Learning Robotic Perception Through Prior Knowledge

vorgelegt von
M.Sc.
Rico Jonschkowski
geb. in Havelberg

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
— Dr. rer. nat. —

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Manfred Opper
Gutachter:	Prof. Dr. Oliver Brock
Gutachter:	Prof. Dr. George Konidaris
Gutachter:	Prof. Dr. Marc Toussaint

Tag der wissenschaftlichen Aussprache: 02. Mai 2018

Berlin 2018

Thesis advisor: Professor Oliver Brock Rico Jonschkowski

Learning Robotic Perception Through Prior Knowledge

Abstract

Intelligent robots must be able to learn; they must be able to adapt their behavior based on experience. But generalization from past experience is only possible based on assumptions or *prior knowledge* (priors for short) about how the world works.

I study the role of these *priors* for learning perception. Although priors play a central role in machine learning, they are often hidden in the details of learning algorithms. By making these priors explicit, we can see that currently used priors describe the world from the perspective of a *passive disinterested observer*. Such *generic AI priors* are useful because they apply to perception scenarios where there is no robot, such as image classification. These priors are still useful for learning robotic perception, but they miss an important aspect of the problem: the robot.

Robots are neither disinterested nor passive. They are trying to *achieve tasks* by *interacting with the world* around them, which adds structure to the problem and affords new kinds of priors, which I call *robotic priors*. The questions are: *What* are the right robotic priors and *how* can they be used to enable learning?

I investigate these questions in three different perception problems based on raw visual input: 1. learning object segmentation for picking up objects in the Amazon picking challenge, 2. learning state estimation for localization and tracking, and 3. unsupervised learning of state representations that facilitate reinforcement learning.

To solve these problems, I propose three sources of prior knowledge—1. *the robot's task*, 2. *robotic algorithms*, and 3. *physics*—and develop ways to encode these priors for the corresponding learning problems. Some of these priors are best encoded as hard constraints that restrict the space of hypotheses considered during learning. Other priors are more suitable to be encoded as preferences for certain hypotheses in the form of learning objectives.

My experiments across these problems consistently show that robotic-specific prior knowledge leads to more efficient learning and improved generalization. Based on these results, I propose to take a prior-centric perspective on machine learning, from which follows that we need robotics-specific machine learning methods that incorporate appropriate priors.

Doktorvater: Professor Oliver Brock Rico Jonschkowski

Lernen von robotischer Wahrnehmung durch Vorwissen

Zusammenfassung

Intelligente Roboter müssen in der Lage sein zu lernen, um ihr Verhalten auf Basis von Erfahrung anzupassen. Um aus spezifischen Erfahrungen allgemeine Schlüsse zu ziehen, bedarf es jedoch Annahmen oder *Vorwissen* über die Welt.

Ich untersuche die Bedeutung dieses Vorwissens für das Lernen von Wahrnehmung. Obwohl Vorwissen eine zentrale Rolle im maschinellen Lernen spielt, ist es oft in den Details der Lernalgorithmen verborgen. Wenn wir dieses Vorwissen explizit machen, wird deutlich, dass aktuell benutztes Vorwissen die Welt aus der Sicht eines *passiven ziellos Beobachters* beschreibt. Solche *allgemeinen KI-Annahmen* sind hilfreich, weil sie auf Wahrnehmungsprobleme wie Bildklassifizierung anwendbar sind, bei denen es keinen Roboter gibt. Solche Annahmen sind auch für das Lernen robotischer Wahrnehmung hilfreich, aber sie übersehen einen wichtigen Aspekt des Problems: den Roboter.

Roboter sind weder ziellos noch passiv. Sie versuchen bestimmte *Aufgaben zu lösen*, indem sie *mit der Welt interagieren*. Dadurch ergibt sich zusätzliche Problemstruktur, die in anderen Arten von Vorwissen genutzten werden kann. Es stellen sich daher die Fragen *was* die richtigen Robotik-Annahmen sind und *wie* diese genutzt werden können, um Lernen zu ermöglichen.

Ich beschäftige mich mit diesen Fragen in drei unterschiedlichen Wahrnehmungsproblemen auf Basis von visuellen Eingaben: 1. Lernen von Objektsegmentierung die es ermöglicht bestimmte Objekte in der Amazon Picking Challenge zu greifen, 2. Lernen von Zustandsschätzung für Lokalisierung und Nachführung und 3. unüberwachtes Lernen von Zustandsrepräsentationen, die bestärkendes Lernen ermöglichen.

Um diese Probleme zu lösen, schlage ich drei Quellen für Vorwissen vor – 1. *die Aufgabe des Roboters*, 2. *Algorithmen aus der Robotik*, 3. *physikalische Gesetze* – und entwickle Möglichkeiten Annahmen aus diesen Quellen in den entsprechenden Lernproblemen zu nutzen. Manche dieser Annahmen lassen sich am besten als *harte Bedingungen* kodieren, die den Raum der möglichen Hypothesen einschränken die beim Lernen in Betracht gezogen werden. Andere Annahmen sind besser dazu geeignet mit ihnen konsistente Hypothesen zu bevorzugen, indem diese Annahmen als Lernzielen implementiert werden.

Meine Experimente in den drei untersuchten Problemen stimmen darin überein, dass robotikspezifische Annahmen Lernen effizienter machen und Generalisierung verbessern. Aufgrund dieser Ergebnisse argumentiere ich für eine Sicht auf maschinelles Lernen, die Vorwissen ins Zentrum der Untersuchung stellt. Aus dieser Sicht folgt, dass wir robotikspezifische Lernmethoden mit entsprechenden Annahmen benötigen.

This text is dedicated to those who challenge beliefs,
search for the truth, and inspire others to do the same.

Acknowledgments

These last five and a half years have been a fantastic experience. They were exciting, challenging, fun, and rewarding. I am grateful to everyone who has helped me walk this path and happy about everyone I have met on that journey.

First of all, I want to wholeheartedly thank my family who have been nothing but supportive and kind to me. In particular, I want to thank my grandma who has taught me reading, which was *not* an easy task; I want to thank my dad and my brother for always being there for me without hesitation, and I want to thank my mom who has taught me gratitude and boldness and who has nudged me—more than anyone else—in directions that in hindsight turned out to be right.

I also want to thank my friends for making this time so enjoyable: Anna-Lena, thank you for making me happy through your kindness and your wits, thank you for going out of your way to support me in these past weeks and for watching Star Wars with me—I love you. Daniel, you have been the best flatmate anyone could ever hope for. Hannes, I am always enthusiastic when we meet, unless we talk about politics of course. Dave, without our early RoboCup project in school, I might never have studied computer science. Thank you for that!

I am also extremely grateful to every member of the RBO lab for the creating an inspiring, supportive, honest, and friendly atmosphere. Thank you, Alexander, Antonin, Arne, Armin, Can, Clemens, Divyam, Előd, Eveline, Florian, Gabriel L., Gabriel Z., Henrietta, Ingmar, Ines, Janika, Jessica, Johannes, José, Kolja, Lizzy, Mahmoud, Malte, Manuel, Marianne, Melinda, Michael, Oliver, Philipp, Raphael, Robert, Roberto, Roman, Sebastian H., Sebastian K., Stanimir, Steffen, Tim, Thomas, Vincent, and Wolf. I could not have dreamed up a better research lab for doing my PhD. Thank you Clemens for 30 cakes, thank you Arne for being relaxed—always, thank you Manuel for the annoyotron, thank you Johannes and Robert for not being mad about me placing the annoyotron in your office, thank you Janika for having held the lab together, thank you Can and Jessica for Tango, thank you Sebastian for great discussions about machine learning and fierce foosball games, thank you Vincent for always being friendly and helpful, thank you Ines for your pep talks during our bouldering sessions, thank you Roberto for last-minute emergency tips for printing this thesis ...

I want to thank the folks from DeepMind—Alistair, Andrei, Andrew, Cosmin, Dan, David, Diego, Fran, Fumin, Gabe, Hayley, Igor, Jon, Kayleigh, Martin, Mateusz, Nicolas, Roland, Tobi, Thomas, Thomas, Tom, Yuval, and many others—for the amazing internship that I was able to spend there. This time showed showed me that inspiring and fun academic research is possible outside of a university. I also want to thank Mike, Erich, and Karol for important advice about which path to take next.

Lastly, I want to thank my committee members, not just for being on my committee but for having been my mentors along the way. Thank you Marc for introducing me to machine learning research. In our weekly discussions during my Master's thesis, you infected me with your enthusiasm about learning which planted the seed for my PhD topic. Thank you George for our meetings at conferences, via skype, and at Brown. You

gave me plenty of helpful advice throughout this journey. And lastly, thank you Oliver, I have learned so much from you about teaching, research, and about being rigorous. You taught me to question everything and to not be afraid of the state of confusion which this approach regularly leads to. You have taught me to restructure my thoughts over and over and over again until it finally all makes sense.

Again, a big thanks to everyone of you and all those that I did not mention here personally. Since this thesis could not have come about without your help, it is dedicated to *you*.

Prepublication and Statement of Contribution

Parts of this thesis have been previously published in the following articles:

[A] Rico Jonschkowski, Clemens Eppner*, Sebastian Höfer*, Roberto Martín-Martín*, and Oliver Brock. Probabilistic Multi-Class Segmentation for the Amazon Picking Challenge. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[B] Clemens Eppner*, Sebastian Höfer*, Rico Jonschkowski*, Roberto Martín-Martín*, Arne Sieverling*, Vincent Wall*, and Oliver Brock. Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems. In *Proceedings of Robotics: Science and Systems (RSS)*, 2016.

[C] Rico Jonschkowski and Oliver Brock. End-To-End Learnable Histogram Filters. In *Workshop on Deep Learning for Action and Interaction at NIPS*, 2016.

[D] Rico Jonschkowski and Oliver Brock. Learning State Representations with Robotic Priors. *Autonomous Robots*, 39(3):407–428, 2015.

[E] Antonin Raffin*, Sebastian Höfer*, Rico Jonschkowski, Oliver Brock, and Freek Stulp. Unsupervised Learning of State Representations for Multiple Tasks. In *NIPS 2016 Workshop on Deep Learning for Action and Interaction*, Barcelona, Spain, 2016.

[F] Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin Riedmiller. PVEs: Position-Velocity Encoders for Unsupervised Learning of Structured State Representations. In *New Frontiers for Deep Learning in Robotics Workshop at RSS*, 2017.

[G] Rico Jonschkowski*, Sebastian Höfer*, and Oliver Brock. Patterns for Learning with Side Information. *arXiv:1511.06429*, 2015.

* contributed equally.

Own Contributions

[A] (©2016 IEEE) I am the sole first author of this paper. I conceived and evaluated the algorithm presented in the paper and made the main contributions to algorithm design, implementation, and paper writing. The next three authors (CE, SH, RMM) contributed to algorithm design, implementation, data collection, evaluation, and paper writing. The last author (OB) gave scientific advice and contributed to paper writing.

[B] I share the first authorship with the indicated authors, all of which contributed equally to system design, implementation, and evaluation as well as paper writing. The last author (OB) conceived the project, gave scientific advice, and contributed to paper writing.

[C] I am the sole first author of this paper. I conceived, designed, implemented, and evaluated the algorithm presented in the paper and made the main contribution to paper writing. The last author (OB) gave scientific advice, and contributed to paper writing.

[D] I am the sole first author of this paper. I conceived, designed, implemented, and evaluated the algorithm presented in the paper and made the main contribution to paper writing. The last author (OB) gave scientific advice and contributed to paper writing.

[E] I am the third author of this paper. I contributed code based on [D] and was involved in proposing the two new priors and formulating them as learning objectives. I also gave scientific advice and contributed to paper writing. The first two authors (SH and AR) share the first authorship. Both were involved in designing, implementing, and evaluating the algorithm and made the main contributions to paper writing. The fourth author (OB) gave scientific advice and contributed to paper writing. The last author (FS) together with SH conceived the project idea. FS also gave scientific advice and contributed to paper writing.

[F] I am the sole first author of this paper. I conceived, designed, implemented, and evaluated the algorithm presented in the paper and made the main contribution to paper writing. The second author (RH) contributed code to evaluate the learned representations for reinforcement learning and helped with running the experiments. The third author (JS) gave scientific advice and contributed to paper writing. The last author (MR) proposed experiments, gave scientific advice, and contributed to paper writing.

[G] I share the first authorship with SH. We contributed equally to the project idea, literature survey, implementation, experiments, and paper writing. The last author (OB) gave scientific advice and contributed to paper writing.

Appearance in this Thesis

Chapters 1 and **2** are original to this thesis.

Chapter 3 contains parts that were previously published in [A] and [B], in particular the description of the segmentation method, the experiments, and the results.

Chapter 4 contains parts that were published in [C], which introduced differentiable histogram filters.* The work on differentiable particle filters is original to this thesis.

Chapter 5 contains parts that were previously published in [D], [E], [F], and [G]. [D] proposed the original algorithm, [E] and [F] introduced the extension discussed in Section 5.6 and 5.7, and [G] generalized insights from this work as discussed in Section 5.8.

Chapters 3-5 extend the publications above by providing additional background information, an extended treatment of task-specific, algorithmic, and robotic priors, and by relating these separate lines of work to each other.

Chapter 6 is original to this thesis.

*They were originally referred to as end-to-end learnable histogram filters.

Contents

1 Introduction

Versatile robots need to react to their environment. Instead of executing their movements blindly, they need to sense their surroundings and act according to their sensory input. Imagine a robot fetching an object; if the object slips out of its hand, the robot should not continue as if the object was still there, but recognize this failure and react to it. Such reactive short-term adaptations are important for intelligent robots but they are not sufficient.

Robots not only need to react, they also need to *adapt how they react* to adjust their behavior *long-term*: If a robot fails to grasp an object many times in a row, it should try to grasp it differently next time; and it should remember what worked and what did not work in order to improve with experience. This long-term adaptation is what I refer to as *learning*. And learning is what this thesis is about.

More specifically, my thesis is about how robots can *learn perception*, which for this thesis I define as *learning to extract task-relevant information from sensory input*. This definition is consistent with Marr (1982) who describes vision as "a process that produces from images of the external world a description that is useful to the viewer and not cluttered with irrelevant information".

Our working hypothesis is that robots need to learn perception in order to learn new tasks. The main argument is the following: Different tasks require information about different aspects of the environment. As a result, versatile robots must have task-general sensors that provide the union of the information required by these tasks. But the high dimensionality of the sensory input and the "clutter" in the form of task-irrelevant information makes learning any specific task very difficult. Therefore, robots need to learn to extract from that high-dimensional input only the relevant information.

The key insight that I am going to present in this thesis is that the *robotic domain*

affords large amounts of prior knowledge that we can leverage to learn perception more efficiently. I will refer to these bits of prior knowledge as *priors* (Bengio et al., 2013). Priors are things that robots do not need to learn either because they are universally true, like the laws of physics, or because they represent heuristics that are useful for the problem domain. By incorporating such priors into the learning process, we can generate learning methods that are specifically tailored to robotics.

1.1 Robotics-Specific Machine Learning

Before going into more detail, I will present additional background information. In the context of artificial intelligence, the subject of learning is called *machine learning* and the main paradigm is the following: Instead of solving a problem directly, e.g. by writing an algorithm, we solve it indirectly by gathering data from the problem and applying a learning algorithm on that data, which then produces a solution for the problem.

The promises of this indirect approach to problem solving are: 1) it can outperform directly engineered solutions, such as in the computer vision domain (Krizhevsky et al., 2012), and 2) it can *automate the process of problem solving*, which is what we need for intelligent robots. Without humans in the loop, robots could apply learning algorithms as they go and learn from experience. But this second promise requires the learning methods to be general enough to find solutions to all problems that a robot might encounter. Therefore, the goal of most machine learning research is to *find algorithms that are maximally general* so that they produce a wide range of useful solutions for different problems based on data from the problem at hand.

However, it is important to understand that *machine learning can never be completely general.* Learning from a blank slate is impossible because learning algorithms can only generalize from observed data points to new cases using additional assumptions, priors, or inductive biases (I use these terms interchangeably). This insight directly follows from the no-free-lunch theorem (Wolpert, 1996) which proves that no machine learning method works better than random guessing when averaged over all possible problems. The only way to improve over random guessing is to *restrict the problem space and incorporate prior knowledge* about this space into the learning method. This view is also reflected in the need for prior probabilities in Bayesian statistics, the need for biases in machine learning (Mitchell, 1980; Geman et al., 1992), and the effectiveness of heuristics (Tversky and Kahneman, 1974; Gigerenzer and Brighton, 2009).

Of course, there are machine learning methods that apply to a wide range of real world problems by incorporating *fairly general priors*, e.g. smoothness, simplicity, or hierarchical structure (Bengio et al., 2013). These priors are so general because they describe the world from the perspective of a *disinterested passive observer*, which makes them applicable to

many learning problems in machine learning that do not include a robot, such as image classification. The same priors are still useful for robotic perception but they do not exploit the additional problem structure of robotics. Since robots do have to solve certain tasks, task-irrelevant information can be ignored in perception. And since robots can perform actions and examine the result, they can reveal additional information about the environment (Bohg et al., 2017). This problem structure affords new priors about the world from the perspective of a *task-driven* and *active* robot.

Our hypothesis is that, in order to achieve efficient learning for robots, we need to encode such *robotic priors* to generate robotic-specific machine learning methods. We must give up generality and commit to the restricted problem space of robotics. We must assume invariant properties of robotic problems that do not need to be learned. And we must find ways to incorporate those as prior knowledge into learning algorithms.

For robotics-specific machine learning, we need to answer two fundamental questions:

1. *What* are the right priors for robotics?

2. *How* can we best incorporate these priors into machine learning?

Although my thesis certainly cannot provide *final answers* to these profound questions, it proposes answers for three specific perception problems in robotics and presents some general insights based on these.

1.2 This Thesis: Robotics-Specific Machine Learning for Perception

My thesis investigates these two fundamental questions—the *what* and *how* of priors in robotics-specific learning—in the context of robotic perception. It consists of three main chapters that each present a perception problem, a set of priors (what), and a way of using these priors for efficient learning (how). In each of these three chapters, I propose a sources for prior knowledge in robotics: *the robot's task*, *robotic algorithms*, and *physics*.

These chapters are ordered such that their problems and solutions increase in generality. The first perception problem is the most narrow one, and it is solved using priors that are specific to the task at hand. The perception problems in the later chapters are more general, as are the priors that I encoded in their solutions. Since these priors are based on robotic algorithms and the laws of physics, the resulting methods are applicable to a wide range of robotic tasks. Taken together, these chapters span a spectrum of perception problems and appropriate priors of different generality and demonstrate multiple ways of incorporating priors into machine learning. The three main chapters are the following:

- **Chapter 3 Learning Object Segmentation Through Task-Specific Priors** addresses the problem of object segmentation in the Amazon picking challenge (APC), where the goal is to find the segment of an image that shows the target object, such that the robot can locate that object and pick it up. This task was challenging because the number of manually annotated training images was limited to a handful per object while the resulting method had to be robust to conditions that deviated from the lab setting, e.g. lighting that could not be controlled in the challenge venue.

  I solved this problem using prior knowledge about the task (what), for example the fact that the objects were placed in a known shelf and that every shelf bin only contained a small number of known objects. I encoded these task-specific priors in the segmentation method by defining a set of useful features such as position in the shelf and by ignoring other objects during segmentation (how). The resulting method only has a small number of learnable parameters and a highly restricted hypothesis space.

- **Chapter 4 Learning State Estimation Through Algorithmic Priors** focuses on the problem of learning to estimate a latent state from a sequence of actions and observations, for example learning to localize in a maze based on camera images and odometry. The difficulty in this task comes from the fact that single observations are insufficient to estimate the state, which is why the method needs to learn how to integrate information from observations and actions over time.

  For this problem, I used the Bayes' filter algorithm as prior knowledge (what) because it makes useful assumptions about the structure of the problem by recursively estimating a probability distribution over states using prediction and measurement update. By implementing this algorithm in a differentiable way, the parameters of the motion model and measurement model can be learned end-to-end by back-propagation (how). Thus, the algorithmic prior is encoded as a restriction of the hypothesis space to only consider valid Bayes filters during learning.

- **Chapter 5 Learning State Representations Through Physics-based Priors** addresses the problem of unsupervised learning of state representations for a given reinforcement learning task. Here, the robot needs to learn to extract from its observations the relevant information for learning a policy that maximizes long term reward. The challenge of this problem is that the robot does not receive any supervision about what information is relevant for the task.

  To solve this problem, I formulated prior knowledge about interacting with the physical world (what). The intuition behind these *physics-based* priors is the following: Since internal state representations must reflect properties of the world, the

same physical laws that apply to the real world must also apply to these internal representations. Therefore, knowledge about physical laws can help to find useful state representations. I implemented this idea by formulating consistency with physics-based priors as learning objectives (how).

Before coming to these three main chapters, the next chapter will first discusses the most relevant concepts for this thesis. This does not only provide background information about machine learning basics, but also presents the a certain perspective on the topic that motivates my work and underlies the following chapters.

2

Background: Machine Learning and Prior Knowledge

In this chapter, I will introduce machine learning concepts from a perspective that focuses on the central role of prior knowledge. This perspective forms the foundation of my thesis. It builds on inductive biases (Mitchell, 1980), the bias-variance dilemma (Geman et al., 1992), the no-free-lunch theorem (Wolpert, 1996), and AI priors (Bengio et al., 2013).

The main idea is simple:

> *Machine learning combines prior knowledge and experience in order to generalize to new situations.*

I will refer to different bits of prior knowledge as *priors* and I will call gathered experience *data*. Machine learning fuses priors and data to generate a function that can predict properties of new unseen *test data*. Making accurate predictions about these test data is the goal and measure of success in machine learning.

2.1 Priors in Bayesian Statistics and Machine Learning

But what do I really mean by *prior*? In Bayesian statistics, the word "prior" refers to the prior probability distribution $p(H)$ that is multiplied by the data likelihood $p(D \mid H)$ and then normalized to compute the posterior $p(H \mid D)$.

$$p(H \mid D) = \eta \, p(D \mid H) \, p(H),$$

where η denotes the normalization and the random variables D and H represent data and the inferred hypothesis. In this way, priors represent knowledge, before taking the data into account, as probability distributions.

In machine learning, the word *prior* is used more broadly in an analogous fashion (Bengio et al., 2013). In the context of my thesis, a prior represents knowledge about a class of learning problems that is available before taking into account data from a specific problem instance. I do not restrict the representation of these priors to be probability distributions.

Priors are the basis on which we interpret data. In this sense, *machine learning methods are the implementation of prior knowledge* about a learning problem. From this fact follows that machine learning methods *cannot be general* because the prior knowledge they encode restricts the space of problems for which they are useful. Without this restriction, machine learning is impossible because data can be interpreted in prohibitively many (usually infinite) ways, as we will see in the next section.

2.2 Machine Learning Needs Priors

2.2.1 Example 1: Find the Rule

Let me illustrate why we need prior knowledge to generalize from data using a simple math puzzle from Pierce (2017). Suppose you are the machine learning method; you receive a set of data and you need to infer a predictive function. How would you continue this sequence of numbers: $x_{1:4} = 1, 2, 4, 7$? What is your best guess for x_5? Please take a few moments and try to solve this problem before reading on. It will make this point more perspicuous.

How can you find the rule? One approach might be to look at the differences from each element to the next. If you do that, you will find that $x_2 = x_1 + 1$, $x_3 = x_2 + 2$ and $x_4 = x_3 + 3$. From these examples you can induce the rule that $x_i = x_{i-1} + i - 1$ with the base case $x_1 = 1$. Using this rule, you would predict x_5 to be $x_4 + 4 = 7 + 4 = 11$. This is a reasonable solution but not the only one. Another rule that is consistent with the observations is $x_i = x_{i-1} + x_{i-2} + 1$ with $x_1 = 1$ and $x_2 = 2$, according to which x_5 should be 12. Equally plausible is the rule $x_i = x_{i-1} + x_{i-2} + x_{i-3}$ with base cases for $i \leq 3$. This rule predicts x_5 to be 13. Now, we have three different functions that explain the data perfectly but that all make different predictions about new data. And how can we be certain that the underlying process that generates the sequence corresponds to a deterministic recursive formula? This sequence could be something entirely different. For example, the numbers 1, 2, 4, 7 also happen to be a sequence of digits in π starting with the $16,992$nd decimal place, $\pi = 3.1415\ldots\mathbf{1247}679\ldots$. If that is the rule which generated

the sequence, then x_5 should be 6 because it is the next digit in line. And we could find endless other explanations for this sequence that lead to yet different predictions.

2.2.2 There is No Free Lunch

Without knowing anything about this sequence, we have no way of deciding which of these rules is the right one and we cannot confidently make any predictions about how the sequence continues. I have illustrated this principle with a simple example, but the same arguments apply to every machine learning problem from image classification to learning autonomous helicopter flight.

Wolpert (1996) formulated the need for prior knowledge mathematically as the *no free lunch theorem* and proved it for supervised learning. The no free lunch theorem roughly states that no machine learning method outperforms random guessing when averaged over all possible learning problems. The only way to improve over random guessing is to restrict the problem space and incorporate prior knowledge about this problem space into the learning method.

2.2.3 The Universal Prior—Is Occam's Razor Sufficient?

The no free lunch theorem proves that we need *some* prior knowledge. But maybe there is a *universal prior* that applies to all problems we are interested in. For example, would it be sufficient to always choose the simplest rule that is consistent with the data? The simplicity prior, also called Occam's razor, states that simpler rules are more likely than complex ones and should be preferred. Is this prior sufficient?

Unfortunately, the simplicity prior alone does not solve our problem because it does not specify *how to measure complexity*. In the example above: Which of the rules is the most complex one? Is the digit sequence of π a more or less complex rule than a recursive formula? There is no objective answer to this question without specifying a common measure of complexity.

One attempt to resolve this dilemma is Solomonoff induction or Kolmorogov complexity (Solomonoff, 1964; Kolmogorov, 1965). These mathematical formulations of Occam's razor force rules into a common representation—in this case the shortest computer program that represents the rule—and then compare the length of these programs. Shorter programs are simpler and therefore more likely. However, this complexity measure depends on the programming language and is therefore not objective either (Leike and Hutter, 2015). A rule that is simple to express in one language might be arbitrarily complex in another one. Finding the right programming language such that the shortest programs are the most likely explanations for the data is simply another way of searching for useful priors. It might be an interesting route to express prior knowledge in the form of a programming

language that serves as a generative model for the world around us, but the need for prior knowledge remains.

2.2.4 Relation to Cognitive Science and Psychology

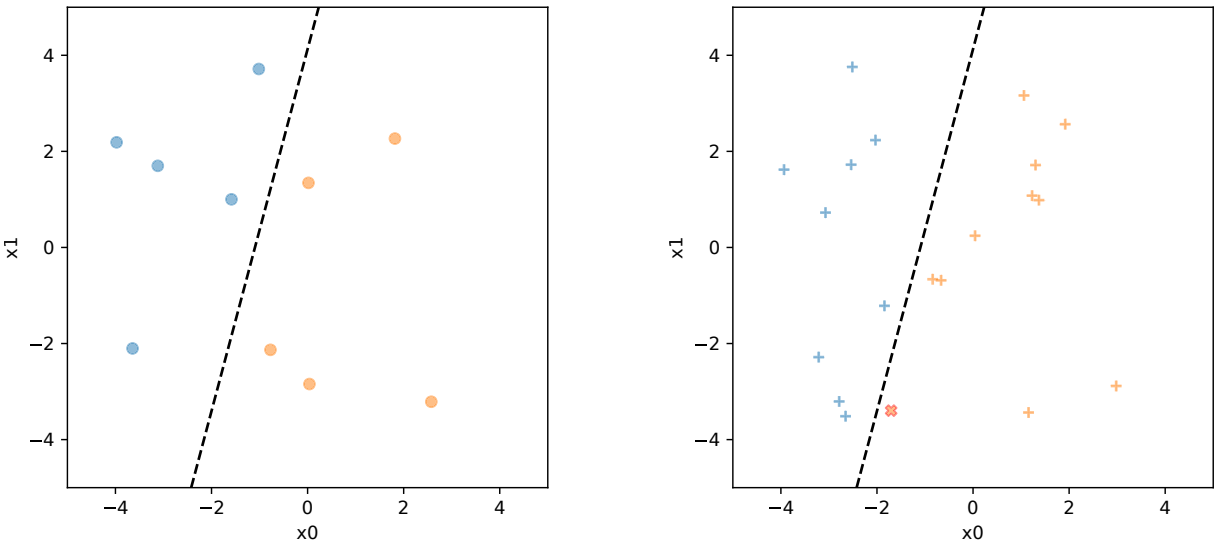
There is an interesting link between this topic and cognitive science. The argument that learning requires prior knowledge does not only apply to *machine* learning, it also applies to learning in biological organisms, including humans. The fact that humans are remarkable at generalizing from very small amounts of data suggests that the human mind exploits large amounts of prior knowledge (Gigerenzer and Brighton, 2009). Research in this area has identified a number of priors, which are usually called cognitive biases and heuristics (Tversky and Kahneman, 1974; Gigerenzer and Brighton, 2009; Baron, 2000). Haselton et al. (2015) suggests that these priors might have been incorporated into our learning mechanism by natural selection because they helped to solve evolutionary relevant problems.

Dubey et al. (2018) investigated human priors in video games by testing humans in systematically altered game versions that are inconsistent with these priors, e.g. because object semantics are masked or because the direction of gravity was changed. They found that, while state of the art reinforcement learning methods perform equally poorly in each of these variants, human performance is substantially better in the original game that does not restrict the use of their priors.

This topic is also related to the nature vs. nurture debate in psychology, a debate about the degree to which human traits are determined by genetics (nature) or shaped by the environment (nurture). We can rephrase this question to "How much is human behavior innate vs. learned from individual experience?". If we view human behavior as a result of learning throughout life, the innate properties correspond to prior knowledge in the "human learning algorithm".

Recently, there has been a very comprehensive meta-study that should finally settle the nature vs. nurture debate. Polderman et al. (2015) presented a meta-analysis of the heritability of human traits that summarizes 50 years of twin studies (17,804 traits from 2,748 publications that include 14,558,903 twin pairs) and comes to the conclusion that "across all traits, the reported relative contribution of genes and environment is equal" (49% genes and 51% environment).

This result does not only show that both nature and nurture (or priors and data) contribute to human behavior, it also demonstrates that their relative contribution is of very limited help to understand how human behavior emerges from nature and nurture. The most relevant question to gain this understanding seems to be how nature and nurture interact. In machine learning, we must analogously study *how* we can encode prior knowledge into learning algorithms to improve generalization.



(a) Training **(b)** Test

Figure 2.1: Logistic regression. Colors denote the true classes for (a) and the predicted classes for (b). The dashed lines denote the learned decision boundary. Prediction errors in (b) are shown as red crosses.

2.3 Encoding Priors into Machine Learning

2.3.1 Example 2: Logistic Regression

So far, I have discussed priors in machine learning on a very abstract level. Now, I would like to ground our discussion in a simple example using logistic regression. Please bear with me while I walk through the technicalities. The example should illuminate *how* prior knowledge can be incorporated in different machine learning components.

Let us look at a supervised learning problem with data from two classes A and B. From each class we have $n = 5$ samples, where each sample is described by a binary class label y and an input vector $\mathbf{x}$, which contains two features $\mathbf{x} = (x_1, x_2)^T$. Our training data consists of a set of these samples $D = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{n}$.

Using these data and prior knowledge about the problem, we want to infer a function $f : \mathbf{x} \mapsto y$ that can correctly predict the label for new samples. If our prior is that A and B are separated by a line in input space, we can find such a line using logistic regression (see Fig. 2.1).

In logistic regression, f has the following form $\hat{y} = f(\mathbf{x}) = \sigma(\mathbf{x}^T\theta)$, where $\hat{y}$ is the predicted class, θ is the weight vector that is being learned, and σ is the sigmoid function $\sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}}$. If we apply f to the whole data set, we get $\hat{\mathbf{y}} = \sigma(\mathbf{X}\theta)$, where $\mathbf{X}$ is the matrix of input features. To construct $\mathbf{X}$, we append the feature vector for each sample

with a constant 1 and stack these vectors such that row i represents the input $\mathbf{x}^{(i)}$:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} \end{pmatrix}.$$

The weight vector consists of three parameters, $\theta = (\theta_0, \theta_1, \theta_2)^T$, where θ_0 is the bias term that multiplies the constant 1 and θ_1 and θ_2 are the weights for the corresponding features x_1 and x_2. For each sample i, f computes $z^{(i)} = \mathbf{x}^{(i)^T}\theta$ and then applies the logistic function to produce a value $p^{(i)}$ between zero and one: $p^{(i)} = \sigma(z^{(i)})$. We use this value as the probability of the sample belonging to class A, such that $1 - p^{(i)}$ denotes the probability of class B.

The sigmoid function maps 0 to a probability of 0.5, negative values to lower probabilities and positive values to higher probabilities (see Fig. 2.2). Since the input to the sigmoid function is linear in $\mathbf{x}$, f defines a line, $0 = z = \theta_0 + x_1\theta_1 + x_2\theta_2$, on which the classes A and B are equally likely. This line splits the input space into a part where $z > 0$, $p > 0.5$, and therefore class A is more likely and a part where $z < 0$, $p < 0.5$, and therefore class B is more likely.

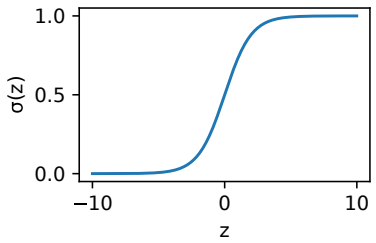
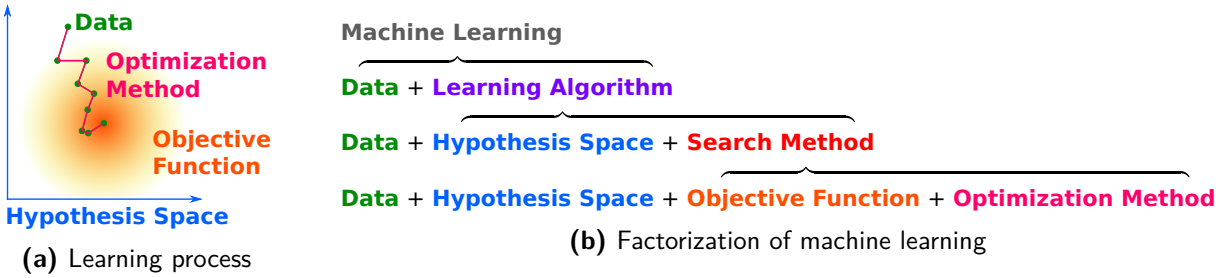


Figure 2.2: Sigmoid function

With logistic regression, we can estimate the parameters of this line from data (see Fig. 2.1a). After training, we can use the learned f to predict the class label for new samples that were not seen during training. The goal is to minimize the expected error on the test data; the function learned in this example makes one prediction error (see Fig. 2.1b).

After having introduced this learning setting, the parameters, and the result, I will next describe how logistic regression estimates the line parameters from data. Using this example, I will introduce the four main components of machine learning and how priors can be encoded into each of them.



(a) Learning process

(b) Factorization of machine learning

Figure 2.3: The components of machine learning. (a) shows how these components interact to fit a hypothesis to data: Starting at some point in the hypothesis space, we estimate the loss with respect to our objective function based on the training data (green dot at the top). Using an optimization method such as gradient descent, we update our hypothesis, evaluate it again and repeat these steps until we find an optimum of the objective function. (b) depicts this factorization of machine learning which applies to most modern machine learning methods.

2.3.2 Four Machine Learning Components

To understand the mechanics of machine learning, we must look at the different machine learning components and their interaction (see Fig. 2.3), as proposed by Domingos (2012). On the coarsest level, machine learning feeds data into a machine learning algorithm (e.g. logistic regression). The learning algorithm often factorizes into a hypothesis space, which is the space of all functions that this algorithm can learn (e.g. linear functions) and a method to search that space for a good hypothesis that is consistent with the data. Usually, the search method can be factorized into an objective function (e.g. correctly classifying the training samples) and an optimization method that moves through the hypothesis space in order to optimize the objective (e.g. gradient descent).

For the logistic regression example, we have talked about the data and its representation as a matrix $\mathbf{X}$ and the hypothesis space that includes all linear class boundaries by representing f as $\sigma(\mathbf{X}\theta)$. Accordingly, each hypothesis is represented by a vector θ, where all possible vectors span the hypothesis space. We will now look at how we search for the right hypothesis by defining an objective function and an optimization method. The objective function measures the *fit* of a hypothesis to the data. The optimization method changes θ in order to optimize the objective.

We cannot directly use classification error as the *objective function* (also called *loss function*) for gradient-based learning because it is not differentiable with respect to the parameters θ. Therefore, we optimize data likelihood instead, which allows us to compute how to change θ to improve the learning objective.

The data likelihood is the product of the probability of all samples according to the learned function $p(D;\theta) = \prod_i p(y^{(i)}|x^{(i)};w) = \prod_i y^{(i)} - f(x^{(i)})$. For numerical stability and well-behaved gradients, we maximize the logarithm of the data likelihood. And since objectives are typically formulated as loss functions that are minimized during learning,

logistic regression uses the negative log likelihood as a loss function,

$$L(D, \theta) = -\sum_i \log\left(y^{(i)} - f(\mathbf{x}^{(i)}; \theta)\right).$$

We minimize this loss function using gradient descent. After initializing the parameters θ, we repeat the following steps until convergence: Using our current parameters θ, we predict the class probabilities of our training samples and compare this prediction to the ground truth to compute the loss L. We then change our parameters by taking a small step in the direction of steepest descent of the loss function with respect to the parameters of our hypothesis $\nabla_\theta L$. The step size is controlled by the gradient magnitude and the learning rate α, which is a small positive number e.g. 0.001:

$$\theta := \theta - \alpha \nabla_\theta L.$$

Following this procedure, logistic regression fits a hypothesis θ by repeatedly updating it based on data in order to optimize the objective function (see Fig. 2.3a). The final hypothesis (a point in the hypothesis space) corresponds to a line in feature space that separates the two classes.

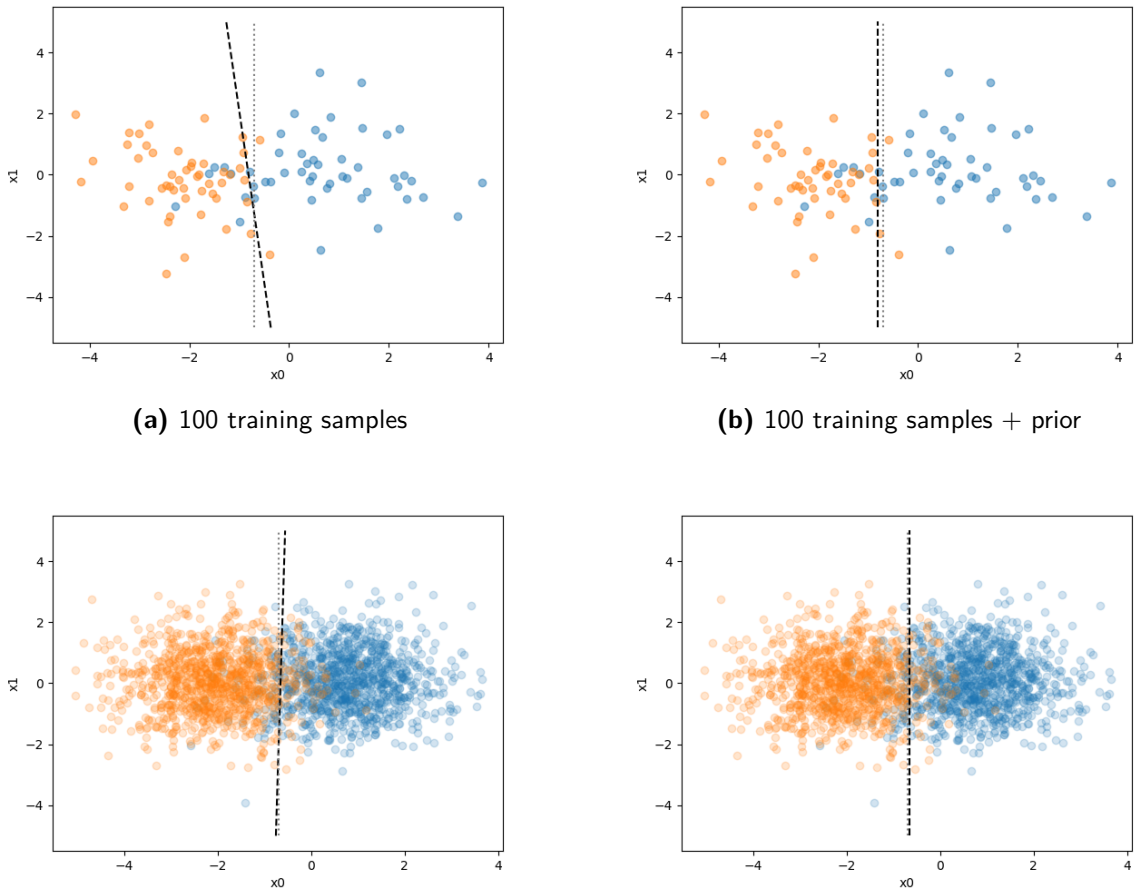
2.3.3 Overfitting and the Bias/Variance Dilemma

Although the parameters are optimized for the training set, the *goal* of machine learning is to achieve good performance, not on the training set, but on an unseen test set. A common problem is *overfitting*, where the learned parameters work well on the training set but do not generalize to the test set.

One way to remedy overfitting is to collect more training data, which reduces the need for generalization—infinite training data from the same distribution as the test data would cover all test samples. Unfortunately, training and test distributions can be different and the amount of training data is limited. Therefore, we need to regularize learning by incorporating assumptions, which can reduce the test error—if they are correct.

Geman et al. (1992) introduced the *bias/variance dilemma* which states that the test error can be decomposed into two parts, bias and variance, where incorrect assumptions lead to high bias and too few assumptions lead to high variance. While methods with high variance overfit to the data (low training error, high test error), biased methods underfit the data (high training error, high test error).

The authors conclude that "the bias/variance dilemma *can* be circumvented if one is willing to give up generality, that is, *purposefully* introduce bias", where the bias they refer to in this statement must be "harmless for the problem at hand". To avoid confusion from using the term "bias" for a component of the test error as well as for a correct



(a) 100 training samples

(b) 100 training samples + prior

(c) 1000 training samples

(d) 1000 training samples + prior

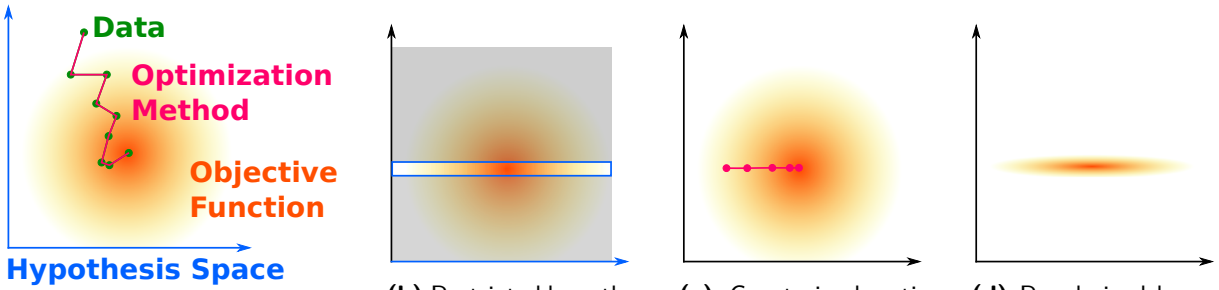
Figure 2.4: Effect of training set size and prior knowledge that x_1 is irrelevant on learned hypothesis (dashed line) vs. true hypothesis (dotted line)

assumption about the problem, I refer to such harmless biases as priors.

2.3.4 Encoding Priors in the Four Components

Let us return to the logistic regression example and assume that we know that the class label only depends on feature x_0 while feature x_1 is irrelevant. In other words, we know that the correct hypothesis must be a *vertical* line in feature space (see Fig. 2.1). By injecting this prior into our logistic regression approach, we can reduce overfitting and learn a hypothesis that is much closer to the true one. This difference is strongest with small amounts of training data (see Fig. 2.4).

But how can we encode this prior into logistic regression? Priors can be encoded in any of the four components that we have identified earlier: data, hypothesis space, optimization method, and learning objective. Depending on the prior, some of the components might be more suitable than others. In this case we can indeed incorporate the prior "x_1



(a) Machine learning components

(b) Restricted hypothesis space

(c) Constrained optimization

(d) Regularized learning objective

Figure 2.5: Every machine learning ingredient can be used to incorporate prior knowledge.

is irrelevant" into any of these four components:

1. Data: We can preprocess the data by removing feature x_1 and only learn the weight for feature x_0 and the bias term.

2. Hypothesis space: We can restrict the hypothesis space to fix weight $\theta_1 = 0$ (see Fig. 2.5b).

3. Optimization method: We can constrain the optimization method such that it initializes θ_1 as 0 and only updates θ_0 during optimization (see Fig. 2.5c).

All three ways of encoding this prior lead to exactly the same result. The final implementation is different.

4. Learning objective: We can add a regularization term to the learning objective that discourages high values of θ_1, e.g. $L_{\text{reg}}(D, \theta) = L(D, \theta) + \lambda\theta_1^2$, where λ is the weight of the regularization term (see Fig. 2.5d). Note how this implementation of the prior is not a hard constraint but a soft objective, which allows for some flexibility to violate our assumption that θ_1 should be 0 if it leads to a better fit to the training data. The parameter λ controls the relative importance of the regularization compared to fitting the training data.

These approaches of encoding priors fit into the categorization of *inductive biases* (Mitchell, 1980) into *restriction biases* and *preference biases*. Restriction biases (1.-3.) are hard constraints that restrict the hypothesis space. Preference biases (4.) express preferences for certain hypotheses, e.g. in the form of learning objectives (Mitchell, 1997, p. 64).

2.4 Machine Learning Problems and Methods

So far, this chapter has focused on the role of prior knowledge in machine learning and presented a perspective on machine learning that forms the foundation of my thesis. This section provides additional machine learning concepts that play a role in this thesis. This section can of course only give a *very* rough overview of machine learning. My goal is to explain how different machine learning concepts relate to each other. For a more detailed introduction, I would like to point the reader to the excellent books that were written on this topic, which are also the source for the content of this section (Mitchell, 1997; Sutton and Barto, 1998; Friedman et al., 2001; Bishop, 2006; Goodfellow et al., 2016).

Machine learning generally considers three different categories of *learning problems*: supervised, unsupervised, and reinforcement learning. I will first provide some intuition about these categories and then describe in detail each problem category and mention standard machine learning methods that are applied to problems in those categories. The main difference between these three categories is the available data and the general learning objective.

Supervised learning, which is the problem category for the example we have discussed, assumes data in the form of input/output pairs and aims to learn a function that can predict the correct output for new inputs. This category is referred to as "supervised" because the outputs used for training, which are also called *labels* or *targets*, can be thought of as being provided by a supervisor that knows the function that we want to learn. To give an illustrative example, supervised learning is like learning to pass a written drivers license test from a stack of practice tests that were correctly filled out by a supervisor.

Unsupervised learning differs from this by not assuming any labels in the data. To reuse the written test analogy, in unsupervised learning, the learner would only receive the practice tests but not the correct answers. Since these *unlabeled data* do not provide enough information to learn how to pass the test, unsupervised learning has a different objective. The goal here is to learn something about the structure of the data, for example that each test question consists of a picture, a short text, and a list of possible answers or that there are clusters of questions that appear similar to each other.

Reinforcement learning is in some sense in between these two categories because the data for reinforcement learning includes a *reward* signal that provides more learning feedback than in unsupervised learning but not as much as in supervised learning. Returning to our driving test example for the last time, here the learner could fill out practice tests and receive feedback about how many points she received—but not what the correct answer would have been. The learner's goal is to maximize the sum of rewards, which in this case is the test score. But from incorrect answers she does not learn which answer would have been the right one and therefore needs to learn from trial and error. This also means

that the training data in reinforcement learning is generated through the interaction of the learner (also called *agent*) and its *environment*.*

Note how these categories relate to the four machine learning components. They categorize machine learning in terms of data and learning objectives, but not in terms of the hypothesis space and the optimization method. I will now discuss each category in more detail and provide an overview of machine learning methods that are applied in each category.

2.4.1 Supervised Learning

As stated in Section 2.3.1, the objective of supervised learning is to infer a function $f : \mathbf{x} \mapsto y$ from a set of labeled training data $D = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{n}$. Supervised learning with continuous labels, $y \in \mathbb{R}$, is called *regression*. Supervised learning with discrete labels, $y \in \mathbb{N}$, is called *classification*. Labels in classification are also called *classes*. For classification, f typically first computes a probability for each discrete class y, which I will denote as $p_y(\mathbf{x})$, and then returns the class that is most likely, $f(\mathbf{x}) = \arg\max_y p_y(x)$. To learn classification using gradient-based approaches, we optimize the probability rather than the choice based on it.

The objective in supervised learning is to make predictions that are close to the true labels. The most common objective functions are *mean squared error* between the prediction and the label for regression and the *log likelihood* (also *cross-entropy*) loss for classification, which represents the likelihood of the true label.

$$L_{\text{mse}}(D, \theta) = \sum_i \left(y^{(i)} - f_\theta(\mathbf{x}^{(i)})\right)^2,$$
$$L_{\text{loglh}}(D, \theta) = -\sum_i \log\left(p_{y^{(i)},\theta}(\mathbf{x}^{(i)})\right),$$

where the θ denotes the learnable parameters. We fit θ to the data D by minimizing the appropriate loss function,

$$\theta^* = \arg\min_\theta L(D, \theta),$$

using gradient descent (as described in Section 2.3.2),

$$\theta := \theta - \alpha \nabla_\theta L.$$

Depending on the hypothesis space that we want to consider, f can have many forms, leading to different learning methods.

*For didactic reasons, this example oversimplifies some aspects of reinforcement learning, which I cover in more detail below.

If we define f to be a linear function $f(x) = \mathbf{x}^T\theta$ and train the parameters by minimizing mean squared error, we get *linear regression*, which is the basis for many other learning methods. Replacing the linear function with a polynomial function leads to *polynomial regression*.

As demonstrated in the example, we can also learn linear functions to separate two classes in a binary classification problem. This approach extends linear regression by applying a sigmoid nonlinearity $\sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}}$ to the result of the linear function z, which results in a method called *logistic regression*. We can generalize this idea to more than two classes by replacing the sigmoid with a softmax nonlinearity $\sigma_k(\mathbf{z}) = \frac{e^{\mathbf{z}_k}}{\sum_l e^{\mathbf{z}_l}}$, which is called *softmax regression*.

Methods that used to be called *artificial neural networks* and are now called *deep learning* (LeCun et al., 2015) are another generalization the ideas above. Standard feedforward networks are stacks of alternating linear functions and nonlinearities (such as sigmoid or piece-wise linear functions).

The idea of gradient-based learning can be applied to arbitrary hypothesis spaces as long as the output of f is differentiable with respect its parameters. As a result, neural network research has constructed a wide range of architectures that define different hypothesis spaces, including *convolutional neural networks* (CNNs, LeCun et al., 1989), and *recurrent neural networks* such as *long short-term memory networks* (LSTMs, Hochreiter and Schmidhuber, 1997). Recently, the field has focused on training networks which are many layers "deep" and renamed itself to *deep learning* to emphasize that the depth of the new networks is the reason for improved performance. The intuition behind this claim is that depth introduces hierarchy as a prior by allowing the network to learn many layers of representations that decompose the problem hierarchically.

There are of course many other supervised learning algorithms. I will briefly mention three particularly common methods for classification. First, *support vector machines* (SVMs, Boser et al., 1992) can also be viewed as an extension of linear regression that replaces the learning objective above with a *maximum-margin* loss and applies the *kernel trick* to learn non-linear functions.

Second, there is a very simple learning algorithm that does not use gradient-based optimization: The *k-nearest neighbors* method (KNN, Fix and Hodges Jr., 1951) "learns" by storing all training samples. It makes predictions for new inputs by finding the k stored samples that are nearest to the new input and returning the label that is most common among these k neighbors.

Third, we can also learn *decision trees* (Breiman et al., 1984) without using gradient information. Decision trees recursively partition the input space into different regions. To make a prediction with a decision tree, we find the region of the new input and report the class probability of the training samples that fell in that same region. Decision trees are learned greedily one partition at a time by choosing the partition that minimizes

the expected classification error. We can also train an *ensemble* of randomly perturbed decision trees and average their result, which is called *random forests*.

2.4.2 Unsupervised Learning

Unsupervised learning only has access to *unlabeled* data $D = \{\mathbf{x}^{(i)}\}_{i=1}^{n}$. The goal of unsupervised learning is to find some description of the data, e.g. by compressing the data set. Similar to supervised learning, compression in unsupervised learning can also be split into a continuous case, which is called dimensionality reduction, and a discrete case, which is called clustering. *Dimensionality reduction* aims to produce a lower-dimensional continuous representation of the data, while *clustering* partitions the data into multiple discrete clusters.

Dimensionality reduction can be applied to high-dimensional data assuming that it does not span the entire space but lies on a lower-dimensional manifold. Dimensionality reduction methods attempt to find that manifold and project the data onto it. If we restrict the hypothesis space to orthogonal linear transformations and define the learning objective to maximize the variance along the manifold, we get *principal component analysis* (Hotelling, 1933). This approach can be generalized to non-linear hypothesis spaces in the form of neural networks using an *auto-encoder* architecture that projects the data non-linearly onto a manifold, projects it back into the original space, and is trained minimizing the reconstruction error (Bengio et al., 2007). Instead of maximizing variance or reconstruction, we can also change the learning objective in a way that the manifold should consist of dimensions (or features) that change slowly over time, which gives us the *slow feature analysis* (SFA, Wiskott and Sejnowski, 2002).

Clustering aims to group training samples into multiple clusters such that similar samples are assigned to the same group. The simplest clustering method is k-means (Jain and Dubes, 1988), which iteratively assigns samples to the nearest cluster-mean and then recomputes the mean based on all assigned samples. We use this method in Chapter 5 to generate features for reinforcement learning.

Instead of compressing the data, we sometimes want to estimate the probability density $p(x)$ from the training samples. We can do this using *kernel density estimation* (Rosenblatt, 1956), which represents $p(x)$ as a mixture of Gaussians with one Gaussian at every data point. Rather than learning an explicit representation of the probability distribution, we can also learn a generative model that we can sample from $\hat{x} \sim p(x)$, which is called *generative modeling*. *Generative adversarial networks* (GANs, Goodfellow et al., 2014) approach this problem by jointly training a generator and a discriminator. The generator generates samples and the discriminator classifies samples into "samples from the training set" and "samples from the generator". The generator is trained to minimize the discriminator accuracy and the discriminator is trained to maximize it.

