Learning Structured Task Related Abstractions

Svetlin Penkov

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2019

Lay Summary

As robots and autonomous agents are to assist people with more tasks in various domains they need the ability to quickly gain contextual awareness in unseen environments and learn new tasks. Currently, autonomous agents tend to utilise data driven learning methods, such as neural networks, which do not allow them to make generalisations and transfer their knowledge to new tasks and environments.

In this thesis we study how to learn task related abstractions and improve the generalisation capabilities of autonomous agents. In particular, we propose to decompose the process of executing a task into two stages: i) perceiving important information from the world and ii) making decisions in order to take an action. We posit that neural networks are suitable for the first stage and computer programs are the most appropriate choice for the second stage.

In the first part of the dissertation, we address the problem of learning to extract meaningful patterns, or symbols, from sensory data assuming that the agent has access to a program for decision making. We first describe a framework applicable to collaborative human-robot interaction setups and then introduce a general algorithm for learning symbols from sensory data which we prove to work with any program for decision making. Our experiments show that programs are not only a suitable tool for representing decision making processes, but also provide autonomous agent with the capabilities to learn useful task related symbols.

In the second part of the thesis, we address the complement problem of learning useful programs assuming that the agent can recognise meaningful patterns from sensory signals. We introduce a novel approach for efficiently learning programs and demonstrate how they can be used to interpret black-box models such as neural networks as well as to discover important task abstractions.

Abstract

As robots and autonomous agents are to assist people with more tasks in various domains they need the ability to quickly gain contextual awareness in unseen environments and learn new tasks. Current state of the art methods rely predominantly on statistical learning techniques which tend to overfit to sensory signals and often fail to extract structured task related abstractions. The obtained environment and task models are typically represented as black box objects that cannot be easily updated or inspected and provide limited generalisation capabilities.

We address the aforementioned shortcomings of current methods by explicitly studying the problem of learning structured task related abstractions. In particular, we are interested in extracting symbolic representations of the environment from sensory signals and encoding the task to be executed as a computer program. We consider the standard problem of learning to solve a task by mapping sensory signals to actions and propose the decomposition of such a mapping into two stages: i) perceiving symbols from sensory data and ii) using a program to manipulate those symbols in order to make decisions. This thesis studies the bidirectional interactions between the agent's capabilities to perceive symbols and the programs it can execute in order to solve a task.

In the first part of the thesis we demonstrate that access to a programmatic description of the task provides a strong inductive bias which facilitates the learning of structured task related representations of the environment. In order to do so, we first consider a collaborative human-robot interaction setup and propose a framework for Grounding and Learning Instances through Demonstration and Eye tracking (GLIDE) which enables robots to learn symbolic representations of the environment from few demonstrations. In order to relax the constraints on the task encoding program which

GLIDE assumes, we introduce the perceptor gradients algorithm and prove that it can be applied with *any* task encoding program.

In the second part of the thesis we investigate the complement problem of inducing task encoding programs assuming that a symbolic representations of the environment is available. Therefore, we propose the π-machine – a novel program induction framework which combines standard enumerative search techniques with a stochastic gradient descent optimiser in order to obtain an efficient program synthesiser. We show that the induction of task encoding programs is applicable to various problems such as learning physics laws, inspecting neural networks and learning in human-robot interaction setups.

"I am going to state the law of gravitation in three different ways, all of which are exactly equivalent, but also completely different once we try to reason with them."
— Richard Feynman

Acknowledgments

I have many people to thank for all these years full of discoveries:

First and foremost, my supervisor Subramanian Ramamoorthy. For he provided me not only with enlightening academic guidance and advice, but also encouraged me to explore the broad world of science.

I thank my family for providing me with the opportunity to follow my lifelong passion of building robots and Victoria for not letting me forget my dreams even in the toughest moments!

Many thanks go to all students from the RAD group - a truly unique mixture of brilliant people. Specifically thanks to Alex for steering me in the right path during my early PhD years and for keeping the greatest robotics lab running for us. I thank my dear friends and colleagues Daniel, Manny and Yordan for the numerous spontaneous and inspiring discussions - I hope there will be many more to come! I would also like to thank Stan for always being one step ahead of me.

This work was supported in part by grants EP/F500385/1 and BB/F529254/1 for the University of Edinburgh School of Informatics Doctoral Training Centre in Neuroinformatics and Computational Neuroscience (www.anc.ac.uk/dtc) from the UK Engineering and Physical Sciences Research Council (EPSRC), UK Biotechnology and Biological Sciences Research Council (BBSRC), and the UK Medical Research Council (MRC).

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Svetlin Penkov

Publications

The following publications have been composed during the course of this doctorate:

S. Penkov, A. Bordallo, S. Ramamoorthy. Inverse Eye Tracking for Intention Inference and Symbol Grounding in Human-Robot Collaboration. In Proc. *Robotics: Science and Systems Workshop on Planning for Human-Robot Interaction* (RSS PHRI), 2016. [164]

S. Penkov, A. Bordallo, S. Ramamoorthy. Physical Symbol Grounding and Instance Learning through Demonstration and Eye Tracking. In Proc. *IEEE International Conference on Robotics and Automation* (ICRA), 2017. [165]

S. Penkov, S. Ramamoorthy. Using Program Induction to Interpret Transition System Dynamics. In Proc. *International Conference on Machine Learning Workshop on Human Interpretability in Machine Learning* (ICML WHI), 2017. Best paper runner-up. [166]

Y. Hristov, **S. Penkov**, A. Lascarides, S. Ramamoorthy. Grounding Symbols in Multi-Modal Instructions. In Proc. *Association for Computational Linguists Workshop on Language Grounding for Robotics* (ACL LGR), 2017. [92]

S. Penkov, S. Ramamoorthy. Learning Programmatically Structured Representations with Perceptor Gradients. *International Conference on Learning Representations* (ICLR), 2019. [167]

To my family.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
DSL	Domain Specific Language
DQN	Deep Q-value Network
DRL	Deep Reinforcement Learning
ETG	Eye Tracking Glasses
FSM	Finite State Machine
GAN	Generative Adversarial Network
GLIDE	Grounding and Learning Instances through Demonstration and Eye tracking
HRI	Human-Robot Interaction
HMM	Hidden Markov Model
LQR	Linear Quadratic Controller
MDP	Markov Decision Process
MRF	Markov Random Field
NMN	Neural Module Networks
NTM	Neural Turing Machine
POMDP	Partially Observable Markov Decision Process
PDA	Pushdown Automaton
RL	Reinforcement Learning
RNN	Recurrent Neural Network

SIFT	Scale-invariant Feature Transform
SLAM	Simultaneous Localisation and Mapping
VAE	Variational Autoencoder

1

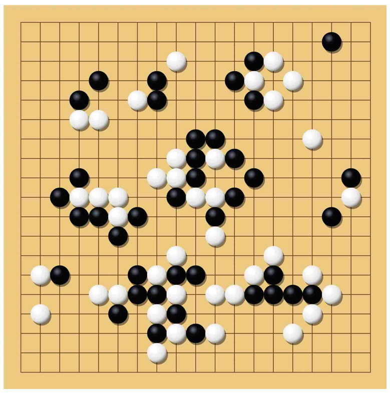
Introduction

1.1 Preface

Autonomous agents have gained unprecedented capabilities with the adoption of data driven learning methods. On one hand, virtual autonomous agents have achieved impressive accomplishments such as defeating the world masters in games with vast search spaces amongst which are chess [34] and Go [194]. On the other hand, while embodied agents have certainly gained from data driven learning methods [5, 132, 133], they still struggle with even simple tasks such as going through a door or turning a valve shut [15]. Long awaited aspirations such as building robotic assistants and co-workers or colonising other planets with cohorts of autonomous robots are even further away from becoming a reality.

So where does this disparity in capabilities arise from? It is somewhat counter-intuitive that autonomous agents outperform people in mentally challenging tasks, while everyday mundane tasks turn out to be beyond their current capabilities. In order to investigate the matter, let us consider two extremely different tasks. The first one is playing Go on a computer against a human and the other one is assisting a human to put together a mechanical assembly in a collaborative Human-Robot Interaction (HRI) setup. The two tasks, which are depicted in Fig. 1, differ significantly from each other in multiple aspects, some of which more subtle than others.

First of all, playing computer Go requires just a computer program, while assisting a human requires building and controlling a physical robot. We are not interested in comparing virtual versus physical systems, so let us assume that we have access to a high quality hardware setup which enables us to process sensory signals and provide



(a) Computer Go. Adapted from [41].

(b) Collaborative assembly. Adapted from [199].

Figure 1: Example tasks.

motion commands through a computer program as well. Now the two problems are equivalent to writing a computer program for each of the tasks.

Secondly, each agent should be able to sense the environment in order to make any decisions. Sensing in computer Go is simply reading the state of the board from memory. The information which the agent obtains from doing so encodes which positions on the board have stones on them or not. In just describing what the agent can sense we have already used concepts such as *board positions* and *stones* that are important for the game. On the other hand, in the HRI task we can access only raw data from the sensors, e. g. cameras, in the hardware setup. Concepts important for the completion of the task, such as *tool position* and *assembly orientation*, cannot be sensed directly in less structured environments, especially when people operate in the same environment. Therefore, the HRI task requires great levels of contextual awareness in order to perceive important concepts related to the task.

Once all the necessary information from the environment has been extracted through sensing and perception, each agent should be able to make decisions about what actions to execute. Decision making is usually a computationally heavy process, especially in domains such as Go, where the size of the state space is so large. In contrast, once the symbols relevant to the task are perceived in the HRI setup, it is simply a matter of planning in a relatively small search space in order to complete the task. Therefore, decision making requires much greater reasoning capabilities for playing Go than for executng a physical task in a collaborative human-robot interaction setup.

Even though both tasks are non-trivial, the complexity emerges at different levels - Go poses a hard searching problem, while collaborative human robot interaction requires the ability to acquire contextual awareness. Given that autonomous agents can play Go better than the best human experts, we can assert that one of the main challenges for developing more capable autonomous agents is to enable them to learn task related concepts more efficiently from raw sensory signals. In general, those concepts can represent objects, relations or any other entity of interest and we refer to them as *symbols*. Consequently, if an agent can recognise task related symbols in the environment, then a task can be viewed as a symbol manipulation process resulting in a desired configuration. Furthermore, given that the set of necessary task related symbols can be perceived, a *program* can represent any task as it is the most powerful apparatus for manipulating symbols. In this dissertation we study how symbolic representations and programs can be utilised by autonomous agents to provide them with the capabilities of learning structured task related abstractions.

Symbols and programs provide powerful abstractions that can potentially express any task and enabling the agent to extract knowledge transferable to both new environments and new tasks. In order to demonstrate the sort of abstractions resulting from the use of symbols and programs to describe a task let us consider the simple physical task "*screw in the bolt*".

1.1.1 *Programs as Task Abstractions*

Usually, when we consider the execution of such a task, we tend to think about planning and determining a sequence of primitive actions which solve the task when executed. A plan is simply a sequence of primitive actions that can be executed one after another, which would ideally solve the task. However, the plans for solving even such a simple task can be quite different from one another if different initial states of the environment are considered. So it must be the case that plain sequences of actions are not sufficient to truly capture a task and a more general, state invariant representation is needed.

In this thesis, we take the view that computer programs offer sufficiently powerful mechanisms of abstraction for representing tasks in general. Ultimately, once executed, a program defines a sequence of instructions. However, through composition, branching and looping, programs can capture any underlying task structure at different abstraction levels of various complexity, depending on the available symbols.

If we describe the task of interest, namely "*screw in the bolt*", as a program that takes as an input the current pose of a bolt and generates a sequence of actions that screw in the bolt then we can execute the same program in any environment, provided that the required input symbols are available. Each execution might produce a different sequence of actions, but achieving the same desired final state. Therefore, if an autonomous agent can learn the programmatic structure of a task, rather than repeating a fixed sequence of actions, obtained, for example, from demonstrations, it should be able to transfer the knowledge captured in a particular setting to other settings.

1.1.2 *Symbols as Environment Abstractions*

The aforementioned program that solves the task "*screw in the bolt*" can only be executed if the robot is able to perceive the symbol "*bolt*" and in particular its current pose in the environment. Importantly, task encoding programs rely not only on the existence of the symbols they operate on, but also on the ability to recognise instances of those symbols in the environment. In other words, if the robot cannot ground the symbol "*bolt*" to the corresponding physical entity in the environment then it will not be able to execute any program which contains that symbol.

Furthermore, there is an infinite number of symbols that can be extracted and learnt from the environment, however only some of them are related to the task and even fewer are useful for the completion of the task. Given that a task encoding program is dependent on the availability of certain symbols, it forms a natural inductive bias that can be exploited for learning to recognise the right set of symbols.

Representing the environment only with symbols of direct relevance to the task not only abstracts away large amounts of the irrelevant raw sensory data, but also

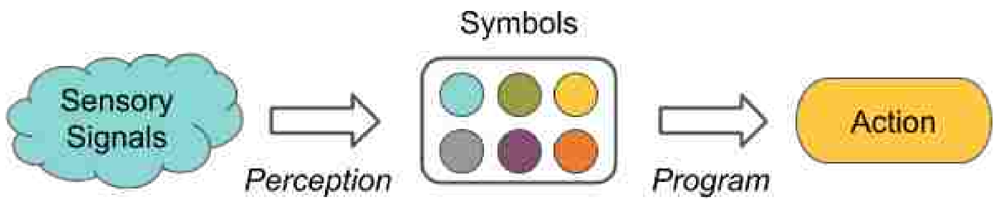


Figure 2: The decomposition of a typical task learning problem considered in this thesis.

enables generalisation to other tasks. For example, recognising the pose of a bolt in the environment is required for screwing it in as well as for unscrewing it, loosening it or tightening it.

Interestingly, there is a strong interdependency between learning to recognise symbols within the environment and learning programs to perform decision making – symbols determine what programs can be executed, while programs determine what symbols are useful to be learnt. Similar ideas in philosophy and linguistics can be traced back to Wittgenstein [224] who suggests that "the meaning of a word is its use in the language". From an Artificial Intelligence (AI) perspective, learning to recognise symbols is considered to be a pattern recognition problem, while learning programs is known as program induction and the two fields have very little overlap. Recent interests in bridging this gap have resulted in interesting research approaching the problem from different perspectives [63, 64, 70, 103, 216]. In this thesis, we focus on studying the interplay between learning task related symbols and programs for improving the capabilities of autonomous agents.

1.2 Problem Statement

We consider the standard problem of learning to solve a task by mapping sensory data to actions. We are interested in the decomposition of such a mapping, as shown in Fig. 2, into two stages: i) perceiving symbols from sensory data and ii) using a program to manipulate those symbols in order to make decisions. The overall goal of this thesis is to study the bidirectional interactions between the perception capabilities of an agent and the programs it can execute in order to solve a task.

In particular, this thesis tackles the following questions:

- Assuming an agent has access to a program working with abstract symbols for decision making, how can it learn to recognise the required symbols from sensory signals?
- Assuming an agent is able to recognise symbols from sensory signals, how can it learn a program representing a certain task?
- How can structured task related abstractions, i. e. symbols and programs, be leveraged for applications to human-robot interaction and, in particular, learning from demonstration?

1.3 Thesis Overview

The thesis is divided into two logical parts. The first one studies the question of grounding symbols to their instances in the environment from sensory signals when a program for decision making is available, while the second one studies the complement problem of learning new programs when symbol instances can be recognised within the environment.

1.3.1 *Physical Symbol Grounding*

Fundamentally, the problem of symbol grounding is related to the philosophical issue of what is the meaning of a symbol and where does it emerge from. We, however, take a more pragmatic perspective and, following the reasoning proposed by Vogt [219], assume that symbols gain their meaning from the physical entities they refer to in the environment. That leads to the well defined problem of physical symbol grounding – matching symbols to their instances in the environment. In general, symbols do not have to refer only to physical objects, they can, in fact, refer to any pattern in the sensory signals. For example, in chapters 3 and 4 we demonstrate that it is possible to ground symbols corresponding to the position of objects.

The simplest possible way to perform physical symbol grounding is to formulate the problem within a supervised learning framework and label a sufficient amount of sensory data. Supervised learning, though, requires the tedious and expensive process of data annotation and it cannot be applied to online situations, where the agent is to learn from new sensory data and experiences.

Therefore, we propose another approach to physical symbol grounding which exploits programs that can be executed using those symbols. The main hypothesis which we consider in the first part of this thesis is:

Task encoding programs provide autonomous agents with physical symbol grounding capabilities.

By task encoding programs we simply mean programs that implement a decision making procedure for choosing an action. The source of complexity in many important and interesting problems, such as collaborative HRI, is in perceiving task related symbols, rather than manipulating them. In such cases a decision making program can be easily designed. For example, the program for screwing in a bolt could follow the steps: i) move close to the head of the bolt, ii) align with its axis, iii) insert tool, iv) apply torque until a certain threshold is reached.

In order to investigate our hypothesis, we first consider a collaborative HRI setup, where a robot is supposed to learn to perform a task by observing a human demonstration. Since, it is natural for people to work with abstract plans that are an intuitive and concise way to represent a task we assume that a task encoding program is readily available. The main challenge thus becomes how to relate the program to the sensory signals acquired during the demonstration. In chapter 3, we study exactly this problem and demonstrate that it is possible to learn to recognise multiple symbols in the environment even from a single human demonstration.

For this purpose, we propose a framework for Grounding and Learning Instances through Demonstration and Eye tracking (GLIDE). Importantly, we utilise multiple sensory signals including several video cameras together with eye tracking information. Extensive amounts of research on oculomotor control during task execution, have revealed that most of the gaze fixations are positioned on locations and items of relevance to the task [84, 125, 185]. Inspired by the regularities of oculomotor control during task execution, we capture data from eye tracking glasses which the human demonstrator wears.

We combine fixation traces with the task encoding program into a *fixation program* and exploit certain properties in order to align the demonstration with the acquired sensory signals. We pose the problem as a Bayesian inference problem over the structure of a graphical model and provide an efficient inference algorithm. A key aspect of our work is that we introduce a novel method for estimating physical 3D locations of visual fixations within the environment which enables the appearance of symbol instances to be learnt. Instance learning is a crucial ability when the robot does not have any knowledge about the model or the appearance of the symbols referred to in the program. We report the results obtained from human experiments and demonstrate that GLIDE successfully performs physical symbol grounding and learns the appearance of symbol instances, thus enabling robots to autonomously execute tasks in initially unknown environments.

One of the key assumptions enabling efficient physical symbol grounding in GLIDE is that the task encoding program in practice represents a plan which is a sequence of actions and does not have any conditional branches or loops. Additionally, eye tracking information serves as a strong supervisory signal which is potentially available only if a person is part of the learning process. These assumptions limit the class of tasks to which GLIDE can be applied to.

Therefore, in chapter 4, we relax those assumptions and consider the general problem of grounding symbols that could serve as an input to *any* program without the need for a person to participate in the learning stage. In order to do this, we pose the question of physical symbol grounding into a reinforcement learning framework and demonstrate that programs of arbitrary complexity can be considered.

In particular, we decompose the policy into a neural network based perception module and a decision making program. We call the neural network which learns to recognise symbols from raw sensory signals a *perceptor* and introduce the perceptor gradients algorithm. We provide a proof that regardless of the decision making program the perceptor can be trained through a REINFORCE estimator [223] in a standard policy gradients setup.

We apply the perceptor gradient algorithm to a set of problems ranging from control of dynamic systems to solving tasks in a custom 2.5D Minecraft environment and demonstrate that it enables neural networks to learn semantically meaningful latent spaces which encode the task related symbols of interest. In contrast to other unsupervised learning methods, our method avoids the problems of non-identifiability and each latent dimension has definite meaning imposed by the task encoding program. Furthermore, by recognising the similarities between the policy gradients setup and the Variational Autoencoder (VAE) [109] we show that it is also possible to generate observations from semantically meaningful latent codes.

1.3.2 *Program Induction*

In the second logical part of this thesis we consider the complement problem of inducing programmatic structure over a set of symbols which can be recognised within the surrounding environment. In particular, we are interested in inducing programs from observation traces, generated by processes occurring in the environment such as an object moving due to the law of physics or a human demonstrating a task.

Explaining and reasoning about processes which underlie observed black-box phenomena enables the discovery of causal mechanisms, derivation of suitable abstract structures and the formulation of more robust predictions. Therefore, instead of simply learning a mapping from the observed state to the observed transition, we propose to learn high level functional programs in order to represent abstract models which capture the invariant structure in the observed data. Thus, the working hypothesis in the second part of the thesis is:

Physically grounded symbols enable autonomous agents to learn task encoding programs efficiently.

We explore this hypothesis in chapter 5 and introduce the π-machine (program-induction machine) – an architecture able to induce interpretable LISP-like programs from observed data traces. We propose an optimisation procedure for program learning based on backpropagation, gradient descent and A* search. We describe an efficient method for extending any gradient descent optimiser to support symbolic variables by assuming the existence of an ordering and exploiting properties of KD-trees.

We apply the proposed method to three problems. Firstly, we consider the problem of system identifcation of dynamical systems and demonstrate that the π-machine is able to induce programs in this domain faster than state of the art methods [188]. Secondly, we demonstrate how learning task related programs can be applied to the problem of explaining the behaviour of a Deep Q-value Network (DQN) agent. Thirdly, we show that when the π-machine is utilised in a learning from demonstration setting, it automatically yields programs that capture abstract knowledge such as spatial relations that are crucial for the domain of collaborative HRI. Lastly, our experimental results show that the π-machine can efficiently induce interpretable programs from individual data traces in any of the considered domains.

1.4 Major Contributions

The key contributions of this thesis are:

- Robotics (HRI): A novel method for 3D eye tracking using eye tracking glasses and monocular Simultaneous Localisation and Mapping (SLAM).
- Robotics (learning from demonstration): A physical symbol grounding framework based on fixation programs and efficient Bayesian inference over the structure of a graphical model.
- Machine Learning (representation learning): A general algorithm for learning symbolic representations from raw data required as an input to a given program.
- Artificial Intelligence (program induction): An algorithm for inducing programs from observed traces that combines classical search techniques with stochastic gradient descent optimisation.

We present the first two contributions in chapter 3 and demonstrate that the GLIDE framework provides robots with the capability of physically grounding symbols to raw sensory signals by observing very few task demonstrations, which to our knowledge has not been demonstrated before. We describe our third contribution in chapter 4 and show that the perceptor gradients algorithm not only enables autonomous agents to learn symbolic representations, but it also outperforms agents trained within the standard Reinforcement Learning (RL) loop with policy gradients on the tasks that we consider. Furthermore, the perceptor gradients algorithm does not impose any independence assumptions on the latent space and so we show that it can cope with a larger class of representation learning problems which widely accepted models such as the βVAE struggle with. Finally, we present our fourth contribution in chapter 5 and demonstrate that the π-machine is much more computationally efficient than state of the art methods for inducing programs from observation traces. We also describe a novel application of the proposed program induction method to explaining the behaviour of a DQN agent. Each one of the contributions presented in this thesis is a step towards bridging the gap between symbolic and subsymbolic methods in an attempt to design better learners capable of extracting symbolic knowledge representations and causal mechanisms from raw sensory signals.

2 Background

This thesis focuses on the problems of learning and utilising two types of task related structured abstractions:

1. Symbolic representations of the environment
2. Programmatic representations of the task

Therefore, we present in this chapter a broad overview of the concepts and work related to the topics of learning symbolic representations and inducing programmatic structure. Those questions rest at the core of artificial intelligence and so have been actively explored from various perspectives ever since the field was founded.

Additionally, we include a section on learning structured abstractions in the particular context of task learning. We consider the problem of learning task related abstractions from the perspectives of reinforcement learning and learning from demonstration with an emphasis on some of the recent neuro-symbolic methods.

The chapter concludes with a discussion of the equivalences which can be found between some of the introduced concepts and that form the conceptual core of this dissertation. A detailed discussion of the work directly related to each of the major contributions of this thesis is presented within the context of each corresponding chapter.

2.1 Symbolic Artificial Intelligence

2.1.1 *Symbols*

Symbolic artificial intelligence methods are based on the assumption that processes associated with intelligence can be simulated by *manipulating* a set of *symbols* [82, 150]. Symbols can be thought of as tokens which represent, or refer to, any entity, pattern or concept. By performing various operations with those symbols such as modifying, replacing or concatenating one can describe a large class of problems. In fact the physical symbol hypothesis made by Newell and Simon [150] suggests that such symbolic systems have the "sufficient and necessary means for general intelligent action". Symbolic AI is based on the theoretic apparatus of formal systems.

2.1.2 *Formal Systems*

The emergence of symbolic AI can be traced back to the development of the field of metamathematics and the study of formal systems [110]. A formal system can be used to derive theorems by the repeated application of inference rules to a set of axioms. Each axiom and any derived theorem must be well formed formulae within the system and so they must be in compliance to a given grammar and alphabet of primitive symbols. In general, a formal system can be represented as the tuple (Σ, Γ, P, A) where Σ is the alphabet of primitive symbols, Γ is a grammar which describes how primitive symbols can be combined in order to obtain well formed formulae. P is a set of rules, any of which can transform a well formed formula into another one. If a rule is applied to a well formed formula that is an axiom, i. e. it is in the set A, then the resulting formula is also well formed and called a theorem. First order logic, polynomial algebra and planar geometry are all examples of formal systems that we are used to. A general purpose computer can emulate any formal system by running a suitable *program*.

2.1.3 *Programs*

In 1937 Alan Turing proposed a universal abstract apparatus, known as the Turing machine, [211] which enabled an infinite number of computations. In fact, any problem which cannot be solved by a Turing machine is not solvable by any other algorithm and so the Turing machine defines the theoretical bounds of what is possible to compute and what not. An interesting question arises from this fact – how can a simple machine following a particular set of rules be able to execute any computation possible? Turing designed his machine such that it would take as an input not only numbers to process, but also a sequence of instructions to be executed on those numbers. Thus the Turing machine is able to follow any valid sequence of instructions fed in as an input. Nowadays we refer to such sequences as programs.

Programs have been at the core of AI since its beginning. In personal correspondence, John McCarthy describes the Dartmouth conference, regarded by many as the seminal event for AI, as [111]:

> *"At present it looks as though we shall concentrate on a problem of devising a way of programming a calculator to form concepts and to form generalizations."*

That description could be interpreted in two different ways one of which seems to be more valid than the other. At one level, "programming a calculator" suggests that the problem of pattern recognition can be solved from a purely computer science perspective resulting in a pattern recognising program. However, state of the art performance in such domains, e. g. computer vision, is currently achieved by neural networks. So it seems that thinking in terms of classic computer science, rule based reasoning and programming does not offer the right set of abstractions. At another level, though, we do program neural networks. Once the notions of layers, activation functions and computational graphs are introduced, programs become an intuitive way to represent them. In fact, any AI model or framework that has been run on a computer has been boiled down at some point to a mere programming task.

