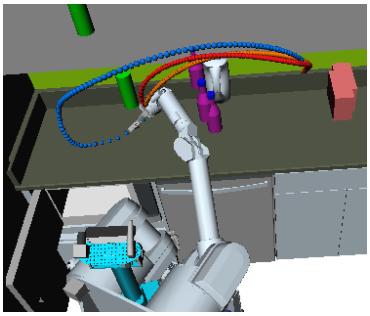
The RKHS trajectory with joint interactions (orange) achieves a smoother trajectory than the waypoint representation (blue). We perform a Wilcoxon t-test ($t(40) = -4.4, p < .001$), and obtains comparable obstacle costs. The method performs updates more conservatively and ensures very smooth trajectories. The other two RKHS variants consider independent joints, GRBF (red) and GRBF derivative (yellow).

Both GRBF and GRBF derivative kernels achieve lower cost trajectories than the waypoint parametrization ($t(40) = -3.7, p < .001$, $t(40) = -4.04, p < .001$), and converge in fewer iterations (approx. 12it.). The smoothness costs are not statistically significantly different that the waypoint representation, after 20 iterations. We show an example scenario with three variants of RKHSs (GRBF-JJ orange ($\lambda = 75$), GRB yellow ($\lambda = 45$), Waypoints blue ($\lambda = 100$)) after 6 iterations (Figure 4.12a), and after 20 iterations (Figure 4.12b).

Figure 4.13 shows the obstacle and smoothness cost per iteration, respectively.



(a) 6 iterations



(b) 20 iterations

Figure 4.12: End-effector from start to goal in cluttered environment. Waypoints (λ=40) (blue), GRBF (1 max point,λ=45,β=1.0) (orange), GRBF-der (1 max point,λ=25,β=1.0) (yellow).

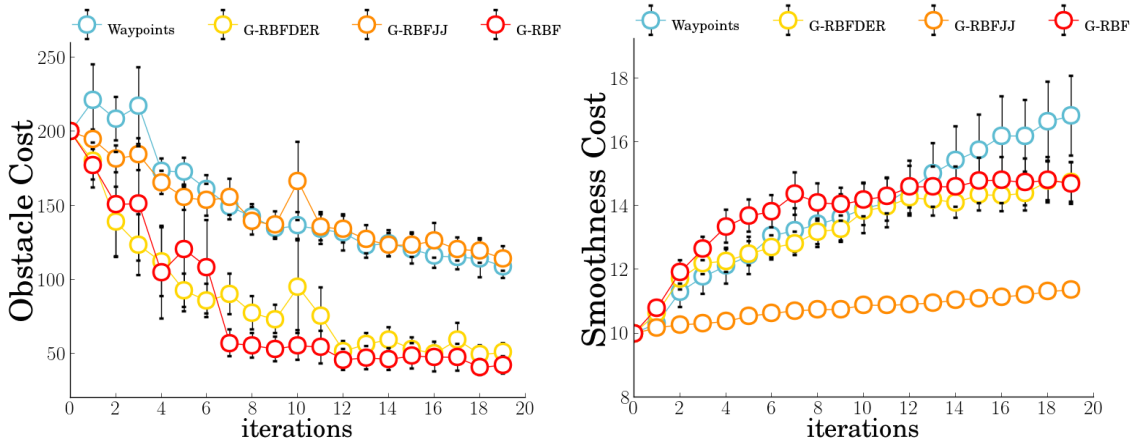


Figure 4.13: Obstacle and smoothness costs in cluttered environment for (GRBF-JJ) GRBF with joint interactions in orange, (GRBF-der) GRBF derivative with independent joints in yellow, (GRBF) GRBF independent joints in red, waypoints in blue, 20 it.

We compare all results according to CHOMP cost functions. As above, smoothness is measured in terms of total velocity (waypoint metric), and obstacle cost is given by distance to obstacles of the full trajectory (not only max-points).

The smoothness and obstacle weights are kept fixed in all scenarios (β=1.0).

4.7.6 7-DOF Experiments in a Constrained Environment

Next, we measured performance in a more constrained task. We placed the robot closer to the kitchen counter, and planned to a fixed goal configuration inside the microwave. We ran over 40 different random initial configurations. Figure 4.14 shows an example of Gaussian RBF with joint interactions (GRBF-JJ orange), independent joints (GRBF-der yellow), and waypoints (blue), after 30 iterations.

In Figure 4.14 we observe the end effector traces after 30 iterations. We optimized the model parameters for this scenario ($\beta = 1.0$, GRBF ($\lambda = 45$), GRBF-JJ ($\lambda = 40$), GRBF-der ($\lambda = 45$), Waypoints ($\lambda = 100$)). In Section 4.7.8 we compare against a non-optimized model.

We report smoothness and obstacle costs per iteration, see Figure 4.15. In more constrained scenarios, the RKHS variants achieve smoother costs than the waypoint representation ($p < .0.01$, GRBF $t(40) = -3.04$, GRBF-JJ $t(40) = 3.12$, GRBF-der $t(40) = -3.8$), for approximately the equivalent obstacle cost after 12 iterations. The joint interactions are smooth but take longer to converge (12 it.), while the GRBF kernels with independent joints converged to a collision free trajectory approximately in the same number of iterations as the waypoints experiments (4 it.).

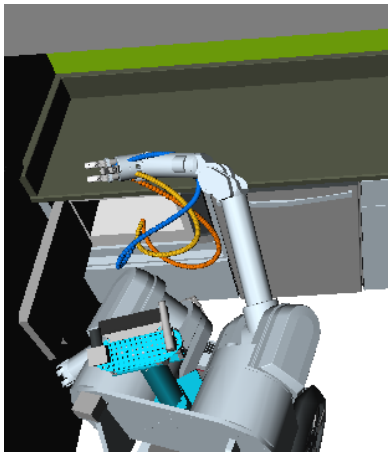


Figure 4.14: 7-DOF robot experiment, plotting the end-effector position from start to goal. Waypoints (λ=40) (blue), Gaussian RBF (1 max point,λ=45,β=1.0) (orange), Gaussian RBF JJ (1 max point,λ=45,β=1.0) (yellow) 30 iterations.

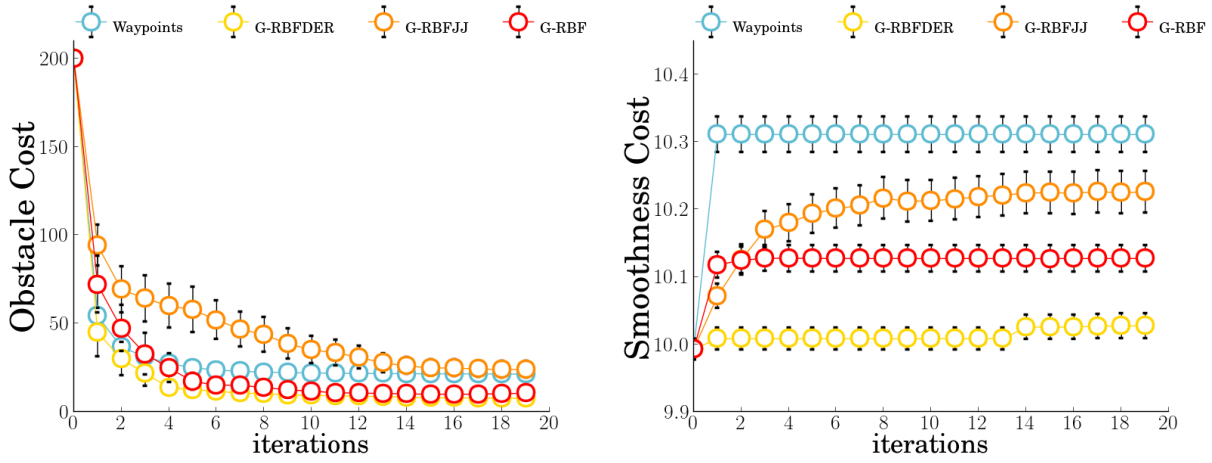


Figure 4.15: Obstacle and Smoothness costs in constrained environment after 30 it. in Fig.4.14

4.7.7 *RBF norm vs. Waypoint parametrization*

In this section, we study the effect of using an RKHS norm vs. the CHOMP norm. We perform an additional experiment, where we compare against an RKHS norm with waypoints. The trajectory becomes as weighted combinations of kernel functions $\xi(t) = \sum_i w_i k(t_i, \cdot) \forall_{t_i \in \mathcal{T}}$, evaluated at equally spaced time points (waypoints).

We performed qualitative experiments in a 2D setting with a 3-DOF planar arm. Figure 4.16 shows the end-effector traces (coloured line) after 10 iterations. We kept the RBF RKHS parameters fixed and optimized the obstacle weight, in order to achieve similar convergence speeds. We compare the waypoint CHOMP (blue) trajectory for the same number of waypoints (20) and only 4 max-cost time points for the GRBF RKHS (brown). The GRBF RKHS with equally spaced points (red) has a smooth behaviour, similar to the GRBF RKHS norm with max points. This qualitative example suggests that optimizing in the RKHS norm yields smoother trajectories, independently of using waypoints or max-cost time points.

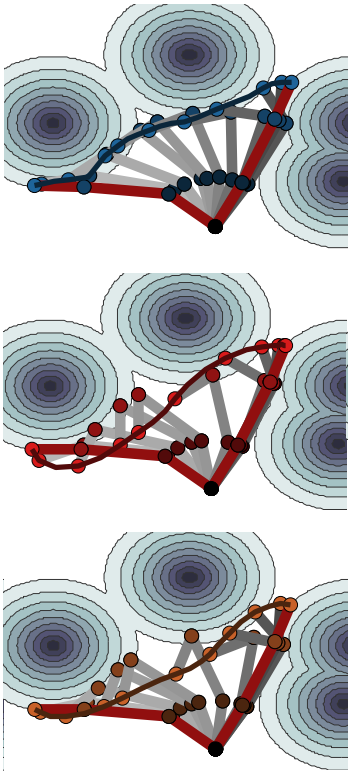


Figure 4.16: 3DoF robot in 2D trajectory profile using different kernels. Obstacle cost field in gray for the same environment and same initial and final configurations (axis fixed). Top: waypoints (blue), middle: Gaussian RBF with waypoints (red), bottom: Gaussian RBF (brown).

4.7.8 *Parameter selection sensitivity*

Here, we show the results for the 7-DOF simulation in constrained environments, without performing hyper-parameter optimization of the model (obstacle cost weight), (§ 4.7.6). We use the same model parameters as in the high clutter experiments (§ 4.7.5) without optimizing for this particular task. We measure performance over 40 different random initial configurations.

Figure 4.17 shows an example of Gaussian RBF with joint interactions in yellow (GRBF-JJ), independent joints in orange (GRBF), and waypoints in blue, after 30 iterations.

We show smoothness and obstacle costs per iteration, see Figure 4.18. Without weight cost tuning the RKHSs perform worse, but still achieve low cost and smooth solutions. The updates with joint interactions are smoother but take longer to converge, while the GRBF kernels with independent joints converged approximately in the same number of iterations as the waypoints experiments (12it.).

In Figure 4.17, we observe the end effector traces after 30 iterations. The joint interactions kernel is smoother compared with the waypoint parametrization, however the full trajectory smoothness is not statistically different when measured in the waypoint smoothness metric. We observe that optimizing the model parameters has a significant impact on the overall method performance, see Section 4.7.6. An interesting research extension could leverage this variability to produce planners with multi-resolution capabilities according to the surrounding environment (large motion in uncluttered scenes with high obstacle cost

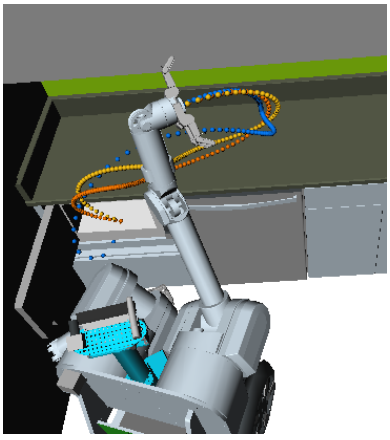


Figure 4.17: 7-DOF robot experiment, plotting the end-effector position from start to goal. Waypoints (λ=40) (blue), Gaussian RBF (1 max point,λ=45,β=1.0) (orange), Gaussian RBF JJ (1 max point,λ=45,β=1.0) (yellow).

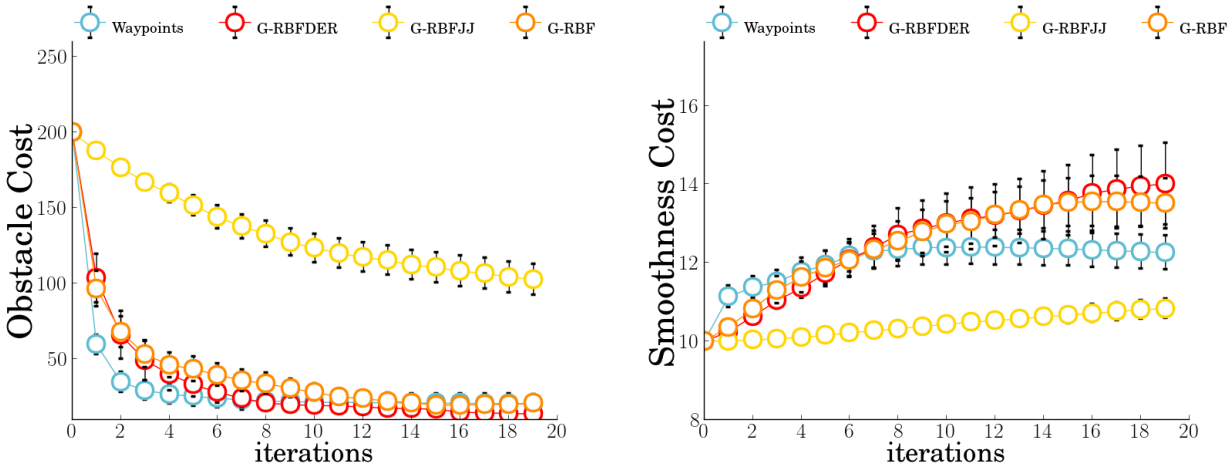


Figure 4.18: Obstacle and Smoothness cost of random trajectories in constrained environment after 30 iterations.

and high kernel widths vs. localized changes lower obstacle cost and with low kernel widths).

4.8 Conclusions

In this chapter, we presented a kernel approach to robot trajectory optimization: we represented smooth trajectories as vector valued functions in an RKHS. Different kernels lead to different notions of smoothness, including commonly-used variants as special cases (velocity, acceleration penalization).

We introduced a novel functional gradient trajectory optimization method based on RKHS representations, and demonstrated its efficiency compared with another optimization algorithm (CHOMP).

This method benefits from a low-dimensional trajectory parametrization that is fast to compute.

Furthermore, RKHSs enable us to plan with kernels learned from user demonstrations, leading to spaces in which more predictable motions have lower norm, and ultimately fostering better human-robot interaction (Dragan et al., 2014).

This work is an important step in exploring RKHSs for motion planning. It describes trajectory optimization under the light of reproducing kernels, which we hope can leverage the knowledge used in kernel based machine learning to develop better motion planning methods.

5
Planning with Method of Moments

In this chapter, we combine predictive models together with algorithms for planning under uncertainty. We introduce Recurrent Predictive State Policy (RPSP) networks, a recurrent architecture that brings insights from predictive state representations to reinforcement learning in partially observable environments. Predictive state policy networks consist of a recursive filter, which keeps track of a belief about the state of the environment, and a reactive policy that directly maps beliefs to actions, to maximize the cumulative reward. We discuss predictive state representations in Section 5.2, the predictive state model component in Section 5.4 and the proposed policy in Section 5.3. The control component is presented in Section 5.5 and the learning algorithm is presented in Section 5.6. In Section 5.8, we describe the experimental setup and results on control tasks: we demonstrate the performance of reinforcement learning using predictive state policy networks in multiple partially observable environments with continuous observations and actions.

5.1 Motivation

In this chapter, we put forth predictive state representations for learning to control dynamical systems in an interactive manner via *reinforcement learning* (RL). We consider an agent — which could be a robot or state machine— that is able to observe and act upon an environment, while receiving feedback/reward in a discrete time controlled process. We fully characterize this process via the *state* $\mathbf{q} \in \mathcal{Q}$ of the environment. This quantity provides a sufficient statistic for predicting future observations $p(o_1, \ldots, o_n \mid a_1, \ldots, a_n)$ conditioned on executed actions, hence correct estimation of the state is crucial to model the system. Given this interaction, the agent aims to learn a function $\pi : \mathcal{Q} \rightarrow \mathcal{A}$, which we denote as a *policy*, that generates actions $\mathbf{a} \in \mathcal{A}$ from the state of the system $\mathbf{q}$. The optimal policy will be the one that provides maximum reward $r_t \in \mathcal{R}$ over a sequence of interactions with the en-

reinforcement learning

policy π

vironment which we call episodes. We assume the agent is interacting with the environment in episodes, where each episode consists of T time steps in each of which the agent takes an action $\mathbf{a}_t \in \mathcal{A}$, and gets an observation $\mathbf{o}_t \in \mathcal{O}$ and a reward $r_t \in \mathbb{R}$. The agent chooses actions based on a stochastic policy $\pi_{\boldsymbol{\theta}}$ parameterized by a parameter vector θ:

$$\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}) \equiv p(\mathbf{a}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}, \boldsymbol{\theta}). \tag{5.1}$$

We would like to improve the policy rewards by optimizing $\boldsymbol{\theta}$ based on the agent's experience in order to maximize the expected long term reward

$$J(\pi_{\boldsymbol{\theta}}) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\gamma^{t-1} r_t \mid \pi_{\boldsymbol{\theta}}\right], \tag{5.2}$$

where $\gamma \in [0, 1]$ is a discount factor. We can express many Robotics tasks in the above framework; see Section 2.7. These problems, in most cases, are high dimensional and continuous. For instance, in robot motion prediction we need to learn continuous actions, in the form of twists and torques of the joints of the robot. Observations are typically continuous and high dimensional, corresponding to the robot's joint angles and velocities. On the other hand, in most NLP tasks it is sufficient to consider a finite set of possible observations and actions for input/output finite state machines, where the tabular setting can still be sustained (Luque et al., 2012).