2.4.3 Reinforcement Learning

Reinforcement learning (RL, Sutton and Barto, 1998) is the problem of learning how to maximize future rewards by interacting with the environment. The standard formalization of this problem is a Markov decision process (MDP, Bellman, 1957), which is a tuple (S, A, T, R, γ) that describes the state space S, the action space A, the transition function T, the reward function R, and the discount factor γ. The term "Markov" in MDPs refers to the following property of the state: given the current state, all future state transitions and rewards are independent of past interactions. In other words: the state summarizes all information from past interactions that the robot needs to select the best action.

Based on the current state $s \in S$, the agent chooses and executes an action $a \in A$, obtains a new state $s' \in S$ according to the transition function $T : S \times A \to S$ and collects a reward $r \in \mathbb{R}$ based on the reward function $R : S \to \mathbb{R}$. The agent's goal is to learn a policy $\pi : S \to A$ that maximizes the expected return $\mathbf{E}(\sum_{t=0}^{\infty} \gamma^t r_t)$, with r_t being the reward collected at time t and $0 < \gamma \leq 1$ the discount factor.

There are a number of different approaches to reinforcement learning that differ, for example, a) in their representation of the policy π and b) in an explicit model of the environment, i.e. whether they learn T and R.

a) Policies can be represented *explicitly* as a function that maps states to actions or *implicitly* as the action choices that maximize a *value function*. Methods that operate on these representations are called *policy-based* methods and *value-based* methods, respectively. Value based methods are able to exploit the recursive definition of the optimal value function (Bellman equation) by performing dynamic programming (Sutton and Barto, 1998). Policy-based methods, on the other hand, allow to include certain priors in an intuitive way by specifying the policy representation. Methods that use both representations are called *actor-critic methods*, where *actor* refers to the explicit policy representation and *critic* refers to the value function.

b) Policies can be learned *model-based* or *model-free*. In model-based learning, the agent has or learns a model that predicts which reward and next state result from taking an action in a given state. It then samples from this model to plan the policy. In model-free learning, the robot uses the experienced samples directly to learn the policy, without building a model first. The main advantage of explicitly learning a model is that it can be used for different tasks (different reward functions but same state transitions), while the policy is only valid for the task it was learned for. Having explicit models also allows to directly include priors about the world by restricting the hypothesis space accordingly.

The reinforcement learning framework can be generalized to partially observable Markov decision processes (POMDPs, Kaelbling et al., 1998), where the agent cannot directly perceive its state s_t but only an observation o_t that depends on s_t. To determine which actions to choose, the agent must take the entire history of observations, actions, and rewards

into account. The MDP and POMDP frameworks are able to describe the interactive loop between the agent and the physical world. They are, therefore, well-suited for formalizing many learning problems in robotics.

2.4.4 Machine Learning Problems in This Thesis

My thesis approaches perception problems in robotics based on all three problem settings.

Chapter 3 Learning Object Segmentation Through Task-Specific Priors employs the supervised learning setting. The goal is to learn to segment objects in images based on training data of correct segmentations.

Chapter 4 Learning State Estimation Through Algorithmic Priors adopts the POMDP setting from reinforcement learning but is not concerned with learning a policy. Instead, it is about learning to estimate the state from a sequence of observations and actions using supervised learning based on state labels.

Chapter 5 Learning State Representations Through Physics-based Priors has the same goal of estimating the state in a POMDP setting, with a particular focus on enabling reinforcement learning. But unlike the previous chapter, it does not assume any state labels and applies an unsupervised learning approach instead.

2.5 Priors in Machine Learning

We have now seen why machine learning needs priors, how they can be encoded, and what classes of machine learning problems and methods exist. In the beginning of this chapter, I stated that such methods are the implementation of priors. So far, I have not discussed *which* priors they are implementing.

This section will categorize the most important priors in machine learning. These priors are often used implicitly. Making them explicit is the goal of this section. I divided these priors into two groups, although this division is not clear-cut: *ubiquitous machine learning priors*, which form the basis of machine learning, and *generic AI priors*, which are encoded by a smaller fraction of machine learning methods.

2.5.1 Ubiquitous ML Priors

Ubiquitous ML priors are so deeply ingrained in the machine learning approach that it is difficult to imagine machine learning without some of them. I will formulate these priors in terms of machine learning components.

Vector Representation

Inputs and outputs are represented as vectors.

Virtually all machine learning methods require the data to be *represented as vectors*, such that a sample is denoted by a fixed number of real values. Based on this representation, they learn a function from a fixed length input vector to a fixed length output vector.

In order to apply the machine learning machinery to a problem, we need to force this problem into a vector representation—e.g. represent text by a *bag of words*, represent graphs by a fixed number of properties, and represent classes by a *one-hot vector*. These transformations into vector space can add information, e.g. by choosing which graph properties are important features, or it can remove information, e.g. the word order in bag of words or inter-class relations in one-hot representations. But only through this representation can we apply machine learning.

IID Data

Data samples are independent and identically distributed (IID).

Most machine learning methods assume data samples to be *IID*, that is each sample is assumed to be drawn from an *identical* unknown distribution *independently* of the other data points (Friedman et al., 2001).

In particular the assumption that training and test set are identically distributed is crucial to machine learning. Machine learning methods degrade quickly when test data is drawn from a different distribution $p(x)$ than during training. This assumption is also essential in how we assess generalization of our machine learning methods.

Smoothness

Similar inputs produce similar outputs.

The *smoothness prior* is the basis of learning because it allows us to generalize from the observed data to new inputs, which have not been seen before, based on training samples that are similar to those. The opposite of smoothness is chaos. In a *chaotic system*, slight changes in the initial conditions can have large impact on the system behavior, which makes prediction very difficult.

The smoothness prior is encoded in the hypothesis spaces of common machine learning methods. Nearest neighbor and kernel methods predict the test labels from labels of the most similar training examples. Decision trees partition the input space into convex regions with associated labels. Linear functions and neural networks interpolate in the input space or some nonlinear representation of it.

Simplicity

Simpler hypotheses are preferable over complex ones.

The *simplicity prior*, also called the *law of parsimony* or *Occam's razor*, is another central prior in machine learning. It controls the complexity of the hypothesis that is induced from a limited amount of data. The prior assumes that simpler hypotheses generalize better than unnecessarily complex ones to new data.

As discussed earlier, there is no objective measure for simplicity. Simplicity has been defined in terms of number of parameters, parameter magnitudes, code length, etc. The simplicity prior has been implemented in equally many ways, for example by choosing hypothesis spaces with few parameters, using small initial values, using early stopping, defining a Gaussian prior probability on the parameters, or—equivalently—using L2 regularization.

The simplicity prior is also related to assuming equations of *low polynomial order*, which is a plausible assumption given that such polynomials accurately describe the physical laws in our universe (Lin and Tegmark, 2016). This assumption is used particularly in classical statistics that often rely on linear models.

Good Local Optima

The gradient of the learning objective leads to good local optima.

Many machine learning methods rely on gradient-based optimization to find the best hypothesis. Since this optimization approach is local, it can only find local optima. For this approach to work, we must assume a *good local optimum*, such that following the gradient of the learning objective leads to a hypothesis that has a similar loss as the global optimum. This assumption seems to be true for high-dimensional parameter spaces and neural networks in particular.

Dauphin et al. (2014) present empirical evidence for the argument that "in contrast to conventional wisdom derived from low dimensional intuition, local minima with high error are exponentially rare in high dimensions." Choromanska et al. (2015) provide a theory for why this result holds for neural networks and conclude that "minima lie in a band [of similar performance] which gets smaller as the network size increases".

These results indicate that high-dimensional parameter spaces (and neural networks in particular) match the implicit assumptions of gradient based optimization, which could partly explain why they work so well.

2.5.2 Generic AI Priors

I adapted the term *generic AI priors* from Bengio et al. (2013), who argued that the key to successful learning is the incorporation of "many general priors about the world around us." The following list of generic AI priors is based on their work and on the work of Lin and Tegmark (2016) who investigated machine learning from a physics perspective. I will formulate these priors in terms of the world.

Independent Properties

Things are composed of independent properties.

This *independent properties* prior assumes that properties which describe the world around us are meaningful independently of each other (Bengio et al., 2013). If one property, such as the color of an object, changes, the other object properties, size, weight, position, etc., retain their meaning.

Let me illustrate what this means for learning with another example: By observing a cat for the first time, we can not only learn something about cats, but also about how things with four legs move, what fur looks like, that mouths produce sounds, etc. When we then see a dog for the first time, we can immediately make certain predictions based on properties that it shares with the cat, e.g. that it moves similarly because it has the same number of legs, even though we have never seen this combination of properties before.

This might sound trivial, but many machine learning methods—such as nearest neighbor approaches, decision trees, Gaussian SVMs—are not able to make this kind of generalization because they learn about local regions in the input space. They only learn about certain combinations of properties but not about the properties themselves. Therefore, those methods cannot generalize to combinations that have not been observed during training.

Such generalizations require the assumption of independent properties, which can be encoded into machine learning using *distributed representations*, which are the core of linear methods and neural networks. Distributed representations can allow an exponential gain in learning efficiency (Bengio et al., 2013), because they allow to learn from a single example not just about the specific combination of properties that it represents but about each property individually, which allows non-local generalization of experience.

Hierarchy

Things are structured hierarchically.

Lin and Tegmark (2016) argue that our universe has *hierarchical structure*, where causally complex structures result from the combination of simpler structures. They give the example that "elementary particles form atoms which in turn form molecules, cells, organisms, planets, solar systems, galaxies, etc".

This prior can be encoded by replicating hierarchical structure in the learning method. Such a structure must turn the concrete into abstract representations through multiple levels of abstractions in a way that allows features at every level to be shared between different parts of the representation of the next level. Bengio et al. (2013) argue that is this sharing and reuse of features allows deep learning to disentangle useful representations.

The idea of shared representations is also the motivation of multi-task learning (Caruana, 1997), which has recently been applied to reinforcement learning (Jaderberg et al., 2016; Mirowski et al., 2016). The same assumption also explains why convolutional networks that were trained on one data set provide features that are useful for other vision-related tasks (Razavian et al., 2014).

The common idea is that the assumption of hierarchy allows effective reuse of features, which increases data-efficiency.

Symmetry

Some properties are invariant under certain transformations.

Symmetry priors assume transformations that do not affect the property at hand (Lin and Tegmark, 2016). Common invariant transformations are translation and rotation. Many properties in the world are similar regardless of their spatial position and orientation. Cats look similar in Berlin and Boston, walk the same way, and make the same sounds. Encoding such invariances into learning drastically reduces the number of parameters.

Symmetry priors are heavily used in computer vision because the content of an image is invariant to, for example, reflection, rotation, translation, and changes in illumination. A common way to encode these symmetries is to augment the training set with transformed versions of the original samples (Krizhevsky et al., 2012). Tobin et al. (2017) used the related idea of domain randomization to learn object detection in a simulator and generalize to real scenes by sampling from these symmetries.

While data augmentation and domain adaptation are flexible ways for encoding symmetries, they are computationally expensive because the symmetries need to be learned instead of being encoded in the hypothesis directly. But some symmetries have been implemented by restricting the hypothesis space. Convolutional networks, for example,

implement translational invariance by spatial weight sharing of the convolution parameters (LeCun et al., 1989).

Another important symmetry is time invariance, the assumption that certain properties do not change over time, e.g. that the transition function in an MDP remains the same.

Locality

Things directly affect only what is in their vicinity.

If all things in the universe could directly influence each other, predictions would require global information about everything. The *locality prior* states that "things directly affect only what is in their immediate vicinity." (Lin and Tegmark, 2016).

There are two types of locality: spatial locality and temporal locality. We can exploit *spatial locality* to reason about the collision of objects, which is only possible if they are close to each other. Spatial locality also applies to image pixels; neighboring pixels are more likely to belong to the same entity than pixels that are farther away. This is why local image filters can capture useful information, such as in convolutional networks (LeCun et al., 1989).

The same concept applies in the time dimension. The past affects the future only through a stream of temporally local effects. Temporal locality can be encoded in sliding window approaches that use information from the last k time steps to make a prediction (Dietterich, 2002). Temporal locality is only the basis of models with Markov states such as MDPs, temporal difference learning, dynamic programming, recurrent neural networks, and Bayes filters.

2.5.3 The Missing Priors

These priors are the condensation of our understanding of machine learning. I agree with Bengio et al. (2013) that refining such a list of priors and incorporating it into machine learning will bring us closer to artificial intelligence. Since we have not achieved that goal, we must ask the question: What priors are we missing?

If we view these priors as a form of subjective physics from the perspective of a robot living in and learning about the real world, it is striking that all priors above are *passive*. They all apply to an disinterested observer learning about the world and making predictions. Although these priors are very helpful in this context, their passivity limits the amount of prior knowledge that they can capture.

Since robots are not disinterested but have certain tasks, and since they can *interact* with the world, there is a wealth of prior knowledge that has not been exploited. I call these *robotic priors*.

In the three main chapters of this thesis, I will propose three sources for robotic priors: the *task*, *algorithms*, and *physics*. For each of these sources, I will define specific priors, encode them into a machine learning approach to a perception problem, and demonstrate their effectiveness for improving generalization.

3

Learning Object Segmentation Through Task-Specific Priors

3.1 Introduction

This chapter focuses on the problem of learning object segmentation for the Amazon picking challenge. The key to solving this problem was to encode prior knowledge about the task into the object segmentation method. The resulting method was data-efficient, robust, and an essential component of our winning entry in this challenge.

The 2015 Amazon picking challenge (APC) was a warehouse logistics challenge that required robots to fulfill an order by autonomously recognizing and picking twelve objects from the bins of a warehouse shelf (see Fig. 3.1). Each bin contained between one and four objects, selected from a set of 25 known objects. The goal was to pick as many of the twelve target objects as possible in 20 minutes (without picking wrong objects). The robot had to be completely autonomous—no human control or intervention was allowed. Our research group and 25 other international teams from academia and industry competed in this challenge. While the challenge includes many subproblems such as hardware design, grasping, and control, a post-APC survey among all participating teams found that perception was the most difficult aspect of this challenge (Correll et al., 2016). Object perception—identifying and localizing target objects—is what this chapter will focus on. We will see how object perception can be simplified by exploiting prior knowledge about the task of the Amazon picking challenge.

From own experience, I know that it is difficult to achieve good performance in this task by applying existing object perception techniques. Our initial tests of off-the-shelf libraries for object recognition, segmentation, and pose estimation revealed substantial

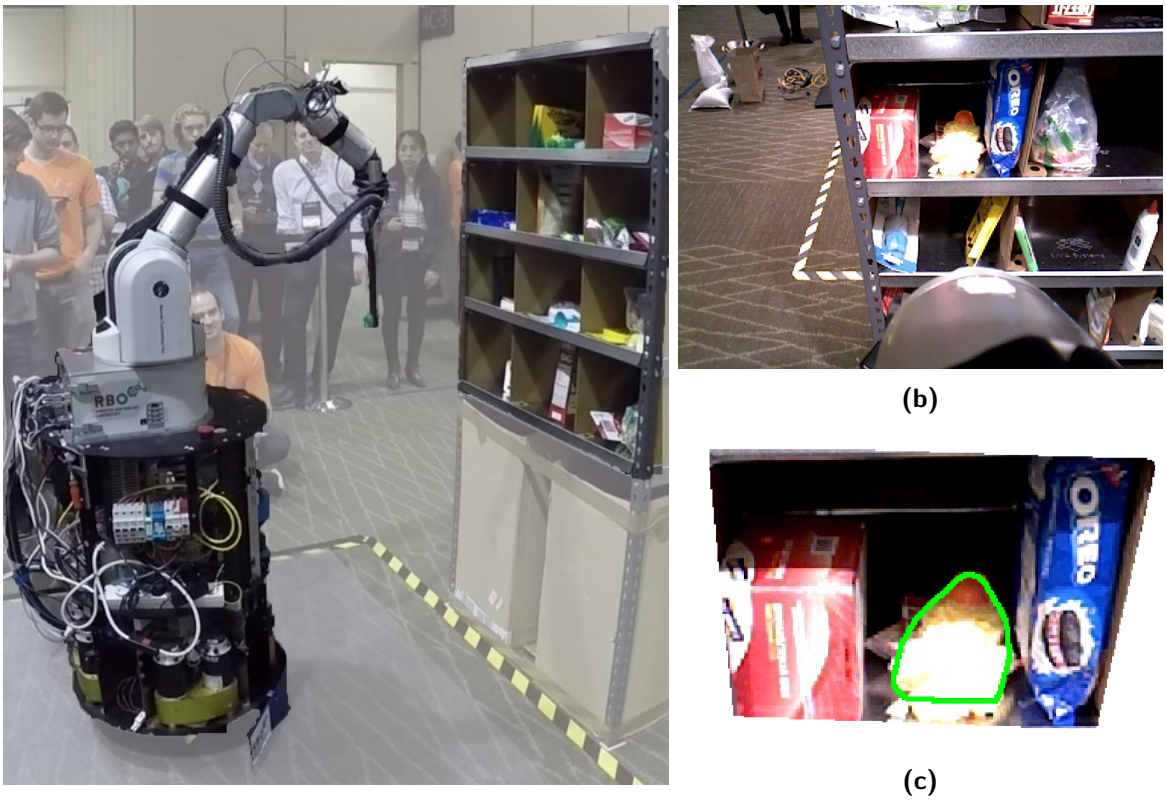


Figure 3.1: Our robot in the Amazon picking challenge: (a) the robot looks for an object in the shelf; it observes the scene from an RGB-D camera that is mounted on its forearm; (b) view from the RGB-D camera; (c) result of the segmentation of the target object (*duck toy*) (©2016 IEEE)

shortcomings in the APC setting. This difficulty stands in contrast to the availability of excellent open source libraries, such as PCL and OpenCV, and occurred for a seemingly simple perception problem. Rennie et al. (2015) performed more detailed tests of LINEMOD, a standard object detection and pose estimation algorithm, which showed only 32% accuracy when applied to the APC setting. Even if they tailor the method to the APC setting, it still only achieves 60% accuracy.

If research teams from academia and industry cannot leverage the achievements of decades of computer vision research in the context of the APC, we must question our assumptions. *Can we achieve generic and task-agnostic perception for robotics?* In our own APC solution, we deviated from the trajectory of solving perception problems in their most general form. Instead, we were successful with a simple perception method that exploits knowledge about the task at hand. Our method probabilistically combines a number of simple color and depth features, designed to take advantage of the characteristics of the warehouse setup. To localize the target object, our system performs object segmentation and classification on an RGB-D image (see Fig. 3.1c).

This task-specific multi-class segmentation approach is a key component of our challenge entry that ultimately enabled us to win the APC. During the competition, our method segmented and identified all twelve objects correctly, which enabled the robot to successfully pick ten of them outperforming all 25 other teams. A detailed description of the complete system can be found in our systems paper (Eppner et al., 2016). Here, I will focus on object perception, in particular segmenting the target object by leveraging task-specific priors.

Of course, the resulting segmentation method is task-specific and tailored to the APC. However, by making these task-specific priors explicit and analyzing their contribution, we can make progress in the direction of more general solutions. Towards this goal, I performed an extensive post-hoc evaluation of our method. I hope that the results provide valuable insight for perception beyond the specific warehouse logistics setting and might help to make progress towards more robust robotic perception.

3.1.1 Contributions

In this chapter, we* make the following contributions:

Conceptual Contributions

- We analyze the task of object perception in the Amazon picking challenge to identify task-specific priors and demonstrate their implementation in an object segmentation

*Throughout this thesis, I will use the first-person plural narrative *we* to describe collaborative work because all work has been collaborative at least with my advisor. I will use the first-person singular narrative *I* to distinguish statements of my own, e.g. in introductions and conclusions.

method.

Technical Contributions

- We propose a simple, data-efficient, and robust task-specific object segmentation method for the Amazon picking challenge. The source code and data are available at: `https://gitlab.tubit.tu-berlin.de/rbo-lab/rbo-apc-object-segmentation`

Empirical Contributions

- We show that task-specific priors can enable data-efficient learning of robust object perception. Our object perception method generalized successfully to a setting outside of the lab environment, a feat which most other teams in the challenge struggled with.
- We perform an extensive post-challenge analysis that reveals the contributions of the different components of our method. We find that a higher amount of machine learning would have improved performance, tight-combination between segmentation and classification is important, missing information can be a source of information, and simple visual post-processing can outperform complex reasoning.

3.1.2 Outline

The rest of this chapter is organized as follows:

- **Section 3.2 Related Work** provides an overview of existing object perception methods.
- **Section 3.3 Task-Specific Priors** discusses the concept of task-specific prior knowledge and its potential for data-efficient learning in robotics. It also analyses the APC setting to extract task-specific priors that can simplify object perception for this task.
- **Section 3.4 Task-Specific Object Perception** introduces our object segmentation approach for the APC and describes its use in the perception system of our challenge entry.
- **Section 3.5 Experiments and Results** reports the perception results in the challenge and performs a detailed post-challenge analysis that investigates of error distribution across different scenarios and comparisons to baselines and different variants of our method.
- **Section 3.6 Conclusion** summarizes this chapter and discusses how to make progress towards more general robotic perception through prior knowledge.

3.2 Related Work

This section gives an overview of related work in object perception.

The perception problem in the APC is an instance of the general object detection and segmentation problem, which is actively researched in computer vision. Results of popular vision competitions such as PASCAL VOC (Everingham et al., 2014) or ImageNet (Russakovsky et al., 2015) show that solutions related to our problem (such as single-object localization, detection, or segmentation) currently receive significant attention and winning entries steadily reduce error metrics over the years. A common theme among those entries are sliding-window approaches using deformable parts models (Felzenszwalb et al., 2010) or deep neural networks in combination with large-scale datasets (Szegedy et al., 2013). They give as output bounding boxes with highly likely object locations, which contain many pixels that are not part of the object. This renders these representations difficult to use in a robotic manipulation context, where accurate, or at least conservative, shape estimates are crucial to decide on appropriate actions.

Multi-class segmentation addresses this problem by identifying for each pixel in an image to which object class it belongs. A popular approach to multi-class image segmentation are conditional random fields (CRFs, Lafferty et al., 2001). They encode local (per-pixel or region) and pairwise statistical preferences, and define an energy whose minimum corresponds to the most likely segmentation. CRFs provide a principled way to integrate different sources of information (e.g. color, texture, shape, location, and depth) and can easily incorporate smoothness constraints (Shotton et al., 2006; Müller and Behnke, 2014). Similar to CRFs, our approach combines different sources of information in a probabilistic fashion. A comparison between a generic CRF and our method is shown in Section 3.5.2.

A more classical yet effective approach to object segmentation is histogram back-projection (Swain and Ballard, 1991). Given the color histogram of the target object, the method back-projects the histogram into the image by replacing each pixel color with the respective bin count of the histogram. Areas with high bin counts are then assumed to be the target object. Our work extends histogram back-projection to a probabilistic version and also incorporates additional non-color features.

In the context of robotic manipulation, approaches to object detection usually aim at estimating the full 6D object pose, and therefore rely more heavily on depth data. Detection and pose estimation can be based on CAD models (Klank et al., 2009), feature point histograms (Rusu et al., 2009, 2010), local keypoint descriptors like SIFT and SURF (Collet et al., 2011), or edge and normal information to address textureless objects as done in LINEMOD (Hinterstoisser et al., 2012). These approaches are based on table top assumptions and do not scale well when confronted with the limited visibility and clutter imposed by the APC setup. For example, LINEMOD (Hinterstoisser et al., 2012) shows already significant translational error with only two items per bin (Rennie et al., 2015).

Although we do not estimate the 6D pose of objects, our results show that the information contained in the segmentation is sufficient for our system to pick successfully.

3.3 Task-Specific Priors

Before describing our approach to object perception in the APC task, we will discuss the idea behind task-specific priors in robotics and relate it to other learning approaches that exploit such priors. Afterwards, we will analyze the APC task for perception challenges and useful problem structure that can be used as task-specific prior knowledge.

Ultimately, robots should be able to solve a wide range of tasks. Since we cannot envision all tasks that robots could potentially have to carry out, they will have to learn these tasks as they go. Does this mean that—in order to make progress towards the goal of task-versatility—we should avoid any assumptions about the robot's tasks?

Even if we want our robots to solve a wide range of tasks, assumptions about these tasks might still be very helpful. For example, there are tasks that are so common that we can anticipate them, such as fetching objects for a human or assembling an object from its pieces. It might be useful to have a "bag of tricks" to handle such tasks most efficiently. There is also structure inherent in a range of tasks, e.g. common subtasks such as picking up, moving, and placing objects or the fact that many tasks can be defined by a goal state such as a certain arrangement of objects. Therefore, it might make sense to have shared behaviors for common subtasks (Schaal, 2006) and adapt these behaviors based on object poses (Pastor et al., 2009).

While we want to exploit prior knowledge about the probable tasks, we also want to give the robot the flexibility to learn new tasks and adapt existing solutions based on experience. Therefore, we need to combine task-specific priors with machine learning. This combination must exploit things that we know about the task in order to make learning—of the things that we do not know—more efficient. How can we approach this problem? Let us look at examples from the literature where researchers have combined task-specific priors and machine learning.

The most common example of using task-specific priors in machine learning is *feature engineering*. If we know which features of the sensory input are task-relevant, we can provide the learning method with these features instead of the raw sensor data. Here, we are essentially solving the first part of the problem by hand (identifying the right features) and only apply learning to the second part (learning feature patterns). This approach has been heavily used to enable efficient learning, in particular in robotics where large amounts of data is difficult to obtain (Kober et al., 2013). For example, the features used by Abbeel et al. (2010) for learning acrobatic helicopter flight from expert demonstrations "take advantage of symmetries of the helicopter" by representing positions, velocities and

accelerations in the body frame of the helicopter. Riedmiller et al. (2007) use task-specific features for autonomous driving such as the cross-track-error, its time derivative, and 3 other features as input for reinforcement learning. As a result they are able to learn to steer a real car after only 20 minutes of training.

Similarly to preprocessing the input of the function we want to learn, we can also post-process the output. The autonomous car example above (Riedmiller et al., 2007) integrates the output of the reinforcement learning controller over time to produce the steering angle. It is also possible to provide robots with task-specific high-level actions. Konidaris et al. (2018) use a set of defined skills to learn a symbolic representation for high-level planning from raw sensory input. These assume the second part of the behavior to be fixed and learn the first part. Naturally, both approaches can also be combined.

Using a fixed pre-processing or post-processing are only two instances of embedding a learning method in a larger system that contains components that are tailored to the task. By defining these components rather than learning them from data, we are restricting the hypothesis space for machine learning (as explained in Section 2.3.2), which improves learning efficiency. To efficiently apply machine learning to a specific robotic task, we need to analyze the task in order to understand which assumptions can be made and then encode these assumptions into a learning method. The rest of this section performs this analysis for object perception in the APC.

3.3.1 Task Analysis: The Amazon Picking Challenge

The task in the Amazon picking challenge consists of autonomously picking twelve out of 25 objects (see Fig. 3.2) from a warehouse shelf and placing them into a storage container (see Fig. 3.1) within 20 minutes. The robot knows which objects are contained in each of the shelf's twelve bins, but not of their exact arrangement inside the bin.

Evaluation Criteria: For each successfully picked target object, the robot receives 10, 15, or 20 points, depending on how many additional objects were in the same bin (between none and three). Objects that are considered difficult to pick grant up to three bonus points. Picking the wrong object results in -12 points.

Figure 3.2: The 25 picking objects from the challenge

Objects: The 25 competition objects vary widely in size and appearance (see Fig. 3.2). Objects range from small spark plugs to bulky boxes, from hard cases to soft plush toys, and from loose objects to those wrapped in plastic. This variety presents a challenge for grasping and perception.

Environment: The items are placed in twelve different bins of the shelf. The robot is allowed to operate in a $2\,m \times 2\,m$ area in front of this shelf. The bins have a rather small opening ($21\,cm \times 28\,cm$) but extend quite far to the back ($43\,cm$). The small opening restricts grasping opportunities, in particular for bins that contain multiple objects.

The environment also poses significant challenges for perception. Due to the narrow bins, objects are often visible from one side only and partially obstructed. The floors of the shelf are made of reflective metal, rendering color and depth information unreliable. During the actual challenge, the lighting conditions were particularly difficult due to very bright spot lights directly above the competition area: objects in the front of each bin appeared to be effectively white, while objects in the back appeared as nearly black.

Perception Challenges in the APC Setting

To identify task-specific priors for perception in the APC setting, we analyze the problem in more detail. We identify challenges and useful problem structure for overcoming these challenges.

No Single Discriminative Property for All Objects: The 25 APC objects were chosen to reflect the large variety present in a warehouse scenario. No single perceptual feature suffices for identification: some objects have distinctive shapes, others are deformable; some objects have distinctive colors, others have similar color histograms, or view-dependent variations; some objects have surfaces amenable to our RGB-D sensor, others are wrapped in plastic bags. We address this problem by combining a variety of features.

Limited Object Visibility: Camera-based perception can only obtain a partial view of an object in the shelf from a particular camera position. Nearby objects sometimes partially occlude others. We address this challenge by training on perceptual data from objects in different poses.

Uncontrolled Lighting: Visual perception is sensitive to lighting conditions. At the challenge venue, the lighting was directly from above and extremely bright, relative to the ambient light. In these conditions, images were nearly saturated in bright regions and appeared black in the remaining ones. To alleviate this problem we transform the RGB image to HSV color space and include features based on depth.

Partial 3D Measurements: Kinect-like sensors do not provide reliable 3D measurements for reflective or translucent materials, such as the plastic-wrapped objects or the metal shelf of the APC. We turn this problem into a source of information by using missing depth values as a feature for segmentation.

Useful Problem Structure in the APC Setting

Small Number of Known Objects: Since the complete set of objects was available and known beforehand, it was possible to collect training data from these objects in different orientations and bin locations.

Few Objects per Bin: Since bins contained at most four known objects, we can ignore all other objects during segmentation. Our method automatically uses the most discriminative features for the particular subset of objects present in a bin.

Known Shelf: Since the objects are placed in a known shelf that can be tracked by the robot, we can use shelf-related features, such as the height of a pixel in the bin or the distance to the tracked shelf model. These features help to discriminate objects of different sizes and to differentiate between objects and the shelf.

3.4 Task-Specific Object Perception

3.4.1 Object Perception Pipeline

To successfully pick specific objects, the robot must recognize and locate the target object. Our system captures images with an RGB-D camera mounted on the robot's forearm and performs three steps: 1. *feature extraction*, 2. *object segmentation*, and 3. *bounding box fitting*. We first extract a number of task-specific features for every pixel of the RGB-D

image. Using statistics about these pixel-features for each object, we find the image segment that has the highest probability of belonging to the target object. We take the point cloud for this segment and fit a bounding box with size of the target object to it. The fitted bounding box allows the robot to decide where and from which direction it should pick up the object.

We will now explain these three steps of our object perception pipeline before giving additional details on the features and the object segmentation.

Feature Extraction

To extract features from the target bin, the robot tracks the shelf in the depth image using the iterative closest point method (ICP, Chen and Medioni, 1992). Based on the tracked shelf, it crops the RGB-D image to only show the target bin and computes the following six features for each pixel: color, visual edges, distance to the tracked shelf model, height above the ground plane of the shelf bin, height in the image plane (for pixels without depth information), and presence/absence of depth information.

These features discriminate between most objects and the shelf. Instead of searching for features that could solve the general object recognition problem, these task-specific features rely on strong assumptions (e.g. that objects are placed in a known shelf) in order to simplify perception. As a result of these assumptions, our perception system was able to handle the difficult lighting conditions during the Amazon picking challenge, non-rigid objects, partial views, and incomplete depth data.

Object Segmentation

Based on manually segmented training images, our method precomputes histograms for each feature and object. It uses these histograms to estimate how likely each feature value in the current image is for each object. By multiplying these likelihoods and normalizing them per pixel, we compute the probability of each pixel in the current image to belong to a specific object. We compute these probabilities for all objects that we know are contained in the target bin and for the bin itself. Our method then smooths these probability images, labels each pixel corresponding to the most probable object, and selects the segment that includes the pixel with the maximum probability for the target object. In the last step, it makes the size of the segment consistent with our expectation and greedily segments objects in sequence, eliminating already segmented objects.

Bounding Box Fitting

The segment estimated by the previous step is now transformed into a point cloud representation, filtered for outliers, and used to fit a bounding box. The fitted bounding box is then compared to the true dimensions of the target object to match the sides of the object correctly. The result of this step is an approximate estimate of position and orientation.

3.4.2 Features

We will now provide a detailed description of the features that we used for segmentation. Based on the analysis from Section 3.3.1, we describe each pixel by six features that jointly discriminate between the objects:

Color: A discrete value in the range $0 - 182$ based on the hue-saturation-value (HSV) color representation, which is relatively invariant to lighting conditions. We project the HSV color space to a single dimension by thresholding appropriately: we set the feature to H (ranging from $0 - 179$) for pixels with distinctive color (S $>$ 90 and V $>$ 20), and otherwise to 180 for white (V $>$ 200), to 181 for gray (200 $>$ V $>$ 50), and to 182 for black pixels (50 $>$ V).

Edge: A binary feature that describes whether the pixel is in the vicinity of a visual edge. We compute this feature by applying Canny edge detection to the image and dilating it with a small elliptical kernel (with a diameter of 11 pixels).

Missing 3D: A binary value representing whether a pixel contains valid depth and therefore 3D information.

Distance to shelf: A continuous value (in mm) that denotes the distance of a pixel to the closest point on the shelf. We estimate this value by tracking the shelf in the RGB-D image. For this we start with an estimate based on the localization and forward kinematics of the robot and refine it using ICP. Pixels without valid depth information are ignored.

Height (3D): A continuous value (in mm) that denotes the shortest distance of a pixel to the ground plane of the bin, computed similarly to the distance-to-shelf feature. Pixels without valid depth information are again ignored.

Height (2D): A continuous value (in mm) that describes the height of the pixel projected onto the (open) front plane of the shelf bin. This feature approximates 3D height and is only used for pixels without valid depth information.

3.4.3 Object Segmentation

We will now describe our multi-class segmentation method (video link: `https://youtu.be/Ry6JzeW0HOM`) in detail. We will first describe how feature statistics can be learned from data and then explain how these statistics are used for segmentation.

Learning Phase

Given a 6D feature vector per pixel, we now explain how to learn the likelihood of the features for the APC objects.

Data Collection: During the preparation of the APC, a dataset was made available that included RGB-D images of all objects from multiple views together with estimated 3D models of the objects.* However, we found that differences in the cameras, viewing angle, and lighting conditions made it difficult to transfer models from this dataset to our robot. Moreover, in this dataset the objects are not inside the APC shelf. Therefore, the likelihood of some shelf dependent features as well as a model for the shelf itself could not be learned from this dataset.

We therefore generated a dataset which closely resembled the competition scenario. We placed the objects in the shelf in different poses and collected RGB-D images and the estimated shelf pose. Finally, we manually segmented the images until we had a sufficient number examples for each object to cover their possible poses (161 samples in total, on average 6 samples per object).

Computing Feature Likelihoods: Based on our dataset, we generate feature likelihoods for each object $o \in O$. For each feature f (e.g., color, height), we compute a histogram from the pixels that belong to the hand-labeled object segments and normalize this histogram to get a likelihood $P(X^{(f)}|O = o)$ over the possible values $x^{(f)} \in X^{(f)}$. To be robust against small changes in feature values, we smooth non-binary likelihood functions with a Gaussian kernel (standard deviations for the smoothing kernel: $\sigma_{\text{color}_{0-179}} = 3$, $\sigma_{\text{color}_{180-182}} = 1$, $\sigma_{\text{dist to shelf}} = 7.5mm$, $\sigma_{\text{height(3D)}} = 3mm$, $\sigma_{\text{height(2D)}} = 6mm$). For robustness to large changes in feature values, we mix the likelihoods with uniform distributions, $P(X^{(f)}|O = o) \leftarrow p_{\text{uni}_f} P_{\text{uni}}(X^{(f)}) + (1 - p_{\text{uni}_f}) P(X^{(f)}|O = o)$, where we use the following parameters for the different features: ($p_{\text{uni}_{\text{color}}} = 0.2$, $p_{\text{uni}_{\text{dist to shelf}}} = 0.05$, $p_{\text{uni}_{\text{edge}}} = 0.4$, $p_{\text{uni}_{\text{miss3D}}} = 0.2$, $p_{\text{uni}_{\text{height(3D)}}} = 0.4$, $p_{\text{uni}_{\text{height(2D)}}} = 0.8$). Thus, even feature values that have never been observed for an object have non-zero probability and do not entirely rule out certain objects. The parameters σ_f and p_{uni_f} define how much we trust feature f.

*`http://rll.berkeley.edu/amazon_picking_challenge/`

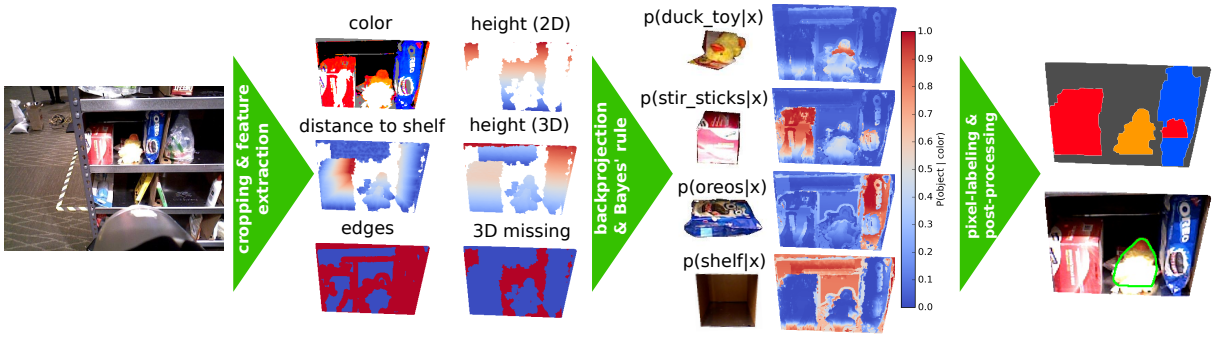


Figure 3.3: Overview of the multi-class segmentation phase of our approach (©2016 IEEE)

Segmentation Phase

The learned feature likelihoods are the base of our multi-class segmentation phase, which is illustrated in Figure 3.3.

Cropping and Feature Extraction: In the first step of the multi-class segmentation phase, we crop the RGB-D image to only contain the bin with the target object. This step removes clutter and distracting objects from other bins. We then compute the 6D feature vector described in Section 3.4.3 for each pixel in the cropped image (see Fig. 3.3). Note that estimating the cropping mask and some of the features (e.g. height) relies on tracking the pose of the shelf.

Back-Projection and Bayes' Rule: In this step, we compute for each object o in the bin and every pixel i the probability that this pixel belongs to the object given its feature vector $\mathbf{X}_i$, $P(O_i = o|\mathbf{X}_i)$. This results in one *posterior image* per object (see Fig. 3.3). To compute them, we iterate over all features f and back-project their likelihoods $P(X^{(f)}|o)$ into the image, i.e., we replace the feature values $x_i^{(f)}$ with their likelihood $P(x_i^{(f)}|o)$, similar to Swain and Ballard (1991). Assuming conditional independence between the features, we multiply their likelihoods for each pixel: $P(\mathbf{X}|o) = P(X^{(\text{color})}|o)P(X^{(\text{height(3D)})}|o)\ldots$. Then, we apply Bayes' rule by multiplying $P(\mathbf{X}|o)$ with an object prior $P(o)$ and normalizing each pixel (see Fig. 3.4). We use a flat prior across all objects except the shelf, which we set to be three times as high so that the method assigns uncertain pixels to the shelf segment rather than to the wrong object.

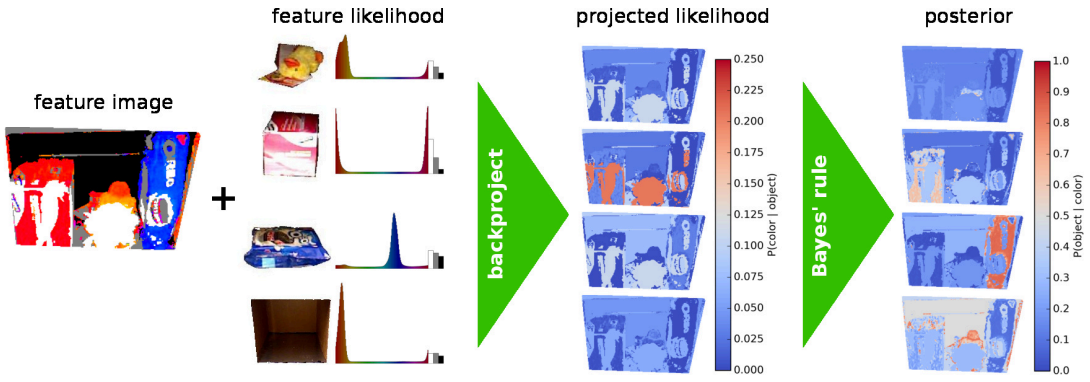


Figure 3.4: Likelihood back-projection and computation of the posterior by the example of the color feature. (©2016 IEEE)

Pixel-Labeling and Post-Processing: Nearby pixels often display the same object and should therefore have similar object probabilities assigned to them. To incorporate such spatial information, we smooth each object's posterior image with a Gaussian kernel ($\sigma = 4$ pixels). This step is related to locating an object by convolving its back-projected histogram with a disk (Swain and Ballard, 1991) and to pairwise potentials in CRFs. The smoothing step evens out single pixels or small regions of much higher or much lower probability than the surrounding area, which makes the segmentation more robust. Here we implicitly exploit that the APC objects are compact and occupy a significant area of the image.

Based on this smoothed posterior image, we label each pixel i as belonging to the object o with the highest posterior $P(o|\mathbf{X}_i)$ and extract connected regions that are assigned to the same object. In case of having multiple disconnected segments for an object, we select the one that includes the maximum in the smoothed posterior image for that object.

As a post-processing step, we make the segment convex. This step incorporates missing object parts and reflects the convexity of most APC objects. Additionally, we look at the size of the segment (number of pixels) and compare it to segment sizes for this object in our dataset. If the segment is considered too large to be correct (larger than 1.2 times its maximum size in our dataset), we reduce its posterior image (by subtracting 0.05) and reassign the object labels. We repeat this until the segment shrinks to a plausible size.

The last post-processing step is a greedy re-labeling based on the following idea: If we are confident about the segmentation of one object, we do not have to consider this object for the rest of the image. We exploit this by sequentially segmenting the objects, greedily starting with the object that we are most certain about, where we measure certainty by segment size. If the segment size is consistent with our dataset, we assume that we have found the correct segment of this object with high probability, reduce its posterior outside of the segment accordingly (by multiplying it with 0.2) and re-normalize. We proceed in the same way with the next most certain object and continue until the target object has

been processed.

3.5 Experiments and Results

We evaluated our method on a dataset that contains 346 manually segmented RGB-D images. The training set includes 161 samples (about six per object) recorded in our lab in Berlin. Our test set consists of three parts: a) three runs (66 samples) recorded in our lab, b) five runs (107 samples) recorded in the challenge venue in Seattle, and c) the actual APC run (12 samples). Unless indicated otherwise, all of the following experiments use (b) as test set. Both our implementation and the dataset are publicly available at `https://gitlab.tubit.tu-berlin.de/rbo-lab/rbo-apc-object-segmentation`.

We measure performance on each test sample by segmenting the target object in the presence of other known objects in the same bin. For each sample, we compare the predicted segment with the manually annotated ground truth. From this comparison, we compute *precision* and *recall* and average them across samples.

Precision describes which portion of the predicted segment is correct. Recall describes which portion of the true segment was predicted. To get a single performance measure, we compute their weighted geometric mean. We use the $F_{0.5}$-score to emphasize precision over recall, because in our task finding even a small portion of the correct object will often allow the robot to grasp it successfully.

3.5.1 Performance Evaluations

Performance at Amazon Picking Challenge

Figure 3.5 shows the result of applying our method in the actual competition run of the 2015 APC at ICRA in Seattle (video link: `https://youtu.be/DuFtwpxQnFI`). Based on the 3D point cloud of the estimated segment, the robot computed a bounding box of the target object, chose the side from which to pick the object, moved its end-effector towards the center of the bounding box until contact, and picked up the object with a vacuum gripper (Eppner et al., 2016). Our system outperformed the other 25 teams by successfully picking ten out of the twelve objects. The robot only failed in two cases. In bin (f), it accidentally picked the plush eggs instead of the spark plug due to an inaccurate picking motion. In bin (k), the robot could not remove the big cheezit box because it got stuck.

Figure 3.5: Segmentation results during the APC run; the green line outlines the segments returned by our method; all segments lie on the correct objects; mean precision: 91%, mean recall: 73%, $F_{0.5}$ score: 0.864 (©2016 IEEE)

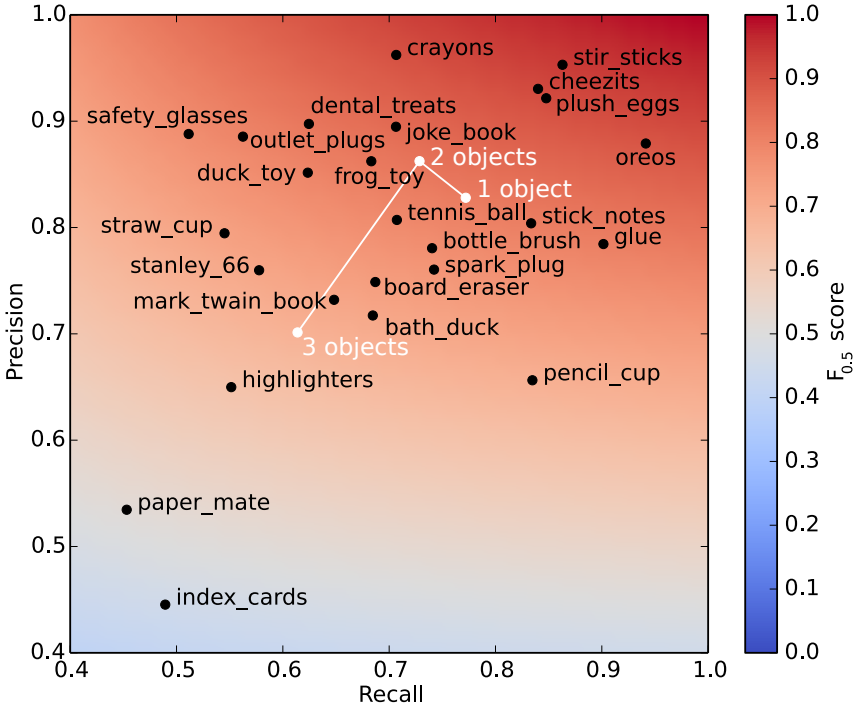


Figure 3.6: Performance by object (black) and number of objects in bin (white) (©2016 IEEE)

Performance by Object

Figure 3.6 shows the performance of our method for different objects (black text) and for different numbers of objects per bin (white text). Our method had most problems with flat objects that are dominated by white color, e.g. the index cards (see Fig. 3.5 (d) to the right), the paper mate (see Fig. 3.5 (c) to the right), and the highlighters (see Fig. 3.5 (e) to the right). Our method most reliably segmented large objects with distinct colors, e.g. the stir sticks (see Fig. 3.5 (c) to the left), the cheezits (see Fig. 3.5 (k)), and the plush eggs (see Fig. 3.5 (f) to the right). These results are consistent with our findings in Section 3.5.3, which show that color is in fact the most important feature.

The number of objects per bin also has a strong impact on the performance of our method (see Fig. 3.6) because with more objects in the bin, the features we are using, like color or height, become much less discriminative.

Figure 3.7 shows typical failure cases: (a) part of an object with similar features is mistaken for the target object, (b) close objects with similar features lead to inaccurate object boundaries, (c) reflections are included in the target object.

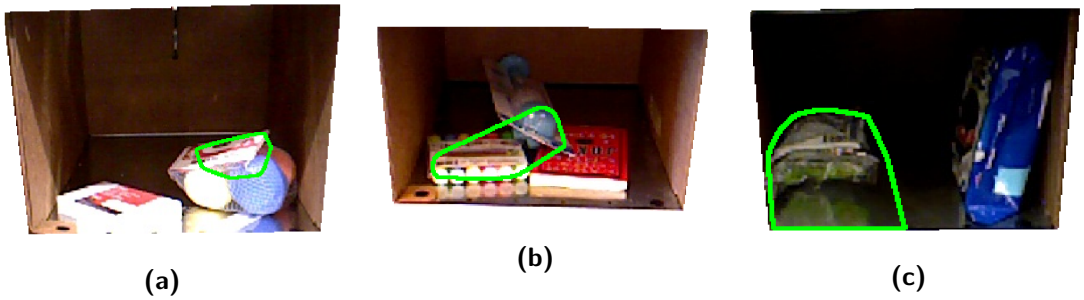


Figure 3.7: Typical failure cases (green line outlines segments found by our method); from left to right: segment detected on the wrong object, segment spanning over several objects, reflections considered as part of an object (©2016 IEEE)

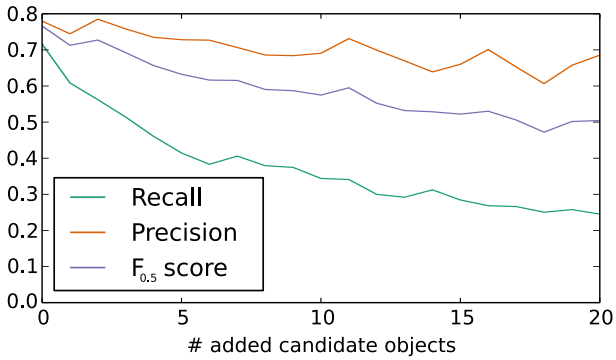


Figure 3.8: Performance trend when the assumption about known objects is reduced (increasing the candidate objects); recall decreases quickly but precision stays relatively stable (©2016 IEEE)

Increasing the Number of Objects per Bin

Our method can be easily scaled to thousands of possible objects as long as there is only a small number of known objects in every bin. As Figure 3.6 suggests, the performance degrades with increasing numbers of candidate objects in the bin. We explicitly augment the list of candidate objects in the bin with objects that were not present. The results in Figure 3.8 show a decrease in recall whereas precision remains relatively stable.

3.5.2 Comparison to CRF

We compared our method against a widely used generic approach to multi-class segmentation: a CRF based on RGB-D input that uses a learned classifier to estimate the pixel-wise probabilities and a predefined pairwise probability (0.99) for neighbouring pixels to have the same label. We used a random forest (Breiman, 2001) as the classifier and regularized it by setting a minimum size of leaves to a fraction of the total number of samples (see Fig. 3.9).

This experiment shows that the baseline works for the data that it was trained on, but does not generalize well to unseen data. Generic regularization does not solve this problem. Compare this to the performance of our method which stays almost constant

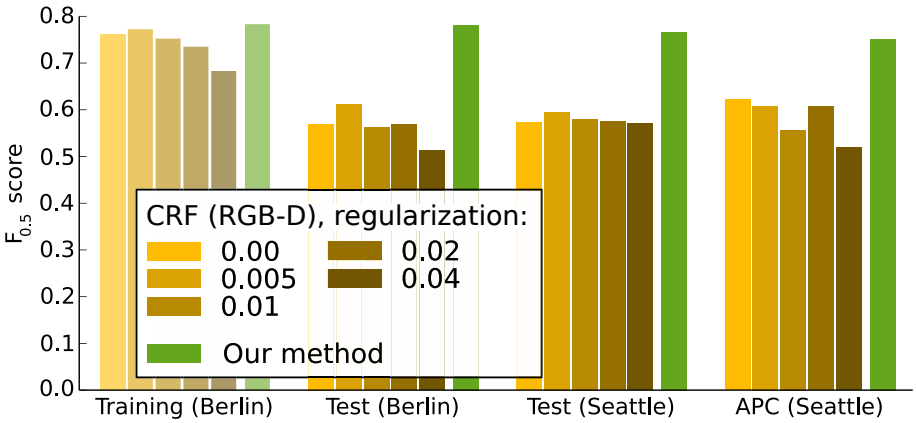


Figure 3.9: Comparison to a generic conditional random field (CRF) with different amounts of regularization; the unary potentials are obtained from a random forest (©2016 IEEE)

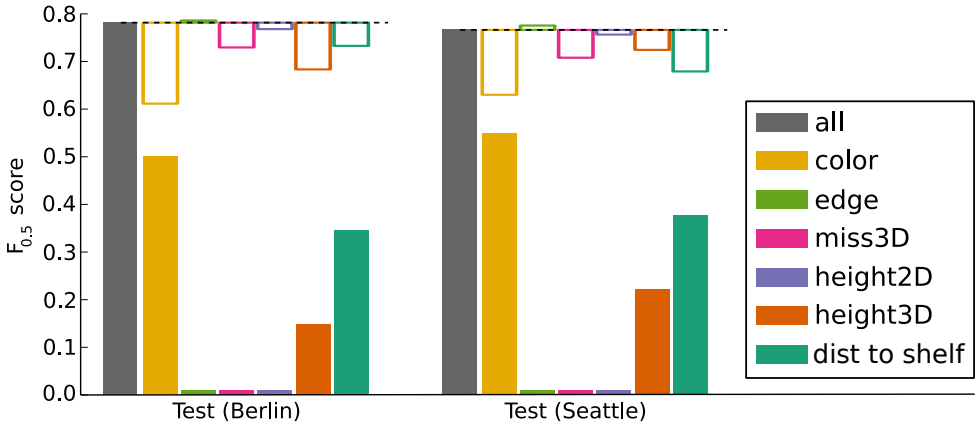


Figure 3.10: Contribution of different features to the overall performance; the gray bar indicates our performance using all features; the filled bars show the performance based on a single feature alone; the empty bars show the change in performance if the corresponding feature is removed and the remaining features are used in the model; in both cases, the size of the bars correspond to the usefulness of the features (©2016 IEEE)

when going from training to test set and even when going to data collected in Seattle under very different lighting conditions.

3.5.3 Variants of our Method

The next experiments evaluate the contribution of the different parts of our algorithm to the final result.

Features

In this experiment we evaluate the importance of each feature to the overall performance of the algorithm (see Fig. 3.10). First, we evaluated the performance of our algorithm *using only one* of the features. We can observe that none of the features alone reaches a performance comparable to using all of them together. However, color is a powerful feature by itself, because it is sufficient to discriminate well between many objects. Using

only the distance to shelf or the height (3D) also obtains relatively good scores because these features discriminate very well between objects and the shelf, which already solves single object bins. The features edge, missing 3D information, and height (2D) alone were not able to produce any segmentation. In the second experiment, we *deactivated one* of the features and measured the performance drop. Again color is the most crucial feature. We observe that the missing 3D feature, which could not produce any segmentation when used alone, contributes substantially to the performance. Interestingly, the height (2D) features seems unnecessary and the edge feature even hurts performance.