Why are programs so powerful? A program encodes a sequence of instructions either by directly listing them or by utilising one of the following structures:

1. *Conditional branching* is necessary for any sort of reasoning.

2. *Loops* enable repetition until a certain condition holds.

3. *Compositionality* enables programs to express arbitrary hierarchies.

Those 3 structures are at the crux of any non-trivial reasoning processes and enable programs to express virtually any problem, but only provided that symbols at the right level of abstraction are available.

As discussed in the previous chapter, a task cannot be expressed just as a sequence of actions (except for simple cases), because different initial conditions and events in the environment would require different sequences. Programs, with their capability to express conditional branching, loops and composition, are particularly well suited for abstracting away details about individual sequences of actions and can serve as general task descriptors. More interestingly, programs can abstract away details about other programs which fits naturally with the idea of options (i.e. temporal abstractions) and composite tasks [18]. Moreover, if a new task is to be considered, compositionality becomes especially advantageous, as the new task encoding program can reuse previous programs or fragments from them. Valkov et al. [214] explore such a setup and demonstrate that programs are extremely well fitted to the problem of lifelong learning. It should also be noted that sub-programs capture atomic procedural knowledge which is correct irrespective of the particular instance of a given problem. Therefore, sub-programs can easily be transferred to new domains, or even new problems [50, 65].

2.1.4 *Transition Systems*

When dealing with programs and their execution one usually resorts to the formalism of transition systems.

A labelled transition system Ω is the tuple $(\mathcal{S}, \mathcal{A}, \delta)$ where $\mathcal{S}$ is a non-empty set of possible states, $\mathcal{A}$ is a non-empty set of labelled transitions and $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function. By starting from an initial state $s_0 \in \mathcal{S}$ and applying the state transition function N times with some chosen transitions $a_0, a_1, \ldots, a_{N-1}$, $a_i \in \mathcal{A}$, one will obtain a sequence of states $s_0, s_1, \ldots, s_N$, $s_i \in \mathcal{S}$, called a trace and denoted as $\mathcal{T}$. Typically, the sequence of transitions is encoded in a program ρ and can be obtained by executing the program. The formalism of transition systems is used to describe the theoretical foundations of the π-machine in chapter 5.

2.1.5 *Limitations*

Theoretically, programs processing symbols should be able to solve any problem, however the ideas behind symbolic AI sound naive from a contemporary perspective as there are fundamental limitations which have not been overcome yet, especially when considering learning from raw data.

Firstly, symbol manipulation makes the important assumption that a dictionary of symbols already exists, however it is far from clear how to generate such a dictionary as symbols are meant to carry semantic content. How can one recognise the existence of symbols from raw data or amongst other less abstract symbols? Typically, one studies the problem of interest and carefully designs a set of symbols which can express both the problem and its corresponding solution. Assuming that the necessary symbols are defined during the design stage introduces another equally hard problem known as the symbol grounding problem [80, 190, 219]. Given a set of abstract symbols, how can they be related to the corresponding entities, concepts and patterns in the raw data they stand for? Usually, the symbol grounding problem is associated with natural language understanding and determining what a particular utterance refers to, however it is a general problem occurring throughout the whole field of artificial intelligence whenever an abstract language is utilised, not just natural language.

Secondly, the ability to manipulate symbols assumes that the rules for manipulation are also already available. Thus, similar issues arise. What is the origin of these rules? They must describe meaningful operations and process only symbols from the set of available ones in order to ensure that the solution of the problem can be obtained within the symbolic system. Therefore, the process of designing rules for manipulating symbols also requires manual intervention by the designer. Due to their hand crafted nature, symbolic systems scale poorly in terms of complexity. Expert systems are a clear example for that, as after a while it becomes impossible to add new rules in order to ensure an up-to-date and exhaustive knowledge representation [161].

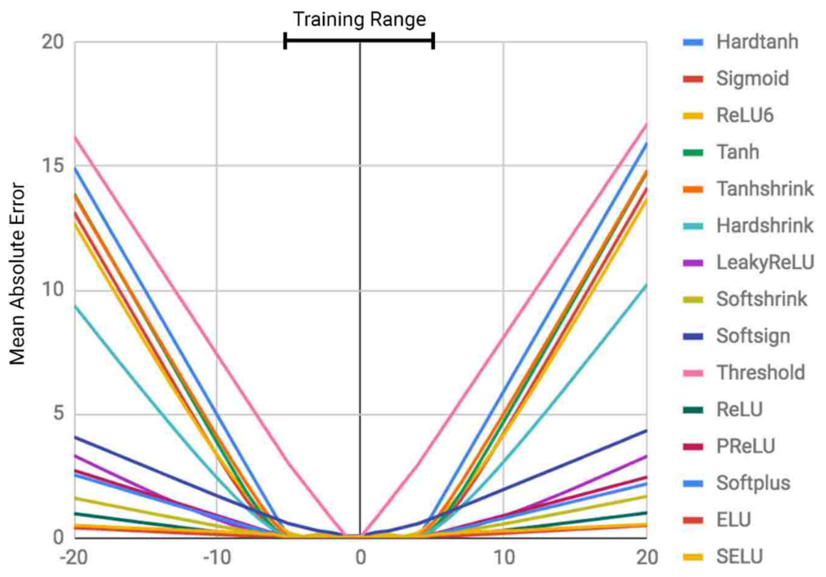


Figure 3: Extrapolation behaviour of a simple neural network trained on the identity function. Each curve corresponds to a different activation function being used. The figure is adapted from [208].

2.1.6 *Symbolic Inductive Bias*

Despite the well known limitations of pure symbolic approaches, recent research efforts have clearly demonstrated that symbols and programs provide strong inductive bias that improves the performance of universal data driven learners such as neural networks.

One of the key limitations of data driven learners is their tendency to memorise the training dataset, rather than finding rules to compute the output from a given input. This problem is particularly exacerbated by the large capacity of deep neural networks. Trask et al. [208] illustrate this issue on a small neural network trained on the simple problem of learning the identity function (see Fig. 3). While the error withing the training range is close to zero for many of the activation functions, extrapolation leads to unbounded errors.

Emulating various symbol manipulating automata such as a push down automaton [72], a Turing machine [71, 72] or a random access machine [121] with neural networks leads to much better extrapolation capabilities. For example, Grefenstette et al. [72], demonstrate that and an Recurrent Neural Network (RNN) augmented with a neural queue or deque can obtain perfect performance in tasks such as copying a sequence, reversing a sequence or bigram flipping with inputs orders of magnitude longer than the ones in the training data. Similar results have been reported by Trask et al. [208] who extend RNNs with arithmetic logic units and Cai, Shin and Song [32] who propose a recursion-based strategy for evaluating the output of an RNN. Hybrid neuro-symbolic learners have been demonstrated to be robust to noisy or mislabelled data, while still learning general rules in a data efficient manner [55].

Another common problem with neural networks is catastrophic forgetting that occurs when a neural network is to learn a set of tasks. Recent research indicates that it is beneficial to combine neural network based perception modules with symbolic representations in order to obtain state of the art lifelong learning systems capable of generalising to unseen problem instances [14, 50, 65, 214].

Symbolic inductive bias is applicable and beneficial even to perception heavy tasks such as character recognition [123], 3D shape recognition [205] and visual question answering [10, 11, 93]. Kolouri et al. [114] show that jointly training over pixels and symbolic representations enables zero shot learning for image classification tasks. Capturing the programmatic structure of visual stimuli enables the generation of unseen before, but nevertheless valid, images [51, 233]. Inverse graphics models that contain data driven learners and a set of rendering programs also achieve state of the art performance in various computer vision tasks [51, 118, 120, 142, 226].

The position which we take in this thesis is that learning symbols or rules (programs) individually is bound to yield unsatisfactory results. There is an intricate interaction between what symbols are useful and what rules can be applied. Therefore, in chapters 3 and 4 we study how programs can guide the learning of symbolic representations, while chapter 5 focuses on the complement question of inducing programs by taking into account properties of the available symbols.

2.2 Representation Learning

Representation learning is a branch of machine learning which studies the topics of learning structured representations from raw data. In particular, representation learning methods attempt to extract useful features from the raw data which improve the performance of a consequent classifier or a predictor [20].

Typically, the problem of representation learning is framed as an unsupervised learning problem where a mapping f is to be found such that input data x is mapped to a feature vector z, or $f(x) = z$. The most general formulation of the problem does not put any restrictions on the domains of x and z or the feature extracting function f.

In practice, depending on the problem under consideration, various assumptions about the data domain, feature space or mapping function are made. Bengio, Courville and Vincent [20] provide a detailed list of a wide range of assumptions explored by the community, however in this section we focus on the ones most relevant to the work in this thesis.

Multiple data generating factors

The unknown underlying distribution from which the observed data is generated has multiple factors which affect the data. For example, the pixel values of an image are influenced by factors such as the number of objects in the image, the position, shape and size of each object, global lighting conditions, texture and surface reflectivity of each object. In general, recovering the exact configuration of the factors that generated the data could be an ill-posed problem as the process of generation could result in a surjective mapping. Therefore, various regularisation techniques, which we discuss in the next sections, are used in practice.

Hierarchical composition

Representations could be built either directly from data or by composing other less abstract representations. Deep learning methods exploit this assumption heavily by stacking large number of layers, each of which produces representations at a certain abstraction level [127]. Simply composing features, however, does not guarantee that one would arrive at an abstraction that is useful for the task at hand. As noted in [20], what good criteria are for learning a useful feature space is currently an open problem.

Shared representations

Good representations should be useful for multiple tasks. If the extracted features from the raw data carry some semantic content, i. e. they have a symbolic structure, then it is quite likely that these symbols are transferable to another tasks. For example, as discussed in chapter 1, learning to recognise the pose of a bolt is applicable to multiple tasks that involve bolts.

Underlying manifolds

The density of the underlying data generating distribution is concentrated around a manifold, rather than uniformly spread across the entire state space. It is usually the case that attempts to generate data points from structured domains such as human skeleton poses, images or gene sequences, by uniform sampling results in invalid configuration.

Simple dependencies

Useful representations should have simple dependencies. In other words, simple compositions of good high level representations should be able to express a large range of tasks. For example, if one uses the potential and kinetic energies to express the motion of a pendulum, then the law of preservation of energy provides a simple relationship.

Having listed some of the major assumptions typically made we now discuss the types of models used for representation learning.

2.2.1 *Probabilistic Models*

Probabilistic models [113] exploit some knowledge or assumptions directly relevant to the underlying data generating distribution $p(x, z)$ which is a joint distribution over the data and feature spaces. The features z are treated as latent random variables in the model, while the data x is represented as observed random variables.

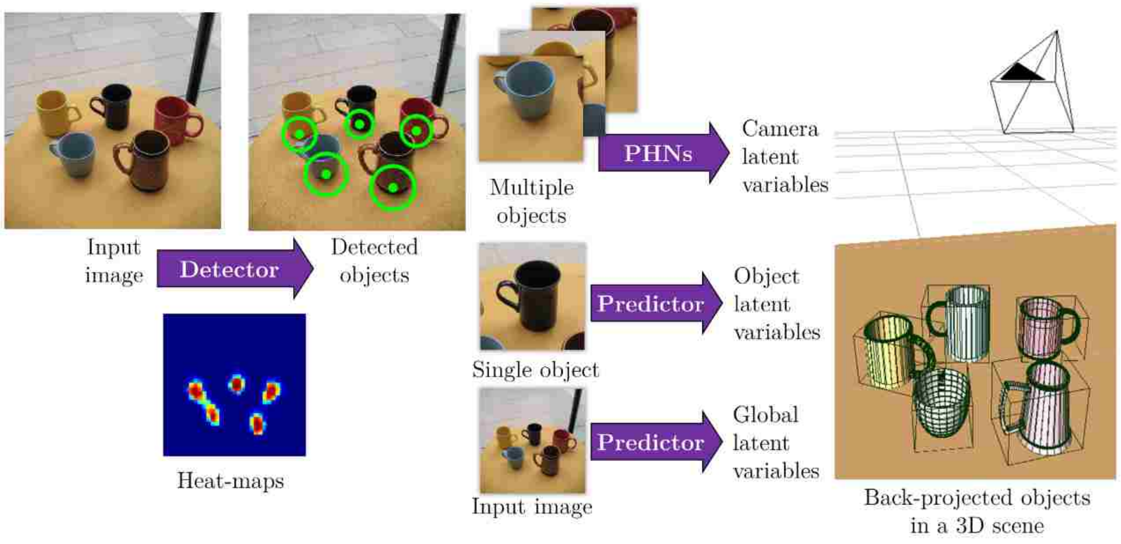


Figure 4: Demonstration of the idea of vision as inverse graphics proposed by [183]. The camera pose and parameters, object and scene properties are treated as latent variables in a directed probabilistic model in order to infer the poses of all objects in the scene. The forward model (likelihood) essentially describes the process of 3D rendering. The figure is adapted from [183].

Depending on the factorisation they assume, probabilistic models fall into two broad classes - directed and undirected models.

2.2.1.1 *Directed models*

Directed models assume that the joint distribution can be factorised into a likelihood distribution $p(x|z)$ and a prior distribution $p(z)$ resulting in

$$p(x, z) = p(x|z)p(z) \tag{1}$$

By applying Bayes rule one obtains an expression for the posterior distribution $p(z|x)$ which can be used to map a new data point to its corresponding point in the feature space

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{2}$$

where the term $p(x)$ is known as the evidence. Typically for any non-trivial problem the calculation of the evidence involves a multidimensional integral rendering the posterior distribution intractable. Therefore, when designing a directed graphical model, one considers the relationship

$$p(z|x) \propto p(x|z)p(z) \quad (3)$$

and approximates the posterior through sampling or approximate inference methods.

The prior encodes any knowledge available a priori and usually is set to enforce some desired properties in the latent space. The likelihood can be viewed as a description of a generative process which maps latent variables to a distribution over the observed data. Therefore, the likelihood model can be sometimes referred to as a generative or forward model. Some probabilistic programming languages such as Anglican [225] (see also Appendix A) take this view and enable the user to specify the likelihood model as a program.

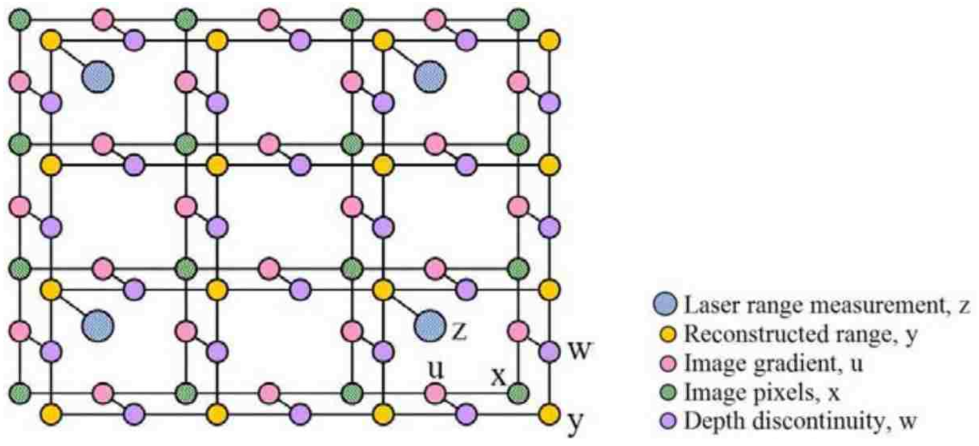
As an example we consider that computer vision can be viewed as a problem of inverse graphics [179, 183]. The key idea is that the likelihood model encodes a 3D rendering process which takes as an input the 3D poses of the objects in the scene, camera and scene properties and generates an image of the virtual scene. In this case, as shown in Fig. 4, the latent variables are the inputs to the rendering procedure and the likelihood is estimated by comparing the rendered with the observed image. The same idea has been applied to even more complex tasks such as estimating the full human skeleton pose from a single image [117] or generating composite 3D shapes matching 2D projections [182].

2.2.1.2 *Undirected models*

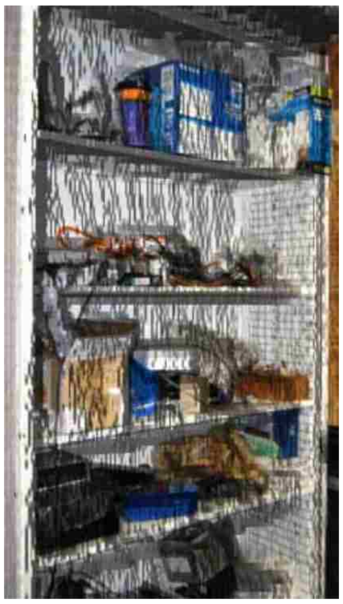
Undirected graphical models, also known as Markov Random Field (MRF), assume a factorisation of the joint distribution which does not contain any conditional factors and is of the form

$$p(x,z) = \frac{1}{Z}\prod_i \nu_i(x)\prod_j \phi_j(z)\prod_k \omega_k(x,z) \quad (4)$$

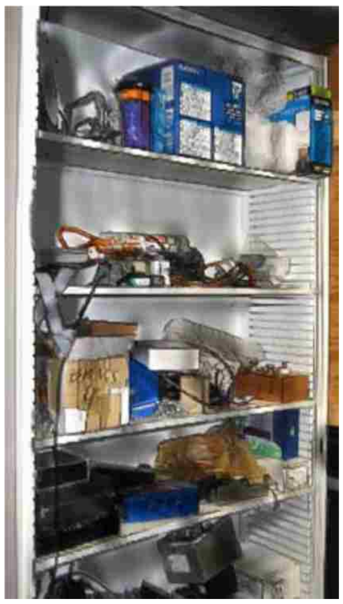
where $\nu_i(x)$, $\phi_j(z)$ and $\omega_k(x,z)$ are positive potential factors which describe the interactions between the observed variables, between the latent variables and between the



(a) A Markov Random Field modelling the interactions between laser depth measurements, stereo depth estimation, image spatial gradients, pixel colours and depth discontinuity.



(b) Input noisy point cloud.



(c) Filtered point cloud.

Figure 5: The undirected probabilistic model **(a)** proposed by [45] for reconstructing 3D point clouds from images and laser measurements. The application of an MRF results in smoother and less noisy output **(c)** compared to the noisy data **(b)**. The figure is adapted from [45].

observed and latent variables, respectively. These potentials must be always positive, but do not necessarily encode distributions as they do not have to integrate to 1. The partitioning function Z ensures that the final product of all factors sums to one.

Undirected models can capture fewer properties of the underlying distribution compared to the directed ones. Instead of describing a generative process, MRFs can be viewed as modelling symmetric interactions.

One of the widest applications of undirected probabilistic models is in image segmentation [156], where pixels interact in a local neighbourhood. Each pixel label is assumed to be a latent variable interacting with neighbouring pixel labels and so resulting in a grid. Figure 5 demonstrates the idea extended to a colour image coupled with depth information as proposed by [45].

In chapter 3, we propose a directed probabilistic model in order to perform physical symbol grounding and exploit some of the properties of fixation programs in order to derive an efficient importance sampling based algorithm.

2.2.2 *Autoencoders*

Another widely accepted approach to representation learning are autoencoders. An autoencoder has two main components: i) an encoder f_θ mapping observed data x to a feature vector z such that $f(x) = z$ and ii) a decoder g_θ performing the inverse mapping from a feature vector z to a reconstruction $\hat{x}$ of the input data point such that $g(z) = \hat{x}$. θ denotes all parameters of the autoencoder.

For a given dataset $\mathcal{D} = x^{(1)}, \dots, x^{(N)}$ the loss function optimised by a basic autoencoder is

$$\mathcal{L}_{AE} = \sum_{i=1}^{N} L(x^{(i)}, g_\theta(f_\theta(x^{(i)}))) \tag{5}$$

where the function $L(x, \hat{x})$ measures the reconstruction error and can be interpreted as a distance metric.

2.2.2.1 *Bottleneck*

If one sets both f_θ and g_θ to be the identity mapping, the the loss in (5) will be at its global minimum of 0. However, the resulting features will be identical to the raw data input which is not useful at all. Therefore, the first step is to bottleneck the autoencoder, meaning that if d_x is the dimensionality of the observed data space and d_z of the feature space then $d_z << d_x$. From an information theory perspective, setting the latent code to be much shorter than the input results in compression of the observed data [137] and so representing it in a more abstract space.

Simply training a basic autoencoder on the data is equivalent to performing dimensionality reduction [20] where the latent space does not necessary have any specific properties which may be desirable. Therefore, various sorts of regularisation techniques are used in order to put more structure into the latent space.

2.2.2.2 *Sparse autoencoder*

As suggested by the name, sparse autoencoders [173] encourage sparse latent codes by adding a an $\ell 1$-norm penalisation term

$$\mathcal{L}_{SAE} = \sum_{i=1}^{N} L(x^{(i)}, g_\theta(f_\theta(x^{(i)}))) + \lambda ||z^{(i)}||_1 \tag{6}$$

where $z^{(i)} = f_\theta(x^{(i)})$. Enforcing sparsity ensures that even if the latent space is of large dimensionality, the autoencoder would still learn interesting features in the data.

2.2.2.3 *Denoising autoencoder*

If an autoencoder is able to filter out noise from its input, then it must have learnt the manifold of the observed data. This is the main assumption behind the denoising autoencoder [218], the loss function of which, is formally defined as

$$\mathcal{L}_{DAE} = \sum_{i=1}^{N} \mathbb{E}_{q(\tilde{x}|x^{(i)})}[L(x^{(i)}, g_\theta(f_\theta(\tilde{x})))] \tag{7}$$

where $\tilde{x}$ denotes a noise corrupted version of the input $x^{(i)}$ by the stochastic process $q(\tilde{x}|x^{(i)})$. The noise model usually considered is salt and pepper noise, where the value of a pixel can be flipped with a small probability. The resulting representations are robust to input noise and are applicable to real world domains such as robotics, in particular localisation and mapping [192].

2.2.2.4 *Contractive autoencoder*

The contractive autoencoder, proposed by Rifai et al. [181], expands on the idea behind the denoising autoencoder and reduces the sensitivity of the autoencoder to small changes in the input. If $J(x^{(i)}) = \frac{df_\theta}{dx}\Big|_{x=x^{(i)}}$ denotes the Jacobian of the encoder at $x^{(i)}$, then the loss function of the contractive autoencoder is formally expressed as

$$\mathcal{L}_{CAE} = \sum_{i=1}^{N} L(x^{(i)}, g_\theta(f_\theta(x^{(i)}))) + \lambda ||J(x^{(i)})||_F \tag{8}$$

where $||.||_F$ denotes the Frobenius norm. Given that the encoder f_θ is usually a nonlinear function the Jacobian is valid only for small perturbations of the input.

2.2.2.5 *Variational autoencoder*

The variational autoencoder [109] is a probabilistic autoencoder which learns a distribution over the latent space instead of providing single point estimates. The encoder f_θ outputs the parameters of a distribution $q_{f_\theta}(z|x)$ which is to approximate the true underlying posterior distribution $p(z|x)$, while the decoder outputs the parameters of the likelihood distribution $p_{g_\theta}(x|z)$. The loss function of the variational autoencoder is

$$\mathcal{L}_{VAE} = \sum_{i=1}^{N} \left\{ -\mathbb{E}_{q_{f_\theta}(z|x^{(i)})} \left[\log p_{g_\theta}(x^{(i)}|z) \right] + D_{KL}(q_{f_\theta}(z|x^{(i)})||p(z)) \right\} \tag{9}$$

where D_{KL} denotes the Kullback–Leibler divergence. Minimising the loss results in i) maximising the log likelihood of the observed data, or in other words reducing the reconstruction error, and ii) ensuring that the approximation distribtion $q_{f_\theta}(z|x)$ is as close to the prior $p(z)$ as possible. A typical choice for the prior is a standard multivariate normal distribution which makes the KL-divergence term in (9) solvable analytically. This, however, imposes significant constraints on the latent space and the types of information it can encode.

2.2.2.6 *Adversarial loss*

Providing the right prior structure on the latent space is crucial for extracting symbolic representations from the data. By symbolic we mean representations which carry some semantic content, not just statistical information about the data. One of the promising research directions on this topic is the use of adversarial losses introduced by the Generative Adversarial Network (GAN) framework. [69].

The key concept behind GANs is that, on one hand, the decoder (or generator) should trick a discriminator that the generated samples come from the real data distribution. On the other hand, the discriminator is simultaneously trained to recognise whether a sample comes from the generator or from the observed data. As a result, the joint adversarial training of a generator and a discriminator imposes an implicit prior on the latent space [139] which is entirely driven by the observed data.

Extensions to the standard autoencoder framework with adversarial losses [37, 107, 139, 140, 207] can learn abstract representations in an unsupervised fashion. For example, the representations learnt from videos by the framework proposed in [43] encode independently the pose a person walking and the scene content.

All of the aforementioned regularisation techniques are purely statistical in nature and the fact that the resulting representations have some semantic meaning is a side effect of the loss function rather than its main objective. Therefore, in chapter 4 we introduce the notion of programmatic regularisation which enforces the latent space to produce symbols that lead to the successful execution of a task encoding program.

2.3 Program Induction

Inductive program synthesis, often referred to as program induction, is the problem of discovering a program from a partial specification such as examples, traces,

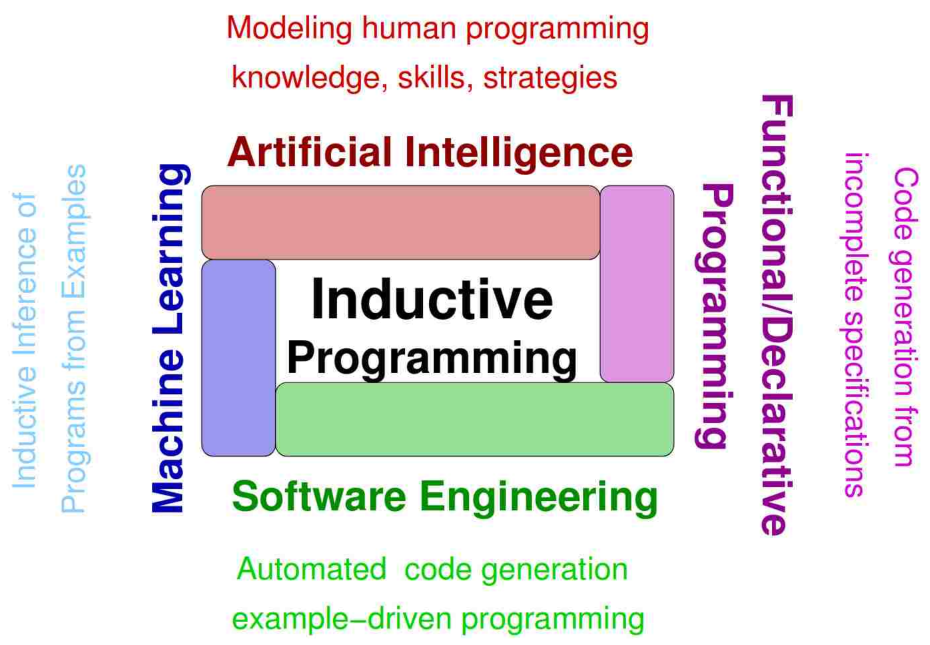


Figure 6: The major fields interested in inductive program synthesis. The figure is reproduced from [187].

natural language or even partial programs [78]. It is a multidisciplinary field spanning AI, programming, software engineering and machine learning as illustrated in Fig. 6.