The standard approach to model controlled processes involves estimating transition and observation probabilities in a predetermined state-space. This process is modeled formally via a Markov decision process (MDP) where the state of the system is fully known (Sutton et al., 1998; Gordon, 1999). In practice, and specially in Robotics domains, it is unrealistic to assume the true state is fully observable and noise-free, see Section 2.6 for a detailed discussion. In fact, observations are often aliased, noisy and insufficient to estimate the state of the system. To accommodate this uncertainty, many robotic learning problems are modeled with partial observability. Formally, MDPs are extended to partially observable Markov decision processes (POMDPs), where the state is unknown and instead a belief over states is maintained, *i.e.*, a probability distribution over the state space (Kaelbling et al., 1998). This formalism already accounts for some modeling uncertainty but still requires prior knowledge about an explicit state representation and a known transition and reward functions. In many Robotics cases it would be hard to provide this knowledge a priori. Next, we will describe two major lines of research that intend to learn a policy by interacting with the environment. The first consists of a model-based approach, where, ideally, we would like the agent to learn a model of

the system dynamics and also learn to optimize a policy. This model based setting establishes a form of reinforcement learning in which experience is gathered via the learned model in the form of simulated episodes— observation, action and reward sequences. Conversely, the second approach consists of a model free perspective, where experience is gathered directly by interacting with the environment and the policy is directly learned from this experience without requiring to build a transition model first.

Model-based learning may help reduce expert bias, when some of the model quantities are pre-specified, such as transition dynamics and reward function. This type of learning attempts to mitigate sample efficiency problems inherently associated with model-free approaches, when interaction with the environment is expensive. In Robotics, reducing the learning time with the real environment is highly desirable, since real experience may damage the robot and is often slower than learning via simulated episodes. In this sense, model-based methods have the advantage of requiring fewer samples and efficiently generalize to unseen situations (Atkeson et al., 1997). The learned policy is, however, strongly dependent on the quality of the model, which could be a potential bottleneck if the algorithm in not robust to model errors.

Recently, there has been significant progress in model free reinforcement learning combined with deep neural networks to learn a policy (Bojarski et al., 2016; Schulman et al., 2015). Deep reinforcement learning combines deep networks as a high level representation of a policy with reinforcement learning as an optimization method, and allows for end-to-end training. In this thesis, we attempt to combine both approaches by taking advantage of learned models to improve sample efficiency and exploit recent advances in model free methods using deep neural networks. Traditional applications of deep learning rely on standard architectures with sigmoid activations or rectified linear units; there is, however, an emerging trend of using composite architectures that contain parts inspired by other algorithms such as Kalman filtering (Haarnoja et al., 2016) and value iteration (Tamar et al., 2016). This composite structure brings forth benefits from other modeling tools, enhancing the performance of standard neural networks.

In this work, we focus on the more realistic scenario of partially observable environments, in which the agent is uncertain about the state of the environment. Instead, the agent has to keep track of a distribution over states, *i.e.*, a belief state, based on the entire history of observations and actions already executed. The typical approach in neural networks community to model partial observations relies recurrent architectures such as Long-Short-Term-Memory (LSTM) (Hochreiter et al., 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014b). How-

ever, these recurrent models are difficult to train due to non-convexity, and their hidden states lack a statistical meaning making them hard to interpret. Predictive states representations (PSRs) (§ 2.5.4) offer an alternative choice of models, whose predictive state may serve as a surrogate for belief over state, with the additional advantage of being interpreted as a sufficient statistics of future observations, conditioned on history and future actions (Littman et al., 2001; Singh et al., 2004a; Rosencrantz et al., 2004b; Boots et al., 2013b). Predictive representations do not require a prior state-space definition, since they emerge directly from observable quantities. This fact, in combination with efficient learning algorithms with theoretical guarantees, makes predictive models an ideal candidate for combining learning and planning.

5.2 *Predictive State Representations*

Predictive State Representations (PSRs) constitute an expressive form of modeling dynamical systems (Jaeger, 1999; Littman et al., 2001). It can be applied for both uncontrolled and controlled processes, so it could be used as a modeling tool in reinforcement learning. PSRs form a class of sequential models that track the dynamical system state via a *predictive state* $q \in \mathcal{Q}$. The predictive state can be described directly in terms of observable quantities, as an expectation over sufficient statistics of future observations. This forms a powerful class of models that subsume POMDPs (Littman et al., 2002). Instead of keeping a belief over the state-space, PSRs characterize the stochastic process by keeping an implicit state representation, that is constructed using the least possible information about the domain (Singh et al., 2004b). The dynamic system state can be described in terms of conditional success probabilities[1] of all tests $p(\boldsymbol{t}_O|\boldsymbol{h};\boldsymbol{t}_A) \in \Sigma^*$, where $\boldsymbol{t}_O = \boldsymbol{o}_{t:t+k-1} = [o_t, o_{t_2}, \ldots, o_{t+k-1}]$ and $\boldsymbol{t}_A = \boldsymbol{a}_{t:t+k-1} = [a_t, a_{t+2}, \ldots a_{t+k-1}]$, $\forall_{o\in\Sigma, a\in A}$ conditioned on all histories $\boldsymbol{h} = \boldsymbol{o}_{t-1:1} = [o_{t-1}, a_{t-1}, \ldots o_1, a_1]$ as seen in Section 2.5.4[2]

$$p(\boldsymbol{t}_O|\boldsymbol{h};\boldsymbol{t}_A) = p(\boldsymbol{o}_{t:t+k-1} \mid \boldsymbol{h};\boldsymbol{a}_{t:t+k-1}). \tag{5.3}$$

Littman et al. (2002) shows that it suffices to maintain a distribution of a potentially small set of *core tests* instead of all possible tests, to define a basis for representing the predictive state. All other tests can then be described as linear combinations of core events. The minimal number of core tests defines the complexity of the PSR, closely related to the rank of the system dynamics matrix, in Section 2.5.4. There are systems with rank K that may not be modeled by POMDPs with any finite number of states. However every system with rank K can be modeled by a PSR with exactly K core tests. Identifying the set of minimal core tests may prove to be a difficult task—*discovery*

[1] We say a test is successful if after executing the sequence of actions the sequence of observations received by the agent coincides with those specified by the test.

[2] for clarity we use ";" to denote that the agent will execute the following sequence of actions from the agent's policy.

problem— however, identifying a subspace that spans the space of all core tests is an easier and more efficient task—leading to *transformed PSRs* (Rosencrantz et al., 2004a). Rosencrantz et al. (2004a), Hsu et al. (2009), and Boots et al. (2011a) derived consistent learning methods for subspace identification of transformed PSRs, based on spectral decomposition of moment statistics. They were able to retrieve model parameters from observable moments that corresponded to those obtained from the true model. Further work of Boots et al. (2013b), Song et al. (2009), and Fukumizu et al. (2013b) extended the moment based approach to kernel embedding of distributions, extending the representation of observations and actions to the continuous domain. This can be useful in many applications where the domain is structured or high dimensional, potentially continuous, such as many problems in Robotics.

5.3 *Predictive Reinforcement Learning*

In this section, we propose a *Recurrent Predictive State Policy* (RPSP) network, a recurrent architecture that consists of a predictive state model acting as a recursive filter and a feed-forward neural network that directly maps predictive state to actions. The filter component represents state as the expectation of sufficient statistics of future observations, conditioned on history and future actions. Moreover, the successive application of the predictive state update procedure (i.e., filtering equations) results in a recursive computation graph that is fully differentiable with respect to model parameters amenable to gradient descent refinement. Therefore, we can treat predictive state models as recurrent networks and apply backpropagation through time (BPTT) (Hefny et al., 2017b; Downey et al., 2017) to optimize model parameters.

The proposed configuration results in a recurrent policy, where the recurrent part is implemented by a PSR instead of an LSTM or a GRU. As predictive states are a sufficient summary of the history of observations and actions, the reactive policy will have rich enough information to make its decisions, as if it had access to a true belief state. RPSPs leverage efficient and statistically consistent initialization by using PSRs as a filtering tool: spectral learning of PSRs provides a theoretically grounded initialization of a core component of the policy. Additionally, they can benefit from interpretable statistical interpretation of the predictive states, and can be evaluated and optimized based on that interpretation. The RPSP network can be trained end-to-end, for example using policy gradients in a reinforcement learning setting (Sutton et al., 2001) or supervised learning in an imitation learning setting (Ross et al., 2011). In this work, we focus on the former.

Throughout the rest of the chapter, we will use $\otimes$ to denote vectorized outer product: $\mathbf{x} \otimes \mathbf{y}$ is $\mathbf{x}\mathbf{y}^\top$ reshaped into a vector. There are two major approaches for policy modeling and optimization. The first consists of a value function-based approach, where we seek to learn a function, typically a deep network, to evaluate the value of each action at each state (a.k.a. Q-value) under the optimal policy. Given the Q function the agent can act greedily based on the estimated values. The second approach is policy optimization, where we learn a function that directly predicts optimal actions (or optimal action distributions), without pre-computing its corresponding value. This function or policy is directly optimized to maximize $J(\theta)$ using policy gradient methods (Schulman et al., 2015; Duan et al., 2016) or derivative-free methods (Szita et al., 2006). The two approaches are related, since they are amenable to variance reduction methods, and are compatible with approximation. However, we focus on the second approach as it is more robust to noisy continuous environments and modeling uncertainty (Sutton et al., 2001; Wierstra et al., 2010).

Our aim is to provide a new class of policy functions that combines recurrent reinforcement learning with recent advances in modeling partially observable environments using predictive state representations (PSRs). There have been previous attempts to combine predictive state models with policy learning. Boots et al. (2011b) proposed a method for planning in partially observable environments. The method first learns a PSR from a set of trajectories collected using an explorative blind policy. The predictive states estimated by the PSR are then considered as states in a fully observable Markov Decision Process. A value function is learned on these states using least squares temporal difference (Boots et al., 2010) or point-based value iteration (PBVI) (Boots et al., 2011b). The main disadvantage of these approaches is that it assumes a one-time initialization of the PSR and does not propose a mechanism to update the model based on subsequent experience, as a result, obtaining poor behaviour when the data is insufficient (the initial PSR is limited). Also, it has been shown that PSRs can benefit greatly from local optimization after a moment-based initialization (Downey et al., 2017; Hefny et al., 2017b).

Hamilton et al. (2014) proposed an iterative method to simultaneously learn a PSR and use the predictive states to fit a Q-function. Azizzadenesheli et al. (2016) proposed a tensor decomposition method to estimate the parameters of a discrete partially observable Markov decision process (POMDP) and used concentration inequalities to choose actions that maximize an upper confidence bound of the reward. This method does not employ a PSR, however it uses a consistent moment-based method to estimate model parameters. One common limitation in the aforementioned methods is that they are restricted to discrete

actions (some even assume discrete observations).

In this section, we introduce RPSP networks, a combination of a predictive state filter, and a reactive policy. This class of policies admits a statistically efficient initialization method, in contrast to typical RNNs, described in Section 2.5.6. However, it also admits an iterative update using policy gradient methods. We propose a novel combined update that exploits the interpretation of PSR states and optimizes both accumulated rewards and predictive accuracy of PSRs. We further make use of RPSPs to learn a policy in partially observable environments with continuous actions and continuous observations.

5.4 *Predictive State Representations of Controlled Models*

In this section, we revisit predictive state representations (§ 2.5.4), which constitute the state tracking (filtering) component of our model, and we discuss their relationship to recurrent neural networks RNNs. We follow the predictive state controlled model formulation provided by Hefny et al. (2017b). However, we could start from alternative predictive models such as predictive state inference machines (Sun et al., 2016a).

RNNs typically consider a history of sequence pairs of observations and actions $\mathbf{a}_1, \mathbf{o}_1, \mathbf{a}_2, \mathbf{o}_2, \ldots, \mathbf{a}_{t-1}, \mathbf{o}_{t-1}$ to compute its hidden state $\mathbf{q}_t$ using a recursive update equation $\mathbf{q}_{t+1} = f(\mathbf{q}_t, \mathbf{a}_t, \mathbf{o}_t)$. This recursive update is often non-linear and does not contemplate observation prediction. We could, however, consider a case where we learn to predict observations through an additional function $g(\mathbf{q}_t, \mathbf{a}_t) \equiv \mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t]$. Because $\mathbf{q}$ is latent, the function g that connects states to the output is unknown and has to be learned separately. In this case, the output could be predicted observations when the RNN is used for prediction, see Figure 5.1 (top).

Predictive state models define a similar recursive state update. However, the state $\mathbf{q}_t$ has a specific interpretation: it corresponds to a *predictive state*, meaning it encodes a conditional distribution of future observations $\mathbf{o}_{t:t+k-1}$ conditioned on future actions $\mathbf{a}_{t:t+k-1}$ (for example, in the discrete case, $\mathbf{q}_t$ could be a vectorized probability table in Eq. 5.3).[3]

The main characteristic of a predictive state is that it is defined entirely in terms of observable quantities. Depending on the choice of features used to compute the conditional expectation, we obtain different mappings from predictive states $\mathbf{q}_t$ to the prediction of $\mathbf{o}_t$ given $\mathbf{a}_t$. It is therefore crucial to consider expressive-enough feature mappings that will allow for a compact, yet accurate representation of the predictions. We consider random feature mappings (Rahimi et al., 2008) with kernel representations to learn this mapping consistently.

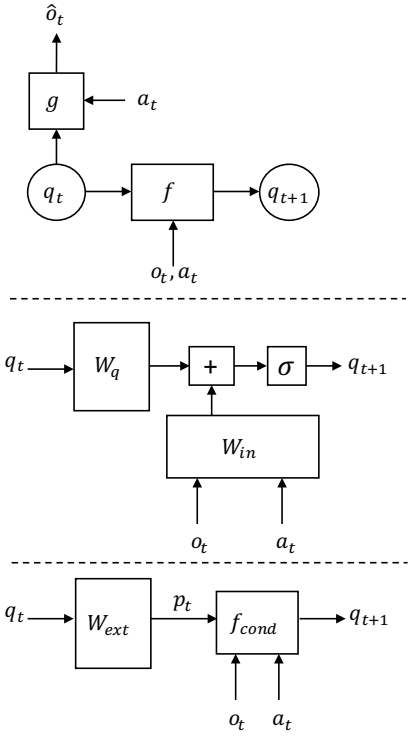


Figure 5.1: a) Computational graph of RNN and PSR and b) the details of the state update function f for both a simple RNN and c) a PSR. Compared to RNN, the observation function g is easier to learn in a PSR (see §5.4).

[3] The length-k depends on the observability of the system. We say a system is k-observable if maintaining the predictive state is equivalent to maintaining the distribution of the latent state of the system.

This is in contrast to a general RNN, where this mapping is unknown and typically requires non-convex optimization to be learned.[4] This characteristic allows for efficient and consistent learning of models with predictive states by reduction to supervised learning (Hefny et al., 2015b; Sun et al., 2016b).

Similar to an RNN, a PSR employs a recursive state update that consists of two steps (Section 2.5.6 for more details): a state extension, and a conditioning step.

The **state extension** defines an *extended state* $\mathbf{p}_t = \mathbb{E}[\xi^o(o_{t:t+k}) \mid \boldsymbol{h}_t; \xi^a(a_{t:t+k})]$ that is obtained from a linear transformation W_{ext} of the predictive state $\mathbf{q}_t = \mathbb{E}[\phi^o(o_{t:t+k-1}) \mid \boldsymbol{h}_t; \phi^a(a_{t:t+k-1})]$. The extended state defines a conditional distribution over an extended window of $k+1$ observations and actions, see Figure 5.2, and the linear map W_{ext} is an additional parameter of the model.

$$\mathbf{p}_t = W_{\text{ext}}\mathbf{q}_t \text{ in Eq. 2.75}$$

The **conditioning step** establishes the filter update equation, by taking into account the current $\mathbf{a}_t$ and $\mathbf{o}_t$, and a known conditioning function f_{cond} to arrive at the consecutive predictive state

$$\mathbf{q}_{t+1} = f_{\text{cond}}(\mathbf{p}_t, \mathbf{a}_t, \mathbf{o}_t) \text{ in Eq. 2.76.}$$

Figure 5.2 depicts the two steps. The conditioning function f_{cond} depends on the representation of $\mathbf{q}_t$ and $\mathbf{p}_t$. For example, in a discrete system, $\mathbf{q}_t$ and $\mathbf{p}_t$ could represent conditional probability tables and f_{cond} amounts to applying Bayes' rule. In the continuous setting using Hilbert space embeddings of distributions (Boots et al., 2013b), the conditioning function f_{cond} is given by the kernel Bayes' rule (Fukumizu et al., 2013b) (non-linear filter, in Eq. 2.77). In this work, we use random Fourier features (RFFs) (Rahimi et al., 2008) to represent the feature mapping of a Hilbert space embedding of a PSR (RFFPSR) as in Section 2.5.6. Observation and action features are based on RFFs of the RBF kernel projected into a lower dimensional subspace using randomized PCA (Halko et al., 2011). We use ϕ to denote this feature function. The observation function g in PSRs is a linear function of state $\mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t] = W_{\text{pred}}(\mathbf{q}_t \otimes \phi(\mathbf{a}_t))$, see the PSR state update scheme in Figure 5.3.

5.4.1 *Learning predictive states representations*

Learning PSRs is carried out in two steps: an initialization procedure using the method of moments (Hefny et al., 2015b) in Section 2.5.6 and a local optimization procedure using gradient descent (Hefny et al., 2017b).

Initialization:

[4] We do not require prediction error minimization to take advantage of PSR initialization. We can still consider a PSR filter without optimizing for prediction error, as seen later in the experimental section.

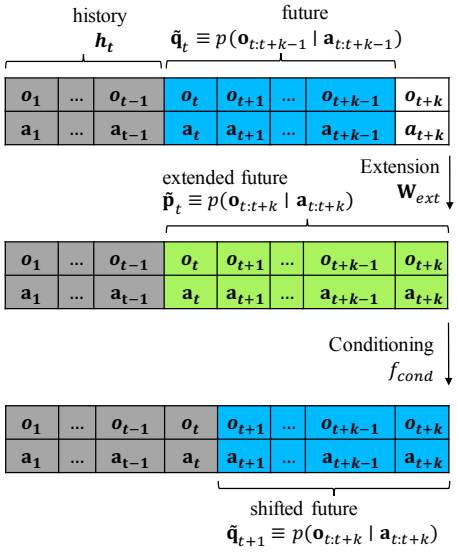


Figure 5.2: Illustration of the PSR extention and conditioning steps.

Here we follow the two-stage regression proposed by Hefny et al. (2015b).[5] The initialization procedure exploits the fact that $\mathbf{q}_t$ and $\mathbf{p}_t$ are represented in terms of observable quantities: since W_{ext} is linear and using Eq. 2.75, then $\mathbb{E}[\mathbf{p}_t \mid \mathbf{h}_t] = W_{\text{ext}}\mathbb{E}[\mathbf{q}_t \mid \mathbf{h}_t]$. Here $\mathbf{h}_t \equiv h(\mathbf{a}_{1:t-1}, \mathbf{o}_{1:t-1})$ denotes a set of features extracted from previous observations and actions (typically from a fixed length window ending at $t-1$). Because $\mathbf{q}_t$ and $\mathbf{p}_t$ are not hidden states, estimating these expectations on both sides can be done by solving a supervised regression subproblem. Given the predictions from this regression, solving for W_{ext} then becomes another linear regression problem. Once W_{ext} is computed, we can perform filtering to obtain the predictive states $\mathbf{q}_t$. We then use the estimated states to learn the mapping to predicted observations W_{pred}, which results in another regression subproblem.