Pixel Labeling and Selection

For this experiment, we divide the multi-class segmentation process into two steps: *pixel labeling* and *selection*. The pixel labeling step assigns an object label to each pixel, which creates image regions of the same label, possibly disconnected. The second step selects one of these regions as the final segment.

We now compare different methods for these two steps. First, we compare four variants of the pixel labeling step. 1. *Max*: assigning to each pixel the label of the most likely object. 2. *Max smooth*: the same as max, but smoothing the probability image first. 3. *Simple graph cut*: formulating the labeling in terms of energy minimization: pixel probabilities are turned into potentials and connected to their four neighbors with a high probability (0.99) of having the same label. Then, we apply graph-cut to find the optimal partition of the graph into labeled segments. 4. *Graph cut using depth edges*: similar to simple graph cut, but we adapt the pairwise probabilities depending on depth differences: very high (0.997) for neighbors with similar depth, high (0.95) if there is no depth information, and low (0.5) for neighbors that cross depth edges. Second, we compare three variants for the selection step: 1. a naive approach of selecting *all* segments, 2. selecting the *largest* segment, and 3. selecting the segment that includes the *maximum in the smoothed probability image*.

The results show that, surprisingly, the selection method has a much larger impact on the performance than the pixel labeling method (see Fig. 3.11a). The simple *max* pixel labeling is defeated by all other methods, especially in combination with the *select-all* method. All other pixel labeling methods are comparable. Selecting all segments naturally leads to the highest recall but lowest precision. Selecting the largest segment (which is common practice) trades recall for precision. The *max smooth* selection, however, does this more efficiently. The intuition behind this difference is the following: in most cases the largest segment is also the segment that includes the maximum probability. In these cases, both methods perform equally well. But in some cases, when they are not the same, it seems to be better to trust the point with highest probability rather than the largest segment. The best combination of methods is using *max smooth* for segmentation

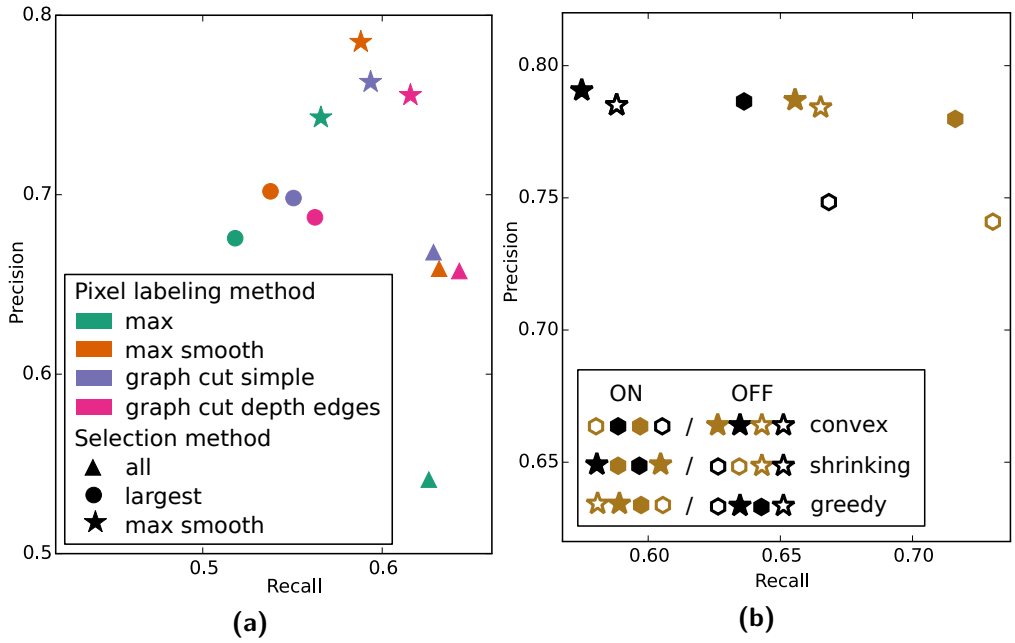


Figure 3.11: Combinations of different methods for (a) pixel labeling and selection and (b) re-labeling and post-processing. (©2016 IEEE)

and selection, which is the combination we used in the APC.

Re-labeling and Post-processing

Our method includes three re-labeling and post-processing steps: making the segment convex, shrinking it if it is too large, and greedily segmenting the image starting with the largest segment. In this experiment, we tested all combinations of these steps. The results (see Fig. 3.11) show that greedy re-labeling (golden color) substantially increases the recall irrespective of which other steps are used. Convexity (hexagon symbol) and shrinking (filled symbol) work best together: Making the segment convex increases the recall and shrinking it, if necessary, improves precision. The best performance is achieved when all three steps are combined.

Random Forest for Pixel Probability Estimation

In this experiment we extend our method to use a random forest classifier (Breiman, 2001) to estimate the posterior images instead of using likelihood back-projection and Bayes' rule. For the given set of objects in the bin, we trained a random forest to discriminate just these objects and then use the probability estimates from the random forest as posterior image in our method. Apart from this, we retained all other steps and used the same features, except that the color feature is replaced by the original HSV values to allow the random forest to deviate from the thresholds we had set manually.

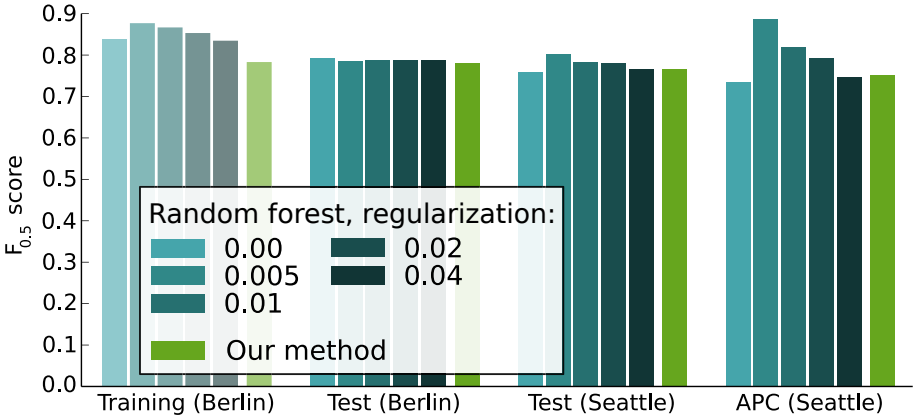


Figure 3.12: Performance with incorporating a random forest classifier into our method (©2016 IEEE)

Our hypothesis was that this hybrid method would suffer from overfitting and not reach the performance of our original method. Surprisingly, the random forest classifier improved our method (see Fig. 3.12). On the one hand, this classifier did introduce a higher variance in performance between training set and test set compared to using likelihood back-projection. This is because it is less restricted, i.e. it does not assume feature independence and can represent highly complex functions. The likelihood estimation, on the other hand, includes many manually tuned parameters, e.g. parameters for smoothing the feature likelihoods or weights for mixing them with uniform distributions, which introduces a bias. Contrary to our hypothesis, the bias we introduced is larger than the variance of the random forest classifier.

3.6 Conclusion

3.6.1 Summary

In this chapter, we used task-specific priors to learn object segmentation for the Amazon picking challenge. We implemented these priors by using task-specific features, by restricting the possible results based on the task conditions, and by probabilistically integrating many task-relevant sources of information. While our object perception method is tailored to one specific task, the findings from our experiments are more general. They can be summarized as follows:

1. Machine learning approaches can replace large parts of the hand-tuned parameters and even learn from very limited amounts of data, but only if appropriately combined with prior knowledge. This insight is reflected by the fact that using random forests with our features and post-processing outperforms the hand-tuned pixel probability estimation component (Section 3.5.3), whereas applying it to RGB-D data directly does not (Section 3.5.2). This finding is backed up by decades of applied

machine learning research, but it still has not become common practice in all areas of robot perception.

2. We must tightly connect segmentation and classification of objects, and explicitly reason about the environment, not only about the target object. This finding is clearly supported by our analysis of the influence of known objects (Section 3.5.1) and the contribution of the distance-to-shelf feature (Section 3.5.3). However, it stands in contrast to many state-of-the art approaches to robotic perception that solve subproblems fully independently, and do not use contextual information.

3. Missing information is an important source of information, as indicated by the contribution of the missing-3D-points feature (Section 3.5.3). Although computing and using this information is cheap, few methods in robot perception exploit it.

4. Our results show that simple visual post-processing can outperform complex reasoning, as exemplified by the fact that Gaussian smoothing on probability images is as effective as optimizing pairwise potentials based on depth-edges (Section 3.5.3). We believe that this finding has practical implications: the theoretically best method does not necessarily give large—if any—improvement over approximate heuristics. Thus, the decision of choosing a method for a particular problem should be supported by empirical data, and not only be based on theoretical soundness.

3.6.2 Machine Learning and Task-Generality

Our results demonstrate the potential of using prior knowledge for learning robotic perception, but they also hint at advantages of shifting more from an engineering approach towards a machine learning approach. The first potential advantage of shifting more towards machine learning is an increase in generality such that perception can be learned for different tasks. The second potential advantage is improved performance, e.g. by replacing hand-tuned parameters (see result 1.) and to tightly connect different modules also during learning (see result 2.) using end-to-end learning.

To leverage the first advantage of increased generality, we need to take into account other sources of prior knowledge that are not necessarily task-specific. To leverage the second advantage of improved performance, we need to rethink how we can use these priors in a way that maximally empowers machine learning to do what it can do well—fitting parameters to data.

In the next chapter, we will take a step in that direction. Instead of exploiting task-specific priors, we will make use of prior knowledge encoded in algorithms that apply to many tasks. We will incorporate these priors in a way that enables end-to-end learning, which optimizes all parameters for perception performance.

4

Learning State Estimation Through Algorithmic Priors

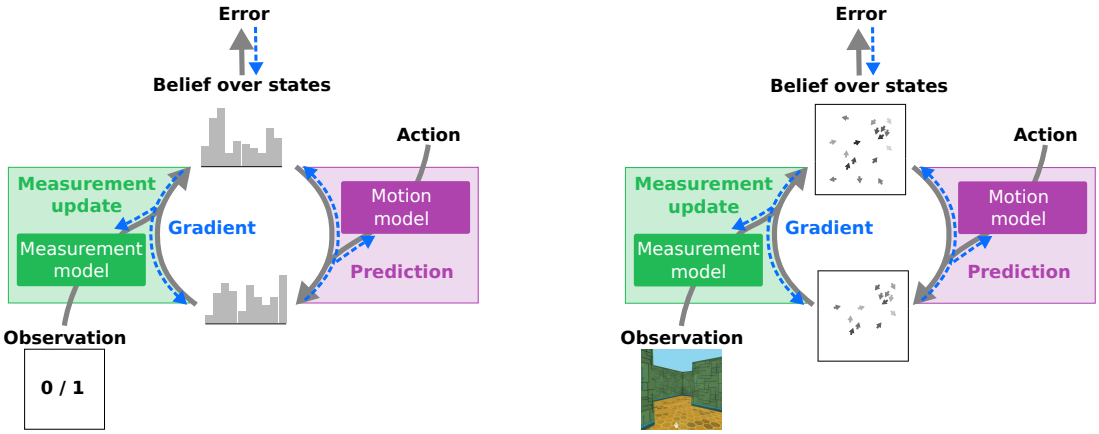
4.1 Introduction

In this chapter, we demonstrate the use of algorithmic priors for the problem of learning state estimation. This work shows how we can use algorithms as priors by implementing them in a differentiable way and fitting their model parameters to data using end-to-end learning.

End-to-end learning tunes all parts of a learnable system for end-to-end performance—which is what we ultimately care about—instead of optimizing each part individually. End-to-end learning excels when the correct learning objectives for individual parts are not known; it therefore has significant potential in the context of complex robotic systems.

Compared to learning each part of a system individually, end-to-end learning puts fewer constraints on the individual parts, which can improve performance but can also lead to overfitting. We must therefore balance end-to-end learning with regularization by incorporating appropriate priors. Priors can be encoded in the form of differentiable network architectures. By defining the network architecture and its learnable parameters, we restrict the hypothesis space and thus regularize learning. At the same time, the differentiability of the network allows us to adapt all parameters of the network in a way that maximizes end-to-end performance.

This approach has been very successful in computer vision. Highly engineered vision pipelines are outperformed by convolutional networks trained end-to-end (He et al., 2015). But this only works because convolutional networks (LeCun et al., 1989) encode priors in the network architecture that are suitable for computer vision—a hierarchy of local filters



(a) Differentiable histogram filter **(b)** Differentiable particle filter

Figure 4.1: Differentiable Bayes filters. Models can be learned end-to-end by backpropagation through the algorithm.

shared across the image. Problems in robotics possess additional structure, for example in physical interactions with the environment. Only by exploiting all available structure will we be able to realize the potential of end-to-end learning in robotics.

But how can we find more architectures like the convolutional network for robotics? Robotic algorithms capture problem structure and often use models that are specific to the task or environment. By making these algorithms differentiable and their models learnable, we can effectively turn them into network architectures. Such networks enable end-to-end learning while also encoding prior knowledge from algorithms, which we call *algorithmic priors*.

Here, we apply *end-to-end learning with algorithmic priors* to state estimation in robotics. In this perception problem, a robot needs to infer the latent state from its observations and actions. Since a single observation can be insufficient to estimate the state, the robot needs to integrate this information over time keeping track of its own uncertainty.

Given the standard assumptions in this problem, *Bayes filters* provide the provably optimal algorithmic structure for solving it (Thrun et al., 2005), recursively updating a probability distribution over states with prediction and measurement update using task-specific motion and measurement models. In this work, we implement Bayes filters in a differentiable way and train their models end-to-end, such that they optimize state estimation performance. We applied this approach to two different Bayes filter variants, which differ in how they represent probability distributions (see Fig. 4.1).

The first variant is a differentiable histogram filter (DHF), a Bayes filter that represents probability distributions with histograms. It uses feedforward networks as learnable models and implements the prediction and update steps as convolutional and multiplicative operations. Our experiments show that our method is more data-efficient than generic neu-

ral networks, improves performance compared to standard histogram filters, and—most importantly—enables unsupervised learning of recursive state estimation loops.

DHFs should be viewed as a proof of concept of end-to-end learning with algorithmic priors. While they show very promising results in simulation, they are not yet practical for real world application, mostly because of the inherent limitations of histograms when representing high-dimensional beliefs. We address this issue by moving to particle based belief representations.

Our second variant, the differentiable particle filter (DPF), extends this work to make it applicable to more realistic state estimation problems. Since DPFs represent probability distributions with samples (or particles), they focus their computational effort on states with high probability, which makes them computationally more efficient. By using appropriate learnable models, we can apply DPFs to state estimation problems based on raw visual observations, such as global localization in a maze based on camera and odometry input.

Our experiments agree with our DHF results: End-to-end learning improves performance compared to using models optimized for accuracy and algorithmic priors regularize learning, which greatly improves performance in state estimation. Compared to generic long short-term memory networks (LSTMs, Hochreiter and Schmidhuber, 1997), DPFs reduce the error rate by ∼80% or require 87% less training data for the same error rate. Additionally, even with end-to-end learning, DPFs remain explainable—we can examine the learned models and their interaction. And finally, while LSTMs fail when tested with a different policy than used for training, DPFs are robust to changing the robot's policy.

4.1.1 Contributions

In this chapter, we make the following contributions:

Conceptual Contributions

- We propose to encode algorithmic priors by making algorithms differentiable and combining them with end-to-end learning, which we demonstrate for two variants of the Bayes filter algorithm.

Technical Contributions

- We introduce differentiable histogram filters, which encode the histogram filter algorithm but allow the models to be learned end-to-end in a supervised and unsupervised way.
- We introduce differentiable particle filters, which encode the (computationally more efficient) particle filter algorithm but allow the models to be learned end-to-end. Our

source code and data are available at
`https://github.com/tu-rbo/differentiable-particle-filters`.

- As part of these implementations, we identified and developed components that should help implementing other grid- and sample-based algorithms in a differentiable manner. For grid-based representations, we implemented the prediction step by convolution and the update step by element-wise multiplication. To work with sample-based representations, we used networks to generate samples based on the reparameterization trick (Kingma and Welling, 2013) or dropout (Srivastava et al., 2014) and performed density estimation to compute the learning objective.

Empirical Contributions

- We show that algorithmic priors enable explainability and improve data-efficiency and generalization.
- We show that end-to-end learning improves performance by learning incorrect models that work in the context of the algorithm.
- We show that the combination of algorithmic priors and end-to-end learning can enable unsupervised learning.
- We show that generic neural networks learn fundamentally different behavior than differentiable algorithms.

4.1.2 Outline

The rest of this chapter is organized as follows:

- **Section 4.2 Background** explains the state estimation problem, Bayes filters, and end-to-end learning.
- **Section 4.3 Algorithmic Priors** introduces the idea of algorithmic priors and their combination with end-to-end learning.
- **Section 4.4 Related Work** gives an overviews of approaches from the literature that combine algorithmic priors and end-to-end learning in state estimation and planning for robotics.
- **Section 4.5 Differentiable Histogram Filters** describes the differentiable implementation of this grid-based Bayes filter variant and multiple approaches to train its models. This section also presents experiments, results, and limitations of differentiable histogram filters.

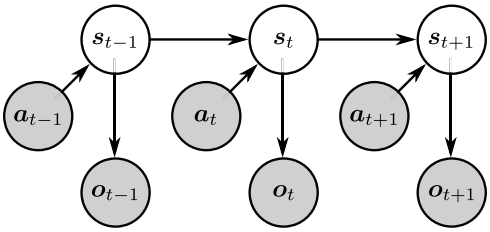


Figure 4.2: Graphical model for state estimation

- **Section 4.6 Differentiable Particle Filters** addresses these limitations by presenting a differentiable implementation of the sample-based Bayes filter and approaches to train its models end-to-end. This section also includes a comprehensive set of experiments on global localization and tracking based on simulated and real data.

- **Section 4.7 Conclusion** summarizes our findings and discusses how alternatives to end-to-end learning present an opportunity to encode additional prior knowledge.

4.2 Background

4.2.1 The State Estimation Problem

We consider the problem of estimating a latent *state* $\boldsymbol{s}$ from a history of *observations* $\boldsymbol{o}$ and *actions* $\boldsymbol{a}$, e.g. a robot's pose from camera images and odometry.* To handle uncertain observations and actions, we estimate a probability distribution over the current state $\boldsymbol{s}_t$ conditioned on the history of observations $\boldsymbol{o}_{1:t}$ and actions $\boldsymbol{a}_{1:t}$, which is called *belief*:

$$\text{bel}(\boldsymbol{s}_t) = p(\boldsymbol{s}_t | \boldsymbol{a}_{1:t}, \boldsymbol{o}_{1:t}).$$

4.2.2 Bayes Filters

If we assume that our problem factorizes as shown in Fig. 4.2, the *Bayes filter* algorithm solves it optimally (Thrun et al., 2005) by making use of the Markov property of the state and the conditional independence of observations and actions. From the Markov property follows that the last belief $\text{bel}(\boldsymbol{s}_{t-1})$ summarizes all information contained in the history of observations $\boldsymbol{o}_{1:t-1}$ and actions $\boldsymbol{a}_{1:t-1}$ that is relevant for predicting the future. Accordingly, the Bayes filter computes $\text{bel}(\boldsymbol{s}_t)$ recursively from $\text{bel}(\boldsymbol{s}_{t-1})$ by incorporating the new information contained in $\boldsymbol{a}_t$ and $\boldsymbol{o}_t$. From assuming conditional independence between actions and observations given the state follows that Bayes filters update the belief

*We denote vectors, such as states, actions, and observations, in bold to set them apart from scalar values.

in two steps: 1) *prediction* using action $\boldsymbol{a}_t$ and 2) *measurement update* using observation $\boldsymbol{o}_t$.

1) The *prediction* step is based on the *motion model* $p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t)$, which defines how likely the robot enters state $\boldsymbol{s}_t$ if it performs action $\boldsymbol{a}_t$ in $\boldsymbol{s}_{t-1}$. Using the motion model, this step computes the *predicted belief* $\overline{\text{bel}}(\boldsymbol{s}_t)$ by summing over all $\boldsymbol{s}_{t-1}$ from which $\boldsymbol{a}_t$ could have led to $\boldsymbol{s}_t$:

$$\overline{\text{bel}}(\boldsymbol{s}_t) = \int p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t)\, \text{bel}(\boldsymbol{s}_{t-1})\, d\boldsymbol{s}_{t-1}. \tag{4.1}$$

2) The *measurement update* uses the *measurement model* $p(\boldsymbol{o}_t \mid \boldsymbol{s}_t)$, which defines the likelihood of an observation $\boldsymbol{o}_t$ given a state $\boldsymbol{s}_t$. Using this model and observation o_t, this step updates the belief using Bayes' rule (with normalization η),

$$\text{bel}(\boldsymbol{s}_t) = \eta\, p(\boldsymbol{o}_t \mid \boldsymbol{s}_t)\, \overline{\text{bel}}(\boldsymbol{s}_t). \tag{4.2}$$

Any implementation of the Bayes filter algorithm for a continuous state space must represent a continuous belief—and thereby approximate it. Different approximations correspond to different Bayes filter implementations, for example Kalman filters, which represent the belief by a Gaussian, histogram filters, which represent it by a histogram, or particle filters, which represent the belief by a set of particles (Thrun et al., 2005).

Kalman Filters

Kalman filters represent all distributions—$\text{bel}(\boldsymbol{s}_t)$, $p(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_t)$, and $p(\boldsymbol{o}_t \mid \boldsymbol{s}_t)$—by Gaussian functions. Thus, the prediction step (Eq. 4.1) becomes the convolution of two Gaussians, and the measurement update (Eq. 4.1) becomes a product of Gaussians, which both maintain the belief to be Gaussian.

These Gaussian representations make Kalman filters computationally very efficient and enable their application to high-dimensional state spaces. But since Gaussians are uni-modal, Kalman filters cannot estimate multi-modal beliefs, i.e. they cannot simultaneously track multiple distinct hypotheses.

The two Bayes filter variants that we discuss next do not have this restriction. These variants will be explained in greater detail because they form the basis of the differentiable Bayes filter implementations in this chapter.

Histogram Filters

Histogram Filters partition the state space into a grid of n cells, $s_1, s_2, \ldots, s_n$, and represent the belief by a histogram with one value per grid cell $b_{i,t} = \text{bel}(S_t = s_i)$.

The histogram filter updates this distribution by moving probability mass between these grid cells and by multiplying each probability based on consistency with the measurement (followed by normalization). This representation allows histogram filters to represent distributions with multiple modes, i.e. track multiple distinct hypothesis simultaneously.

The histogram filter implements the prediction step (Eq. 4.1) by moving probability mass between grid cells based on the motion model and the current action,

$$\forall_k : \quad \bar{b}_{k,t} = \sum_i p(S_t = s_k \mid S_{t-1} = s_i, \boldsymbol{a}_t)\, b_{i,t-1}.$$

The histogram filter implements the measurement update (Eq. 4.2) by multiplying the probability for each bin with the observation likelihood at the corresponding state:

$$\begin{aligned} \forall_k : \quad & \tilde{b}_{k,t} = p(\boldsymbol{o}_t \mid S_t = s_k)\, \bar{b}_{k,t}, \\ \forall_k : \quad & b_{k,t} = \frac{\tilde{b}_{k,t}}{\sum_i \tilde{b}_{i,t}}. \end{aligned}$$

Particle Filters

Particle filters approximate the belief with particles (or samples) $\mathcal{S}_t = \boldsymbol{s}_t^{[1]}, \boldsymbol{s}_t^{[2]}, \ldots, \boldsymbol{s}_t^{[n]}$ with weights $w_t^{[1]}, w_t^{[2]}, \ldots, w_t^{[n]}$. The particle filter updates this distribution by moving particles, changing their weights, and resampling them, which duplicates or removes particles proportionally to their weight. Resampling makes this Bayes filter implementation efficient by focusing the belief approximation on the most relevant parts of the state space.

The particle filter implements the prediction step (Eq. 4.1) by moving each particle stochastically, which is achieved by sampling from a generative motion model,

$$\forall_i : \quad \boldsymbol{s}_t^{[i]} \sim p(\boldsymbol{s}_t \mid \boldsymbol{a}_t, \boldsymbol{s}_{t-1}^{[i]}). \tag{4.3}$$

The particle filter implements the measurement update (Eq. 4.2) by setting the weight of each particle to the observation likelihood—the probability of the current observation conditioned on the state represented by the particle,

$$\forall_i : \quad w_t^{[i]} = p(\boldsymbol{o}_t \mid \boldsymbol{s}_t^{[i]}). \tag{4.4}$$

The particle set is then resampled by randomly drawing particles $\boldsymbol{s}_t^{[i]}$ proportionally to their weight $w_t^{[i]}$ before the filter performs the next iteration of prediction and update.

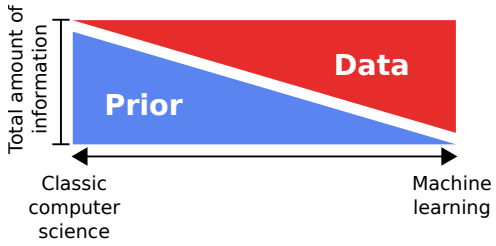


Figure 4.3: Information sources: prior and data

4.2.3 End-to-End Learning and Differentiability

If we want to use the Bayes filter structure as a prior and fit the measurement model and the motion model to data, we can essentially do one of two things: a) learn the models *in isolation* to optimize a quality measure of the model or b) learn the models *end-to-end*, i.e. train the models as part of the entire system and optimize the end-to-end performance.

In either way, we might want to optimize the models using gradient descent, for example by computing the gradient of the learning objective with respect to the model parameters using *backpropagation* (repeated application of the chain rule). Therefore, the motion model and the measurement model need to be differentiable regardless of whether we choose option a) or option b). For b) end-to-end learning, we need to backpropagate the gradient through the histogram filter algorithm (not to change the algorithm but to compute how to change the models to improve the algorithm's output). Therefore, in addition to the models, the algorithm itself needs to be differentiable.

4.3 Algorithmic Priors

Every information that is contained in the solution to a problem must either be provided as prior knowledge or learned from data. Different approaches balance these sources of information differently. In the classic approach to computer science, all required information is provided by a human (e.g. in the form of algorithms and models). In the machine learning approach, only a minimal amount of prior knowledge is provided (in form of a learning algorithm) while most information is extracted from data. When trading-off how much and which information should be provided as a prior or extracted from data, we should consider the entire spectrum rather than limit ourselves to these two end points (see Fig. 4.3).

In the context of robotics, for example, it is clear that the left end of this spectrum will not enable intelligent robots, because we cannot foresee and specify every detail for solving a wide range of tasks in initially unknown environments. Robots need to collect data and learn from them. But if we go all the way to the right end of the spectrum, we need large amounts of data, which is very difficult to obtain in robotics where data collection

is slow and costly. Fortunately, robotic tasks include rich structure that can be used as prior. Physics, for example, governs the interaction of any robot and its environment and physics-based priors can substantially improve learning (Scholz et al., 2014; Jonschkowski and Brock, 2015). But robotic tasks include additional structure that can be exploited.

Every algorithm that has proven successful in robotics implicitly encodes information about the structure of robotic tasks. We propose to use this robotics-specific information captured by robotic algorithms and combine it with machine learning to fill in the task-specific details based on data. By combining such algorithmic priors and machine learning, we can strike the right balance between generality and data-efficiency.

Algorithms and machine learning can be combined in different ways, using algorithms either 1) as fixed parts of solutions, 2) as parts of the learning process, or 3) as both.

The first approach learns models in isolation and then uses them in algorithms, e.g. in robot control based on learned motion models (Nguyen-Tuong and Peters, 2011) (note that the choice of control algorithm does not influence how the models are learned).

The second approach uses algorithms as teachers to generate training data, which is used to learn a function that generalizes beyond the capabilities of the teacher algorithm. Examples include using background subtraction to provide training data for learning image segmentation (Zeng et al., 2016) and using trajectory optimization to supervise reinforcement learning methods (Levine and Koltun, 2013).

The third approach—the one that we are focusing on in this chapter—uses the algorithms in both the learning process and the solution. The main advantage of this approach is that models are optimized for the algorithms that use them (instead of optimizing a proxy objective such as model accuracy). To achieve this, we need to make the algorithms differentiable, such that we can compute how changes in the model affect the output of the algorithm, which allows to train the models end-to-end. This idea has been applied to different algorithms, e.g. in the form of neural Turing machines (Graves et al., 2014) and neural programmer-interpreters (Reed and de Freitas, 2015). The following section gives an overview of similar approaches in robotics.

4.4 Related Work

There is a surge of recent work that combines algorithmic priors and end-to-end learning for planning and state estimation with histogram-based and Gaussian belief representations.

Planning with Known State: Tamar et al. (2016) introduced value iteration networks, a differentiable planning algorithm with models that can be optimized for value iteration. Their key insight is that value iteration in a grid based state space can be represented by convolutional neural networks. Silver et al. (2017) proposed the predictron, a differentiable embedding of the TD(λ) algorithm in a learned state space. Okada et al. (2017) proposed path integral networks, which encode an optimal control algorithm to learn continuous tasks.

State Estimation (and Planning) with Histograms: After we introduced the differentiable histogram filter (Jonschkowski and Brock, 2016), Shankar et al. (2016) and Karkus et al. (2017) combined histogram filters and QMDP planners in a differentiable network for planning in partially observable environments. Gupta et al. (2017) combined differentiable mapping and planning in a network architecture for navigation in novel environments. All of these approaches use convolution to operate on a grid based state space.

State Estimation with Gaussians: Haarnoja et al. (2016) presented a differentiable Kalman filter with a Gaussian belief and an end-to-end learnable measurement model from visual input. Watter et al. (2015) and Karl et al. (2017) learn a latent state space that facilitates prediction. These approaches use (locally) linear dynamics models and Gaussian beliefs.

In the following section, we will introduce differentiable histogram filters that operate on *histogram-based* belief representations using convolution. This work illustrates the idea of differentiable Bayes filters with a non-Gaussian belief representation, shows how they can be learned end-to-end, and demonstrates the feasibility of unsupervised learning with such methods.

Afterwards, we will build on this work and extend it to particle filters with *sample-based* belief representations. Sample-based representations can be advantageous because they can represent multi-modal distributions (unlike Gaussians) while focusing the computational effort on states of high probability (unlike histograms). But sample-based representations introduce new challenges for differentiable implementations, e.g. generating samples from networks, performing density estimation to compute gradients, and

handling non-differentiable resampling. These are the challenges that we tackle in Section 4.6.

4.5 Differentiable Histogram Filters

This section introduces differentiable histogram filters (DHFs), a differentiable implementation of the histogram filter algorithm that allows both motion model and measurement model to be learned end-to-end by backpropagation through time (Werbos, 1990). Alternatively, we can view the DHF as a new recurrent neural network architecture that implements the structure of a histogram filter (see Fig. 4.4) and thereby incorporates the Bayes filter priors.

4.5.1 Implementation

The remainder of this section describes how histogram filters can be implemented in a differentiable way and how their models can be learned in isolation or end-to-end. We will first describe how to represent the belief, then explain the two update steps and the models involved in them, and finally present different learning objectives that can be used for optimizing the models.

To comply with the deep learning framework (Goodfellow et al., 2016), we will define DHFs using vector and matrix operations. We will also introduce additional priors for computational or data efficiency. For the sake of readability, we assume a one-dimensional state space here. All formulas can be easily adapted to higher dimensions. We write specific parameter and model choices in *italic* to distinguish them from the more generic description of DHFs.

Belief

The histogram over states is implemented as a vector $\boldsymbol{b}$ of probabilities with one entry per bin,

$$\boldsymbol{b}_t = [\text{bel}(S_t = 1), \text{bel}(S_t = 2), \ldots, \text{bel}(S_t = |S|)].$$

We can also think of the belief as a neural network layer where the activation of each unit represents the value of a histogram bin. The belief $\boldsymbol{b}_t$ constitutes the output of the histogram filter at the current step t and an input at the next step $t + 1$—together with an action a_t and an observation o_{t+1} (see Fig. 4.4).

Prediction

The most direct implementation of the prediction step (which we replace shortly) defines a learnable function f for the motion model, $f : s_t, s_{t-1}, a_{t-1} \mapsto p(s_t \mid s_{t-1}, a_{t-1})$, and employs f in the prediction step (Eq. 4.1). The equation can be vectorized for computational efficiency by defining an $|S| \times |S|$ action-parameterized matrix $\boldsymbol{F}(a)$ with $F_{i,j}(a) = f(i, j, a)$, such that $\overline{\boldsymbol{b}}_t = \boldsymbol{F}(a_{t-1})\boldsymbol{b}_{t-1}$.

However, this approach is computationally expensive because it requires $|S|^2$ evaluations of f for a single prediction step. We can make this computation more efficient, if we assume robot motion to be local and invariant across the state space, i.e.

$$p(s_t \mid s_{t-1}, a_{t-1}) = p(\Delta s_t \mid a_{t-1}),$$
$$\forall t |\Delta s_t| \leq k,$$

where $\Delta s_t = s_t - s_{t-1}$ and k is the maximum state change. Accordingly, we define a new learnable function for the motion model, $g : \Delta s_t, a_{t-1} \mapsto p(\Delta s_t \mid a_{t-1})$ and use g instead of f. For vectorization, we define a $(2k+1)$-dimensional vector $\boldsymbol{g}(a)$, whose elements $g_i(a) = g(i-k-1, a)$ represent the probabilities of all positive and negative state changes up to k. We can now reformulate the prediction step (Eq. 4.1) as a convolution $(*)$,

$$\overline{\boldsymbol{b}}_t = \boldsymbol{b}_{t-1} * \boldsymbol{g}(a_{t-1}),$$

where the belief $\boldsymbol{b}_{t-1}$ is convolved with the motion kernel $\boldsymbol{g}(a_{t-1})$ for action a_{t-1} (see Fig. 4.4).

Motion Model: The learnable motion model g can be implemented as any feedforward network that maps Δs and a to a probability. The prior that $\boldsymbol{g}(a)$ represents a probability mass function, i.e. that the elements of $\boldsymbol{g}(a)$ should be positive and sum to one, can be enforced using the softmax nonlinearity on the vector of unnormalized network outputs $\tilde{\boldsymbol{g}}(a)$, such that $g_i(a) = \frac{e^{\tilde{g}_i(a)}}{\sum_j e^{\tilde{g}_j(a)}}$.

Another useful prior for g is smoothness with respect to Δs and a, i.e. that similar combinations of Δs and a lead to similar probabilities. This smoothness is the reason why (for standard feedforward networks), we should use Δs as an input rather than as index for different output dimensions. With additional knowledge about robot motion, we can replace smoothness by a stronger prior.

For the experiments in this chapter, we assumed linear motion with zero mean Gaussian noise, and therefore defined the motion model with only two learnable parameters α and

σ and the obligatory normalization,

$$\tilde{g}(\Delta s, a) = e^{-\frac{(\Delta s - \alpha a)^2}{\sigma^2}}$$
$$g(\Delta s, a) = \frac{\tilde{g}(\Delta s, a)}{\sum_{j=-k}^{k} \tilde{g}(j, a)}.$$

Measurement Update

Analogously to the motion model in the prediction step, we define a learnable function h that represents the measurement model for the measurement update, $h : s_t, o_t \mapsto p(o_t \mid s_t)$. To vectorize the update equation (Eq. 4.2), we define a vector $\boldsymbol{h}(o)$ with elements $h_i(o) = h(i, o)$, such that the measurement update corresponds to element-wise multiplication ($\odot$) with this vector,

$$\tilde{\boldsymbol{b}}_t = \boldsymbol{h}(o) \odot \overline{\boldsymbol{b}}_t,$$

followed by a normalization, $\boldsymbol{b}_t = \frac{\tilde{\boldsymbol{b}}_t}{\sum_j \tilde{b}_{t,j}}$ (see Fig. 4.4).

Measurement Model: The learnable function h that represents the measurement model can again be implemented by any feedforward network. Since h corresponds to $p(o_t \mid s_t)$—a probability distribution over observations—it needs to be normalized across observations, not across states. To realize the correct normalization, we need to compute the unnormalized likelihood vector $\tilde{\boldsymbol{h}}(o)$ for every observation o and compute the softmax over the corresponding scalars in different vectors rather than over the scalars of the same vector: $\boldsymbol{h}(o) = \frac{e^{\tilde{\boldsymbol{h}}(o)}}{\sum_{o'} e^{\tilde{\boldsymbol{h}}(o')}}$. If the observations are continuous instead of discrete, this summation must be approximated using sampled observations.

For the experiments in this chapter, we represented h by a network with three hidden layers of 32 rectified linear units (Nair and Hinton, 2010), followed by a linear function and a normalization as described above. Using the observation and state as input rather than output dimensions again incorporates the smoothness prior on these quantities.

Learning

We can learn the motion model g and the measurement model h using different learning objectives based on different sequences of data. We will first look at a number of supervised learning objectives that require $o_{1:T}$, $a_{1:T}$, $s_{1:T}$, and sometimes $x_{1:T}$—the underlying continuous state. Then, we will describe unsupervised learning that only needs $o_{1:T}$ and $a_{1:T}$.

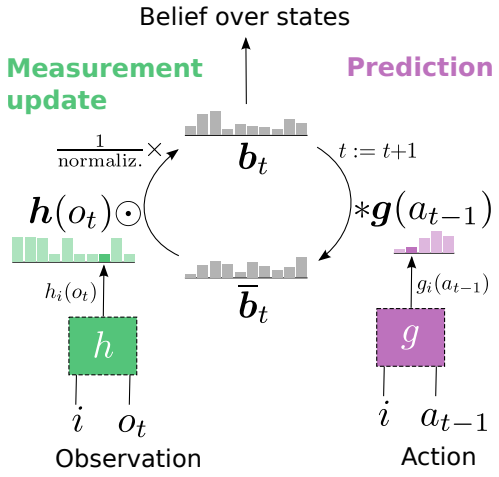


Figure 4.4: Differentiable histogram filter. Motion model (purple) and measurement model (green) are learned; the algorithmic structure is given ($*$: convolution, $\odot$: element-wise multiplication).

Supervised Learning in Isolation: Both models can be learned in isolation by optimizing an objective function, e.g. the cross-entropy between experienced state change / observation and the corresponding outputs of g and h,

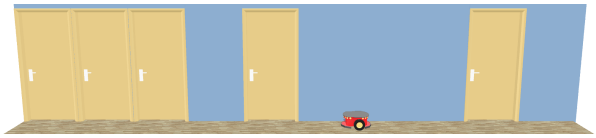
$$L_g = -\frac{1}{T-1} \sum_{t=2}^{T} \boldsymbol{e}^{(\Delta s_t - k - 1)} \log(\boldsymbol{g}(a_{t-1})),$$
$$L_h = -\frac{1}{T} \sum_{t=1}^{T} \boldsymbol{e}^{(o_t)} \log(\boldsymbol{h}(o_t)),$$

where $\boldsymbol{e}^{(i)}$ denotes a standard basis vector with all zeros except for a one at position i, that is the position that represents the experienced state change or observation.

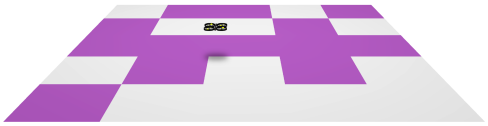
Supervised End-To-End Learning: Due to our differentiable implementation, the models can also be learned end-to-end using backpropagation through time (Werbos, 1990), which we apply on several overlapping subsequences of length C (in our experiments, $C = 32$). In the corresponding learning objectives, we compare the belief at the final time step of this subsequence with the true state. If we want to optimize the accuracy of the filter with respect to its discrete states, we can again use a cross-entropy loss,

$$L_{\text{acc.}} = -\frac{1}{T-C} \sum_{t=C+1}^{T} \boldsymbol{e}^{(s_t)} \log(\boldsymbol{b}_t^{(t-C:t)}),$$

where $\boldsymbol{b}_t^{(t-C:t)}$ denotes the final belief at time step t when the histogram filter is applied on the subsequence that spans steps $t-C$ to t. Alternatively, we might want to optimize other objectives, e.g. the mean square error with respect to the underlying continuous



(a) Hallway localization task



(b) Drone localization task

Figure 4.5: Randomly sampled environments per task. Motion and measurement models are unknown.

state,

$$L_{\text{mse}} = -\frac{1}{T-C}\sum_{t=C+1}^{T}(x_t - \mathbf{x}\boldsymbol{b}_t^{(t-C:t)})^2,$$

where $\mathbf{x}$ denotes a vector of the continuous values to which the discrete states correspond, such that $\mathbf{x}\boldsymbol{b}_t^{(t-C:t)}$ is the weighted average of these values according to the final belief in this subsequence.

Unsupervised End-To-End Learning: By exploiting the structure of the histogram filter algorithm and the differentiability, we can even train the models without any state labels by predicting future observations, but later use the models for state estimation. Similarly to supervised end-to-end learning, we apply the filter on different subsequences of length C, but then we follow this with D steps without performing the measurement update (in our experiments, $D = 32$). Instead, we use the measurement model to predict the observations. $\text{Pred}(o_t) = \sum_{s_t} p(o_t \mid s_t)\overline{\text{Bel}}(s_t) = \boldsymbol{h}(o_t)\overline{\boldsymbol{b}}_t$. To predict the probabilities for all observations, we define a matrix $\boldsymbol{H}$ with elements $H_{i,j} = h(i,j)$, using h as defined in Section 4.6.1. Putting everything together, we get the following loss for unsupervised end-to-end learning.

$$L_{\text{unsup.}} = -\frac{1}{(T-C)D}\sum_{t=C+1}^{T}\sum_{d=1}^{D}\boldsymbol{e}^{(o_{t+d})}\log(\boldsymbol{H}^\top\overline{\boldsymbol{b}}_{t+d}^{(t-C:t+d)}),$$

where the outer sum iterates over different subsequences and the inner sum iterates over predicted observations for a number of future steps.

4.5.2 Experiments and Results

To test DHFs, we consider the problem of learning to estimate the robot's state in unknown environments with partial observations. In this problem, we compare standard histogram filters for which the models are learned in isolation (HF), differentiable histogram filters (DHFs), and two-layer long-short-term memory networks (LSTMs, Hochreiter and Schmidhuber, 1997). The models of the HFs are learned by optimizing the loss functions

L_g and L_h presented in the previous section. For the DHFs and LSTMs, we compare end-to-end learning using $L_{\text{acc.}}$, L_{mse}, and $L_{\text{unsup.}}$.

Our results show that 1) the algorithmic priors in HFs and DHFs increase data efficiency for learning localization compared to generic LSTMs, 2) end-to-end learning improves the performance of DHFs compared to HFs, and 3) DHFs are able to learn state estimation without state labels.

Experiment: Learning State Estimation in Unknown Environments

Before analyzing the results, we will briefly describe the problem setting in more detail. We are focusing on partially observable localization tasks: a robot moves through an environment by performing actions and receives partial observations, such that it needs to filter this information over time to estimate its state, i.e. its position. In our experiments, the robot does not know the environment beforehand and thus has to learn from data how to recursively estimate its state.

We performed experiments in two continuous localization tasks: a) a hallway localization task (Thrun et al., 2005) and b) a drone localization task (see Fig. 4.5). The tasks are similar in that they have continuous actions and binary observations (door/wall and purple/white tile), both of which are subject to 10% random error. The tasks differ in the dimensionality of the continuous state. In the hallway task, the robot only needs to estimate a one-dimensional state (its position along the hallway), which for all methods is discretized into 100 states. The drone localization task has a two-dimensional state, which is discretized into 50 bins per dimension resulting in 2500 bins in total. In both tasks, the motion and the measurement models are unknown and localization needs to be learned from data, i.e. a sequence of observations, actions, and—in the supervised setting—states produced by the robot moving randomly through the environment.

Result: Improved Data-Efficiency

Hallway task: We performed multiple experiments in the hallway localization task with different amounts of training data. The learning curves with respect to mean squared error for supervised learning show large differences in data efficiency (see solid lines in Fig. 4.6a): DHFs require substantially less training samples than LSTMs to achieve good performance (2000 rather than $>$ 8000). HFs are even more data-efficient but quickly stop improving with additional data.

Drone task: For the drone localization task, we performed an experiment using 4000 training steps (see Table 4.1). Our results show that this data is sufficient for the DHF (but not for the LSTM) to achieve good performance. Our method only required a similar amount of data as for the 1D hallway task, even though the histogram size had increased from 100 to 2500 bins.

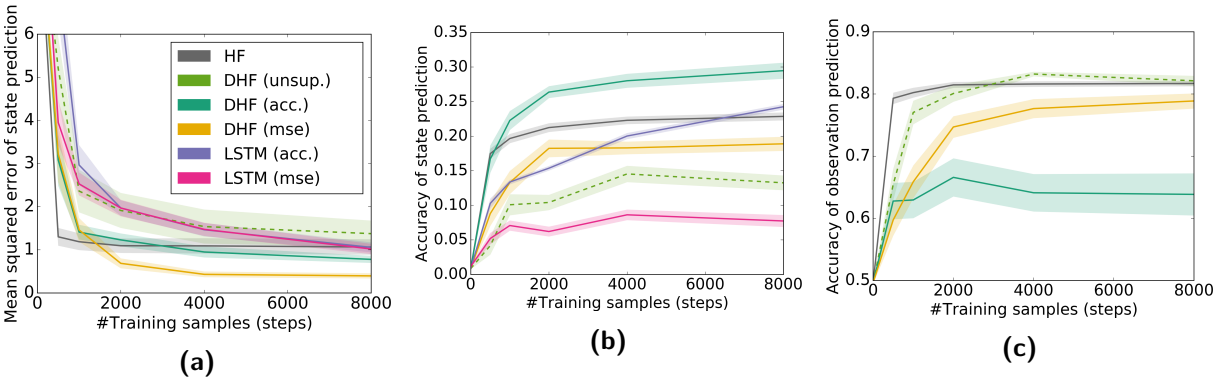


Figure 4.6: Hallway task, learning curves for different metrics: (a) mean squared error of estimating the continuous state—lower is better, (b) accuracy of estimation the discrete state—higher is better, (c) accuracy of predicting the next 32 observations—higher is better. The legend specifies both the architecture and the learning objective. Lines show means, shaded surfaces show standard errors. The dashed line highlights unsupervised learning (no state labels). LSTMs trained for state estimation cannot predict observations and therefore are not included in (c).

Table 4.1: Drone task: test performance of different methods with 4000 training samples

Method	MSE (state)	Acc. (state)	Acc. (obs.)
HF	0.22	0.05	**0.81**
DHF (unsup.)	0.22	0.03	**0.81**
DHF (acc.)	0.39	**0.17**	0.40
DHF (mse)	**0.16**	0.08	0.66
LSTM (acc.)	3.03	0.03	–
LSTM (mse)	0.50	0.06	–

Discussion: The priors encoded in the DHF improve data efficiency because any information contained in these priors does not need to be extracted from data. This leads to better generalization, e.g. the ability to robustly and accurately track multiple hypotheses (see Fig.4.7).

Note on computational limits: The size of the histogram is exponential in the number of state dimensions. A comparison between the 1D and the 2D task suggests that data might not be the bottleneck for applying the method to higher dimensional problems, since the data requirements were similar. However, the increased histogram size directly translates into longer training times, such that computation quickly becomes the bottleneck for scaling this method to higher-dimensional problems. Addressing this problem requires to change the belief representation, e.g. to particles, which we will do in the last part of this chapter.

Result: Optimization of End-to-End Performance

Hallway task: While HFs excel with very few data, DHFs surpass them if more than 2000 training samples are available (see gray and yellow lines in Fig. 4.6a). For the mean

squared error metric, the best method is the DHF with a mean squared error objective (yellow line). However, if we care about a different metric, e.g. accuracy of estimating the discrete state, the methods rank differently (see Fig. 4.6b). The best method for the previous metric (yellow line) is outperformed by HFs (gray line) and even more so by DHFs that are optimized for accuracy (teal line). For yet another metric, i.e. accuracy of predicting future observations, HFs outperform both other approaches but are equal to DHFs optimized for predicting future observations (see Fig. 4.6c).

Drone task: The results of the drone localization task show the same pattern (see Table 4.1). The best method for every metric is the DHF that optimizes this metric.

Discussion: DHFs perform better than HFs because they optimize the models for the filtering process (with respect to the metric they were trained for) rather than optimizing model accuracy. This can be advantageous because "inaccurate" models can improve end-to-end performance. Compare the measurement model learned in isolation (HF) to the models learned end-to-end (DHF) in Fig. 4.7a (the correct model would have probabilities 0.9 in door states and probabilities 0.1 in non-door states due to 10% measurement error).

Result: Enabling Unsupervised Learning

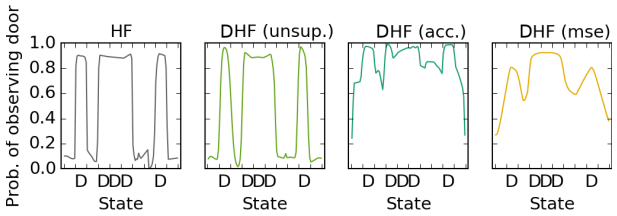
Hallway and drone tasks: In both tasks, unsupervised DHFs were similar to HFs and better than all other methods for predicting future observations. Interestingly, they also had comparatively low mean squared error for state estimation even though they had never seen any state labels (see dashed green line in Fig. 4.6 and second line in Table 4.1). In fact, the qualitative results for both tasks show a remarkable similarity between the learned models and the estimated belief between HFs and unsupervised DHFs (compare HF and DHF (unsup.) in Fig. 4.7 and Fig. 4.8).

Discussion: DHFs can learn state estimation purely based on observations and actions. By predicting future observations using the structure of the histogram filter algorithm, the method discovers a state representation that works well with this algorithm, which is surprisingly close to the "correct" models learned by HFs, although no state labels are used.

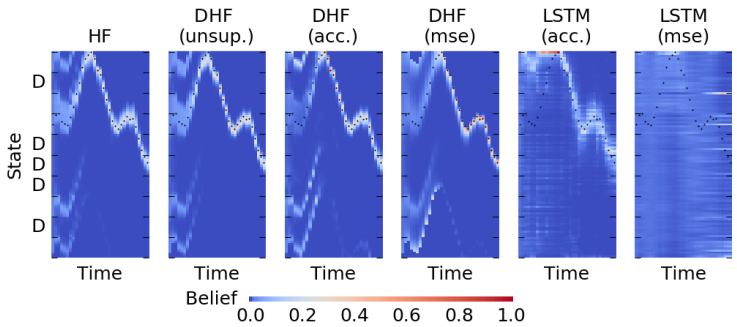
4.5.3 Limitations

Differentiable histogram filters present a proof-of-concept but they are not practical for most robotic tasks.

The main limitations of DHFs are: 1) computation scales exponentially with the number of dimensions due to the state space discretization; 2) formulating the prediction step using convolution requires state-invariant motion; 3) the normalization in the measurement model relies on discrete observations.



(a) Learned measurement models



(b) Belief over time during a test run. Black dots mark the true trajectory.

Figure 4.7: Hallway navigation task: (a) learned measurement models for one environment (D=door state) and (b) belief evolution for a single test run in this environment. All methods used 4000 training samples.

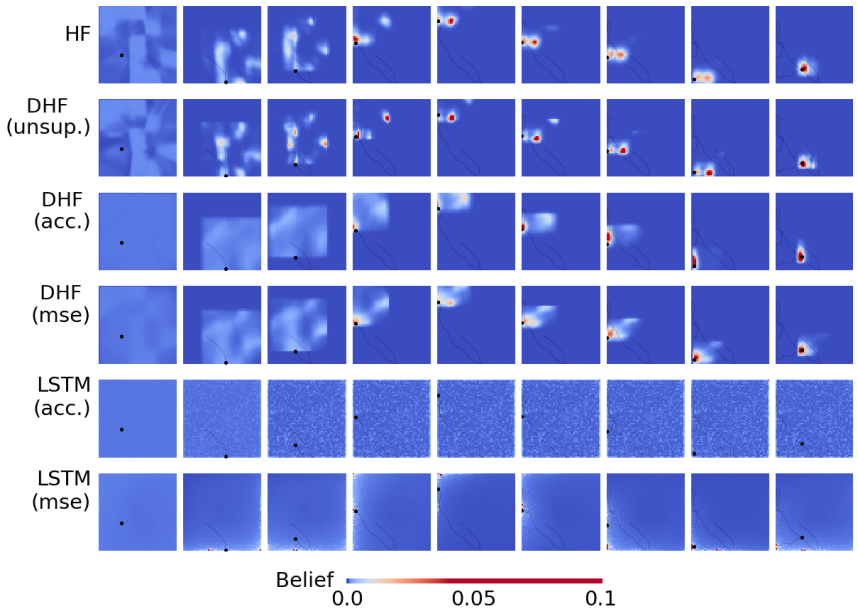


Figure 4.8: Drone localization task: belief evolution during a single test run for different methods. Black dots/lines show the true position/trajectory of the drone. All methods used 4000 training samples.

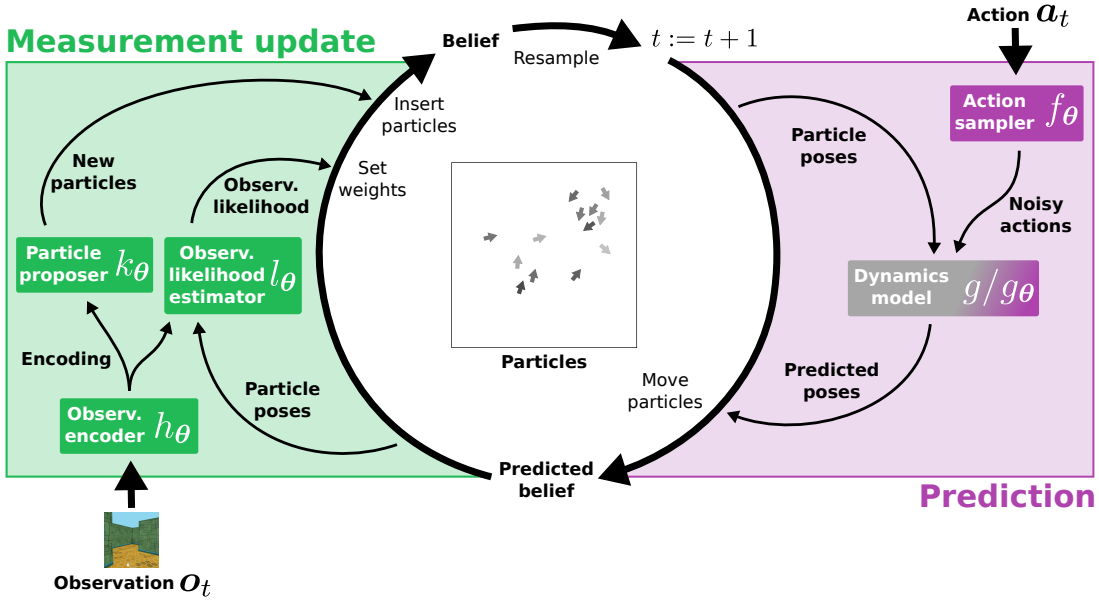


Figure 4.9: DPF overview. Models in prediction and measurement update can be learned end-to-end. Boxes represent models and color indicates that they can be learned.