2.3.1 *User Intent*

Program induction is a challenging problem since the provided specification by the user is incomplete. There is usually millions of programs which will satisfy the provided partial specification, however, only few of them will capture the intent of the user. The underlying program induction procedure could be biased towards finding short programs, or readable programs, but eventually the information provided by the user is of crucial importance. Therefore, the application of inductive program synthesis to real life domains requires some sort of interactive programming [75–77] where the user is able to seamlessly provide more information when required. Gulwani, Polozov and Singh [78] are of the opinion that this poses more challenging problems to the human-computer interaction community than to the program synthesis one.

2.3.2 *Program Space*

The general formulation of the program synthesis problem is undecidable and so methods typically perform various sorts of heuristic search in the space of possible programs [78]. Searching is in itself a hard combinatorial problem due to the size and intricacy of the program space. Not only does the size of the space grow exponentially with the length of the program, but even simple programs could have complex structure and be built by arbitrary composition of sub-programs. While the first phenomenon is widely referred to as the curse of dimensionality, we refer to the second one as the curse of compositionality.

Programs need to be represented in a space which balances a trade off between expressiveness and efficiency. On one hand if a program space can express a large set of problems, then it must contain a large number of simple instructions leading to long programs. On the other hand, if a Domain Specific Language (DSL) is used, then the resulting programs will be shorter, but will be able to express only a small subset of problems. For example, Anglican [225] is a DSL for describing probabilistic models and running inference on them (see Appendix A).

2.3.3 *Search Techniques*

There is a vast amount of research on the topic of search techniques for program synthesis (see [78] for an extensive up-to-date review), which can be separated in 4 broad classes.

2.3.3.1 *Enumerative search*

Enumerative search is equivalent to a bottom-up search where a program is constructed by gradually expanding and modifying it while ensuring its correctness. As a result the space of all programs is traversed in a certain order. These methods are often improved significantly by pruning the search space or by traversing it in a particular order.

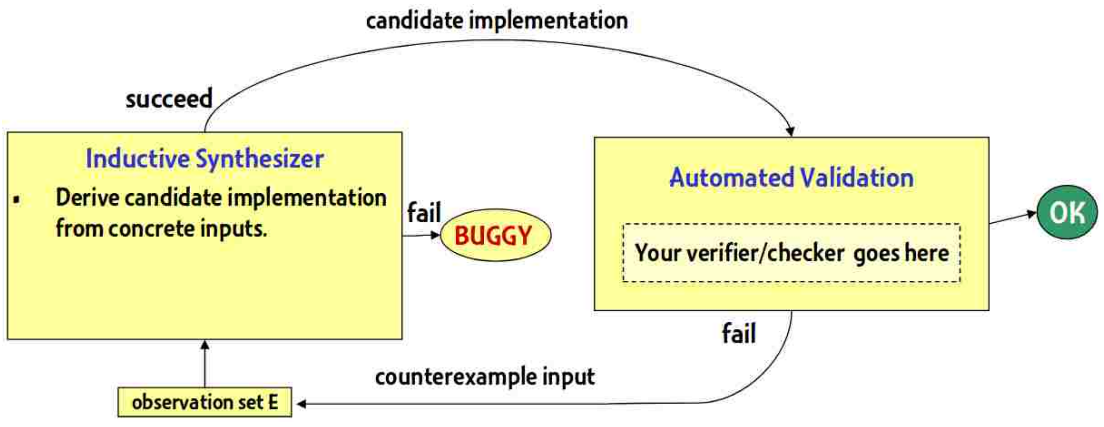


Figure 7: A diagram of the counter example guided inductive synthesis method introduced by [195]. At every step of the induction process the candidate program is passed through a verifier which either accepts it or generates counter example where the program fails. Each counterexample serves as an additional constraint to the synthesiser. The figure is reproduced from [195].

2.3.3.2 *Deductive search*

Deductive search [141] can be viewed as a top-down search where one generates a subprogram and then considers the problem of generating another subprograms to replace the input arguments of the first one. This process requires the inverse semantics of the initial program to be determined [172].

2.3.3.3 *Constraint satisfaction*

One can formulate program synthesis into a constraint satisfaction problem by, for example, expressing it in boolean algebra or first order logic and utilising a SAT or SMT solver, respectively. Of particular importance is the idea of counterexample guided inductive synthesis introduced by Solar-Lezama et al. [196] and depicted in Fig. 7. At each step of the inductive process the program is run through a verifier that either returns a counterexample at which the program fails or simply accepts it. Each generated counter example serves as a new constraint to the the solver.

2.3.3.4 *Statistical techniques*

In the domain of inductive program synthesis, statistical methods are typically applied as an augmentation to the core search procedure in order to exploit data driven

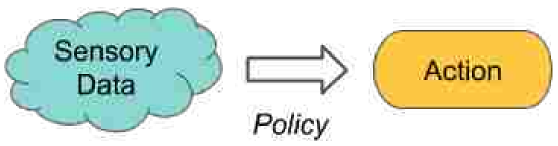


Figure 8: A policy is a *direct* mapping from sensory data to an action.

heruistiscs. Genetic programming [115] is one of the earliest statistical approaches utilised for program induction. Another approach is to formulate program synthesis as a probabilistic inference problem and perform sampling to obtain the solution [35, 49, 123, 152, 182]. Recently, deep learning techniques have been integrated in various ways into program synthesis systems. Neural guided search has become one of the most explored approaches where neural networks are trained on large amounts of data to bias the search such that the program space is traversed in an efficient order [17, 128, 171, 217, 238]. Furthermore, the success of deep learning, has lead to the emergence of neural program induction and neural program synthesis [104]. In neural program induction a neural architecture, at the core of which is typically an RNN augmented with some sort of memory, is trained to generate the right output for a given input. The field was pioneered by the introduction of the neural Turing machines by Graves, Wayne and Danihelka [70]. In neural program synthesis a neural architecture is trained to output entire programs which can later be run on new inputs, examples include [8, 29, 44].

In chapter 5 we propose the π-machine which is a novel framework for inducing concise functional LISP-like programs from traces. It combines an enumerative A* search strategy with a stochastic gradient descent optimiser in order to prune the search space.

2.4 Task Learning

2.4.1 *Reinforcement Learning*

RL is concerned with the problem of how an autonomous agent can learn to execute a certain task in the environment. Learning is guided by a reward signal that is obtained from the environment and reflects how well the agent is performing. At each time step the agent senses the environment and chooses an action which results in a certain reward signal. The main objective of RL is to choose actions such that the total expected reward is maximised [198]. A mapping from an observed state to an action is known as a policy and the mapping which achieves the objective is called the optimal policy. State of the art RL methods represent the policy as a deep neural network capable of capturing nonlinear relationships. Learning a *direct* mapping (see Fig. 8) straight from sensory data to actions, especially as a neural network, brings several challenges.

Firstly, the generalisation capabilities of the agent to variations in the environment or the task are limited [13] due to the data driven training procedure. Once the agent has successfully learnt how to solve a specific task in a given environments, even slight perturbations in the input sensory data could result in wrong decisions. Secondly, it is not obvious at all how the agent could deal with changes in the specification of the task. Even though the same task related concepts are likely to be needed, the decision making process might be slightly different. Lastly, it is a research topic of its own how to inspect and interpret decisions made by neural networks [147], or providing guarantees about their performance [48, 145].

In order to address these issues recent RL work puts an emphasis on learning representations from sensory data. Explicitly learning representations results in policies robust to shifts in the operational domain [89] and are transferable to a certain extent to new tasks [153, 158, 174].

2.4.2 *Robotics & Learning from Demonstration*

Even though RL methods have been applied to physical robots [5, 57, 132, 133], they tend to require a vast number of training episodes which not only take a long time, but also wear out the mechanical components of the robot. A more suitable approach is to utilise a small number of expert demonstrations in order to quickly learn new tasks. Learning from demonstration is particularly well suited for collaborative HRI setups, where people constantly interact with the robot.

At its core, learning from demonstration is analogous to the problem of program induction, where a user expresses their intent by executing some actions and the robot has to extract a policy which solves the task. Current methods in robotics for deriving a policy from demonstrations can be classified into 2 main groups, which we discuss in the following sections.

2.4.2.1 *Model-Free*

The most simple approach for deriving a policy from demonstrations is to frame the problem into a supervised learning framework where a predictor is to output an action for each observed state (see Fig. 8). A common assumption is that the observed demonstrations contain independent samples of state-action pairs. Ross, Gordon and Bagnell [184] recognise that this assumption is not correct and provide an algorithm for interactive learning from demonstration. Even though direct mapping methods excel at producing optimal behaviours they generalise poorly and so current research efforts focus on augmenting them with qualitative models [175, 204].

2.4.2.2 *Model-Based*

In comparison to training virtual agents, working with physical robots imposes multiple constraints such as limited amount of data and inability to train faster than real time. Therefore, model based methods for task learning from demonstration are becoming more popular. The key idea is to learn a model of the environment and then utilise it to plan with it at runtime and generate actions. Structured environment models have already been shown to greatly improve the learning efficiency of autonomous agents [175, 204].

A common approach in robotics is to perform inverse reinforcement learning [2], where the model is an induced reward function of the demonstrator. Once the reward function is available, standard RL methods can be utilised in order to solve the task. Inverse RL has been applied with impressive results to performing aerobatic manoeuvres with a helicopter [3] and interactively navigating around people [215].

Models that combine physical and visual understanding of the world are also becoming more common. For example, Sermanet et al. [191] train an autoencoder on videos of the same demonstration from various points of view. Once the autoencoder is trained, the robot is able to execute new tasks from a single video demonstration using the learnt representations.

Another novel idea is to learn models of how to learn new tasks. Finn et al. [58] introduce meta learning and demonstrate that by training a model to adapt to new tasks, rather than simply to solve a single one, they obtain extremely efficient learning performances from single observations. This approach has been extended in [234] by training another model from a corpora of videos in order to encode rich physical and visual prior knowledge about the world and improve the generalisation capabilities of the agent even further.

The aforementioned models are built on deep neural architectures and so they inherit all the problems related to black-box models. Therefore, more structured methods, typically based on learning graph representations [7, 227], have also been proposed. For example, Xiong et al. [227] demonstrate that learning a spatial temporal and causal And-Or graph [210] enables a robot to fold a cloth from only few demonstrations. Furthermore, it is also possible to learn from demonstrations the pre-conditions and post-conditions of actions in order to perform classical planning at runtime [4].

Despite the considerable achievements in the field of learning by demonstration, one of the key challenges remaining is to determine what is the right representation of the environment as well as how to encode the task within the chosen representation [12, 97, 240].

2.4.3 *Cognitive Science*

Interestingly, there is evidence from cognitive science and neuroscience, that the human brain does use structured task related abstractions.

Reverberi, Görgen and Haynes [178] perform an experiment where participants are asked to remember and apply a set of simple and compound rules while being in an fMRI scanner. After training classifiers to detect simple rules from the activity in the lateral prefrontal cortex, the researchers were also able to decode the compound rules, suggesting that the code used to store the rules is compositional.

Furthermore, Cole et al. [40] have performed a series of experiments studying human performance in rapid instructed task learning [39] and demonstrate that classifiers trained to identify abstract rules based on practiced task activity patterns successfully generalised to novel tasks.

2.5 Conceptual Isomorphisms

Programs can be viewed both as an automatic formal systems interpreted by a digital computer [82] or a transition system [16]. Markov processes and Makrov decision processes are particular instances of transition systems which are widely used in the domain of autonomous agents [198]. Aligning these frameworks, as shown in Table 1 reveals some of the ideas which lie at the core of the contributions of this thesis.

One of the key observations is that task execution is equivalent to computation and so the process of decision making can be encoded as a program. This idea, in its purest, most naive version, is the main conjecture of symbolic artificial intelligence. We, however, recognise the fact that agents receive only sensory signals from the environment, which are equivalent to the alphabet of a formal system, not to the well-fromed formulae that are at a higher abstraction level. In other words, some sort of representation learning is needed in order to describe environment states with symbols which a program can operate with. This is the key idea behind GLIDE in chapter 3 and the perceptor in chapter 4. The π-machine, introduced in chapter 5, is based on the observation that task execution can be described as a transition system in order to learn programs from observed traces.

Formal Systems	Digital Computers	Transition Systems	Autonomous Agents
alphabet	0 and 1 (bits)	state components	sensory signals
well-formed formulae	any machine word	any state	any environment state
axioms	input	initial state	initial state
generation rules	program	transition relation	decision making & environment dynamics
theorems	output	resulting states	resulting states
proof	computation	system evolution	task execution

Table 1: The observed conceptual isomorphisms between formal systems, transition systems, digital computers and autonomous agents which lie at the conceptual core of this thesis.

3 Physical Symbol Grounding from Demonstration

3.1 Introduction

Despite the recent significant advances in the capabilities of autonomous humanoid robots [15], much remains to be done before robots are able to function effectively in complex human environments. This is especially the case when understanding of contextualized information is required within cluttered and dynamic environments such as the collaborative HRI assembly setup shown in Fig. 9.

Naturally, humans work with abstract plans as they are often an intuitive and concise way to represent a task. The ability to understand and autonomously execute high level instructions is crucial for the development of intuitive and efficient human-robot interfaces. However, high level instructions contain symbols and concepts which need to be interpreted and grounded within the environment in order to enable the execution of the plan. HRI research related to interpreting abstract instructions (usually in the form of natural language) approaches the problem as grounding the unknown references within an instruction to a set of predefined actions and observed objects or locations in the environment [112, 203]. The problem of learning the mapping between abstract symbols and their physical instances in the environment, also known as the problem of *physical symbol grounding* [219], connects the idea of situated robotics [30] to the more general problem of symbol grounding [80] which is one of the core challenges in artificial intelligence [200].

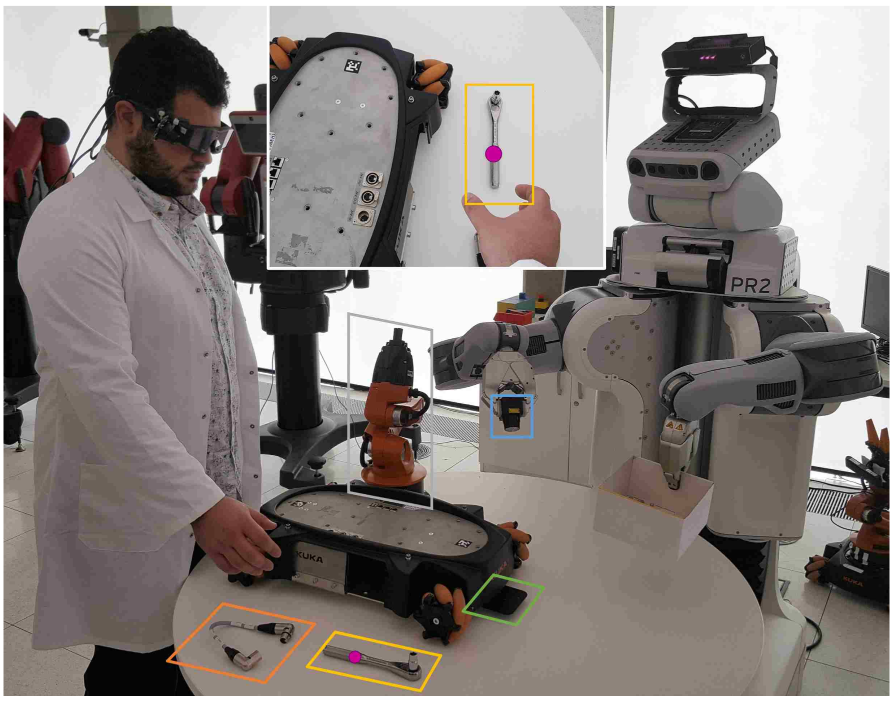


Figure 9: Collaborative assembly settings demand understanding of contextualized information within cluttered and dynamic environments in order to understand human instructions and intents. Recognising task relevant objects and locations is also necessary in order to instantiate high level plans. As we demonstrate in this chapter, eye tracking information enables both symbol grounding and learning the appearance of symbol instances.

Importantly, it is often the case that a robot may not have any knowledge about the model or the appearance of a symbol instance and so assuming a fully observable environment imposes considerable constraints on the robustness, scalability and applicability of the existing methods. In this chapter, we introduce a framework for Grounding and Learning Instances through Demonstration and Eye tracking (GLIDE) which allows for simultaneously grounding symbols to their instances in the environment and learning their appearance, thus relaxing the full observability constraint. GLIDE enables robots to instantiate a high level plan within an initially unknown environment and then complete the task autonomously.

The proposed framework is based on learning from a small number of demonstrations by a person wearing eye tracking glasses. Considering the exponential development of technologies such as ubiquitous computing and the Internet of Things, being able to naturally communicate with a robot while wearing an eye tracking device seems an extremely plausible future. We exploit the fact that fixations are highly dependent on the task and provide information about the location of task related items. We describe a methodology for recording 3D fixation coordinates in the environment and call the sequence of fixations during task execution a *fixation program*. We formulate the problem of mapping fixation programs to high level task plans as probabilistic inference and demonstrate how this mapping can be used to ground plan symbols and learn appearance distributions of symbol instances in the environment. We recognise probabilistic programming as a tool well suited for the problem and so use Anglican [225] (see also Appendix A) for the implementation. We tested GLIDE on experimental data from human demonstrations and confirm that it successfully performs physical symbol grounding.

The main contributions presented in this chapter are:

- Methodology for recording 3D fixations within the environment based on monocular visual SLAM.
- An inference algorithm exploiting the properties of fixation programs in order to ground symbols and localise their instances in the environment.
- An algorithm for learning the appearance of symbol instances when no previous knowledge is present.

3.2 Related Work

3.2.1 *Symbol Grounding*

The problem of connecting symbols to their meaning was introduced as the "Chinese Room" experiment [190] and later formally defined as the symbol grounding problem by Harnard [80]. A symbol is any object that is part of a symbol system and symbols are arbitrary in their shape. A symbol system is a set of symbols and syntactic rules for manipulating them on the basis of their shapes (not their meanings). Since robots are embedded and situated agents, our interpretation of symbol meaning follows the *physical symbol grounding* paradigm proposed by [219] and is a "functional relation between a form and a referent". In other words, the meaning of a symbol is its relation to the physical entity in the environment which the symbol is referring to.

Previous work related to symbol grounding and robotics focuses on human-robot interaction scenarios where natural language is used to provide commands to the robot [112, 203]. The syntactic constituents of the utterance are grounded to entities of interest in the environment - objects, locations, trajectories, actions, events. Interestingly, the utterance can also be semantically parsed to a program expressed in a robot control language [143] which the robot can execute. Multimodal models fusing gestures and language have also been proposed [144, 222]. However, a common assumption is that all entities of interest in the environment are observable. GLIDE aims to relax this constraint by enabling simultaneous grounding of the plan symbols and learning of the appearance of their instances. Importantly, the proposed framework utilises a high level planning language, instead of natural language, for the representation of plans.

3.2.2 *Eye Tracking During Task Execution*

It was Yarbus who first demonstrated in 1967 that fixation patterns are higly influenced by the task to be executed [230]. He asked subjects to examine a picture (*The Unexpected Visitor*, Fig. 10 top left) and gave them various tasks related to the image such as "Estimate the material circumstances of the family in the picture" or "Give the ages of the people". All seven tasks resulted in significantly different eye movements as show in Fig. 10.

Eye tracking has been used by cognitive scientists for long time in order to determine the mechanisms for oculomotor control and the associated cognitive processes. Most of the work so far has been conducted in laboratory conditions and has focused on processes such as reading and looking at images [86, 176]. Technological advances have recently enabled eye tracking during natural behaviour [84, 125]. Studying actual environment perception is important as it provides more realistic data which sometimes contradicts the predictions from laboratory experiments [59].

Oculomotor control has been studied during the performance of various tasks such as sandwich making, tea making, driving, playing table tennis, playing cricket and others (see [125] for an extensive review). The results from those studies show

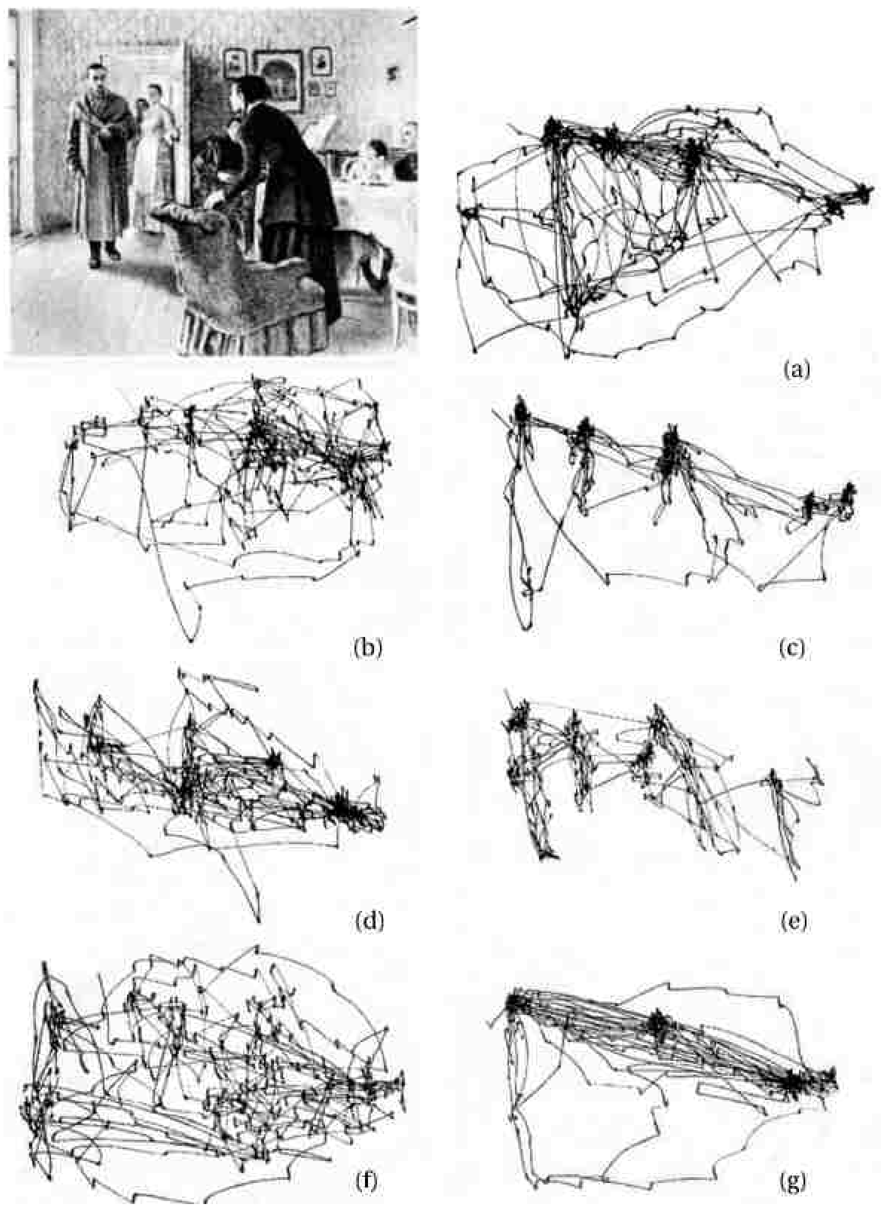


Figure 10: Examining a picture. **(a)** Free examination. **(b)** Estimate the material circumstances of the family in the picture. **(c)** Give the ages of the people. **(d)** Sumarise what the family had been doing before the arrival of the 'unexpected visitor'. **(e)** Remember the clothes worn by the people. **(f)** Remember the position of the people and objects in the room. **(g)** Estimate how long the unexpected visitor had been away from the family. The figure is adapted from [201].

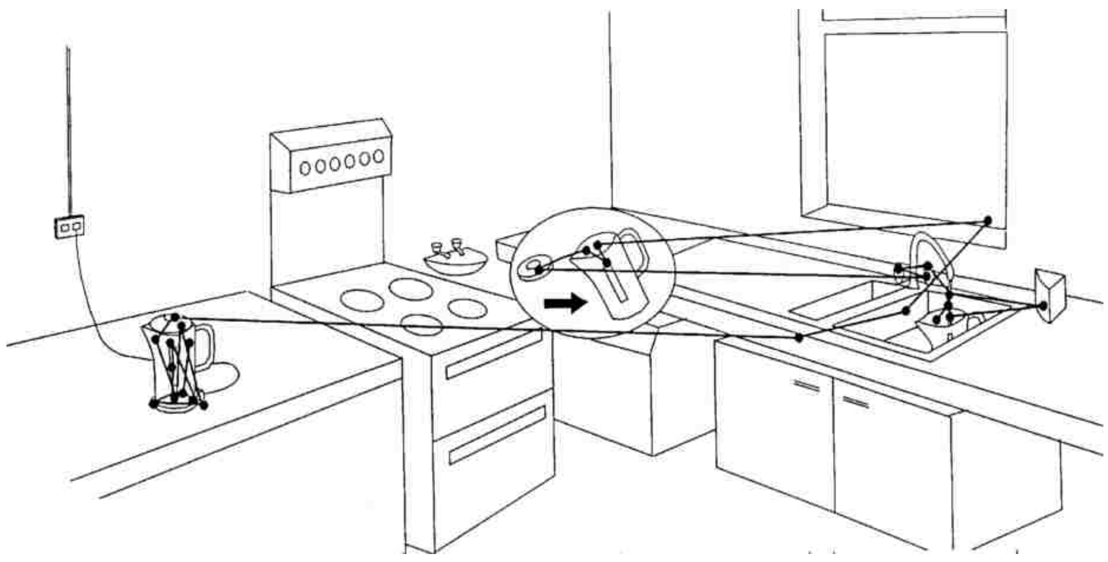


Figure 11: Fixations during the execution of a complex natural task. The figure is reproduced from [126].

that eye movements during task execution are task specific and are barely affected by low level features in the environment [83, 84].

This supports the active vision paradigm according to which the vision system actively seeks information that is relevant to the current cognitive activity [185]. For example, Land et al. conducted a study [126] in which participants were asked to prepare a cup of tea in a regular kitchen environment while their eye movements were recorded. The majority of fixations were on objects related to the task despite the complexity of the environment and the free motion of the participants as shown in Fig. 11.