[5] Specifically, we use the joint stage-1 regression variant for initialization.

In the random Fourier PSR (RFFPSR), we use linear regression for all subproblems (which is a reasonable choice with kernel-based features). This ensures that the two-stage regression procedure is free of local optima.

Local Optimization:

Although the PSR initialization procedure is consistent, it is based on method of moments and hence is not necessarily statistically efficient, since it requires a large amount of data to correctly build the moment estimates. Therefore estimation of the filter parameters can benefit from local optimization to help reduce mean square prediction error. Similarly to RNNs, PSRs also define a recursive computation graph, whose update is translated into

RPSP filter

$$\mathbf{q}_{t+1} = f_{\text{cond}}(W_{\text{ext}}(\mathbf{q}_t), \mathbf{a}_t, \mathbf{o}_t)) \tag{5.4}$$

$$\mathbb{E}[\mathbf{o}_t \mid \mathbf{q}_t, \mathbf{a}_t] = W_{\text{pred}}(\mathbf{q}_t \otimes \phi(\mathbf{a}_t)). \tag{5.5}$$

With a differentiable f_{cond}, the PSR can be trained using backpropagation through time (BPTT) (Werbos, 1990) to minimize prediction error. In this way PSRs consist of a special recurrent structure whose state representation and update function is implicitly defined by the choice of kernel, leading to consistent forms of initialization, which can be made better by consecutive BPTT.

5.5 *Recurrent Predictive State Policy (RPSP) Networks*

We now introduce Recurrent Predictive State Policies (RPSPs). In this section, we formally describe their components, followed by a policy learning algorithm in §5.6.

RPSPs consist of two fundamental components: a state tracking component, which models the state of the system, and is able to predict future observations; followed by a reactive policy, which could be

a possibly non-linear transformation of predictive states to estimate a distribution over actions, shown in Figure 5.3.

The integrated model seeks to improve a policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{ao}_1 \dots \mathbf{ao}_{t-1})$ by letting an agent interact with its environment. RPSPs learn the distribution over actions from a set of trajectories, *i.e.*, sequences of action-observation-reward triplets. At each time-step t, we get an observation $\mathbf{o}_t$ and an immediate reward r_t after executing $\mathbf{a}_t$.

This policy is parametrized by $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{PSR}, \boldsymbol{\theta}_{\text{re}}\}$ corresponding to the filter and reactive component respectively. The predictive model parameters $\boldsymbol{\theta}_{\text{PSR}} = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$ correspond respectively to the initial predictive state estimate, the linear weights from predictive states to extended states W_{ext}, and the linear regression matrix that predicts consecutive observations $\hat{\mathbf{o}}_{t+1} = W_{\text{pred}}\ \mathbf{q}_t$.

For the reactive component, we consider a stochastic non-linear policy $\pi_{\text{re}}(\mathbf{a}_t \mid \mathbf{q}_t) \equiv p(\mathbf{a}_t \mid \mathbf{q}_t; \boldsymbol{\theta}_{\text{re}})$ parametrized by $\boldsymbol{\theta}_{\text{re}}$. Since we are dealing with continuous actions, we assume a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_t, \Sigma)$ with parameters

$$\boldsymbol{\mu} = \varphi(\mathbf{q}_t; \boldsymbol{\theta}_\mu); \qquad \sigma = \exp \mathbf{r}^2, \tag{5.6}$$

where φ is a function (e.g., a feed-forward network) parametrized by $\boldsymbol{\theta}_\mu$, and $\mathbf{r}$ is a learnable vector. In the following section, we describe how these parameters can be learned.

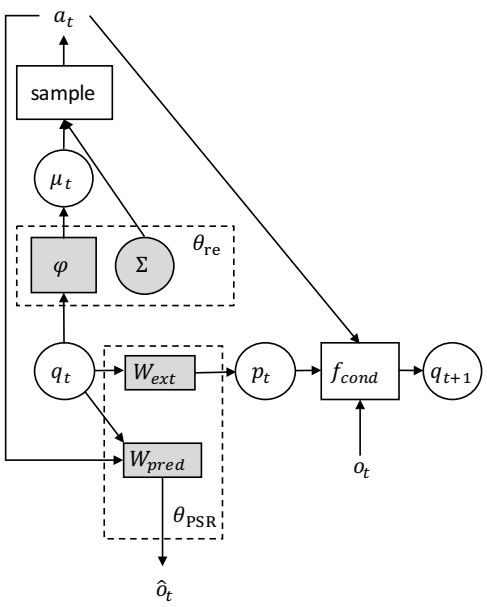


Figure 5.3: RPSP network: The predictive state is updated by a linear extension W_{ext} followed by non-linear conditioning f_{cond}. A linear predictor W_{pred} is used to predict observations, which is used to regularize training loss (see §5.6). A feed-forward reactive policy maps the predictive states $\mathbf{q}_t$ to a distribution over actions.

5.6 *Learning Recurrent Predictive State Policies*

RPSPs can make use of a principled initialization of the predictive layer, using moment-matching techniques (Hefny et al., 2015b) and update its parameters by refinement via gradient descent. In the initialization phase, we gather experience from an exploration policy, typically a blind Gaussian policy, and use it to initialize the PSR, as described in §5.4.

It is worth noting that this initialization procedure depends solely on sequences of observations and actions and does not take into account the reward signal. This can be particularly useful in environments where informative reward signals are infrequent.

In the second phase, starting from an initialized PSR and an initial random reactive policy, we iteratively collect trajectories using the current policy and use them to update the RPSP parameters. We update simultaneously both the reactive policy $\boldsymbol{\theta}_{\text{re}} = \{\boldsymbol{\theta}_\mu, \mathbf{r}\}$ and the predictive model $\boldsymbol{\theta}_{\text{PSR}} = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$ parameters, as detailed in Algorithm 8.

Let $p(\boldsymbol{\tau} \mid \boldsymbol{\theta})$ be the distribution over trajectories induced by the policy $\pi_{\boldsymbol{\theta}}$. The iterative parameter update seeks to locally minimize the following RPSP loss

RPSP loss

Algorithm 8: Recurrent Predictive State Policy network Optimization (RPSPO)
Input: Learning rate η.
Sample initial trajectories: $\{(o_t^i, a_t^i)_t\}_{i=1}^M$ from π_{exp}.
Initialize PSR:
$\boldsymbol{\theta}_{\text{PSR}}^0 = \{\mathbf{q}_0, W_{\text{ext}}, W_{\text{pred}}\}$ via 2-stage regression in §5.4.1.
Initialize reactive policy $\boldsymbol{\theta}_{\text{re}}^0$ randomly.
for $n = 1 \dots N_{max}$ iterations **do**
  for $i = 1, \dots, M$ batch of M trajectories from π^{n-1}: **do**
    Reset episode: a_0^i.
    for $t = 0 \dots T$ *roll-in* in each trajectory: **do**
      Get observation o_t^i and reward r_t^i.
      Filter $\mathbf{q}_{t+1}^i = f_t(\mathbf{q}_t^i, \mathbf{a}_t^i, \mathbf{o}_t^i)$ in (Eq. 5.4).
      Execute $\mathbf{a}_{t+1}^i \sim \pi_{re}^{n-1}(\mathbf{q}_{t+1}^i)$.
  Update $\boldsymbol{\theta}$ using $\mathcal{D} = \{\{\mathbf{o}_t^i, \mathbf{a}_t^i, r_t^i, \mathbf{q}_t^i\}_{t=1}^T\}_{i=1}^M$:
    $\boldsymbol{\theta}^n \leftarrow \textsc{Update}(\boldsymbol{\theta}^{n-1}, \mathcal{D}, \eta)$, as in §5.6.
Return: $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\text{PSR}}, \boldsymbol{\theta}_{\text{re}})$.

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= \alpha_1 \ell_1(\boldsymbol{\theta}) + \alpha_2 \ell_2(\boldsymbol{\theta}) \\ &= -\alpha_1 J(\pi_\theta) + \alpha_2 \sum_{t=0}^{T} \mathbb{E}_{p(\boldsymbol{\tau}|\boldsymbol{\theta})} \left[\| W_{pred}(\mathbf{q}_t \otimes \mathbf{a}_t) - \mathbf{o}_t \|^2 \right],\end{aligned} \tag{5.7}$$

which combines negative expected returns $J(\pi_\theta)$, with PSR prediction error. We minimize 1-step prediction error instead of general k-future prediction error, as often is used in PSRs (Hefny et al., 2017b; Boots et al., 2011a). This serves as an attempt to avoid biased estimates induced by non causal statistical correlations of observations ($\mathbf{o}_{t+i}$) with future actions ($\mathbf{a}_{t+i+1:t+k}$) when performing on-policy updates using a non-blind policy. Previous work obviates this problem by updating PSR parameters from a blind policy (Boots et al., 2011a; Hamilton et al., 2014).

An RPSP is essentially a special type of RNN, therefore any policy gradient algorithm could be used with an initialized RPSP. However, optimizing the PSR parameters to maintain low prediction error can be interpreted as a form of regularizing the network, based on the statistical interpretation of the recursive filter. In this approach, we optimize policy parameters $\boldsymbol{\theta}$ by minimization of a joint loss function, in Eq. 5.7, where $\alpha_1, \alpha_2 \in \mathbb{R}$ are hyper-parameters that determine the importance of the expected return and prediction error respectively[6]; later we describe how α_1 and α_2 can be optimized in §5.6.3: the negative expected return $\ell_1 = -J(\pi)$, and prediction cost ℓ_2 respectively.

[6] We only require one parameter for the optimization but later we rescale by variance each term and denote this scaling parameter α

Gradient-based policy search has been a successful approach in many applications in reinforcement learning (Peters et al., 2008; Deisenroth et al., 2013). Policy gradient methods provide a robust and powerful technique to estimate the optimal policy, even when we consider noisy state information (Baxter et al., 2001). They can account for continuous actions and are compatible with policies that have internal

memory (Wierstra et al., 2010). These characteristics make them ideal candidates to combine with a predictive model. We learn an update of the policy parameters using batches of trajectories by following a descent direction of the negative expected return $\boldsymbol{\theta}_\pi^{n+t} = \boldsymbol{\theta}_\pi^n + \alpha \nabla_{\boldsymbol{\theta}} J(\pi)$ with a specified learning rate $\alpha \in \mathbb{R}$, until the process converges.

RPSPs are a special type of a recurrent network policy, hence it is possible to adapt existing policy gradient methods, such as REINFORCE (Williams, 1992), to the joint loss in (5.7). In the following subsections, we propose different update variants: a joint update in Section 5.6.1 and an alternate variant in Section 5.6.2 based on two policy gradient methods: a REINFORCE style update, and a natural gradient variant based on Trust Region Policy Optimization (TRPO) (Schulman et al., 2015).

5.6.1 *Joint Variance Reduced Policy Gradient (VRPG)*

In this variant, we follow a REINFORCE method (Williams, 1992) to obtain a stochastic gradient of $J(\pi)$ (the first term in Eq. 5.7), using the likelihood ratio trick (Glynn, 1990; Aleksandrov et al., 1968). Let $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} \gamma^{t-1} r_t$ be the cumulative discounted reward for trajectory $\boldsymbol{\tau}$ given a discount factor $\gamma \in [0, 1]$, and let $J(\pi) = \mathbb{E}_{p(\tau|\boldsymbol{\theta})}[R(\tau)]$ be the expected cumulative discounted reward. From the likelihood ratio trick $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau}|\boldsymbol{\theta}) = p(\boldsymbol{\tau}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\tau}|\boldsymbol{\theta})$ we can compute the REINFORCE gradient $\nabla_{\boldsymbol{\theta}} J(\pi)$ as

cumulative discounted reward $R(\tau)$

REINFORCE gradient

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \mathbb{E}_{\boldsymbol{\tau} \sim p(\boldsymbol{\tau}|\boldsymbol{\theta})}[R(\boldsymbol{\tau}) \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{q}_t)],$$

In practice, we use a variance reducing variant of policy gradient (Greensmith et al., 2001) given by

$$\nabla_{\boldsymbol{\theta}} J(\pi) = \mathbb{E}_{\boldsymbol{\tau} \sim p(\boldsymbol{\tau}|\boldsymbol{\theta})} \sum_{t=0}^{T} [\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{q}_t)(R_t(\boldsymbol{\tau}) - b_t)], \tag{5.8}$$

where we replace the cumulative trajectory reward $R(\boldsymbol{\tau})$ by a reward-to-go function $R_t(\boldsymbol{\tau}) = \sum_{j=t}^{T} \gamma^{j-t} r_j$ computing the accumulated discounted future reward starting from t. We also denote $R_t(\tau_i)$ as the return at step i. b_t represents a variance reducing baseline. To further reduce variance we use a *baseline* $b_t \equiv \mathbb{E}_{\boldsymbol{\theta}}[R_t(\boldsymbol{\tau}) \mid \mathbf{a}_{1:t-1}, \mathbf{o}_{1:t}]$ which estimates the expected reward-to-go conditioned on the current policy and depends only on history and current observation. In the proposed implementation, we assume $b_t = \mathbf{w}_b^\top \mathbf{q}_t$ for a parameter vector $\mathbf{w}_b$ that is estimated using linear regression. Given a batch of M trajectories, a stochastic gradient of $J(\pi)$ is obtained by replacing the expectation in (5.8) with the empirical expectation over trajectories in the batch.

A stochastic gradient of the prediction error (the second term in Eq. 5.7) can be obtained using backpropagation through time (BPTT) (Werbos, 1990). With an estimate of both gradients, we have an estimate of the gradient of (5.7) which can be used to update the parameters through gradient descent, see Algorithm 9.

Algorithm 9: UPDATE (VRPG)

Input: $\boldsymbol{\theta}^{n-1}$, trajectories $\mathcal{D}=\{\boldsymbol{\tau}^i\}_{i=1}^M$, and learning rate η.

1: Estimate a linear baseline $b_t = \mathbf{w}_b^\top \mathbf{q}_t$, from the expected reward-to-go function for the batch $\mathcal{D}$:

$$\mathbf{w}_b = \arg\min_{\mathbf{w}} \left\| \frac{1}{TM} \sum_{i=1}^{M} \sum_{t=1}^{T_i} R_t(\boldsymbol{\tau}_t^i) - \mathbf{w}^\top \mathbf{q}_t \right\|^2.$$

2: Compute the VRPG loss gradient w.r.t. $\boldsymbol{\theta}$, in Eq. 5.8:

$$\nabla_{\boldsymbol{\theta}} \ell_1 = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=0}^{T_i} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i|\mathbf{q}_t^i)(R_t(\boldsymbol{\tau}^i) - b_t).$$

3: Compute the prediction loss gradient:

$$\nabla_{\boldsymbol{\theta}} \ell_2 = \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T_i} \nabla_{\boldsymbol{\theta}} \left\| W_{pred}(\mathbf{q}_t^i \otimes \mathbf{a}_t^i) - \mathbf{o}_t^i \right\|^2.$$

4: Normalize gradients $\nabla_{\boldsymbol{\theta}} \ell_j = \text{NORMALIZE}(\boldsymbol{\theta}, \ell_j)$, in Eq. 5.10.

5: Compute joint loss gradient as in Eq. 5.7:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \alpha_1 \nabla_{\boldsymbol{\theta}} \ell_1 + \alpha_2 \nabla_{\boldsymbol{\theta}} \ell_2.$$

6: Update policy parameters: $\boldsymbol{\theta}^n = \text{ADAM}(\boldsymbol{\theta}^{n-1}, \nabla_{\boldsymbol{\theta}} \mathcal{L}, \eta)$

7: **Return:** $\boldsymbol{\theta}^n = (\boldsymbol{\theta}_{\text{PSR}}^n, \boldsymbol{\theta}_{\text{re}}^n, \eta)$.

Algorithm 10: UPDATE (Alternating Optimization)

Input: $\boldsymbol{\theta}^{n-1}$, trajectories $\mathcal{D} = \{\boldsymbol{\tau}^i\}_{i=1}^M$.

1: Estimate a linear baseline $b_t = \mathbf{w}_b^\top \mathbf{q}_t$, from the expected reward-to-go function for the batch $\mathcal{D}$:

$$\mathbf{w}_b = \arg\min_{\mathbf{w}} \left\| \frac{1}{TM} \sum_{i=1}^{M} \sum_{t=1}^{T_i} R_t(\boldsymbol{\tau}_t^i) - \mathbf{w}^\top \mathbf{q}_t \right\|^2.$$

2: Update $\boldsymbol{\theta}_{\text{PSR}}$ using the joint VRPG loss gradient in Eq. 5.7:

$$\boldsymbol{\theta}_{\text{PSR}}^n \leftarrow \text{UPDATE VRPG}(\boldsymbol{\theta}^{n-1}, \mathcal{D}).$$

3: Compute descent direction for TRPO update of $\boldsymbol{\theta}_{\text{re}}$:

$v = H^{-1} g$, where

$$H = \nabla_{\boldsymbol{\theta}_{\text{re}}}^2 \sum_{i=1}^{M} D_{KL}\left(\pi_{\boldsymbol{\theta}^{n-1}}(\mathbf{a}_t^i|\mathbf{q}_t^i) \mid \pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i|\mathbf{q}_t^i)\right),$$

$$g = \nabla_{\boldsymbol{\theta}_{\text{re}}} \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T_i} \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t^i|\mathbf{q}_t^i)}{\pi_{\boldsymbol{\theta}^{n-1}}(\mathbf{a}_t^i|\mathbf{q}_t^i)} (R_t(\boldsymbol{\tau}^i) - b_t).$$

4: Determine a step size η through a line search on v to maximize the objective in (5.9) while maintaining the constraint.

5: $\boldsymbol{\theta}_{\text{PSR}}^n \leftarrow \boldsymbol{\theta}_{\text{PSR}}^{n-1} + \eta v$

6: **Return:** $\boldsymbol{\theta}^n = (\boldsymbol{\theta}_{\text{PSR}}^n, \boldsymbol{\theta}_{\text{re}}^n)$.

5.6.2 *Alternating Optimization*

TRPO:

In this section, we describe a method that utilizes a natural gradient-style update, Trust Regression Policy Optimization (TRPO (Schulman

et al., 2015)). TRPO offers an alternative to the vanilla natural policy gradient methods, that has shown superior performance in practice (Duan et al., 2016). It uses a natural gradient update and enforces a constraint on the change in the policy at each update step, measured by KL-divergence between conditional action distributions. This constraint along with the natural gradient correction ensure smoother changes of the policy parameters.