In the next section, we will demonstrate how all of these limitations can be overcome, by extending the histogram filter to a particle filter: 1) the state space does not need to be discretized but the particles focus computation at states of high belief; 2) the prediction step with particles allows state dependent motion models; 3) omitting the normalization in the measurement model still results in successful state estimation, which removes the requirement for discrete observations.

4.6 Differentiable Particle Filters

This section introduces the *differentiable particle filter* (DPF), an end-to-end differentiable implementation of the particle filter with learnable motion and measurement models. We can also view DPFs as a new recurrent network architecture that encodes the algorithmic prior from particle filters in the network structure (see Fig. 4.9).

To extend the histogram filter to a particle filter, we need to replace the belief histogram with a set of particles. We represent each particle as a vector that describes the particle location in state space and its weight. The belief is then represented by a vector of particles. The measurement update computes the particle weights similarly to the measurement update in the histogram filter. The prediction step does not perform convolution, but moves particles (updates their location in state space) by sampling from a probabilistic motion model. After each time step, particles are resampled according to their weight. The particle set is initialized by proposing particles based on the current observation.

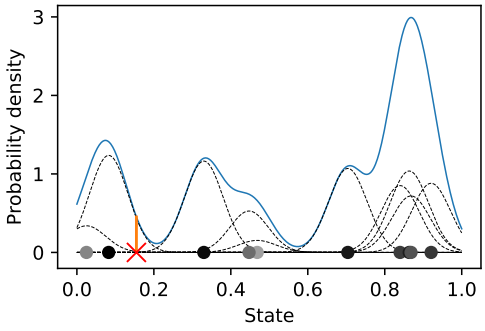


Figure 4.10: Computing the gradient for end-to-end learning requires density estimation from the predicted particles (gray circles, darkness corresponds to particle weight). After converting the particles into a mixture of Gaussians (blue), we can compute the belief at the true state (orange bar at red x) and maximize it.

4.6.1 Implementation

This section describes our DPF implementation. Our source code, which is based on TensorFlow (Abadi et al., 2015) and Sonnet (DeepMind, 2017), is available at:
`https://github.com/tu-rbo/differentiable-particle-filters`

Belief

DPFs represent the belief at time t by a set of weighted particles, $\text{bel}(\boldsymbol{s}_t) = (S_t, \boldsymbol{w}_t)$, where $S \in \mathbb{R}^{n \times d}$ describes n particles in d-dimensional state space with weights $\boldsymbol{w} \in \mathbb{R}^n$. At every time step, DPFs update the previous belief $\text{bel}(\boldsymbol{s}_{t-1})$ with action $\boldsymbol{a}_t$ and observation $\boldsymbol{o}_t$ to get $\text{bel}(\boldsymbol{s}_t)$ (see Fig. 4.9).

Prediction

The prediction step moves each particle by sampling from a probabilistic motion model (Eq. 4.3). We implement this step by splitting the motion model into an *action sampler* f, which creates a noisy action $\hat{\boldsymbol{a}}^{[i]}$ per particle, and a *dynamics model* g, which moves each particle i according to $\hat{\boldsymbol{a}}^{[i]}$.

$$\hat{\boldsymbol{a}}_t^{[i]} = \boldsymbol{a}_t + f_{\boldsymbol{\theta}}(\boldsymbol{a}_t, \boldsymbol{\epsilon}^{[i]} \sim \mathcal{N}), \tag{4.5}$$

$$\boldsymbol{s}_t^{[i]} = \boldsymbol{s}_{t-1}^{[i]} + g(\boldsymbol{s}_{t-1}^{[i]}, \hat{\boldsymbol{a}}_t^{[i]}), \tag{4.6}$$

where $f_{\boldsymbol{\theta}}$ is a feedforward network, $\boldsymbol{\theta}$ are all parameters of the DPF, and $\boldsymbol{\epsilon}^{[i]} \in \mathbb{R}^d$ is a noise vector drawn from a standard normal distribution. Using a noise vector as input to learn a generative model via backpropagation is known as the reparameterization trick (Kingma and Welling, 2013). Here, this trick enables $f_{\boldsymbol{\theta}}$ to learn to sample from action-dependent motion noise. The resulting noisy actions are fed into g, which simulates how these actions change the state. If we know the underlying dynamics model we can directly implement

its equations in g, which we do for our experiments. Alternatively, we could replace g by a feedforward network $g_{\boldsymbol{\theta}}$ and learn the dynamics from data (see Table 4.2).

Measurement Update

The measurement update uses the observation to compute particle weights (Eq. 4.4). DPFs implement this update and additionally use the observation to propose new particles (see Fig. 4.9). The DPF measurement model consists of three components: a shared *observation encoder* h, which encodes an observation $\boldsymbol{o}_t$ into a vector $\boldsymbol{e}_t$, a *particle proposer* k, which generates new particles, and an *observation likelihood estimator* l, which weights each particle based on the observation.

$$\boldsymbol{e}_t = h_{\boldsymbol{\theta}}(\boldsymbol{o}_t), \tag{4.7}$$

$$\boldsymbol{s}_t^{[i]} = k_{\boldsymbol{\theta}}(\boldsymbol{e}_t, \boldsymbol{\delta}^{[i]} \sim B), \tag{4.8}$$

$$w_t^{[i]} = l_{\boldsymbol{\theta}}(\boldsymbol{e}_t, \boldsymbol{s}_t^{[i]}), \tag{4.9}$$

where $h_{\boldsymbol{\theta}}$, $k_{\boldsymbol{\theta}}$, and $l_{\boldsymbol{\theta}}$ are feedforward networks based on parameters $\boldsymbol{\theta}$; The input $\boldsymbol{\delta}^{[i]}$ is a dropout vector sampled from a Bernoulli distribution. Here, dropout is not used for regularization but as a source of randomness for sampling different particles from the same encoding $\boldsymbol{e}_t$ (see Table 4.2).

Particle Proposal and Resampling

DPFs combine two processes that generate particles: the first one proposes particles from the current observation (as described above) and the second one resamples existing particles based on their weight. During filtering, DPFs move gradually from the first process, which generates possible hypotheses, to the second, which tracks and weeds out these hypotheses. The ratio of proposed particles follows an exponential function, in our case 0.7^t with t=0 at the beginning of the filtering process.

DPFs implement resampling by stochastic universal sampling (Baker, 1987). Since resampling is not differentiable, it stops the gradient such that end-to-end learning can only optimize the models to improve the current output, neglecting how these changes might affect the output in future time steps. This limitation is not problematic for supervised learning because predicting the correct Markov state is the right objective at every time step. Unsupervised learning on the other hand will probably require backpropagation through multiple time steps—which is an important topic in future work.

Table 4.2: Feedforward networks for learnable DPF models

$f_{\boldsymbol{\theta}}$:	2 x fc(32, relu), fc(3) + mean centering across particles
$g_{\boldsymbol{\theta}}$:	3 x fc(128, relu), fc(3) + scaled by $\mathrm{E}_t[\mathrm{abs}(\boldsymbol{s}_t - \boldsymbol{s}_{t-1})]$
$h_{\boldsymbol{\theta}}$:	conv(3x3, 16, stride 2, relu), conv(3x3, 32, stride 2, relu), conv(3x3, 64, stride 2, relu), dropout(keep 0.3), fc(128, relu)
$k_{\boldsymbol{\theta}}$:	fc(128, relu), dropout*(keep 0.15), 3 x fc(128, relu), fc(4, tanh)
$l_{\boldsymbol{\theta}}$:	2 x fc(128, relu), fc(1, sigmoid scaled to range [0.004, 1.0])

fc: fully connected, conv: convolution, *: applied at training and test time

Supervised Learning

DPF models can be learned from sequences of supervised data $\boldsymbol{o}_{1:T}$, $\boldsymbol{a}_{1:T}$, $\boldsymbol{s}^*_{1:T}$ using maximum likelihood estimation, where we want to maximize the belief at the true state $\mathrm{bel}(\boldsymbol{s}^*_t)$. To estimate the probability density at $\boldsymbol{s}^*_t$ from a set of particles, we treat each particle as a Gaussian in a mixture model with weights $\boldsymbol{w}_t$ (see Fig. 4.10). For a sensible metric across state dimensions, we scale each dimension by dividing by the average step size $\mathrm{E}_t[\mathrm{abs}(\boldsymbol{s}^*_t - \boldsymbol{s}^*_{t-1})]$. This density estimation enables individual and end-to-end learning.

Individual Learning of the Motion Model: We optimize the motion model individually to match the observed motion noise by sampling states $\boldsymbol{s}^{[i]}_t$ from $\boldsymbol{s}^*_{t-1}$ and $\boldsymbol{a}_t$ using Eq. 4.5-4.6 and maximizing the data likelihood as described above, $\boldsymbol{\theta}^*_f = \mathrm{argmin}_{\boldsymbol{\theta}_f} - \log p(\boldsymbol{s}^*_t \mid \boldsymbol{s}^*_{t-1}, \boldsymbol{a}_t; \boldsymbol{\theta}_f)$. If the dynamics model g is unknown, we train $g_{\boldsymbol{\theta}}$ by minimizing mean squared error between $g(\boldsymbol{s}^*_{t-1}, \boldsymbol{a}_t)$ and $\boldsymbol{s}^*_t - \boldsymbol{s}^*_{t-1}$.

Individual Learning of the Measurement Model: The particle proposer $k_{\boldsymbol{\theta}}$ is trained analogously to the motion model by sampling $\boldsymbol{s}^{[i]}_t$ from $\boldsymbol{o}_t$ using Eq. 4.7-4.8 and maximizing the Gaussian mixture at the true state $\boldsymbol{s}^*_t$.

We train the observation likelihood estimator $l_{\boldsymbol{\theta}}$ (and $h_{\boldsymbol{\theta}}$) by maximizing the likelihood of observations in their state and minimizing their likelihood in other states, $\boldsymbol{\theta}^*_{h,l} = \mathrm{argmin}_{\boldsymbol{\theta}_{h,l}} - \log(\mathrm{E}_t[l_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\boldsymbol{o}_t), \boldsymbol{s}^*_t)]) - \log(1 - \mathrm{E}_{t_1,t_2}[l_{\boldsymbol{\theta}}(h_{\boldsymbol{\theta}}(\boldsymbol{o}_{t_1}), \boldsymbol{s}^*_{t_2})])$.

End-To-End Learning: For end-to-end learning, we apply DPFs on overlapping subsequences and maximize the belief at all true states along the sequence as described above,

$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}} - \log \mathrm{E}_t[\mathrm{bel}(\boldsymbol{s}^*_t; \boldsymbol{\theta})].$$

4.6.2 Experiments and Results

We evaluated DPFs in two state estimation problems in robotics: *global localization* and *visual odometry*. We tested global localization in simulated 3D mazes based on vision

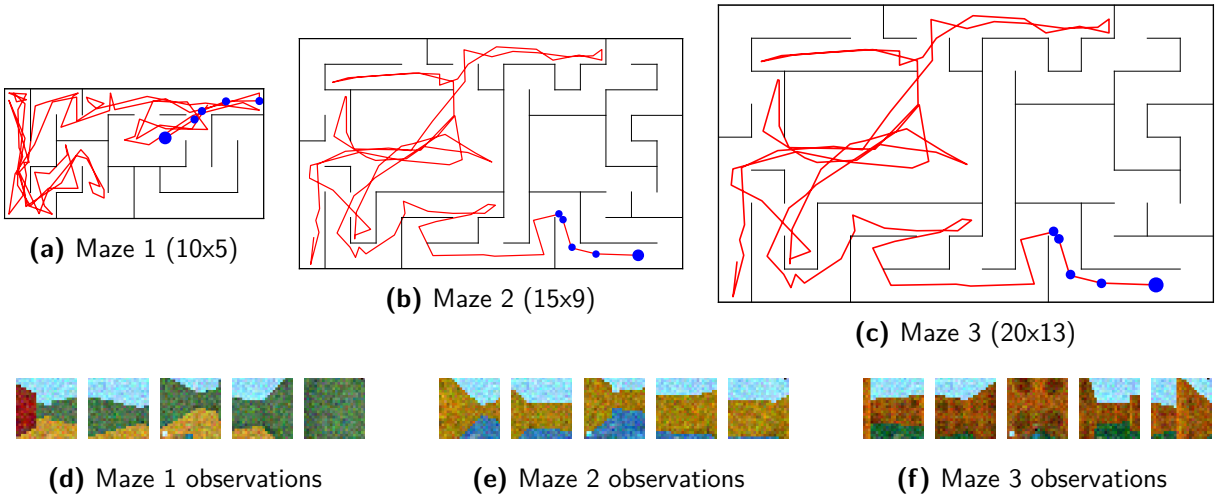
and odometry. We focused on this task because it requires simultaneously considering multiple hypotheses, which is the main advantage of particle filters over Kalman filters. Here, we evaluated two things: a) the effect of end-to-end learning compared to individual learning and b) the influence of algorithmic priors encoded in DPFs by comparing to generic LSTMs. To show the versatility of DPFs and to compare to published results with backprop Kalman filters (BKFs, Haarnoja et al., 2016), we also apply DPFs to the KITTI visual odometry task (Geiger et al., 2013). The goal is to track the position and orientation of a driving car based on a first-person-view video. In both tasks, DPFs use the known dynamics model g but do not assume any knowledge about the map of the environment and learn the measurement model entirely from data.

Our global localization results show that 1) algorithmic priors enable explainability, 2) end-to-end learning improves performance but sequencing individual and end-to-end learning is even more powerful, 3) algorithmic priors in DPFs improve performance compared to LSTMs reducing the error by 70-80%, and 4) algorithmic priors lead to policy invariance: While the LSTM baseline learns localization in a way that stops working when the robot behaves differently ($\sim$84% error rate), localization with the DPF remains useful with different policies ($\sim$15% error rate).

In the visual odometry task, DPFs perform as well as BKFs even though the task exactly fits the capabilities and limitations of Kalman filters—tracking a unimodal belief from a known initial state. This result demonstrates the versatility of DPFs because the task differs from the first one in a number of properties: higher frequency, longer sequences and a 5D state instead of a 3D state. The result also shows that DPFs work on real data and are able to learn measurement models that work for visually diverse observations based on less than 40 minutes of video.

Experiment: Learning Global Localization

In this global localization task, we want to estimate the position and orientation of a robot based on visual input. For this purpose, we took the navigation environments from DeepMind Lab (Beattie et al., 2016) and removed all objects and unique wall textures to ensure partial observability. We then collected data by letting the simulated robot wander through the maze (see Fig. 4.11). The robot followed a hand-coded policy that moves in directions with high depth values from RGB-D input and performs 10% random actions. For each maze, we collected 1000 trajectories of 100 steps with one step per second for training and testing. As input for localization, we only used RGB images and odometry and added random disturbances to make the task more realistic. For the observations, we randomly cropped the rendered 32×32 RGB images to 24×24 and added Gaussian noise with $\sigma = 20$ (see Fig. 4.11d-f). As actions, we used odometry information that corresponds to the change in position and orientation since the last time step in the robot's local frame.



(a) Maze 1 (10x5)

(b) Maze 2 (15x9)

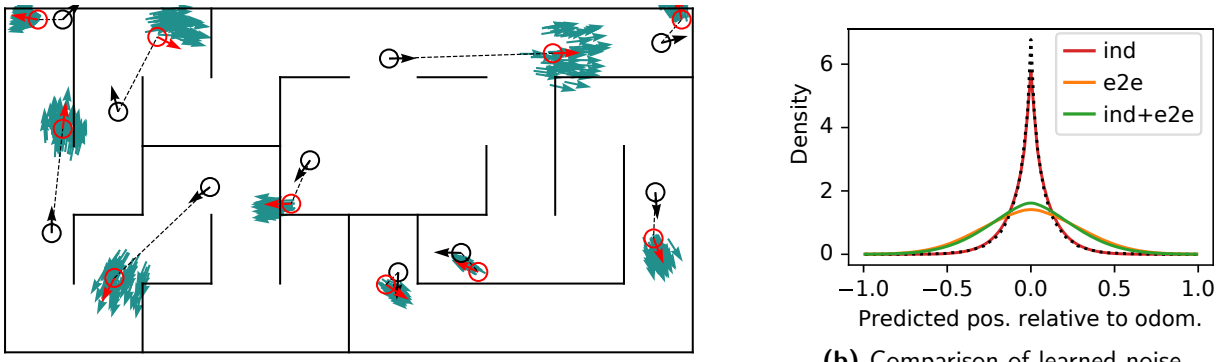
(c) Maze 3 (20x13)

(d) Maze 1 observations

(e) Maze 2 observations

(f) Maze 3 observations

Figure 4.11: Three maze environments. Red lines show example trajectories of length 100. Blue circles show the first five steps, of which the observations are depicted below.



(a) Predictions with learned motion model

(b) Comparison of learned noise

Figure 4.12: Learned motion model. (a) shows predictions (cyan) of the state (red) from the previous state (black). (b) compares prediction uncertainty in x to true odometry noise (dotted line).

We corrupted the actions with multiplicative Gaussian noise with $\sigma = 0.1$. All methods were optimized on short trajectories of length 20 with Adam (Kingma and Ba, 2014) and regularized using dropout (Srivastava et al., 2014) and early stopping. We will now look at the results in this task.

Result: Algorithmic Priors Enable Explainability

Due to the algorithmic priors in DPFs, the models remain interpretable even after end-to-end learning. We can therefore examine a) the motion model, b) the measurement model, and c) their interplay during filtering. Unless indicated otherwise, all models were first learned individually and then end-to-end.

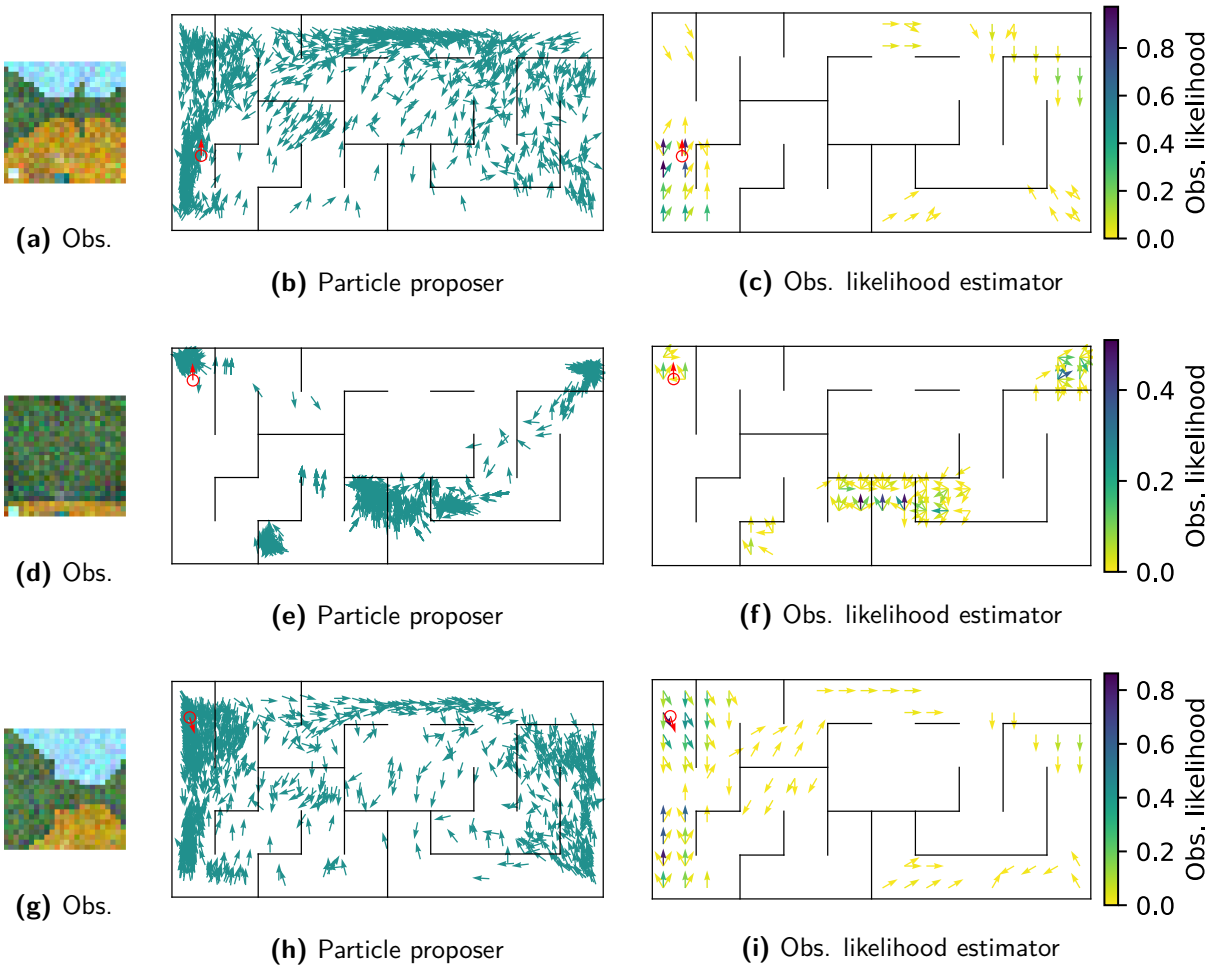


Figure 4.13: Learned measurement model. Observations, corresponding model output, and true state (red). To remove clutter, the observation likelihood only shows above average states.

Motion Model: Fig. 4.12a shows subsequent robot poses together with predictions from the motion model. These examples show that the model has learned to spread the particles proportionally to the amount of movement, assigning higher uncertainty to larger steps. But how does this behavior depend on whether the model was learned individually or end-to-end?

Fig. 4.12b compares the average prediction uncertainty using models from different learning schemes. The results show that individual learning leads to the "correct" model that exactly describes the odometry noise (compare the red and the dotted black line). End-to-end learning, on the other hand, generates models that overestimate the noise (green and orange lines), which matches the insights of experts in state estimation who report that "many of the models that have proven most successful in practical applications vastly overestimate the amount of uncertainty" (Thrun et al., 2005, p. 118).

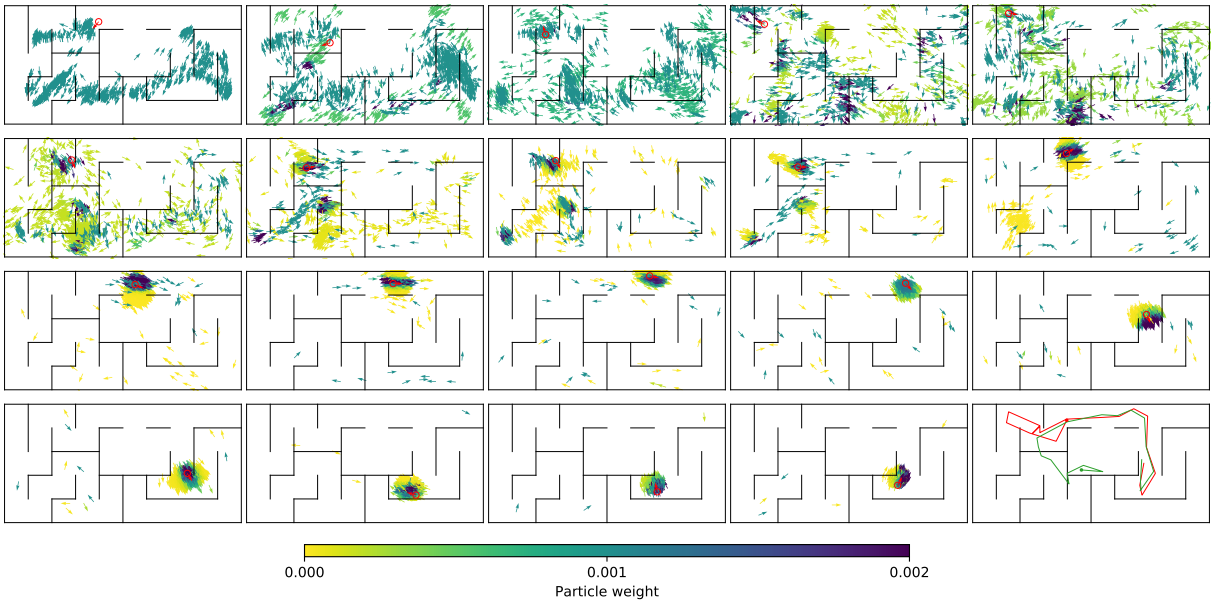
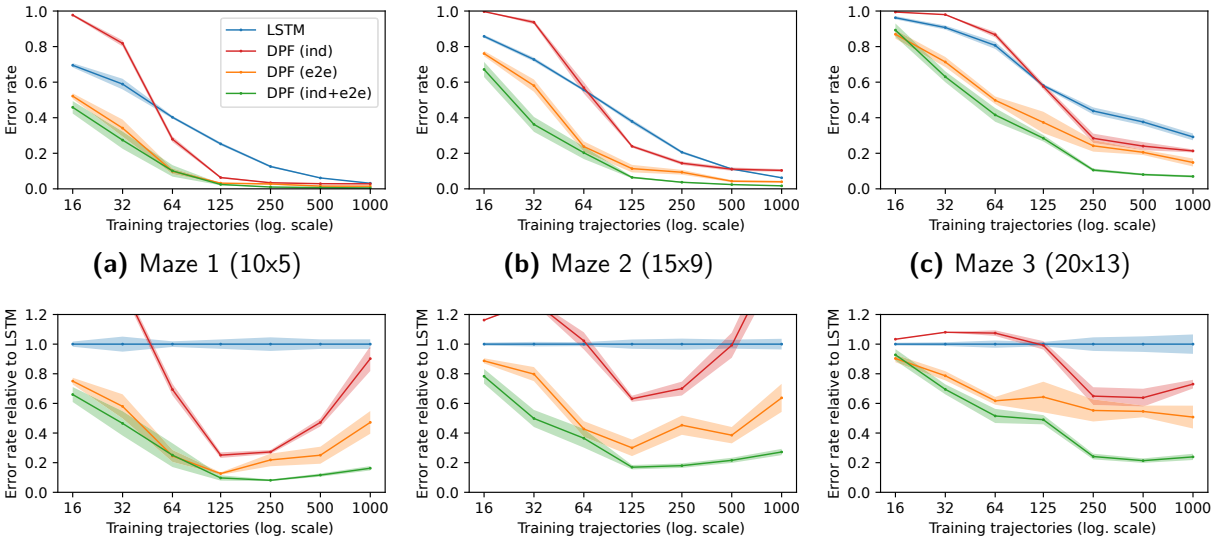


Figure 4.14: Global localization with DPFs. One plot per time step of a test trajectory: true state (red), 1000 particles (proposed particles have weight 0.001). Last plot: the weighted particle mean (green) matches the true state after the first few steps.

Measurement Model Fig. 4.13 shows three example observations and the corresponding outputs of the measurement model: proposed particles and weights depending on particle position. Note how the model predicts particles and estimates high weights at the true state and other states in locally symmetric parts of the maze. We can also see that the data distribution shapes the learned models, e.g. by focusing on dead ends for the second observation, which is where the robot following the hand-coded policy will look straight at a wall before turning around. Similar to motion models, end-to-end learned measurement models are not accurate but effective for end-to-end state estimation, as we will see next.

Model Interplay: Figure 4.14 shows how the models work together in the DPF. During the first five steps, the filter mainly proposes new particles and only slowly shifts to resampling according of the particle weights. After this initialization, the filter reduces the number of main hypotheses to three in step seven and to one in step ten. In the remaining steps, the filter tracks the state by moving and spreading the particles according to the motion model and by refocusing the belief based on observation likelihood.



(a) Maze 1 (10x5) **(b)** Maze 2 (15x9) **(c)** Maze 3 (20x13)

(d) Maze 1 (10x5), relative to LSTM **(e)** Maze 2 (15x9), relative to LSTM **(f)** Maze 3 (20x13), relative to LSTM

Figure 4.15: Learning curves in all mazes (a-c), also relative to LSTM baseline (d-f). ind: individual learning, e2e: end-to-end learning. Shaded areas denote standard errors.

Result: End-to-End Learning Improves Performance

To quantify the effect of end-to-end learning on state estimation performance, we compared three different learning schemes for DPFs: individual learning of each model (**ind**), end-to-end learning (**e2e**), and both in sequence (**ind+e2e**). We evaluated the performance in all three mazes and varied the amount of training trajectories along a logarithmic scale from 32 to 1000. We measured localization performance by *error rate*, where we consider a prediction erroneous if the distance to the true state, divided by $\mathrm{E}_t[\mathrm{abs}(\boldsymbol{s}_t - \boldsymbol{s}_{t-1})]$, is greater than 1.

The resulting learning curves in Fig. 4.15a-c show that end-to-end learned DPFs (orange line) consistently outperform individually trained DPFs (red line) across all mazes. Individual training is worst with few training trajectories (less than 64) but also plateaus with more data (more than 125 trajectories). In both cases, the problem is that the models are not optimized for state estimation performance. With few data, the training does not take into account how the unavoidable model errors will affect filtering performance. With lots of data, the models will work according to the individual learning objectives but those might not be aligned optimally with the end-to-end filtering objective. End-to-end learning consistently leads to improved performance for the same reasons.

Performance improves even more when we sequence individual and end-to-end learning (green line in Fig. 4.15a-c). Individual pretraining helps because it incorporates additional information about the function of each model into the learning process, while end-to-end learning incorporates information about how these models affect end-to-end performance.

Naturally, it is beneficial to combine both sources of information.

Result: Algorithmic Priors Improve Performance

To measure the effect of the algorithmic priors encoded in DPFs, we compare them with a generic neural network baseline that replaces the filtering loop with a two-layer long-short-term memory network (LSTMs, Hochreiter and Schmidhuber, 1997). The baseline architecture is kept as similar as possible to the DPF architecture—the same structure as $h_{\boldsymbol{\theta}}$ followed by 2 x lstm(512), 2 x fc(256, relu), and fc(3)—and is trained end-to-end to minimize mean squared error.

The comparison between DPF (ind+e2e) and the LSTM baseline (blue) in Fig. 4.15a-c shows that the error rate of DPF (ind+e2e) is lower than for LSTM for all mazes and all amounts of training data. Also in all mazes, DPF (ind+e2e) achieve the final performance of LSTM already with 125 trajectories, $\frac{1}{8}$ of the full training set.

We performed a small ablation study in maze 2 to quantify the effect the known dynamics model on this performance. When the dynamics model is learned, the final error rate for DPFs increases from 1.6% to 2.7% compared to 6.0% error rate for LSTMs. This shows that knowing the dynamics model is helpful but not essential for DPF's performance.

To visualize the performance relative to the LSTM baseline in more detail, we divided all learning curves by performance of the baseline (see Fig. 4.15d-f). Since DPFs include additional prior knowledge compared to LSTMs, we might expect them to have higher bias and lower variance. Therefore, we would expect DPFs relative error to be lowest with small amounts of data and highest with large amounts of data (the green curves in Fig. 4.15d-f should go up steadily from left to right until they cross the blue lines). Surprisingly, these curves show a different trend: DPFs relative performance to LSTMs improves with more data and converges to about $\frac{1}{10}$ to $\frac{1}{3}$ of the error rate. There could be a slight upwards trend in the end, but on a logarithmic data axis it would take a tremendous amount of data to close the gap. This result suggests that the priors from the Bayes filter algorithm reduce variance without adding bias—*that these algorithmic priors capture some true structure about the problem*, which data does not help to improve upon.

Result: Algorithmic Priors Lead to Policy Invariance

To be useful for different tasks, localization must be policy-invariant. At the same time, the robot must follow some policy to gather training data, which will inevitably affect the data distribution, add unwanted correlations between states and actions, etc.

We investigated how much the different methods overfit to these correlations by changing the policy between training and test, using two policies A and B. Policy A refers to the heuristic exploration policy that we used for all experiments above (see Section 4.6.2). Policy B uses the true pose of the robot, randomly generates a goal cell in the maze,

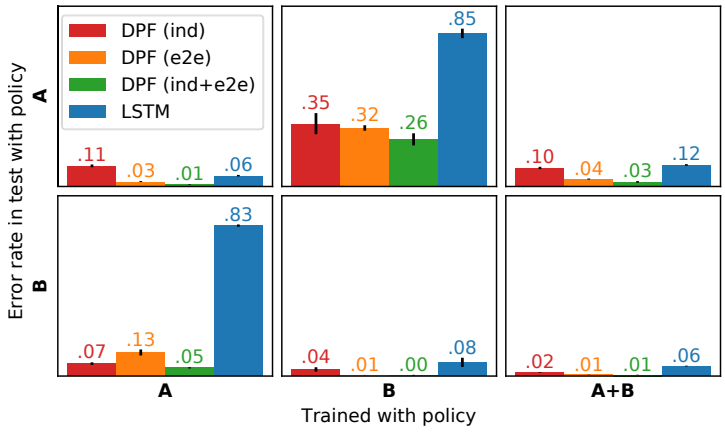


Figure 4.16: Generalization between policies in maze 2. A: heuristic exploration policy, B: shortest path policy. Methods were trained using 1000 trajectories from A, B, or an equal mix of A and B, and then tested with policy A or B.

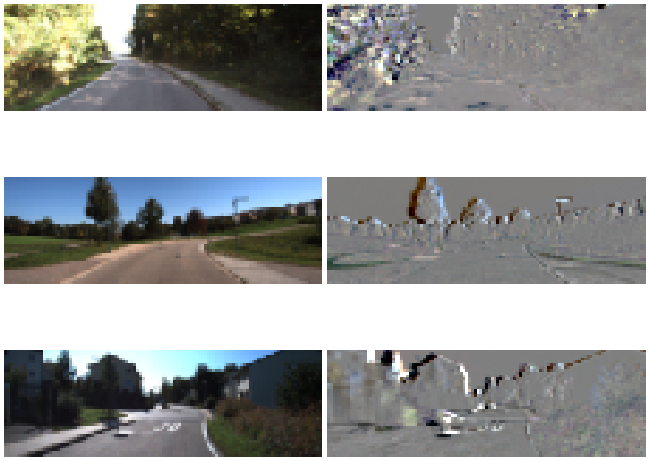
computes the shortest path to the goal, and follows this path from cell to cell using a simple controller mixed with 10% random actions.

The results in Fig. 4.16 show that all methods have low error rates when tested on their training policy (although DPFs improve over LSTMs even more on policy B). But when we use different policies for training and test, LSTM's error rate jumps to over 80%, while DPF (ind+e2e) still works in most cases (5% and 26% error rate).

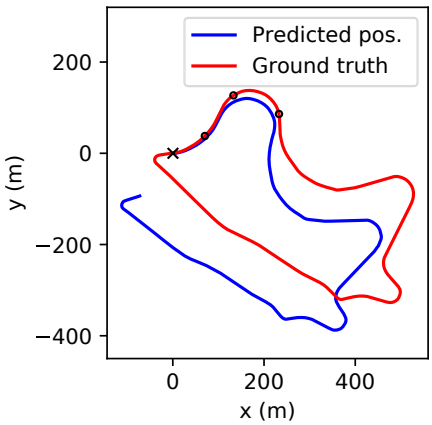
The LSTM baseline is not able to generalize to new policies because it does not discriminate between actions and observations and fits to any information that improves state estimation. If the training data includes correlations between states and actions (e.g. because the robot moves faster in a long hallway than in a small room), then the LSTM learns this correlation. Put differently, the LSTM learns to infer the state from the action chosen by the policy. The problem is that this inference fails if the policy changes. The algorithmic priors in DPFs prevent them from overfitting to such correlations between states and actions because DPFs cannot directly infer states from actions.

DPFs generalize better from A to B than from B to A. Since generalization from B to A is equally difficult for DPFs with individually learned models, the error increase cannot come from overfitting to correlations in the data through end-to-end learning, but is most likely because the states visited by policy A cover those visited by policy B but not vice versa.

The alternative approach to encoding policy invariance as a prior is to learn it by adding this variance to the data. Our results show that if we train on combined training data from both policies (A+B), all methods perform well in tests with either policy. This approach in the spirit of domain randomization and data augmentation helps DPFs because it covers the union of the visited states and (additionally) helps the LSTMs by including state-action correlations from both policies. But to make the LSTM localization truly policy invariant such that it would work with any new policy C, the training data has



(a) Visual input (image and difference image) at time steps 100, 200, and 300 (indicated in (b) by black circles)



(b) Trajectory 9; starts at (0,0)

Figure 4.17: Visual odometry with DPFs. Example test trajectory

to cover the space of all policies in an unbiased way, which is difficult for any interesting problem.

Experiment: Learning Visual Odometry

To validate our simulation results on real data, we applied DPFs on the KITTI visual odometry data set, which consists of data from eleven trajectories of a real car driving in an urban area for a total of 40 minutes. The data set includes RGB stereo camera images as well as the ground truth position and orientation of the car in an interval of ∼0.1 seconds. The main challenge of this task is to generalize in a way that works across highly diverse observations because the method is being tested on a road that it has never seen during training. Since the the car drives on different roads in every trajectory, it is not possible to extract global information about the car's position from the images. Instead, we need to estimate the car's translational and angular velocity from the stream of images and integrate this information over time to track the car's position and orientation.

We tackle this problem with a DPF in a five dimensional state space, which consists of the position, orientation, forward velocity and angular velocity. DPFs learn to perform visual odometry from a known initial state using a simple first-order *dynamics model* g and a learnable action sampler $f_{\boldsymbol{\theta}}$. Since there is no information about the action of the driver, the action sampler produces zero mean motion noise on the velocity dimensions, which is then evaluated with the measurement model. To keep the comparison as fair as possible, we used the same network architecture for the observation encoder $h_{\boldsymbol{\theta}}$ that was used in the backprop Kalman filter paper (Haarnoja et al., 2016), which takes as input the current image and the difference image to the last frame (see Fig. 4.17). Our observation likelihood estimator $l_{\boldsymbol{\theta}}$ weights particles based on their velocity dimensions and the encoding $h_{\boldsymbol{\theta}}(\boldsymbol{o}_t)$.

Table 4.3: KITTI visual odometry results

	Test 100	Test 100/200/400/800
Translational error (m/m)		
BKF*	0.2062	0.1804
DPF (ind)	0.1901 ± 0.0229	0.2246 ± 0.0371
DPF (e2e)	**0.1467 ± 0.0149**	0.1748 ± 0.0468
DPF (ind+e2e)	0.1559 ± 0.0280	**0.1666 ± 0.0379**
Rotational error (deg/m)		
BKF*	0.0801	0.0556
DPF (ind)	0.1074 ± 0.0199	0.0806 ± 0.0153
DPF (e2e)	0.0645 ± 0.0086	0.0524 ± 0.0068
DPF (ind+e2e)	**0.0499 ± 0.0082**	**0.0409 ± 0.0060**

Means ± standard errors; * results from Haarnoja et al. (2016)

Since, the initial state is known, we do not use a particle proposer. We train the DPF individually and end-to-end, only using the velocity dimensions for maximum likelihood estimation.

We evaluated the performance following the same procedure as in the BKF paper. We used eleven-fold cross validation where we picked one trajectory for testing and used the rest for training, for which we used subsequences of length 50. We evaluated the trained model on the test trajectory by computing the average error over all subsequences of 100 time steps and all subsequences of 100, 200, 400, and 800 time steps.

Result: Sample-Based Representations Outperform Gaussians

Table 4.3 compares our results to those published for BKFs (Haarnoja et al., 2016). We can see that DPFs outperform BKFs, in particular for short sequences where they reduce the error by ∼30%. Any improvement over BKFs in this task is surprising because Gaussian beliefs seem sufficient to capture uncertainty in this task. The improvement could come from the fact that DPFs sample-based distributions can represent long tailed probabilities for the motion noise and the belief. These results demonstrate that DPFs generalize to different tasks and can be successfully applied to real data.

4.7 Conclusion

4.7.1 Summary

We proposed to combine prior knowledge captured in algorithms with end-to-end learning by making the algorithm differentiable and such that their parameters can be learned in a way that optimizes the algorithm's performance. We demonstrated the feasibility of this idea in the context of state estimation in robotics by introducing differentiable histogram filters and differentiable particle filters.

Our experiments show substantial advantages of combining end-to-end learning with algorithmic priors. End-to-end learning optimizes models for performance while algorithmic priors enable explainability and regularize learning, which improves data-efficiency and generalization. Our results also demonstrate that this combination can enable unsupervised learning.

I believe that using algorithms as algorithmic priors will improve generalization of end-to-end learned systems and make them more applicable for robotics. The components of our DHF and DPF implementations, such as prediction by convolution, sample generation, and density estimation, will be useful for producing differentiable versions of other grid- and sampling-based algorithms.

4.7.2 Alternatives to End-to-End Learning

In this chapter, we used prior knowledge to restrict the hypothesis space and combined it end-to-end learning objectives. End-to-end learning is powerful because it optimizes the performance of the entire system rather than the performance of individual components. But the success of this approach hinges on the availability of labeled data that we can use for end-to-end learning.

In many cases, labels are not available because it is difficult to measure the true state of a system, because it is unrealistic that robots learning autonomously would have access to that information, or because we do not even know which state representation best facilitates solving a given problem.

Without labels for perception, we use end-to-end learning to optimize perception as part of the larger system that describes behavior "from pixels to torques". The problem of this approach is again one of regularization. To prevent overfitting of such a complex system, we need to incorporate additional prior knowledge.

The overfitting problem is amplified if we assume that robot will mostly not learn from labeled examples that specify the torques it should apply for a given situation. For robots that mostly learn from unlabeled data and sparse reward signals, end-to-end learning objectives do not provide sufficient feedback to optimize the learnable parameters in a

way that will generalize well. What is the alternative?

We already got a glimpse of an alternative in this chapter, when we trained DHFs in an unsupervised way by predicting observations rather than states. This example shows how we can use additional learning objectives as a proxy for the actual objective, which is end-to-end performance. As we will see in the next chapter, predicting data is not the only kind of objective, but we can have a range of learning objectives that relate data and predictions in a number of interesting ways. And we will also see how these learning objectives can capture powerful prior knowledge about interactions with the physical world, which we call robotic priors.

5

Learning State Representations Through Physics-based Priors

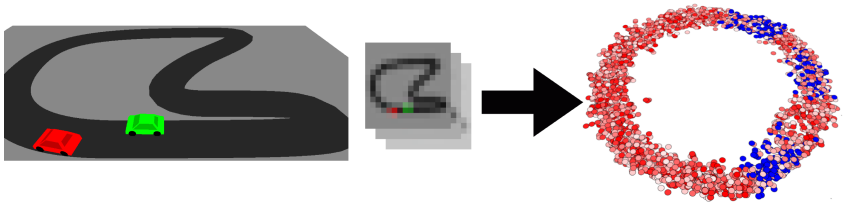
5.1 Introduction

This chapter proposes an alternative to the end-to-end learning approach from the previous chapter based on proxy learning objectives. It focuses on encoding prior knowledge not by restricting the hypothesis space but by defining appropriate learning objectives.

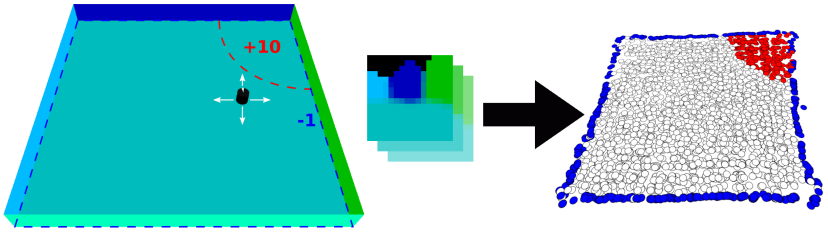
The problem setting for this chapter is reinforcement learning, where the robot should learn a behavior that maps observations to actions in a way that maximizes long term reward. Learning this function end-to-end by trial and error is inefficient because of the sparse reward signal. Instead, we split the behavior into two parts, an observation-state-mapping that extracts a task-relevant state description from the current observation and a policy takes this state and decides on the next action.

This chapter addresses the problem of learning the first part—the mapping from observations to task-relevant states—in an unsupervised way, i.e. without any state labels. The goal is to learn state representations that facilitate learning the second part, the policy, from a sequence of observations, actions, and rewards.

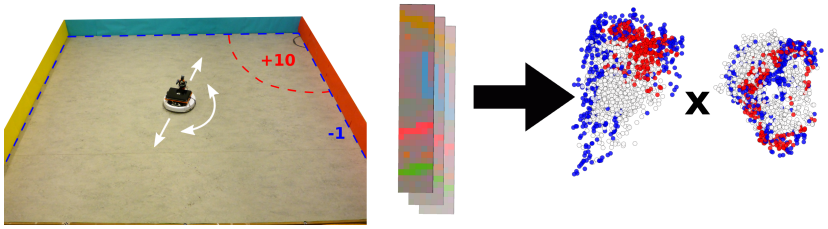
Towards this end, we exploit prior knowledge about how robots interact with their environment. Regardless of the task, robots always interact with the real world, which is governed by the laws of physics. As the robot's internal state captures properties of the world, the same laws that apply to the physical world must also apply to this internal state representation. Robots can learn representations that are consistent with physics by exploiting prior knowledge about interacting with the physical world, which we term



(a) simulated slot car racing task



(b) simulated (and simplified) navigation task



(c) navigation task with a real robot

Figure 5.1: State representations learned from visual input in (a) a simulated slot car racing task, (b) a simulated (and simplified) navigation task, and (c) a navigation task with a real robot. For (a) and (b) the state representations clearly capture the pertinent information for each task, i.e., the position of the slot car on the circular race track and the location of the mobile robot, respectively. For (c) the learned state representation is five-dimensional and, therefore, difficult to inspect visually. The plot shows two projections of the state space onto the first principal components.

robotic priors.*

In the main part of this chapter, we formulate five robotic priors. Then, we formulate consistency with these priors in the form of loss functions, such that they can be used as learning objectives to state representation learning. Finally, we explain how, by minimizing this loss function, we can learn state representations that allow robots to effectively learn new tasks using standard reinforcement learning. We call this method *learning with robotic priors* (LRP).

We analyze our approach in depth in simulated robotic tasks based on visual observations: a slot car racing task with two cars and a navigation task (see Figure 5.1). We verify these results in an experiment with a real robot in a similar navigation task (see Section 5.6.2). These tasks contain distractors that influence the visual observations of

*Since *physics* was the first source of prior knowledge that we had identified, we referred to these priors also as *robotic priors* in our publications. To stay consistent with the literature, I will use the same term in this chapter. Please note that in the introduction and conclusion, I use *robotic priors* more inclusively to refer to prior knowledge in robotics, which can come from the task, algorithms, or physics.

the robot but that are irrelevant for the task. With LRP, the robot learns to extract a low-dimensional state representation from high-dimensional visual observations that ignores these task-irrelevant distractions. We show that this change of representation greatly simplifies the subsequent learning problem and thereby substantially improves reinforcement learning performance. We also demonstrate that state representations that were learned for one task can also be transferred to new tasks.

We also present two extensions of this approach: 1) learning state representations for multiple tasks using an additional prior and 2) learning structured state spaces that consist of a position and velocity part by adding this structure as a hard constraint on the hypothesis space.

Finally, we relate this approach to learning with side information, which is an alternative perspective on machine learning that can be viewed as a generalization of the approach presented in this chapter. The main idea is that relating side information (such as actions and rewards) to learnable functions (such as a function from observations to states) can help to learn representations and regularize learning.

5.1.1 Contributions

In this chapter, we make the following contributions:

Conceptual Contributions

- We identify a set of robotic priors and implement consistency with these priors in the form of learning objectives.

- We generalize insights from this approach by relating it to learning with side information, a new perspective on machine learning, in which our work is only one particular pattern among many others.

Technical Contributions

- We introduce learning with robotic priors (LRP), a method for unsupervised learning of state representations that implements robotic priors as learning objectives. LRP learns state representations without supervision that ignore task-irrelevant information and improves generalization of subsequent reinforcement learning. Our source code and data is available at: `https://github.com/tu-rbo/learning-state-representations-with-robotic-priors`

- We present a multi-task extension, MT-LRP, which classifies experience into different tasks and learns separate state representations for each task. MT-LRP

works by adding two new learning objectives and implementing the observation-state-mapping as a gated network.

- We present position velocity encoders (PVEs), which extend LRP to learn a state space that is structured into a position and a velocity part by combining additional learning objectives with constraints in the hypothesis space.

Empirical Contributions

- We show that robotic priors enable state representation learning that is invariant to perspective, ignores distractors, identifies task-dimensionality, improves reinforcement learning, and transfers to related tasks. The presented extensions demonstrate that this approach scales to multiple tasks and is well suited for learning structured state representations.

5.1.2 Outline

The rest of this chapter is organized as follows:

- **Section 5.2 Background and Related Work** introduces the state representation learning problem and surveys related approaches to learning state representations
- **Section 5.3 Robotic Priors** explains the concept of robotic priors and defines the priors that we use in this work.
- **Section 5.4 Optimizing Consistency with Robotic Priors** formulates these priors in the form of learning objectives and demonstrates how they can be optimized in a state representation learning algorithm.
- **Section 5.5 Experiments and Results** describes the experiments and analysis experimental results.
- **Section 5.6 Extension 1: Learning State Representations for Multiple Tasks** extends this approach to multi-task learning.
- **Section 5.7 Extension 2: Adding Position-Velocity Structure to the State** extends this approach to learning structured state spaces.
- **Section 5.8 Relation to Learning with Side Information** relates our work on learning state representations to learning with side information, which presents a new perspective on machine learning and the use of prior knowledge through learning objectives.
- **Section 5.9 Conclusion** summarizes the findings of this chapter and discusses future research directions.

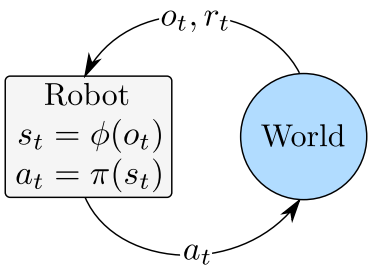


Figure 5.2: The robot-world-interaction. At time t, the robot computes the state s_t from its observation o_t using observation-state-mapping ϕ. It chooses action a_t according to policy π with the goal to maximize future rewards $r_{t+1:\infty}$.

5.2 Background and Related Work

5.2.1 The State Representation Learning Problem

As robots must rely on their task-general sensors, they cannot directly perceive their task-specific state. Instead, they must extract the state from sensory observations. *State representation learning* is the problem of learning to extract a state description from a history of interactions (i.e. previous observations, actions, and rewards) in order to enable efficient learning of the policy. We formalize this problem as learning a mapping ϕ to the current state, such that $s_t = \phi(o_{1:t}, a_{1:t-1}, r_{1:t})$. Given s_t, the robot selects the next action according to its policy: $a_t = \pi(s_t)$.

In this chapter, we focus on a special instance of this problem assuming that the state can be estimated from a single observation. This reduces the state representation learning problem to learning an observation-state-mapping ϕ, where $s_t = \phi(o_t)$ (see Figure 5.2).

To make this simplification, we must assume that the observation has the Markov property. Technically, this turns the problem into a Markov decision process such that we could use the raw observation as state. However, as we will see later in this chapter, the distance metric in the observation space (e.g., camera images) often does not enable the robot to learn the task. Even with Markov observations, robots must learn more useful state representations.

Note that sensory input of robots is often local and, therefore, observations are non-Markov for many robotic tasks. For some tasks, we can make observations Markov by including sensory inputs from multiple time steps in o_t. However, the dimensionality of o_t multiplies with the number of time steps that are taken into account and, ultimately, the state can depend on the entire history of observations, actions, and rewards. Alternatively, one could learn a state-filter instead of a observation-state-mapping (similar to the previous chapter but without any supervision). With every new action, observation, and reward, the filter would update the state. We think that it is crucial to investigate these extensions in future work, e.g. based on the differentiable particle filters (see Chap-

ter 5). For now, we assume Markov observations and restrict ourselves to learning an observation-state-mapping. In the next section, we outline how we address this problem in a robotics-specific way.

5.2.2 Approaches to State Representation Learning

State representation learning is an instance of representation learning for interactive problems. The goal of state representation learning is to find a mapping from observations—or, more generally, from histories of interactions—to states that allow choosing the right actions. State representation learning is related to the problem of abstraction selection in hierarchical reinforcement learning, which deals with selecting a state representation from a given set (van Seijen et al., 2014; Konidaris and Barto, 2009) or choosing a subset of state dimensions from a given representation (Cobo et al., 2014). By contrast to this work, we want to emphasize the need to construct state representations from sensory input without relying on predefined task-specific state features.

Note that state representation learning is more difficult than the standard dimensionality reduction problem, which is addressed by multi-dimensional scaling (Kruskal, 1964) and other methods (Roweis and Saul, 2000; Tenenbaum et al., 2000; Hadsell et al., 2006) because they require knowledge of distances or neighborhood relationships between data samples in state space. The robot, on the other hand, does not know about semantic similarity of sensory input beforehand. In order to know which observations correspond to similar situations with respect to the task, it has to solve the reinforcement learning problem (see Section 5.2.1), which it cannot solve without a suitable state representation. The question is: What is a good objective for state representation learning? We will now look at different objectives that have been proposed in the literature and relate them to our robotic priors (which we will define in Section 5.3.2 and formulate as objectives in Section 5.4.1).

Compression of Observations: Lange et al. (2012) obtain state representations by compressing observations using deep autoencoders. This approach relies on the prior that there is a simple (low-dimensional) state description and on the prior that this description is a compression of the observations. While we use the same simplicity prior, we believe that it is important to also take time, actions, and rewards into account.

Temporal Coherence: Slow feature analysis (Wiskott and Sejnowski, 2002) finds a mapping to states that change as slowly as possible, guided by the prior that many properties in our world change slowly over time. This method has been used to identify a representation of body postures of a humanoid robot (Höfer et al., 2010) as well as for solving reinforcement learning tasks with visual observations (Legenstein et al., 2010). Luciw and Schmidhuber (2012) showed that slow feature analysis can approximate proto-value functions (Mahadevan and Maggioni, 2007), which form a compact basis for all value functions. Incorporating the same prior, dimensionality reduction methods have used temporal distance to estimate neighborhood relationships (Jenkins and Matarić, 2004).

Slowness or temporal coherence is an important robotic prior that LRP also relies on. However, the purpose of the state is to provide a sufficient statistic based on which the robot can choose the right actions. Therefore, the actions of the robot must also be considered for learning the state representation. The following methods and ours take this valuable information into account.