Furthermore, When observing an environment with natural physics, e. g. balls rolling on a flat surface and bouncing off each other, evidence suggests that eye movements take into account the underlying causal model [66].

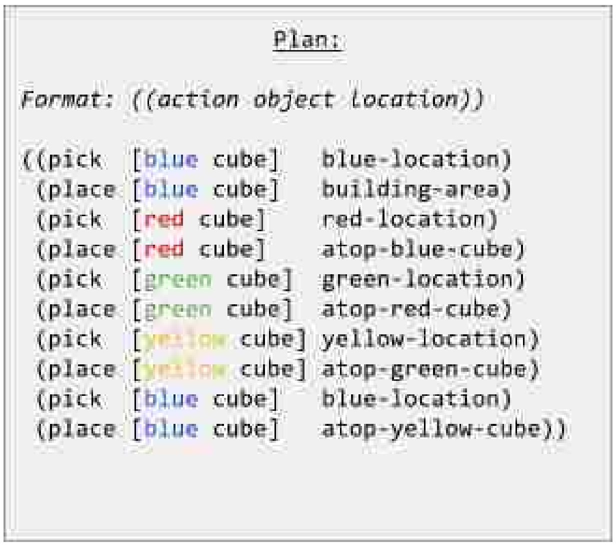
Eye Tracking Glasses (ETG) provide fixation locations as 2D points within the image from a first person point of view camera mounted on the device and facing the scene. Paletta et al. demonstrated that fixations can be projected onto a precomputed 3D map, by utilising Scale-invariant Feature Transform (SIFT) features in the scene images [157], while Pfeiffer et al. rely on artificial fiducial markers in order to estimate 3D fixation locations [169]. Our approach utilises monocular visual SLAM in order to reduce the constraints on the scene.

The predominant use of eye tracking in the field of robotics is in HRI, where gaze information enables the recognition of human behaviour [231] and the execution of anticipatory actions [96]. However, we focus on the question of how eye tracking can guide instruction grounding and perception. In this line of thought, Papadopoulos et al. treat fixations within an image as a noisy supervisory signal for the training of visual object class detectors [159], however they do not reason about fixations during the execution of multi-action tasks.

3.2.3 *Activity Recognition by Sequential Probabilistic Models*

Hidden Markov Models (HMMs) are widely used for activity recognition by casting the problem of partitioning observed trajectories into a probabilistic inference one [19, 52, 74, 229]. Importantly, HMM's are well suited for applications to the programming by demonstration domain, in particular modelling and analysing human motions, as they are robust to spatio-temporal variability in the observed multimodal data [98, 213].

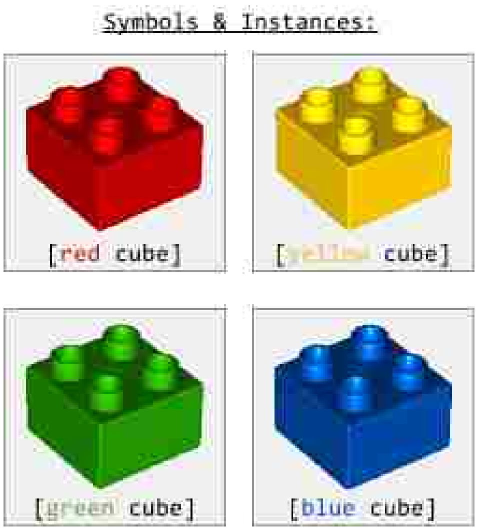
There is a large range of HMM extensions that allow for various trajectory structures to be captured. For example, hierarchical HMMs can infer composite actions that are themselves built up from primitive actions [56, 151] and sticky hierarchical HMMs can infer partitionings that favour fewer changes in the latent state by imposing a sticky Dirichlet prior [95]. Of particular relevance to the probabilistic model that we consider is the factorial HMM which assumes that there are multiple hidden state variables influencing the observations [26, 67]. In our case, we assume that both the action being executed and the environment state influence the observed fixations. The probabilistic graphical model that we propose borrows, to some extent, ideas from the aforementioned HMM variants, however we consider the problem of learning the structure of the model together with learning its parameters from data.



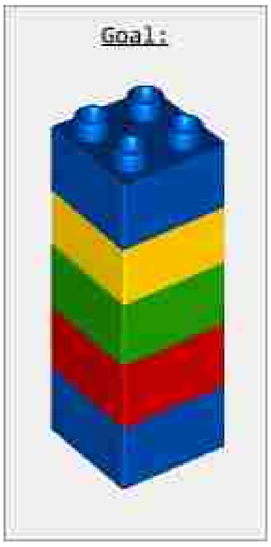
Plan:

```
Format: ((action object location))

((pick  [blue cube]   blue-location)
 (place [blue cube]   building-area)
 (pick  [red cube]    red-location)
 (place [red cube]    atop-blue-cube)
 (pick  [green cube]  green-location)
 (place [green cube]  atop-red-cube)
 (pick  [yellow cube] yellow-location)
 (place [yellow cube] atop-green-cube)
 (pick  [blue cube]   blue-location)
 (place [blue cube]   atop-yellow-cube))
```

(a) Symbolic plan representation.



(b) Plan symbols and their physical instances.



(c) Task goal.

Figure 12: The representation of a plan **(a)** which solves an example task with the goal of building a tower with five colour cubes in a particular order **(c)**. All plan symbols need to be grounded to their physical instances **(b)** in order to enable autonomous execution of the task.

3.3 Problem Definition

Initially, a task such as building the tower of cubes shown in Fig. 12c is demonstrated by a person wearing eye tracking glasses. We make no assumptions about the availability of prior knowledge related to the environment, however the task to be demonstrated $\mathcal{T}$ is predetermined. Additionally, the robot has access to a dictionary with primitive actions $\mathcal{A}$ which the person can execute in an arbitrary sequence in order to solve the task. As illustrated in Fig. 12a, we represent plans for solving the task as a sequence of `(action object location)` tuples where `action` is symbol corresponding to an action from $\mathcal{A}$, `object` is a symbol referring to a physical object in the environment and `location` is a symbol referring to a physical location in the environment, sometimes dependent on the previous action. Even though `action` is a primitive action and known in advance, it can be executed by the robot only after the `object` and `location` symbols are grounded to their instances within the environment as shown in Fig. 12b. Given that a person wears an ETG device and demonstrates how to execute the plan, the problem we address is how eye tracking information can be used to ground plan symbols to their instances in the environment and learn to recognise those in order to perform the demonstrated task autonomously.

3.4 Method

We split the problem of simultaneous symbol grounding and instance learning in three parts. First, we describe how to estimate 3D fixation locations within the environment in order to map fixations to physical locations in the environment. Secondly, we show how to localise the instance of a plan symbol by exploiting the properties of a fixation program. Lastly, once the location of an instance is known, we describe how to learn its appearance in order to enable automatic recognition.

3.4.1 *3D Eye Tracking*

Mobile eye trackers provide fixation information in pixel coordinates corresponding to locations in the image of a first person view camera. Relying only on that information requires all items of interest in the environment to be detected from the image in order to determine which one is fixated. However, in a collaborative assembly scenario, such as the one shown in Fig. 9, there are several sensors which can be used - the robot has an RGBD sensors, multiple cameras and laser scanners. Additionally, the environment can also be highly sensorised. In order to take advantage of multiple sensors, instead of a single first person view camera, we estimate the 3D fixation locations, enabling the projection of fixations in the frame of any sensor in the environment.

In order to achieve that, we calibrate the first person view camera and utilise the ORBSLAM algorithm [148], which is a monocular visual SLAM algorithm, in order to estimate the 6D pose of the eye tracking glasses in real time. Since a single camera is used, the obtained measurements are correct only up to a scale factor. Therefore, we have developed a calibration procedure which relies on detecting several APRIL tags [155] initially present in the scene. First, the physical distance d_{calib} between two different view points is estimated using the calibration parameters of the camera and the tags. Then, the distance d_{vslam} is calculated from the estimated poses by ORBSLAM and the scale is calculated as d_{calib}/d_{vslam}. Additionally, the transformation between the world frame and the origin of the ORBSLAM frame is also also computed during the calibration procedure. Once the 6D pose of the glasses is calculated within the world frame, the fixation locations are projected from the first person view camera image to the 3D environment by ray casting and finding the intersection with a 3D model of the environment. As a result, fixations can be represented as 3D locations in the environment instead of just pixel coordinates. We are interested in scenarios where table top manipulation is required, so we project fixations to the plane of the table top resulting in 2D points with physical coordinates. It is also possible to intersect the ray with a point cloud of the environment [157]. A diagram visualising the process is shown in Fig. 13, where the 3D model of the environment is assumed to be the table top surface.

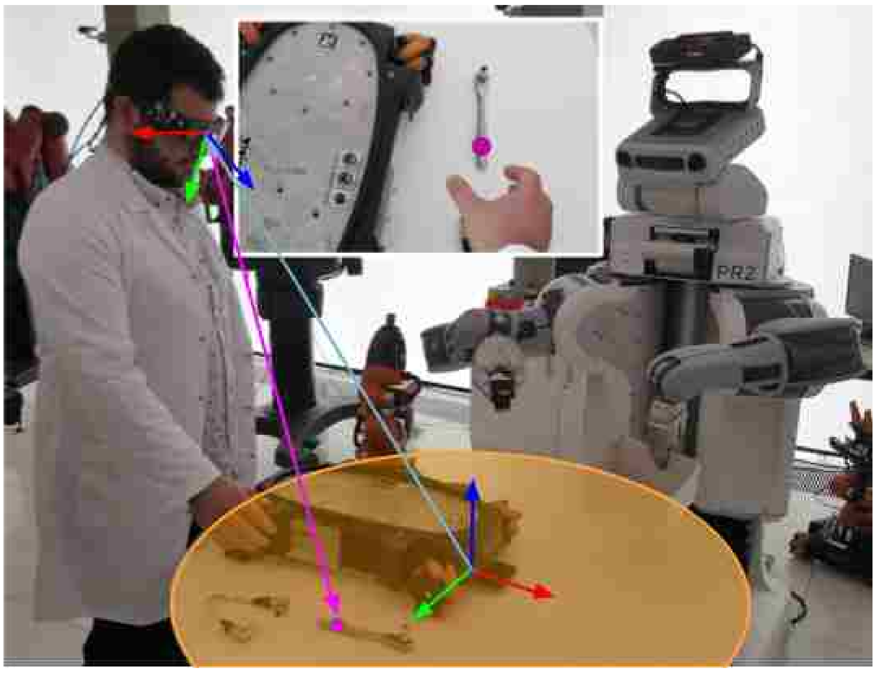


Figure 13: The pose of the ETG device is estimated with respect to a global frame of reference using ORBSLAM [148]. Then the fixation location within the image of the ETG camera is used to project a 3D ray and intersect it with a 3D model of the environment, the table top surface in this case, in order to obtain a 3D location of the fixation within the environment.

Utilising a visual monocular SLAM algorithm for pose estimation of the eye tracking glasses imposes fewer constraints on the scene, however it puts demands on the hardware. Throughout our work we use the SMI ETG 1 device which provides fixations information at a rate of $60Hz$ and the first person view camera has a frame rate of $30FPS$. Initial experiments revealed that this frame rate is insufficient for the ORBSLAM algorithm since people rotate their heads relatively fast, adding significant blur to the images. Therefore, we have attached an extra camera to the eye tracking glasses which provides $120FPS$ and enables robust head tracking. The transformation between the frame of the eye tracking glasses camera and the high frame rate one is also estimated during the calibration procedure.

3.4.2 *Model Definition*

In order to solve the problem of physical symbol grounding we define a generative probabilistic model shown in Fig. 14. The task $\mathcal{T}$ and the fixation program $F_1 \dots F_T$ are

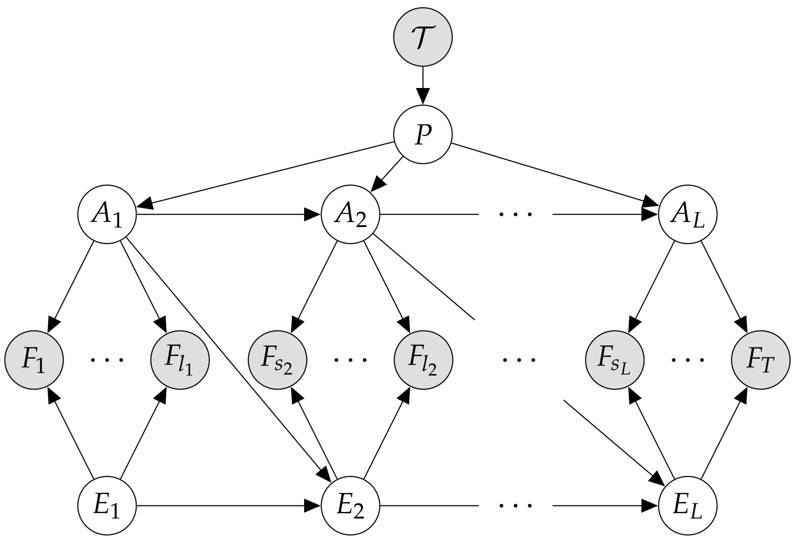


Figure 14: The proposed probabilistic model for physical symbol grounding which is based on the idea that "task and context determine where you look" [185].

observed, where $F_i \in \mathbb{R}^2$ and T is the total number of observed fixations. P encodes any valid plan which successfully completes the task. Following the idea that "task and context determine where you look" [185], the action $A_i \in \mathcal{A}$ induces a set of fixations $F_{s_i} \dots F_{l_i}$ on a certain item of interest related to the action, where $s_1 = 1$ and $l_L = T$. In the context of planning, an item of interest could be any of the symbols in the plan. For example, given the plan in Fig. 12a, an item of interest might be either a particular cube or a location such as the `building-area`. The fixations $F_{l_i+1} \dots F_{s_{i+1}-1}$ are observed during the transition from one item of interest to another and are not represented in the graphical model in order to avoid clutter.

Importantly, the sequence of fixations $F_{s_i} \dots F_{l_i}$ depends on the actual physical location of the item of interest. The main assumption behind the proposed framework is that symbol instances cannot be recognised prior to learning, therefore we model the belief about the position of item m as a normal distribution $\mathcal{N}_m(\mu_m, \Sigma_m)$ over possible locations in the environment. Since fixations are projected to the table top plane $\mu_m \in \mathbb{R}^2$ and Σ_m is a 2×2 covariance matrix. μ_m and Σ_m are latent variables which we are interested in inferring in order to consequently learn the instance of symbol m. The set of tuples $\{(\mu_i, \Sigma_i) : i = 1 \dots M\}$ which encodes the state of the environment is denoted as E, where M is the number of items of interest, or equivalently the number of symbol instances. We assume that the changes in the environment from state E_i to state E_{i+1} can be caused only by the action A_i.

3.4.3 *Inference*

Assuming a uniform prior over the possible locations of each item and that a task can be uniformly chosen from a set of predefined ones, we are interested in solving the following inference problem

$$p(E_{1:L}|F_{1:T},\mathcal{T}) \propto \sum_{P,\, A_{1:L},\, s_{1:L},\, l_{1:L}} p(F_{1:T}|A_{1:L},E_{1:L})p(A_{1:L}|P)p(P|\mathcal{T}) \tag{10}$$

where we use the notation $X_{a:b}$ to represent the sequence $X_a, X_{a+1}, \ldots, X_{b-1}, X_b$. The inference problem which we need to solve can be viewed as a structure learning problem as the variables $s_{1:L},\ l_{1:L}$ encode the connectivity of the probabilistic graphical model.

The key insight, which we make in order to obtain an efficient inference procedure, is that fixation programs encode information about why a certain point is fixated (what action is performed) and where the fixation is located (where the action item of interest is). Reasoning about those two aspects can be performed independently, as encoded by the probabilistic graphical model in Fig. 14:

$$p(F_{1:T}|A_{1:L},E_{1:L}) \propto p(F_{1:T}|A_{1:L})p(F_{1:T}|E_{1:L}) \tag{11}$$

By exploiting the properties of fixation programs, we split the inference problem in 4 parts in order to obtain a computationally feasible solution using importance sampling.

3.4.3.1 *Generating Actions*

The term $p(P|\mathcal{T})$ encodes the probability of a plan given the demonstrated task and is computed by a high level planner $\mathcal{P}$. The planner is assumed, by utilising the dictionary of primitive actions $\mathcal{A}$, to find the set of all plans $P_{\mathcal{T}}$ that successfully achieve the task. $p(P|\mathcal{T})$ is defined as a uniform categorical distribution over $P_{\mathcal{T}}$. By biasing the categorical distribution, it is possible to represent any preferences with respect to a plan, for example shorter plans are selected with higher probability.

$p(A_{1:L}|P)$ is similar to an indicator function and assigns a probability of 1 to the sequence of actions $A_{1:L}$ which is defined by plan P and 0 to any other sequence.

3.4.3.2 *Generating Fixation Program Segments*

The main difficulty in expressing the likelihood of the observed fixations given the actions as shown in (11) is the fact that there are T fixations and L actions. The fixations induced by action A_i are $F_{s_i:l_i}$, while the fixations made during a transition from action A_i to A_j are given by $F_{l_i+1:s_j-1}$. If there are L actions then there are $L-1$ additional transitions resulting in $2L-1$ segments of the sequence of fixations in total. Thus, the term $p(F_{1:T}|A_{1:L})$ in (11) can be rewritten as

$$p(F_{1:T}|A_{1:L}) = p(F_{1:l_1}, F_{l_1+1:s_2-1}, F_{s_2:l_2} \dots F_{s_L:l_L}|A_{1:L}) \tag{12}$$

Similar to the stick breaking analogy for sampling from a Dirichlet distribution [202], the sequence $s_1, l_1, s_2, l_2, \dots s_L, l_L$ can be viewed as points where the fixation program is split in $2L-1$ segments with total length equal to T. Therefore, $s_1, l_1, s_2, l_2, \dots s_L, l_L$, are sampled from a $(2L-1)$-dimensional symmetric Dirichlet distribution with concentration parameter α and normalised such that $s_1 = 1$ and $l_L = T$. Since there are no zero length segments in the fixation program α should be greater than 1. Empirical tests showed that setting $\alpha = 2L$ yields a stable heuristic. It should be noted that sampling the fixation program segments is equivalent to sampling the structure of the graphical model in Fig. 14.

3.4.3.3 *Evaluating Fixation Program Likelihood*

Fixations on items of interest are clustered on the item, while fixations made during transition are sparse and have relatively large distance between each other. Therefore, the likelihood $p(F_{1:T}|A_{1:L})$ is modelled as

$$p(F_{1:T}|A_{1:L}) = \prod_{i}^{T-1} L_{item}(F_i, F_{i+1}) L_{trans}(F_i, F_{i+1}) \tag{13}$$

The likelihood of two consecutive fixations F_i and F_j during an action A_k is defined as an exponential distribution over the distance between them $d_m = ||F_i - F_j||$, thus

$$L_{item}(F_i, F_j) = \begin{cases} \lambda_m e^{-\lambda_m d_m} & \text{if } [i,j] = [s_k, l_k] \\ 1 & \text{otherwise} \end{cases} \tag{14}$$

which means that fixations closer together are more likely. λ_m is the mean distance between consecutive fixation during an action and is learnt from labelled data.

The likelihood of two consecutive fixations F_i and F_j during a transition from A_k to A_{k+1} is defined as an exponential distribution over the $d_s = D_{max} - ||F_i - F_j||$, where D_{max} is the maximum possible distance between F_i and F_j, and so

$$L_{trans}(F_i, F_j) = \begin{cases} \lambda_s e^{-\lambda_s d_s} & \text{if } [i,j] = [l_k + 1, s_{k+1} - 1] \\ 1 & \text{otherwise} \end{cases} \tag{15}$$

In this case more distant fixations are more likely and λ_s is the mean distance between consecutive fixation during a transition and is also learnt from labelled data. D_{max} can be learnt from data as well, however we have set it simply to the diameter of the circular table on which the experiments were conducted. We perform importance sampling by using the described likelihood function in order to obtain the estimates $\hat{s}_1, \hat{l}_1, \hat{s}_2, \hat{l}_2, \ldots \hat{s}_L, \hat{l}_L$.

3.4.3.4 *Estimating Item of Interest Locations*

Once the partitioning of the fixation program is estimated, inferring the location of each item of interest is performed through maximum likelihood estimation. For each item of interest i we fit $\mathcal{N}_i(\mu_i, \Sigma_i)$ to the fixations segment $F_{\hat{s}_i:\hat{l}_i:}$ corresponding to the action A_i. Due to the noisy nature of the eye tracking signal we constrain Σ_i to be a diagonal matrix and so avoiding potential problems with overfitting to the fixation clusters.

The pseudo code in Alg. 1, which is optimised for clarity rather than efficiency, summarises the proposed algorithm. We have implemented it with the probabilistic

Algorithm 1: Inferring the locations of items of interest in the environment.

```
Input: $\mathcal{T}$, $\mathcal{A}$, $\mathcal{P}$, $\lambda_m$, $\lambda_s$, $D_{max}$
Data: $F_{1:T}$
Output: $E = \{(\mu_1, \Sigma_1), (\mu_2, \Sigma_2), \ldots, (\mu_M, \Sigma_M)\}$
1  // Find all possible plans for the task
2  $P_{\mathcal{T}} = \mathcal{P}(\mathcal{T})$
3  $samples \leftarrow []$
4  for $n$ from $1$ to $N_{samples}$ do
5      // Step 1: Generate actions
6      $P^n \leftarrow$ Sample $UniformCategorical(P_{\mathcal{T}})$
7      $L^n \leftarrow GetLength(P^n)$
8      $A^n_{1:L^n} \leftarrow GetActions(P^n)$
9      // Step 2: Generate fixation program segments
10     $\alpha^n \leftarrow 2L^n$
11     $[s^n_1, l^n_1, \ldots s^n_L, l^n_L] = T \times$ Sample $Dir(\alpha^n)$
12     // Step 3: Evaluate fixation program likelihood
13     $w^n \leftarrow 1.0$
14     for $i$ from $1$ to $T-1$ do
15         if $BothDuringAction(F_i, F_{i+1})$ then
16             $d_m \leftarrow ||F_i - F_{i+1}||$
17             $w^n \leftarrow w^n \times \lambda_m \exp(-\lambda_m d_m)$
18         if $BothDuringTransition(F_i, F_{i+1})$ then
19             $d_s \leftarrow D_{max} - ||F_i - F_{i+1}||$
20             $w^n \leftarrow w^n \times \lambda_s \exp(-\lambda_s d_s)$
21     // Store sample
22     $sample \leftarrow \{w^n : [s^n_1, l^n_1, \ldots s^n_L, l^n_L]\}$
23     Append $sample$ to $samples$
24 $[\hat{s}_1, \hat{l}_1, \ldots \hat{s}_L, \hat{l}_L] = WeightedAverage(samples)$
25 // Step 4: Estimate item of interest locations
26 $E \leftarrow []$
27 for $m$ from $1$ to $M$ do
28     $(\mu_m, \Sigma_m) = FitNormal(F_{\hat{s}_m:\hat{l}_m})$
29     Append $(\mu_m, \Sigma_m)$ to $E$
```

programming language Anglican [225] as it provides an elegant way of building generative probabilistic models and sampling from them. See Appendix A for a brief tutorial on probabilistic programming with Anglican.

3.4.4 *Instance Learning*

Once the symbol locations are estimated, the symbol appearances are learnt during the instance learning stage in order to enable automatic recognition. As suggested by [159], grounded fixations can be used as a noisy supervisory signal for any type of classifier. Additionally, since the estimated locations can be projected in the frame of any sensor it is possible to train multimodal classifiers as well. We propose a simple vision based algorithm for instance learning as an initial step towards the development of more sophisticated systems. Given a set of fixations $F_{s_i:l_i}$ which correspond to an item of interest, it is projected onto the image of a camera viewing the scene resulting in $f_{s_i:l_i}$. For each projected fixation, an image crop is made centred at the fixation with size proportional to the variance of $\mathcal{N}_i$. The difference between every two crops is computed in order to estimate the colour of the background. After that each crop is resized to enclose the largest foreground object it contains. The resized crops are used to compute the distribution over the colour and size of the symbol instance. This information is sufficient to detect the object in simple environments such as the table top setup. It should be noted that different sensors will lead to different appearance distributions and so a more general representation can be learnt by taking multiple sensors into account.

3.4.5 *Experimental Setup*

We have conducted human experiments where the PR2 robot is shown how to build a tower of five colour cubes such as the one in Fig. 12a. Each of the 5 participants demonstrated the task 10 times, working on a table top, while wearing eye tracking glasses which we have modified as can be seen in Fig. 15.

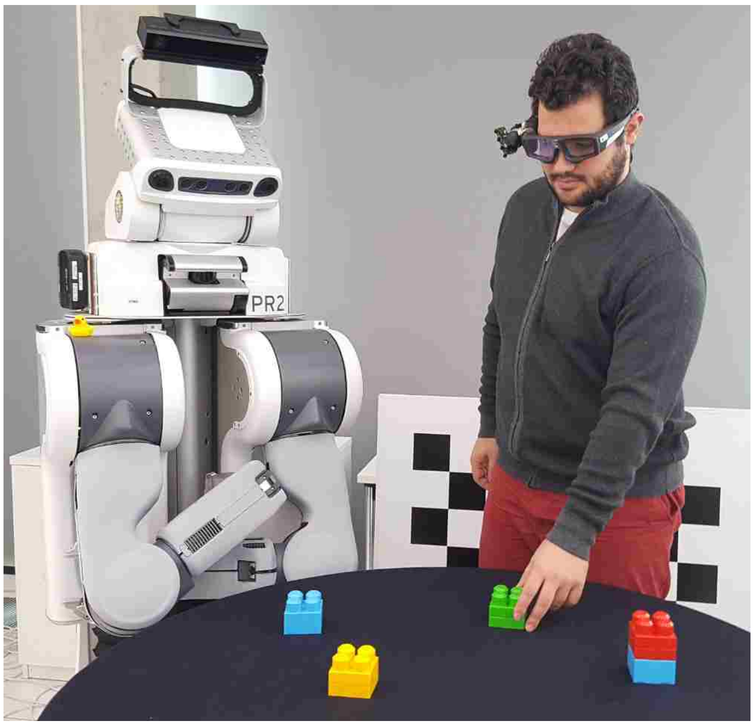


Figure 15: The experimental setup used for recording demonstrations of building a tower with five colour cubes. During the demonstration the person wears eye tracking glasses which we have modified to enable monocular SLAM based localisation.

We recorded eye tracking information as well as video feeds from multiple cameras together with the poses estimated by ORBSLAM. One of the cameras, which is fixed on the ceiling, is used for estimating ground truth locations of the cubes.