Each TRPO update is a solution to the following constrained optimization problem: *TRPO gradient*

$$\begin{aligned}\boldsymbol{\theta}^{n+1} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\pi^n)} \left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{q}_t)}{\pi^n(\mathbf{a}_t \mid \mathbf{q}_t)}(R_t(\boldsymbol{\tau}) - b_t)\right] \\ \text{s. t. } \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\pi^n)}\left[D_{KL}\left(\pi^n(.|\mathbf{q}_t) \mid \pi_{\boldsymbol{\theta}}(.|\mathbf{q}_t)\right)\right] \leq \epsilon,\end{aligned} \tag{5.9}$$

where π^n is the policy induced by $\boldsymbol{\theta}^n$, R_t and b_t are the reward-to-go and baseline functions defined in §5.6.1, and ϵ is an hyper-parameter that determines the allowed amount of change. In practice, we optimize using conjugate gradient with line search, where we use an efficient implementation using Fisher information matrix vector products, as proposed in Schulman et al. (2015). For RPSP networks, we extend TRPO by introducing an additional constraint that minimizes the Frobenius norm of the prediction matrix $\|W^n_{\text{pred}} - W^{\theta}_{\text{pred}}\|_F$ to account for updates of the PSR parameters. This constraint also ensures smooth changes of predictive parameters.

Alternating Optimization:
Next, we investigate a hybrid method: we would like to benefit from TRPO-type updates, but we observed that TRPO tends to be computationally intensive with recurrent architectures, mostly due to the need of performing a line-search and using conjugate gradient. Instead, we resort to the following alternating optimization: in each iteration, we use TRPO to update the reactive policy parameters $\boldsymbol{\theta}_{\text{re}}$, which involve only a feedforward network. Then, we use a gradient step on (5.7), as described in §5.6.1, to update the PSR parameters $\boldsymbol{\theta}_{\text{PSR}}$, see Algorithm 10.

We can summarize the final algorithm as follows (see Algorithm 8 for details): first, we initialize the filtering parameters $\boldsymbol{\theta}_{\text{PSR}}$ using a moment-based method: we generate initial trajectories $\{(\mathbf{o}^i_t, \mathbf{a}^i_t)_{t=1}^T\}_{i=1}^M$ from an exploratory policy π_{exp} (e.g. random Gaussian), and train a PSR using two-stage regression, described in §5.4.1.

After initialization, we start an iterative learning process: at each step we update the predictive state using the current action/observation pair (filtering), in Eq. 5.4. Then we sample an action from the distribution over actions that the network outputs $\mathbf{a_t} \sim \pi_{\text{re}}$. At the end of an episode we update the full set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{PSR}}, \boldsymbol{\theta}_{\pi_{\text{re}}}\}$ using backpropagation through time (Werbos, 1990).

Algorithm 8 makes use of a subroutine, Update, that updates policy parameters in order to minimize the corresponding loss function Eq. 5.7, either in a joint §5.6.1 or alternating approach §5.6.2. For clarity, we provide the pseudo-code for the joint and alternating update steps defined in this UPDATE step. We show the joint VRPG update step in Algorithm 9, and the alternating (Alternating Optimization) update in Algorithm 10.

5.6.3 *Variance Normalization*

It is difficult to make sense of the values of α_1, α_2, especially if the gradient magnitudes of their respective losses are not comparable. We have verified empirically that these magnitudes are very unbalanced. So, we propose a principled approach for finding these relative weights.

We use $\alpha_1 = a_1\tilde{\alpha}_1$ and $\alpha_2 = a_2\tilde{\alpha}_2$, where a_i is a relative important factor of l_i to the combined objective $\mathcal{L}$. $\tilde{\alpha}_1$ and $\tilde{\alpha}_2$ are dynamically adjusted to maintain the property that the gradient of each loss weighted by $\tilde{\alpha}$ has unit (uncentered) variance, in Eq. 5.10. To do so, we maintain the variance of the gradient of each loss through exponential averaging and use it to adjust the weights.

$$\mathbf{v}_i^{(n)} = (1-\beta)\mathbf{v}_i^{(n-1)} + \beta \sum_{\boldsymbol{\theta}_j \in \boldsymbol{\theta}} \|\nabla_{\boldsymbol{\theta}_j}^{(n)} \ell_i\|^2 \tag{5.10}$$

$$\tilde{\alpha}_i^{(n)} = \frac{1}{\sqrt{\sum_{\boldsymbol{\theta}_j \in \boldsymbol{\theta}} \mathbf{v}_{i,j}^{(n)}}},$$

where $\beta \in \mathbb{R}$ is the exponential averaging constant.

5.7 *Connection to RNNs with LSTMs/GRUs*

RPSP-networks and RNNs both define recursive models that are able to retain information about previously observed inputs. BPTT for learning predictive states in PSRs bears many similarities with BPTT for training hidden states in LSTMs or GRUs. In both cases the state is updated via a series of alternate linear and non-linear transformations. For predictive states the linear transformation $\mathbf{p}_t = W_{\text{ext}}\ \mathbf{q}_t$ represents the system prediction: from expectations over futures $\mathbf{q}_t$ to expectations over extended features $\mathbf{p}_t$. The non-linear transformation, in place of the usual activation functions (tanh, ReLU), is replaced by f_{cond} that conditions on the current action and observation to update the expectation of the future statistics $\mathbf{q}_{t+1}$ in Eq. 2.76. When we consider linear transformations W_{pred} and W_{ext} we refer to transformations between kernel representations, between Hilbert Space Em-

beddings. It is worth noting that these transformations represent non-linear state updates, as in RNNs, but where the form of the update is defined by the choice of representation of the state. For Hilbert Space embeddings it corresponds to conditioning using Kernel Bayes' Rule, in Eq. 2.77. PSRs have the additional benefit of having a clear interpretation as a prediction of future observations and could be trained based on that interpretation. They can be initialized using moment matching techniques, with good theoretical guarantees. In contrast RNNs may exploit heuristics for an improved initialization (Zimmermann et al., 2012), however, defining the best strategy does not guarantee good performance. In this work the proposed initialization, provides guarantees in the infinite data assumption (Hefny et al., 2017b).

5.8 Experiments

We evaluate the RPSP-network's performance on a collection of reinforcement learning tasks using OpenAI Gym Mujoco environments.[7] We learn a policy for Swimmer, Walker2D, Hopper and Cart-Pole environments with off-the-shelf environment specifications, see Figure 5.7. We consider only partially observable environments: only the angles of the joints of the agent are visible to the network, without velocities.

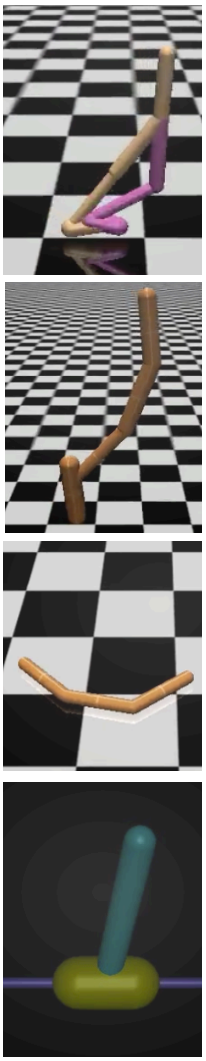


Figure 5.4: OpenAI Gym Mujoco environments: Walker2d, Hopper, Swimmer, Cart-Pole (Inverted-Pendulum) (top-down).

[7] https://gym.openai.com/envs#mujoco We ensure all models and baselines use the same OpenAI Gym base environment settings.

Proposed Models:
We consider an RPSP with a predictive component based on RFFPSR, as described in §5.4.1 and §5.5. For the RFFPSR we use 1000 random Fourier features on observation and action sequences followed by a PCA dimensionality reduction step to d dimensions. We report the results for the best choice of $d \in \{10, 20, 30\}$.

We initialize the RPSP with two stage regression on a batch of M_i initial trajectories (100 for Hopper, Walker and Cart-Pole, and 50 for Swimmer, equivalent to 10 extra iterations, or 5 for Swimmer). We then experiment with both joint VRPG optimization (**RPSP-VRPG**) described in §5.6.1 and alternating optimization (**RPSP-Alt**) in §5.6.2. For RPSP-VRPG, we use the gradient normalization scheme described in §5.6.3.

Additionally, we consider an extended variation (denoted by **+obs**) that concatenates the predictive state with a window w of previous observations as an extended form of predictive state $\tilde{\mathbf{q}}_t = [\mathbf{q}_t, \mathbf{o}_{t-w:t}]$. If PSR learning suceeded perfectly, this additional information would be redundant; however, we observe in practice that providing observations helps the model learn faster and more stably.

Competing Models:
We compare our models to a finite memory model (**FM**) and a recur-

rent neural network with GRUs (**GRU**). We have experimented with both LSTMs and GRUs, but the later provided better results. The finite memory models are analogous to RPSP, but replace the predictive state with a window of past observations. We tried three variants, FM1, FM2 and FM5, with window size of 1, 2 and 5 respectively. Note that FM1 effectively assumes a fully observable environment. We compare to GRUs with $16, 32, 64$ and 128-dimensional hidden states. We optimize network parameters using the *RLLab* (https://github.com/openai/rllab) implementation of TRPO with two different learning rates ($\eta = 10^{-2}, 10^{-3}$). In each model, we use a linear baseline for variance reduction where the state of the model (i.e., past observation window for FM, latent state for GRU and predictive state for RPSP) is used as the predictor variable.

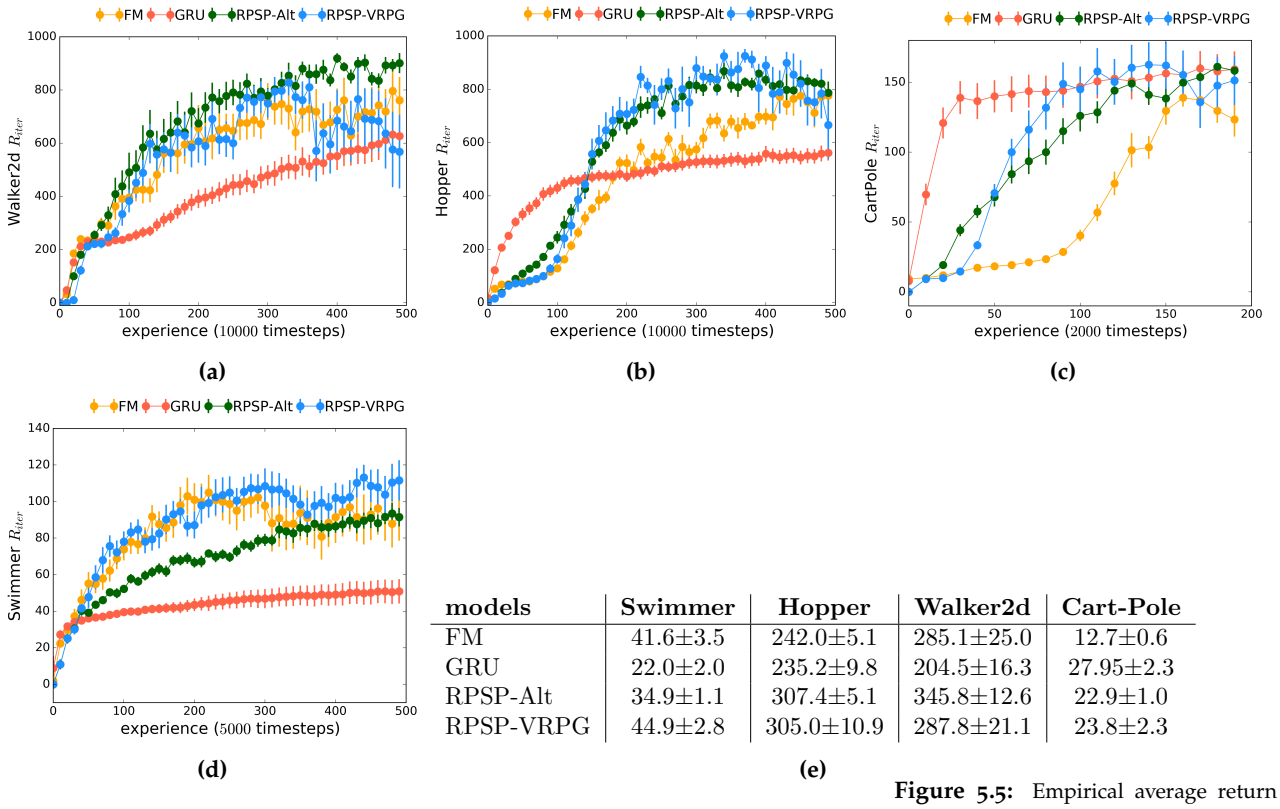
Evaluation Setup:
We run each algorithm for a number of iterations based on the environment (see Figure 5.5). After each iteration, we compute the average return $R_{iter} = \frac{1}{M}\sum_{m=1}^{M}\sum_{j=1}^{T_m} r_m^j$ on a batch of M trajectories, where T_m is the length of the m^{th} trajectory. We repeat this process using M=10 different random seeds and report the average and standard deviation of R_{iter} over the 10 seeds for each iteration.

For each environment, we set the number of samples in the batch to 10000 and the maximum length of each episode to 200, 500, 1000, 1000 for Cart-Pole, Swimmer, Hopper and Walker respectively. For RPSP, we found that a step size of 10^{-2} performs well for both VRPG and alternating optimization in all environments. The reactive policy contains one hidden layer of 16 nodes with ReLU activation. For all models, we report the results for the choice of hyper-parameters that resulted in the highest mean cumulative reward (area under curve).

5.9 *Results*

Cross-environment performance:
Figure 5.5 shows the empirical average return vs. the amount of interaction with the environment (experience) measured in time steps. For RPSP networks, the plots are shifted to account for the initial trajectories used to initialize the RPSP filtering layer. The amount of shifting is equivalent to 10 trajectories. We report the best variant for each model (+obs for RPSP-VRPG Hopper,Swimmer and Walker2d, and +obs for RPSP-Alt Swimmer and Walker, FM5 for Hopper and Walker2d, and FM2 for CartPole and Swimmer, d=32 for GRU CartPole, d=128 for Hopper and d=64 for Swimmer and Walker2d).

We report the cumulative reward for all environments in Table **??**. For all environments except CartPole, a variant of RPSP wins over the



models	Swimmer	Hopper	Walker2d	Cart-Pole
FM	41.6±3.5	242.0±5.1	285.1±25.0	12.7±0.6
GRU	22.0±2.0	235.2±9.8	204.5±16.3	27.95±2.3
RPSP-Alt	34.9±1.1	307.4±5.1	345.8±12.6	22.9±1.0
RPSP-VRPG	44.9±2.8	305.0±10.9	287.8±21.1	23.8±2.3

(e)

Figure 5.5: Empirical average return over a batch of $M = 10$ trajectories of T time steps. (a) Walker $T = 1000$, (b) Hopper $T = 1000$, (c) Cart-Pole $T = 200$, (d) Swimmer $T = 500$ environments. Finite memory model $w = 2$ (FM:orange), RNN with GRUs (GRU:red), RPSP with joint optimization (RPSP-VRPG: blue), RPSP with alternate optimization (RPSP-Alt: green). For RPSP models, the graphs are shifted to the right to reflect the use of extra trajectories for initialization. (e) Summary of results: area under curve for accumulated return ± standard error over 10 trials.

baselines. For CartPole, however, convergence was slower than the baselines, probably due to the very small trajectories used to initialize the PSR (the pole would fall immediately after a few iterations (5 it.) resulting in very poor initialization). For CartPole the top RPSP model performs better than the FM model (t-test, $p < 0.01$) and it is not statistically significantly different than the GRU model. For Swimmer our best performing model is only statistically better than FM model (t-test, $p < 0.01$), while for Hopper our best RPSP model performs statistically better than FM and GRU models (t-test, $p < 0.01$) and for Walker2d RPSP outperforms only GRU baselines (t-test, $p < 0.01$). We also note that RPSP-Alt provides similar performance to the joint optimization (**RPSP-VRPG**), but converges faster.

These results suggest that RPSP-networks provide an overall high performing method to learn a continuous stochastic policies, and in particular that RPSP-networks are more reliably good than competing methods across environments.

To quantify this trend we evaluate the aggregate performance of each method in all environments. We report the average area under the curve $AUC = \frac{1}{4}\sum_{env}\sum_{n=1}^{100} R^{env}_{ns_{env}} / \max_i R^{env}_i$, where $\max_i R^{env}_i$ is the reward at step i in environment env and s_{env} is a step size that calibrates the number of iterations in each environment (2 Cart-Pole, 3 for Swimmer and 5 for Hopper and Walker).

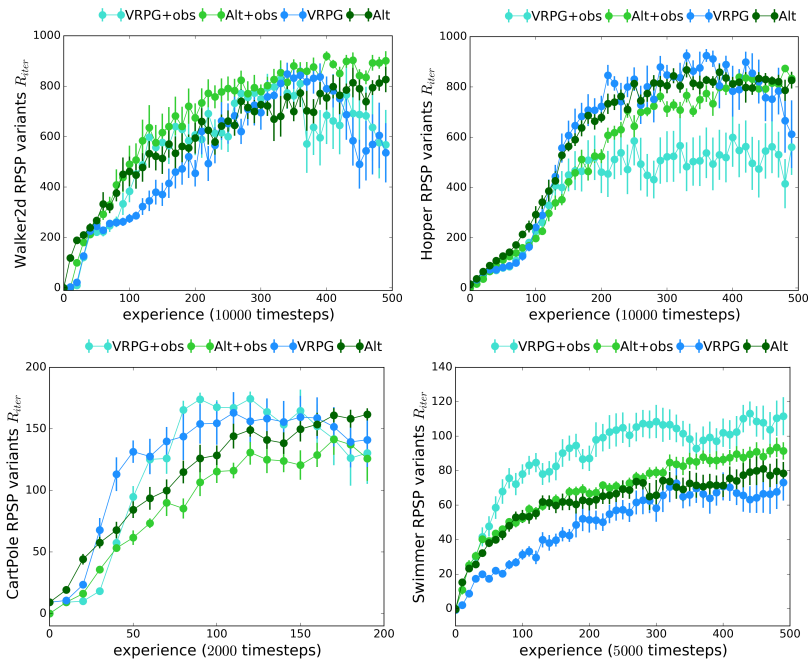


Figure 5.6: Empirical average return over a batch of $M = 10$ trajectories of T time steps. (Left to right top down): Walker $T = 1000$, Hopper $T = 1000$, Cart-Pole $T = 200$, Swimmer $T = 500$ environments. Comparison of RPSP-nets variants RPSP with joint optimization (VRPG: purple), RPSP with joint optimization plus observations (VRPG+obs: cyan), RPSP with alternate optimization (RPSP-Alt: dark green), RPSP with alternate optimization plus observations (RPSP-Alt: light green). for initialization.

Minimal Comparisons: Next, we conducted a series of minimal comparisons where we intended to investigate the benefit of the proposed contributions.

The RPSP model is based on the following components: (1) a state filter using PSR (2) a consistent initialization using two-stage regression (3) an end-to-end training of filter and policy (4) using observation prediction loss to regularize training. We define variants of RPSP that lack some of these features, in order to evaluate which of these features contributes to performance.