Predictive and Predictable Actions: These approaches find state representations in which actions lead to simple, predictable transformations. Action respecting embeddings, proposed by Bowling et al. (2005), aim at a state space in which actions are distance-preserving. Sprague (2009) introduced predictive projections, which find a representation such that actions applied to similar states result in similar state changes. Predictive state representations, proposed by Littman et al. (2002), define states as success probabilities for a set of tests, where a test is a prediction about future observations conditioned on future actions. Boots et al. (2011) showed how predictive state representations can be learned from visual observations. As we will see, these ideas are related to the proportionality prior, the causality prior, and the repeatability prior in this chapter.

The problem with these methods—and all other methods discussed so far—is that they generate task-general state representations. This is problematic when the robot lives in a complex environment and there is no common state representation that works for all tasks. Therefore, we will use the reward to focus on the task-specific aspects of the observations and ignore information irrelevant for the task.

Interleaving State Representation Learning and Reinforcement Learning: The approaches presented so far learn state representations first to then use them for reinforcement learning. We will now discuss approaches that combine these steps. Piater et al. (2011) use decision trees to incrementally discriminate between observations with inconsistent state-action values according to the reinforcement learning algorithm. This method is comparable to an earlier approach of Singh et al. (1995), which minimizes the error in the value function by clustering states. Menache et al. (2005) also adapt the state representation during reinforcement learning; they represent the state as a set of basis functions and adapt their parameters in order to improve the value function estimate.

Methods in this category rely on causality of values. They assume that the value is attributable to the state. To compute the value, they must solve the reinforcement learning problem. These steps can be decoupled by factorizing the value function into the reward function and the state transition function. This is used by the following approaches, and also by ours.

Simultaneously Learning the Transition Function: In earlier work (Jonschkowski and Brock, 2013), we proposed to learn the state transition function together with the state representation to maximize state predictability while simultaneously optimizing temporal coherence. A drawback of this approach is that it ignores the reward and, therefore, cannot distinguish task-relevant from irrelevant information.

Simultaneously Learning Transition Function and Reward Function: Some approaches jointly learn an observation-state-mapping, a transition function, and a reward function, differing in their learning objective. Hutter (2009) proposes minimizing the combined code length of the mapping, transition function, and reward function. Duell et al. (2012) learn these functions to predict future rewards conditioned on future actions. Jetchev et al. (2013) maximize state predictability and reward discrimination to learn a symbolic state representation.

These approaches build models of state transitions and rewards to enforce state predictability and reward discrimination. Contrary to this approach, we define our learning objective in terms of distances between state-samples, similar to the idea of multidimensional scaling (Kruskal, 1964). In this way, we can assure the existence of transition and reward functions for the state representation without having to model them explicitly.

5.3 Robotic Priors

5.3.1 Priors About Interacting with the Physical World

According to Bengio et al. (2013), the key to successful representation learning is the incorporation of "many general priors about the world around us." They proposed a list of *generic priors* for artificial intelligence and argue that refining this list and incorporating it into a method for representation learning will bring us closer to artificial intelligence. This is exactly what we are trying to do in the context of robotics. However, we believe that the problem of artificial intelligence is too broad and that therefore generic priors are too weak. We try to find stronger priors about the problem structure by focusing on robotic tasks, which involve interacting with the physical world. We call such priors *robotic priors*.

Scholz et al. (2014) follow the same idea of using physics based priors to learn a forward model (or transition function) of the world. Instead of using a generic hypothesis space for the forward model, they use a restricted parametric hypothesis space based on physics. This can be very helpful for robotic tasks because we know that the right forward model must lie in this restricted hypothesis space. However, this work assumes to already have a suitable state representation consisting of poses and velocities as well as knowledge about the exact semantics of every state-dimension. Our work focuses on how a robot could learn to extract such a state representation from sensory input.

The interaction between the robot and the real world is structured by the laws of physics. From this fact, we can derive robotic priors that capture characteristics of all robotic tasks based on physical interaction.

5.3.2 Five Robotic Priors

In this section, we define five robotic priors.

Simplicity: *For a given task, only a small number of world properties are relevant.* This prior is related to Occam's razor, a widely accepted principle in science. In our context, Occam's razor will favor state representations that exclude irrelevant information, thereby leading to a lower-dimensional reinforcement learning problem and improving generalization.

Slowness / Temporal Coherence: *Task-relevant properties of the world change gradually over time* (Wiskott and Sejnowski, 2002). This prior is related to Newton's first law of motion. Physical objects have inertia and change their velocity only gradually as a result of external forces. However, temporal coherence also applies to more abstract properties than physical motion, as most changes in the world occur gradually. The temporal coherence prior favors state representations that obey this principle as the robot transitions between states.

Proportionality: *The amount of change in task-relevant properties resulting from an action is proportional to the magnitude of the action.* This prior results from Newton's second law of motion, $F = m \cdot a$. If an action represents the application of a certain force on an object of a fixed mass, the acceleration evoked by this force is constant. This holds true for robot movements and physical interactions with other objects but also generalizes to more abstract processes. As the robot does not know the magnitudes of its actions beforehand, this prior enforces that multiple executions of the same action result in the same amount of state change.

Causality: *The task-relevant properties, together with the action, determine the reward.* This and the next prior resemble Newton's third law of motion or, more generally, causal determinism. If the same action leads to different rewards in two situations, these situations must differ in some task-relevant property and should thus not be represented by the same state. Consequently, this prior favors state representations that include the relevant properties to distinguish these situations.

Repeatability: *The task-relevant properties and the action together determine the resulting change in these properties.* This prior is analogous to the previous one—for states instead of rewards—and also results from Newton's third law of motion. This principle is enforced by favoring state representations in which the consequences of actions are similar if they are repeated in similar situations. The repeatability prior and the causality prior together constitute the Markov property of states.

Note that most of these priors are defined in terms of actions and rewards. Thus, they do not apply to passive systems that can only observe but not act. These priors are also not generic artificial intelligence priors applicable to *all* tasks and environments, as artificial environments can be very different from our world, e.g., not obeying Newton's laws of motion. However, restricting the problem space to the physical world allows us to define useful priors.

But even in the physical world, there are still counterexamples for each prior. Proportionality does not hold when the robot hits a wall and its position remains constant even though it attempts to move with a certain velocity. Causality is violated due to sensory aliasing when the robot cannot distinguish two situations with different semantics. Repeatability is contradicted by stochastic actions. As we will see, all of these counterexamples are present in the experiments in this chapter and LRP is robust against them.

Our approach can handle such counterexamples because the robotic priors are not strict assumptions that have to hold for the method to work. Instead, LRP finds a state representation that is consistent with these priors as much as possible. As the robotic priors capture the general structure of interactions with the physical world, they are useful, even in the presence of counterexamples.

5.3.3 Additional Priors in Extensions 1 and 2

The two extensions that are described after the evaluation of the original method, introduce additional robotic priors. Extension 1 handles multi-task learning using the following prior:

Task-Coherence: *The task only changes between episodes not within an episode.* This prior can be viewed as a form of slowness assumption on the task. The task of a robot naturally will change much slower than its state. Here, we assumed that tasks can only change between episodes but this assumption can potentially be relaxed.

Extension 2 introduces structure into the state space, which allows it to define priors specifically for positions, velocities, and accelerations, leading to a new opportunities for defining robotic priors. The new priors introduced by this extension are:

Variation: *Positions of relevant things vary.* As the robot explores its task and manipulates its environment, the positions of task-relevant objects (including itself) will vary—otherwise there is not much that the robot could learn.

Conservation: *Velocity magnitudes change slowly.* This prior derives from the *law of conservation of energy*, which states that the total energy in a closed system remains constant.

Controllability: *Controllable things are relevant.* The objects that can be controlled by the robot are likely relevant for its task. If the robot acts by applying forces, controllable things could be those whose accelerations correlate with the actions of the robot.

5.4 Optimizing Consistency with Robotic Priors

In this section, we formulate state representation learning as an optimization problem by turning our robotic priors into loss functions. Then, we turn the theory into a method that minimizes this loss function, thereby learning a state representation that reflects the priors defined above.

5.4.1 Formulation as Optimization Problem

We will now turn the robotic priors into a loss function L that is minimized when the state representation is most consistent with the priors. We construct loss terms for all robotic priors (except for the simplicity prior, see below) and define L as their weighted sum

$$\begin{aligned} L(D, \hat{\phi}) = \omega_{\mathrm{t}} L_{\text{temporal coherence}}(D, \hat{\phi}) + \omega_{\mathrm{p}} L_{\text{proportionality}}(D, \hat{\phi}) \\ + \omega_{\mathrm{c}} L_{\text{causality}}(D, \hat{\phi}) + \omega_{\mathrm{r}} L_{\text{repeatability}}(D, \hat{\phi}). \end{aligned}$$

Each of these terms is computed for an observation-state-mapping $\hat{\phi}$ and data of the robot interacting with the world, $D = \{o_t, a_t, r_t\}_{t=1}^{n}$, which consist of sensory observations o, actions a, and rewards r for n consecutive steps. The observation-state-mapping $\hat{\phi}$ is then learned by minimizing $L(D, \hat{\phi})$ (the $\hat{\cdot}$ indicates that ϕ changes during learning).

By linearly combining these loss terms, we assume independence between the robotic priors. They could also be combined non-linearly, but the existence of independent counterexamples for each individual prior supports our assumption. There is a weight ω assigned to each loss term because the typical magnitude varies significantly between these terms. The weights can counteract this imbalance. Additionally, the weighting allows us to stress the importance of individual priors by using a higher weight for the corresponding loss term.

We will now describe how the individual robotic priors are defined as loss terms. For better readability, we will write $\hat{s}_t$ instead of $\hat{\phi}(o_t)$ when we refer to the state at time t according to the observation-state-mapping $\hat{\phi}$.

Simplicity Loss: The simplicity prior is not formulated as a loss term but implemented by enforcing the state representation to be of fixed low dimensionality. Future work should explore formulations of this prior as a loss term, e.g., defined in terms of sparsity, the number of dimensions, or other properties of the state representation.

Temporal Coherence Loss: States must change gradually over time. We denote the state change as: $\Delta\hat{s}_t = \hat{s}_{t+1} - \hat{s}_t$. The temporal coherence loss is the expected squared magnitude of the state change.

$$L_{\text{temp.}}(D, \hat{\phi}) = \mathbf{E}\Big[\|\Delta\hat{s}_t\|^2\Big].$$

Proportionality Loss: If the robot has performed the same action at times t_1 and t_2, the states must change by the same magnitude $\|\Delta\hat{s}_{t_1}\| = \|\Delta\hat{s}_{t_2}\|$.

The proportionality loss term is the expected squared difference in magnitude of state change after the same action was applied:

$$L_{\text{prop.}}(D, \hat{\phi}) = \mathbf{E}\Big[(\|\Delta\hat{s}_{t_2}\| - \|\Delta\hat{s}_{t_1}\|)^2 \;\Big|\; a_{t_1} = a_{t_2}\Big].$$

Causality Loss: Two situations at times t_1 and t_2 must be dissimilar if the robot received different rewards in the following time step, even though it had performed the same action, $a_{t_1} = a_{t_2} \wedge r_{t_1+1} \neq r_{t_2+1}$.

The similarity of two states is 1 if the states are identical and approaches 0 with increasing distance between them. To compute the similarity of state pairs, we can use any differentiable similarity function. Previously, we have proposed to use the exponential of the negative distance, $e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|}$, following research from psychology (Shepard, 1987). However, the bell shaped function, $e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}$, is a reasonable alternative and in our experience leads to improved performance.

Therefore, we define the causality loss as the expected similarity of the state pairs for which the same action leads to different rewards:

$$L_{\text{caus.}}(D, \hat{\phi}) = \mathbf{E}\Big[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \;\Big|\; a_{t_1} = a_{t_2}, r_{t_1+1} \neq r_{t_2+1}\Big].$$

Repeatability Loss: States must be changed by the actions in a repeatable way. If the same action was applied at times t_1 and t_2 and these situations are similar (have similar state representations), the state change produced by the actions should be equal, not only in magnitude but also in direction.

We define the repeatability loss term as the expected squared difference in the state change following the same action, weighted by the similarity of the states:

$$L_{\text{rep.}}(D, \hat{\phi}) = \mathbf{E}\Big[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2 \;\Big|\; a_{t_1} = a_{t_2}\Big].$$

5.4.2 Learning with Robotic Priors (LRP)

We will now show how a linear mapping from observations to states can be learned by minimizing the loss function. We call the resulting method *learning with robotic priors* (LRP) because it optimizes consistency with those priors.

Choosing the Weights: The weights in the loss function balance the relative importance of the different robotic priors. We believe that all proposed priors are important. Therefore, we chose the weights such that they provide gradients with similar magnitudes for the tasks we are interested in: $\omega_{\text{t}} = 1$, $\omega_{\text{p}} = 5$, $\omega_{\text{c}} = 1$, $\omega_{\text{r}} = 5$.

Note that little effort was put into choosing these parameters because LRP is robust against a range of weight assignments. Simple uniform weighting also worked well in all of our experiments (Jonschkowski and Brock, 2014).

Approximating the Loss Function: We compute the expected values in the loss function by taking the mean over training samples. For the proportionality loss, the causality loss, and the repeatability loss, this would require taking into account all $O(n^2)$ pairs of training samples. For reasons of computational efficiency, we approximate each of these computations by only considering a subset of all pairs: For each training sample, we randomly choose 10 other samples such that the resulting pairs fulfill the conditions of the respective loss term. The mean is then taken over this subset of training sample pairs.

Learning a Linear Observation-State-Mapping: LRP learns a linear observation-state-mapping,

$$\hat{s}_t = \hat{\phi}(o_t) = \hat{W}(o_t - \mu_o),$$

where μ_o is the mean of all observations during training and $\hat{W}$ is a weight matrix that is adapted by performing gradient descent on the approximated loss function L. Linear functions form a very limited hypothesis space, but LRP can easily be extended to non-linear functions approximators, e.g., using feature expansion, kernel approaches, or artificial neural networks, as we will demonstrate in Sections 5.6 and 5.7.

Note that subtracting the mean observation is equivalent to subtracting the mean state and does not affect the loss function, which is defined over relative and not over absolute properties of the state samples. The only purpose of this step is to center the state representation. This can be useful for subsequent reinforcement learning depending on which form of function approximation is used (neural networks, for example, rely on mean centered input data).

Regularization: Without regularization, this learning scheme can lead to overfitting. Minimizing the loss for the training data can potentially increase the loss for unseen test data. In this case, the learned observation-state-mapping would not generalize well to new data. We approach this problem by introducing a L1 regularization on the weights of the observation-state-mapping. We find the best observation-state-mapping by solving the following optimization:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[L(D, \phi) + \lambda l_1(\phi) \right],$$
$$\text{where } \; l_1(\phi) = \sum_{i,j} |W_{i,j}|.$$

The regularization parameter λ can vary greatly depending on the task. However, it can be estimated using cross-validation. In all our experiments, we do this by using the

first 80% of the data for training and the remaining 20% as validation set. Then we do a grid search over different values for $\lambda \in \{0, 0.0003, 0.001, 0.003, ..., 3, 10\}$ and pick the value that achieved the lowest loss on the validation data after 100 steps of gradient descent. Using this value, we learn the final state representation on the entire data set for 100 steps.

Gradient Descent: The representation learning process starts with initializing the weight matrix W with small random weights that are uniformly chosen from the interval $[-0.05, 0.05]$. Using this observation-state-mapping, it projects all experienced observations into the state space, computes the loss of this state representation and the gradient with respect to W, and changes W accordingly. This process is repeated, in our case for 100 learning steps. We dynamically adapt the step size for each weight dimension using improved Rprop with weight backtracking (Igel and Hüsken, 2003).

Analytical Gradient: The gradient can be computed analytically by partial derivatives with respect to all parameters $\hat{W}_{i,j}$ for each loss term (see Appendix A.1 for the derivation). The weighted sum of these terms, together with the gradient of the regularization term, gives the total gradient.

$$\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{temp.}}(D, \hat{\phi}) = \mathbf{E}\Big[2 \underbrace{(\hat{s}_{i,t+1} - \hat{s}_{i,t})}_{\Delta \hat{s}_{i,t}} \underbrace{(o_{j,t+1} - o_{j,t})}_{\Delta o_{j,t}}\Big].$$

$$\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{prop.}}(D, \hat{\phi}) = \mathbf{E}\Big[2 \left(\|\Delta \hat{s}_{t_2}\| - \|\Delta \hat{s}_{t_1}\|\right) \Big(\frac{\Delta \hat{s}_{i,t_2} \Delta o_{j,t_2}}{\|\Delta \hat{s}_{t_2}\|} - \frac{\Delta \hat{s}_{i,t_1} \Delta o_{j,t_1}}{\|\Delta \hat{s}_{t_1}\|}\Big) \;\Big|\; a_{t_1} = a_{t_2}\Big].$$

$$\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{caus.}}(D, \hat{\phi}) = \mathbf{E}\Big[-2e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} (\hat{s}_{i,t_2} - \hat{s}_{i,t_1}) (o_{j,t_2} - o_{j,t_1}) \;\Big|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\Big].$$

$$\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{rep.}}(D, \hat{\phi}) = \mathbf{E}\Big[2e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \Big((\Delta \hat{s}_{i,t_2} - \Delta \hat{s}_{i,t_1}) (\Delta o_{j,t_2} - \Delta o_{j,t_1}) - (\hat{s}_{i,t_2} - \hat{s}_{i,t_1}) (o_{j,t_2} - o_{j,t_1}) \|\Delta \hat{s}_{t_2} - \Delta \hat{s}_{t_1}\|^2\Big) \;\Big|\; a_{t_1} = a_{t_2}\Big].$$

Reinforcement Learning Method: To evaluate the utility of learned state representations, we used a standard reinforcement learning method that can handle continuous state spaces: fitted Q-iteration (Sutton and Barto, 1998) based on normalized radial basis function (RBF) features that are repeated for every action.

We generate these features by applying the k-means algorithm on all state samples to find 100 radial basis function centers. We set the standard deviation of the radial basis functions to be half of the average distance from a center to the closest next center. The activation of the radial basis functions are normalized such that they sum to one. This vector of state features is repeated for every action and stacked to build, for example, a 2500 dimensional feature vector when there are 25 discrete actions. In this state-action-feature vector, all elements are zero except for the radial basis function activations for the corresponding action.

The state-action-value-function, or Q-function, describes the value of applying a certain action in a certain state. We define it as a linear function of the radial basis feature vector, $\hat{Q}(s,a) = \hat{\beta}^T f_{\text{RBF}}(s,a)$. We initialize $\hat{\beta}$ with small random values and perform one step of Q-iteration to estimate the state-action-values of all training samples.

$$\forall_t : \hat{\mathbb{Q}}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_a \left[\hat{Q}(s_{t+1}, a)\right],$$

where $\gamma = 0.9$ is used to exponentially discount future rewards. We then fit the linear function $\hat{Q}$ to the estimated state-action-values $\hat{\mathbb{Q}}(s_t, a_t)$ by assigning $\hat{\beta}$ to the vector that minimizes the sum of squared errors:

$$\hat{\beta} = \underset{\beta}{\text{argmin}} \sum_t \Big(\underbrace{\beta^T f_{\text{RBF}}(s_t, a_t)}_{Q(s_t,a_t)} - \hat{\mathbb{Q}}(s_t, a_t) \Big)^2.$$

We alternate Q-iteration and Q-fitting until convergence. The resulting Q-function allows the robot to greedily choose the actions with the highest value, $\hat{\pi}(s) = \underset{a}{\text{argmax}} \left[\hat{Q}(s,a)\right]$.

5.5 Experiments and Results

In this section, we extensively evaluate LRP in simulated and real robotic tasks with visual observations. We test how well LRP can map 768-dimensional visual observations (16×16 pixels for red, green, and blue) into a two-dimensional or five-dimensional state space.

First, we look at the learning process in detail to understand how the state representation, the loss, and the gradient evolve. Then, we analyze learned state representations to gain insight into the capabilities of our approach. We start by comparing learned state

representations for a simple navigation task* when the robot observes the scene from different perspectives, having either an egocentric view or a top-down view. The results show that, in both cases, LRP learns a mapping to the same pertinent dimensions.

Next, we investigate in a slot car racing task† how LRP can handle task-irrelevant distractors. To the best of our knowledge, this is the first time that this problem is addressed in state representation learning, even though we view it as essential (in any real-world environment, the observations of robots will be subject to task-irrelevant distractions). We will see that LRP can separate task-relevant properties of the observation from irrelevant information. Then, we will also introduce distractors into the navigation task and see that the performance of LRP is not influenced by them. After that, we analyze how the state representations for both tasks change if they are given more dimensions than necessary to solve the task. The results show that, in one of the tasks, LRP can identify the minimal state dimensionality irrespective of the distractors.

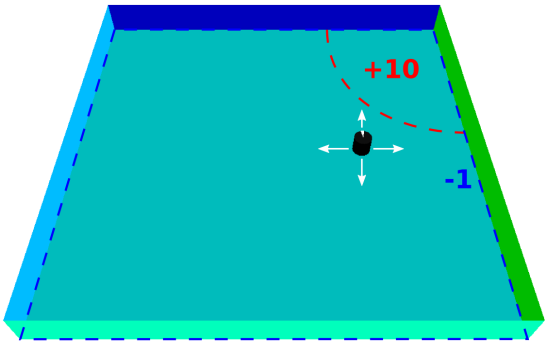
Finally, we demonstrate that the learned state representations can substantially improve the performance of subsequent reinforcement learning. We explain how this effect results from improved generalization. Additionally, we show that state representations learned for one task can also enable reinforcement learning in other related tasks. In the end, we verify our results from simulation in an experiment with a real robot.

5.5.1 Learning Process

During learning, the observation-state-mapping is continually changed according to the gradient of the loss function to minimize the loss function of the resulting state representation. In this experiment, we will look at this learning process in detail for one run of the simple navigation task.

Simple Navigation Task: In the simple navigation task (see Figure 5.3a), the robot is located in a square-shaped room of size 45×45 units with 4-units-high walls of different colors. The robot has a height and diameter of 2 units. The orientation of the robot is fixed but it can control its up-down and left-right velocity choosing from $\{-6, -3, 0, 3, 6\}$ units per time step. The robot thus has 25 discrete actions. These actions are subject to Gaussian noise with 0 mean and standard deviation of 10% of the commanded velocity. The task of the robot is to move to the top right corner without hitting a wall. If the distance to this corner is less than 15 units, the robot gets a reward $+10$ unless it is touching a wall, in which case it gets a negative reward of -1. The robot perceives its environment through a camera with a wide angle lens (field of view 300°). The 16×16-pixel RGB image is represented as a 768-dimensional observation vector. The example

*The navigation task is based on experiments in the literature (Boots et al., 2011; Sprague, 2009).
†The slot car racing task is inspired by an experiment of Lange et al. (2012).



(a) Simple navigation task

(b) Observation

Figure 5.3: Simple navigation task with fixed orientation. The robot gets positive reward in the top right corner and negative reward when it runs into a wall. Subfigure (b) shows the observation in scene (a).

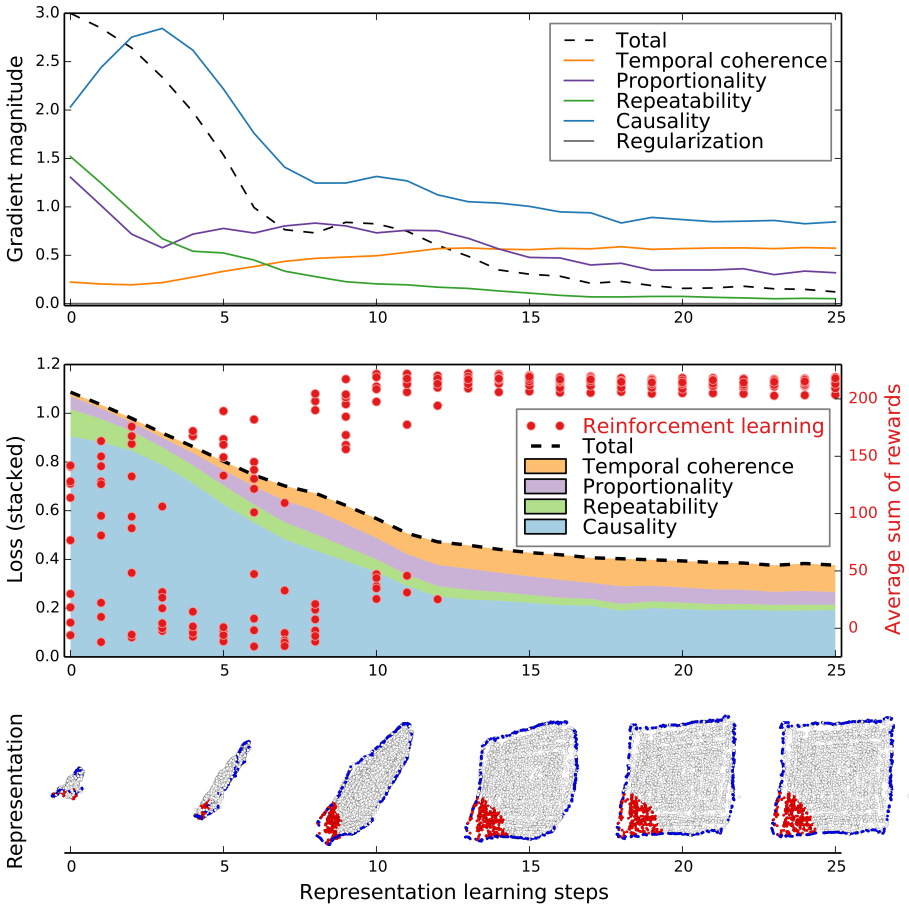


Figure 5.4: State representation learning process. The plots show how the gradient magnitude (top), the loss on validation samples (middle), the reinforcement learning success (middle), and the state representation (bottom) change during learning. Every red dot in the middle plot shows the evaluation of one policy based on the learned state representation. Every dot in the bottom plot represents one observation mapped into the state space. The colors denote the reward that was received in that situation ($\mathrm{red} = +10$, $\mathrm{blue} = -1$, regularization: $\lambda = 0.03$).

observation (see Figure 5.3b) shows the dark blue wall in the middle and the green and the light blue wall on either side of the image.

Experimental design: The robot explored its environment performing 5000 random actions. Based on this experience, it learned a mapping from the 768-dimensional observation space to a two-dimensional state representation. We interrupted the learning process after every learning step, applied a standard reinforcement learning method to learn a policy, and evaluated it for 20 episodes of 25 steps starting from different initial positions. We repeated this reinforcement learning and evaluation cycle ten times for every learning step due to the randomness in the reinforcement learning method.

Results—Loss Gradient Unfolds State Representation: The learning process starts from a randomly initialized observation-state-mapping and the corresponding state representation (see bottom left in Figure 5.4). Every learning step changes the mapping and unfolds the state representation. When this process converges, the state representation resembles the task-relevant properties of the world—in this case the location of the robot (see bottom right in Figure 5.4).

The observation-state-mapping is changed following the loss gradient (see dashed line in top plot in Figure 5.4). This gradient is the sum of the loss term gradients, each of which can point in a different direction (see colored lines in top plot, Figure 5.4). The causality gradient pushes apart states that lead to different rewards after performing the same action. It is most active in the beginning when all states are near each other. The temporal coherence gradient pulls consecutive states closer together. It becomes more active when the state representation is spread out. The repeatability gradient enforces deterministic state changes and is also most active in the beginning when the state representation is still very chaotic. The proportionality gradient equalizes distances across the state space. Its activation declines as the representation increasingly complies with this objective.

Throughout the training process, the causality loss and gradient are larger then the corresponding terms of the other priors. This results from the fact that causality directly opposes the other priors. Causality can be optimized by expanding the state space while all other robotic priors can be optimized by collapsing all states to a single point. To reach a balance, the causality gradient must be as large as the other gradients combined.

In the beginning, the different gradient terms do not (entirely) oppose each other and there is a large total gradient according to which the observation-state-mapping is modified. Following the total gradient, its magnitude decreases until a local minimum of the loss function is reached. This is where the different gradients point in opposing directions, the total gradient is zero, and the state representation converges.

Note that the resulting state representation discriminates between states that have different delayed rewards although immediate rewards are identical. For example, most

locations in the room have the same immediate reward, zero, but they are still represented by different states. It is important to differentiate between them because they lead to different delayed rewards after performing a number of actions. LRP discriminates between these situations due to a synergy of the causality prior and the repeatability prior. The causality prior discriminates between different immediate rewards. The repeatability prior enforces deterministic state changes and thereby propagates this discrimination to situations from which states with different rewards can be reached. The temporal coherence prior and the proportionality prior make this process more robust using additional knowledge.

Results—Loss as Proxy for Reward: As the loss decreases during learning, the reinforcement learning success increases (see middle plot, Figure 5.4). In the beginning, the reinforcement success varies greatly between different evaluations. This is due to the randomness in the reinforcement learning method we used, especially in random assignment of radial basis function centers. But during the learning process this changes in two ways.

First, the reinforcement learning failures (with total reward less than 50) stop after learning step twelve. These failures are due to incorrect generalization from experience to new situations. They are caused by a state representation that does not capture the pertinent dimensions of the task. With a good state representation, however, every execution of the reinforcement learning method generalizes well.

Second, the successful reinforcement learning trials (with more than total reward of 50) become better. There is a strong correlation between this increase and the loss decrease.

Both results show that minimizing the loss function can be a good proxy objective for maximizing reward when learning a state representation. If we wanted to use the reward as objective directly, every representation learning step would involve learning policies and evaluating them by performing actions in the real world. Even worse, the number of required evaluations increases exponentially with the number of parameters in the observation-state-mapping, because we would have to compute a gradient with that many dimensions. The loss function, on the other hand, allows to directly compute the gradient.

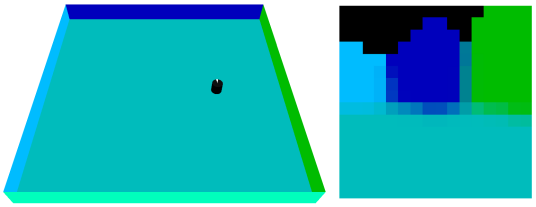
5.5.2 Invariance to Perspective

To investigate whether LRP is invariant to perspective, we test it in two versions of a simple navigation task with different visual observations, viewing the scene from the robot's perspective and viewing the scene from the top. In both versions, the robot learns a state representation that reflects its location which is exactly the information required to solve the task.

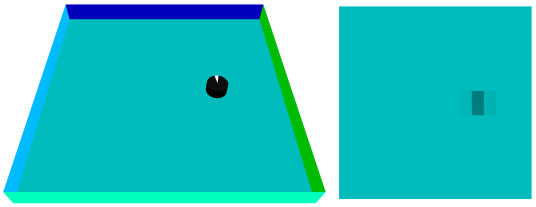
Perspectives in the Simple Navigation Task: The simple navigation task, as described in section 5.5.1, has egocentric observations from a camera on the robot. We will refer to this task as egocentric view version of the simple navigation task (see Figure 5.5a). The top-down view version of this task is identical to this version, except for the observations which are images of the entire scene taken from the top by a static camera. In these images, the robot shows as a dark spot against the background. For the robot to be always visible, we had to increase its diameter to 4 units in this version of the task (see Figure 5.5b).

Experimental Design: We performed the following experiment for both versions of the task. The robot explored its environment performing 5000 random actions and learned a mapping from the 768-dimensional observation space to a two-dimensional state representation based on this experience.

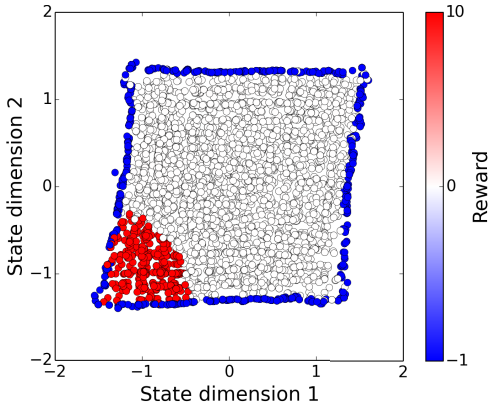
Result—Equivalent State Representations: To compare the learned state representations, we have plotted the state estimates for 5000 training steps based on egocentric view observations (see Figure 5.5c) and based on top-down view observations (see Figure 5.5d). In both cases, the samples form a square in state space, suggesting that the state is an estimate of the location of the robot in the square room. We can show that this is in fact what is learned by coloring each state sample according to the reward that the robot received in this state during training. The result resembles very accurately the reward definition in Figure 5.3a. The learned state representation is equivalent in both versions of the task. There are two orthogonal axes in the state representations that correspond to the coordinates of the robot. Of course, these axes in state space do not have to align between experiments; they can be rotated or flipped.



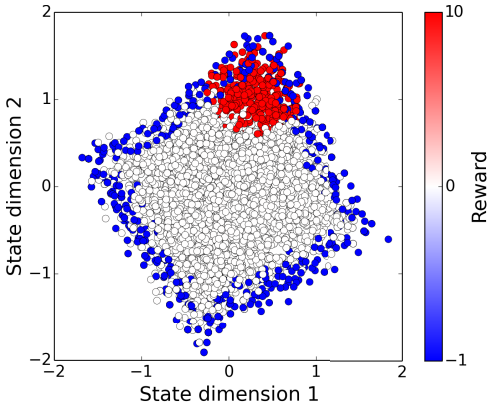
(a) Egocentric view version



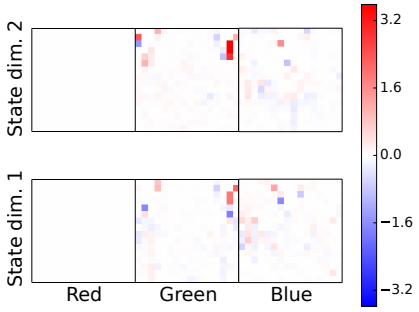
(b) Top-down view version



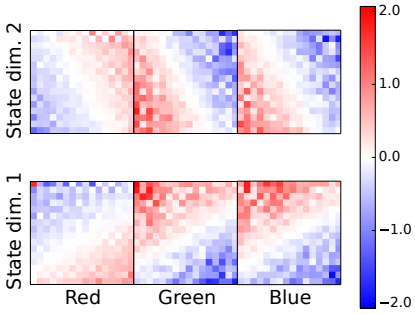
(c) State representation (egocentric)



(d) State representation (top-down)



(e) Observation-state-mapping (egocentric)



(f) Observation-state-mapping (top-down)

Figure 5.5: Results for two versions of the simple navigation task with fixed orientation. The observation of the robot is either an egocentric view (a) or a top-down view (b) of the scene. (c) and (d) show the representations of 5000 training samples in state space for both versions. Each dot results from applying the learned observation-state-mapping $\phi^\star$ to an observation. The color relates the state samples to the reward received in that situation during training. (e) and (f) display the learned observation-state-mappings, i.e., the respective weight matrices which linearly project pixel values in the different color channels to state dimensions. Regularization and final loss on validation samples: egocentric version, $\lambda = 0.03$, $L = 0.36$; top-down version, $\lambda = 0.1$, $L = 0.52$.

Result—Different Observation-State-Mappings: In the two versions of the task, the sensory observations of the robot are very different. Nevertheless, it learned a mapping from these observations to the task-relevant dimensions: the location of the robot. Note that the mappings from observations to states must be very different to result in this identical state representation (see Figures 5.5e and 5.5f).

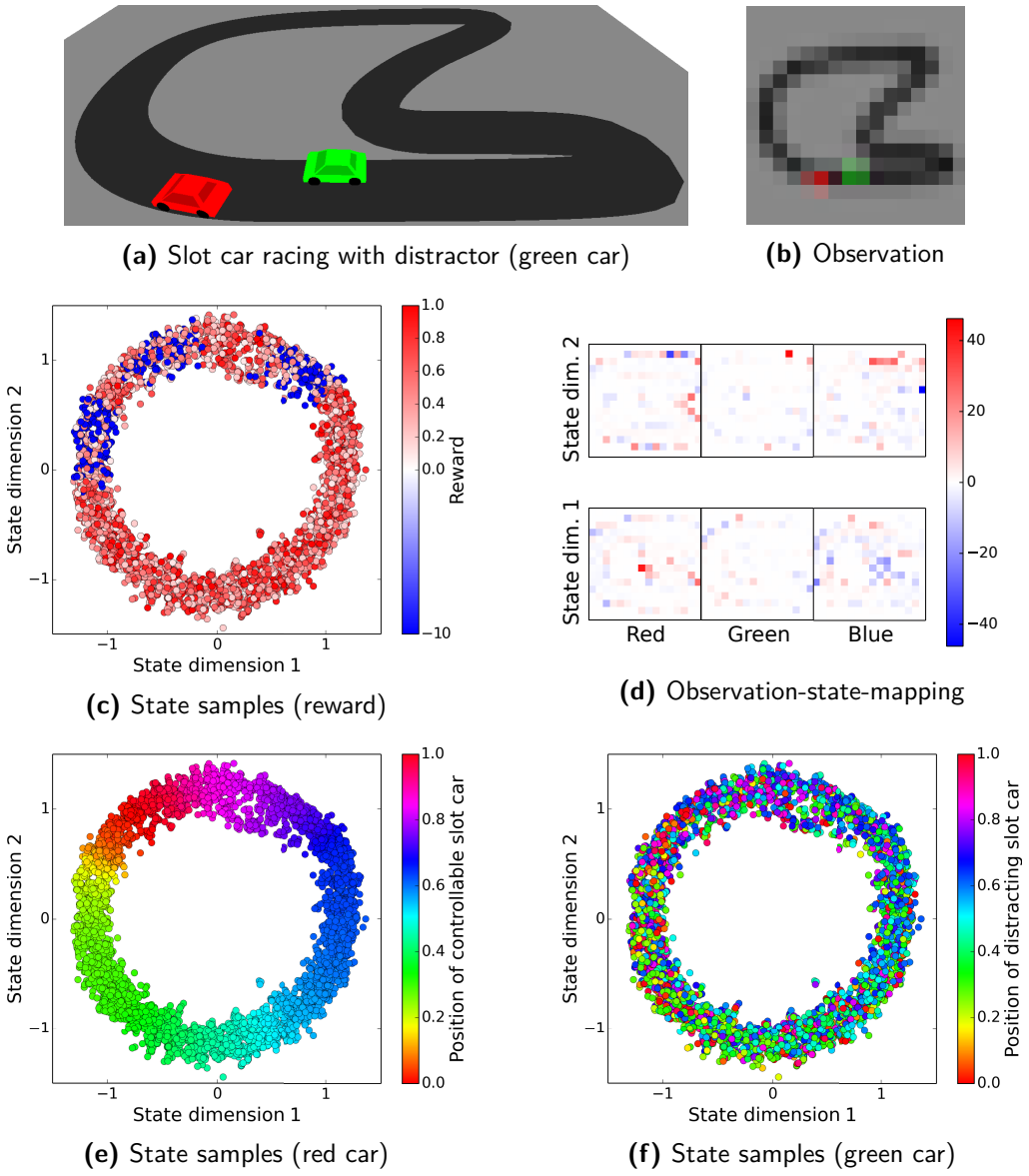
Result—Loss Explains Quality Difference: Even though the state representations for both versions are equivalent in principle, they are of different quality. The state representation based on egocentric observations is much clearer compared to the more blurry state representation based on top-down view observations, where situations with different rewards are mapped to similar states. Reinforcement learning based on such a representation will be less successful compared to the state representation based on egocentric observations. Presumably, the low resolution top-view observations cannot be (linearly) mapped to the location of the robot with high accuracy because they do not contain sufficient information.

This quality difference is also reflected in the final loss on validation samples L. For the egocentric version, $L = 0.36$ and for the top-down version of the task, $L = 0.52$. This reconfirms that the defined loss is a sensible quality measure for state representations.

5.5.3 Ignoring Distractors

In this experiment, we test whether LRP distinguishes task-relevant properties of the observations from irrelevant information. First, we investigate this in a slot car racing task with two cars. While the robot observes two slot cars, it can only control one of them. The other car does not play a role in this task apart from potentially distracting the robot. The robot does not know beforehand which car is relevant for the task. Second, we extend the simple navigation task with task-irrelevant distractors and test LRP in this scenario.

The Slot Car Racing Task: An example scene from the slot car racing task is shown in Figure 5.6a. The robot can control the velocity of the red car, choosing from $[0.01, 0.02, \ldots, 0.1]$ units per time step. The velocity is subject to zero mean Gaussian noise with standard deviation of 10% of the commanded velocity. The robot's reward is proportional to the commanded velocity—unless the car goes too fast in a sharp turn and is thrown off the track. In this case, the robot gets a negative reward of -10. The robot cannot control the green slot car. The velocity of this car is chosen randomly from the same range as for the red car. The green slot car does not influence the reward of the robot or the movement of the red car. The robot observes the scene from the top through a 16×16-pixel RGB image (see Figure 5.6b).



(a) Slot car racing with distractor (green car)

(b) Observation

(c) State samples (reward)

(d) Observation-state-mapping

(e) State samples (red car)

(f) State samples (green car)

Figure 5.6: Results for the slot car racing task (a) with visual observations (b). Plots (c), (e), and (f) show the learned state representation. The color relates state samples to reward (c), the position of the relevant car (e), and the position of the distractor (f). Plot (d) shows the weight matrix of the learned observation-state-mapping. The regularization and final validation loss are $\lambda = 0.001$ and $L = 0.49$.

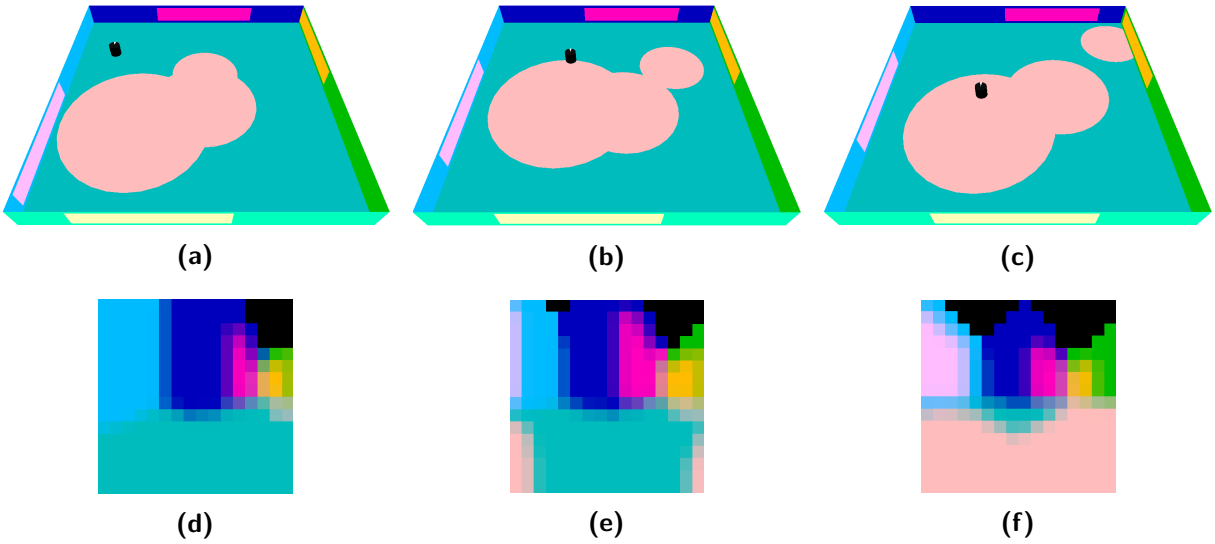
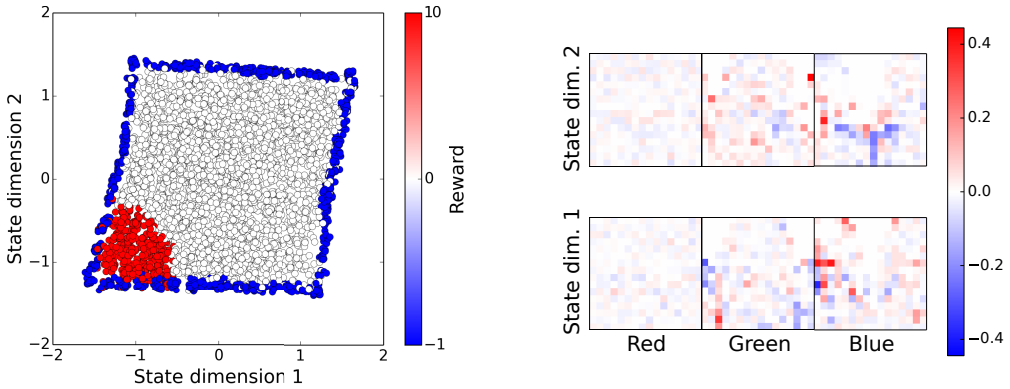


Figure 5.7: Distractors in the simple navigation task. (a-c) show three situations at an interval of ten time steps. The robot is exploring while the distractors move randomly. (d-f) show the corresponding observations (note how they are influenced by the distractors).

Experimental Design: The robot explored randomly for 5000 time steps and then learned a mapping from 768-dimensional observations to two-dimensional states.

Result—Relevant Information Extracted: To understand the learned state representation, we have plotted the states of the 5000 training steps with one dot per state sample (see Figure 5.6c). The states form a circle which corresponds to the topology of the track. We have colored the state samples according to the reward. The three blue clusters in this plot correspond to the three sharp turns on the track, where the robot had routinely driven too fast during training and thus received negative reward. We also colored the states according to the ground truth position of the red slot car (see Figure 5.6e) and the green slot car (see Figure 5.6f). The figures show that the position along this circle in state space corresponds to the position of the controllable slot car on the track. One round of the red slot car corresponds to a circular trajectory in state space. LRP was able to distinguish task-relevant from irrelevant information in the observations and, at the same time, found a compressed representation of these pertinent properties.



(a) State representation

(b) Observation-state-mapping

Figure 5.8: Results for simple navigation task with distractors. Regularization and final validation loss are $\lambda = 0.3$ and $L = 0.36$.

Result—Two-Dimensional Representation of Position: The position of the slot car on the track is one-dimensional and, thus, could be captured in one-dimensional state representation. However, such a representation would not conform with the temporal coherence prior. The positions at the beginning and the end of the track would be maximally apart in such a representation, even though they are actually very close together due to the circularity of the track. To represent a circular one-dimensional property, we need a two-dimensional Euclidean space.

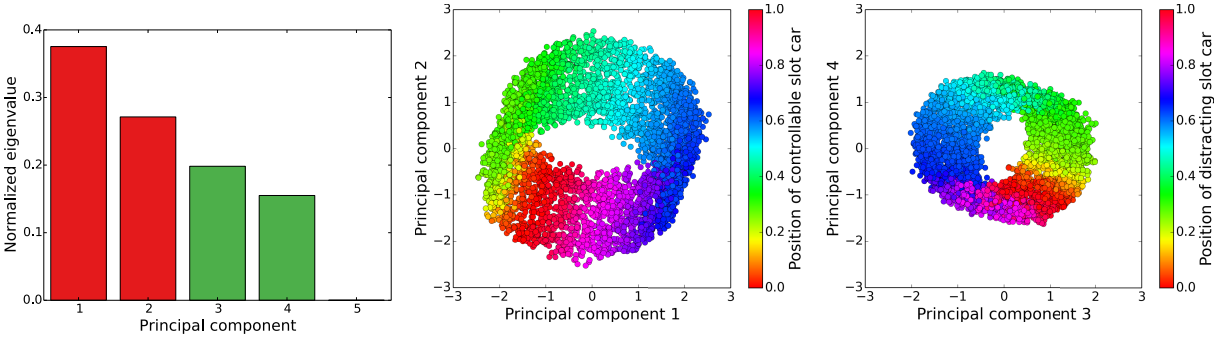
Simple Navigation Task with Distractors: To verify the results from the slot car racing task in the simple navigation task, we added seven distractors to this task: three circles on the floor and four rectangles on the walls that move randomly (see Figure 5.7). The distractors are observed by the robot but do not influence its movement or reward. They are irrelevant for the task and should thus not be included in the state representation.*

*The colors of the distractors were chosen to be equal to their background in the green and the blue color channel of the RGB image. They can be ignored by not taking into account the observation dimensions that correspond to the red color channel.

Experimental design: The robot explored randomly for 5000 time steps and then learned a mapping from 768-dimensional observations to a two-dimensional state.

Result—Identical State Representation: The state representation that the robot learned in the presence of visual distractors captures the task-relevant properties and entirely ignores the distractors. In fact, the learned representation is identical to the one learned without the presence of distractors (compare Figures 5.8a and 5.5c). Also, the validation loss after training is 0.36 in both tasks.

Result—Different Observation-State-Mappings: There are major differences in the learned observation-state-mappings between the task with and without distractors (compare Figures 5.8b and 5.5e). In the task including the distractors, the weights are more evenly distributed across the image. In this task, the highest weight is smaller by an order of magnitude compared to the task without distractors. This makes sense because the observations vary much more in this task, leading to stronger regularization ($\lambda = 0.3$ instead of 0.03) and a more robust mapping.



(a) Eigenvalues of state samples

(b) Components 1 and 2 represent the controllable red car

(c) Components 3 and 4 represent the distracting green car

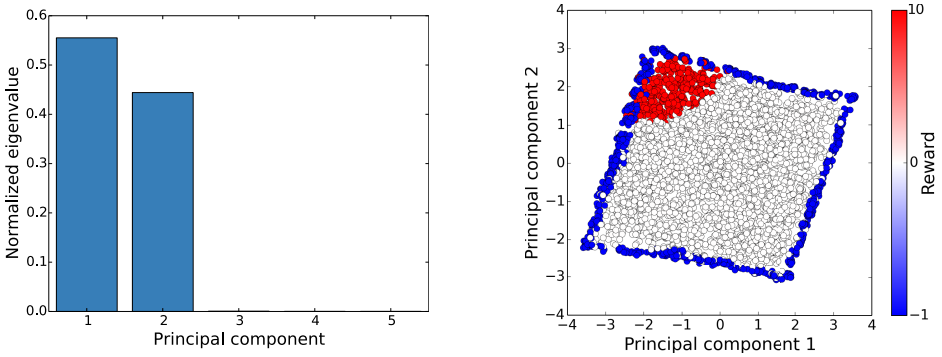
Figure 5.9: Results for the slot car task with a five-dimensional state space ($\lambda = 0.003$, $L = 0.45$).

5.5.4 Mapping to a Higher-Dimensional State Space

In the previous experiments, we gave the robot an appropriate number of dimensions for the state representation. In this section, we investigate what happens in these same examples when the robot learns state representations with more dimensions than necessary. To this end, we repeated the experiments for the slot car task and for the simple navigation task with distractors, only now learning a five-dimensional instead of a two-dimensional state representation. After exploration for 5000 time steps and state representation learning, we took the 5000×5-matrix M containing the estimated states for these experiences and performed a principal component analysis of this matrix.

Result—Alternative Explanations for the Reward: In the slot car task, the state sample matrix M has rank four. There are two larger eigenvalues and two smaller eigenvalues (see Figure 5.9a). If we project the state samples on their first two principal components, we can see that the dimensions with the larger eigenvalues correspond to the position of the controllable red slot car on the race track (see Figure 5.9b). The third and fourth principal component correspond to the position of the non-controllable green slot car (see Figure 5.9c).

If the green car is irrelevant for the task, why is it represented in the state? The robot maximizes state dissimilarity between situations where it received different rewards even though it performed the same action. If the robot chooses the same velocity but the slot car is thrown off one time while it stays on track another time, it makes the states of these two situations dissimilar. The most powerful discrimination between these situations is the position of the red slot car. But sometimes small differences in position or the stochasticity of the actions can make the difference between the two outcomes. The robot thus finds alternative explanations like the position of the green slot car. The eigenvalues show that this property has a lower impact on the state than the position of the controllable red slot car. LRP includes these alternative explanations if there are



(a) Eigenvalues of state samples

(b) Projected state samples

Figure 5.10: Results for the navigation task with a five-dimensional state space ($\lambda = 1$, $L = 0.36$).

enough dimensions in the state space. When the state space is limited, the method focuses on pertinent dimensions as shown in Section 5.5.3.

Result—Identifying the Dimensionality of the Task: For the navigation task, we find that all but the first two eigenvalues of M are close to zero (see Figure 5.10a). The rank of the matrix is effectively two. This means that all state samples lie on a plane in the five-dimensional state space. We can visualize this plane by projecting the state samples on their first two principal components (see Figure 5.10b). The state space again corresponds to the location of the robot just as in the two-dimensional experiment. Apparently, the robot does not need to include additional properties of the world into the state representation in order to explain the reward. Thus, even with a five-dimensional state space, the robot learns that the task is two-dimensional and captures only those properties of its observation ignoring the distractors.

5.5.5 Improved Performance in Reinforcement Learning

The preceding experiments have shown some promising features of LRP. But in the end, the quality of state representations can only be measured by their utility for subsequent learning. In this experiment, we will see that LRP can substantially improve reinforcement learning performance and that it needs very few data to do so.

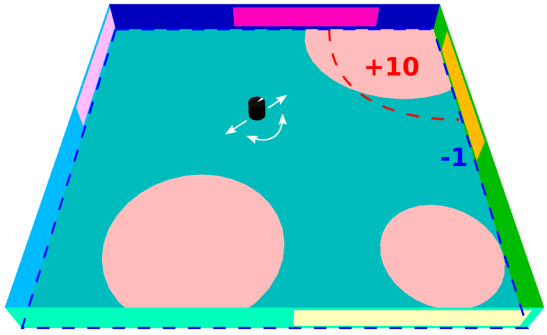


Figure 5.11: Extended navigation task with distractors.

Extended Navigation Task: To construct a more challenging task for this experiment, we extended the navigation task by allowing the robot to change its orientation. The robot can turn and move forwards or backwards choosing its rotational velocity from $[-30, -15, 0, 15, 30]$ degrees per time step and its translational velocity from $[-6, -3, 0, 3, 6]$ units per time step for a total of 25 actions. All actions are subject to Gaussian noise with 0 mean and 10% standard deviation. This task includes the same visual distractors described before. The objective of the task also did not change. The robot must move to within 15 units of the top right corner, where it gets a reward of 10 unless it runs into a wall, in which case it gets a reward of -1 (see Figure 5.11).

Experimental design: The robot explored its environment ϵ-greedy. With probability 0.9, it performed a random action, otherwise it performed the best action according to its policy. After every 500 time steps, the exploration was interrupted. From its accumulated experience, the robot learned an observation-state-mapping and a policy which it used in the following 500 exploration steps. After learning, the robot was also tested for 20 episodes, each consisting of 50 steps starting from random initial configurations. Based on these tests, we computed the average sum of rewards. This cycle of exploration, learning, and testing was carried out until the robot had explored for 7500 time steps. The entire learning experiment was repeated ten times.