3.5 Experimental Results

3.5.1 *3D Eye Tracking*

The first step in the evaluation of GLIDE is to analyse the results obtained by the proposed 3D eye tracking methodology. Correct pose estimation of the eye tracking glasses is crucial for the projection of fixations on the environment, therefore we have tested multiple visual SLAM algorithms on the table top setup which we are interested in. ORBSLAM relies on image feature points and does not assume that they belong to a single plane. Thus, it is able to use feature points detected both on the table and in the environment which we found to be crucial. The only problems which we

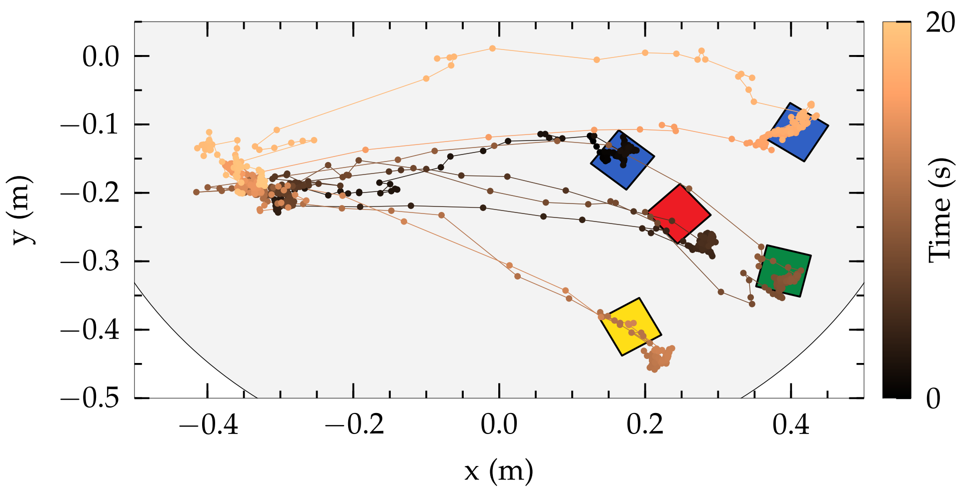


Figure 16: A typical fixation program recorded during a demonstration of how to build a tower of 5 colour cubes in the sequence of blue, red, green, yellow, blue (from bottom to top).

experienced were with people leaning over the table, looking at it from closer and so limiting the number of visible feature points. However, the issue was easily resolved by tilting the camera up slightly and recalibrating its transformation.

A typical fixation program recorded during a demonstration of how to build a tower of 5 colour cubes in the sequence of blue, red, green, yellow, blue (from bottom to top) is shown in Fig. 16. Firstly, it can be noticed that fixations are indeed clustered on items of interest for each action and are sparse in the transition stages, which is a key assumption in the proposed inference algorithm. We used one demonstration from each participant to estimate the overall mean distances between fixations for the two cases $\lambda_m = 0.81cm$ and $\lambda_s = 6.29cm$ respectively. As expected, λ_s is an order of magnitude greater than λ_m. This difference is crucial as it provides valuable information in order to align the plan with the observed fixations.

Furthermore, cluster locations are not directly on the cubes, but often at the edge or even slightly aside. On one hand, this can be explained by the approximation of projecting fixations on the table top, instead of the top side of each cube. This distortion effect can be easily noticed in the building area where each consequent cube violates the planar assumption stronger than the previous one and the fixations form a diagonal cluster. The person is usually positioned at approximately $(0.0, -0.6)$ which matches the angle of the diagonal cluster. On the other hand, people are known to fixate on task critical locations such as grasp points [101].

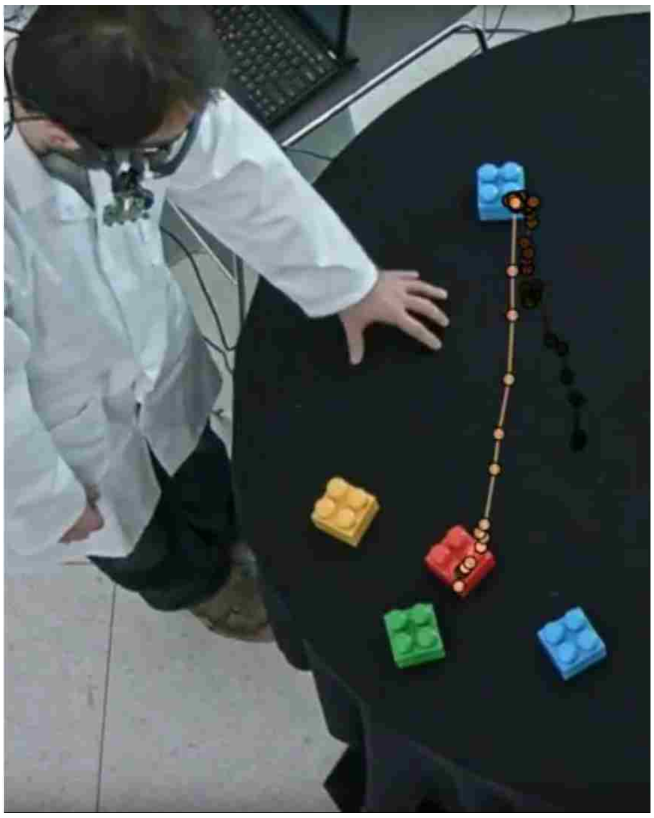


Figure 17: A short sequence of recorded 3D fixations from an experimental trial is projected into the ceiling camera frame of reference and overlayed on the captured image.

Another interesting feature is that trajectories from the first block to the building area follow almost straight line paths, while the last one is noticeably curved. This pattern is present in most of the fixation programs which we recorded and we attribute it to the fact that people change their focal plane during the transition, however we are not able to detect that and simply project the fixation onto the table top. This can be avoided by monitoring the 3D optical axis of each eye and find the intersection between them in order to truly estimate a 3D fixation. The SMI ETG 1 device provides such information however we found it to be extremely noisy. Therefore, we rely on the point of regard within the first person view camera image which is less noisy, but abrupt changes with large magnitude are often observed. One of the preprocessing steps that we employed is to remove any fixations which are out of the table surface.

Lastly, we demonstrate that once the fixation locations are estimated within the 3D environment they can be easily projected to the frame of any available sensor. In Fig. 17, we have overlayed on the image from the camera mounted on the ceiling a short fraction of the recorded fixations during one of the experimental trials.

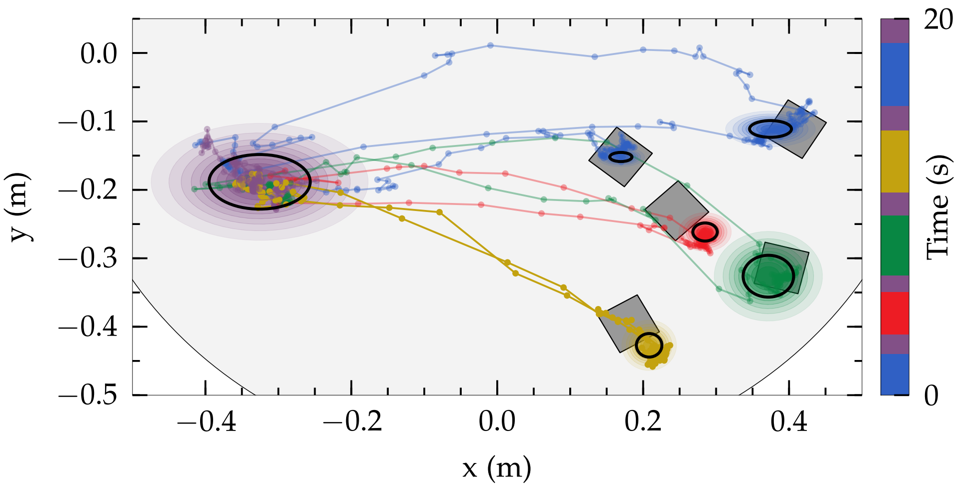


Figure 18: The results from running the proposed inference algorithm on the fixation program in Fig. 16 in order to determine the location of each symbol instance in the environment. The ellipses represent the inferred $\mathcal{N}_m(\mu_m, \Sigma_m)$ for each item of interest. The black ellipses correspond to a single standard deviation. The purple ellipse represents the location of the building area obtained by averaging the 5 building steps. The transition fixations are not explicitly visualised.

3.5.2 *Localisation of Symbol Instances*

Next we proceed with the evaluation of the proposed inference algorithm. The inferred locations from the fixation program in Fig. 16 are shown in Fig. 18. Each $\mathcal{N}_m(\mu_m, \Sigma_m)$ for each item m is visualised as an ellipse with the corresponding colour. The purple ellipse represents the estimated location for the building area which was calculated by averaging the locations of the five building steps.

It can be seen that the fixation program was segmented correctly and each symbol instance was successfully localised. The building area has the greatest variance which is expected as there are 5 actions depending on it. In order to evaluate the performance of the inference algorithm we have plotted in Fig. 19 the distribution of localisation error as a histogram by analysing all recorded demonstrations. The error is calculated independently for each cube and it is defined as the distance between the ground truth location estimated by a top view camera and the inferred location from the fixation program. 73% of the cubes are successfully localised with an error of less than $10cm$ which is comparable to the cube size of $6.5cm$.

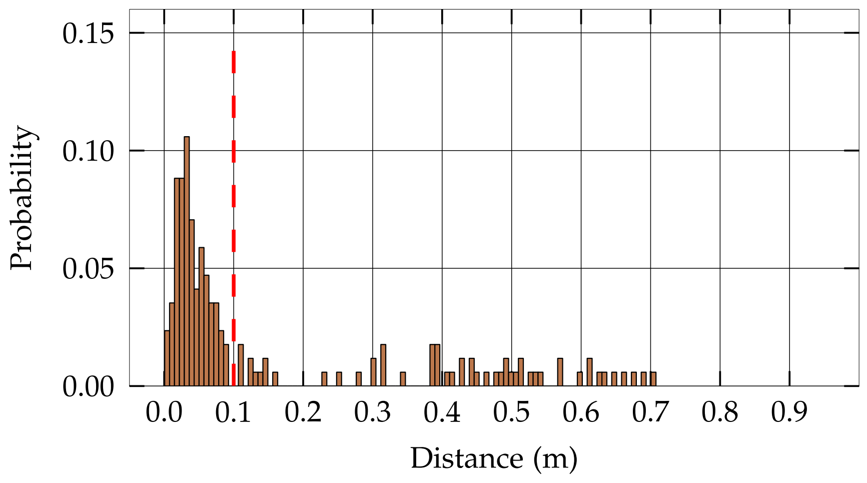
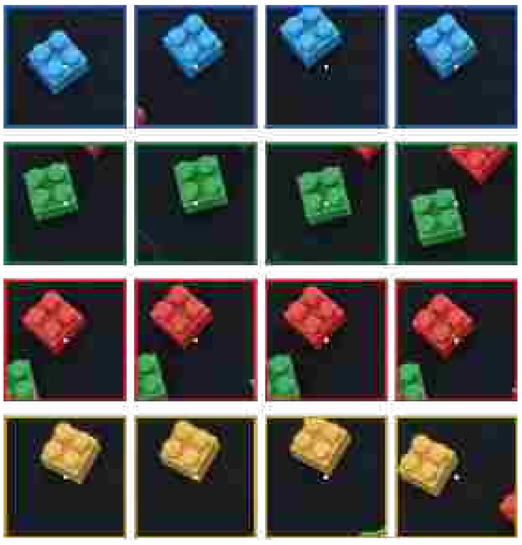


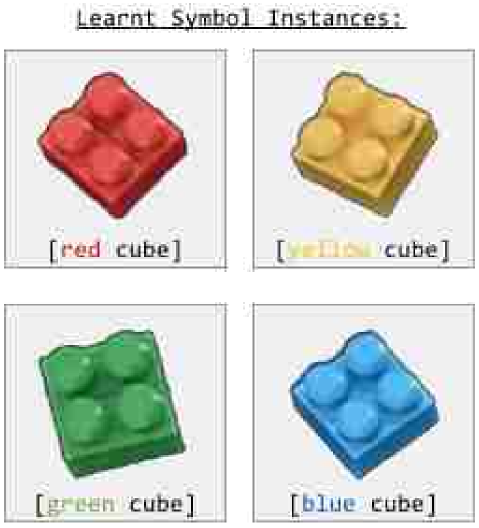
Figure 19: A probability mass distribution over the error in localising each of the cubes in each of the recorded demonstrations. The error is calculated as the distance between the ground truth locations and the inferred locations from the fixation programs.

Manual inspection of erroneous cases revealed that wrong localisation is predominantly caused by noisy eye tracking data that we have not filtered out. However, since we segment the fixation program and the beginning and the end are fixed, usually only a small number of item of interest locations are affected by the noise.

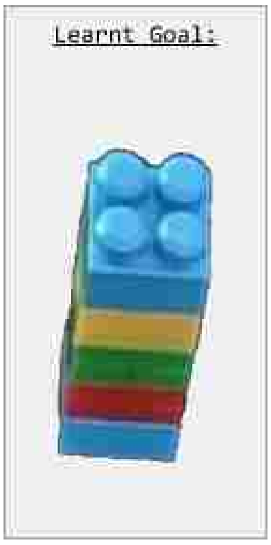
3.5.3 *Instance Learning*

The last step in the evaluation of the proposed framework is to examine the performance of the vision based instance learning algorithm. Given that the location of a symbol instance is inferred, we can take image crops around the corresponding fixations as shown in Fig. 20a. Each crop contains the item of interest being fixated with occasional additional items also partially visible due to the cluttered table top. Nevertheless, the proposed instance learning algorithm manages to filter the extra items out and calculates an appearance distribution over the size, colour and pixel values of each symbol instance. The mean value of each symbol instance is visualised in Fig. 20b. Manual visual inspection of the learnt appearance models for each cube throughout all demonstrations revealed that 71% of the instances were correctly learnt. Furthermore, the same approach can be used to learn the appearance of the task goal as demonstrated in Fig. 20c.


(a) Symbolic plan representation.



(b) Plan symbols and their physical instances.



(c) Task goal.

Figure 20: Images of symbol instances extracted by utilising the results from performing inference over the recorded fixation program **(a)**. Each of the crops is centred at a fixation belonging to the corresponding action segment. Those images are used to ground the plan symbols to their physical instances and learn their appearance **(b)**. The visualised instances are the mean of the learnt appearance distributions. The approach can also be used to learn the appearance of the task goal **(c)**.

There is an interesting trade off between the number of demonstrations, number of sensors and generalisation capabilities of the learnt appearance models. The ones shown in Fig. 20b are learnt from a single static camera from a single demonstration. While they look surprisingly similar to the real cubes, they have overfit the appearance of the symbol instance and slight changes in the point of view or lighting conditions will render them invalid. Therefore, it is an interesting problem on its own how to combine multiple sensors, possibly moving, with a number of demonstrations in order to arrive at a more general appearance model of the given symbol instance.

3.6 Concluding Remarks

In this work we have presented how the interpretation of high level instructions together with 3D eye tracking data can actively guide perception and enable robots to instantiate symbolic plans without prior knowledge about the symbol instances present in the environment. This type of situations are inevitable if we are to deploy autonomous robots in actual human environments so that they can improve our daily lives.

We have demonstrated that fixation programs in the context of plan execution provide not only information about where people look at, but also why they look there. This property has enabled us to develop a generative probabilistic model for fixation programs together with an efficient inference algorithm. The proposed instance learning algorithm attempts to answer the last question about what people see when they look at a particular location. Thus, GLIDE is able to interpret unknown symbol references present in single instructions or entire plans by mapping them to 1) a location within the environment and 2) a corresponding physical appearance. We demonstrated the capabilities of GLIDE on experimental data where 73% of the symbols were correctly localised and the physical appearance of 71% of the symbols was successfully learnt.

One of the main limitations of the proposed inference algorithm is that it will fail to segment the fixation program properly if informative actions such as searching are executed by the demonstrator. One way to resolve this issue is to add a prior over the number of fixation clusters in the fixation program. However, the planner should also be able to predict such actions. In fact, the planner adds another constraint which should be discussed. We assume that the planner is able to generate all possible plans for accomplishing the task. This is possible for tasks with low branching factor where the order is important. However, if we consider a task which has a high branching factor and the order does not matter then the number of plans grows exponentially rendering an exhaustive search approach infeasible. A potential solution of this issue is to keep a recursive estimate of the most probable plan currently being executed. This, however, will not work if no knowledge about the environment is available. Therefore, it is left for future work to integrate natural language instructions in GLIDE in order to learn the symbolic plan and then instantiate it in the environment.

In conclusion, we introduced GLIDE - a framework for simultaneous Grounding and Learning Instances through Demonstration and Eye tracking. We demonstrated that it successfully manages to ground symbols and learn their appearance by applying it to experimental data. The key insight was the definition of fixation programs as associated traces of demonstrations and fixation sequences. This enabled us to explore how eye tracking can guide the instantiation of high level plans as well as perception and environment understanding. Those are key capabilities necessary for the successful deployment of robots in human environments.

4

Perceptor Gradients

In the previous chapter we demonstrated how a simple sequential program can be used to guide perception and learn to recognise symbols important for the execution of the task. In this chapter we present a general approach which enables complex programs with data dependent information flows to guide the learning of symbolic representations. We pose the problem in a reinforcement learning framework and demonstrate the capabilities of learning to ground symbols both in continuous control as well as stochastic high level planning problems.

4.1 Introduction

One of the reasons for the widespread success of deep neural architectures is their ability to learn hierarchical representations of data with increasing levels of abstraction as the hierarchy is ascended [127]. On one hand, neural networks are particularly instrumental in training autonomous agents to execute various task ranging from manipulation [132, 133], locomotion [85, 163] and continuous control [46, 135] to playing games [79, 88, 146] and planning [162, 221]. On the other hand, Deep Reinforcement Learning (DRL) methods require large amounts of data, typically millions of observations [13, 46] and tend to overfit to the observed data distribution [197].

Learning structured representations of the environment has been shown to result in sample efficient methods able to learn transferable skills [53, 64, 79, 103]. However, it is still an open question how to bias the learning procedure such that it finds abstract representations that are relevant and useful for the task to be performed. Therefore, we formulate the problem of structured representation learning into a reinforcement learning framework ensuring that task execution guides the representation learning procedure.

We posit that task encoding programs provide strong inductive bias for learning symbolic representations of the environment as they capture the symbolic structure of the task. By symbolic representations we refer to representations that carry semantic content and so can be grounded to objects, relations or, in general, any pattern in the environment. Hybrid DRL methods based on imitation learning already use human demonstrations and supervision [38, 97], however it is far easier for many tasks to provide a high level program rather than recording demonstrations and extracting the demonstrator's intent from them.

In this chapter we introduce the perceptor gradients algorithm which decomposes a typical policy, assumed to be represented as a neural network, mapping from observations to actions into i) a perceptor network that maps observations to symbolic representations and ii) a provided task encoding program which is executed on the perceived symbols in order to generate an action. We consider both a feedforward as well as an autoencoding perceptor where the program acts as a regulariser on the latent space. Imposing such a regulariser, not only provides a strong inductive bias encouraging disentaglement of the latent features, but also resolves the non-identifiability issues inherent in most of the existing autoencoders.

We prove that the perceptor can be trained with *any* task encoding program using the REINFROCE estimator [223]. The task encoding program is not required to be differentiable and can be of arbitrary structure containing branches and loops; the only requirement is that it always halts producing an action for a set of input symbols in a finite time.

We apply the perceptor gradients algorithm to the problem of balancing a cart-pole system with an Linear Quadratic Controller (LQR) from pixel observations and demonstrate that the learnt representations directly correspond to task related symbols required by the LQR such as linear position and angle. Furthermore, we demonstrate similar results in a Minecraft-like environment, where we consider the problems of navigating and finding an item from a 2.5D rendering of the world using a general purpose A* planner as the task encoding program. Since the learnt latent dimensions are disentangled and identifiable we show that we can generate new structured observations according to a symbolic specification.

The main contributions presented in this chapter are:

- A representation learning method with programmatic regularisation capable of learning symbolic task related representations.
- A training procedure that we prove to support any task encoding program.

4.2 Related Work

4.2.1 *Statistical Disentanglement*

The most general methods for learning disentangled representations are based on the VAE [109] and modify the loss function in order to impose some statistical properties on the learnt approximation distribution such as maximising component independence [90] or minimising total correlation [36, 107]. Assuming different priors over subsets of the latent variables allows for additional shaping of the latent space [36, 37]. Additionally, carefully crafted curriculum learning procedures can also result in disentangled representations [119].

It is not clear, however, why independence driven statistical properties would result in structured representations which are useful for the execution of a particular task. For example, a large body of literature in cognitive science provides evidence that context has a significant impact on perception and task performance [154] suggesting that the brain relies on correlations in the observed patterns.

4.2.2 *Model-Based Disentanglement*

Model-based methods for disentangling latent factors are better suited for learning task related representations as they leverage particular domain knowledge. Even though model-based disentanglement is not an explicit research sub-field there is a lot of work falling in this class.

A large fraction of the work is focused on learning latent representations which are suitable for predicting and controlling physical systems [99, 220]. Watter et al. [220] embed images into a latent state space where they impose locally linear structure of the trajectory and learn to control various physical systems. Fraccaro et al. [60] consider the problem of predicting the evolution of a physical system and propose the combination of a linear Gaussian state space model with a VAE, called Kalman Variational Autoencoder, in order to learn a latent space where appearance and dynamics are disentangled. The idea has been extended to Bayesian filters in general, where the key requirement is that the transition function is differentiable [106]. Another approach is to shape the latent space according to a physical model as proposed by Bezenac, Pajot and Gallinari [22]. They consider the problem of forecasting sea surface temperature and leverage a theoretical physical model in order to learn a disentangled latent space corresponding to the velocity field advecting the temperature. Moreover, if the observation space has a known partitioning then this information can also be used for learning disentangled representations [6, 25].

Model-based disentanglement methods have been applied to scene understanding as well. By explicitly reasoning about the object in the scene as well as the point of view, one can learn disentangled representations of the scene both for 2D [54] and 3D [53] scenes. Furthermore, Denton et al. [43] learn to disentangle pose and content from video by proposing an adversarial loss framework which implicitly encodes the fact that space-time is smooth and continuous.

Of particular relevance to our work is the idea of learning compositional visual concepts by enforcing the support of set theoretic operations such as union and intersection in the latent space [91]. Furthermore, Kurutach et al. [122] demonstrate how to perform planning in a latent space learnt from demonstrations.

We propose to use programs as a general model description which, in theory, should allow for many of the aforementioned models to be reproduced in a single framework.

4.2.3 *Guided Learning*

Hybrid DRL methods based on imitation learning already rely on human supervision or demonstrations. However, in many cases it might be much easier to simply provide a high level program describing the task.

For example, natural language instructions provide coarse programmatic structure and have been shown to achieve state of the art performance results on the Montezuma's Revenge game [105]. Simple high-level programmatic fragments such as `if hasKey: unlockDoor() else: fetchKey()` can significantly relax the problems related to delayed credit assignment. Furthermore, this fragment clearly indicates to the learner that the predicate `hasKey` is a useful one to learn.

In many cases the difficulty in an RL problem arises from perceiving the right symbols, rather than complex decision making. Li et al. [134] show that the underlying complexity of many problems that we try to learn end-to-end is much smaller than the parameter space of the model which is trained. For example, they demonstrate that the problem of training a neural network to balance a cart-pole system has an intrinsic dimensionality of size 4, regardless of the number of weights. This result is not surprising at all since the underlying dynamic system has a state space of size 4. The task of controlling a cart-pole system is well studied and can be optimally performed by a linear quadratic controller, however extracting the relevant information from pixels is a problem well suited for neural networks.

Learning semantically disentangled representations is equivalent to learning a vocabulary of symbols which can be used to construct expressions in a language compatible with the underlying mechanisms that generated the observed data. The vocabulary of learnt symbols has a direct impact on the type of programs which can be expressed and so the type of problems that can be solved. Importantly, hard problems can often be easily solved by constructing simple programs which utilise symbols that exploit the structure of the given problem. For example, describing the constraint motion of a rigid body (e.g. a pendulum) with Newton's laws of motion is much more cumbersome, than simply defining the Lagrangian function using potential and kinetic energy.

4.2.4 *Neuro-symbolic Architectures*

While there are numerous neural network architectures that are able to recognise symbolic structures such as, for example, spatial relations [186, 209, 236] there are only a few hybrid neuro-symbolic architectures proposed that combine the learning capabilities of neural networks and the generalisation power of symbolic methods.

Gaunt et al. [65] introduce a framework for differentiable programming with neural libraries. They embed neural network blocks within differentiable programs and perform program induction and training of the neural blocks jointly. Their results indicate that learning with programmatic structure induces inductive bias that favours strong generalisation capabilities.

Neuro-symbolic methods have also been proposed for RL [63, 64], however these methods are still in their infancy imposing a large number of assumptions and relying on ad-hoc feature engineering. Moreover, they are yet to be applied even to some simple and common tasks such as balancing a pendulum or playing ATARI pong. Another neuro-symbolic method can be found in [62], however the policy decomposition, which we propose, to a perceptor network and task encoding programs allows the perceptor gradients algorithm to be applied to various tasks, not just image rendering.

At its core, the proposed perceptor gradients algorithm is a neuro-symbolic approach as we utilise both neural networks as universal learners [42] and programs to guide the learning process.

4.2.5 *Structured Reinfrocement Learning*

As already discussed, we cast the problem of learning task related representations into an RL framework and insert a task encoding program into the training loop. Even though our work is focused on learning symbolic representations it is also related to some research on structured reinforcement learning.

Firstly, Ha and Schmidhuber [79] learn predictive world models that operate in a latent space obtained from an autoencoder. They, however, train the autoencoder and the predictive model one after the other, instead of jointly, thus not leveraging the possibility to learn task related representations.

Secondly, Garnelo, Arulkumaran and Shanahan [64] propose a neuro-symbolic architecture for RL which can learn both useful representations and simple symbolic rules at the cost of large number of assumptions. While we do not learn symbolic rules, we support any program and limit our assumptions to the standard ones made by the policy gradients algorithm.