RPSP variants: augmented PSR filter

Here, we change the state tracking layer and consider an augmented version that concatenates predictive states with a window of previous observations.

We provide a complete comparison of RPSP models using augmented states with observations for each environment in Figure 5.6. We compare with both joint optimization (**VRPG+obs**) and an alternating approach (**Alt+obs**). Augmented predictive states with a window of observations ($w = 2$) provide better results in particular for joint optimization. This extension might mitigate prediction errors, improving information carried over by the filtering states.

Predictive vs. latent states:

In the next experiment, we replace the PSR with a GRU that is ini-

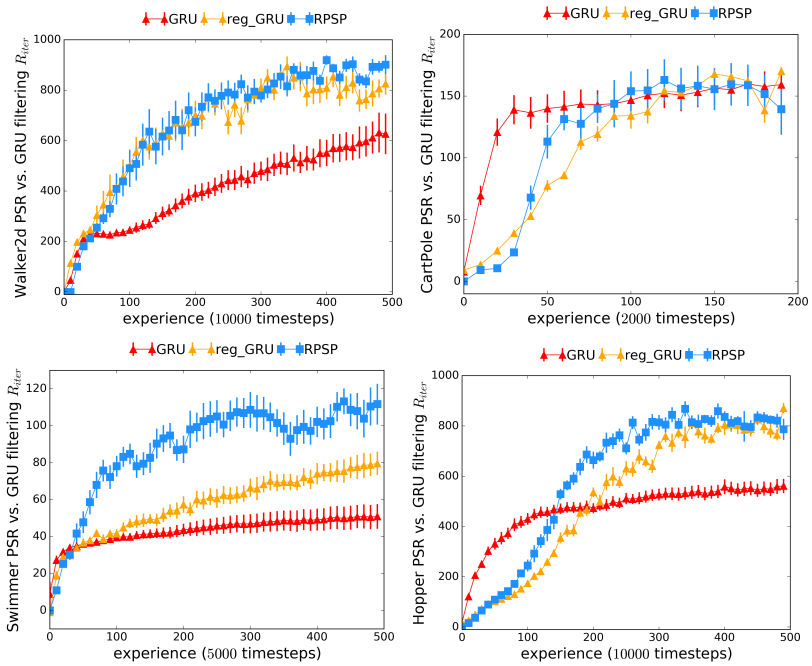


Figure 5.7: GRU vs. RPSP filter comparison for other Walker and CartPole environments. GRU filter without regularization loss (GRU:red), GRU filter with regularized predictive loss (reg_GRU: yellow), RPSP (RPSP:blue)

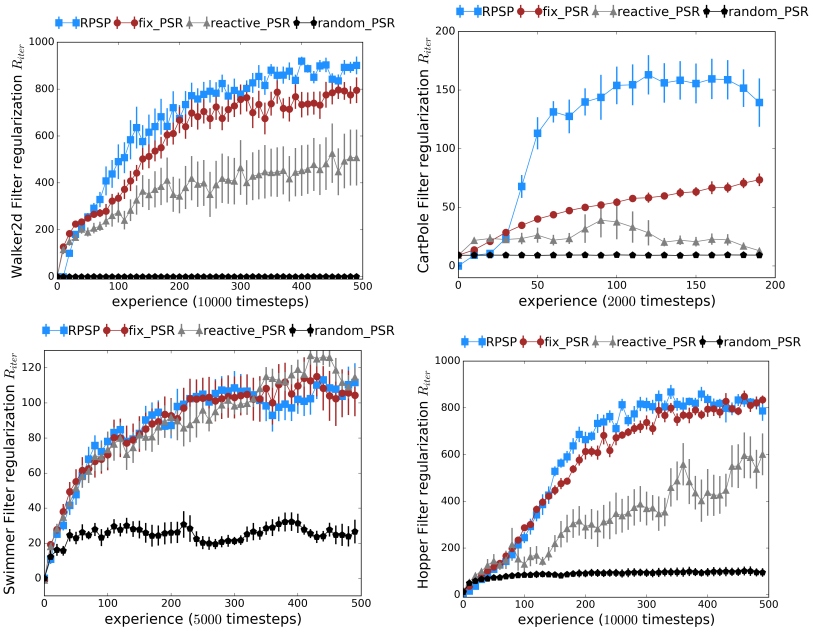


Figure 5.8: Predictive filter regularization effect for Walker2d, CartPole and Swimmer environments. RPSP with predictive regularization (RPSP:blue), RPSP with fixed PSR filter parameters (fix_PSR:red), RPSP without predictive regularization loss (reactive_PSR: grey).

tialized using backpropagation through time. This is analogous to the predictive state decoders proposed in (Venkatraman et al., 2017), where observation prediction loss is included when optimizing a GRU policy network (**reg_GRU**).[8] Figure 5.7 shows that a GRU model is inferior to a PSR model, where the initialization procedure is consistent and does not suffer from local optima.

[8] The results we report here are for the partially observable setting which is different from the reinforcement learning experiments in (Venkatraman et al., 2017).

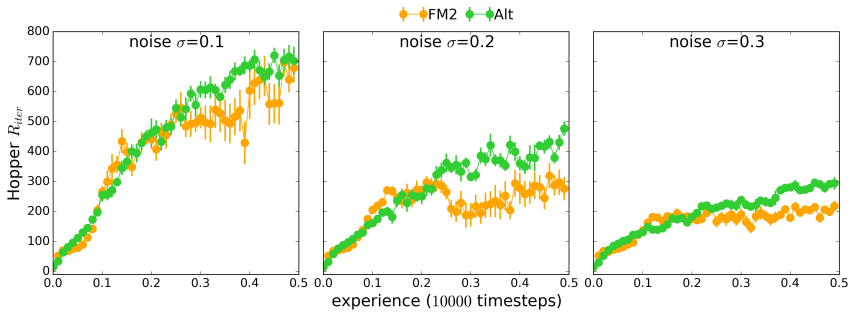


Figure 5.9: Empirical average return over 10 trials with a batch of $M = 10$ trajectories of $T = 1000$ time steps for Hopper. (Left to right) Robustness to observation Gaussian noise $\sigma = \{0.1, 0.2, 0.3\}$, best RPSP with alternate loss (Alt) and Finite Memory model (FM2).

Predictive regularization:
One component of RPSP is the inductive bias that predicting future observations helps find a meaningful state. In this experiment, we aim to test this assumption. We compare three variants of RPSP: one where the PSR is randomly initialized (**random_PSR**), another one where the PSR is fixed at the initial value (**fix_PSR**), and a third one where we train the RPSP network without prediction loss regularization (i.e. we set α_2 in (5.7)) to 0 (**reactive_PSR**).

Figure 5.8 demonstrates that these variants are inferior to our model, showing the importance of two-stage initialization, end-to-end training and observation prediction loss respectively.

Effect of observation noise:
We also investigated the effect of observation noise on the RPSP model and the competitive FM baseline by applying Gaussian noise of increasing variance to observations. Figure 5.9 shows that while FM was very competitive with RPSP in the noiseless case, RPSP has a clear advantage over FM in the case of mild noise. However, the performance gap vanishes if excessive noise is applied.

Finite Memory models:
Next, we present all finite memory models used as baselines. Figure 5.11 shows finite memory models with three different window sizes $w = 1, 2, 5$ for all environments. We report in the main comparison the best of each environment ($FM2$ for Walker, Swimmer, CartPole, and $FM1$ for Hopper).

GRU baselines:
In this section we report results obtained for RNN with GRUs using the best learning rate $\eta = 0.01$. Figure 5.10 shows the results using different number of hidden units $d = 16, 32, 64, 128$ for all the environments.

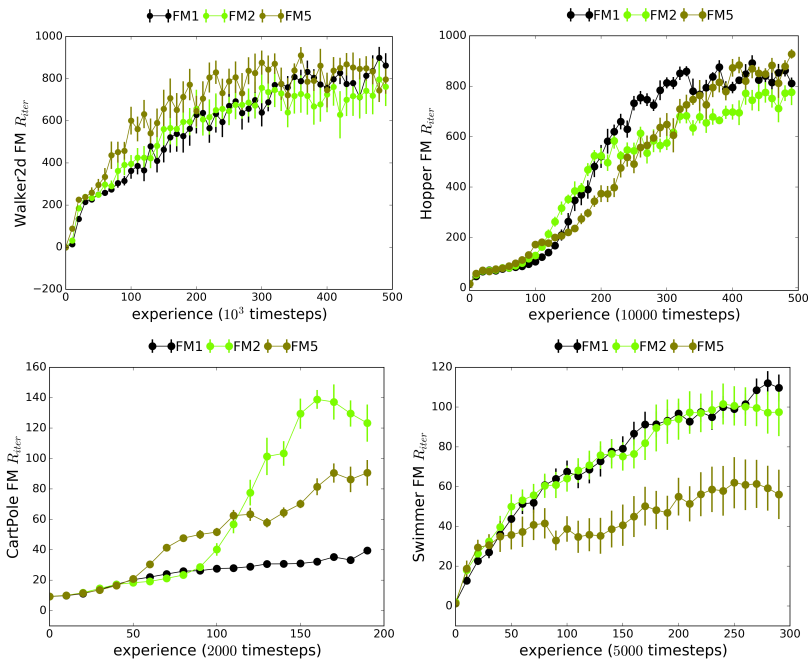


Figure 5.10: Empirical expected return using RNN with GRUs $d = 16$ (green), $d = 32$ (blue), $d = 64$ (red) and $d = 128$ (yellow) hidden units. (top-down left-right) Swimmer,Walker, Hopper and Cart-Pole.

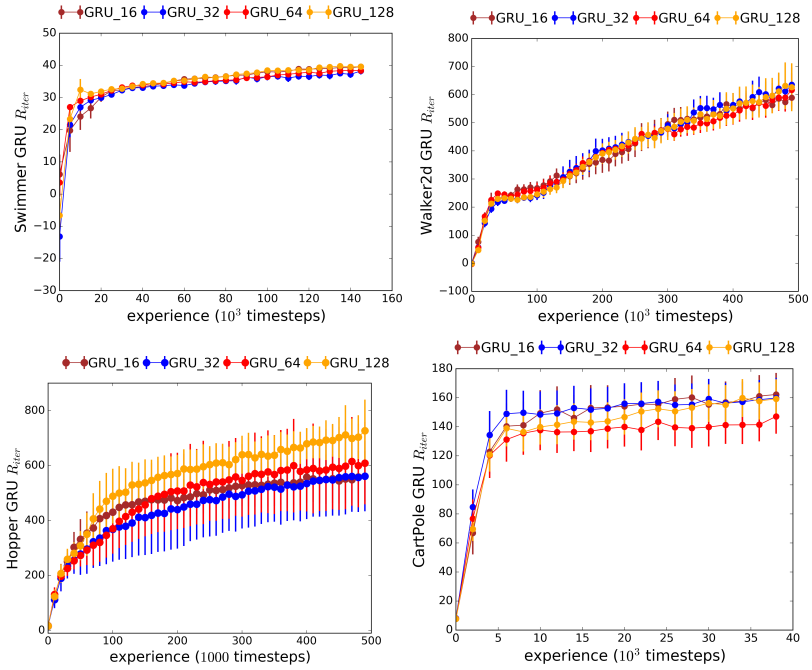


Figure 5.11: Empirical expected return using finite memory models of $w = 1$ (black), $w = 2$ (green), $w = 5$ (red) window sizes. (top-down left-right) Walker, Hopper, Cart-Pole, and Swimmer.

5.10 *Conclusions*

In this chapter, we proposed RPSP-networks, which combine ideas from predictive state representations with recurrent networks for reinforcement learning. This chapter poses an interesting research di-

rection that combines sample efficient method of moments with deep learning architectures. We make use of existing PSR learning algorithms to provide a statistically consistent initialization of the state tracking component, and propose gradient-based learning methods to minimize both expected return and prediction error over batches of trajectories. including GRUs, and finite memory models. We empirically show the efficacy of the proposed approach in terms of speed of convergence and overall expected return for different OpenAI Gym environments.

6
Conclusions

In this chapter, we review the most important contributions of this thesis in Section 6.1, and provide some insights on possible future research directions in Section 6.2.

6.1 Summary of contributions

In this thesis, we explored kernel-based methods and moment matching approaches to sequential prediction and planning problems. Sequential models are relevant to many fields in Computer Science; in particular, we focused on Natural Language Processing (NLP) and Robotics. We developed related machine learning algorithms for solving common problems in NLP and Robotics, despite the inherent core differences the two fields possess. NLP deals with large discrete observation spaces, while Robotics problems have a continuous nature typically describing the position or velocity of a robot.

In this thesis, we explore method of moments (MoM) to learn sequential models for sequence labeling tasks in NLP (§ 3), and we learn to predict and plan in continuous sequential systems in Robotics (§ 5). We further explore kernel approaches to design efficient algorithms for planning in Robotics, we develop a robot motion planner algorithm for trajectory optimization in reproducing kernel Hilbert spaces (§ 4), and we make use of kernel based representations to model and plan a robot's pose in a partially observable environment (§ 5).

First, we explored sequential models in the *latent variable learning* setting, where the latent states are interpretable and can provide additional information about the observations, in the form of labels. We introduced an efficient sequence labeling method for generative models in Chapter 3. We extended MoM techniques to semi-supervised learning settings, where we learn a sequence labeling task using contextual information from a set of *anchor* observations to disambiguate state. We further extended anchor learning to a weakly supervised setting by designing an algorithm that is able to handle small inputs

of labeled data and large amounts of unlabeled data. We provided an extension of the algorithm to log-linear models by considering a continuous feature observation space. The proposed method outperformed other supervised and semi-supervised methods. We reported results on POS-tagging tasks with small supervision, in a scarcely annotated language (Malagasy), and for Twitter (§ 3). This proposed approach was particularly well suited for learning with large amounts of unlabeled data, under weak supervision, while training an order of magnitude faster than any previous work.

In Robotics as well, we make use of sequential models to predict continuous observations in Chapter 5. We exploit another variant of MoM approaches, *spectral learning*, where the latent space is never explicitly recovered; instead it is used to increase the expressiveness of the model. We rely on kernel approaches to learn a continuous non-linear observation mapping and exploit linear sequential predictive models in kernel spaces using predictive state representations (PSRs). We combined predictive models with planning problems for Robotics, in Chapter 5. We make use of models of the system to improve the performance of planning algorithms. We proposed a novel class of policies for planning under uncertainty, where the latent space is related to a belief over the system state. We introduced recurrent predictive state policies (RPSP) networks, by combining PSRs with recurrent networks. We exploited existing PSR learning algorithms to serve as a recurrent state filter of the network while leveraging recent advances in deep reinforcement learning techniques to optimize a policy. This work provided the means to have a statistically consistent initialization of the state tracking component using current moment-based approaches (Hefny et al., 2017b). Lastly, we provided a gradient-based learning method to minimize both expected return and prediction error, providing a combined form of updating the predictive model and the learned policy end-to-end. We empirically analyzed different RPSP variants against other recurrent and predictive models. We showed the efficacy of the proposed approach in terms of speed of convergence and overall expected return for different partially observable environments using an OpenAI Gym simulator.

Finally, we combined kernel-based approaches with continuous planning problems in Robotics in Chapter 4. In this chapter, we departed from sequential systems and MoM learning and focused on kernel based methods for robot planning problems. We introduced a kernel approach to to robot motion planning using a gradient based optimization algorithm. We framed trajectory planning as function optimization in a reproducing kernel Hilbert space (RKHS), where we considered trajectories as elements on a vector valued RKHS. We further provided an efficient algorithm where the norm induced by the

kernel defined a good metric to the optimization problem and could be associated with different notions of smoothness, including commonly-used variants as special cases (velocity, acceleration penalization). We demonstrated the efficiency of this functional gradient trajectory optimization in RKHSs, compared with a common optimization algorithm (CHOMP) (Zucker et al., 2013). We showed that the proposed method benefited from a low-dimensional trajectory parametrization that was fast to compute. The inherent smoothness of the RKHS allowed us to achieve faster convergence, by taking larger steps without breaking smoothness. We extended planning to vector valued spaces, important to model robot joint interactions, and further provided forms of incorporating kinematic and dynamic constraints along the trajectory. This work presents an important step in exploring RKHSs for motion planning. We described trajectory optimization under the light of reproducing kernels, which we hope can leverage the knowledge used in kernel based machine learning to develop better motion planning methods.

6.2 *Future directions*

In this section, we provide a few possible future research directions and discuss some open research questions pertaining to this thesis.

6.2.1 *Extend anchor learning to continuous observation spaces*

In Chapter 3 we propose a moment-based learning method for latent variable models that relies on anchor observations. These observations provide an unambiguous map between certain observation types, *i.e.*, elements in a finite dictionary to each state variable (or labels in this case). Although we present algorithms that are able to handle continuous observation representations, we only consider anchor mappings to certain discrete observation types. A possible more general form of anchor learning could extend this notion to certain events in time that are associated with a given hidden state variable. This insight would allow for continuous representations and consider anchors as observation "tokens" instead of types. We could think of an extension of the proposed algorithm to capture information about anchor events in time and their context. Hidden states would still correspond to a finite set but the observations may be continuous and the association is done in time (anchors in time). For each hidden state in Eq. 3.10 we can model the expectation in time over all contexts of every instance that is unambiguously associated with that particular state. "Would correlations of these anchor tokens with their contextual information be sufficient to learn its latent variables for any particular time?" is

an open question that naturally arises from this interpretation. The anchor learning algorithm would be a suitable algorithm for dealing with anchor tokens, since it provides an efficient form of looking at these correlations.

6.2.2 *The effect of impure anchors*

Another possible future direction of research relates to the theoretical analysis of impure anchors. In Chapter 3, we use anchor words as placeholder for latent variables. This substitution is exact in the case of pure anchors, meaning anchor words that only occur with a particular state. In this case alone, we may exchange state contextual information with anchor words contextual information. Nevertheless, form a practical point of view finding such observations be difficult and in some cases they may only occur very sparsely, leading to poor evaluation of the contexts. Therefore, it would be useful to consider relaxing this notion, as we empirically observe, and understand from a theoretical perspective what are the trade-offs between purity and how well we may estimate the hidden states or in turn its context. Work on hyperspectral unmixing in the remote sensing community provides some analysis towards this direction (Bioucas-Dias et al., 2012), in particular when anchors, or end-members as it is denoted in this community, lie in the facet of the simplex (occur with just a few hidden variables) or when they occur with a given hidden variable with high probability. It still remains an open question how well the each of these variants would perform as substitutes for hidden variables in the context of the proposed approach in Chapter 3, and also in conjunction with semi-supervised information.