We performed the experiment multiple times with the same reinforcement learning method and different state representations: the five-dimensional state representation from LRP, the five slowest features of the observations (computed using linear slow feature analysis (Wiskott and Sejnowski, 2002)), the first five principal components of the observations, and the raw 768-dimensional observation. To get an upper bound on the reinforcement learning performance, we also compared with a simpler version of this task without distractors in which the robot has access to its ground truth pose. In this case it uses its position (normalized to $[-1, 1]$) and the cosine and sine of its orientation as state, which we consider an optimal representation for this task.

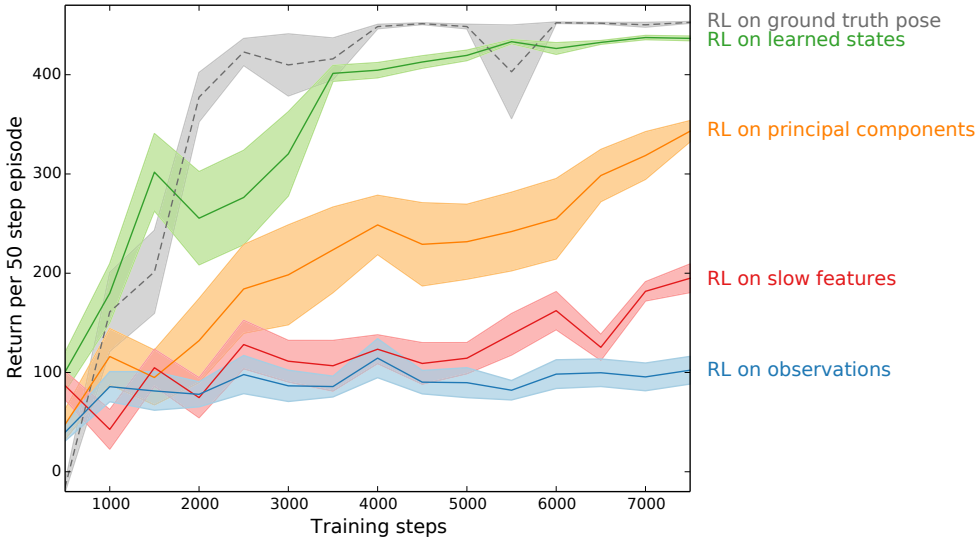


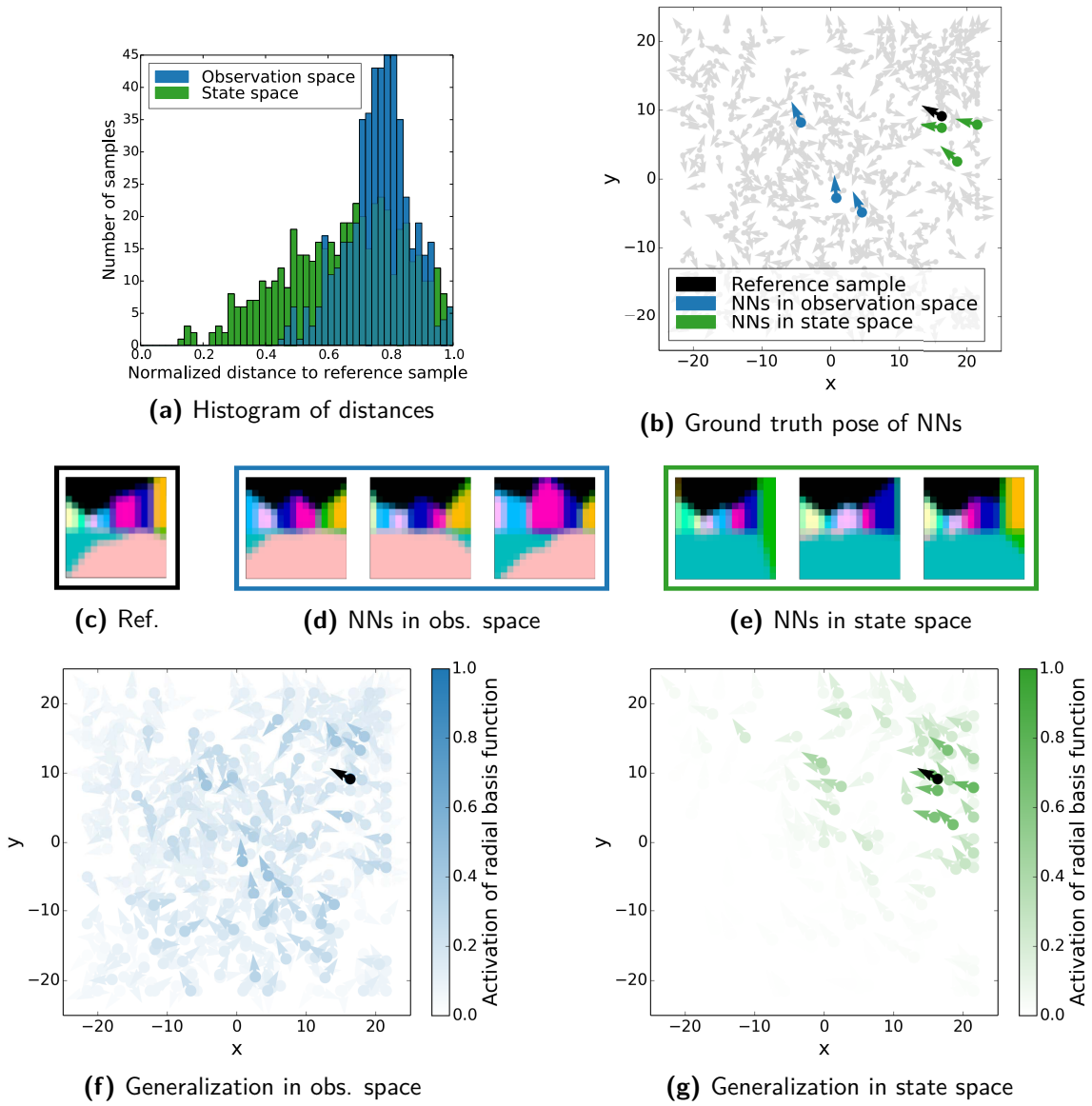
Figure 5.12: Reinforcement learning (RL) curves for different state representations. Lines show means, surfaces display their standard errors.

Result—Improved Generalization: We want to start analyzing the results in Figure 5.12, by comparing LRP (green) against performing reinforcement learning directly on the raw observations (blue). These results show that the robot was not able to learn the task by directly applying reinforcement learning on the observations. When tested, the robot usually rotated in place or did not move at all. Using the state representation found by LRP, however, it learned this task getting an average reward of about 430 per 50 step episode. This corresponds to reaching the goal area after 7 steps on average without ever running against a wall. Where does this difference come from? LRP basically acts as a regularization on the learning problem by extracting the right information from sensory input. This leads to better generalization from experiences to new situations.

Virtually every learning algorithm, including the one we used for reinforcement learning, generalizes according to the smoothness prior, which is: "similar inputs lead to similar outputs" or in this case "similar states require similar actions". Therefore, experiences are generalized to situations that have a similar representation where similarity is usually defined in terms of Euclidean distance. How do the distance metrics implied by the observation space and the state space compare?

We examined this with another experiment. In this experiment, the robot explored randomly for 5000 steps and learned a five-dimensional state representation based on this experience. For the following analysis, we used every tenth of the training samples (to make sure that they are sufficiently different from each other). We chose one of these samples as a reference and computed the distances between the reference sample and all other samples both in observation space and in state space. This analysis reveals two important differences that have a huge impact on generalization.

First, the distances from the reference to every other sample have a much smaller



(a) Histogram of distances

(b) Ground truth pose of NNs

(c) Ref.

(d) NNs in obs. space

(e) NNs in state space

(f) Generalization in obs. space

(g) Generalization in state space

Figure 5.13: Distances and generalization in the observation space (blue) compared to the learned state space (green). (a) shows the distances of different observation samples to the reference sample (c). (d) and (e) display the nearest neighbors (NNs) of the reference sample. The ground truth pose of these samples is displayed in (b). (f) and (g) illustrate how the distance metric affects generalization.

variance in observation space compared to state space. In other words: Every sample is roughly equally far away in observation space. The nearest neighbor (NN) of the reference is almost half as far away as the most distant sample (see Figure 5.13a). This effect is caused by the high dimensionality of the observation space. In this space, everything is far apart. One would need an exponential number of samples to adequately fill the space. This poses a major problem for the smoothness prior in subsequent learning. In order to generalize to the most similar situations, one will automatically also generalize to other less similar situations because their distances from the reference are alike.

Second, the nearest neighbors of the reference sample in observation space can have very different semantics with respect to the task. To illustrate this, we show the ground truth pose of the reference sample and its three nearest neighbors in both observation space and state space (see Figure 5.13b). The nearest neighbors in state space correspond to similar poses such that it makes sense to generalize knowledge about which action to take from one situation to its neighbors. For the nearest neighbors in observation space, this is not the case. The reference observation (see Figure 5.13c) and its nearest neighbors in observation space (see Figure 5.13d) exemplify why observations from these different poses are close in observation space: The arrangement of distractors in their visual input coincides. The alignment of the walls, on the other hand, which provides information about the robot's location, differs greatly from the reference observation.

Note that this second issue is not specific to the pixel-representation. The same applies to different image representations: histogram of colors, histogram of gradients, point-features, and in fact every generic representation. Generic representations must include these task-irrelevant distractors because for other tasks, for example "follow the red rectangle", this is the important information and other information is irrelevant and distracting.

Both issues violate the smoothness prior that is used by the reinforcement learning method. The consequences for generalization are shown in Figures 5.13f and 5.13g. The color shows the strength of the generalization from the reference sample to every other sample and vice versa (computed as the activation of a radial basis function centered at the reference sample).

Result—Pertinent State Representation: The two baselines, principal component analysis (orange) and slow feature analysis (red), also improve reinforcement learning performance to some degree due to the low dimensionality of the state representation. However, these methods also suffer from being distracted by information in the visual input that is not task-related.

Both methods cannot distinguish relevant from irrelevant information because they are purely based on observations. LRP takes actions and rewards into account and is, therefore, able to learn state representations that pertain to the task.

The reason why principal component analysis still performs reasonably well is that it

finds those dimensions that can explain the largest variances in the observations and the robot's pose is one important factor for that.

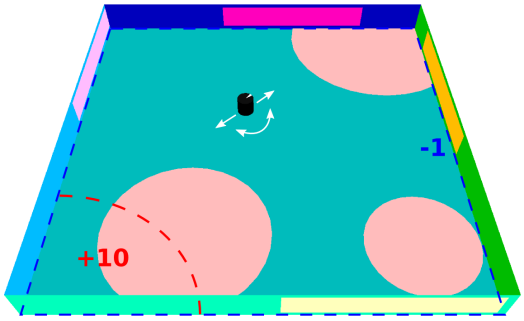
Result—Almost as Useful as Ground Truth Pose: We now compare the results of our approach to the upper bound of the reinforcement learning method—using the ground truth pose of the robot as state (gray, dashed line, see Figure 5.12). We have to keep in mind how much easier this task is compared to coping with 768-dimensional visual observations influenced by distractors. Still, LRP is able to learn a state representation that is almost as useful for reinforcement learning as the true pose of the robot. The final difference in average reward corresponds to taking one additional step until reaching the goal. The state representation learned by LRP was almost as good as the best omniscient state representation that we could think of.

Result—Little Training Data Required: Interestingly, LRP required few data to learn useful representations. Even at the very beginning, the learning curves based on the ground truth pose (gray) and based on the learned representation (green) are comparable. This shows that LRP needs less training data to learn a useful state representation than the reinforcement learning method needs to learn the right policy.

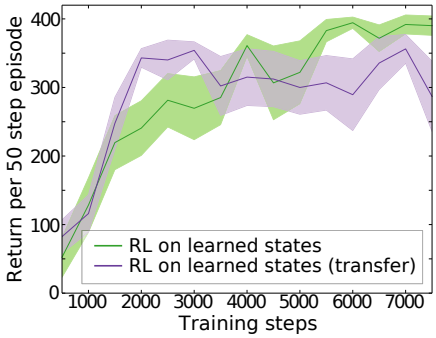
5.5.6 Transfer Learning

Ultimately, versatile robots must be able to learn a multitude of tasks, many of which are related. For effective learning, experience must be reused and transferred between tasks. For different tasks, the robot often needs to extract different information from its sensory input. If, for example, the robot is given the task to follow an object instead of moving to a fixed location, the absolute pose of the robot becomes meaningless and an object that was a distraction for the navigation task becomes important for the new task. While different tasks generally require different state representations, many tasks are also related, for example moving to different locations in the same environment. Such related tasks share or partly share useful state representations, for example the robot pose. Other tasks are composed of subtasks, for example moving to a certain location, picking something up, and then delivering it to another location. When tasks are combined from subtasks, their state representations could also be combined from the subtasks' state representations, which would allow to incrementally learn even very complex state representations.

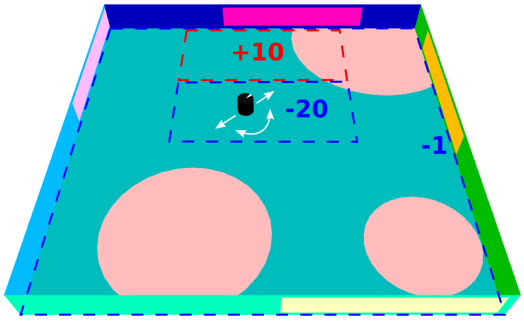
Such an incremental learning scheme requires that the learned state representations are general enough to be reused in higher level tasks. This is not self-evident for state representations learned for a specific task. Navigating to the top right corner, for example, only requires to know the direction and the distance to this corner. While these features form a useful state representation for this specific task, they do not enable the robot



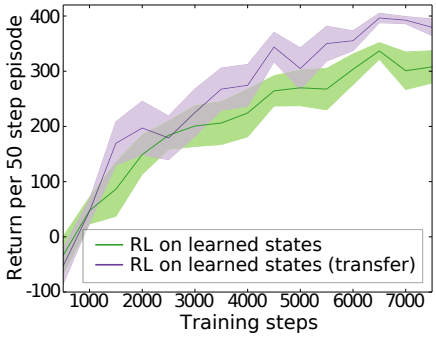
(a) Task A



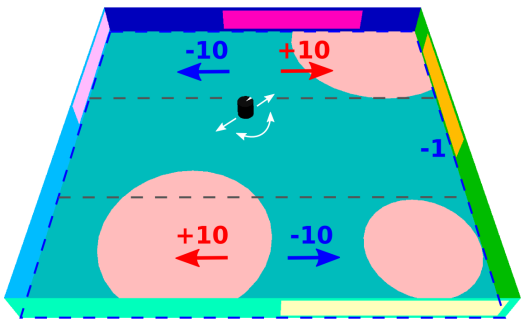
(b) Learning curves for task A



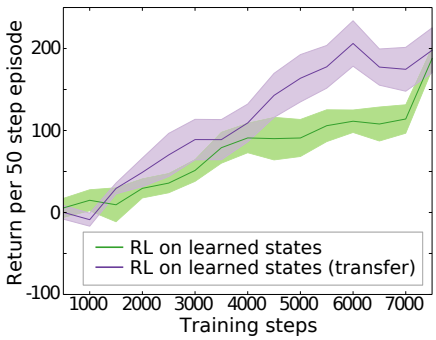
(c) Task B



(d) Learning curves for task B



(e) Task C



(f) Learning curves for task C

Figure 5.14: Transferring learned state representations from the extended navigation task to new tasks. Lines show means, surfaces display mean estimate standard errors

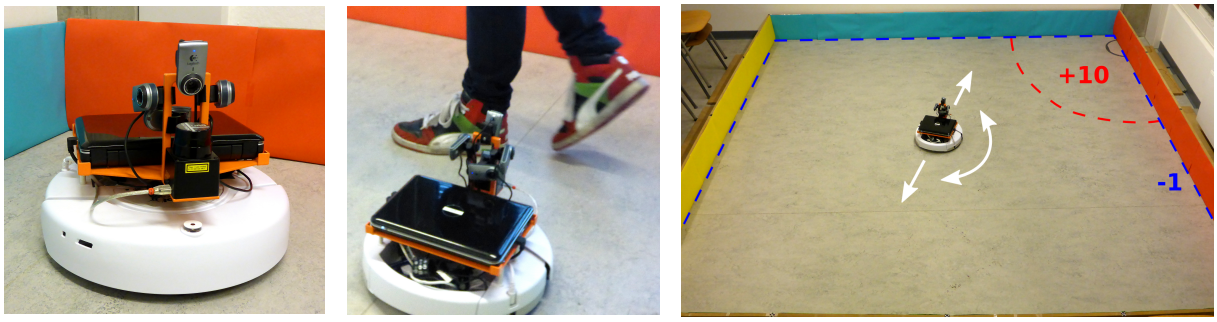
to navigate to other locations. Interestingly, our previous experiments suggest that this is not the kind of state representation that LRP produces. Instead, the learned state representations capture the task dimensions in a general way even though they were learned from data for a specific task. In this experiment, we want to quantify how reusable these state representations are across related tasks. We measure this by the utility of applying an observation-state-mapping that was learned for the extended navigation task in a set of new related tasks.

Three New Tasks: Task A is identical to the extended navigation task with distractors, except that the goal area has been changed to the opposite corner (see Figure 5.14a). In task B, the goal area is a rectangle in the top of the room. In this area, the robot receives a reward of +10. Next to this rectangle, there is another rectangular area in which the robot receives negative reward of −20 (see Figure 5.14c). In task C, the robot receives reward of +10 for moving right and negative reward of −10 for moving left when it is in the top third of the room. In the bottom third of the room, this is reversed: the robot is rewarded positively for moving left and negatively for moving right (see Figure 5.14e).

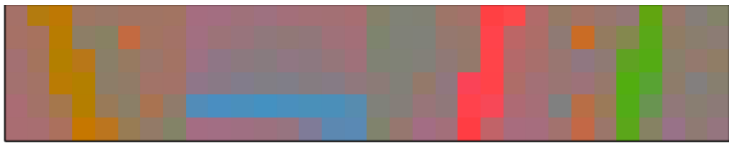
Experimental Design: In this experiment, we measure the utility of a transferred state representation for reinforcement learning based on the cumulative reward obtained by the robot. We use the same design as in Section 5.5.5, except that the observation-state-mapping is not learned continually during the trial in which it is tested. Instead, the state representation is learned from 5000 random exploration steps. The learned observation-state-mapping is then used in the new tasks without changing it. We compare the utility of state representations learned specifically for the new tasks and those transferred from the extended navigation task.

Result—Successful Transfer of Learned State Representations: We compare the reinforcement learning performance in tasks A, B, and C based on the transferred state representation to the performance based on the state representation learned for each specific task (see Figure 5.14). Since both state representations lead to similar learning curves, the transferred state representation must be as useful for solving tasks A, B, and C as the state representations specifically learned for each task. These results show that, across this set of related tasks, LRP is invariant to which specific task it was learned for.

Since task A is equivalent to the extended navigation task, it may be surprising that the reinforcement learning performance in task A is not as good as in the experiment in Section 5.5.5 (compare green curves in Figures 5.14b and 5.12). We think that this performance difference depends on whether the same or different data were used for state representation learning and reinforcement learning. When the learned observation-state-



(a) Mobile robot **(b)** Distractor **(c)** Real navigation task



(d) Observation from cameras (left, front, right, back)

Figure 5.15: The navigation task with a real robot (c). The observation (d) is a combined image from the down-sampled output of the four RGB cameras attached to the robot (a). (d) shows the yellow wall to the left, the blue wall in front, the red wall to the right, and the green wall in the back of the robot (three cameras are tilted). LRP was applied on this observation without any additional preprocessing. (b) shows a distractor in front of the robot.

mapping is applied to new data, it will not generalize perfectly. While the learned policy seems to be robust against these deviations during testing, the reinforcement learning method appears to be more sensitive, which might result in a non-optimal policy. For tasks B and C the learning curves suggest that the performance based on the transferred state representation might even be better than the performance based on the task-specific state representation. Overall, these results are very promising as they demonstrate the potential to reuse the learned state representations which is a prerequisite for incrementally combining them to form more complex state representations.

5.5.7 Verification on a Real Robot

Simulations are a great tool to conduct experiments under ideal conditions. However, simulations always carry the risk of ignoring important aspects of the problem. Therefore, we also need to verify our results in a real world scenario. For a video of this experiment, see: `https://youtu.be/BolevVGJk18`

Real navigation task: We tried to replicate the extended navigation task as closely as possible using an iRobot Create equipped with four standard web cams (see Figure 5.15a). This robot was placed in a square room with yellow, blue, red, and green walls of length 3.2 meters (see Figure 5.15c). The camera settings (especially contrast and saturation) were chosen such that the colored walls are clearly visible. The images of the four web cams were down-sampled to 8×6 pixels and combined into a single image which the robot received as 576-dimensional observation (see Figure 5.15d).

For computing the reward and for analyzing the results, it was necessary to estimate the pose of the robot. This was done using a particle filter based on input from a laser scanner on the robot. Neither the pose information, nor the laser scanner input was used by the robot during the task.

Experimental Design: The experimental design was similar to the navigation tasks in simulation. The robot explored ϵ-greedy (see Figure 5.16d) and was interrupted after every 500 steps to learn an observation-state-mapping and a policy using the same methods and parameters as in the simulation experiments. After 2000 training steps (about 50 minutes), the robot was tested five times starting from different positions in the room. The resulting trajectories are shown in Figure 5.16e. During the entire experiment, a person was acting as a visual distractor (see Figure 5.15b). The person walked around randomly, avoiding collisions with the robot, but otherwise moving independently from it.

Result—Verification on a Real Robot: After 2000 training steps, the robot had learned to move to the top right corner in the presence of the visual distractor (see Figure 5.16e).

This experiment reproduces our findings from simulation in the real world. LRP is able to extract the right information to solve this task from visual input, even in the presence of distractors. At least in this simple setting, it was able to cope with illumination changes, shadows and occlusion from the distractor, non-Gaussian noise in actions due to delays in the wireless connection and changes in the battery level, and many other effects that were not modeled in simulation.

5.6 Extension 1: Learning State Representations for Multiple Tasks

In this section, we extend LRP to an algorithm for learning state representations for multiple tasks by learning with robotic priors (MT-LRP). MT-LRP is able to acquire different low-dimensional state representations for multiple tasks in an unsupervised fashion. Importantly, MT-LRP does not require knowledge about which task is executed at

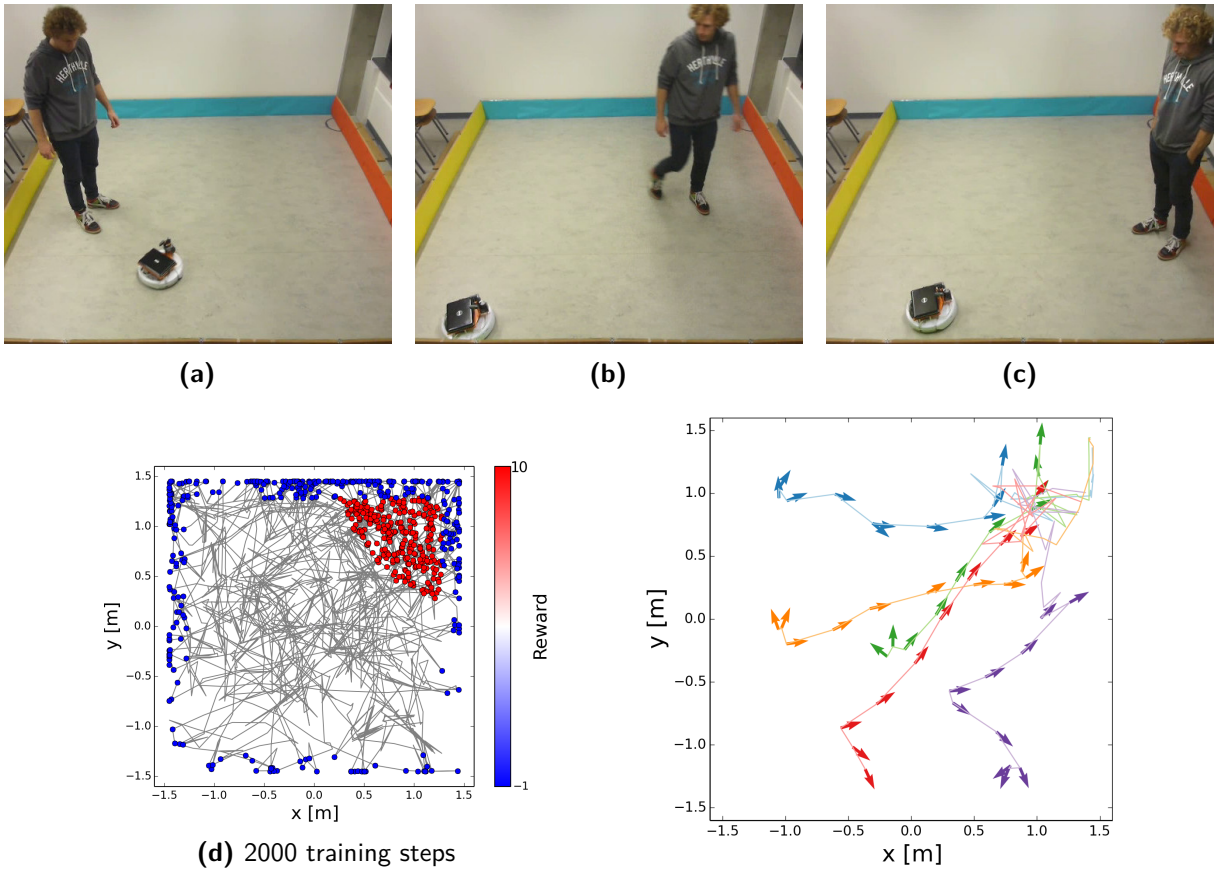


Figure 5.16: Training and results on a real robot. (a-c) show the robot and the moving distractor during the training phase. (d) shows the trajectory of the robot during its 2000 training steps. (e) displays five robot trajectories using the learned observation-state-mapping and policy.

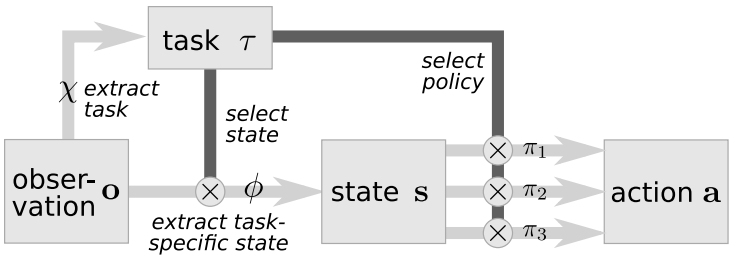


Figure 5.17: Overview of the gated network for state representation learning for multiple tasks.

a given time or about the number of tasks involved. This is an important requirement for robotic life-long learning, where robots should be able to determine autonomously if a task requires a separate state representation (grasping a pen is different from opening a door) or not (grasping a red or brown cup can be achieved with the same state representation). The representations learned with MT-LRP enable the use of standard reinforcement learning methods to compute effective policies from few data.

MT-LRP is implemented as two neural networks, coupled by a *gating* mechanism (Sigaud et al., 2015) as illustrated in Figure 5.17. The first network, χ, detects which task is being executed and selects the corresponding state representation. The second network, ϕ, learns task-specific state representations, as described above. This gated network architecture is similar to the one proposed by Droniou et al. (2015). Their network simultaneously learns discrete classes jointly with continuous class variations (called *submanifolds*) in an unsupervised way. In our approach, we learn discrete tasks rather than discrete classes; we learn task-specific state representations rather than class-specific submanifolds. We train the two coupled networks simultaneously optimizing consistency with *robotic priors* as described above, but add an additional prior of *task consistency*. Both networks learn from raw sensor data, without supervision and solely based on the robot's experiences.

We show in simulation experiments that MT-LRP is able to learn multiple state representations and task detectors from raw observations and that these representations allow to learn better policies from fewer data when compared with prior state representation learning methods. We also analyze the contribution to this result of each the method's individual components.

5.6.1 Multi-Task Learning with Robotic Priors (MT-LRP)

Multi-Task State Representations: Consider a scenario in which an agent is learning multiple distinct tasks. For each task $\tau \in \{1, \ldots, T\}$, the agent now requires a task-specific policy $\pi_\tau : S_\tau \rightarrow A$. We approach the problem by learning a task-specific *state representation* $\phi_\tau : O \rightarrow S_\tau$ for each policy, and a *task detector* χ which determines the task, given the current observation. We will consider a probabilistic task-detector $\chi : O \rightarrow [0, 1]^T$ that assigns a probability to each task being active.

In order to solve the full multi-task RL problem, we must learn χ, $\{\phi_\tau\}_{\tau\in\{1,\dots,T\}}$ and $\{\pi_\tau\}_{\tau\in\{1,\dots,T\}}$. We propose to address this problem by MT-LRP, a method that jointly learns χ and $\{\phi_\tau\}_{\tau\in\{1,\dots,T\}}$ from raw observations, actions, and rewards. MT-LRP then uses the state representations $\{\phi_\tau\}$ to learn task-specific policies $\{\pi_\tau\}_{\tau\in\{1,\dots,T\}}$ (using standard RL methods), and switches between them using the task detector χ. To solve the joint learning problem, MT-LRP generalizes LRP in the following way: (i) we replace the linear observation-state-mapping from the original method with a *gated neural network*, where the gates act as task detectors that switch between different task-specific observation-state-mappings; (ii) we extend the list of robotic priors by the prior of *task coherence*, which allows us to train multiple task-specific state representations without any specification (or labels) of tasks and states.

Gated Neural Network Architecture: We use a gated neural network architecture as shown schematically in Fig. 5.17. The key idea is that both the task detector χ as well as the state representation ϕ are computed from raw inputs. However, the output of the task detector *gates* the output of the state representation. Effectively, this means the output of $\chi(\mathbf{o})$ decides which task-specific state representation ϕ_τ is passed further to the policy, which is also gated by the output of $\chi(\mathbf{o})$.

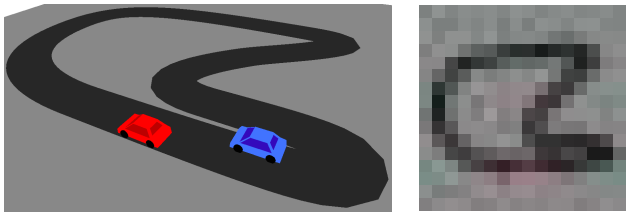
Formally, $\chi(\mathbf{o}) = \sigma(\chi_{\text{pre}}(\mathbf{o}))$ is composed of a function χ_{pre} with T-dimensional output and a *softmax* σ that ensures that χ computes a proper probability distribution over tasks. The probabilities are then used to gate ϕ. To do this, we decompose ϕ into a *pre-gating function* ϕ_{pre} that extracts features shared across all tasks (i.e. "multi-task" in the sense of Caruana (1997)), and a $T \times M \times N$ *gating tensor* $\mathbf{G}$ that encodes the T (linear) observation-state mappings ($M = \dim(\mathbf{s})$ and N is the output dimension of ϕ_{pre}). The value of the state's i-th dimension $\mathbf{s}_i$ computes as the expectation of the dot product of gating tensor and $\phi_{\text{pre}}(\mathbf{o})$ over the task probabilities $\chi(\mathbf{o})$:

$$s_i = \phi_i(\mathbf{o}) = \sum_{k=1}^{T} \chi_k(\mathbf{o}) \; \langle \mathbf{G}_{k,i,:}, \phi_{\text{pre}}(\mathbf{o}) \rangle. \tag{5.1}$$

Learning Objective: To train the network, we extend the robotic prior loss L_{RP} by a *task-coherence prior* L_τ:

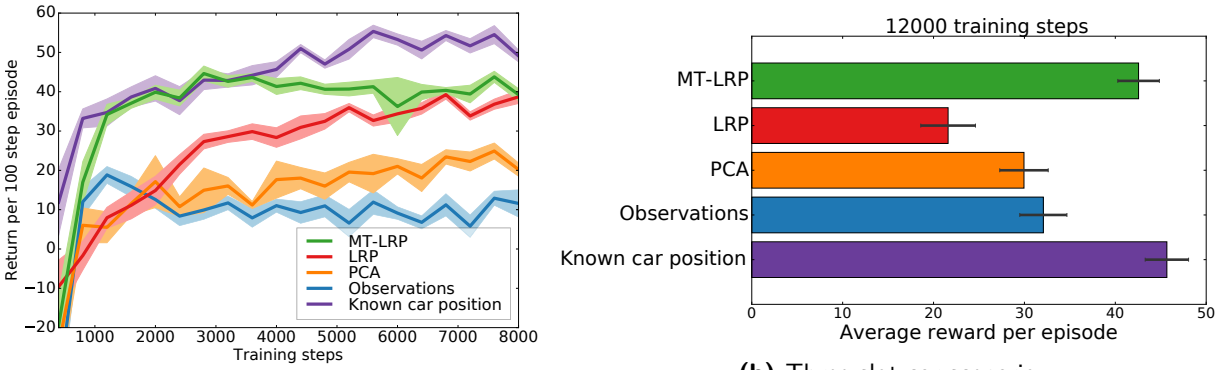
$$L = L_{\text{RP}}(D, \phi) + \omega_\tau L_\tau(D, \chi), \tag{5.2}$$

where ω_τ is a scalar weight balancing the influence of the additional loss term. Task coherence is the assumption that a task only changes between training episodes, not within the same episode. It does not presuppose any knowledge about the number of tasks or the task presented in an episode, but it exploits the fact that task switching



(a) Slot car racing **(b)** Observation

Figure 5.18: Two tasks. The robot is controlling either the red or the blue car.



(a) Two slot car scenario **(b)** Three slot car scenario

Figure 5.19: Reinforcement learning performance (mean and standard error) for different state representations for the slot car task two or three cars (static visual cue).

weakly correlates with training episodes. It applies directly to the output of the task detector, $\chi(\mathbf{o})$, and consists of two terms:

$$
\begin{aligned}
L_\tau^{\text{con+sep}} &= L_\tau^{\text{con}} + L_\tau^{\text{sep}} \\
&= \mathbf{E}\Big[H(\chi(\mathbf{o}_{t_1}), \chi(\mathbf{o}_{t_2})) \;\Big|\; \text{episode}_{t_1} = \text{episode}_{t_2}\Big] && (5.3) \\
&\quad + \mathbf{E}\Big[e^{-H(\chi(\mathbf{o}_{t_1}), \chi(\mathbf{o}_{t_2}))} \;\Big|\; \text{episode}_{t_1} \neq \text{episode}_{t_2}\Big], && (5.4)
\end{aligned}
$$

where H denotes the cross-entropy $H(p, q) = -\sum_x p(x) \log q(x)$. The first term L_τ^{con} enforces *task consistency* during an episode. We use it to penalize χ if it assigns different task distributions to inputs $\mathbf{o}_{t_1}$, $\mathbf{o}_{t_2}$ that belong to the same episode. The second term L_τ^{sep} expresses *task separation* and encourages χ to assign tasks to different episodes. This loss is complementary to task consistency, as it penalizes χ if it assigns similar task distributions to $\mathbf{o}_{t_1}$, $\mathbf{o}_{t_2}$ from different episodes.

5.6.2 Experiments and Results

We evaluate MT-LRP in the *multi-task slot-car racing* scenario, where the agent controls one of multiple cars (Figure 5.18), with the goal of traversing the circuit as fast as possible

without leaving the track due to speeding in curves. However, the agent does not know a priori which car it controls, and only receives the raw visual signal as input. Additionally, uncontrolled cars driving at random velocity, act as visual distractors. We turn this scenario into a multi-task problem in which the agent must learn to control *each* car, where controlling the different cars corresponds to separate tasks. To indicate the car that the agent controls, we indicate a picture of the car in the upper left corner.

We will now present the main results of our experiments: (i) we show that MT-LRP enables the agent to extract better representations for RL; (ii) we provide insight in how the learner detects/learns task and state representations; (iii) we show the contribution of each of the task-coherence loss terms.

Result: Extracts Better State Representations

Figure 5.19a shows the learning curves for RL based on state representations learned by the different methods in the two-slot-car scenario (*static visual cue* condition, see supplementary material). No method reaches the performance of the upper baseline, mainly due to aliasing errors resulting from the low image resolution. MT-LRP gets very close to the performance of the upper baseline, especially for very low amounts of training data ($d < 2500$), whereas LRP does not even attain this level of performance for the full training set $d = 8000$ as LRP can only learn to extract the position of all cars, not the relevant one. The gap between MT-LRP and LRP increases even more if we add another car (Figure 5.19b).

Result: Detects All Tasks

To gain more insight into what is learned, we analyze the state representations extracted by MT-LRP in Figure 5.20. Each point in the figure corresponds to one observation, markers indicate the task and colors the most active gate unit. We see that the first gate unit (blue) is always active for task 1 (circle), and the second gate unit for task 2, and that the states reflect the circular structure of the slot car racing track. We thus conclude that MT-LRP has learned to identify the tasks and to represent the position of each car on the track.

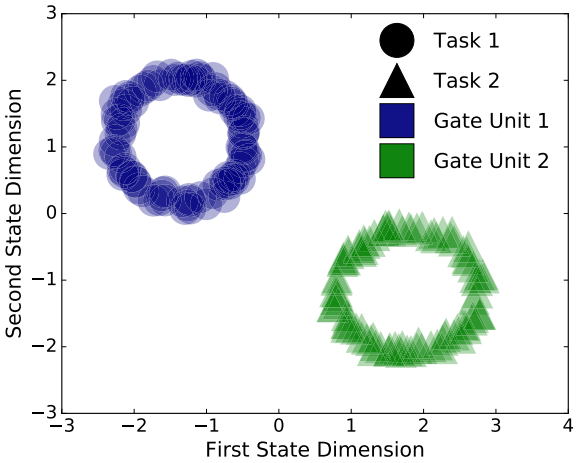


Figure 5.20: Multi-task state representation. Markers denote task and colors denote gate units (which coincide perfectly).

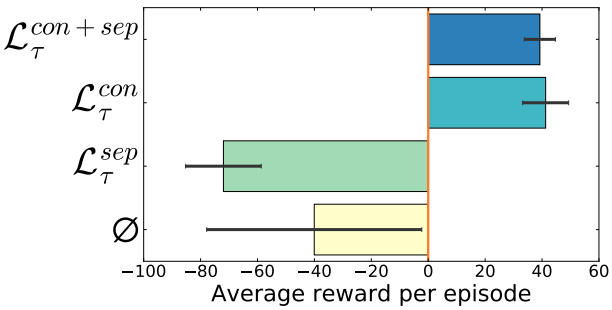


Figure 5.21: Performance after 8000 training steps with different learning objectives.

Result: Task-Consistency Prior is Necessary

To understand the influence of the different task-coherence prior variants, we compared their performance in Figure 5.21. We see that relying solely on the robotic priors gives poor results, mainly because the gate units are not used properly: more than one gate unit is activated per task (χ has high entropy). Adding the task-separation prior forces the network to use as many gates as possible (five in our case), leading to bad state representations. Interestingly, using task consistency only gives roughly the same result as using task consistency and task separation. This indicates that the robotics prior loss is sufficient to encourage the learner to separate different tasks: however, the task-consistency loss is required to guide the learner to identify the different tasks.

5.7 Extension 2: Adding Position-Velocity Structure to the State

Position and velocity are fundamental components of state representations in robotics. In this extension, we investigate how robots can learn position-velocity representations without supervision. Towards this end, we split the state representation into a velocity state and a position state and incorporate robotic priors about position and velocity in the form of model constraints and learning objectives.

Our extension to position-velocity states is inspired by work on physics-based priors in model-based reinforcement learning by Scholz et al. (2014), which proposed to learn a physically plausible dynamics model given a position-velocity representation. Here, we turn their approach around and ask: How could we learn a position-velocity representation from sensory input without specifying which positions and velocities are relevant for the

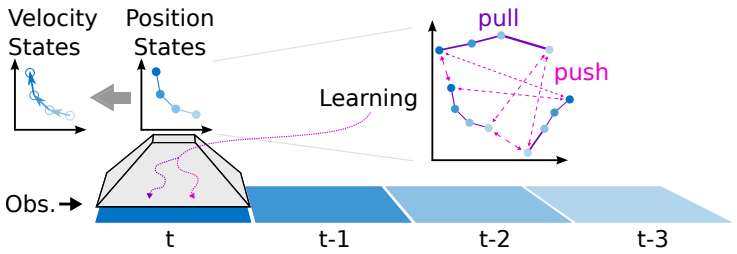


Figure 5.22: PVEs encode an observation into a low-dimensional position state. From a sequence of such position states, they estimate velocities. PVEs learn the encoding by optimizing consistency of positions and velocities with robotic priors.

task?

The answer we are proposing, the position-velocity encoder, works by incorporating prior knowledge in two different ways, which fit Mitchell's categorization of *inductive biases* into restriction biases and preference biases (Mitchell, 1997, p. 64). *Restriction biases* restrict the hypothesis space that is considered during learning, such as our constraint in the PVE model to estimate velocities from finite differences in position (rather than learning to extract velocity information from a sequence of observations). *Preference biases* express preferences for certain hypothesis, such as our loss functions for training PVEs, which measure inconsistency with robotic priors.

Our method, the *position-velocity encoder* (PVE), implements a hard model constraint by estimating velocity states from finite differences in position states. This constraint fixes the relation between these two parts of the state representation. Additionally, PVEs include soft objectives that measure consistency with robotic priors. These objectives are optimized during learning and shape which information is encoded and how multiple state samples relate to each other. Both ingredients work together to learn an encoding into a structured state representation that includes *position states*, which describe information from a single observation, and *velocity states*, which describe how this information changes over time.

Figure 5.22 shows the position encoder that maps observations (blue rectangles) to position states (blue dots). The velocity state—the time derivative of the position state—is approximated from finite differences in position. This structured state space allows us to formulate new robotic priors, specifically for positions and velocities, in the form of learning objectives.

In our experiments, we apply position-velocity encoders to simulated control tasks from pixels. We show that PVEs are able to discover the topology and the dimensionality of the task, that they can learn equivalent representations from different camera perspectives, that they capture information about the true positions and velocities of physical objects in the scene, and that reinforcement learning based on the learned position-velocity state can produce precise control.

5.7.1 Position-Velocity Encoders (PVEs)

Position-Velocity Encoders (PVEs) learn to map raw observations into a structured state space that consists of a position part and a velocity part by combining two key ideas:

1. PVEs encode the current observation into a *position state* and estimate a *velocity state* from finite differences in position (more details in Sec. 5.7.1).

2. PVEs are trained by optimizing consistency with robotic priors about positions, velocities, and accelerations (more details in Secs. 5.7.1 & 5.7.1).

Model

The PVE model consists of a convolutional network and a numerical velocity estimation. The convolutional network ϕ encodes a visual observation $\mathbf{o}_t$ into a low-dimensional position-state $\mathbf{s}_t^{(p)}$, where superscript (p) stands for *position*.

$$\mathbf{s}_t^{(p)} = \phi(\mathbf{o}_t).$$

From the difference of the last two position states $\mathbf{s}_t^{(p)}$ and $\mathbf{s}_{t-1}^{(p)}$, the model estimates the velocity state $\mathbf{s}_t^{(v)}$:

$$\mathbf{s}_t^{(v)} = \alpha(\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}),$$

where α is a hyperparameter that subsumes $\frac{1}{\text{timestep}}$ and scales velocity states. It is important that velocity states have the right scale relative to position states in order to create a sensible metric in the combined state $\mathbf{s}_t$, which we construct by stacking the position state and the velocity state.

$$\mathbf{s}_t = \begin{bmatrix} \mathbf{s}_t^{(p)} \\ \mathbf{s}_t^{(v)} \end{bmatrix}.$$

We can also use finite differences to estimate acceleration (or jerk, jounce, etc.). We do not include these derivatives in the state because we assume that the robot controls accelerations by its actions. But we do use the acceleration state in some loss functions. We compute the acceleration state $\mathbf{s}_t^{(a)}$ in the same way as the velocity state but we omit the scaling since we will not use accelerations in the combined state space:

$$\mathbf{s}_t^{(a)} = \mathbf{s}_t^{(v)} - \mathbf{s}_{t-1}^{(v)}.$$

Robotic Priors and Learning Objectives

The encoder ϕ is trained by making the combined state space consistent with a set of robotic priors, which we will describe in this section. These priors use the structured state space and are specific to positions, velocities, and accelerations. Consistency with these priors is defined in the form of loss functions that are minimized during learning. The following list of robotic priors should be understood as an exploration into this matter, not as a final answer.

Variation: *Positions of relevant things vary.* As the robot explores its task and manipulates its environment, the positions of task-relevant objects (including itself) will vary—otherwise there is not much that the robot could learn. If we assume that positions of relevant objects vary in the robot's experience, the internal representation of such positions must also vary; random pairs of position states should not be similar. Therefore, we optimize consistency with the variation prior by minimizing the expected similarity between random pairs of position states,

$$L_{\text{variation}} = \mathbf{E}\Big[e^{-\|\mathbf{s}_a^{(p)} - \mathbf{s}_b^{(p)}\|}\Big],$$

where we use $e^{-\text{distance}}$ as a similarity measure that is 1 if the distance is 0 and that goes to 0 with increasing distance between the position states, which is exactly what we want.

Slowness: *Positions change slowly.* Physical objects do not teleport; they do not change their position arbitrarily from one second to the next. To make the internal position state consistent with the slowness prior, we minimize the expected squared distance between consecutive position states,

$$L_{\text{slowness}} = \mathbf{E}\Big[\|\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}\|^2\Big].$$

Since this change in position is directly connected to the rate of position change (or velocity), we can also write down the same loss using the velocity state:

$$L_{\text{slowness}} = \mathbf{E}\left[\left\|\frac{\mathbf{s}_t^{(v)}}{\alpha}\right\|^2\right],$$

where α is the scaling hyperparameter defined earlier. This reformulation hints at a different interpretation of slowness, which is simply: *velocities are low.*[*]

[*]Note that defining the slowness prior to mean *velocities are low* translates to the loss function $L_{\text{slowness}} = \mathbf{E}[(\mathbf{s}_t^{(v)})^2] = \mathbf{E}[(\alpha(\mathbf{s}_t^{(p)} - \mathbf{s}_{t-1}^{(p)}))^2]$, which depends on the scaling parameter α. We use the other formulation to make this loss independent of α because we want to change α during training without affecting this loss (see Sec 5.7.1 for more details).

Inertia: *Velocities change slowly.* Since physical objects have inertia, they resist changes to their velocity (both in direction or magnitude). If we assume limited forces to overcome this resistance, velocities should only change by small amounts. Note how the inertia prior corresponds to the slowness prior applied to velocities.

$$L_{\text{inertia}} = \mathbf{E}\Big[\|\mathbf{s}_t^{(v)} - \mathbf{s}_{t-1}^{(v)}\|^2\Big] = \mathbf{E}\Big[\|\mathbf{s}_t^{(a)}\|^2\Big].$$

This formulation of the inertia prior focuses on large velocity changes due to the square in the loss function. Alternatively, we can define the loss function based on *absolute* changes.

$$L_{\text{inertia (abs)}} = \mathbf{E}\Big[\|\mathbf{s}_t^{(a)}\|\Big].$$

Small changes in velocity have a higher weight in the second loss compared to the first loss. We found that combining both losses leads to better results than using either one of them.

Conservation: *Velocity magnitudes change slowly.* This prior derives from the *law of conservation of energy*, which states that the total energy in a closed system remains constant. As the robot applies forces to the environment, we do not have a closed system. Additionally, we cannot estimate, e.g. kinetic energy without knowing the masses of objects, let alone potential energy stored in springs etc. Still, we want to enforce the same idea of keeping the absolute amount of energy, or in our case "movement" similar in consecutive time steps.

$$L_{\text{conservation}} = \mathbf{E}\Big[\big(\|\mathbf{s}_t^{(v)}\| - \|\mathbf{s}_{t-1}^{(v)}\|\big)^2\Big].$$

Controllability: *Controllable things are relevant.* The objects that can be controlled by the robot are likely relevant for its task. If the robot acts by applying forces, controllable things could be those whose accelerations correlate with the actions of the robot. Accordingly, we can define a loss function per action dimension i to optimize covariance between action dimension i and accelerations in a state dimension i.

$$\begin{aligned} L_{\text{controllability (i)}} &= e^{-\text{Cov}(\mathbf{a}_{t,i}, \mathbf{s}_{t+1,i}^{(a)})} \\ &= e^{-\mathbf{E}\Big[\big(a_{t,i} - \mathbf{E}[a_{t,i}]\big)\big(s_{t+1,i}^{(a)} - \mathbf{E}[s_{t+1,i}^{(a)}]\big)\Big]}. \end{aligned}$$

Note that we used this loss in only one of the tasks—ball in cup—because the above priors were insufficient. The results for this task are still preliminary. A complete solution of this task and a deeper investigation into other formulations of controllability are part of future work.

Training Procedure

We train PVEs by minimizing a weighted sum of the loss functions described above using gradient descent. This section explains the training procedure in detail.

Data Gathering: First, the robot gathers data by exploring its environment. Since we are using a reinforcement learning setting, the data consist of sequences of observations, actions, and rewards. Most of the presented loss functions only use observations, the controllability loss also uses actions, but none of our current losses uses the reward signal.

Loss Computation: We iterate through the collected data in mini batches, which consist of a small set of short sequences. For each mini-batch, we compute the loss functions by replacing expectations with statistical averages.*

Loss Combination: We combine these losses in a weighted sum. Finding the right weights is important because they balance how much each prior is enforced during learning. We determined these weights empirically by adjusting them until the gradients in the encoder parameters had similar magnitudes for all priors. Future work should try to automate this process of weight tuning, potentially by applying the same heuristic in an automated fashion.

Parameter Updates: For each mini-batch, we compute the gradient† of the combined loss with respect to the encoder parameters using symbolic auto-differentiation (Abadi et al., 2015) and perform an update using the Adam optimizer (Kingma and Ba, 2014). We iterate this process until convergence.

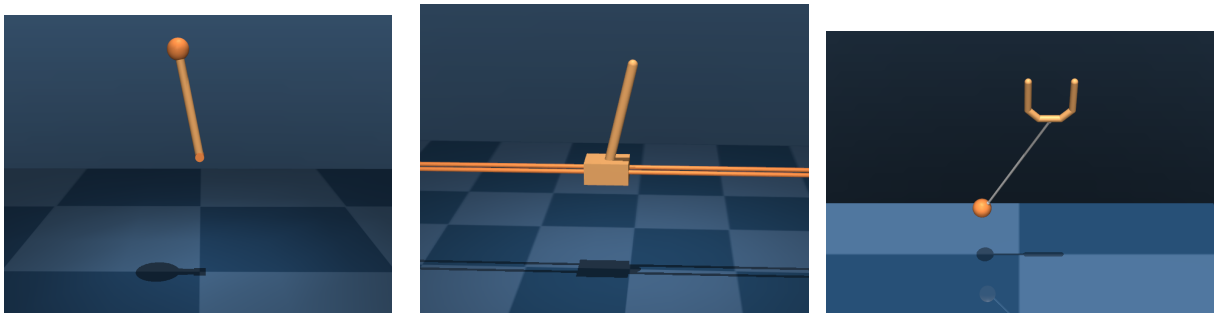
Velocity Scaling Curriculum: While training PVEs, we follow a curriculum that in the beginning focuses on positions and only later also takes velocities into account. This curriculum is implemented by changing the velocity scaling parameter α. In the first phase, we train with $\alpha = 0$ until convergence. In the second phase, we increase α linearly from 0 to its final value and train until convergence again. In phase one, only the first two priors, variation and slowness, are active. Surprisingly, these two are powerful antagonists that can unfold the topology of the position-state space. The second phase mainly smooths the state space such that velocities can be accurately estimated from finite differences.

*For the variation loss, we sample all pairs of experiences with the same time step in different sequences of the mini batch. For all other losses we consider all samples in the mini batch.

†Some of the gradients can only be computed after adding small Gaussian noise to the encoded states.

Table 5.1: Loss weights per task (same for a and b).

	Task a	Task b	Task c
Variation	1.0	1.0	1.0
Slowness	1.0	1.0	1.0
Inertia	0.1	0.1	0.001
Inertia (abs)	0.1	0.1	0.02
Conservation	0.2	0.2	0.005
Controllability (i=1,2)	0.0	0.0	0.5



(a) Inverted pendulum **(b)** Cart-pole **(c)** Ball in cup

Figure 5.23: Three control tasks from pixel input

Hyperparameters: We used the following hyperparameters in our experiments. The convolutional network had three convolutional layers with 16, 32, and 64 channels, kernel size 5x5, and stride 2, followed by three fully connected layers of sizes 128, 128, and 5 (for a 5-dimensional position state). Every layer except the last one was followed by a ReLu nonlinearity (Nair and Hinton, 2010). The mini-batch size was 32 sequences of 10 steps each. The maximum velocity scaling α was 10. The weights for the different losses are shown in Table 5.1.

5.7.2 Experiments and Results

We applied PVEs to a series of simulated control tasks from raw pixel input (see Fig. 5.23). All tasks use the MuJoCo simulator (Todorov et al., 2012). For each task, we collected a batch of training data that consists of 1000 short trajectories of 20 steps by randomly sampling start configurations with different positions and velocities and applying a random policy.

Tasks

Inverted Pendulum: The inverted pendulum is a stick that is fixed to a rotary joint at one end. The goal is to swing it up and balance it upright by applying torques at the joint. However, the motor is not strong enough to pull the pendulum up all at once. Instead the pendulum must be swung back and forth to generate enough (but not too much) momentum.

Cart-Pole: The cart-pole task is an extension of the inverted pendulum task. Since the pole is attached to a cart with a passive joint, it can only be swung up by accelerating the car correctly, which requires precise control.

Ball in Cup: This task includes a cup and a ball attached to the bottom of the cup with a string. The goal is to move the cup in such a way that the ball lands in the cup. In our version of the task, cup and ball can only move in the plane.

Result: PVEs Learn Position-Velocity Representations

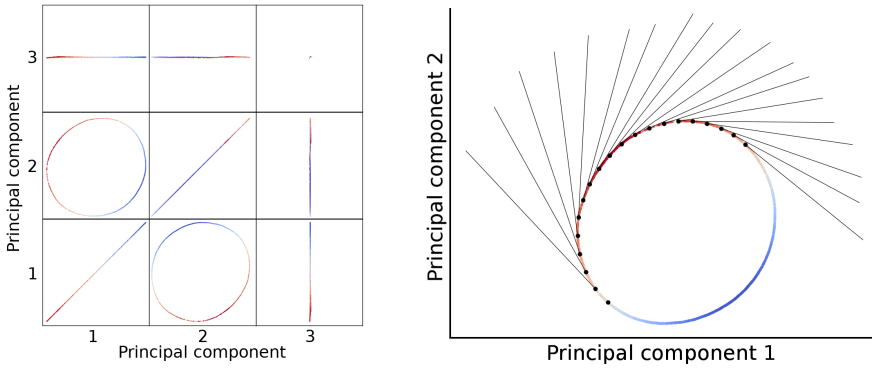
For each task, we will now look at the learned state representations. We visualize 5-dimensional position-states by projecting to their principal components.