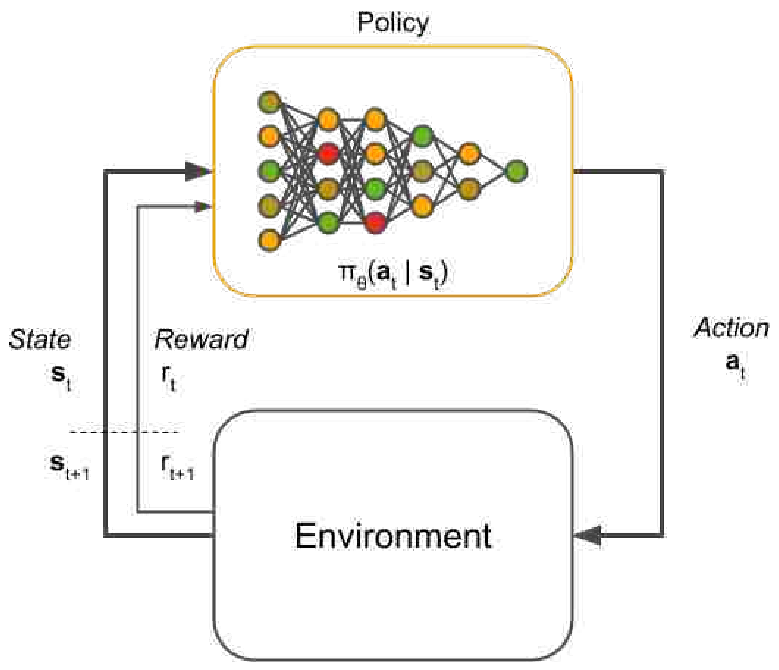
Lastly, the idea of programmatically interpretable RL is introduced by Verma et al. [216]. They induce a program from a pre-trained black-box policy where the program assumes that symbolic representations of the world are already available. We study this problem in chapter 5 as it is complement to the method proposed in this chapter.

4.3 Problem Definition

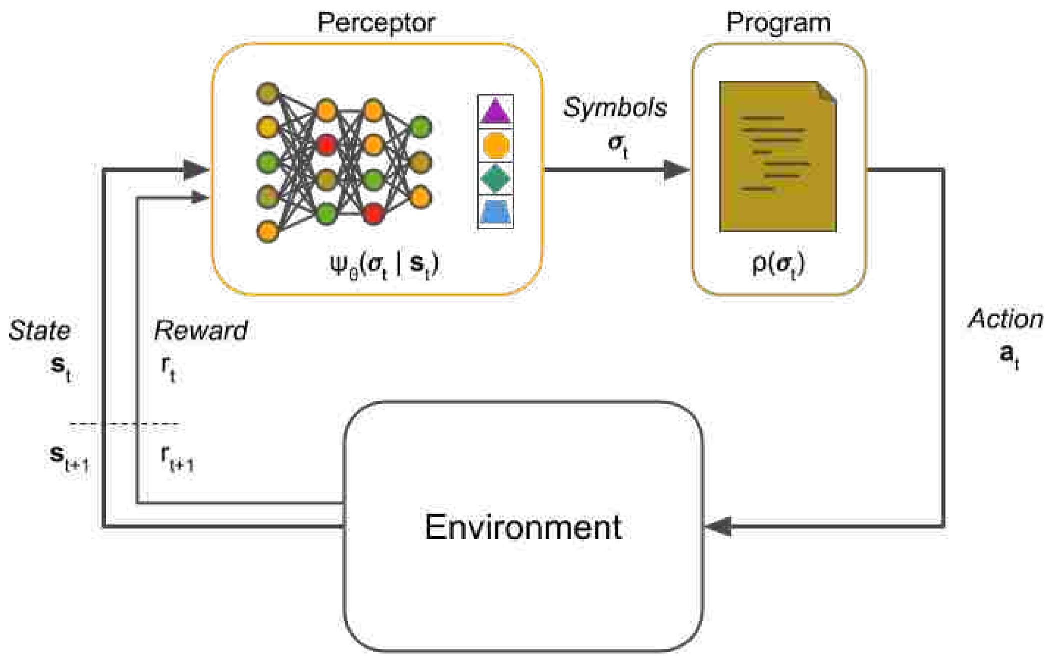
Let us consider the Markov Decision Process (MDP) represented as the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, P_0)$ where $\mathcal{S}$ is the set of possible states, $\mathcal{A}$ is the set of possible actions, P is the state transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, γ is the reward discounting factor and P_0 is the probability distribution over the initial states. At each time step t, the agent observes a state $\mathbf{s}_t$ and chooses an action $\mathbf{a}_t$ which results in a certain reward r_t. The problem, which standard RL is concerned with, is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$, mapping states to actions, such that it maximises the discounted total reward $R = \sum_{t=0}^{T} \gamma^t r_t$ known as the return, where we have assumed a finite time horizon of length T.

In particular, we consider the stochastic policy $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$ belonging to a family of functions parameterised by θ such that $\pi_\theta : \mathcal{S} \to \mathcal{P}_{\mathcal{A}}(\mathcal{A})$ where $\mathcal{P}_{\mathcal{A}}$ is a probability measure over the set $\mathcal{A}$. The standard RL setup is depicted in Fig. 21a.

We are interested in decomposing the policy π_θ into i) a perceptor $\psi_\theta : \mathcal{S} \to \mathcal{P}_\Sigma(\Sigma)$, where Σ is a set of task related symbols and $\mathcal{P}_\Sigma$ is a probability measure over it, and ii) a task encoding program $\rho : \Sigma \to \mathcal{A}$ such that $\rho \circ \psi_\theta : \mathcal{S} \to \mathcal{A}$. Intuitively, ρ



(a) The standard RL setup where a policy outputs an action per observed state.



(b) The proposed policy decomposition into a perceptor and a program. The perceptor outputs a set of symbols per observed state which are processed by the program in order to generate an action.

Figure 21: Comparison between the standard RL setup and the proposed policy decomposition into a perceptor and a program.

can be viewed as any functional program which is not required to be pure as nothing in our approach prevents it from internally using a block of memory for storing information or even communicating with external devices. This policy decomposition is shown on Fig. 21b.

4.4 Method

Firstly, we provide a quick derivation of the policy gradients algorithm which serves as the basis of the perceptor gradients algorithm that we propose. Secondly, we prove that the perceptor ψ_θ can be trained using the REINFORCE estimator [223] for any program ρ.

4.4.1 *Policy Gradients*

Starting from an initial state of the environment $\mathbf{s}_0 \sim P_0(\mathbf{s})$ and following the policy π_θ for T time steps results in a trace $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)$ of $T+1$ state-action pairs and T reward values $(r_1, r_2, \dots, r_T)$ where $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$. The return value R_t of state $\mathbf{s}_t$ in a trace τ is then defined as:

$$R_t(\tau) = \sum_{i=t}^{T} \gamma^{i-t} r(\mathbf{s}_t, \mathbf{a}_t) \tag{16}$$

The probability of a certain trace depends both on the rolled out policy to generate the trace as well as the dynamics of the environment:

$$p(\tau;\theta) = p(\mathbf{s}_0) \prod_{i=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \tag{17}$$

The objective of the policy gradients algorithm is to find an optimal policy π_θ^* by searching over the parameter space, such that the total expected return is maximised. Formally, the total expected return is

$$J(\theta) = E_{\tau \sim p(\tau;\theta)}\left[R_0(\tau)\right] = \int p(\tau;\theta) R_0(\tau)\, d\tau \tag{18}$$

and so we would like to solve

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \int p(\tau;\theta) R_0(\tau)\, d\tau \tag{19}$$

Following the derivation from [223], the gradient of $J(\theta)$ can be expressed as

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int p(\tau;\theta) R_0(\tau)\, d\tau && (20)\\ &= \int \nabla_\theta p(\tau;\theta) R_0(\tau)\, d\tau && (21)\\ &= \int p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) R_0(\tau)\, d\tau && (22)\\ &= E_{\tau\sim p(\tau;\theta)}\left[\nabla_\theta \log p(\tau;\theta) R_0(\tau)\right] && (23)\end{aligned}$$

The expected value in (23) can be approximated by Monte Carlo sampling. In order to do that, the policy π_θ is rolled out for n episodes

$$\nabla_\theta J(\theta) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p(\tau^{(i)};\theta) R_0(\tau^{(i)}) \tag{24}$$

resulting in algorithm known as n-step policy gradients [189]. Using (17), the gradient of the log-likelihood of a trace sample $\tau^{(i)}$ can be expressed as

$$\begin{aligned}&\nabla_\theta \log p(\tau^{(i)};\theta) = \\ &\quad= \nabla_\theta \log \left[p(\mathbf{s}_0^{(i)}) \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}^{(i)}|\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) \right] && (25)\\ &\quad= \nabla_\theta \left[\log p(\mathbf{s}_0^{(i)}) + \sum_{t=0}^{T-1} \log p(\mathbf{s}_{t+1}^{(i)}|\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) + \sum_{t=0}^{T-1} \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) \right] && (26)\\ &\quad= \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) && (27)\\ &\quad= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) && (28)\end{aligned}$$

Importantly, the calculation of the gradient does not depend on the dynamics model of the environment. It depends only on the gradient of the policy with respect to its parameters. Given that we can select the parametric family of the policy, the gradient should be possible to compute. For example, if the policy is represented by a neural network, then the gradient can be estimated by backpropagation. By combining (24) and (28), one obtains the gradient of the expected return:

$$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) R_0(\tau^{(i)}) \tag{29}$$

In order to reduce the variance of the estimator Williams [223] introduced the notion of return baseline which in its most general form is a function $b_\phi(\mathbf{s}_t)$ that produces a baseline value per state and the parameters ϕ can be learnt as well. Additionally, causal constraints can also be applied to (29) since the return at time t depends only on consecutive actions, not on previous ones. These modifications lead to the improved estimator

$$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)} | \mathbf{s}_t^{(i)}) \left(R_t(\tau^{(i)}) - b_\phi(\mathbf{s}_t^{(i)}) \right) \tag{30}$$

The quantity $A_t = \left(R_t(\tau^{(i)}) - b_\phi(\mathbf{s}_t^{(i)}) \right)$ is also known as the advantage of action $\mathbf{a}_t$.

4.4.2 *Policy Decomposition*

Let us now consider the following factorisation of the policy

$$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t) = p(\mathbf{a}_t | \boldsymbol{\sigma}_t) \psi_\theta(\boldsymbol{\sigma}_t | \mathbf{s}_t) \tag{31}$$

where $\boldsymbol{\sigma}_t \in \Sigma$ are the symbols extracted from the state $\mathbf{s}_t$ by the perceptor ψ_θ. In general $p(\mathbf{a}_t | \boldsymbol{\sigma}_t)$ could be any distribution, however, we are interested in exploiting the programmatic structure supported by the symbols $\boldsymbol{\sigma}_t$ and so we set the distribution as follows

$$p(\mathbf{a}_t | \boldsymbol{\sigma}_t) = \delta_{\rho(\boldsymbol{\sigma}_t)}(\mathbf{a}_t) \tag{32}$$

which is a Dirac delta distribution centered on the output of the task encoding program $\rho(\boldsymbol{\sigma}_t)$ for the input symbols $\boldsymbol{\sigma}_t$. Even though the program should produce a single action, it could internally work with distributions and simply sample its output.

Decomposing the policy into a program and a perceptor

$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \delta_{\rho(\boldsymbol{\sigma}_t)}(\mathbf{a}_t)\ \psi_\theta(\boldsymbol{\sigma}_t|\mathbf{s}_t) \tag{33}$$

enables the description of programmatically structured policies, while being able to learn the required symbolic representation from data.

4.4.3 *Perceptor Gradients*

Theorem 4.1 (Perceptor Gradients) *For any decomposition of a policy π_θ into a program ρ and a perceptor ψ_θ such that*

$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \delta_{\rho(\boldsymbol{\sigma}_t)}(\mathbf{a}_t)\ \psi_\theta(\boldsymbol{\sigma}_t|\mathbf{s}_t) \tag{34}$$

the gradient of the log-likelihood of a trace sample $\tau^{(i)}$ obtained by following π_θ is

$$\nabla_\theta \log p(\tau^{(i)};\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \psi_\theta(\boldsymbol{\sigma}_t^{(i)}|\mathbf{s}_t^{(i)}) \tag{35}$$

Proof. Combining (28) and (34) gives

$$\begin{aligned}
\nabla_\theta \log p(\tau^{(i)};\theta) &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)}|\mathbf{s}_t^{(i)}) \\
&= \sum_{t=0}^{T-1} \nabla_\theta \log \left[\delta_{\rho(\boldsymbol{\sigma}_t^{(i)})}(\mathbf{a}_t^{(i)})\ \psi_\theta(\boldsymbol{\sigma}_t^{(i)}|\mathbf{s}_t^{(i)})\right] && (36)\\
&= \sum_{t=0}^{T-1} \left[\nabla_\theta \log \delta_{\rho(\boldsymbol{\sigma}_t^{(i)})}(\mathbf{a}_t^{(i)}) + \nabla_\theta \log \psi_\theta(\boldsymbol{\sigma}_t^{(i)}|\mathbf{s}_t^{(i)})\right] && (37)\\
&= \sum_{t=0}^{T-1} \nabla_\theta \log \psi_\theta(\boldsymbol{\sigma}_t^{(i)}|\mathbf{s}_t^{(i)}) && (38)
\end{aligned}$$

■

Theorem 4.1 has an important consequence – no matter what program ρ we choose, as long as it outputs an action in a finite amount of time, the parameters θ of the perceptor ψ_θ can be learnt with the standard policy gradients algorithm. Intuitively,

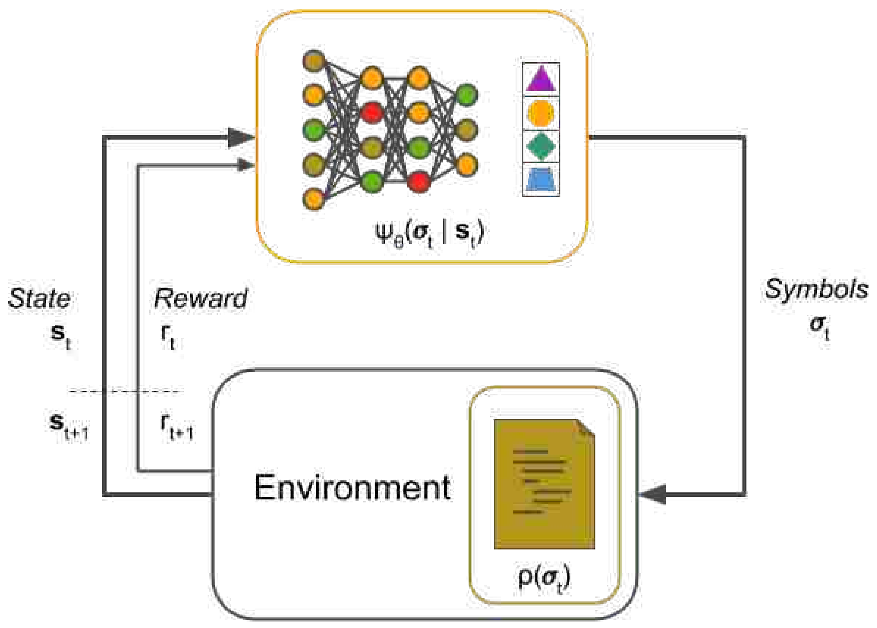


Figure 22: Theorem 4.1 can be intuitively understood by imagining that the program is absorbed into the environment, which is treated as a black box, and the actions are actually recognised symbols.

the program applies a fixed transformation on the output of the perceptor and so it can be thought of as being absorbed by the environment as illustrated in Fig. 22.

4.4.4 *Feedforward Perceptor*

From theorem 4.1 and (30), we derive the following loss function

$$\mathcal{L}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}\left\{\mathcal{L}_\psi(\tau^{(i)}, \theta) + \mathcal{L}_b(\tau^{(i)}, \phi)\right\} \tag{39}$$

where

$$\mathcal{L}_\psi(\tau^{(i)}, \theta) = \sum_{t=0}^{T-1} \log \psi_\theta(\boldsymbol{\sigma}_t^{(i)} | \mathbf{s}_t^{(i)}) \left(R_t(\tau^{(i)}) - b_\phi(\mathbf{s}_t^{(i)})\right) \tag{40}$$

$$\mathcal{L}_b(\tau^{(i)}, \phi) = \sum_{t=0}^{T-1} \left(R_t(\tau^{(i)}) - b_\phi(\mathbf{s}_t^{(i)})\right)^2 \tag{41}$$

$$\tag{42}$$

and the rolled out trajectory

$$\tau^{(i)} = (\mathbf{s}_0, \boldsymbol{\sigma}_0, \mathbf{s}_1, \boldsymbol{\sigma}_1, \ldots, \mathbf{s}_T, \boldsymbol{\sigma}_T) \tag{43}$$

contains $T+1$ sampled state-symbol pairs. Optimising (39) with respect to θ and ϕ is performed by backpropagation and stochastic gradient descent.

Algorithm 2 demonstrates how to rollout a policy decomposed into a perceptor and program in order to obtain trajectory samples $\tau^{(i)}$.

Algorithm 2: Perceptor rollout for a single episode

```
    Input: $\psi_\theta$, $\rho$
    Output: $\tau$, $r_{1:T}$
1   for $t = 0$ to $T$ do
2       $\mathbf{s}_t \leftarrow$ observe environment
3       $\boldsymbol{\sigma}_t \leftarrow$ sample from $\psi_\theta(\boldsymbol{\sigma}|\mathbf{s}_t)$
4       $\mathbf{a}_t \leftarrow \rho(\boldsymbol{\sigma}_t)$
5       $r_t \leftarrow$ execute $\mathbf{a}_t$
6       append $(\mathbf{s}_t, \boldsymbol{\sigma}_t)$ to $\tau$
```

The overall perceptor gradients learning procedure is summarised in Alg. 3.

Algorithm 3: Perceptor gradients

```
1   $(\theta, \phi) \leftarrow$ Initialise parameters
2   repeat
3       for $i = 1$ to $n$ do
4           $\tau^{(i)}, r^{(i)}_{1:T} \leftarrow$ rollout$(\psi_\theta, \rho)$
5           for $t = 0$ to $T$ do
6               $R^{(i)}_t \leftarrow \sum_{i=t}^{T} \gamma^{i-t} r^{(i)}_t$
7               $A^{(i)}_t \leftarrow \left(R^{(i)}_t - b_\phi(\mathbf{s}^{(i)}_t)\right)$
8       $\mathcal{L}_\psi \leftarrow \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} \log \psi_\theta(\boldsymbol{\sigma}^{(i)}_t|\mathbf{s}^{(i)}_t) A^{(i)}_t$
9       $\mathcal{L}_b \leftarrow \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1} \left(R^{(i)}_t - b_\phi(\mathbf{s}^{(i)}_t)\right)^2$
10      $\mathcal{L} \leftarrow \mathcal{L}_\psi + \mathcal{L}_b$
11      $\mathbf{g} \leftarrow \nabla_{\theta,\phi}\mathcal{L}(\theta, \phi)$
12      $(\theta, \phi) \leftarrow$ Update parameters using $\mathbf{g}$
13  until convergence of parameters $(\theta, \phi)$;
```

4.4.5 *Autoencoding Perceptor*

The perceptor is a mapping such that $\psi_\theta : \mathcal{S} \to \mathcal{P}_\Sigma(\Sigma)$ and so if we augment it with a generator $\omega_v : \Sigma \to \mathcal{P}_\mathcal{S}(\mathcal{S})$, parameterised by v, we should be able to generate states (observations) from a structured symbolic description. Thus ψ_θ and ω_v form an autoencoder, where the latent space is Σ. Furthermore, the resulting autoencoder can be viewed as a variational autoencoder and efficiently trained using the reparameterisation trick [109] as the sampled values for $\boldsymbol{\sigma}_t$ during the perceptor rollout can be reused in the training of the generator ω_v.

We augment the loss in (39) with a reconstruction term such that

$$\mathcal{L}_{auto}(\theta, \phi, v) = \frac{1}{n} \sum_{i=1}^{n} \left\{ \mathcal{L}_\psi(\tau^{(i)}, \theta) + \mathcal{L}_b(\tau^{(i)}, \phi) + \mathcal{L}_\omega(\tau^{(i)}, v) \right\} \tag{44}$$

where

$$\mathcal{L}_\omega(\tau^{(i)}, \theta) = \sum_{t=0}^{T-1} \log \omega_v(\mathbf{s}_t^{(i)} | \boldsymbol{\sigma}_t^{(i)}) \tag{45}$$

and $\mathcal{L}_\psi$, $\mathcal{L}_b$ and $\tau^{(i)}$ are defined as before.

4.5 Experimental Results

4.5.1 *Cart-Pole Balancing*

We first consider the problem of balancing a cart-pole system by learning symbolic representations from the raw image observations. The cart-pole system is well studied in optimal control theory and it is typically balanced with a Linear Quadratic Controller (LQR) [239]. We exploit this knowledge and set the program ρ to implement an LQR. The perceptor ψ_θ is a convolutional nerual network that learns the state space representation of the observed images and feeds it into the program in order to generate a control action. A diagram of the overall experiment is shown in Fig. 23.

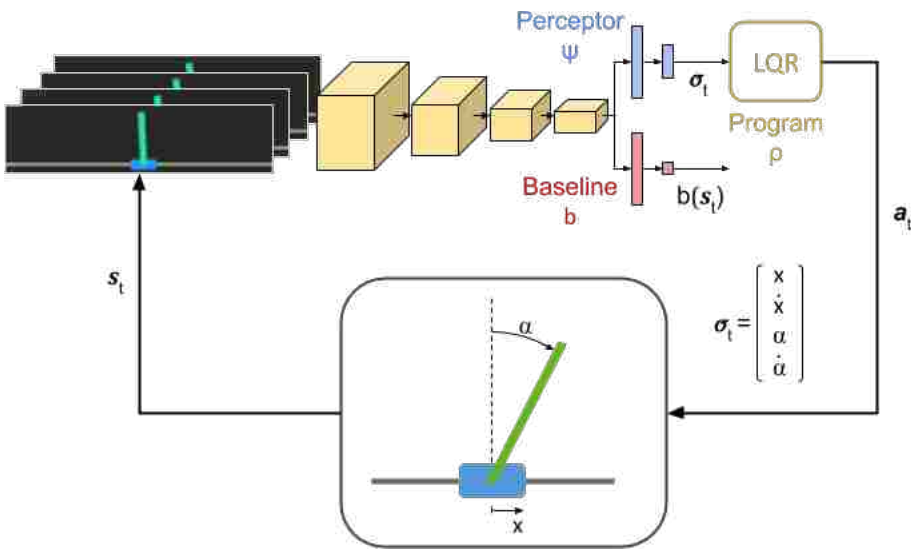


Figure 23: A diagram of the cart-pole experimental setup. The perceptor extracts the state of the system from images and feeds it into an LQR controller in order to balance the system.

4.5.1.1 *State Space Model*

We define the state vector as

$$\boldsymbol{\sigma} = \begin{bmatrix} x & \dot{x} & \alpha & \dot{\alpha} \end{bmatrix}^T \tag{46}$$

where $x \in \mathbb{R}$ is the linear position of the cart and $\alpha \in \mathbb{R}$ is the angle of the pendulum with respect to its vertical position as shown in Fig. 23. By following the derivation in [124] of the linearised state space model of the system around the unstable equilibrium $[0\ 0\ 0\ 0]^T$ (we ignore the modelling of the gearbox and the motor) we set the system matrix as

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{gml}{LM-ml} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{L-ml/M} & 0 \end{bmatrix} \tag{47}$$

and the input matrix as

$$B = \begin{bmatrix} 0 \\ \frac{1}{M-ml/L} \\ 0 \\ -\frac{1}{ML-ml} \end{bmatrix} \tag{48}$$

where m is the mass of the pole, M is the mass of the pole and the cart, l is half of the pendulum length, g is the gravitational acceleration, L is set to $\frac{I+ml^2}{ml}$ and $I = \frac{ml^2}{12}$ is the moment of inertia of the pendulum. We use the cart-pole system from OpenAI gym where all the parameters are pre-specified in the source code, namely $m = 0.1$, $M = 1.0$ and $l = 0.5$.

4.5.1.2 *LQR Program*

Given the state space representation of the system we can design a feedback control law $u = -K\boldsymbol{\sigma}$, where u corresponds to the force applied to the cart and K is a gain matrix. In an LQR design, the gain matrix K is found by minimising the quadratic cost function

$$J = \int_0^\infty \boldsymbol{\sigma}(t)^T Q \boldsymbol{\sigma}(t) + u(t)^T R u(t) \; dt \tag{49}$$

where Q and R are parameters penalising certain states of control inputs. We set them to $Q = 10^3\, I_4$, where I_4 is a 4×4 identity matrix, and $R = 1$ as proposed in [124]. We estimated $K = [-1 \;\; -2.25 \;\; -30.74 \;\; -7.07]$ by using the `python-control` package [1].

We finally set the program ρ to be

$$\rho(\boldsymbol{\sigma}) = \mathbf{a} = \begin{cases} 1 & \text{if } -K\boldsymbol{\sigma} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

resulting in 2 discrete actions according to the specification of the OpenAI gym cart-pole environment.

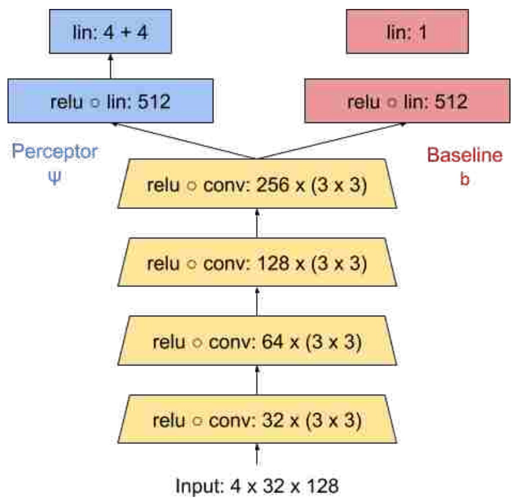

lin: 4 + 4
lin: 1
relu ∘ lin: 512
relu ∘ lin: 512
Perceptor
Ψ
Baseline
b
relu ∘ conv: 256 x (3 x 3)
relu ∘ conv: 128 x (3 x 3)
relu ∘ conv: 64 x (3 x 3)
relu ∘ conv: 32 x (3 x 3)
Input: 4 x 32 x 128

Figure 24: Architecture of the perceptor ψ_θ and the baseline network b_ϕ. Convolutions are represented as #filters $\times$ filter size and all of them have a stride of 2. Linear layers are denoted by the number of output units.

4.5.1.3 *Feedforward Perceptor*

The input of the perceptor is a stack of 4 consecutive grayscale 32×128 images that we render the cart-pole system onto as shown in Fig. 23. This is a setup similar to the one proposed in [146] which preserves temporary information in the input such that it can be processed by a convolutional neural network. The architecture of the perceptor ψ_θ is shown in Fig. 24. Note that the perceptor shares its convolutional layers with the baseline network b_ϕ. The output of the perceptor are the mean and the diagonal covariance matrix of a 4-dimensional normal distribution.