6.2.3 *Combining randomized planners with kernel representations*

RKHSs provide an expressive and natural metric for representing and optimizing trajectories, as proposed in Chapter 4. Another line of work in robot motion planning finds trajectories based on sampling approaches. Methods such as rapidly exploring random trees (RRTs) and probabilistic roadmaps (PRMs) explore the configuration space of the robot by building a graph from sampled configurations, and then searching for a viable path within this graph. This family of methods provide an efficient form of obtaining viable trajectories but lack the optimality provided by trajectory optimization algorithms, such as the one proposed in Chapter 4. An interesting possible line of work could try to bring together these two families of motion planning, by leveraging expressiveness of RKHSs with efficiency of randomized approaches. Francis et al. (2017) provides some initial step in this direction, but this still remains an open research direction.

6.2.4 *Exploring natural gradient approaches for RPSPs*

Current natural gradient descent algorithms, such as the one used in the alternating optimization approach, in Chapter 5, require the computation of the Fisher information matrix as a quadratic approximation of the KL divergence constraint in Eq. 5.9. Schulman et al. (2015) provides an efficient form of computing this constraint in terms of an approximate Hessian-vector product, but this step combined with the line search, required to compute the conjugate gradient step, is still relatively slower compared with the REINFORCE approach. As a possible future research direction, it would be of interest to take advantage of the steady changing monotonically improving policy parameters with natural gradient methods, while reducing its computation time.

6.2.5 *Exploit predictive models for improved real robot experiments*

Model-based learning offers a sample efficient way of learning a policy using simulated experience gathered from the model. The most common difficulty in model-based approaches lies in the model estimation itself. Learning an accurate model is sometimes harder than directly optimizing the policy parameters. PSRs, however, grant a computationally effective form of estimating a model of the environment solely based on observations. The proposed work on policy learning with PSRs (RPSPs), in Chapter 5, learns both simultaneously a policy and a predictive model from scratch. It would be interesting to extend PSRs to learn in a real robot experiment, for instance, study how we could use the learned model to improve real robot sample efficiency, either as an efficient experience generator for learning a real robot policy, or as an initial policy. Analogous to the setting of transfer learning, it would be interesting to learn how we can learn RPSPs using simulated data such that the model would perform well with just a few iterations of refinement in the real robot (Finn et al., 2017; Rusu et al., 2016). Alternatively, we may learn a mixture of experts Shazeer et al., 2017; Eigen et al., 2013 both in real robot experiments and simulated environments and optimize their combination. Other forms of improvement can be achieved by closing the loop between simulations and real-robot trials (Christiano et al., 2016), by learning a policy of the residual error between the predictive simulations and real-robot samples. This policy class could be an RPSP, whose prediction error is between the predicted simulation and the real robot, or the prediction of the residual. This however would not improve the sample complexity of the real experience, which may suggest that a combination of the two approaches, transfer learning with predicting the residuals, could provide an efficient and robust approach.

6.2.6 *Connections with deep learning*

Overall, the method of moments provides a statistically consistent alternative to learning sequential systems, however when compared with current state-of-the-art models, in particular using deep-learning approaches they have not been as successful as desired. Spectral methods possess some disadvantages when compared with deep neural networks: they assume a fixed filtering function, which may fail when modeling more complex, non-stationary systems. Spectral methods are also very sensitive, meaning performance degrades quickly if we fall out of the learned region, making them more vulnerable to model mismatch. It is also worth noting that although both type of methods are demanding in terms of samples, most spectral learning variants have been proposed for the unsupervised setting, while deep networks have been mostly used in a supervised way.

In Robotics, Sun et al. (2016a) provides an approach with promising results using spectral learning for time-variant systems, however comparisons to deep learning approaches have not been done in this case. In NLP, on the other hand, deep learning methods achieve state of the art performance in different tasks. Although many of these improvements may come from highly optimized parameter tuning (Dhillon et al., 2015a; Stratos et al., 2015) compared with spectral learning approaches, they still provide more flexibility, by allowing modular characteristics such as attention mechanisms over the entire sequence (Bahdanau et al., 2015), gating mechanisms and memory (Hochreiter et al., 1995; Cho et al., 2014a), and more flexible filtering functions. Exploring this additional flexibility in method of moments provides very promising future research directions. Also in kernel approaches there has been further work trying to combine kernel methods and deep architectures, by improving on one side theoretical understanding of deep methods (Belkin et al., 2018), and on the other side leveraging the expressive power and scalability of deep architecture into kernel learning (Zhuang et al., 2011; Cho et al., 2009; Wilson et al., 2016; Song et al., 2018).

List of Figures

List of Tables

Bibliography

[Abb+08] Pieter Abbeel, Adam Coates, Timothy Hunter, and Andrew Y. Ng. "Autonomous Autorotation of an RC Helicopter". In: *Experimental Robotics, The Eleventh International Symposium, ISER 2008, July 13-16, 2008, Athens, Greece*. 2008, pp. 385–394.

[Aga+97] Pankaj Agarwal, Jean claude Latombe, Rajeev Motwani, and Prabhakar Raghavan. "Nonholonomic Path Planning for Pushing a Disk Among Obstacles". In: *icra*. 1997.

[Aga+99] Pankaj K Agarwal, Boris Aronov, and Micha Sharir. "Motion planning for a convex polygon in a polygonal environment". In: *Discrete & Computational Geometry* 22.2 (1999), pp. 201–221.

[Ale+68] V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva. "Stochastic Optimization". In: *Engineering Cybernetics* 5 (1968), pp. 11–16.

[Alt+06] Yasemin Altun and Alexander J. Smola. "Unifying Divergence Minimization and Statistical Inference Via Convex Duality". In: *Learning Theory, 19th Annual Conference on Learning Theory, COLT*. 2006.

[Ama98] S. Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Comput.* 10.2 (1998), pp. 251–276.

[Ana+12a] A. Anandkumar, D. Hsu, and S.M. Kakade. "A Method of Moments for Mixture Models and Hidden Markov Models". In: *COLT* (2012).

[Ana+12b] Anima Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. "Tensor decompositions for learning latent variable models". In: *arXiv preprint arXiv:1210.7559* (2012).

[Ana+14] Animashree Anandkumar, Rong Ge, and Majid Janzamin. "Guaranteed Non-Orthogonal Tensor Decomposition via Alternating Rank-1 Updates". In: *CoRR* (2014).

[Ana+15] Animashree Anandkumar, Daniel Hsu, Majid Janzamin, and Sham Kakade. "When Are Overcomplete Topic Models Identifiable? Uniqueness of Tensor Tucker Decompositions with Structured Sparsity". In: *J. Mach. Learn. Res.* 16.1 (Jan. 2015), pp. 2643–2694. ISSN: 1532-4435.

[Aro+98] Boris Aronov, Mark de Berg, A. Frank van der Stappen, Petr Švestka, and Jules Vleugels. "Motion Planning for Multiple Robots". In: *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*. SCG '98. Minneapolis, Minnesota, USA: ACM, 1998, pp. 374–382. ISBN: 0-89791-973-4.

[Aro50] N. Aronszajn. "Theory of Reproducing Kernels". In: *Transactions of the American Mathematical Society*. 1950.

[Aro+13] Sanjeev Arora, Rong Ge, Yoni Halpern, David Mimno, David Sontag Ankur Moitra, Yichen Wu, and Michael Zhu. "A Practical Algorithm for Topic Modeling with Provable Guarantees". In: *Proc. of International Conference of Machine Learning*. 2013.

[Aro+12] Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra. "Computing a nonnegative matrix factorization - provably". In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*. 2012.

[Aro+01] Sanjeev Arora and Ravi Kannan. "Learning mixtures of arbitrary gaussians". In: *STOC*. ACM, 2001, pp. 247–257.

[Ary+94] Sunil Arya, David M Mount, Nathan Netanyahu, Ruth Silverman, and Angela Y Wu. "An optimal algorithm for approximate nearest neighbor searching in fixed dimensions". In: 1994.

[Atk98] C. G. Atkeson. "Nonparametric Model-Based Reinforcement Learning". In: *Advances in Neural Information Processing Systems - 10*. MIT Press, 1998, pp. 1008–1014.

[Atk+97] C. G. Atkeson and J. C. Santamaria. "A comparison of direct and model-based reinforcement learning". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. 1997, 3557–3564 vol.4.

[Azi+16] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. "Reinforcement Learning of POMDP's using Spectral Methods". In: *CoRR* abs/1602.07764 (2016).

[Bae+93] Ricardo A BaezaYates, Joseph C Culberson, and Gregory JE Rawlins. "Searching in the plane". In: *Information and computation* 106.2 (1993), pp. 234–252.

[Bag+01] Drew Bagnell and Jeff Schneider. "Autonomous helicopter control using reinforcement learning policy search methods". In: *Internatinal Conference on Robotics and Automation*. 2001.

[Bah+15] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. "End-to-End Attention-based Large Vocabulary Speech Recognition". In: *CoRR* abs/1508.04395 (2015).

[Bai+13a] R. Bailly, X. Carreras, F. M. Luque, and A. Quattoni. "Unsupervised Spectral Learning of WCFG as Low-rank Matrix Completion". In: *EMNLP*. 2013.

[Bai+13b] Raphaël Bailly, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. "Unsupervised Spectral Learning of WCFG as Low-rank Matrix Completion". In: *Proc. of Empirical Methods in Natural Language Processing*. 2013, pp. 624–635.

[Bal+12a] B. Balle and M. Mohri. "Spectral Learning of General Weighted Automata via Constrained Matrix Completion". In: *Neural Information Processing Systems Foundation*. 2012.

[Bal13] Borja Balle. "Learning finite-state machines: algorithmic and statistical aspects". PhD thesis. PhD thesis, Universitat Politecnica de Catalunya, 2013.

[Bal+11] Borja Balle, Ariadna Quattoni, and Xavier Carreras. "A Spectral Learning Algorithm for Finite State Transducers". In: (2011).

[Bal+12b] Borja Balle, Ariadna Quattoni, and Xavier Carreras. "Local loss optimization in operator models: A new insight into spectral learning". In: *ICML* (2012).

[Bal+12c] Borja Balle and Mehryar Mohri. "Spectral learning of general weighted automata via constrained matrix completion". In: *Advances in Neural Information Processing Systems*. 2012, pp. 2168–2176.

[Bau+70] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". English. In: *The Annals of Mathematical Statistics* 41.1 (1970), pp. 164–171. ISSN: 00034851.

[Bax+01] Jonathan Baxter and Peter L. Bartlett. "Infinite-horizon policy-gradient estimation". In: *Journal of Artificial Intelligence Research* (2001).

[Bei+00] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. "Learning Functions Represented As Multiplicity Automata". In: *J. ACM* 47.3 (May 2000), pp. 506–530. ISSN: 0004-5411.

[Bel+18] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. "To understand deep learning we need to understand kernel learning". In: *CoRR* abs/1802.01396 (2018).

[Bel+10] Mikhail Belkin and Kaushik Sinha. "Toward Learning Gaussian Mixtures with Arbitrary Separation". In: *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*. 2010, pp. 407–419.

[Ben09] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends® in Machine Learning* 2.1 (2009), pp. 1–127. ISSN: 1935-8237.

[Ber+11] J. van den Berg, P. Abbeel, and K. Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information." In: *International Journal Robotics Research* (2011).

[BK+10] Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. "Painless unsupervised learning with features". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. ACL. 2010, pp. 582–590.

[Ber+10] Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. "Painless Unsupervised Learning with Features". In: *Human Language Technologies: Conference of the North American Association of Computational Linguistics*. 2010.

[Ber+79] Jean Berstel and Luc Boasson. "Transductions and context-free languages". In: *Ed. Teubner* (1979), pp. 1–278.

[BD+12] José M. Bioucas-Dias, Antonio J. Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul D. Gader, and Jocelyn Chanussot. "Hyperspectral Unmixing Overview: Geometrical, Statistical, and Sparse Regression-Based Approaches". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5 (2012), pp. 354–379.

[Bla+98] A. Blake and M. Isard. *Active Contours*. Springer-Verlag New York, Inc., 1998.

[Bla+08] M. B. Blaschko and C. H. Lampert. "Correlational spectral clustering". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8.

[Ble+03] D.M. Blei, A.Y. Ng, and M. I. Jordan. "Latent dirichlet allocation". In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022.

[Boj+16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. "End to End Learning for Self-Driving Cars." In: *CoRR* abs/1604.07316 (2016).

[Boo+11a] Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. "Closing the learning-planning loop with predictive state representations". In: *IJRR* 30.7 (2011).

[Boo+11b] Byron Boots, Sajid Siddiqi, and Geoffrey Gordon. "Closing the Learning Planning Loop with Predictive State Representations". In: *I. J. Robotic Research*. Vol. 30. 2011, pp. 954–956.

[Boo+13a] Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. "Hilbert Space Embeddings of Predictive State Representations". In: *UAI*. 2013.

[Boo+13b] Byron Boots, Arthur Gretton, and Geoffrey J. Gordon. "Hilbert Space Embeddings of Predictive State Representations". In: *Proc. 29th Intl. Conf. on Uncertainty in Artificial Intelligence (UAI)*. 2013.

[Boo+10] Byron Boots and Geoffrey J Gordon. "Predictive State Temporal Difference Learning". In: *NIPS*. 2010.

[Bra+11] S. R. K. Branavan, David Silver, and Regina Barzilay. "Learning to Win by Reading Manuals in a Monte-Carlo Framework". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*. HLT '11. Portland, Oregon: Association for Computational Linguistics, 2011, pp. 268–277. ISBN: 978-1-932432-87-9.

[Bro02] Khatib O. Brock O. "Elastic strips: A framework for motion generation in human environments". In: *International Journal Robotics Research*. 2002.

[Bro+92] Peter F. Brown, Peter V. de Souza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. "Class-based *N*-gram Models of Natural Language". In: *Computational Linguistics* 18.4 (1992), pp. 467–479.

[Bru+08] S. Charles Brubaker and Santosh Vempala. "Isotropic PCA and Affine-Invariant Clustering". In: *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. 2008, pp. 551–560.

[Car+71] J. W. Carlyle and A. Paz. "Realizations by Stochastic Finite Automata". In: *J. Comput. Syst. Sci.* 5.1 (1971).

[Cha+14a] Arun Chaganty and Percy Liang. "Estimating Latent-Variable Graphical Models using Moments and Likelihoods". In: *ICML*. 2014.

[Cha+14b] Arun T. Chaganty and Percy Liang. "Estimating Latent-Variable Graphical Models using Moments and Likelihoods". In: *Proc. of International Conference on Machine Learning*. 2014.

[Cha96] JT Chang. "Full reconstruction of Markov models on evolutionary trees: identifiability and consistency". In: *Mathematical biosciences* 137.1 (1996), pp. 51–73. ISSN: 0025-5564.

[Cha+15] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. "Learning to Search Better Than Your Teacher". In: *CoRR* abs/1502.02206 (2015).

[Cha+14c] Kai-Wei Chang, Scott Wen-tau Yih, Bishan Yang, and Chris Meek. "Typed Tensor Decomposition of Knowledge Bases for Relation Extraction". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. ACL – Association for Computational Linguistics, 2014.

[Cha+08] Kamalika Chaudhuri and Satish Rao. "Learning Mixtures of Product Distributions Using Correlations and Independence". In: *21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008*. 2008, pp. 9–20.

[Chi+07] Tsung-Han Chiang, Mehmet Serkan Apaydin, Douglas L. Brutlag, David Hsu, and Jean-Claude Latombe. "Using Stochastic Roadmap Simulation to Predict Experimental Quantities in Protein Folding Kinetics: Folding Rates and Phi-Values". In: *Journal of Computational Biology* 14.5 (2007), pp. 578–593.

[Cho+14a] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[Cho+14b] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Empirical Methods in Natural Language Processing, EMNLP*. 2014, pp. 1724–1734.

[Cho+09] Youngmin Cho and Lawrence K. Saul. "Kernel Methods for Deep Learning". In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta. Curran Associates, Inc., 2009, pp. 342–350.

[Cho+17] Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. *Coarse-to-Fine Question Answering for Long Documents*. Jan. 2017.

[Chr+16] Paul F. Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba. "Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model". In: *CoRR* abs/1610.03518 (2016).

[Coh+13a] S. B. Cohen, G. Satta, and M. Collins. "Approximate PCFG Parsing Using Tensor Decomposition". In: *Proceedings of NAACL*. 2013.

[Coh+12] S. B. Cohen, K. Stratos, M. Collins, D. P. Foster, and L. Ungar. "Spectral Learning of Latent-Variable PCFGs". In: *ACL*. 2012.

[Coh+14] Shay B. Cohen and Michael Collins. "A Provably Correct Learning Algorithm for Latent-Variable PCFGs". In: *Proc. of Association for Computational Linguistics*. 2014.

[Coh+13b] Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle H. Ungar. "Experiments with Spectral Learning of Latent-Variable PCFGs". In: *ACL*. 2013, pp. 148–157.

[Coh+13c] Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle Ungar. "Experiments with Spectral Learning of Latent-Variable PCFGs". In: *Proc. of North American Association of Computational Linguistics*. 2013.

[Com+08] Pierre Comon, Gene Golub, Lek-Heng Lim, and Bernard Mourrain. "Symmetric Tensors and Symmetric Tensor Rank". In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008), pp. 1254–1279.

[Con85] John B. Conway. *A course in functional analysis*. Springer, 1985.

[Cut+90] Mark R. Cutkosky and Robert D. Howe. "Dextrous Robot Hands". In: ed. by S. T. Venkataraman and T. Iberall. New York, NY, USA: Springer-Verlag New York, Inc., 1990. Chap. Human Grasp Choice and Robotic Grasp Analysis, pp. 5–31. ISBN: 0-387-97190-4.

[Das99] Sanjoy Dasgupta. *Learning Mixtures of Gaussians*. Tech. rep. UCB/CSD-99-1047. EECS Department, University of California, Berkeley, 1999.

[Das+07] Sanjoy Dasgupta and Leonard J. Schulman. "A Probabilistic Analysis of EM for Mixtures of Separated, Spherical Gaussians". In: *Journal of Machine Learning Research* 8 (2007), pp. 203–226.

[Das+00] Sanjoy Dasgupta and Leonard J. Schulman. "A Two-Round Variant of EM for Gaussian Mixtures". In: *UAI*. Morgan Kaufmann, 2000, pp. 152–159.

[Day+97] Peter Dayan and Geoffrey E. Hinton. "Using Expectation-Maximization for Reinforcement Learning". In: *Neural Computation* 9.2 (1997), pp. 271–278.

[Dee+90a] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. "Indexing by latent semantic analysis". In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407.

[Dee+90b] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. "Indexing by latent semantic analysis". In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407.

[Dei+13] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A survey on policy search for robotics". In: (2013).

[Dei+11] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *In Proceedings of the International Conference on Machine Learning*. 2011.

[Dem+77] A.P. Dempster, N.M. Laird, and D.B. Rubin. "Maximum likelihood from incomplete data via the E.M. algorithm". In: *JRSS* (1977), pp. 1–38.

[Dhi+11] P. S. Dhillon, D. Foster, and L. Ungar. "Multiview learning of word embeddings via cca". In: *NIPS*. 2011.