Inverted Pendulum: The state representation learned by the PVE is shown in Figure 5.24a, where we can see the encoding of test observations into the position-state space. Each dot is the position encoding of a single image. The color denotes the amount of reward that was achieved in that instance.

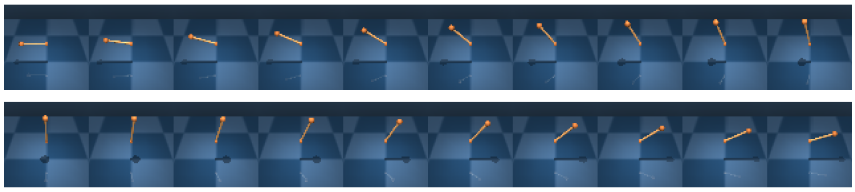
The plot shows a number of interesting results. First, observations that correspond to similar rewards are encoded close together in position space. Second, the position states form a circle, which makes sense because the inverted pendulum moves in a circle. Third, all principal components after the first two are close to zero. This means that the circular encoding lies on a plane in the five-dimensional space—the PVE discovered that the task is two dimensional.*

Next, we will look at the estimated velocities in the learned space. In Figure 5.24b, we overlayed encoded training data colored by reward with the encoding of a single sequence of observations shown in Figure 5.24c. The position states are marked with black dots and the velocity state vectors are drawn as lines. In the observation sequence, the pendulum swings from the left side to the top and then to the right side. Similarly, the encoded positions move from a medium reward region via the high-reward region (red color) to the

*Even though the task only has one positional degree of freedom (the angle of the pendulum), we need at least two dimensions if we want a Euclidean metric to make sense in this space, such that there are no jumps as from 360 to 0 degrees in an angular representation.



(a) Encoded position states **(b)** Overlayed state sequence



(c) Observation sequence

Figure 5.24: For the inverted pendulum, PVEs learn a circular position representation that allows accurate velocity estimation. Each dot in (a) and (b) is the encoding of a single observation. The color denotes the reward received with the observation (red = high, blue = low). Black dots in (b) show the encoding of the observation sequence in (c). Black lines show the estimated velocities. **Supplementary videos**: `http://youtu.be/ipGe7Lph0Lw` shows the learning process, `http://youtu.be/u0bQwz89h1I` demonstrates the learned PVE.

medium reward region on the other side. During this motion, the velocity estimations are tangential to the circle in the position space with minimal noise, which should be useful for controlling the pendulum (see video links in Fig. 5.24).

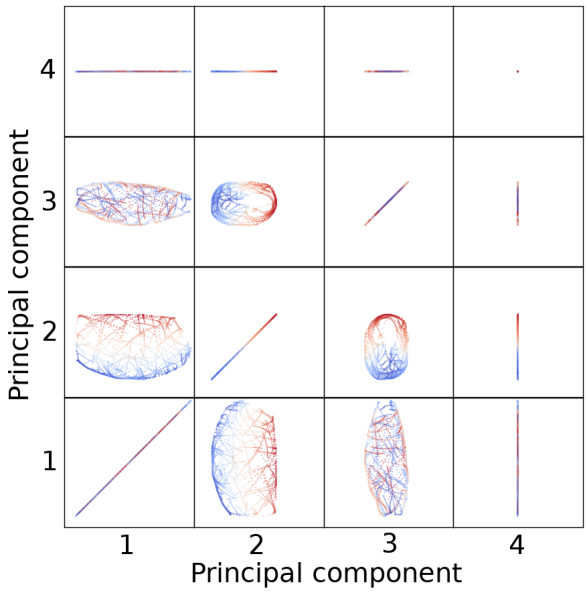
Cart-Pole: Here, we compare PVEs on two different observations: 1) using a moving camera that follows the cart by rotating sideways, 2) using a static camera that covers the entire region in which the cart moves. Figure 5.25 shows the learned position representations for both perspectives.

This experiment demonstrates how PVEs can learn equivalent internal representations (compare Figs. 5.25a and 5.25b) from observations that look very different (Figs. 5.25e, 5.25f). For both kinds of observations, the state samples form a tube, the length of which corresponds to the position of the cart, while the circular part represents the position of the pole. Here, the PVE uses three of the five dimensions and thereby discovers the three-dimensional nature of the given task.

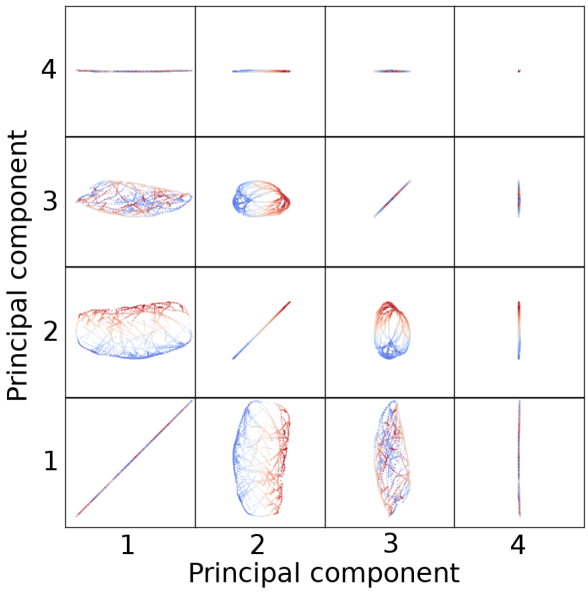
The observation sequence from the moving camera (Fig. 5.25e) shows the cart moving to the left while the pole falls down on the right side. The PVE represents this trajectory (Fig. 5.25c) by moving from the high-reward red region to the blue region, which reflects the movement of the pole, and to the right side, which corresponds to sideways movement of the cart. The observation sequence from the static camera (Fig. 5.25f) shows the pole swinging up while the cart moves to the right. Accordingly, the encoded trajectory (Fig. 5.25d) goes to the red region and to the right side (right and left are swapped between these two representations).

Ball in Cup: The results for this task are preliminary. The task is challenging due to the movement of the cup, which is inconsistent with some of our robotic priors. The cup is confined to a small region and controlled by the robot allowing rapid movements and changes of direction. The cup can be moved from one end of its position range to the other end in a few time steps. Therefore, the slowness prior does not hold here (unless we sampled observations at a higher frequency). Additionally, the robot can apply large forces on the cup, leading to large accelerations and jerky movements, which are again inconsistent with many of our priors on changes in velocity. As a result, PVEs struggle with encoding the cup, which we will quantify in the following section.

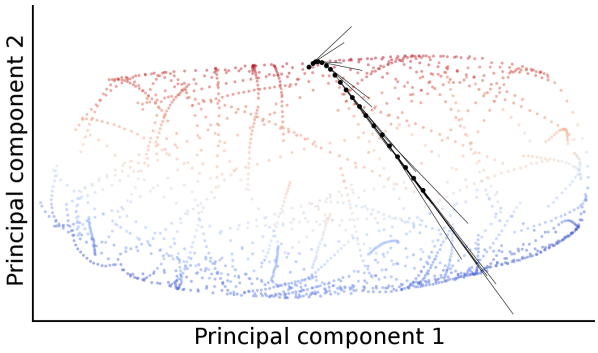
To approach this problem, we added the controllability prior, which enforces that things controlled by the robot are encoded into the state. This improved the resulting state representation (see Fig. 5.26). While the semantics of the state representation are not as clear as for the previous tasks, the representation uses four dimensions, which makes sense for two objects in a plane. Additionally, the goal states (ball in cup) are clearly separated from the other states. As we will see in the following section, the information about the cup is still very noisy, which is probably why reinforcement learning based on PVEs does



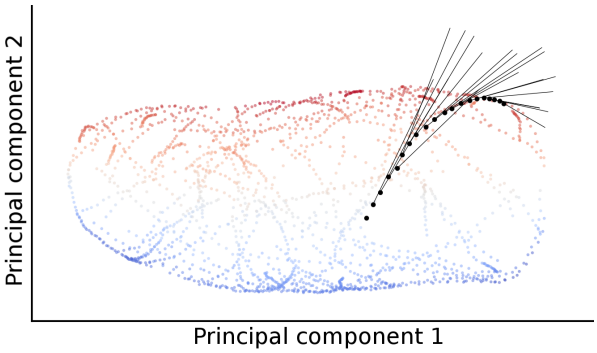
(a) Learned representation (moving camera)



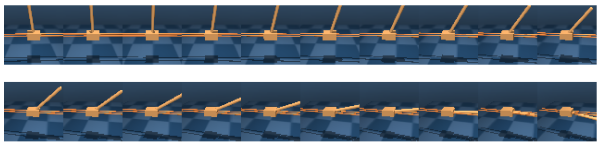
(b) Learned representation (static camera)



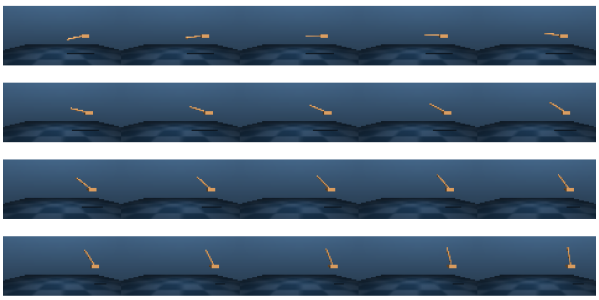
(c) Encoded sequence (moving camera)



(d) Encoded sequence (static camera)

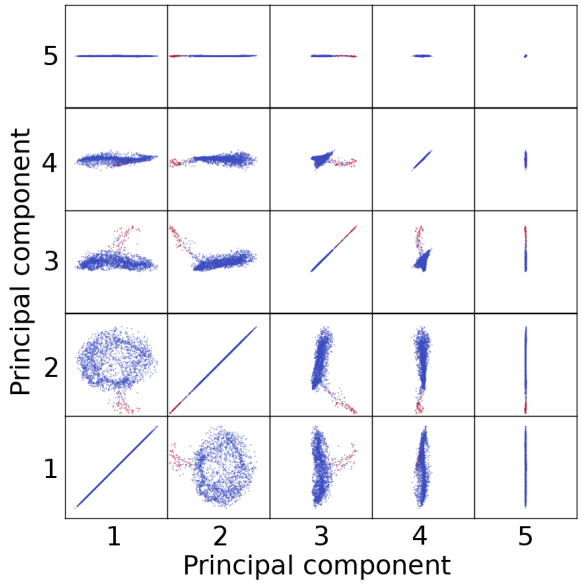


(e) Observation sequence (moving camera)

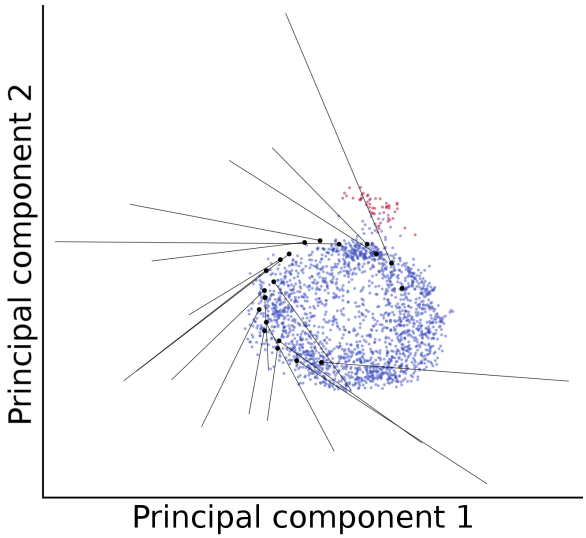


(f) Observation sequence (static camera)

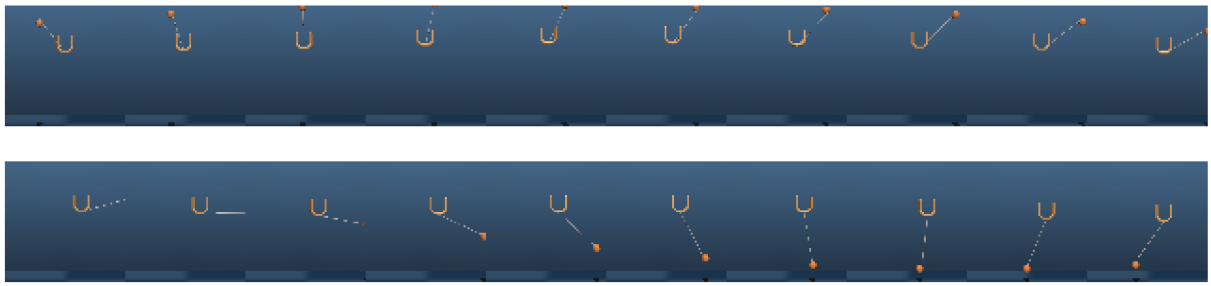
Figure 5.25: For cart-pole, PVEs learn equivalent state representations from different observations. **Supplementary videos**: learning process for the moving camera `http://youtu.be/RKlciWWuJfc` and static camera `http://youtu.be/MYxrA1Bw6MU`, learned PVE with the moving camera `http://youtu.be/67QZRsLNTAE`.



(a) Learned representation



(b) Encoded sequence



(c) Observation sequence

Figure 5.26: Learned position-velocity representation for ball in cup. **Supplementary videos**: `http://youtu.be/3fLaSL8d4TY` shows the learning process, `http://youtu.be/lIhEGv5kLFo` demonstrates the learned PVE.

not reach the same performance as in the other tasks. This result makes the ball in cup task a good candidate for a next step on extending PVEs by revising and adding robotic priors.

Result: Learned States Allow Regression to True Positions and Velocities

To measure the quality of the learned position-velocity representation, we performed regression from the learned postion-velocity state to true positions and velocities of relevant objects. Here, we trained a fully connected neural network with 3 hidden ReLu layers of 256 units each for 200 steps with Adam. We normalized the true positions and velocities and performed supervised learning from the learned position-velocity state to the true features minimizing mean squared error. After training on observations from 1000 times

20 steps, we tested with 100 times 20 test samples. The resulting test errors are shown in Table 5.2.

Table 5.2: Comparison of mean squared test errors.

Inverted pendulum	
$\cos(\theta_{\text{pole}})$	0.0003
$\sin(\theta_{\text{pole}})$	0.0002
$\dot{\theta}_{\text{pole}}$	0.0003

Ball in cup	
x_{cup}	0.0622
y_{cup}	0.0645
x_{ball}	0.0187
y_{ball}	0.0294
$\dot{x}_{\text{cup}}$	0.6654
$\dot{y}_{\text{cup}}$	0.6372
$\dot{x}_{\text{ball}}$	0.1535
$\dot{y}_{\text{ball}}$	0.2359

Cart-pole (different cameras)		
	moving	static
x_{cart}	0.0007	0.0015
$\cos(\theta_{\text{pole}})$	0.0013	0.0021
$\sin(\theta_{\text{pole}})$	0.0012	0.0033
$\dot{x}_{\text{cart}}$	0.0069	0.0198
$\dot{\theta}_{\text{pole}}$	0.0110	0.0264

When we compare these errors, we find that the errors are lowest for the pendulum task, which makes sense because the range of possible observations is so small in this task, that it is well covered by the training data. For the cart-pole the errors are still very low for position, but higher for the estimated velocities because noise in the position states is increased when computing velocities from finite differences. Also, the errors double when we go from the moving camera setting to the static camera setting. From this difference, we can predict that control should be easier in the first setting. Finally, for ball in cup, the errors are again much larger for the reasons discussed earlier. The estimation of the cup velocity is particularly challenging.

Note that we performed this regression test to measure how well these properties are encoded in the state. We do not use the state labels for training the representation and we do not use them for learning control. In the following section, we will measure the utility of the learned representation by reinforcement learning performance based on these representations.

Result: Learned Representations Enabling Reinforcement Learning

In this experiment, we learn control for these tasks with neural fitted Q-iteration (NFQ, Riedmiller, 2005) based on the encoding learned by PVEs. As a baseline, we use untrained PVEs with randomly initialized encodings in this preliminary work (we will thoroughly compare to other methods in future work). For the policy, we used a fully connected neural network with two hidden layers of 250 sigmoid units per layer. We trained it two times for 30 episodes after each training epoch. We rescaled rewards to be non-positive and omitted discounting. We repeated actions for multiple time steps (4 for the pendulum and cart-pole tasks, 6 for the ball in cup task).

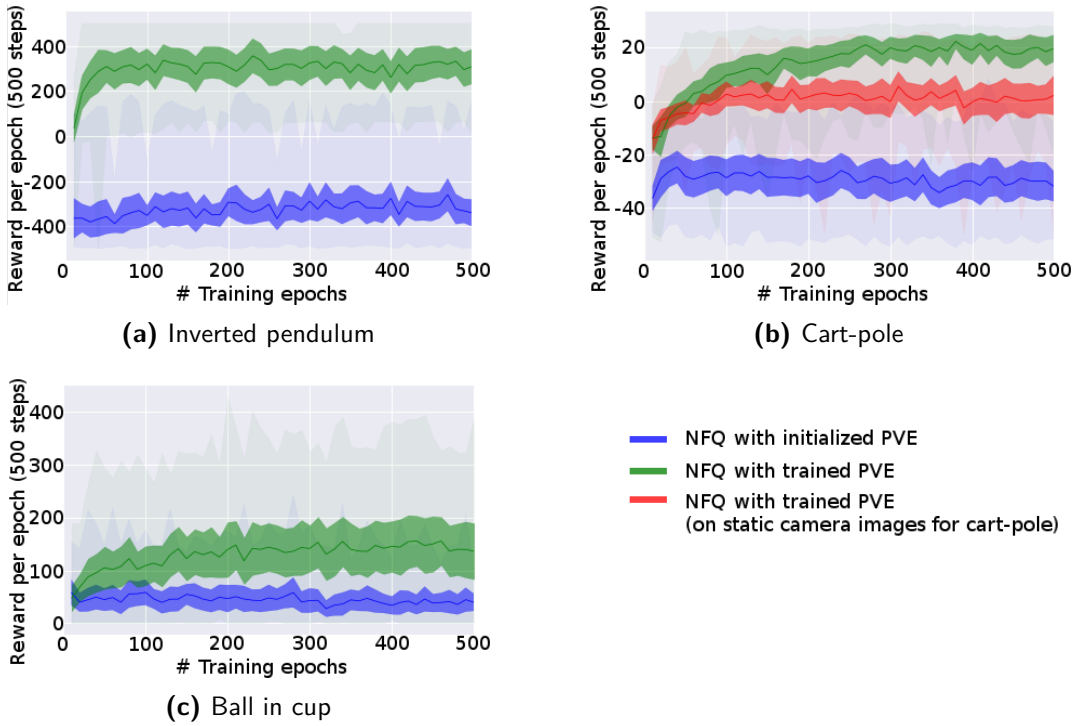


Figure 5.27: Reinforcement learning performance for different tasks based on state representations learned by PVEs. Lines show means of 50 trials, darker shading shows standard errors, lighter shading shows range from minimum to maximum values.

The resulting learning curves are shown in Figure 5.27. The blue curves show the baselines with random encodings, which do not allow learning any of the three tasks. The green and red curves show reinforcement learning based on PVEs that were trained on a batch of 1000 trajectories of 20 steps. For the inverted pendulum and for the cart-pole task, the green curves reach optimal performance after only 50 and 300 epochs. The red curve, which shows the performance based on the static camera perspective does not reach optimal performance, probably due to the more noisy state estimation discussed in the previous section. At this point, it is not clear whether this issue comes from the low resolution in the input or from the fact that the position of the pole and the cart are more strongly coupled in these observations which makes learning the state encoding more difficult. Lastly, for the ball in cup task, the learned control beats the baseline consistently and (as the light green maximum shading shows) more successful control using the learned representation is possible. But due to the noisy state estimation, this is not sufficient for solving the task consistently. Future work could start from here and investigate which priors are missing to solve this and more realistic robotic tasks.

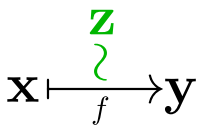


Figure 5.28: Side information $\mathbf{z}$ is related to function $f(\mathbf{x}) = \mathbf{y}$.

5.8 Relation to Learning with Side Information

In this section, we relate our work to *learning with side information* (Vapnik and Vashist, 2009; Chen et al., 2012) and thereby generalize the idea of formulating prior knowledge as learning objectives. The main idea of learning with side information is that we cannot only learn functions from input/output pairs, as is done in supervised learning, but also use other side information for learning if we know how this side information relates to the learnable function. Learning with robotic priors, for example, uses the actions, rewards, and time as side information to learn a mapping from observations to states but the same idea applies to other problems.

For many problems, there are additional data $\mathbf{z}$ available that are neither the input $\mathbf{x}$ nor the output $\mathbf{y}$ of function f but that carry valuable information about how f maps $\mathbf{x}$ to $\mathbf{y}$, as illustrated in Fig. 5.28. We refer to this kind of data as *side information* (Chen et al., 2012), also known as privileged information (Vapnik and Vashist, 2009). Examples for side information are

1. intermediate results computed by the true underlying f,
2. output of a related function (with input $\mathbf{x}$) that shares computations with f,
3. input of a related function (with output $\mathbf{y}$) that shares computations with f, or
4. relations between inputs $\mathbf{x}_i$ and $\mathbf{x}_j$ or between outputs $\mathbf{y}_i$ and $\mathbf{y}_j$.

Example: Suppose we want to estimate a function from the input/output samples: $3 \mapsto 14$, $5 \mapsto 30$, and $2 \mapsto 9$. From looking at these data, it is not immediately obvious what the true underlying function is. However, if we provide side information and the prior that they correspond to intermediate values that f computes, in this case $3 \mapsto \mathbf{9} \mapsto 14$, $5 \mapsto \mathbf{25} \mapsto 30$, and $2 \mapsto \mathbf{4} \mapsto 9$, we see that the function first squares its input and then adds five to the intermediate result, $f(\mathbf{x}) = \mathbf{x}^2 + 5$. Side information together with a prior about how they relate to f reveal the underlying function.

Incorporating priors about how $\mathbf{z}$ relates to f is what we call *learning with side information*. By enforcing consistency with these priors, we regularize learning which improves generalization. Note that we use side information *only during training, not for prediction*. There are a number of approaches in the literature that (often implicitly) follow the paradigm of learning with side information and demonstrate impressive results. Here,

we connect these lines of work, make the underlying paradigm explicit, and attribute the improved generalization to the use of priors enabled by side information.

Learning with side information provides an effective way to incorporate priors into the *learning objective* by exploiting data that are neither input nor output data of the target function f and are only required during training time. Note the difference to unsupervised and semi-supervised learning which only consider additional *input* data.

5.8.1 Definition of Learning with Side Information

In *learning with side information*, we estimate a function $f : \mathbf{x} \to \mathbf{y}$ and optionally an auxiliary function β by minimizing two objectives, the *main objective* L_f and the *side objective* $L_\mathbf{z}$:

$$\begin{aligned}&\operatorname{argmin}_f \; L_f(f \mid \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N),\\ &\operatorname{argmin}_{f,\beta} \; L_\mathbf{z}(f, \beta \mid \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, \{\mathbf{z}_j\}_{j=1}^M).\end{aligned}$$

To define L_f, we assume a supervised learning setting in which the goal is to estimate a function $f : \mathbf{x} \to \mathbf{y}$ from a set of N input/output pairs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$. Then, L_f corresponds to a standard supervised learning objective, e.g. mean-squared error for regression or hinge loss for classification.

The side objective is captured by $L_\mathbf{z}$, which depends on *side information* $\mathbf{z}$ and can include the auxiliary function β. The exact form of $L_\mathbf{z}$, $\mathbf{z}$, and β depends on the *pattern* applied (Sec. 5.8.2). For all patterns, $\mathbf{z}$ are data that are neither from the input space nor from the output space of f but carry valuable information about f, and are only needed for learning, not for prediction. Hence, the training data include M side information samples in addition to the N input/output pairs, $D = (\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, \{\mathbf{z}_j\}_{j=1}^M)$. Each of the side information samples relates to one or more input/output samples, commonly $M = N$ or $M = N^2$.

To exploit $\mathbf{z}$ for learning f, we formulate priors about how $\mathbf{z}$ relates to f in the side objective $L_\mathbf{z}$. To express $L_\mathbf{z}$, many patterns require f to be split into two functions, ϕ and ψ, where ϕ maps $\mathbf{x}$ to an intermediate representation $\mathbf{s}$, and ψ predicts $\mathbf{y}$ based on $\mathbf{s}$, hence $\mathbf{y} = f(\mathbf{x}) = \psi(\phi(\mathbf{x})) = \psi(\mathbf{s})$. This split exposes the representation $\mathbf{s}$ and facilitates the formulation of $L_\mathbf{z}$ by relating $\mathbf{s}$ and $\mathbf{z}$, possibly using β. Often it allows us to omit ψ and $\mathbf{y}$ from $L_\mathbf{z}$, i.e. to define $L_\mathbf{z}(\phi, \beta \mid \{\mathbf{x}\}, \{\mathbf{z}\})$. For example, in the multi-task pattern (Sec. 5.8.2) the intermediate representation $\mathbf{s}$ is shared amongst the main task of predicting $\mathbf{y}$ with function $\psi(\mathbf{s})$ and an auxiliary task of predicting $\mathbf{z}$ with $\beta(\mathbf{s})$. The auxiliary task regularizes the shared function ϕ and improves generalization for the main task.

Note that we intentionally kept this formalization narrow to improve readability. It is

straightforward to extend the ideas presented here to a reinforcement learning setting, to multiple types of side information, to multiple intermediate representations, and to more than one side objective.

Training Procedures

Since learning with side information requires us to optimize multiple learning objectives affected by different subsets of training data and functions, we need appropriate training procedures. We have identified three common training procedures that differ with respect to the order in which they (i) optimize the two objectives and (ii) modify the functions f and β:

Simultaneous learning jointly trains f and β by optimizing a weighted sum of the two learning objectives L_f and $L_{\mathbf{z}}$ (Weston et al., 2008). This procedure introduces the need to find a good weighting of the different learning objectives, which might be difficult if the gradients of the objectives differ by orders of magnitude and vary during learning.

If we split f into ϕ and ψ, as described in the previous section, we can choose among two additional procedures. In the *decoupled procedure*, we first optimize the side objective $L_{\mathbf{z}}(\phi, \beta \mid \{\mathbf{x}\}, \{\mathbf{z}\})$, while adapting ϕ and β to learn the intermediate representation $\mathbf{s}$. Then, we optimize the main objective $L_f(\phi, \psi \mid \{\mathbf{x}, \mathbf{y}\})$, while keeping ϕ (and β) fixed. This simple procedure is only applicable if the side objective provides enough guidance to learn a task-relevant representation $\mathbf{s}$, whereas the simultaneous procedure is also applicable for "weak" side objectives $L_{\mathbf{z}}$. To alleviate this problem, the *pre-train and finetune procedure* first applies the decoupled procedure, but then optimizes $L_f(\phi, \psi \mid \{\mathbf{x}, \mathbf{y}\})$ while adapting ϕ, too, in order to fine-tune $\mathbf{s}$ for the task. This strategy is popular in deep learning as unsupervised pre-training (Erhan et al., 2010) and can be applied analogously for learning with side information. For this procedure to have an effect, L_f must not be convex (otherwise, the pre-training step would be unlearned).

5.8.2 Patterns for Learning with Side Information

We will now present different approaches for learning with side information, which we have grouped into patterns. We describe for each pattern the general idea, the underlying prior, the side information $\mathbf{z}$, the side objective $L_{\mathbf{z}}$, and the auxiliary function β. We point to successful applications of each pattern (summarized in Appendix A.2.2) and visualize the patterns with schemas as in Fig. 5.29.

How to read the schemas: The schemas represent computation flow graphs where functions (drawn as arrows) connect variables (represented as nodes), both of which follow the definitions from Section 5.8.1. Predictions of variables are indicated by $\hat{\cdot}$. The target function is depicted in black. Additional elements that are only required at training time and can be omitted during prediction are shown in gray, except for side information and the corresponding learning objectives, which are highlighted in green. Learning objectives are visualized by connecting variables with $\sim$ to denote that the objective enforces similarity between these variables. The $=$ sign (see Fig. 5.34) indicates that a function is replicated (e.g. by weight sharing).

Note that these graphs are *not* probabilistic graphical models (PGMs). We provide PGMs as a complementary visualization of causal dependencies in Appendix A.2.1. In contrast, the computation flow graphs are advantageous for the purpose of this section since 1. they discriminate between variables and functions, 2. they expose the sequence of computation, 3. they visualize the learning objectives, and thus 4. are easily converted into neural networks, which are employed by most of the related works reviewed in this section.

Direct Pattern

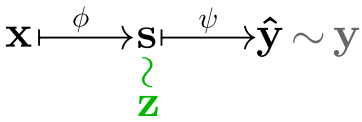


Figure 5.29: Direct pattern

The direct pattern leverages known, intermediate results of the computation performed by f. Given these intermediate results as side information $\mathbf{z}$, we can learn a function ϕ that transforms $\mathbf{x}$ into the representation $\mathbf{s}$ such that $\mathbf{s} \sim \mathbf{z}$, as shown in Fig. 5.29. No auxiliary function β is required. The pattern is only applicable if $\mathbf{z}$ makes it easier to predict $\mathbf{y}$, and if $\mathbf{x}$ contains enough information to predict $\mathbf{z}$. The example in Section 5.8 is an instance of this pattern.

To formalize this pattern, we use a suitable side objective $L_{\mathbf{z}} = L_{\text{direct}}(\phi \mid \{\mathbf{x}, \mathbf{z}\})$ from supervised learning that enforces the representation $\mathbf{s}$ to be equal to the side information $\mathbf{z}$.

Applications: Machine learning approaches in computational biology frequently use this pattern to combine understanding from biology research with learning. For example, in contact prediction, the goal is to predict which parts of a folded protein are close to each other based on the DNA sequence that describes the protein. Virtually all learning-based approaches to this problem first predict intermediate representations $\mathbf{s}$, such as secondary structures (local 3D structure categories), and then use $\mathbf{s}$ to predict contacts (Cheng and Baldi, 2007). The representation $\mathbf{s}$ can be reliably estimated which greatly facilitates learning ϕ.

Knowledge transfer (Vapnik and Izmailov, 2015) uses this pattern, but includes an additional step of extracting features $\beta(\mathbf{z})$ from the side information. Function ϕ is then learned by regression, such that $\mathbf{s} \sim \beta(\mathbf{z})$. They also suggest augmenting $\mathbf{s}$ with the original input $\mathbf{x}$. Similarly, Chen et al. (2012) suggest to reconstruct only highly predictive features of $\mathbf{z}$ using a modified version of AdaBoost.

Multi-Task Pattern

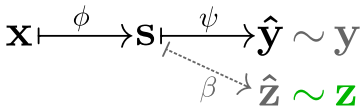


Figure 5.30: Multi-task pattern

This pattern applies when the side information $\mathbf{z}$ are outputs of a related function (with input $\mathbf{x}$) that shares computations with the function we want to estimate. As illustrated in Fig. 5.30, the pattern assumes that the target function $f = \psi \circ \phi$ and the related function $\beta \circ \phi$ share ϕ and therefore have the same intermediate representation $\mathbf{s} = \phi(\mathbf{x})$. By training the representation to predict both $\mathbf{y}$ using ψ, and $\mathbf{z}$ using the auxiliary learnable function $\beta : \mathbf{s} \to \mathbf{z}$, we incorporate the prior that *related tasks share intermediate representations.* This pattern corresponds to multi-task learning (Caruana, 1997), a type of transfer learning (Pan and Yang, 2010).

To apply the multi-task pattern, we can use any suitable learning objective from supervised learning $L_{\mathbf{z}} = L_{\text{multi-task}}(\phi, \beta \mid \{\mathbf{x}, \mathbf{z}\})$ in order to learn to predict $\mathbf{z}$ from $\mathbf{x}$.

Applications: Multi-task learning has been successfully applied in a wide variety of tasks (Caruana, 1997; Pan and Yang, 2010). Recently, Zhao and Itti (2015) proposed to use object pose information to improve object recognition in a convolutional deep neural network. Similarly, Levine et al. (2015) use image classification and pose prediction as side information to teach a robot remarkable vision-based manipulation skills, such as stacking lego blocks or screwing caps onto bottles.

Irrelevance Pattern: A special case of the multi-task patterns exploits knowledge about unrelated tasks, by enforcing the prediction of the side information to be orthogonal to the main task (Romera-Paredes et al., 2012). This idea is formalized by forcing ψ to be orthogonal to the auxiliary prediction function β (see Fig. 5.31), which allows to use knowledge about irrelevant distractors present in the input data. However, it is unclear how to efficiently formulate the orthogonality constraints between ψ and β for the non-linear case.

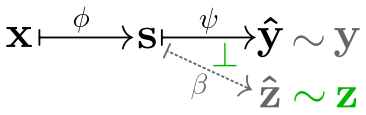


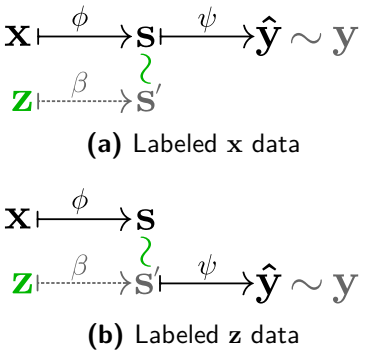
Figure 5.31: Exploiting irrelevant side information.

Multi-View Pattern

The multi-view pattern is complementary to the multi-task pattern, treating side information as input instead of output. It applies when $\mathbf{z}$ are inputs of a related function (with output $\mathbf{y}$) that share computations with f. This pattern corresponds to multi-view learning (Sun, 2013).

When we treat $\mathbf{z}$ as auxiliary input, we can use it in two different ways: explicitly by *correlating* it with the original input $\mathbf{x}$ (Fig. 5.32), or implicitly by *predicting* the target output (Fig. 5.33). In both cases, we learn functions $\phi : \mathbf{x} \mapsto \mathbf{s}$ and $\beta : \mathbf{z} \mapsto \mathbf{s}'$, such that $\mathbf{s} \sim \mathbf{s}'$.

The *multi-view (correlation) pattern* assumes that *correlated representations computed from related inputs are a useful intermediate representation for predicting the target output.* It can be formalized with a learning objective that enforces the correlation between $\phi(\mathbf{x})$ and $\beta(\mathbf{z})$, e.g. the mean squared error $L_{\mathbf{z}} = L_{\text{multi-view}}(\phi, \beta \mid \{\mathbf{x}, \mathbf{z}\}) = \sum_i ||\phi(\mathbf{x}_i) - \beta(\mathbf{z}_i)||^2$. If we apply the decoupled training procedure, i.e. only optimize the objective, we have to add constraints, e.g. unit variance, to $L_{\text{multi-view}}$ in order to avoid the trivial solution of having a constant intermediate representation. In case ϕ and β are linear, $L_{\text{multi-view}}$ with unit variance corresponds to Canonical Correlation



(a) Labeled $\mathbf{x}$ data

(b) Labeled $\mathbf{z}$ data

Figure 5.32: Multi-view (correlation) pattern

Analysis (CCA).

Applications: The pattern is often employed in multi-modal scenarios (Sun, 2013). Chen et al. (2014) show how to enhance object recognition from RGB-only images by leveraging depth data as side information during training. In computational neuroscience, the pattern is widely used to learn from multiple modalities (e.g., EEG and fMRI) or across subjects (Dähne et al., 2014). The pattern can also be applied for clustering (Feyereisl and Aickelin, 2012). The idea is to repeatedly cluster on both $\{\mathbf{x}_i\}$ and $\{\mathbf{z}_j\}$ and then return the clustering of $\mathbf{x}$ with the highest agreement with $\mathbf{z}$. In a recent article, Wang et al. (2015) suggest and compare deep architectures that combine multi-task and multi-view learning, and show that a deep canonically correlated auto-encoder gives superior results for visual, speech, and language learning.

The *multi-view prediction pattern* is based on the prior that *predicting the target output from related inputs requires similar intermediate representations.* It trains the functions $\phi : \mathbf{x} \mapsto \mathbf{s}$ and $\beta : \mathbf{z} \mapsto \mathbf{s}'$ such that both $\mathbf{s}$ and $\mathbf{s}'$ map to the target output using the same prediction function ψ, e.g. using weight sharing. Since $\mathbf{s}$ and $\mathbf{s}'$ are coupled to $\mathbf{y}$ via the main objective, we do not only regularize ϕ, but also ψ.

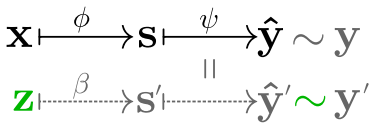


Figure 5.33: Multi-view prediction pattern

Despite their similarities, we are not aware of any systematic comparison of multi-view and multi-task learning. Neither have we found applications of the prediction pattern in the literature. Our experiments provide a first empirical comparison of these patterns (Sec. 5.6.2).

Pairwise Patterns

Pairwise patterns use side information $\mathbf{z}_{ij}$ that carry information about the relationship between samples i and j to shape the intermediate representation, e.g. the difference between their intermediate representations (Fig. 5.34, the = indicates weight sharing).

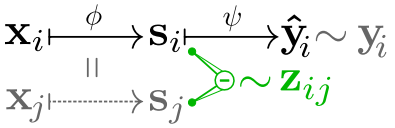


Figure 5.34: Pairwise pattern

Pairwise Similarity/Dissimilarity Pattern: If the side information gives information about similarity of samples with respect to the task, we can impose the prior that *samples that are similar (dissimilar) according to their side information should have similar (dissimilar) intermediate representations.* Such side information is often available as information about local neighborhoods of samples (Tenenbaum et al., 2000). Another powerful source of similarity information are time sequences, since temporally subsequent samples often have similar task-relevant properties, as exploited by slow feature analysis (SFA) and temporal coherence (Wiskott and Sejnowski, 2002; Weston et al., 2008). Additionally, an intelligent teacher can provide information about which samples are similar (Vapnik and Izmailov, 2015).

Similarity can be enforced with a squared loss on the distance between similar samples:

$$L_{\text{sim.}}(\phi \mid \{\mathbf{x}, \mathbf{z}\}) = \sum_{i,j} ||\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)||^2 \ \mathbb{1}(\mathbf{z}_{ij} = \text{sim.}),$$

where $\mathbb{1}$ denotes the indicator function. Solely using this objective might lead to trivial solutions where all samples are mapped to a constant. We can resolve this problem by imposing additional balancing constraints on $\mathbf{s}$ (Weston et al., 2008) or selectively push samples apart that are dissimilar according to the side information (or optionally according to the labels $\mathbf{y}$):

$$L_{\text{dis.}}(\phi \mid \{\mathbf{x}, \mathbf{z}\}) = \sum_{i,j} \sigma(||\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)||) \ \mathbb{1}(\mathbf{z}_{ij} = \text{dis.}),$$

where σ is a function that measures the proximity of dissimilar samples in representation $\mathbf{s}$. Candidates for σ are the margin-based $\sigma(d) = \max(0, m - d^2)$ for some pre-defined margin m (Hadsell et al., 2006), the exponential of the negative distance $\sigma(d) = e^{-d}$ (Jonschkowski and Brock, 2014), or the Gaussian function $\sigma(d) = e^{-d^2}$ (Jonschkowski and Brock, 2015). Another way to avoid trivial solutions is to impose an input-reconstruction objective, e.g. by using an auto-encoder (Watter et al., 2015).

Vapnik and Izmailov (2015) incorporate similarity information into support vector machines by replacing the free slack variables with a function of $\mathbf{z}$. This method incorporates the prior that slack variables should be similar for samples with similar side information.

Applications: This pattern has been shown to successfully guide the learner in identifying task-relevant properties of $\mathbf{x}$. Hadsell et al. (2006) show how to learn a lighting invariant pose representation of objects in the NORB dataset. Weston et al. (2008) show that regularizing a convolutional network with a temporal coherence objective outperforms pure supervised object classification in the COIL-100 dataset by 20% in terms of recognition accuracy.

Recent works show how to apply this pattern to reinforcement learning settings. Watter et al. (2015) exploit the time sequence to jointly learn a state representation and the world dynamics from raw observations for a variety of standard tasks, such as cart-pole balancing. Jonschkowski and Brock (2015) apply the pattern in a robot navigation task, and show how leveraging temporal and robot action information enable the robot to learn a state representation from raw observations, despite the presence of visual distractors.

Note that this pattern only preserves *local* similarities between samples. If the side information provides a global distance metric, Weston et al. (2008) propose to formulate side objectives for learning a distance-preserving mapping of $\mathbf{x}$ to $\mathbf{z}$, e.g. based on multi-dimensional scaling (Kruskal, 1964). Alternatively, the distance metric can be learned using side information (Fouad et al., 2013).

Pairwise Transformation Pattern: Instead of exploiting only binary similarity information between samples, the pairwise transformation pattern exploits continuous information about the relative transformations between samples, to make *the internal representation (or parts of it) consistent or equivariant with the known relative transformations.* Such side information is often available in robot and reinforcement learning settings.

Consistency with the transformations $\mathbf{z}$ can be enforced in different ways: (a) Hinton et al. (2011) require the transformation $\mathbf{z}$ to affect $\mathbf{s}$ in a known way, and suggest the *transforming autoencoder model* shown in Fig. 5.35a to learn such an $\mathbf{s}$. The idea is to learn to reconstruct the transformed input from the original input and the known transformation. (b) If the transformations in $\mathbf{s}$ are unknown, Jayaraman and Grauman (2015) suggest to learn these transformations as an auxiliary task using the pattern depicted in Fig. 5.35b. (c) We can also turn this approach around and try to predict the transformation based on the original and the transformed representation (Agrawal et al., 2015) as depicted in Fig. 5.35c. All three variants (a)-(c) enforce equivariance of $\mathbf{s}$ with respect to the relative transformations, and can be trained using supervised side objectives. (d) Instead of optimizing for equivariance, we can also enforce that the same transformation has the same effect, when applied to different samples (Fig. 5.35d). When transformations are discrete, we formalize this by penalizing the squared difference of the change in

internal representation after applying the same transformation:

$$L_{\text{transf.}}(\phi \mid \{\mathbf{x}, \mathbf{z}\}) = \sum_{i,j} ||\Delta\phi(\mathbf{x}_i) - \Delta\phi(\mathbf{x}_j)||^2 \mathbb{1}(\mathbf{z}_i = \mathbf{z}_j),$$

where Δ denotes the change caused by the transformation, i.e. $\Delta\phi(\mathbf{x}_i) = \phi(\mathbf{x}_{i+1}) - \phi(\mathbf{x}_i)$ for sequential data. This objective can be extended to continuous transformations by replacing the indicator function with a similarity function $\sigma(\mathbf{z}_i - \mathbf{z}_j)$ from Section 5.8.2. Variants of this pattern allow to enforce only locally consistent transformations, by multiplying $\sigma(\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j))$, or to enforce only consistent magnitudes of change by comparing norms $||\Delta\phi(\mathbf{x}_i)||$ (Jonschkowski and Brock, 2015).

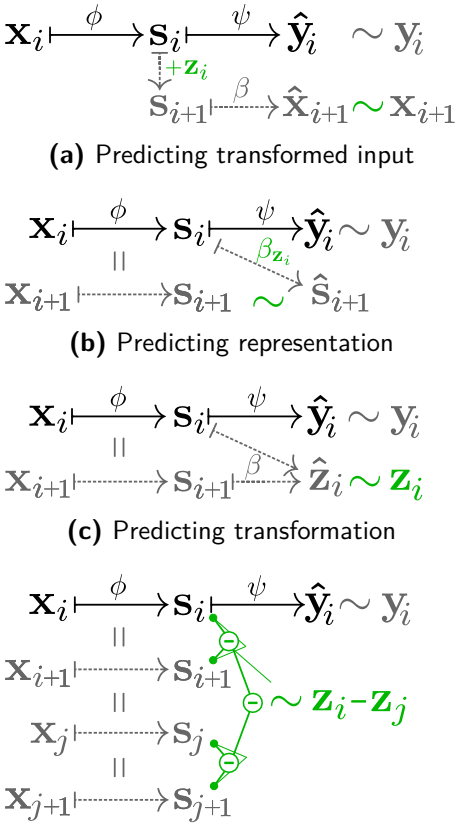


Figure 5.35: Pairwise transf. patterns

Applications: Many results in the literature demonstrate the usefulness of the pairwise transformation pattern. Agrawal et al. (2015) report that using relative pose information as side information can reduce the error rate on MNIST by half with respect to pure supervised learning. They also demonstrate the approach for scene recognition on the SUN dataset, and show that pre-training using limited of relative pose side information is almost as good class-based supervision. Jayaraman and Grauman (2015) demonstrate a recognition accuracy of $\approx 50\%$ on the KITTI dataset, outperforming pure supervised learning (41.81% accuracy) and SFA (47.04%). Interestingly, both works enforce learning a pose *equivariant* representation, although the classification task they address requires *invariance*. It is still unclear why equivariant representations help in such tasks (Lenc and Vedaldi, 2014).

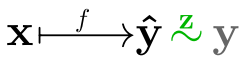


Figure 5.36: Label distances

Label Distance Pattern: The label distance pattern is a special case of the pairwise pattern, where the side information defines distances between labels, not samples (see Fig. 5.36). An instance of this pattern, often used in structured prediction, is hierarchical multi-class learning (Silla Jr. and Freitas, 2010), where a hierarchy is imposed on the labels to penalize misclassifications between samples with similar classes less severely.

5.9 Conclusion

5.9.1 Summary

We have presented a new approach to state representation learning in robotics: learning with robotic priors (LRP). The first key idea to this approach is to focus on state representation learning in the physical world instead of trying to solve the general problem of state representation learning in arbitrary (artificial) environments. Reducing the problem domain in this way allows us to use robotics-specific prior knowledge. We strongly believe that such prior knowledge will be vital to create versatile robots.

The second key idea is a specific way to use prior knowledge for state representation learning. We evaluate representations by how consistent they are with our priors about the world. We proposed five robotic priors—simplicity, temporal coherence, proportionality, causality, and repeatability—and showed how they can be turned into an objective for state representation learning.

Our experiments demonstrate that the inclusion of robotic priors can greatly improve learning performance and generalization. The magnitude of this effect is significant, in spite of the generality of the employed priors. These experiments have been reproduced and extended by Lesort et al. (2017).

We have also presented two extensions of the original approach. The first extension enables multi-task state representation learning, MT-LRP, where the agent simultaneously needs to cluster its experience into multiple tasks and to learn a state representation for each of them. This extension differs from the original approach by implementing the observation-state-mapping as a gated neural network and by adding the prior of task coherence, which is encoded in the learning objective. Our experiments confirmed that MT-LRP is effective at simultaneously identifying tasks and learning task-specific state representations. This capability is beneficial for scaling reinforcement learning to realistic scenarios that require dedicated skills for different tasks.

The second extension are position-velocity encoders (PVEs), which are able to learn state representations structured into a position and a velocity part without supervision and without requiring image reconstruction. PVEs combine two key ideas, 1) to con-

strain the model to estimate velocities in the correct way from positions and 2) to train the position encoder by optimizing consistency with robotic priors, which are specific to positions, velocities, and accelerations. We have shown how structuring the state space into positions and velocities creates new opportunities for formulating useful constraints and learning objectives. Our experiments show that this allows PVEs to learn effective state representations for continuous control tasks.

Finally, we have related the idea of incorporating prior knowledge in the form of loss functions to *learning with side information*, which provides a new perspective on machine learning and complements existing paradigms such as supervised learning, representation learning, and deep learning. We have connected learning with side information with various methods in the literature that use side information implicitly. This allowed us to systematically analyze these methods and extract patterns from them that show how they incorporate different priors about how side information relates to the target function. Since different priors coincide with different learning tasks, we hypothesized that the performance of these patterns will vary strongly depending on the applicability of the corresponding prior. Our experiments confirm this hypothesis and demonstrate that learning with side information can substantially improve generalization.

5.9.2 Future Directions: Structured State Representations

Interacting with the physical world is the essential definition of robotics and an important source of prior knowledge. This is what differentiates robots from other artificial agents that operate in games etc. Physical interaction provides a wealth of structure that, in this chapter, we only begun to explore.

In the initial work, we demonstrated the possibility and the advantage of defining such priors about physical interaction even in purely unstructured state representations—and this approach can certainly be improved by finding better priors.

But I expect a much larger leap when we add more structure into the state representation. The two extensions that I have presented go in this direction, structuring the state into multiple tasks or into different time derivatives (position, velocity, acceleration). But there is ample room for additional structure such as objects, hierarchies, other intentional agents, etc. By having such structure in the state space, we can formulate more powerful learning objectives that enforce physically consistent representations.

In addition to simplifying representation learning, structured state spaces will also facilitate learning forward models, because this learning problem, too, can make use of priors based on the state structure. Better models will enable model-based reinforcement learning, an approach that can potentially be much more data-efficient than the currently dominant model-free reinforcement learning.

6 Conclusion

6.1 Machine Learning Research

I would like to start this chapter with some general conclusions about machine learning research from the perspective of a roboticist. This section is inspired by Domingos (2012), who wrote an excellent paper on "a few useful things to know about machine learning", where he described important machine learning insights that do not usually appear in scientific publications. Here, I will state my own insights about machine learning research.

6.1.1 Learning Is a Means, Not an End

Learning is a means to create intelligence, not an end in itself. If we can solve a problem without learning and the solution is effective, robust, and general enough, then there is no reason to apply machine learning. In this thesis, I made the argument many times that we cannot learn *everything*, that we need to assume *something*. Successfully engineered solutions (or parts thereof) are good candidates for useful assumptions.

Thus, our first reaction to a solution of a robotics problem that does not rely on any machine learning should not be whether we could also learn it from data, but whether machine learning could improve this solution by making it more effective by replacing manual parameter tuning or more robust and general by adapting parameters to a given scenario. If the answer to these questions is "no", then there is no reason for learning. If it is "yes", we should not only consider replacing the engineered solution altogether but also consider complementing it with learning where learning is most useful.

6.1.2 Problems and Data Sets Are Not the Same

In machine learning research, we often equate data sets with scientific problems that we want to solve, as if data sets were somehow given to us. But instead of being the problem, data sets are an ingredient for solving a problem—just as learning algorithms are. Gathering data is an important part of machine learning research that is not discussed much in the literature but that often requires tremendous effort.

In order to learn object segmentation for the Amazon picking challenge (see Chapter 3), for example, we had to perform all of the following steps to gather a single labeled training sample: Select between one and five objects, enter their identifiers into the sample meta-data, arrange these objects in a shelf bin, make the robot move its arm-mounted camera in front of the bin, record an RGB-D image, transfer the image and meta-data from the robot to another computer, manually create segmentation masks for all objects using image editing software, and add the image, the object masks, and meta-data to the data set.

Due to this laborious process, we used only about six labeled images per object. We had actually generated more samples than that, but had to select a subset for each object that covers the views of an object in an unbiased way because some objects have very different appearances depending on which side faces the camera and too many samples of one view would make it more difficult to recognize the object from other views. Even with all this effort, the data set is not the same as the problem: object segmentation in the Amazon picking challenge.

During the challenge, many teams struggled with perception because the test conditions differed from their training set. The challenge venue had bright lights at the ceiling and some objects were slightly different from those the teams had received earlier. Apparently, they came in different variants depending on when and where they were ordered from Amazon. A number of teams modified the test conditions (within the rules of the challenge) to match the data set they had used for training their models. They added shades to block the bright lights and were allowed to use their own object variants. This anecdote really illustrates how brittle learned models can be when they are tested in conditions different from the training set.

Since we ultimately want to solve problems not data sets, we should acknowledge that training data are part of the solution, that gathering such data requires a lot of effort, which can crucially limit the scalability of machine learning, and that test data can only be a proxy for the real problem.

6.1.3 Reinforcement Learning Requires Instrumented Environments

Reinforcement learning can potentially remove the need for manual data collection by defining a setting for autonomous learning. In the reinforcement learning setting, the robot continuously perceives the state, performs actions, receives feedback in form of a reward signal, and adapts its behavior so as to maximize long-term reward. This setting captures the interactivity of learning, allows continued adaptation rather than having a distinct training and test phase, and most importantly does not require manual data collection or any other form of human intervention.

At second glance, however, reinforcement learning requires either *substantial* human intervention or highly instrumented environments, i.e. environments that contain sensors, computation, and mechanisms to enable learning. The first problem is that reinforcement learning methods often assume access to the ground truth state of the system, an unrealistic assumption that has motivated my work in Chapter 5.

But even if robots learn from raw observations instead of states, they might still rely on the true state to compute the reward. In my real world experiments in Section 5.5.7, where a mobile robot was rewarded whenever it entered the top right corner of a room, I had to equip the robot with a laser range finder and perform global localization with a particle filter to estimate whether the robot is in the goal region and should receive a reward. While the robot's learned behavior did not depend on this information, generating the reward signal, which is needed for learning the behavior, required the true state.

But the reward is not the only reason for why we need instrumented environments to enable reinforcement learning. The environment must also retain a state that allows to fullfil the task and enable the robot to explore safely and to recover from failures. Levine et al. (2016), for example, apply reinforcement learning to grasping and enable safe exploration by restricting the learnable policy and by using a soft gripper. They maintain the environment in a state that allows continuous learning by picking objects from bins with sloped walls and only moving the objects above the bin, such that they never fall outside of it, where the robot would not be able to grasp them. Riedmiller et al. (2007) learn to steer a car with reinforcement learning which required a recovery policy that took over when the car left the track. This recovery policy must already solve (a variant of) the task that is being learned here. Episodic tasks as in the work of Levine et al. (2015), require a reset to the initial state after each episode, which can be done manually or by instrumenting the environment appropriately.

Although reinforcement learning appears to be a natural framework for robot learning, autonomous reinforcement learning requires substantial instrumentation of the environment. Only through manual or automatic processes to generate the reward, to reset the state, and to ensure safe exploration, can reinforcement learning be applied to robotics.

6.1.4 Simulation Is Useful But No Silver Bullet

Due to the difficulties with data collection and instrumented environments described above, reinforcement learning research is often done in simulation. Most of my own experiments presented in Chapters 4 and 5 are performed in simulation because this offers a number of advantages.

Simulation makes it very easy to gather large amounts of labeled data. In Chapter 4, I used simulation to collect thousands of trajectories of a robot walking through different mazes on a single day, without having to care about hardware issues, battery life, changes in illumination, etc. These experiments are repeatable, adaptable, and scalable, so that I could easily regenerate the data, generate it using a different robot behavior, and generate more data if needed. Because simulation provides access to the underlying state, labels such as the position and orientation of the robot can be directly read out, which makes simulation suitable for supervised learning. And simulation is even more valuable for reinforcement learning.

To develop supervised learning methods, we can apply them on data sets many times without having to collect the data again. But for reinforcement learning, this is not possible. Reinforcement learning methods can only be tested by interacting with an environment. If those evaluations would have to be run in the real world, the development of reinforcement learning methods would virtually come to a stop because every learning experiment during the development of a new method would take an impractical amount of time. Here, simulation plays a crucial role for accelerating or even enabling progress.