4.5.1.4 *Learning Performance*

In this experimental setup the perceptor is able to learn from raw image observations the symbolic representations required by the LQR controller. The average reward obtained during training is shown in Fig. 25. We compare the performance of the perceptor gradients algorithm to a standard policy gradients algorithm, where we have replaced the program with a single linear layer with sigmoid activation. The perceptor obtains an average reward close to the maximum of 199 approximately after 3000 iterations compared to 7000 iterations for the standard policy, however the obtained reward has greater variance. Intuitively this can be explained with the fact

1 http://python-control.org

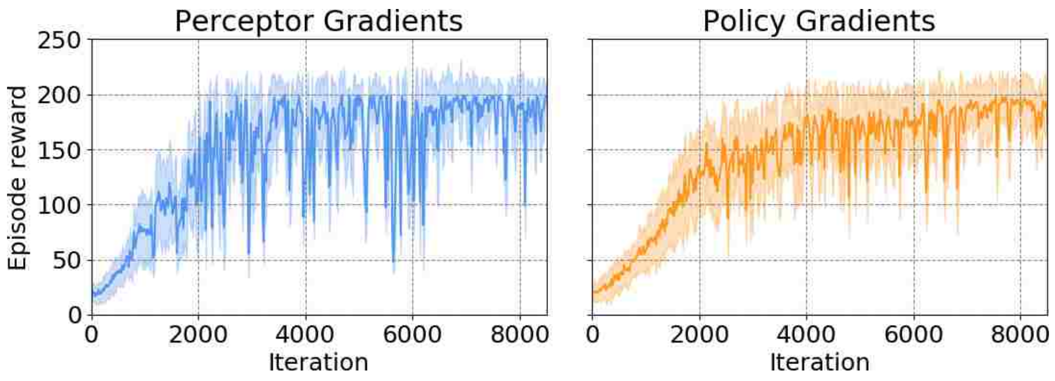


Figure 25: Learning performance at the cart-pole balancing task of the perceptor gradients algorithm (left) compared to standard policy gradients (right).

that the program encodes a significant amount of knowledge about the task which speeds up the learning, but also defines a much more constrained manifold for the latent space that is harder to be followed during stochastic gradient descent.

4.5.1.5 *Perceptor Latent Space*

Given that the perceptor has learnt representations that lead to the successful balancing of the cart-pole system one should expect that the latent space matches the symbolic state space representation of the system. Careful analysis of (49) reveals that this is the case, however, the relationship between the latent space and the symbolic state representation is governed by an additional transformation. Given that Q is a matrix of the form λI_4 where λ is a scalar, then any rotation matrix M applied to $\boldsymbol{\sigma}$ will have no impact on the cost because

$$(M\boldsymbol{\sigma})^T \lambda I_4 M\boldsymbol{\sigma} = \boldsymbol{\sigma}^T M^T \lambda I_4 M\boldsymbol{\sigma} = \boldsymbol{\sigma}^T \lambda I_4 M^T M\boldsymbol{\sigma} = \boldsymbol{\sigma}^T \lambda I_4 \boldsymbol{\sigma} \tag{51}$$

since rotation matrices are orthogonal. Furthermore, scaling the cost function will not shift the locations of the optima. Therefore, the latent representations learnt by the perceptor are the same as the symbolic state representation up to scale and rotation. We use one episode of labelled data to find this linear transformation through constrained optimisation.

The transformed output of the perceptor for an entire episode is shown in Fig. 26. The position and angle representations learnt by the perceptor match very closely the ground truth. However, both the linear and angular velocities do not match the

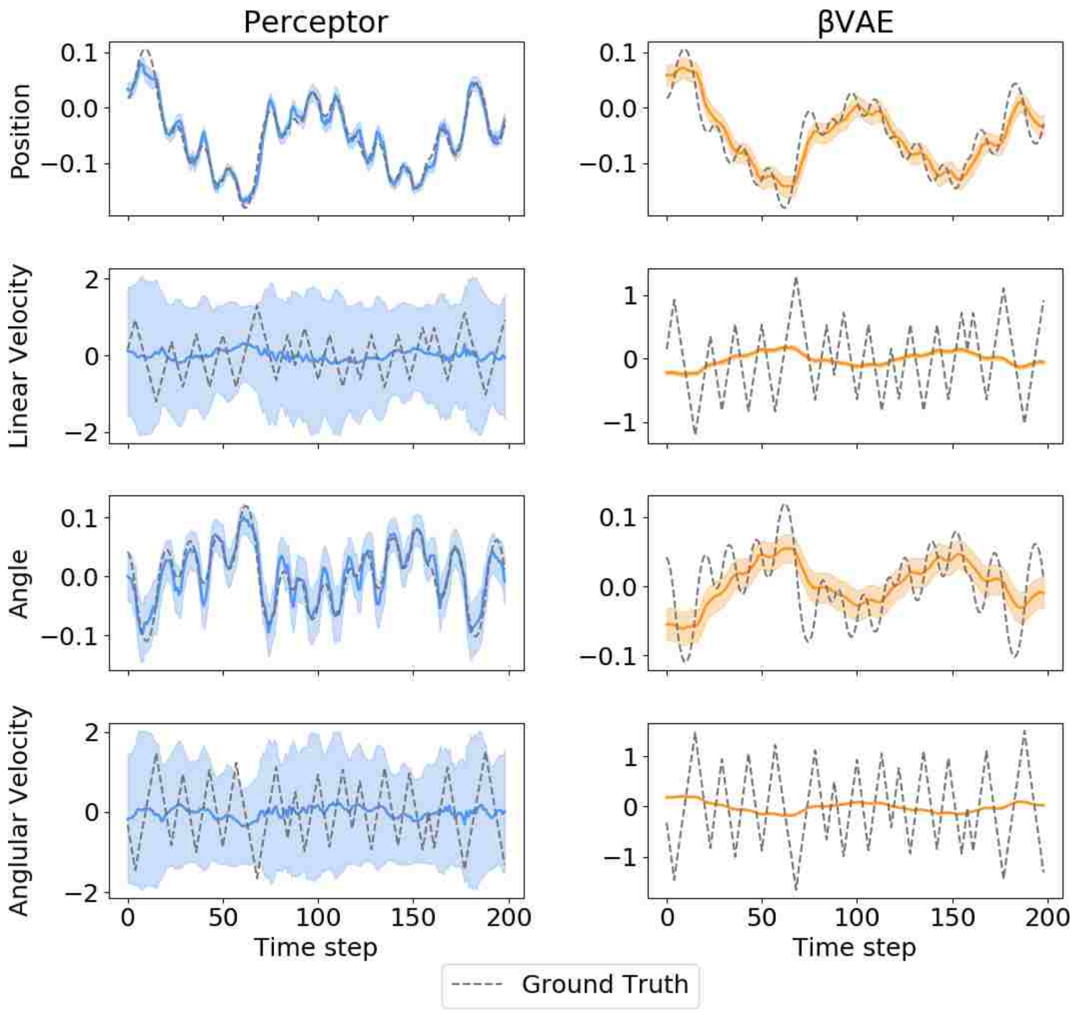


Figure 26: The latent space learnt by a perceptor (left) and βVAE (right). The thick coloured lines represent the predicted mean, while the shaded regions represent $\pm 1\sigma$ of the predicted variance.

true values and the associated variance is also quite large. Investigating the results, we found out that the LQR controller is able to balance the system even if the velocities are not taken into account. Given that the performance of the task is not sensitive to the velocity estimates, the perceptor was not able to ground the velocities from the symbolic state representation to the corresponding patterns in the observed images.

We compared the representations learnt by the perceptor to the ones learnt by a βVAE [90]. In order to do so, we generated a dataset of observations obtained by controlling the cart-pole with the perceptor for 100 episodes. We used the same architecture for the βVAE encoder as the one shown in Fig. 24 and replaced the convolutions with transposed convolutions for the decoder. Since the latent dimensions learnt by an autoencoder are non-identifiable, we performed linear regression between the learnt latent space and the ground truth values. As it can be seen from the results in Fig. 26 the representations learnt by the βVAE do capture some of the symbolic state structure, however they are considerably less precise than the ones learnt by the perceptor. On the one hand, the βVAE does not manage to extract the velocities either, but on the other, it collapses the distribution resulting in high certainty of the wrong estimates.

The loss function of the βVAE is defined as

$$\mathcal{L}_{VAE} = \sum_{i=1}^{N} \left\{ -\mathbb{E}_{q_{f_\theta}(z|x^{(i)})} \left[\log p_{g_\theta}(x^{(i)}|z) \right] + \beta D_{KL}(q_{f_\theta}(z|x^{(i)})||p(z)) \right\} \tag{52}$$

where D_{KL} denotes the Kullback–Leibler divergence. Assuming a standard normal distribution as the prior $p(z)$, Higgins et al. [90] propose to increase the weight β of the KL term in order to learn disentangled representations. The results shown in Fig. 26 were obtained with $\beta = 0.1$, otherwise the regularisation on the latent space forcing the latent dimensions to be independent was too strong. It has been recognised by the community that there are better ways to enforce disentanglement such as minimising total correlation [36, 107]. However, these methods also rely on the independence assumption which in our case does not hold. The position and angle of the cart-pole system are clearly entangled and so we need the program to regularise the latent space as it captures the underlying interactions.

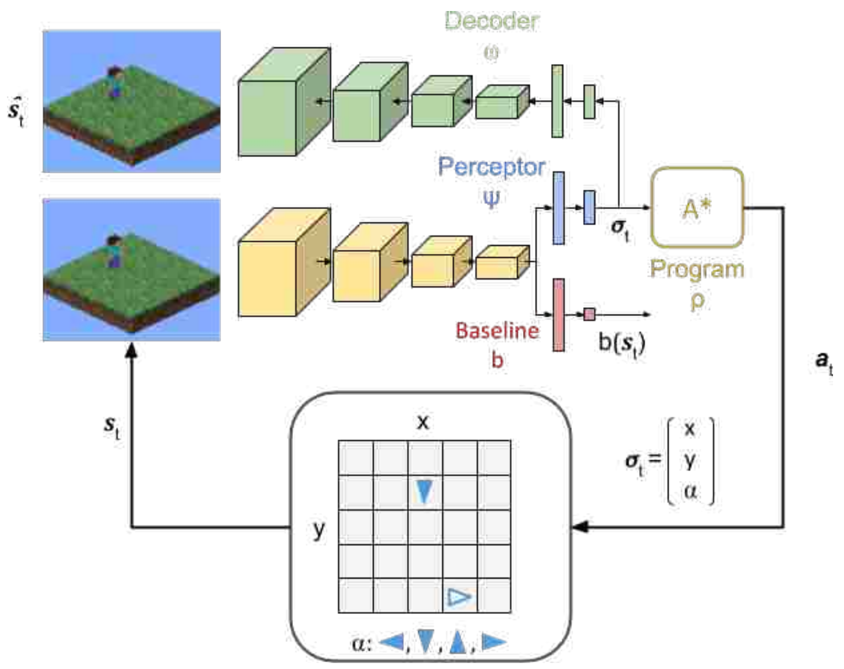


Figure 27: A diagram of the 'go to pose' experimental setup. The perceptor extracts the pose of the agent from images and feeds it into an A* planner in order to chose an action.

The overall results of this experiment clearly indicate that programs are not only sufficient for learning grounded symbolic representations, but are in fact necessary when task related abstractions are to be learnt.

4.5.2 *Minecraft: Go to Pose*

4.5.2.1 *Task Description*

We now consider the problem of navigating to a certain location by learning symbolic representations from images. In particular, we consider a 5×5 grid world where an agent is to navigate from its location to a randomly chosen goal pose. The agent receives +1 reward if it gets closer to the selected goal pose, +5 if it reaches the position and +5 if it turns to the right direction. To encourage optimal paths, the agent also receives -0.5 reward at every timestep. A symbolic representation of the task together with the 2.5D rendering of the environment is shown in Fig. 27. We express the state of the environment as

$$\boldsymbol{\sigma} = \begin{bmatrix} x & y & \alpha \end{bmatrix}^T \tag{53}$$

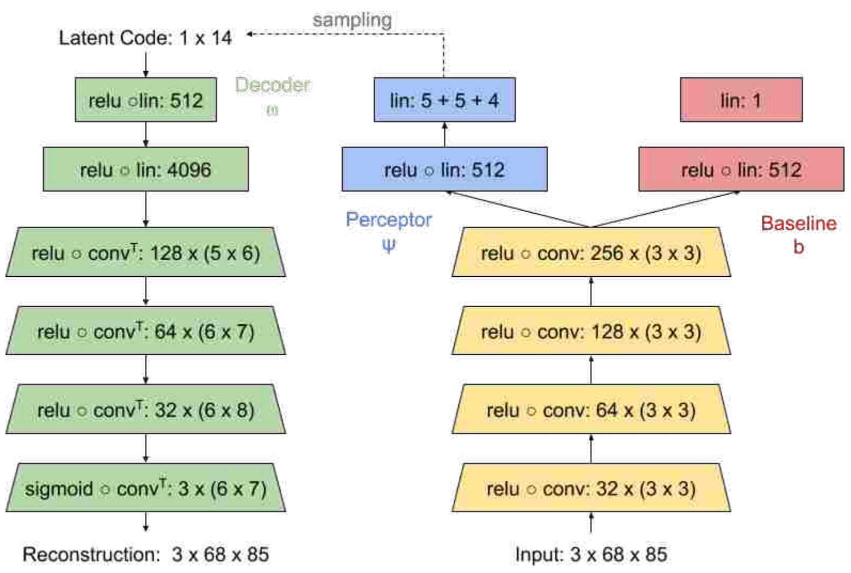

sampling
Latent Code: 1 x 14
relu ∘lin: 512
Decoder
ω
relu ∘ lin: 4096
relu ∘ conv^T: 128 x (5 x 6)
relu ∘ conv^T: 64 x (6 x 7)
relu ∘ conv^T: 32 x (6 x 8)
sigmoid ∘ conv^T: 3 x (6 x 7)
Reconstruction: 3 x 68 x 85
lin: 5 + 5 + 4
relu ∘ lin: 512
Perceptor
Ψ
lin: 1
relu ∘ lin: 512
Baseline
b
relu ∘ conv: 256 x (3 x 3)
relu ∘ conv: 128 x (3 x 3)
relu ∘ conv: 64 x (3 x 3)
relu ∘ conv: 32 x (3 x 3)
Input: 3 x 68 x 85

Figure 28: Architecture of the perceptor ψ_θ, the decoder ω_v and the baseline network b_ϕ. Convolutions are represented as #filters $\times$ filter size and all of them have a stride of 2. Transposed convolutions ($conv^T$) are represented in the same way. Linear layers are denoted by the number of output units.

where $x, y \in \{1, 2, 3, 4, 5\}$ are categorical variables representing the position of the agent in the world and $\alpha \in \{1, 2, 3, 4\}$ represents its orientation.

4.5.2.2 *A* Program*

Given that the pose of the agent $\boldsymbol{\sigma}$ and the goal pose G are known this problem can be easily solved using a general purpose planner. Therefore, in this experiment the program ρ implements a standard $A*$ planner. In comparison to the simple control law program we used in the cart-pole experiments, ρ in this case is a much more complex program as it contains several loops, multiple conditional statements as well as a priority queue. For the experiments in this section we directly plugged in the implementation provided by the `python-astar` package [2]. At every timestep, a path is found between the current $\boldsymbol{\sigma}_t$ produced by the perceptor and the goal G randomly chosen at the beginning of the episode such that the agent either moves to one of the 4 neighbouring squares or rotates in-place at $90°$, $180°$ or $270°$. The output action $\mathbf{a}_t = \rho(\boldsymbol{\sigma}_t)$ is set to the first movement in the found path.

2 https://pypi.org/project/astar/

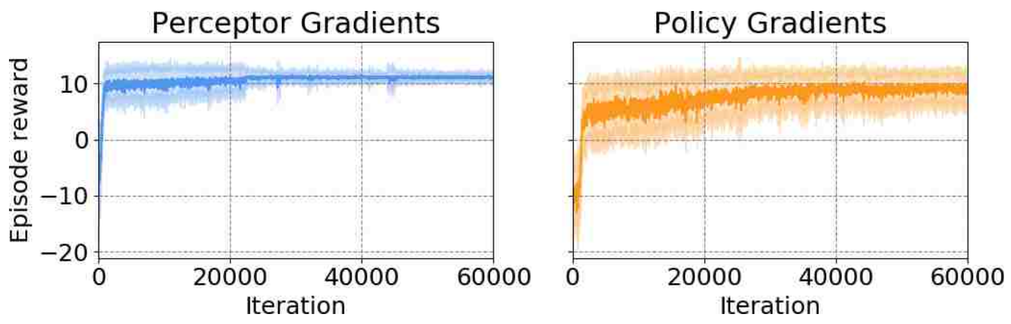


Figure 29: Learning performance at the 'go to pose' task of the perceptor gradients (left) compared to the policy gradients (right).

4.5.2.3 *Autoencoding Perceptor*

For this experiment we utilise an autoencoding perceptor, the architecture of which is shown in Fig. 28. The encoder outputs the parameters of 3 categorical distributions corresponding to x, y and α variables. These distributions are sampled to generate a latent code that is put through the decoder. We use the Gumbel-Softmax reparameterisation of the categorical distributions [100] such that gradients can flow from the decoder through the latent code to the encoder.

4.5.2.4 *Learning Performance*

In this experimental setup the perceptor is able to learn from raw image observations the symbolic representations required by the A* planner. The average reward obtained during training is shown in Fig. 29. Again, we compare the performance of the perceptor gradients to a standard policy gradients algorithm, where we have replaced the program with a single linear layer with softmax output. Additionally, we rendered an arrow in the observed images to represent the chosen goal such that the policy network has access to this information as well. In only 2000 iterations the perceptor obtains an average reward close to the optimal one of approximately 11.35, while it takes more than 30000 iterations for the policy gradients algorithm to approach an average reward of 10. Furthermore, given the highly structured nature of the environment and the task, the perceptor gradients agent eventually learns to solve the task with much greater reliability than the pure policy gradients one. This result clearly demonstrates the potential of neuro-symbolic approaches where neural

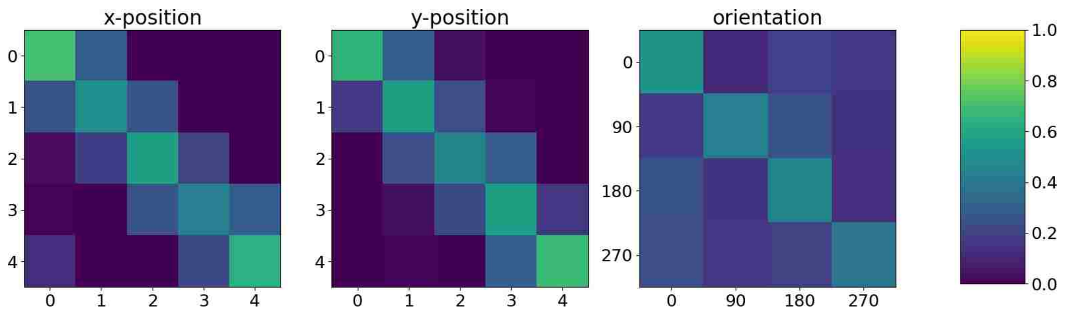


Figure 30: Confusion matrices between the values predicted by the perceptor (horizontally) and the true (vertically) values for each of the symbolic state components.

architectures are used to ground symbols to input data patterns while a task encoding program manipulating the grounded symbols chooses actions.

4.5.2.5 *Perceptor Latent Space*

In contrast to the cart-pole perceptor, in this case the latent space directly maps to the symbolic state description of the environment. We collected a dataset of 1000 episodes and compared the output of the perceptor to the ground truth values. The results are summarised in the confusion matrices in Fig. 30. The results indicate that the perceptor has learnt to correctly ground the symbolic state description to the observed data. Despite the fact that there are some errors, especially for the orientation, the agent still learnt to perform the task reliably. This is the case because individual errors at a certain timestep have a small impact on the eventual successful solution of the task due to the robust nature of the abstractions supported by the symbolic A* planner. For example, whether the agent is one or two steps to the left away from the goal pose the resulting action will be the same one of moving one step to the left. These two states, while very different in the input image, are essentially equivalent as they belong to the same partition of the problem state space.

4.5.2.6 *Generating Observations*

Since for this experiment we have trained an autoencoding perceptor we are also able to generate new observations (images of the environment) from a symbolic specification. In Fig. 31, we have chosen a sequence of symbolic state descriptions, forming a trajectory, that we have passed through the decoder in order to generate the

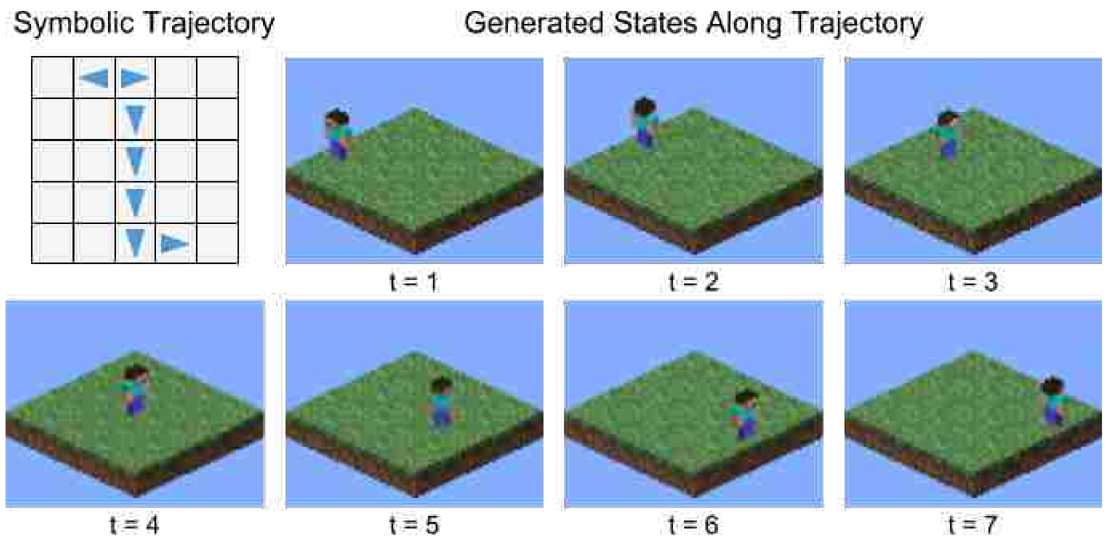


Figure 31: Sampled images of states following a symbolically defined trajectory.

corresponding images. Given the lack of noise in the rendering process we are able to reconstruct images from their symbolic descriptions almost perfectly. Orientation seems to be the hardest symbol to ground given the confusion matrices in Fig. 30 as well as the fact that the agent is not correctly oriented in the generated image corresponding to $t = 5$ in Fig. 31.

4.5.3 *Minecraft: Collect Wood*

4.5.3.1 *Task Description*

The last task we consider is navigating to and picking up an item, in particular a block of wood, from the environment. In addition to the motion actions, the agent can now also pick up an item if it is directly in front of it. The pick action succeeds with 50% chance thus introducing stochasticity in the task. As discussed by Levesque [131], this problem cannot be solved with a fixed length plan and requires a loop in order to guarantee successful completion of the task. The agent receives +5 reward whenever it picks up a block of wood. We expand the state to include the location of the wood block resulting in

$$\boldsymbol{\sigma} = \begin{bmatrix} x & y & \alpha & x_w & y_w \end{bmatrix}^T \tag{54}$$

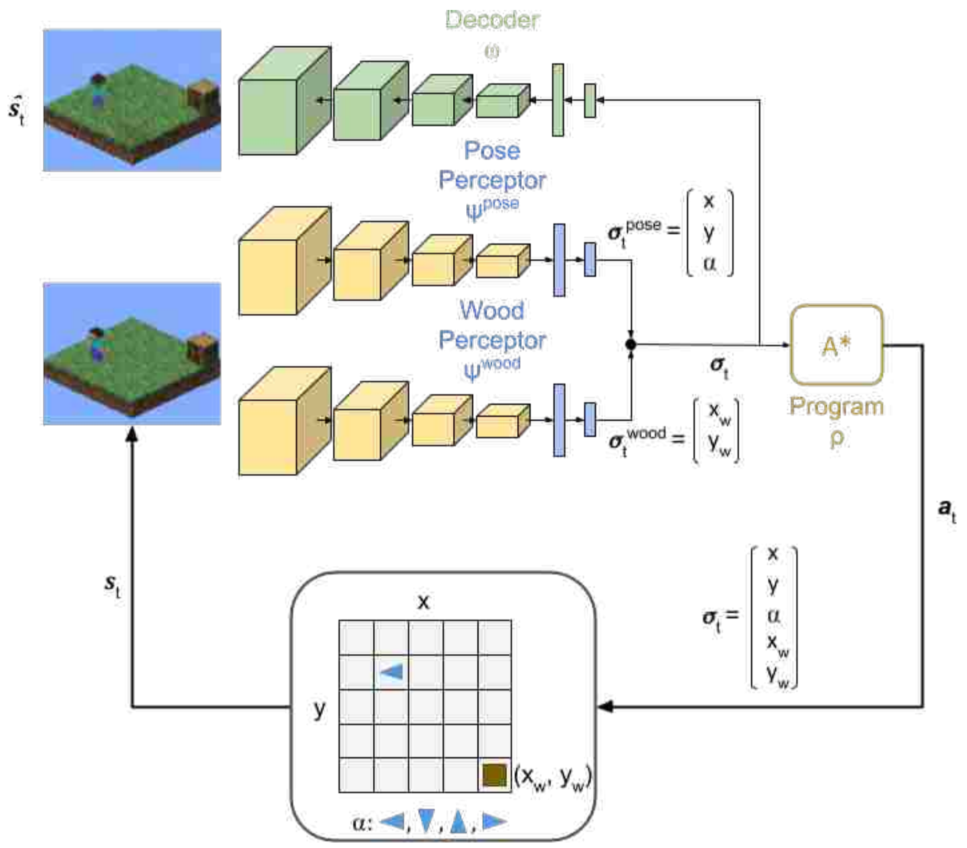


Figure 32: A diagram of the 'collect wood' experimental setup. One perceptor recognises the pose of the agent, while the other recognises the position of the wood block. The pose perceptor is pre-trained on the 'go to pose' task and is trained further in tandem with the wood perceptor. Both the A* planner and the decoder utilise the joint latent space of the pose and wood perceptors. The baseline networks are not illustrated in the figure for clarity purposes.