[Dhi+15a] Paramveer Dhillon, Dean Foster, and Lyle Ungar. "Eigenwords: Spectral Word Embeddings". In: *JMLR* (2015).

[Dhi+15b] Paramveer S. Dhillon, Dean P. Foster, and Lyle H. Ungar. "Eigenwords: Spectral Word Embeddings". In: *Journal of Machine Learning Research* 16 (2015), pp. 3035–3078.

[Don+03] David Donoho and Victoria Stodden. "When Does Non-Negative Matrix Factorization Give Correct Decomposition into Parts?" In: MIT Press, 2003, p. 2004.

[Dow+17] Carlton Downey, Ahmed Hefny, and Geoffrey J Gordon. "Practical Learning of Predictive State Representations". In: *arXiv:1702.04121*. 2017.

[Dra+14] A. Dragan and S. Srinivasa. "Familiarization to Robot Motion". In: *International Conference on Human-Robot Interaction (HRI)* (2014).

[Dua+16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. 2016, pp. 1329–1338.

[Dup+05] P. Dupont, F. Denis, and Y. Esposito. "Links Between Probabilistic Automata and Hidden Markov Models: Probability Distributions, Learning Models and Induction Algorithms". In: *Pattern Recogn.* 38.9 (Sept. 2005), pp. 1349–1371. ISSN: 0031-3203.

[Eig+13] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. "Learning Factored Representations in a Deep Mixture of Experts". In: *CoRR* abs/1312.4314 (2013).

[Fin+17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *CoRR* abs/1703.03400 (2017).

[Fis12] R. A. Fisher. "On an absolute criterion for fitting frequency curves". In: *Messenger of Mathematics* 41 (1912), pp. 155–160.

[Fli74] Michael Fliess. "Matrices de Hankel". In: *Journal Math. Pures Appl.* 53.9 (1974), pp. 197–222.

[Fra+17] Gilad Francis, Lionel Ott, and Fabio Ramos. "Stochastic Functional Gradient Path Planning in Occupancy Maps". In: *arXiv:1705.05987* (May 2017).

[Fuk+13a] Kenji Fukumizu, Le Song, and Arthur Gretton. "Kernel Bayes' Rule: Bayesian Inference with Positive Definite Kernels". In: *Journal of Machine Learning Research* 14 (2013), pp. 3753–3783.

[Fuk+13b] Kenji Fukumizu, Le Song, and Arthur Gretton. "Kernel Bayes' rule: Bayesian inference with positive definite kernels". In: *Journal of Machine Learning Research* 14.1 (2013), pp. 3753–3783.

[Fuk+13c] Kenji Fukumizu, Le Song, and Arthur Gretton. "Kernel Bayes' rule: Bayesian inference with positive definite kernels". In: *Journal of Machine Learning Research* 14.1 (2013).

[Gar+13] Dan Garrette, Jason Mielens, and Jason Baldridge. "Real-World Semi-Supervised Learning of POS-Taggers for Low-Resource Languages". In: *Proc. of Association for Computational Linguistics*. 2013.

[Gen+06] Tao Geng, Bernd Porr, and Florentin Wörgötter. "Fast biped walking with a reflexive controller and real-time policy searching". In: *Advances in Neural Information Processing Systems*. 2006, pp. 427–434.

[Gha+99] Zoubin Ghahramani and Matthew J. Beal. "Variational Inference for Bayesian Mixtures of Factor Analysers". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999, pp. 449–455.

[Gim+11] Gimpel, Schneider, OConnor, Das, Mills, Eisenstein, Heilman, Yogatama, Flanigan, and Smith. "Part-of-speech Tagging for Twitter: Annotation, Features, and Experiments". In: *ACL*. 2011.

[Gly90] Peter W. Glynn. "Likelihood ratio gradient estimation for stochastic systems". In: *Communications of the ACM* 33.10 (1990), pp. 75–84.

[God60] V. P. Godambe. "An Optimum Property of Regular Maximum Likelihood Estimation". In: *Ann. Math. Statist.* 31.4 (Dec. 1960), pp. 1208–1211.

[Gol+12] Yoav Goldberg and Joakim Nivre. "A Dynamic Oracle for Arc-Eager Dependency Parsing". In: *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India*. 2012, pp. 959–976.

[Goo+16] James Goodman, Andreas Vlachos, and Jason Naradowsky. "Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. 2016.

[Gor99] Geoffrey J. Gordon. "Regret Bounds for Prediction Problems". In: *Proceedings of the Twelfth Annual Conference on Computational Learning Theory, COLT 1999, Santa Cruz, CA, USA, July 7-9, 1999*. 1999, pp. 29–40.

[Gre+01] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. 2001.

[Gri+14] Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. "Don't Until the Final Verb Wait: Reinforcement Learning for Simultaneous Machine Translation". In: *Empirical Methods in Natural Language Processing*. Doha, Qatar, 2014.

[Gul+94] Vijaykumar Gullapalli, Andrew G. Barto, and Roderic A. Grupen. "Learning Admittance Mappings for Force-Guided Assembly". In: *Proceedings of the 1994 International Conference on Robotics and Automation, San Diego, CA, USA, May 1994*. 1994, pp. 2633–2638.

[Haa+16] Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. "Backprop KF: Learning Discriminative Deterministic State Estimators". In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016.

[Hag+06] Aria Haghighi and Dan Klein. "Prototype-Driven Learning for Sequence Models". In: *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*. 2006.

[Hal+11] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Rev.* (2011).

[Ham+14] William Hamilton, Mahdi Milani Fard, and Joelle Pineau. "Efficient Learning and Planning with Compressed Predictive States". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 3395–3439.

[He+16] Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tieyan Liu, and Wei-Ying Ma. "Dual Learning for Machine Translation". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 820–828.

[Hef+17a] Ahmed Hefny, Carlton Downey, Zita Marinho, Wen Sun, Siddhartha Srinivasa, and Geoffrey J. Gordon. "Predictive State Models for Prediction and Control in Partially Observable Environments". In: *Conference on Robot Learning (CoRL)*. 2017.

[Hef+17b] Ahmed Hefny, Carlton Downey, and Geoffrey J Gordon. "Supervised Learning for Controlled Dynamical System Learning". In: *arXiv:1702.03537*. 2017.

[Hef+15a] Ahmed Hefny, Carlton Downey, and Geoffrey J Gordon. "Supervised Learning for Dynamical System Learning". In: *NIPS*. 2015.

[Hef+15b] Ahmed Hefny, Carlton Downey, and Geoffrey J Gordon. "Supervised Learning for Dynamical System Learning". In: *Advances in Neural Information Processing Systems*. 2015.

[Hel65] Alex Heller. "On Stochastic Processes Derived From Markov Chains". In: *Ann. Math. Statist.* 36.4 (Aug. 1965), pp. 1286–1291.

[Hin+06] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667.

[Hit27] Frank L. Hitchcock. "The Expression of a Tensor or a Polyadic as a Sum of Products". In: *Journal of Mathematics and Physics* 6.1-4 (1927), pp. 164–189. ISSN: 1467-9590.

[Hoc+95] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: (1995).

[Hoc+97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), pp. 1735–1780.

[Hof+08] T. Hofmann, B. Schölkopf, and A. J. Smola. "Kernel methods in machine learning". In: *Annals of Statistics* (2008).

[Hof99] Thomas Hofmann. "Probabilistic Latent Semantic Analysis". In: *In Proc. of Uncertainty in Artificial Intelligence, UAI'99*. 1999, pp. 289–296.

[Hsu+09] D. Hsu, S. M. Kakade, and T. Zhang. "A Spectral Algorithm for Learning Hidden Markov Models". In: *COLT*. 2009.

[Hsu+12a] D. Hsu, S. M. Kakade, and T. Zhang. "A spectral algorithm for learning hidden Markov models". In: *JCSS* 78.5 (2012).

[Hsu+12b] Daniel Hsu, Sham M. Kakade, and Tong Zhang. "A spectral algorithm for learning hidden Markov models". In: *Journal of Computer and System Sciences* 78.5 (2012), pp. 1460–1480.

[Iof+15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[Ito92] Hisashi Ito. "An Algebraic Study of Discrete Stochastic Systems". Unpublished doctoral unpublished dissertation. Bunkyo-ku, Tokyo: University of Tokyo, 1992.

[Jae99] Herbert Jaeger. *Characterizing Distributions of Stochastic Processes By Linear Operators*. 1999.

[Jae00] Herbert Jaeger. "Observable Operator Models for Discrete Stochastic Time Series". In: *Neural Computation* 12.6 (June 2000), pp. 1371–1398.

[Jen06] J. Jensen. "Sur les fonctions convexes et les inégalités entre les valeurs moyennes". In: *Acta Mathematica* 30 (1906), pp. 175–193.

[Jor+99] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. "An Introduction to Variational Methods for Graphical Models". In: *Machine Learning* 37.2 (1999), pp. 183–233.

[Kae+98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and Acting in Partially Observable Stochastic Domains". In: *Artif. Intell.* 101.1-2 (May 1998), pp. 99–134. ISSN: 0004-3702.

[Kal+11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. "Stochastic Trajectory Optimization for Motion Planning". In: *IEEE International Conference on Robotics and Automation*. 2011.

[Kan+08] Ravindran Kannan, Hadi Salmasian, and Santosh Vempala. "The Spectral Method for General Mixture Models". In: *SIAM J. Comput.* 38.3 (2008), pp. 1141–1156.

[Kap05] Hilbert J Kappen. "Path integrals and symmetry breaking for optimal control theory". In: *Journal of statistical mechanics: theory and experiment* 2005.11 (2005), P11011.

[Kav+94] Lydia Kavraki, Petr Svestka, Jean Latombe, and Mark Overmars. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*. Tech. rep. Stanford, CA, USA, 1994.

[Kim+71] G. S. Kimeldorf and G. Wahba. "Some Results on Tchebycheffian Spline Functions". In: *Journal of Mathematical Analysis and Applications*. 1971.

[Kir12] D.E. Kirk. *Optimal Control Theory: An Introduction*. Dover Books on Electrical Engineering. Dover Publications, 2012. ISBN: 9780486135076.

[Kob+10] Jens Kober, Betty Mohler, and Jan Peters. "From Motor Learning to Interaction Learning in Robots". In: ed. by Olivier Sigaud and Jan Peters. Vol. 264. Studies in Computational Intelligence. Springer Verlag, 2010. Chap. Imitation and Reinforcement Learning for Motor Primitives with Perceptual Coupling, pp. 209–225.

[Kob+09] Jens Kober and Jan Peters. "Policy Search for Motor Primitives in Robotics". In: *Advances in Neural Information Processing Systems 22 (NIPS 2008)*. Cambridge, MA: MIT Press, 2009, pp. 849–856.

[Koh+04] Nate Kohl and Peter Stone. "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2004.

[Kre78] E. Kreyszig. *Introductory Functional Analysis with Applications*. Krieger Publishing Company, 1978.

[Kru77] Joseph B. Kruskal. "Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics". In: *Journal of Linear Algebra and Applications* 18.2 (1977), pp. 95–138. ISSN: 0024-3795 (print), 1873-1856 (electronic).

[Kul+15a] Volodymyr Kuleshov, Arun Tejasvi Chaganty, and Percy Liang. "Simultaneous diagonalization: the asymmetric, low-rank, and noisy settings". In: *CoRR* abs/1501.06318 (2015).

[Kul+15b] Alex Kulesza, Nan Jiang, and Satinder Singh. "Spectral Learning of Predictive State Representations with Insufficient Statistics". In: *AAAI Conference on Artificial Intelligence*. 2015.

[Laf+01] J. Lafferty, A. McCallum, and F. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proc. of International Conference of Machine Learning*. 2001.

[Lan+98] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. "An Introduction to Latent Semantic Analysis". In: *Discourse Processes 25* (1998), pp. 259–284.

[Lat06] Lieven De Lathauwer. "A Link between the Canonical Decomposition in Multilinear Algebra and Simultaneous Matrix Diagonalization". In: *SIAM Journal on Matrix Analysis and Applications* 28.3 (2006), pp. 642–666.

[Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991. ISBN: 079239206X.

[Lau04] Adam Daniel Laud. *THEORY AND APPLICATION OF REWARD SHAPING IN REINFORCEMENT LEARNING*. 2004.

[Lau+08] Fabien Lauer and Gérard Bloch. "Switched and PieceWise Nonlinear Hybrid System Identification". In: *Hybrid Systems: Computation and Control: 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*. Ed. by Magnus Egerstedt and Bud Mishra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 330–343. ISBN: 978-3-540-78929-1.

[Lav+00] Steven M. Lavalle, James J. Kuffner, and Jr. "Rapidly-Exploring Random Trees: Progress and Prospects". In: *Algorithmic and Computational Robotics: New Directions*. 2000, pp. 293–308.

[Lee+17] Gilwoo Lee, Zita Marinho, Aaron M. Johnson, Geoffrey J. Gordon, Siddhartha S. Srinivasa, and Matthew T. Mason. *Unsupervised Learning for Nonlinear PieceWise Smooth Hybrid Systems*. ArXiv preprint: 1710.00440v1. ArXiv. 2017.

[Lev+15] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.

[Lia+03] Gang Liang and Bin Yu. "Maximum pseudo likelihood estimation in network tomography". In: *Signal Processing, IEEE Transactions on* 51.8 (2003), pp. 2043–2053.

[Lin88] Bruce G. Lindsay. "Composite Likelihood Methods". In: *Contemporary Mathematics* 80 (1988), pp. 221–239.

[Lit+02] Michael Littman, Richard Sutton, and Satinder Singh. "Predictive Representations of State". In: *Neural Information Processing Systems Foundation*. 2002.

[Lit+01] Michael L. Littman, Richard S. Sutton, and Satinder Singh. "Predictive Representations of State". In: *In Advances In Neural Information Processing Systems 14*. MIT Press, 2001, pp. 1555–1561.

[Liu+89] Dong Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* 45 (1989), pp. 503–528.

[Lju99] Lennart Ljung. *System identification - Theory for the User*. Prentice-Hall, 1999.

[LP81] Tomas Lozano-Perez. "Automatic planning of manipulator transfer movements". In: *IEEE Transactions on Systems, Man, and Cybernetics* 11 (1981), pp. 681–698.

[LP+79] Tomás Lozano-Pérez and Michael A. Wesley. "An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles". In: *Commun. ACM* 22.10 (Oct. 1979), pp. 560–570. ISSN: 0001-0782.

[Lui+12] Marco Lui and Timothy Baldwin. "langid.py: An Off-the-shelf Language Identification Tool". In: *Proc. of Association of Computational Linguistics System Demonstrations*. 2012, pp. 25–30.

[Luq+12] Franco M. Luque, Ariadna Quattoni, Borja Balle, and Xavier Carreras. "Spectral Learning for Non-deterministic Dependency Parsing". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. EACL '12. Avignon, France: Association for Computational Linguistics, 2012, pp. 409–419. ISBN: 978-1-937284-19-0.

[Man+99] Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language processing*. Cambridge, MA: MIT Press, 1999.

[Man+15] Jonathan H. Manton and Pierre-Olivier Amblard. "A Primer on Reproducing Kernel Hilbert Spaces". In: *Foundations and Trends® in Signal Processing* 8.1–2 (2015), pp. 1–126. ISSN: 1932-8346.

[Mar+16] Zita Marinho, André F. T. Martins, Shay B. Cohen, and Noah A. Smith. "Semi-Supervised Learning of Sequence Models with the Method of Moments". In: *Proceedings of Empirical Methods for Natural Language Processing Conference EMNLP*. 2016.

[Mat+05] T. Matsuzaki, Y. Miyao, and J. Tsujii. "Probabilistic CFG with latent annotations". In: *Proc. of Annual Meeting on Association for Computational Linguistics*. 2005, pp. 75–82.

[McC+00] Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. "Maximum Entropy Markov Models for Information Extraction and Segmentation". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 591–598. ISBN: 1-55860-707-2.

[Mer94] Bernard Merialdo. "Tagging English Text with a Probabilistic Model". In: *Computational Linguistics* 20.2 (1994), pp. 155–171.

[Mic+05] Charles A. Micchelli and Massimiliano A. Pontil. "On Learning Vector-Valued Functions". In: *Neural Comput.* (2005).

[Mit+06] Noriaki Mitsunaga, Christian Smith, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. "Robot Behavior Adaptation for Human-Robot Interaction based on Policy Gradient Reinforcement Learning". In: *Journal of the Robotics Society of Japan* 24.7 (2006), pp. 820–829.

[Miy+96] H. Miyamoto, S. Schaal, F. Gandolfo, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato. "A Kendama learning robot based on bi-directional theory". In: 9.8 (1996). clmc, pp. 1281–1302.

[Moh09] Mehryar Mohri. "Weighted Automata Algorithms". In: *Handbook of Weighted Automata*. Ed. by Manfred Droste, Werner Kuich, and Heiko Vogler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 213–254. ISBN: 978-3-642-01492-5.

[Moi+10] Ankur Moitra and Gregory Valiant. "Settling the Polynomial Learnability of Mixtures of Gaussians". In: *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. 2010, pp. 93–102.

[Mol+06] G. Molenberghs and G. Verbeke. *Models for Discrete Longitudinal Data*. Springer Series in Statistics. Springer New York, 2006. ISBN: 9780387289809.

[Mos+06] Elchanan Mossel and Sebastien Roch. "Learning nonsingular phylogenies and Hidden Markov Models". In: *Ann. Appl. Probab.* 16 (2006), pp. 583–614.

[Nar+16] Karthik Narasimhan, Adam Yala, and Regina Barzilay. "Improving Information Extraction by Acquiring External Evidence with Reinforcement Learning". In: *CoRR* abs/1603.07954 (2016).

[Ng+04] Andrew Y. Ng and H. Jin Kim. "Stable adaptive control with online learning". In: *NIPS*. 2004, pp. 977–984.

[Ngu+15] Thang Nguyen, Jordan Boyd-Graber, Jeff Lund, Kevin Seppi, and Eric Ringger. "Is your anchor going up or down? Fast and accurate supervised topic models". In: *Proc. of North American Association for Computational Linguistics*. Denver, Colorado, 2015.

[O'C+10] Brendan O'Connor, Michel Krieger, and David Ahn. "TweetMotif: Exploratory Search and Topic Summarization for Twitter". In: *ICWSM*. 2010.

[Orr+98] Genevieve B. Orr and Klaus-Robert Mueller, eds. *Neural Networks : Tricks of the Trade*. Vol. 1524. Lecture Notes in Computer Science. Springer, 1998.

[Osb+16] Dominique Osborne, Shashi Narayan, and Shay B. Cohen. "Encoding Prior Knowledge with Eigenword Embeddings". In: *Transactions of the Association of Computational Linguistics* (2016).

[Owo+13] Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. "Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters". In: *Proc. of North American Association for Computational Linguistics*. 2013.