Nevertheless, it is important to keep in mind that simulation is only a proxy for reality—that, ultimately, we want to apply these algorithms on real robotic systems. Therefore, we have to be careful about our use of simulations in machine learning research. The first risk of using simulations is that we develop our algorithms for a learning setting in which exploration is safe, data is cheap, and instrumentation of the environment is easy instead of assuming a more realistic learning scenario. This trend can be seen in recent reinforcement learning publications, which often use millions of interactions with the environment (Lillicrap et al., 2015; Heess et al., 2017).

The second risk of experimenting only in simulations is that we might overfit to them by encoding priors into our learning algorithms which apply to the simulation but not to the real world. This can easily happen because simulations have to simplify many aspects of reality. At the same time, we might fail to take advantage of properties of the real world that are not present in simulation such as how soft hands can simplify control for grasping (Deimel and Brock, 2016).

Therefore, we need to constantly assess how much our simulated environments reflect properties of the real world. While we should use simulation to develop our methods, we must also verify our results using real data and real experiments.

Chapters 4 and 5 demonstrate this approach of developing a method for robot learning in simulation and verifying the results on real data or in real experiments.

6.2 The Role of Priors in Machine Learning

Throughout this thesis, I have argued that priors are essential for machine learning and that adding prior knowledge has the potential to improve generalization, in particular in robot learning. In this section, I will argue that we should put priors at the center of our perspective on machine learning. I will first discuss limitations of separating machine learning into supervised learning, unsupervised learning, and reinforcement learning, which is the predominant perspective on machine learning. Then, I will argue for taking a perspective that focuses on the role of prior knowledge in machine learning. I will relate this perspective to deep learning, which has generated powerful tools for studying machine learning from the perspective of encoding prior knowledge. I will conclude this section by describing the implications of the prior-centric perspective for robotics.

6.2.1 Combining Supervised, Unsupervised, and Reinforcement Learning

The majority of machine learning research belongs to one of three problem categories: supervised learning, reinforcement learning, and unsupervised learning (see Section 2.4). This factorization of research enabled the field to focus on different parts of the machine learning problem. Supervised learning focuses on approximating functions from samples of their input and output, unsupervised learning focuses on discovering structure in unlabeled data, and reinforcement learning focuses on learning interaction from trial and error. This focused research has produced important theories, methods, and tools for these sub-problems.

But it is important to note that in their pure forms, none of these standard learning paradigms will enable the kind of learning that we would need for intelligent robots.

Pure supervised learning only relies on labeled data, which is laborious to collect and potentially limits generalization when test and training scenario are different (see Section 6.1.2).

Pure reinforcement learning often requires instrumented environments (see Section 6.1.3). Unless the reward function provides a gradient towards solving the task, reinforcement learning has an additional limitation: It requires the robot to solve the task by chance to even enable learning. Without "stumbling upon" the solution, the robot does not receive the reward that it needs for learning the task.

Pure unsupervised learning can extract structure from abundant unsupervised data, but without any supervision or a reward signal there is no feedback about how the robot

should change its behavior.

If we accept learning as a means to create artificial intelligence, instead of as an end in itself (see Section 6.1.1), it follows that intelligent robots should learn by extracting information from all available sources: scarce but highly informative labeled data, abundant unlabeled data, and trial and error. By combining these paradigms, they can complement each other and remedy their weaknesses.

There are a number of recent approaches that go in this direction, for example using auxiliary supervised or unsupervised tasks in reinforcement learning (Jaderberg et al., 2016; Mirowski et al., 2016), initializing a policy by learning from demonstrations and fine-tuning using reinforcement learning (Abbeel et al., 2010), or using supervised learning to generalize from multiple policies trained by model-based reinforcement learning (Levine and Koltun, 2013). But these combinations still rely on the same standard learning objectives from the original subfields, which is an unnecessary restriction.

Supervised, unsupervised, and reinforcement learning provide standard learning objectives, which are really special cases. By combining these paradigms, there is an opportunity to consider other learning objectives. In Chapter 5, I have demonstrated that if we apply the unsupervised learning goal of finding lower dimensional manifolds in the reinforcement learning setting, we can generate much richer objective functions. This is possible because the reinforcement learning problem provides sequences of observations, actions, and rewards, which afford more assumptions than plain unlabeled data assumed by pure unsupervised learning. Learning with side information generalizes this idea of expressing priors about how different variables relate to each other (see Section 5.8).

If we combine these different problem categories, we need another paradigm to structure our thinking about machine learning. I believe that priors should be at the center of such a new paradigm.

6.2.2 Prior-Centered Machine Learning

Before explaining further why I think that we should adopt a machine learning paradigm that puts priors at the center, I want to clearly state I do not claim credit for this idea. The importance of prior knowledge in machine learning has been expressed many times (Mitchell, 1980; Geman et al., 1992; Wolpert, 1996; Bengio et al., 2013). In this section, I do not presume to present a new view on machine learning, but merely to summarize arguments for why putting priors at the center of machine learning would indeed be a good idea.

Mitchell (1980) argues that we need priors or biases: "If consistency with the training instances is taken as the sole determiner of appropriate generalizations, then a program can never make the inductive leap necessary to classify instances beyond those it has observed. Only if the program has other sources of information, or biases for choosing

one generalization over the other, can it non-arbitrarily classify instances beyond those in the training set."

Geman et al. (1992) analyzes the bias-variance trade-off and comes to the conclusion that "learning complex tasks is essentially impossible without the a priori introduction of carefully designed biases into the machines's architecture" and that "identifying the right 'preconditions' is *the* substantial problem in neural modeling.".

Wolpert (1996) raises the questions, "Can one actually get something for nothing in supervised learning? Can one get useful, caveat-free theoretical results that link the training set and the learning algorithm to generalization error, without making assumptions concerning the target?", and refute them by proving the no free lunch theorem.

Bengio et al. (2013) reviews the field of representation learning and conclude that machine learning must incorporate "many general priors about the world around us." They propose a list of priors for artificial intelligence and argue that refining this list and incorporating it into machine learning will bring us closer to artificial intelligence.

Logic, theory, and practice all show that priors are essential for machine learning. The open question is not whether machine learning needs prior knowledge, but how specific or general those priors should be for the problem of (embodied) artificial intelligence.

Putting priors at the center of machine learning has practical implications for machine learning research. Since the role of priors is to enable generalization, we should consider changing the way we test our learning algorithms, putting more emphasis on increasing the "inductive leap" that we require from our learning methods by introducing differences between the training and test conditions. We must search for priors that allow generalization beyond the current capabilities of learning systems.

My experiments in Section 4.6.2 show a step in this direction, where I tested the generalization of learned localization to different robot behavior. The result showed that generic deep learning models are not able to perform this generalization, while the algorithmic prior in the differentiable particle filter enabled this generalization.

A recent study by de Bruin et al. (2018) also demonstrated the impact of priors described in this thesis on generalization. They integrated state representation learning into deep reinforcement learning and performed experiments in a simulated driving task. Their results show that robotic priors from Chapter 5 only slightly improve performance in the training domain but substantially improve generalization to new test domains.

6.2.3 The True Potential of Deep Learning

Although deep learning (LeCun et al., 2015) has already produced a range of impressive results and changed the field of machine learning, I think the true potential of deep learning is to enable prior-centered machine learning.

In some sense, deep learning is already doing that because it is most successful in computer vision using convolutional neural networks, which encode prior knowledge about visual input. New network architectures generated by deep learning research define different hypothesis spaces and thereby new ways of encoding prior knowledge.

But the *real potential* of deep learning results from the tools that have been developed by the deep learning community. These tools tremendously facilitate the implementation of new hypothesis spaces and learning objectives. Deep learning software frameworks such as TensorFlow (Abadi et al., 2015) allow the definition of arbitrary parameterized computation graphs, i.e. hypothesis spaces. Through auto-differentiation, they can perform gradient-based optimization of the parameters for arbitrary learning objectives. Without auto-differentiation, gradients need to be computed by hand (see Appendix A.1) and are only valid for a certain combination of learning objective and hypothesis space.

In addition to auto-differentiation, which has been known for much longer (Pearlmutter and Siskind, 2008), these frameworks include powerful learnable models with appropriate initialization, a suite of optimization methods, and transparent use of powerful GPUs for computation. My work on differentiable Bayes filters (see Chapter 4) would have been virtually impossible without relying on these frameworks.

These software tools enable a new kind of programming that includes learnable parts, which some refer to as *differentiable (functional) programming* (Olah, 2015). This new programming paradigm will greatly simplify the encoding of prior knowledge.

6.2.4 Robotics-Specific Machine Learning

The flip side of encoding prior knowledge into machine learning is that it reduces generality. By including priors about the problem, we automatically exclude problems that are inconsistent with these priors from the domain of the learning method. We might still be able to apply a learning method outside of its domain but it will not work very well. The differentiable particle filters described in Chapter 4, for example, are well suited for learning state estimation. But if we apply them to a language task, they will perform very poorly. There is no free lunch—improved generalization in one task comes at cost of reduced generalization in another one (Wolpert, 1996).

The application of prior-centric machine learning to robotics will lead to robotics-specific machine learning methods. While those methods might perform poorly for non-robotic problems, they should improve generalization in robotics and thereby help to make

Table 6.1: Main chapters

Chapter	Problem	Prior (what)	Implementation (how)
3	Object segmentation in the Amazon picking challenge	Task-specific priors: known objects and shelf, object convexity	Task-specific features, post-processing
4	State estimation from non-Markov observations	Algorithmic priors: Bayes filter structure	Restricted hypothesis space for end-to-end learning
5	Unsupervised learning of state representations	Robotic priors based on Newtonian physics	Proxy learning objectives

progress towards understanding embodied intelligence and creating intelligent robots.

The robotic domain is rich in structure and affords a number of robotic priors. Research in robotics-specific machine learning comes back to the two questions that I raised in the introduction of my thesis:

1. *What* are the right priors for robotics?

2. *How* can we best incorporate these priors into machine learning?

This thesis investigated multiple possible answers to these questions in the context of robotic perception.

6.3 This Thesis

In this thesis, I have approached three different perception problems in robotics by combining machine learning with prior knowledge. In each of the three main chapters, I focused on a different source of prior knowledge and a different way of incorporating this knowledge into learning methods (see Table 6.1), which are potential answers to the *what* and *how* questions.

The three main chapters lie along a spectrum from application-centered research to basic research. Chapter 3 addresses a narrow problem scope using very specific priors. The perception method developed in that chapter is in some sense the strongest because it is closest to a realistic application, but it is also most narrow in scope. The other two chapters increase the problem scope and the generality of the employed priors. The methods presented in Chapter 5 are more general than those in Chapter 4 because they do not require labeled data, but they are also more limited because they can only estimate the state from a single observation instead of learning a recursive update loop.

These problems were matched with priors along a similar continuum from specific to generic, where task-specific priors about the Amazon picking challenge are the most specific, followed by algorithmic priors that make use of algorithmic solutions for a certain category of problems, and finally robotic priors based on the laws of physics, which apply to all robots.

We also saw different ways of incorporating those priors, varying from very strong constraints in the form of predefined features and post-processing steps, coupled with a minimal amount of learning (basically statistics over features). In the second chapter, we algorithmic priors implemented as hard constraints that restrict the hypothesis space, combined with end-to-end learning to optimize all parameters for end-to-end performance. In the final chapter, we saw robotic priors implemented as soft learning objectives, which were optimized as a proxy instead of end-to-end performance.

6.3.1 What: Tasks, Algorithms, Physics

What are the right priors for robotics? In my thesis, I proposed three potential sources of prior knowledge: tasks, algorithms, and physics.

Task-specific priors capture knowledge about certain tasks that a robot is expected to perform. Task-specific priors can capture very detailed information about any property that is invariant in a task, such as improving object perception using a model of the shelf that the objects are placed in. Such priors have a narrow applicability and are not very useful for others tasks, but for the task at hand, they can provide strong regularization and thus enable highly data efficient learning.

Algorithmic priors capture assumptions in algorithms, which are more abstract than task-specific information. The Bayes filter algorithm, for example, applies too many state estimation problems, such as localization, mapping, or object tracking, because it makes assumptions about the general structure of the state estimation problem rather than the specifics in one task or environment. Thus, algorithmic priors can help to improve generalization across a larger range of robotic tasks.

Physics-based priors capture the structure of physical interaction between the robot and the environment. The laws of physics imposes structure such as momentum, which can be used to learn a representation of the world that is consistent with physics. Since robots always interact with the physical world—regardless of their specific task—robotic priors are widely applicable in robotics.

Note how these priors differ from the *ubiquitous machine learning priors* and the *generic AI priors* from Section 2.5. While these priors apply to a passive disinterested observer, the robotic priors in this thesis take the tasks and interaction into account.

We do not have to choose a single source of prior knowledge. Since all of these priors capture different information, they can complement each other. Ultimately, robots need a set of priors that together enables efficient learning and this set will include ubiquitous ML priors, generic AI priors, robotic priors, and others that we are still missing.

Priors that capture a wide range of tasks will be important for equipping robots with the necessary versatility. Specifics such as shelf-models should certainly be learned. But certain narrow priors, such as perception that is tailored to distinguish human faces, could

be useful if they help to solve tasks that are sufficiently prevalent.

6.3.2 How: Hypothesis Spaces and Learning Objectives

How can we best incorporate these priors into machine learning? In the introduction, I described machine learning by its four components—data, hypothesis space, learning objective, and optimization method—and argued that we can encode prior knowledge into each of them. In this thesis, I focused on two of these components: hypothesis spaces and learning objectives. Interestingly, encoding priors into these two components correspond to the two types of inductive biases that were proposed by Mitchell, restriction biases and preference biases (Mitchell, 1997, p. 64).

Restriction biases restrict the hypothesis space and thereby exclude certain hypothesis. Restricting the hypothesis space is a rigid way of encoding prior knowledge because the learning method cannot, under any circumstances, produce a hypothesis outside of the hypothesis space.

Preference biases define learning objectives that express a preference for one hypothesis over another. Such learning objectives are often combined with others, for example by using them as a regularization term in addition to a supervised learning objective.* Compared to restriction biases, preference biases are more flexible because learning can produce hypotheses that are not fully consistent with a learning objective (have non-zero loss). When we encode priors in objective functions, consistency with these priors is balanced with other learning objectives.

Whether it is better to encode a prior by restricting the hypothesis space or by defining an objective function, depends on the nature of the prior. Soft constraints such as the slowness prior, for example, are well suited to be encoded in a learning objective that expresses a preference for slowly changing states. Hard constraints such as a particle-based representation of the belief or a position-velocity structure in the state lend themselves to be encoded by defining an appropriate hypothesis space.

While restricting the hypothesis space is more rigid than defining learning objectives, it has the advantage of simplifying the learning process. By restricting the hypothesis space, the method trivially achieves consistency with the encoded prior, even before any optimization of the parameters takes place.

The perspective that learning objectives encode prior knowledge also sheds light on the dichotomy between end-to-end learning and individual learning. By training parts of a system individually, we encode prior knowledge about the function of these parts. By training parts *in* the complete system to optimize end-to-end performance, we encode

*Standard regularization objectives, such as L1 or L2 regularization are defined over the learnable parameters but ignore the data. The learning objectives presented in Chapter 5 regularize learning by incorporating data. In general, this approach is called learning with side information (see Section 5.8).

prior knowledge about how the part is embedded into the larger system, e.g. which inputs the part will receive from other components and how errors in its output affect overall system performance. Our results in Chapter 4 suggest that combining these objectives leads to best performance.

6.4 Research Vision: A Robot-Learning System

Apart from future work on structured state representations, which I have outlined in Section 5.9.2, I would like to work towards integrating different ideas presented in this thesis into a robot-learning system.

There is accumulating evince for the need and the potential of integrating the approaches presented in this thesis. The post-APC evaluation of our object segmentation method in Chapter 3 showed that the method would benefit from integrating it more strongly with machine learning. Our experiments with differentiable histogram filters in Chapter 4 suggest that state estimation can be learned without supervision by combining restrictions of the hypothesis space with appropriate learning objectives (see Section 4.5.2). The position-velocity encoder in Chapter 5 is able learn state spaces that are structured into positions and velocities, which could facilitate learning forward models for reinforcement learning. A recent study by de Bruin et al. (2018) integrated our state representation learning objective into deep reinforcement learning and showed they can substantially improve generalization for reinforcement learning to new test domains.

My research vision is to integrate these different approaches into a combined robot-learning system.

This system would define robot behavior by a network of modules that can be rigidly defined or learned. Candidates for rigid implementations are perception skills such as the perception of articulated objects (Martín-Martín et al., 2016), or planning mechanisms such as logic-geometric programming (Toussaint, 2015). Both of these can potentially facilitate learning other modules by backpropagating gradients through them. Other modules can be learned using a restricted hypothesis space, such as the differentiable particle filter presented in Chapter 4. And some modules can be fairly general learnable models, e.g. deep neural networks (LeCun et al., 2015). The arrangement of these modules could be fixed initially, but it might be useful to at least partially learn how they assemble the overall system.

Based on the results presented in this thesis, there will likely not be a single learning objective for optimizing this network of modules, but many objectives that range from end-to-end performance of the entire system, to objectives on collections of modules that improve their interplay, and to objectives on single modules. These objectives will relate to any *side information* (see Sec. 5.8) that is useful, optimizing overall reward, using

supervision in the form of demonstrations and instructions, making predictions, and making different internal estimations consistent with each other, for example by specifying learning objectives on intermediate state representations as in Chapter 5).

To make progress towards this research vision, we need to start integrating solutions into a combined system to study their interaction while continuing to improve our understanding of the individual modules. Ultimately, the questions remain the same that I asked in the beginning: What are the right priors for robotics and how can we encode them into learning? This thesis has shed some light on these questions in the context of robot perception. There is still much more to be done but I believe that by continuing research on robotic priors and by integrating them into a robot learning system, we can realize this research vision.

6.5 Final Thoughts

In a recent debate about "deep learning, structure, and innate priors" (See, 2018), Yann LeCun described the need for prior knowledge in machine learning as a "necessary evil" while Christopher Manning considered it a "necessary good". I clearly side with Christopher Manning because priors in machine learning represent our understanding of the world from the perspective of a learning embodied agent and I find it inspiring to seek that understanding. I think that focusing on priors does not only make machine learning research more promising, but also more exciting.

A

A.1 Derivatives of Learning Objectives

Temporal Coherence

$$
\begin{aligned}
L_{\text{temp.}}(D, \hat{\phi}) &= \mathbf{E}\Big[\|\Delta \hat{s}_t\|^2\Big] \\
&= \mathbf{E}\Big[\|\hat{s}_{t+1} - \hat{s}_t\|^2\Big] \\
&= \mathbf{E}\left[\sum_k (\hat{s}_{k,t+1} - \hat{s}_{k,t})^2\right] \\
&= \mathbf{E}\left[\sum_k \left(\sum_l \hat{W}_{k,l}\,(o_{l,t+1} - o_{l,t})\right)^2\right] \\
\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{temp.}}(D, \hat{\phi}) &= \mathbf{E}\left[\frac{\partial}{\partial \hat{W}_{i,j}} \sum_k \left(\sum_l \hat{W}_{k,l}\,(o_{l,t+1} - o_{l,t})\right)^2\right] \\
&= \mathbf{E}\left[\frac{\partial}{\partial \hat{W}_{i,j}} \left(\sum_l \hat{W}_{i,l}\,(o_{l,t+1} - o_{l,t})\right)^2\right] \\
&= \mathbf{E}\left[2\left(\sum_l \hat{W}_{i,l}\,(o_{l,t+1} - o_{l,t})\right)(o_{j,t+1} - o_{j,t})\right] \\
&= \mathbf{E}\left[2(\hat{s}_{i,t+1} - \hat{s}_{i,t})\,(o_{j,t+1} - o_{j,t})\right] \\
&= \mathbf{E}\Big[2\,\Delta \hat{s}_{i,t}\,\Delta o_{j,t}\Big]
\end{aligned}
$$

Causality

$$\begin{aligned}
L_{\text{caus.}}(D, \hat{\phi}) &= \mathbf{E}\left[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \;\middle|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\right] \\
&= \mathbf{E}\left[e^{-\sum_k (\hat{s}_{k,t_2} - \hat{s}_{k,t_1})^2} \;\middle|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\right] \\
&= \mathbf{E}\left[e^{-\sum_k \left(\sum_l \hat{W}_{k,l}\left(o_{l,t_2} - o_{l,t_1}\right)\right)^2} \;\middle|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\right]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \hat{W}_{i,j}} L_{\text{caus.}}(D, \hat{\phi}) &= \mathbf{E}\left[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \left(-2 \sum_l \hat{W}_{i,l}\left(o_{l,t_2} - o_{l,t_1}\right)\right)\left(o_{j,t_2} - o_{j,t_1}\right) \;\middle|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\right] \\
&= \mathbf{E}\left[-2e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}\left(\hat{s}_{i,t_2} - \hat{s}_{i,t_1}\right)\left(o_{j,t_2} - o_{j,t_1}\right) \;\middle|\; a_{t_1} = a_{t_2}, r_{t_1} \neq r_{t_2}\right]
\end{aligned}$$

Proportionality

$$\begin{aligned}
L_{\text{prop.}}(D, \hat{\phi}) &= \mathbf{E}\Big[\left(\|\Delta \hat{s}_{t_2}\| - \|\Delta \hat{s}_{t_1}\|\right)^2 \;\Big|\; a_{t_1} = a_{t_2}\Big] \\
&= \mathbf{E}\Big[\Bigg(\sqrt{\sum_k \Big(\sum_l W_{k,l}\left(o_{l,t_2+1} - o_{l,t_2}\right)\Big)^2} \\
&\qquad - \sqrt{\sum_k \Big(\sum_l W_{k,l}\left(o_{l,t_1+1} - o_{l,t_1}\right)\Big)^2}\Bigg)^2 \;\Big|\; a_{t_1} = a_{t_2}\Big]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial W_{i,j}} L_{\text{prop.}}(D, \hat{\phi}) &= \mathbf{E}\Big[2\left(\|\Delta \hat{s}_{t_2}\| - \|\Delta \hat{s}_{t_1}\|\right)\Big(\frac{1}{2\|\Delta \hat{s}_{t_2}\|} \underbrace{\frac{\partial}{\partial W_{i,j}}\left(\sum_l W_{i,l}\left(o_{l,t_2+1} - o_{l,t_2}\right)\right)^2}_{2\Delta \hat{s}_{i,t_2} \Delta o_{j,t_2}} \\
&\qquad - \frac{1}{2\|\Delta \hat{s}_{t_1}\|} \underbrace{\frac{\partial}{\partial W_{i,j}}\left(\sum_l W_{i,l}\left(o_{l,t_1+1} - o_{l,t_2}\right)\right)^2}_{2\Delta \hat{s}_{i,t_1} \Delta o_{j,t_1}}\Big) \;\Big|\; a_{t_1} = a_{t_2}\Big] \\
&= \mathbf{E}\Big[2\left(\|\Delta \hat{s}_{t_2}\| - \|\Delta \hat{s}_{t_1}\|\right)\left(\frac{\Delta \hat{s}_{i,t_2} \Delta o_{j,t_2}}{\|\Delta \hat{s}_{t_2}\|} - \frac{\Delta \hat{s}_{i,t_1} \Delta o_{j,t_1}}{\|\Delta \hat{s}_{t_1}\|}\right) \;\Big|\; a_{t_1} = a_{t_2}\Big]
\end{aligned}$$

Repeatability

$$L_{\text{rep.}}(D, \hat{\phi}) = \mathbf{E}\Big[e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2 \;\Big|\; a_{t_1} = a_{t_2}\Big]$$

$$\frac{\partial}{\partial W_{i,j}} L_{\text{rep.}}(D, \hat{\phi}) = \mathbf{E}\Big[\|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2 \underbrace{\frac{\partial}{\partial W_{i,j}} e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}}_{*} + e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} \underbrace{\frac{\partial}{\partial W_{i,j}} \|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2}_{**} \;\Big|\; \dots\Big]$$

$$* = -e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} 2(\hat{s}_{i,t_2} - \hat{s}_{i,t_1})\,(o_{j,t_2} - o_{j,t_1})$$

$$\begin{aligned} ** &= 2\left(\sum_l \hat{W}_{i,l}\,(o_{l,t_2+1} - o_{l,t_2} - o_{l,t_1+1} + o_{l,t_1})\right)(o_{j,t_2+1} - o_{j,t_2} - o_{j,t_1+1} + o_{j,t_1}) \\ &= 2(\Delta\hat{s}_{i,t_2} - \Delta\hat{s}_{i,t_1})\,(\Delta o_{j,t_2} - \Delta o_{j,t_1}) \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial W_{i,j}} L_{\text{rep.}}(D, \hat{\phi}) = \mathbf{E}\Big[&-\|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2 e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2} 2(\hat{s}_{i,t_2} - \hat{s}_{i,t_1})\,(o_{j,t_2} - o_{j,t_1}) \\ &+ 2e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}(\Delta\hat{s}_{i,t_2} - \Delta\hat{s}_{i,t_1})\,(\Delta o_{j,t_2} - \Delta o_{j,t_1}) \;\Big|\; a_{t_1} = a_{t_2}\Big] \\ = \mathbf{E}\Big[&2e^{-\|\hat{s}_{t_2} - \hat{s}_{t_1}\|^2}\Big((\Delta\hat{s}_{i,t_2} - \Delta\hat{s}_{i,t_1})(\Delta o_{j,t_2} - \Delta o_{j,t_1}) \\ &- \|\Delta\hat{s}_{t_2} - \Delta\hat{s}_{t_1}\|^2(\hat{s}_{i,t_2} - \hat{s}_{i,t_1})(o_{j,t_2} - o_{j,t_1})\Big) \;\Big|\; a_{t_1} = a_{t_2}\Big] \end{aligned}$$

A.2 Learning with Side Information

A.2.1 Patterns as Probabilistic Graphical Models

To complement the computation flow schemas of the patterns used throughout the paper, we provide an interpretation of the main patterns as probabilistic graphical models (PGMs). These models treat the variables and functions introduced in Sec. 5.8.1 as random variables, represented as nodes. Arrows between these random variables indicate causal relationships. Gray nodes indicate observable, and white nodes latent random variables. The latent functions can be learned by performing inference in these models.

The PGMs for the four main patterns are shown in Fig. A.1. The variables $\mathbf{x}, \mathbf{s}, \mathbf{z}$ and $\mathbf{y}$ are observable random variables and are part of the training data D. The functions ϕ, ψ and β have become latent random variables, which the observed variables are conditioned on. We now discuss aspects of individual patterns.

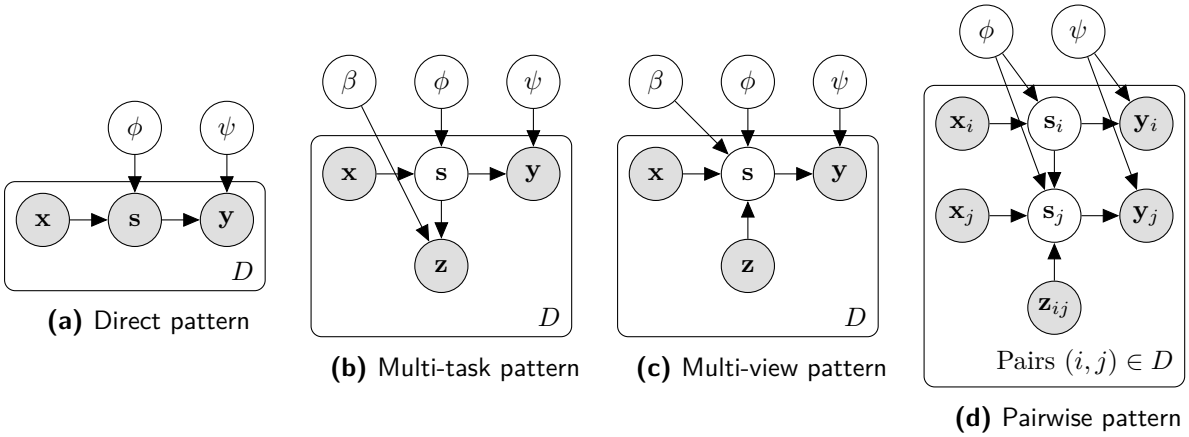


Figure A.1: Probabilistic graphical models for patterns

The *direct pattern* is shown in Fig. A.1a. In comparison to its computation flow graph (Fig. 5.29), the side information $\mathbf{z}$ is considered as drawn from the distribution over $\mathbf{s}$, and therefore $\mathbf{z}$ does not appear in the graphical model of the direct pattern.

Fig. A.1b and Fig. A.1c show the PGM for the *multi-task* and *multi-view pattern*, respectively (computation flow graphs: Fig. 5.30 and Fig. 5.32). We see that they are structurally similar and only differ on whether $\mathbf{z}$ depends on $\mathbf{s}$ and β or whether $\mathbf{s}$ depends on $\mathbf{z}$ and β. In this regard, they are equivalent to their corresponding computation flow schemas apart from the fact that the PGM for the multi-view pattern conceals how the variables $\mathbf{x}$ and $\mathbf{z}$ belong to the functions ϕ and β.

Finally, the prototypical *pairwise pattern* is shown in Fig. A.1a (computation flow graph: Fig. 5.34). Notice that here $\mathbf{s}_j$ is conditioned on $\mathbf{z}_{i,j}$ and $\mathbf{s}_i$, reflecting the fact that $\mathbf{z}_{i,j}$ is information about how $\mathbf{s}_j$ relates to $\mathbf{s}_i$.

Several interesting research questions arise from the probabilistic view on learning with side information. The majority of the reviewed literature uses non-probabilistic loss functions, mostly for training neural networks. Translating them into probabilistic ones is an interesting, but non-trivial research question, as the recent work on variational auto-encoders (which are a probabilistic version of auto-encoders) shows (Kingma and Welling, 2013). A similar question arises on the relationship of the side objectives and prior probability distributions on ϕ, ψ, β and $\mathbf{s}$. It would be interesting to investigate whether certain side objectives can be shown to be equivalent to priors in the Bayesian sense, similar to the well-known fact that L2 regularization is equivalent to a Gaussian prior.

A.2.2 Overview of Related Work

In the following table, we summarize related works that apply learning with side information. Since an exhaustive list of references for each pattern is beyond the scope of this paper, we include works that span a wide variety of instantiations of the proposed patterns and refer to survey articles if available.

Abbreviations: AE=auto-encoder, CCA=canonical correlation analysis, ED=eigen decomposition, GMLVQ=generalized matrix learning vector quantization, kNN=k-nearest-neighbors, LBP=locally binary pattern, MMD=maximum mean discrepancy, NN=neural network, RBM=restricted Boltzmann machine, RL=reinforcement learning, SGD=stochastic gradient descent, SL=supervised learning (classification unless stated otherwise), SVM=support vector machine, UL=unsupervised learning

Pattern	**Side Objective**	**Articles**	**Method, Train. Procedure**	**Application: Task, Input, Dataset**	**Side Information**
Direct (Fig. 5.29)	SVM loss	Cheng and Baldi (2007)	SVM *(decoupl.)*	SL: Contact prediction on sequences	Secondary (3D) structure categories
	Regression on highly predictive features of side information	Chen et al. (2012)	AdaBoost+ *(simul.)*	SL on images: Digit (Vapnik and Vashist, 2009), facial expression (Cohn-Kanade)	Holistic image descriptions, LBP features from high-res images
	Regression loss	Vapnik and Izmailov (2015)	SVM with knowledge transfer *(decoupl.)*	SL on images Theoretical analysis: learning using privileged information	Image sections
Multi-task (Fig. 5.30)	Various supervised: hinge, MSE, softmax	Caruana (1997)	NN *(simul.)*	SL: pneumonia detection	Hematocrit, white blood cell count, potassium
		Evgeniou and Pontil (2004)	SVM *(simul.)*	SL: exam score prediction	One task per school
		Levine et al. (2015)	Conv. NN *(decoupl.)*	RL on RGB-D: robot manipulation	Image class, object pose
		Zhao and Itti (2015)	Conv. NN *(simul.)*	SL on images (YYY-20M)	Object pose
		Pan and Yang (2010)	Survey	SL, UL	-
		Baxter (2000); Ando and Zhang (2005) Maurer (2006)		Theoretical analysis of multi-task learning	-
Multi-view (Fig. 5.32)	Kernel CCA+soft margin SVM hinge loss	Farquhar et al. (2005)	SVM-2K *(simul.)*	SL on images (PASCAL-VOC)	Keypoint features (SIFT)
	AE reconstruction error	Ngiam et al. (2011)	RBM / NN *(simul.)*	SL on video/audio (various, e.g. CUAVE, AVLetters)	Video/audio
	Adjusted rand index, mutual information	Feyereisl and Aickelin (2012)	k-means	UL on images: MNIST	Poetic descriptions
	Kernel CCA [+MMD for domain adaption]	Chen et al. (2014)	Kernel SVM on kernel descriptor features	SL on RGB: gender (EURECOM, LFW-a), object (RGB-D O.D., Catech-256)	RGB-D
	SPoC (non-linear CCA)	Dähne et al. (2014)	Linear/quadratic, ED *(decoupl.)*	SL: Mental state prediction	EEG diff. subject

	CCA+AE reconstruction error	Wang et al. (2015)	Deep Canonically Correlated AE and others *(simul.)*	SL on images (MNIST), speech (XRMB), word embedding (WMT2011)	Noisy images, articulations, 2nd language
	-	Sun (2013)	Survey	SL, UL	-
	-	Blum and Mitchell (1998) Wang and Zhou (2010)		Theoretical analysis of multi-view learning	-
Pairwise Similarity (Fig. 5.34)	Slowness (first equation in Sec. 5.8.2 with $\mathbf{z}_{ij} = \mathbb{1}\{j = i + 1\}$ and covariance constraints).	Wiskott and Sejnowski (2002); Legenstein et al. (2010)	Linear/quadratic, ED *(decoupl.)*	RL on images: Navigation (physical simulation)	Time index
	Equations in Sec. 5.8.2 with margin-based $\sigma(d)$ (see Section 5.8.2)	Hadsell et al. (2006)	Conv. NN, *(decoupl.)*	UL on images: dimensionality reduction (MNIST, NORB)	
		Weston et al. (2008)	Conv. NN *(simul.)*	SL on images (MNIST, COIL100)	
	State predictability +variational AE	Watter et al. (2015)	CNN *(simul.)*	RL: inverted pendulum, cart-pole, robot arm	
	Adapted SVM loss	Vapnik and Vashist (2009)	SVM with similarity control *(simul.)*	SL: protein classification, finance market prediction, digit recognition	3D protein structure, future events, textual description
	Distance metric learning	Fouad et al. (2013)	GMLVQ/kNN *(decoupl.)*	SL: images (MNIST); galaxy morphology	Poetic descriptions; spectral features
(Fig. 5.36)	Hierarchical multi-class loss	Silla Jr. and Freitas (2010)	Survery on hierarchical classification (SL)		Label similarity
Pairwise Transformation (Fig. 5.35a)	Softmax	Hinton et al. (2011)	NN; transforming AE *(simul.)*	SL for pose prediction (MNIST, 3D simulation)	Relative pose
(Fig. 5.35b)	See Hadsell et al. (2006)	Jayaraman and Grauman (2015)	Siamese-style conv. NN *(simul.)*	SL on images (NORB, KITTI, SUN)	Relative pose (discretized; with k-means)
(Fig. 5.35c)	Softmax	Agrawal et al. (2015)	Siamese-style conv. NN *(pre-train/fine-tune)*	SL on images (MNIST, SF, KITTI)	Relative pose (discretized; uniformly)
	Various	Jonschkowski and Brock (2015)	Linear, SGD *(decoupl.)*	RL: control, navigation,	Actions, rewards, time
Irrelevance (Fig. 5.31)	$L \approx \|\|\psi^T \beta\|\|_{\mathrm{F}}^2$, with ψ, β linear, $\|\| \cdot \|\|_{\mathrm{F}}^2$ Frobenius norm.	Romera-Paredes et al. (2012)	Linear, orthogonal matrix factorization	SL on images: emotion detection (JAFFE)	Subject identity

Bibliography

Overview of Amazon Picking Challenge 2015. `http://pwurman.org/amazonpickingchallenge/2015/details.shtml`. Accessed: 2017-11-14.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `http://tensorflow.org/`, 2015.

Pieter Abbeel, Adam Coates, and Andrew Y.T. Ng. Autonomous Helicopter Aerobatics through Apprenticeship Learning. *The International Journal of Robotics Research*, 29 (13):1608–1639, 2010.

Pulkit Agrawal, Joao Carreira, and Jitendra Malik. Learning to See by Moving. *arXiv:1505.01596*, 2015.

Rie Kubota Ando and Tong Zhang. A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data. *Journal of Machine Learning Research*, 6: 1817–1853, 2005.

James E. Baker. Reducing Bias and Inefficiency in the Selection Algorithm. In *Proceedings of the International Conference on Genetic Algorithms (ICGA)*, pages 14–21, 1987.

Jonathan Baron. *Thinking and Deciding*. Cambridge University Press, 2000.

Jonathan Baxter. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research (JAIR)*, 12:149–198, 2000.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, and others. Deepmind Lab. *arXiv:1612.03801*, 2016.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 153–160, 2007.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2006.

Avrim Blum and Tom Mitchell. Combining Labeled and Unlabeled Data with Co-Training. In *Proceedings of the Annual Conference on Computational Learning Theory (COLT)*, pages 92–100, 1998.

Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S. Sukhatme. Interactive Perception: Leveraging Action in Perception and Perception in Action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

Byron Boots, Sajid M. Siddiqi, and Geoffrey J. Gordon. Closing The Learning-Planning Loop with Predictive State Representations. *International Journal of Robotics Research*, 30(7):954–966, 2011.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Conference on Computational Learning Theory (COLT)*, pages 144–152, 1992.

Michael Bowling, Ali Ghodsi, and Dana Wilkinson. Action Respecting Embedding. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 65–72, 2005.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. CRC press, 1984.

Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

Jixu Chen, Xiaoming Liu, and Siwei Lyu. Boosting with Side Information. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, pages 563–577, 2012.

Lin Chen, Wen Li, and Dong Xu. Recognizing RGB Images by Learning from RGB-D Data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1418–1425, 2014.

Yang Chen and Gérard Medioni. Object Modelling by Registration of Multiple Range Images. *Image and Vision Computing*, 10(3):145–155, 1992.

Jianlin Cheng and Pierre Baldi. Improved Residue Contact Prediction Using Support Vector Machines and A Large Feature Set. *BMC Bioinformatics*, 8(1):113, 2007.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The Loss Surfaces of Multilayer Networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 192–204, 2015.

Luis C. Cobo, Kaushik Subramanian, Charles L. Isbell Jr., Aaron D. Lanterman, and Andrea Lockerd Thomaz. Abstraction from Demonstration for Efficient Reinforcement Learning in High-Dimensional Domains. *Artificial Intelligence*, 216(1):103–128, 2014.

Alvaro Collet, Manuel Martinez, and Siddhartha S. Srinivasa. The MOPED Framework: Object Recognition and Pose Estimation for Manipulation. *The International Journal of Robotics Research*, 30(10):1284 –1306, 2011.

Nikolaus Correll, Kostas E. Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M. Romano, and Peter R. Wurman. Analysis and Observations From the First Amazon Picking Challenge. *IEEE Transactions on Automation Science and Engineering*, PP(99):1–17, 2016.

Sven Dähne, Vadim V. Nikulin, David Ramírez, Peter J. Schreier, Klaus-Robert Müller, and Stefan Haufe. Finding Brain Oscillations with Power Sependencies in Neuroimaging Data. *NeuroImage*, 96:334–348, 2014.

Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2933–2941, 2014.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Integrating State Representation Learning into Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, PP(99):1–1, 2018.

DeepMind. Sonnet: TensorFlow-Based Neural Network Library. `https://github.com/deepmind/sonnet`, 2017.

Raphael Deimel and Oliver Brock. A Novel Type of Compliant and Underactuated Robotic Hand for Dexterous Graspings. *The International Journal of Robotics Research*, 35(1-3):161–185, 2016.

Thomas G. Dietterich. Machine Learning for Sequential Data: A Review. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition (SSPR)*, pages 15–30, 2002.

Pedro Domingos. A Few Useful Things to Know about Machine Learning. *Communications of the ACM*, 55(10):78–87, 2012.

Alain Droniou, Serena Ivaldi, and Olivier Sigaud. Deep Unsupervised Network for Multimodal Perception, Representation and Classification. *Robotics and Autonomous Systems*, 71:83–98, 2015.

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating Human Priors for Playing Video Games. In *Workshop at International Conference on Learning Representations (ICLR)*, 2018.

Siegmund Duell, Steffen Udluft, and Volkmar Sterzing. Solving Partially Observable Reinforcement Learning Problems with Recurrent Neural Networks. In *Neural Networks: Tricks of the Trade*, volume 7700, pages 709–733. Springer Berlin Heidelberg, 2012.

Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martín-Martín, Arne Sieverling, Vincent Wall, and Oliver Brock. Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems. In *Proceedings of Robotics: Science and Systems (RSS)*, 2016.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why Does Unsupervised Pre-training Help Deep Learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

Mark Everingham, S.M. Ali Eslami, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2014.

Theodoros Evgeniou and Massimiliano Pontil. Regularized Multi-Task Learning. In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD)*, pages 109–117. ACM, 2004.

Jason Farquhar, David Hardoon, Hongying Meng, John S. Shawe-taylor, and Sandor Szedmak. Two View Learning: SVM-2K, Theory and Practice. In *Advances in Neural Information Processing Systems (NIPS)*, pages 355–362, 2005.

Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

Jan Feyereisl and Uwe Aickelin. Privileged Information for Data Clustering. *Journal of Information Science*, 194:4–23, 2012.

Evelyn Fix and Joseph L. Hodges Jr. Discriminatory Analysis-Nonparametric Discrimination: Consistency Properties. Technical report, California Univ Berkeley, 1951.

Shereen Fouad, Peter Tino, Somak Raychaudhury, and Petra Schneider. Incorporating Privileged Information Through Metric Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 24(7):1086–1098, 2013.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer New York, 2001.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision Meets Robotics: The KITTI Dataset. *The International Journal of Robotics Research*, 32 (11):1231–1237, 2013.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural Networks and the Bias/Variance Dilemma. *Neural computation*, 4(1):1–58, 1992.

Gerd Gigerenzer and Henry Brighton. Homo Heuristicus: Why Biased Minds Make Better Inferences. *Topics in Cognitive Science*, 1(1):107–143, 2009.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv:1410.5401*, 2014.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive Mapping and Planning for Visual Navigation. *arXiv:1702.03920*, 2017.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: Learning Discriminative Deterministic State Estimators. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4376–4384, 2016.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1735–1742, 2006.

Martie G. Haselton, Daniel Nettle, and Damian R. Murray. The Evolution of Cognitive Bias. In *The Handbook of Evolutionary Psychology*. John Wiley & Sons, Inc., 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385*, 2015.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of Locomotion Behaviours in Rich Environments. *arXiv:1707.02286*, 2017.

Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient Response Maps for Real-Time Detection of Texture-Less Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming Auto-Encoders. In *Proceedings of the International Conference on Artificial Neural Networks ICANN*, pages 44–51, 2011.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Sebastian Höfer, Manfred Hild, and Matthias Kubisch. Using Slow Feature Analysis to Extract Behavioural Manifolds Related to Humanoid Robot Postures. In *Proceedings of the International Conference on Epigenetic Robotics (EpiRob)*, pages 43–50, 2010.

Harold Hotelling. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, 24(6):417, 1933.

Marcus Hutter. Feature Reinforcement Learning: Part I: Unstructured MDPs. *Journal of Artificial General Intelligence*, 1(1):3–24, 2009.

Christian Igel and Michael Hüsken. Empirical Evaluation of the Improved RPROP Learning Algorithms. *Neurocomputing*, 50(1):105–123, 2003.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

Dinesh Jayaraman and Kristen Grauman. Learning Image Representations Equivariant to Ego-Motion. *arXiv:1505.02206*, 2015.

Odest Chadwicke Jenkins and Maja J. Matarić. A Spatio-Temporal Extension to ISOMAP Nonlinear Dimension Reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, page 56, 2004.

Nikolay Jetchev, Tobias Lang, and Marc Toussaint. Learning Grounded Relational Symbols from Continuous Data for Abstract Reasoning. In *Autonomous Learning Workshop at the IEEE International Conference on Robotics and Automation*, 2013.

Rico Jonschkowski and Oliver Brock. Learning Task-Specific State Representations by Maximizing Slowness and Predictability. In *Proceedings of the International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ER-LARS)*, 2013.

Rico Jonschkowski and Oliver Brock. State Representation Learning in Robotics: Using Prior Knowledge about Physical Interaction. In *Proceedings of Robotics: Science and Systems*, 2014.

Rico Jonschkowski and Oliver Brock. Learning State Representations with Robotic Priors. *Autonomous Robots*, 39(3):407–428, 2015.

Rico Jonschkowski and Oliver Brock. End-To-End Learnable Histogram Filters. In *Workshop on Deep Learning for Action and Interaction at the Conference on Neural Information Processing Systems (NIPS)*, 2016.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101(1):99–134, 1998.

Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-Net: Deep Learning for Planning under Partial Observability. In *Advances in Neural Information Processing Systems (NIPS)*, pages 4697–4707, 2017.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep Wariational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. *arXiv:1605.06432*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114*, 2013.

Ulrich Klank, Dejan Pangercic, Radu Bogdan Rusu, and Michael Beetz. Real-Time Cad Model Matching for Mobile Manipulation and Grasping. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 290–296, 2009.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, 2013.

Andrei N Kolmogorov. Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, 1(1):1–7, 1965.

George Konidaris and Andrew G. Barto. Efficient Skill Learning Using Abstraction Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1107–1112, 2009.

George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012.

Joseph B. Kruskal. Multidimensional Scaling by Optimizing Goodness of fit to a Nonmetric Hypothesis. *Psychometrika*, 29(1):1–27, 1964.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous Reinforcement Learning on Raw Visual Input Data in a Real World Application. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553): 436–444, 2015.

Yann A. LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

Robert Legenstein, Niko Wilbert, and Laurenz Wiskott. Reinforcement Learning on Slow Features of High-Dimensional Input Streams. *PLoS Computational Biology*, 6(8): e1000894, 2010.

Jan Leike and Marcus Hutter. Bad Universal Priors and Notions of Optimality. In *Proceedings of the Conference on Learning Theory (COLT)*, pages 1244–1259, 2015.

Karel Lenc and Andrea Vedaldi. Understanding Image Representations by Measuring their Equivariance and Equivalence. *arXiv:1411.5908*, 2014.

Timothée Lesort, Mathieu Seurin, Xinrui Li, Natalia Díaz Rodríguez, and David Filliat. Unsupervised State Representation Learning with Robotic Priors: A Robustness Benchmark. *arXiv:1709.05185*, 2017.

Sergey Levine and Vladlen Koltun. Guided Policy Search. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *arXiv:1504.00702*, 2015.

Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Large-Scale Data Collection. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, pages 173–184, 2016.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *arXiv:1509.02971*, 2015.

Henry W. Lin and Max Tegmark. Why Does Deep and Cheap Learning Work so Well? *arXiv:1608.08225*, 2016.

Michael L. Littman, Richard S. Sutton, and Satinder Singh. Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1555–1561, 2002.

Matthew Luciw and Juergen Schmidhuber. Low Complexity Proto-Value Function Learning from Sensory Observations with Incremental Slow Feature Analysis. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 279–287, 2012.

Sridhar Mahadevan and Mauro Maggioni. Proto-Value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research*, 8(10):2169–2231, 2007.

David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* MIT Press, 1982.

Roberto Martín-Martín, Sebastian Höfer, and Oliver Brock. An Integrated Approach to Visual Perception of Articulated Objects. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5091–5097, 2016.

Andreas Maurer. Bounds for Linear Multi-Task Learning. *Journal of Machine Learning Research*, 7:117–139, 2006.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis Function Adaptation in Temporal Difference Reinforcement Learning. *Annals of Operations Research*, 134(1):215–238, 2005.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to Navigate in Complex Environments. 2016.

Tom M. Mitchell. *The Need for Biases in Learning Generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ., 1980.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Andreas C. Müller and Sven Behnke. Learning Depth-Sensitive Conditional Random Fields for Semantic Segmentation of RGB-D Images. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6237, 2014.

Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal Deep Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 689–696, 2011.

Duy Nguyen-Tuong and Jan Peters. Model Learning for Robot Control: A Survey. *Cognitive Processing*, 12(4):319–340, 2011.

Masashi Okada, Luca Rigazio, and Takenobu Aoshima. Path Integral Networks: End-to-End Differentiable Optimal Control. *arXiv:1706.09597*, 2017.

Christopher Olah. Neural Networks, Types, and Functional Programming. `http://colah.github.io/posts/2015-09-NN-Types-FP/`, 2015. Accessed: 2018-02-25.

Sinno J. Pan and Qiang Yang. A survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and Generalization of Motor Skills by Learning from Demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 763–768, 2009.

Barak A. Pearlmutter and Jeffrey M. Siskind. Reverse-Mode AD in a Functional Framework: Lambda the Ultimate Backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7, 2008.

Justus Piater, Sébastien Jodogne, Renaud Detry, Dirk Kraft, Norbert Krüger, Oliver Kroemer, and Jan Peters. Learning Visual Representations for Perception-Action Systems. *International Journal of Robotics Research*, 30(3):294–307, 2011.

Rod Pierce. Sequences-Finding A Rule. `http://www.mathsisfun.com/algebra/sequences-finding-rule.html`, 2017. Accessed: 2017-09-21.

Tinca J.C. Polderman, Beben Benyamin, Christiaan A. De Leeuw, Patrick F. Sullivan, Arjen Van Bochoven, Peter M. Visscher, and Danielle Posthuma. Meta-analysis of the Heritability of Human Traits Based on Fifty Years of Twin Studies. *Nature genetics*, 47(7):702, 2015.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features Off-The-Shelf: An Astounding Baseline for Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 512–519, 2014.

Scott Reed and Nando de Freitas. Neural Programmer-Interpreters. *arXiv:1511.06279*, 2015.

C. Rennie, R. Shome, K. E. Bekris, and A. F. De Souza. A Dataset for Improved RGBD-based Object Detection and Pose Estimation for Warehouse Pick-and-Place. *arXiv:1509.01277*, 2015.

Martin Riedmiller. Neural Fitted Q Iteration-First Experiences With a Data Efficient Neural Reinforcement Learning Method. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 317–328, 2005.

Martin Riedmiller, Mike Montemerlo, and Hendrik Dahlkamp. Learning to Drive a Real Car in 20 Minutes. In *Proceedings of the International Conference Frontiers in the Convergence of Bioscience and Information Technologies (FBIT)*, pages 645 –650, 2007.

Bernardino Romera-Paredes, Andreas Argyriou, Nadia Berthouze, and Massimiliano Pontil. Exploiting Unrelated Tasks in Multi-Task Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 951–959, 2012.

Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, pages 832–837, 1956.

Sam T. Roweis and Lawrence K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, 290(5500):2323–2326, 2000.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, pages 1–42, 2015.

Radu Bogdan Rusu, Nico Blodow, and Beetz Michael. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3212–3217, 2009.

Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2155–2162, 2010.

Stefan Schaal. Dynamic Movement Primitives-A Framework for Motor Control in Humans and Humanoid Robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer New York, 2006.

Jonathan Scholz, Martin Levihn, Charles L. Isbell, and David Wingate. A Physics-Based Model Prior for Object-Oriented MDPs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

Abigail See. Deep Learning, Structure and Innate Priors-A Discussion between Yann LeCun and Christopher Manning. `http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html`, 2018. Accessed: 2018-02-28.

Tanmay Shankar, Santosha K. Dwivedy, and Prithwijit Guha. Reinforcement Learning via Recurrent Convolutional Neural Networks. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 2592–2597, 2016.

Roger N. Shepard. Toward a Universal Law of Generalization for Psychological Science. *Science*, 237(4820):1317–1323, 1987.

Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 1–15, 2006.

Olivier Sigaud, Clément Masson, David Filliat, and Freek Stulp. Gated Networks: An Inventory. *arXiv:1512.03201*, 2015.

Carlos N. Silla Jr. and Alex A. Freitas. A Survey of Hierarchical Classification Across Different Application Domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2010.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, and Andre Barreto. The Predictron: End-to-End Learning and Planning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3191–3199, 2017.

Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement Learning with Soft State Aggregation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 361–368, 1995.

Ray J Solomonoff. A Formal Theory of Inductive Inference Part I. *Information and Control*, 7(1):1–22, 1964.

Nathan Sprague. Predictive Projections. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1223–1229, 2009.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Shiliang Sun. A Survey of Multi-View Machine Learning. *Neural Computing and Applications*, 23(7-8):2031–2038, 2013.

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

Michael J. Swain and Dana H. Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep Neural Networks for Object Detection. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2553–2561, 2013.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value Iteration Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2154–2162, 2016.

Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A Physics Engine for Model-Based Control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

Marc Toussaint. Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1930–1936, 2015.

Amos Tversky and Daniel Kahneman. Judgment under Uncertainty: Heuristics and Biases. *Science*, 185(4157):1124–1131, 1974.

Harm van Seijen, Shimon Whiteson, and Leon J. H. M. Kester. Efficient Abstraction Selection in Reinforcement Learning. *Computational Intelligence*, 30(4):657–699, 2014.

Vladimir Vapnik and Rauf Izmailov. Learning Using Privileged Information: Similarity Control and Knowledge Transfer. *Journal of Machine Learning Research*, 16:2023–2049, 2015.

Vladimir Vapnik and Akshay Vashist. A New Learning Paradigm: Learning using Privileged Information. *Neural Networks*, 22(5-6):544–557, 2009.

Wei Wang and Zhi-Hua Zhou. A New Analysis of Co-Training. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1135–1142, 2010.

Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On Deep Multi-View Representation Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1083–1092, 2015.

Manuel Watter, Jost Tobias Springberg, Joschka Boedecker, and Martin Riedmiller. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2746–2754, 2015.

Paul J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep Learning via Semi-supervised Embedding. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1168–1175, 2008.

Laurenz Wiskott and Terrence J. Sejnowski. Slow Feature Analysis: Unsupervised Learning of Invariances. *Neural Computation*, 14(4):715–770, 2002.

David H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

Andy Zeng, Kuan-Ting Yu, Shuran Song, Daniel Suo, Ed Walker Jr., Alberto Rodriguez, and Jianxiong Xiao. Multi-View Self-Supervised Deep Learning for 6D Pose Estimation in the Amazon Picking Challenge. *arXiv:1609.09475*, 2016.

Jiaping Zhao and Laurent Itti. Improved Deep Learning of Object Category using Pose Information. *arXiv:1607.05836*, 2015.