[Pan+95] J. Pan, L. Zhang, and D. Manocha. "Collision-free and Smooth Trajectory Computation in Cluttered Environments". In: *International Journal Robotics Research*. 1995.

[Pao+07] Simone Paoletti, Aleksandar Lj. Juloski, Giancarlo Ferrari-Trecate, and René Vidal. "Identification of Hybrid Systems: A Tutorial". In: *Eur. J. Control* 13.2-3 (2007), pp. 242–260.

[Pap+98] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[Par+14] A. Parikh, S. Cohen, and E. Xing. "Spectral Unsupervised Parsing with Additive Tree Metrics". In: *ACL*. 2014.

[Par+12a] Ankur Parikh, Le Song, Mariya Ishteva, Gabi Teodoru, and Eric P. Xing. "A Spectral Algorithm for Latent Junction Trees". In: *CoRR*. 2012.

[Par+12b] Ankur P. Parikh, Lee Song, Mariya Ishteva, Gabi Teodoru, and Eric P. Xing. "A spectral algorithm for latent junction trees". In: *Proc. of Uncertainty in Artificial Intelligence*. 2012.

[Par+11] Ankur P. Parikh, Le Song, and Eric P. Xing. "A Spectral Algorithm for Latent Tree Graphical Models". In: *ICML*. Omnipress, 2011, pp. 1065–1072.

[Par+12c] C. Park, J. Pan, and D. Manocha. "ITOMP: Incremental Trajectory Optimization for Real-Time Replanning in Dynamic Environments." In: *in Proc. of the International Conference on Automated Planning and Scheduling (ICAPS)*. 2012.

[Par+70] E. Parzen, STANFORD UNIV CALIF Dept. of STATISTICS., and Department of Statistics Stanford University. *Statistical Inference on Time Series by Rkhs Methods*. Defense Technical Information Center, 1970.

[Par62] Emanuel Parzen. "Extraction and Detection Problems and Reproducing Kernel Hilbert Spaces". In: *Journal of the Society for Industrial and Applied Mathematics Series A Control* 1.1 (1962), pp. 35–62.

[Pau+17] Romain Paulus, Caiming Xiong, and Richard Socher. "A Deep Reinforced Model for Abstractive Summarization". In: *CoRR* abs/1705.04304 (2017).

[Pea94] Karl Pearson. "III. Contributions to the mathematical theory of evolution". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 185 (1894), pp. 71–110.

[Pet+08] J. Peters and S. Schaal. "Reinforcement Learning of Motor Skills with Policy Gradients". In: *Neural Networks* 21.4 (2008), pp. 682–697.

[Pet+11] Slav Petrov, Dipanjan Das, and Ryan McDonald. "A Universal Part-of-Speech Tagset". In: *ArXiv*. 2011.

[Pre97] Lutz Prechelt. "Early Stopping - but when?" In: *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*. Springer-Verlag, 1997, pp. 55–69.

[Pre+92] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

[Qua+05] Ariadna Quattoni, Michael Collins, and Trevor Darrell. "Conditional Random Fields for Object Recognition". In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, 2005, pp. 1097–1104.

[Qua+14] Ariadna Quattoni, Borja Balle, Xavier Carreras, and Amir Globerson. "Spectral Regularization for Max-Margin Sequence Tagging". In: *Proc. of International Conference of Machine Learning*. 2014, pp. 1710–1718.

[Qui+93] S. Quinlan and O. Khatib. "Elastic Bands: Connecting Path Planning and Control." In: *IEEE International Conference on Robotics and Automation*. 1993.

[Rab89] Lawrence R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[Rah+08] Ali Rahimi and Benjamin Recht. "Random Features for Large-Scale Kernel Machines". In: *NIPS*. 2008.

[Rat+09] N. Ratliff, M. Zucker, J.A. Bagnell, and S. Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *IEEE International Conference on Robotics and Automation*. 2009.

[Rat+15] N. Ratliff, M. Toussaint, and S. Schaal. "Understanding the Geometry of Workspace Obstacles in Motion Optimization". In: *IEEE International Conference on Robotics and Automation*. 2015.

[Rat+07] N.D. Ratliff and J. A. Bagnell. "Kernel Conjugate Gradient for Fast Kernel Machines." In: 2007.

[Raw+13] K. Rawlik, M. Toussaint, and S. Vijayakumar. "Path Integral Control by Reproducing Kernel Hilbert Space Embedding". In: *International Joint Conference on Artificial Intelligence*. IJCAI. 2013, pp. 1628–1634.

[Red+84] Richard A. Redner and Homer F. Walker. "Mixture Densities, Maximum Likelihood and the EM Algorithm". In: *SIAM Review* 26.2 (1984), pp. 195–239.

[Rie28] F. Riesz. *Sur la decomposition des operations fonctionnelles lineaires*. 1928.

[Rim+92] E. Rimon and D. E. Koditschek. "Exact robot navigation using artificial potential functions". In: *IEEE Transactions on Robotics and Automation* 8.5 (1992), pp. 501–518. ISSN: 1042-296X.

[Rob+10] John W. Roberts, Lionel Moret, Jun Zhang, and Russ Tedrake. "Motor Learning at Intermediate Reynolds Number: Experiments with Policy Gradient on the Flapping Flight of a Rigid Wing". In: *From Motor Learning to Interaction Learning in Robots*. 2010, pp. 293–309.

[Rod+13] Jordan Rodu, Dean P. Foster, Weichen Wu, and Lyle H. Ungar. "Using Regression for Spectral Estimation of HMMs". In: *Statistical Language and Speech Processing: First International Conference, SLSP 2013, Tarragona, Spain, July 29-31, 2013. Proceedings*. Ed. by Adrian-Horia Dediu, Carlos Martín-Vide, Ruslan Mitkov, and Bianca Truthe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 212–223. ISBN: 978-3-642-39593-2.

[Ros+04a] Matthew Rosencrantz, Geoffrey J. Gordon, and Sebastian Thrun. "Learning low dimensional predictive representations". In: *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. 2004.

[Ros+04b] Matthew Rosencrantz and Geoff Gordon. "Learning low dimensional predictive representations". In: *ICML*. 2004, pp. 695–702.

[Ros+11] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. 2011, pp. 627–635.

[Ros+10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686 (2010).

[Rub+04] Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Monte Carlo Simulation, Randomized Optimization and Machine Learning*. Springer-Verlag, 2004.

[Rus+16] Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. "Sim-to-Real Robot Learning from Pixels with Progressive Nets". In: *CoRR* abs/1610.04286 (2016).

[Sal+78] Arto Salomaa and Matti Soittola. "Automata-Theoretic Aspects of Formal Power Series". In: *Texts and Monographs in Computer Science*. 1978.

[SA12] Yoni Halpern David Mimno Ankur Moitra David Sontag Yichen Wu Michael Zhu Sanjeev Arora Rong Ge. "A practical algorithm for topic modeling with provable guarantees". In: (2012).

[Sat+02] Masa-aki Sato, Yutaka Nakamura, and Shin Ishii. "Reinforcement Learning for Biped Locomotion". In: *Artificial Neural Networks — ICANN 2002: International Conference Madrid, Spain, August 28–30, 2002 Proceedings*. Ed. by José R. Dorronsoro. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 777–782. ISBN: 978-3-540-46084-8.

[Sch+02] B. Schölkopf and A. J. Smola. *Learning with Kernels*. Cambridge, MA: The MIT Press, 2002.

[Sch+01] B. Scholkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[Sch+13] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization". In: *Robotics: Science and Systems Conference*. 2013.

[Sch+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. JMLR Workshop and Conference Proceedings, 2015.

[Sch98] Hinrich Schütze. "Automatic word sense discrimination". In: *Computational Linguistics* 24.1 (1998), pp. 97–123.

[Sch61] M. P. Schützenberger. "On the definition of a family of automata". In: *Information and Control* 4 (1961), pp. 245 –270.

[Sha+14] Uri Shalit and Gal Chechik. "Coordinate-descent for Learning Orthogonal Matrices Through Givens Rotations". In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML'14. Beijing, China: JMLR.org, 2014, pp. I–548–I–556.

[ST+04] J. Shawe-Taylor and N. Cristianni. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[Sha+17] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer". In: *arXiv preprint arXiv:1701.06538* (2017).

[Sid+07] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. "A Constraint Generation Approach to Learning Stable Linear Dynamical Systems". In: *NIPS*. 2007.

[Sid+10a] Sajid Siddiqi, Byron Boots, and Geoffrey J. Gordon. "Reduced-Rank Hidden Markov Models". In: *AISTATS*. 2010.

[Sid+10b] Sajid M. Siddiqi, Byron Boots, and Geoffrey J. Gordon. "Reduced-rank hidden markov models". In: *AISTATS* (2010).

[Sin+03] S. Singh, M. Littman, N.K.Jong, D. Pardoe, and P. Stone. "Learning Predictive State Representations". In: *ICML*. 2003.

[Sin+15] Sameer Singh, Tim Rocktäschel, and Sebastian Riedel. "Towards Combined Matrix and Tensor Factorization for Universal Schema Relation Extraction". In: *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing, VS@NAACL-HLT 2015, June 5, 2015, Denver, Colorado, USA*. 2015, pp. 135–142.

[Sin+04a] Satinder Singh, Michael R. James, and Matthew R. Rudary. "Predictive State Representations: A New Theory for Modeling Dynamical Systems". In: *UAI*. 2004.

[Sin+04b] Satinder P. Singh, Michael R. James, and Matthew R. Rudary. "Predictive State Representations: A New Theory for Modeling Dynamical Systems". In: *UAI*. 2004, pp. 512–518.

[Sin+16] Aman Sinha and John C Duchi. "Learning Kernels with Random Features". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 1298–1306.

[Smi+05] Noah A. Smith and Jason Eisner. "Contrastive Estimation: Training Log-linear Models on Unlabeled Data". In: *Proc. of Association for Computational Linguistics*. 2005, pp. 354–362.

[Smo+07a] A. Smola, A. Gretton, and B. Schölkopf. "A Hilbert Space Embedding for Distributions". In: *International conference on Algorithmic Learning Theory*. 2007, pp. 13–31.

[Smo+07b] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. "A Hilbert space embedding for distributions". In: *In Algorithmic Learning Theory: 18th International Conference*. 2007.

[Son+18] H. Song, J. J. Thiagarajan, P. Sattigeri, and A. Spanias. "Optimizing Kernel Machines Using Deep Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2018), pp. 1–13. ISSN: 2162-237X.

[Son+09] L. Song, J. Huang, A. Smola, and K. Fukumizu. "Hilbert space embeddings of conditional distributions". In: *International Conference of Machine Learning*. 2009.

[Son+10a] L. Song, B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola. "Hilbert Space Embeddings of Hidden Markov Models". In: *ICML*. 2010.

[Son+10b] Le Song, Byron Boots, Sajid M. Siddiqi, Geoffrey J. Gordon, and Alexander J. Smola. "Hilbert Space Embeddings of Hidden Markov Models". In: *ICML*. 2010, pp. 991–998.

[Sou09] A. Souloumiac. "Joint diagonalization: Is non-orthogonal always preferable to orthogonal?" In: *2009 3rd IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. 2009, pp. 305–308.

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[Str+15] Karl Stratos, Michael Collins, and Daniel Hsu. "Model-based word embeddings from decompositions of count matrices". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1. 2015, pp. 1282–1291.

[Str+13] Karl Stratos, Alexander M. Rush, Shay B. Cohen, and Michael Collins. "Spectral learning of refinement HMMs". In: *Proc. of Computational Natural Language Learning*. 2013.

[Str+16] Karl Stratos, michael Collins, and daniel J. Hsu. "Unsupervised Part-Of-Speech Tagging with Anchor Hidden Markov Models". In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 245–257.

[Str+01] Malcolm J. A. Strens and Andrew W. Moore. "Direct Policy Search using Paired Statistical Tests". In: *ICML*. Morgan Kaufmann, 2001, pp. 545–552.

[Sun+16a] Wen Sun, Arun Venkatraman, Byron Boots, and J. Andrew Bagnell. "Learning to Filter with Predictive State Inference Machines". In: *Proceedings of the 2016 International Conference on Machine Learning (ICML-2016)*. 2016.

[Sun+16b] Wen Sun, Arun Venkatraman, Byron Boots, and J. Andrew Bagnell. "Learning to Filter with Predictive State Inference Machines". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1197–1205.

[Sun14] Xu Sun. "Exact Decoding on Latent Variable Conditional Models is NP-Hard". In: *CoRR* abs/1406.4682 (2014).

[Sut+01] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12 (Proceedings of the 1999 conference)*. MIT Press, 2001, pp. 1057–1063.

[Sut+99] Richard S. Sutton, David A. Mcallester, Satinder P. Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus R. Müller. The MIT Press, 1999, pp. 1057–1063.

[Sut+98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[Szi+06] I. Szita and A. Lörincz. "Learning Tetris Using the Noisy Cross-Entropy Method". In: *Neural Computation* 18.12 (2006), pp. 2936–2941.

[TB01] John T. Betts. "Practical Methods for Optimal Control Using Nonlinear Programming". In: *SIAM J. Comput.* (Jan. 2001).

[Tam+16] Aviv Tamar, Sergey Levine, Pieter Abbeel, Yi Wu, and Garrett Thomas. "Value Iteration Networks". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, pp. 2146–2154.

[Ted+05] Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. *Learning to walk in 20 minutes*. 2005.

[Tei61] Henry Teicher. "Identifiability of Mixtures". In: *The Annals of Mathematical Statistics* 32.1 (Mar. 1961), pp. 244–248.

[Ter02] S. Terwijn. "On the Learnability of Hidden Markov Models". In: *International Colloquium on Grammatical Inference*. 2002.

[Tho+15] Michael R. Thon and Herbert Jaeger. "Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework". In: *Journal of Machine Learning Research* 16 (2015), pp. 103–147.

[Tit+85] D.M. Titterington, A.F.M. Smith, and U.E. Makov. *Statistical Analysis of Finite Mixture Distributions*. Wiley, New York, 1985.

[Tod+05] E. Todorov and W. Li. "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems". In: *in Proc. of the American Control Conference (ACC)*. 2005.

[Tou+11] Marc Toussaint, Amos Storkey, and Stefan Harmeling. "Expectation-Maximization methods for solving (PO)MDPs". In: *Bayesian Time Series Models*. Ed. by Silvia Chiappa David Barber A Taylan Cemgil. Cambridge University Press, 2011, pp. 388–413.

[Upp97] D. R. Upper. "Theory and Algorithms for Hidden Markov Models and Generalized Hidden Markov Models". Ph.D. Dissertation. Mathematics Department, University of California, Berkeley, 1997.

[VO+96] P. Van Overschee and B. De Moor. *Subspace identification for linear systems : theory, implementation, applications*. Kluwer Academic publ, 1996.

[Var+11] Cristiano Varin, Nancy Reid, and David Firth. "An overview of composite likelihood methods". In: *Statistica Sinica* 21.1 (2011), pp. 5–42. ISSN: 19968507.

[Var+05] Cristiano Varin, Gudmund Høst, and Skare. "Pairwise Likelihood Inference in Spatial Generalized Linear Mixed Models". In: *Comput. Stat. Data Anal.* 49.4 (June 2005), pp. 1173–1191. ISSN: 0167-9473.

[Vav09] Stephen A. Vavasis. "On the Complexity of Nonnegative Matrix Factorization". In: *SIAM J. on Optimization* 20.3 (Oct. 2009), pp. 1364–1377. ISSN: 1052-6234.

[Vem+04] Santosh Vempala and Grant Wang. "A spectral algorithm for learning mixture models". In: *Journal of Computer and System Sciences* 68.4 (2004). Special Issue on FOCS 2002, pp. 841 –860. ISSN: 0022-0000.

[Ven+17] Arun Venkatraman, Nicholas Rhinehart, Wen Sun, Lerrel Pinto, Byron Boots, Kris Kitani, and James Andrew Bagnell. "Predictive State Decoders: Encoding the Future into Recurrent Networks". In: *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2017.

[Vla+09] Nikos Vlassis and Marc Toussaint. "Model-free reinforcement learning as mixture learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 2009, pp. 1081–1088.

[Wah99] G. Wahba. *Advances in Kernel Methods*. MIT Press, 1999. Chap. Support Vector Machines, Reproducing Kernel Hilbert Spaces, and Randomized GACV.

[Wai+08] Martin J. Wainwright and Michael I. Jordan. "Graphical models, exponential families, and variational inference". In: *Foundations and Trends in Machine Learning* 1.2 (2008), pp. 1–305.

[Wan+16] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. "Sample efficient actor-critic with experience replay". In: *arXiv preprint arXiv:1611.01224* (2016).

[Wel03] Lloyd R. Welch. "Hidden Markov Models and the Baum-Welch Algorithm". In: *IEEE Information Theory Society Newsletter* 53.4 (2003).

[Wer90] Paul J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* (1990).

[Wie+10] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. "Recurrent Policy Gradients". In: *Logic Journal of the IGPL* 18.5 (Oct. 2010), pp. 620–634.

[Wil92] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning*. 1992, pp. 229–256.

[Wil+16] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. "Deep Kernel Learning". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, 2016, pp. 370–378.

[Xan+16] Marios Xanthidis, Ioannis M. Rekleitis, and Jason M. O'Kane. "RRT+ : Fast Planning for High-Dimensional Configuration Spaces". In: *CoRR* abs/1612.07333 (2016).

[Yua+10] M. Yuan and T. Cai. "A reproducing kernel Hilbert space approach to functional linear regression". In: *Annals of Statististics* (2010).

[Zha+95] J. Zhang and A. Knoll. "An Enhanced Optimization Approach for Generating Smooth Robot Trajectories in the Presence of Obstacles". In: *in Proc. of the European Chinese Automation Conference*. 1995.

[Zho08] D. Zhou. "Derivative Reproducing properties for kernel methods in learning theory". In: *Journal of Computational and Applied Mathematics* (2008).

[Zho+14] Tianyi Zhou, Jeff A. Bilmes, and Carlos Guestrin. "Divide-and-Conquer Learning by Anchoring a Conical Hull". In: *Advances in Neural Information Processing Systems*. 2014.

[Zhu+99] Xiaojin Zhu, Stanley F. Chen, and Ronald Rosenfeld. "Linguistic features for whole sentence maximum entropy language models". In: *Sixth European Conference on Speech Communication and Technology, Eurospeech*. 1999.

[Zhu+11] Jinfeng Zhuang, Ivor W. Tsang, and Steven C.H. Hoi. "Two-Layer Multiple Kernel Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 909–917.

[Zim+12] Hans-Georg Zimmermann, Christoph Tietz, and Ralph Grothmann. "Forecasting with Recurrent Neural Networks: 12 Tricks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 687–707. ISBN: 978-3-642-35289-8.

[Zuc+13] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. A. Bagnell, and S. Srinivasa. "CHOMP: Covariant Hamiltonian Optimization for Motion Planning". In: *International Journal Robotics Research*. 2013.