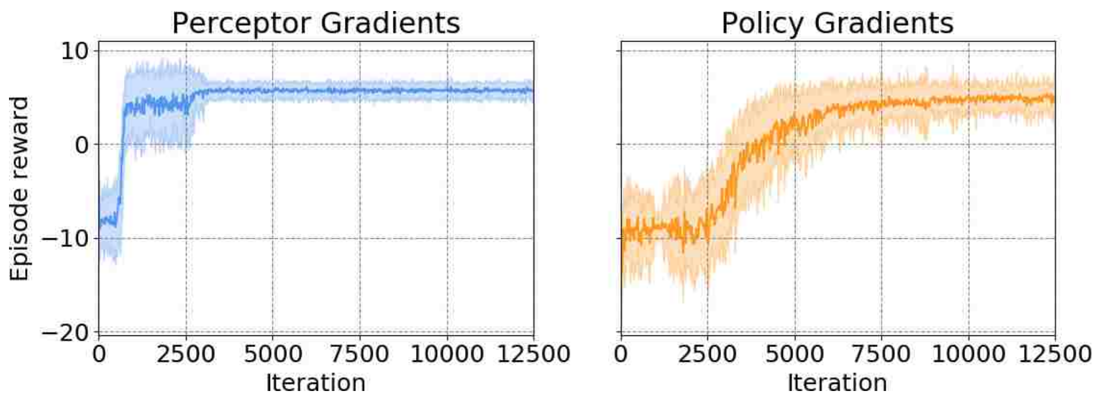


Figure 33: Learning performance at the 'collect wood' task of the perceptor gradients (left) compared to the policy gradients (right).

where $x_w, y_w \in \{1, 2, 3, 4, 5\}$ are categorical variables representing the location of the block of wood and x, y, and α are defined as before. The symbolic state representation as well as the rendered observations are shown in Fig. 32.

4.5.3.2 *Stacked Perceptors*

In the cart-pole experiment the controller balances the system around a fixed state, whereas in the navigation experiment the A* planner takes the agent to a randomly chosen, but known state. Learning to recognise both the pose of the agent and the position of the item is an ill posed problem since there is not a fixed reference frame - the same sequence of actions can be applied to the same relative configuration of the initial state and the goal, which can appear anywhere in the environment. Therefore, we reuse the pose perceptor from the previous experiment and combine it with another perceptor that is to learn to recognise the position of the wood block. Importantly, we train both of them jointly, but keep the learning rate for the pre-trained perceptor considerably lower than the one for the wood perceptor. This ensures that no catastrophic forgetting occurs during the initial stages of training. The entire experimental setup is shown in Fig. 32.

4.5.3.3 *Learning Performance*

As shown in Fig. 33 the agent is quickly able to solve the task with the stacked configuration of perceptors. It significantly outperforms the neural network policy as it achieves optimal performance in less than 3000 iterations. These results indicate that

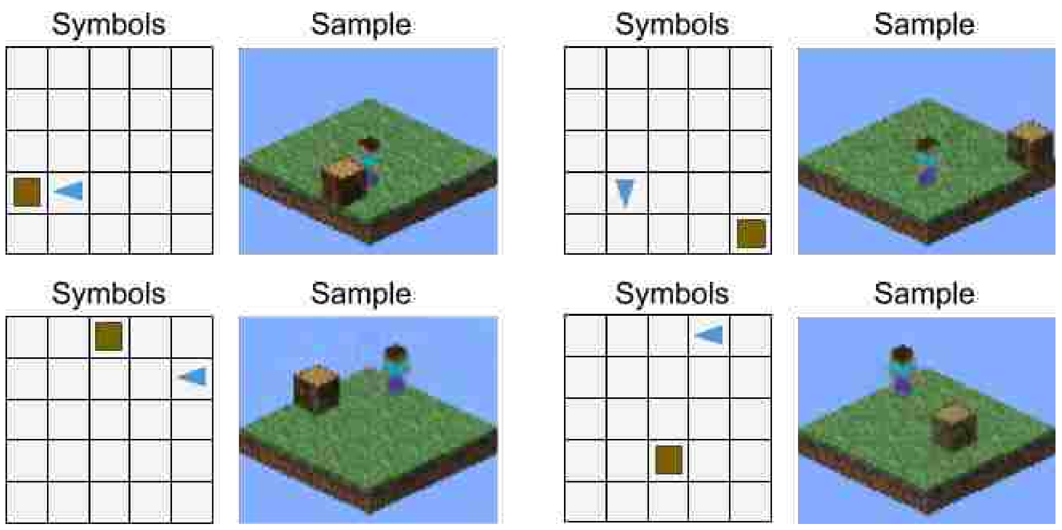


Figure 34: Sampled images from a symbolic specification over the joint latent space of the pose and wood perceptors.

lifelong learning can be achieved by maintaining a pool of perceptors responsible for various symbols, similarly to the neural libraries proposed by Gaunt et al. [65].

4.5.3.4 *Generating Observations*

Stacking perceptors preserves the symbolic nature of the latent space. In Fig. 34 we have shown a set of generated state samples from a symbolic specification over the joint latent space. The samples not only look realistic, but also take occlusions into account correctly (see top left sample). Overall the joint latent space samples have more artefacts and blurry regions compared to the ones in Fig. 31 which can potentially be tackled with more advanced autoencoding architectures.

4.5.4 *Ablation Experiments*

In order to study the contribution of the decoder to the performance of the agent in the Minecraft tasks we conducted a set of ablation experiments where we replaced the autoencoding perceptor with a feedforward one. Figure 35 and Figure 36 show the learning performance at the 'go-to pose' and 'collect-wood' tasks, respectively, with a feedforward perceptor. Overall, the results indicate that the main effect of the decoder is to decrease the variance of the obtained reward during training. The feedforward

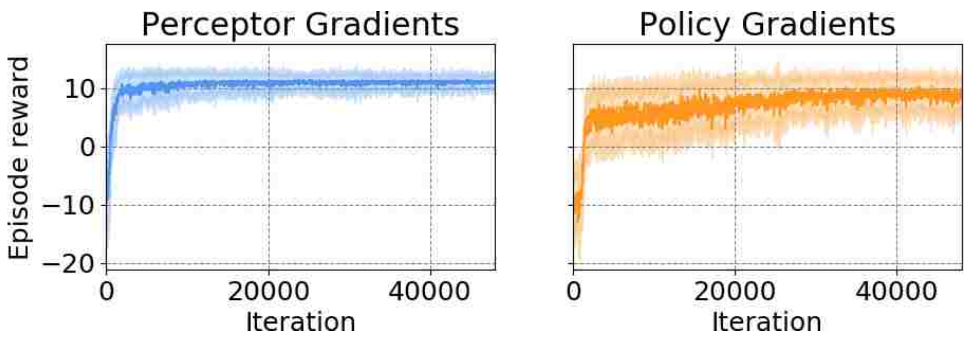


Figure 35: Learning performance with a feedforward perceptor at the 'go to pose' task of the perceptor gradients (left) compared to the policy gradients (right).

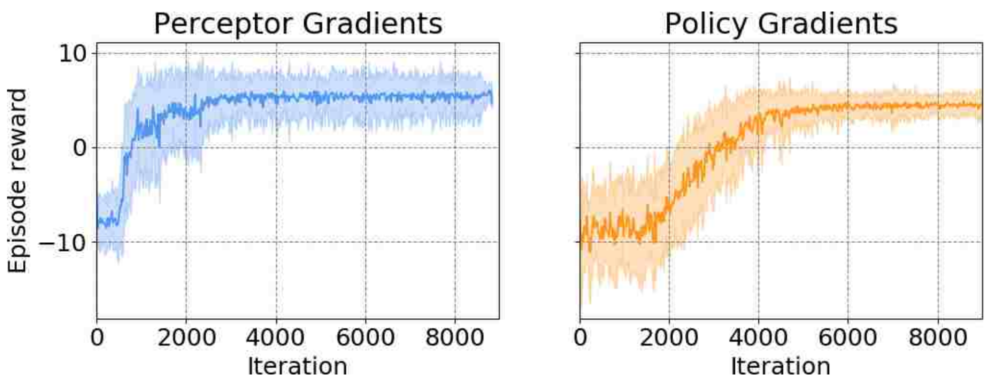


Figure 36: Learning performance with a feedforward perceptor at the 'collect wood' task of the perceptor gradients (left) compared to the policy gradients (right).

perceptor manages to ground the position of the agent slightly more accurately than the autoencoding perceptor, however the accuracy of the orientation has decreased. The reason for this is that orientation has little effect on the performance of the agent as it can move to any square around it, regardless of its heading. This is similar to the cart-pole task where the linear and angular velocities had little effect on the LQR performance.

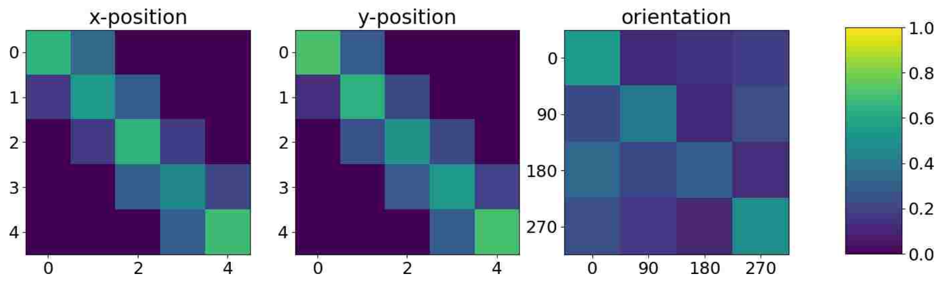


Figure 37: Confusion matrices between the values predicted by the feedforward perceptor (horizontally) and the true values (vertically) for each of the symbolic state components.

4.6 Discussion

The proposed perceptor gradients algorithm is a step towards bridging the gap between symbolic AI and neural networks in an attempt to combine the best of both approaches. We have successfully applied the algorithm to a small set of synthetic tasks, but there are limitations that need to be addressed, before it can be applied to realistic, much more challenging, problems.

Firstly, running the perceptor gradients algorithm on a real system, such as a real cart pole system observed by a camera means that the method should deal with noise and model inaccuracies. While the perceptor network can undoubtedly work with real images, writing an LQR program corresponding to the actual system, as opposed to its idealised model, will certainly be challenging. Perhaps a fruitful approach would be to introduce free parameters in the task encoding program and fit them to the observed data in an online fashion, essentially performing online system identification. Of course, leaving too many degrees of freedom in the task encoding program, combined with the initially random perceptor network could result into poor learning performance. Furthermore, RL is a particularly data inefficient method and naively deploying it on real systems is impractical. Imitation learning and sim-to-real transfer are certainly key for any successful application of the perceptor gradients algorithm to a real system.

Furthermore, the latent symbolic state is assumed to have a fixed size and to encode a flat collection of symbols. In practice, however, the latent space can vary in size as well as contain symbols built on top of other symbols. Representing a symbolic state of a variable length, while capturing hierarchical structure, can be achieved by utilising an RNN as a perceptor network. Importantly, this can result in extra symbols being produced and so any task encoding program working with an RNN perceptor should be robust to such errors. For example, if we consider a Minecraft task where multiple wooden blocks can exist, then any symbolic state of the world, and transitions, should be checked for consistency and possibly corrected if an error occurs.

Finally, the learning performance depends not only on the correctness of the program, but on its abstraction power as well. For example, the Minecraft agent is not able to recognise its position with perfect accuracy (see Fig. 30) because being 1 and

2 squares to the left from the goal are seen as equivalent states by the A* program, which will output go to right action in both cases. Therefore, there is an implicit trade off between the learning capabilities of the agent and the abstraction power of the task encoding program. One possible way to overcome this problem is by learning multiple tasks using a single perceptor and a task encoding program per task. In fact, the autoencoding perceptor can be viewed as a feedforward perceptor with one additional task of reconstructing the input image, where the task encoding program is simply passing the latent symbolic state through a deconvolutional network that is being learnt. However, choosing a suitable set of tasks that do not lead to conflicting learning updates is largely an open question.

4.7 Concluding Remarks

In this chapter we introduced the idea of decomposing a policy into a perceptor and a program resulting in a novel representation learning algorithm able to learn task related representations. We proposed the perceptor gradients algorithm and proved that it can be applied with any task encoding program that produces an action in a finite amount of time.

Our experimental results demonstrate that various task related abstractions can be learnt both in continuous and discrete domains. We showed that programs can be viewed as a powerful regulariser of the latent space that is able to encourage task related interactions between the latent dimensions. An interesting direction for research based on the proposed framework is to study closer the relationship between programs and manifolds in the symbolic latent space.

Importantly, we demonstrated that once the task related symbols are grounded to their visual appearance we can generate new images according to a symbolic specification. It is to our knowledge that no other representation learning method supports the generation of observation samples according to a semantically meaningful specification.

5

The π-Machine

In the previous two chapters of the thesis we studied the problem of learning symbolic representations when a task encoding program is available. In this chapter we study the complement problem of inducing task encoding programs when symbolic representations of the data are available. We combine stochastic gradient descent with A* search in order to efficiently induce programs from small number of observation traces. We demonstrate that the proposed method successfully learns programs describing physical systems, agent policies and human demonstrations.

5.1 Introduction

Learning to solve a task simply by observing a demonstration is one of the core problems in robotics [12]. While the observed state action transitions form a sequence that solves the particular instance of the task, it is rarely the case that the same sequence will be successful for other initial conditions of the world. The key challenge is to determine the underlying mechanisms that have generated the observed trace. Thus, learning from demonstration is a particular instance of the more general problem of learning transition system models. Other instances of the same problem range from system identification of dynamical systems [188] and inference of human choice behaviour [28, 68] to reverse engineering the behaviour of a device or computer program from observations and traces [212].

Programs are not only expressive enough to model a vast range of transition systems, but they also provide a structured and formal description of the underlying data generating mechanisms. With the increasing use of these learnt models in the inner loops of decision making systems, e. g., in robotics and human-machine interfaces, it has become necessary to ensure not only that these models are accurate predictors of behaviour, but also that their causal mechanisms are exposed to the system designer in a more interpretable manner. There is also the need to explain the model in terms of counterfactual reasoning [24], e. g., what would we expect the system to do if a certain variable were changed, or model checking [16] of longer term properties including safety and large deviations in performance.

In order to address the needs of learning structured transition system models that are amenable to inspection, we propose to learn high level functional programs representing abstract models that capture the invariant structure in the observed data. In addition to our findings so far that programs provide valuable inductive bias when learning structured representations from raw data, recent works have demonstrated the usefulness of programs in capturing human-like concepts [123]. Moreover, programmatic model descriptions boost generalisation and enable knowledge transfer [216]. Finally, programs are also significantly more amenable to model checking and human interpretability compared to statistical models fit to the data.

In this chapter, we introduce the π-machine (program-induction machine) – an architecture which is able to induce LISP-like programs from observed transition system data traces in order to explain various phenomena. Inspired by differentiable neural computers [70, 71], the π-machine, as shown in Fig. 38, is composed of a memory unit and a controller capable of learning programs from data by exploiting the scalability of stochastic gradient descent. The final program obtained after training, however, is not an opaque object encoded in the weights of a controller neural network, but a LISP-like program which provides a rigorous and interpretable description of the observed phenomenon. A key feature of our approach is that we allow the user to specify the properties they are interested in understanding, as well as the context in which the data is to be explained, by providing a set of symbols and predicates of interest. By exploiting the equivalence between computational graphs and functional programs we describe a hybrid optimisation procedure based on backpropagation, gradient descent, and A* search which is used to induce programs from data traces.

We evaluate the performance of the π-machine on three different problems. Firstly, we apply it to data from physics experiments and show that it is able to induce programs which represent fundamental laws of physics. The learning procedure has access to relevant variables, but it does not have any other prior knowledge regarding physical laws which it has discovered in the same sense as in [188] although far more computationally tractably. We then study the use of the proposed procedure in explaining control policies learnt by a DQN agent. Starting from behaviour traces of a reinforcement learning agent that has learnt to play the game of Pong, we demonstrate how the π-machine learns a functional program to describe that policy. Finally, we consider the domain of learning by demonstration in human-robot interaction, where the π-machine successfully induces task encoding programs that capture the structure of the human demonstration. In this domain, the learnt program plays a key role in enabling the grounding of abstract knowledge (e.g., in natural language commands) in the embodied sensory signals that robots actually work with.

5.2 Related work

5.2.1 *Learning Structured Task Representations*

Assuming that a set of task related symbols can be grounded and observed in the environment, structured task representations have been shown to improve the robustness and generalisation capabilities of the agent. Kansky et al. [103] propose to learn networks of first order logic templates (schemas) applicable to the observed entities and perform planning in order to perform the task. They show that Schema networks outperform various neural architectures especially when tested on novel tasks within the same domain. The proposed schemas are limited, however, to first order logic relations between the observed symbols. A more general approach would be to induce task encoding programs. Verma et al. [216] propose to learn programs replicating the behaviour of a pre-trained DRL agent. They express the programs into a domain specific language which can represent multiple Proportional-Integral-Derivative controllers guarded by different conditions. We follow a similar idea in order to explain the behaviour of a DQN agent, however, we induce programs from single observation traces, rather than by querying the underlying policy that generated the traces.

5.2.2 *Program Learning and Synthesis*

Program learning and synthesis has a long history, with the long-standing challenge being the high complexity deriving from the immense search space. Following classic and pioneering work such as by Shapiro [193] who used inductive inference in a logic programming setting, others have developed techniques based on a variety of methods ranging from SAT solvers [196] to genetic algorithms [188], which tend to scale poorly hence often become restricted to a narrow class of programs. Recently, deep neural networks have been augmented with a memory unit resulting in models similar to the original von Neumann architecture. These models can induce programs through stochastic gradient descent by optimising performance on input/output examples [70, 71, 73] or synthetic execution traces [33, 136, 177]. Programs induced with such neural architectures are encoded in the parameters of the controller network and are, in general, not easily interpretable (particularly from the point of view of being able to ask counterfactual questions or performing model checking). Interestingly, paradigms from functional programming such as pure functions and immutable data structures have been shown to improve performance of neural interpreters [116]. Another approach is to perform neural program synthesis and directly generate the source code of the output program which yields consistent high level programs. Usually, these types of approaches require large amounts of labelled data - either program input/output examples [17, 44] or input paired with the desired output program code [232].

Determining how many input/output examples or execution traces are required in order to generalise well is still an open research problem. In this chapter, however, we focus our attention more on the explanatory power afforded by programs rather than on the broader problems of generalisation in the space of programs. While these characteristics are of course related, we take a view similar to that of Ribeiro, Singh and Guestrin [180], arguing that it is possible to build from locally valid program fragments which provide useful insight into the black-box processes generating the data. Program synthesis methods that combine symbolic structures with neural architectures have been shown to be extremely efficient in learning from input/output pairs [31, 160].

In this line of work, Andreas et al. [11] introduced Neural Module Networkss (NMNs) for solving visual question answering tasks. An NMN is built up from a set of interconnected modules (simple differentiable programs) which form a computational graph that can be optimised with gradient descent techniques and whose structure is determined by syntactic parsing of natural language queries. NMNs not only achieve state of the art results on various visual question answering tasks, but also provide a programmatic description of the reasoning process executed by the network in order to answer the query. There are multiple extensions to NMN attempting to replace the syntactic parser with a flexible end-to-end architecture search, but they usually start with behavioural cloning of the parser based NMN [10, 93, 94, 102]. In comparison, we do not assume the presence of natural language description of the underlying data generating process, but rather work only with observed task related symbols and search the space over possible program structures with classic planning methods. We introduce a hybrid optimisation procedure that combines gradient descent and A* search which is at the core of the π-machine. Additionally, NMNs provide only partial explanation as individual modules are black box neural networks themselves, whereas we induce high level programs with clearly defined functions. We demonstrate that the π-machine is able to learn informative and interpretable high-level LISP-like programs, even just from a single observation trace.

5.2.3 *Explainability and Interpretability*

The immense success of deep neural network based learning systems and their rapid adoption in numerous real world application domains has renewed interest in the interpretability and explainability of learnt models [1]. There is recognition that Bayesian rule lists [130, 228], decision trees and probabilistic graphical models are interpretable to the extent that they impose strong structural constraints on models of the observed data and allow for various types of queries, including introspective and counterfactual ones. In contrast, deep learning models usually are trained 'per query' and have numerous parameters that could be hard to interpret. Zeiler and Fergus [237] introduced deconvolutional networks in order to visualise the layers of convolutional

1 See, for instance, the end user concerns that motivate the DARPA Explainable AI Programme: http://www.darpa.mil/program/explainable-artificial-intelligence

networks and provide a more intuitive understanding of why they perform well. Similar approaches can be seen in [23, 108], in the context of autonomous driving. Zahavy, Ben-Zrihem and Mannor [235] describe Semi-Aggregated Markov Decision Process in order to analyse and understand the behaviour of a DQN based agent. Methods for textual rationalisation of the predictions made by deep models have also been proposed [81, 87, 129]. While all of these works provide useful direction, we need many more methods, especially generic ones, that need not be hand-crafted to explain specific aspects of individual models. In this sense, we follow the model-agnostic explanation approach of Ribeiro, Singh and Guestrin [180], who provide "textual or visual artefacts" explaining the prediction of any classifier by treating it as a black-box. Similarly to the way in which [180] utilise local classifiers composed together to explain a more complex model, we present an approach to incrementally constructing functional programs that explain a complex transition system from more localised predicates of interest to the user.

The π-machine treats the process which has generated the observed data as a black-box and attempts to induce a LISP-like program which can be interpreted and used to explain the data. We show that the proposed method can be applied both to introspection of machine learning models and to the broader context of autonomous agents where the induced programs encode the task of interest.

5.3 Problem Definition

Consider the labelled transition system $\Omega(\mathcal{S}, \mathcal{A}, \delta)$ where $\mathcal{S}$ is a non-empty set of states, $\mathcal{A}$ is a non-empty set of actions, each parametrised by $\theta \in \mathbb{R}^D$, and $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function. We define an observation trace $\mathcal{T}$ as a sequence of observed state-action pairs $(s_t, a_t(\theta_t)) \in \mathcal{S} \times \mathcal{A}$ generated by the recursive relationship $s_{t+1} = \delta\,(s_t, a_t(\,\theta_t))$ for $1 \leq t \leq T$. We are interested in inducing a LISP-like functional program ρ which when executed by an abstract machine is mapped to an execution trace $\mathcal{T}_\rho$ such that $\mathcal{T}_\rho$ and $\mathcal{T}$ are equivalent according to an input specification.

We represent the abstract machine as another labelled transition system $\Pi(\mathcal{M}, \mathcal{I}, \varepsilon)$ where $\mathcal{M}$ is the set of possible memory state configurations, $\mathcal{I}$ is the set of supported

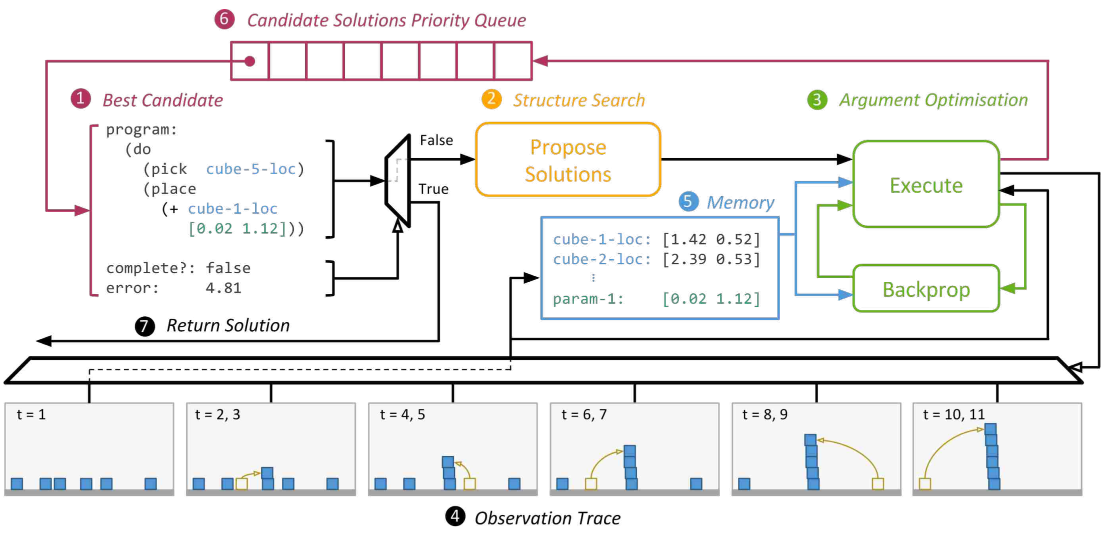


Figure 38: Overall architecture of the π-machine. The current best candidate solution (1) is used to propose new, structurally more complex candidates (2). Each one of the new candidate programs ρ is optimised (3) through gradient descent by comparing its execution trace to the observation trace (4). The observation trace in this case is a demonstration of a tower building task. During execution, the program has access to memory (5) which stores both state variables and induced parameters. All new candidate programs are scored based on their performance and complexity and are inserted in the candidate solutions priority queue (6). Once the execution trace of a candidate matches the observation trace the final solution is returned (7).

instructions and $\varepsilon : \mathcal{M} \times \mathcal{I} \rightarrow \mathcal{M}$ specifies the effect of each instruction. We consider two types of instructions – primitive actions which emulate the execution of $a \in \mathcal{A}$ or arithmetic functions $f \in \mathcal{F}$ such that $\mathcal{I} = \mathcal{A} \cup \mathcal{F}$. Furthermore, a set of observed state variables $\mathcal{M}_v \subseteq \mathcal{S}$, which vary over time, are stored in memory together with a set of induced free parameters $\mathcal{M}_p$. The variables in $\mathcal{M}_v$ correspond to the available symbols extracted from the observed data and form a context which the program is built on. A custom detector $\mathcal{D}_v$, operating on the raw data stream, could be provided for each variable, thus enabling the user to make queries with respect to different contexts and property specifications.

The execution of any arbitrarily structured program containing primitive actions results in a sequence of actions. Therefore, we represent a program ρ as a function which maps a set of input variables $x_v \subset \mathcal{M}_v$ and a set of free parameters $x_p \subset \mathcal{M}_p$ to a finite sequence of actions $\hat{a}_1(\hat{\theta_1}), \ldots \hat{a}_{T'}(\hat{\theta_{T'}})$. We are interested in inducing a program which minimises the total error between the executed and the observed actions:

$$L(\rho) = \sum_{t=1}^{\min(T,T')} \sigma_{act}(\hat{a}_t, \hat{\theta}_t, a_t, \theta_t) + \sigma_{len}(T, T') \tag{55}$$

The function σ_{act} measures the similarities between two actions, while σ_{len} compares the lengths of the generated and observed action traces. By providing the error functions σ_{act} and σ_{len} one can target different aspects of the observation trace to be explained as they specify when two action traces are considered similar.

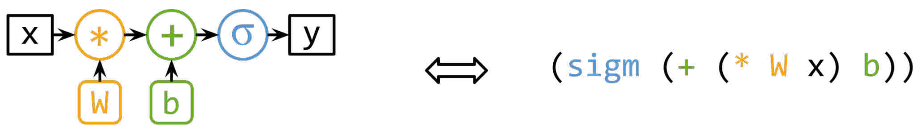
5.4 Method

The proposed program induction procedure is based on two major steps. Firstly, we explain how a given functional program can be optimised such that the loss $L(\rho)$ is minimised. Secondly, we explain how the space of possible programs can be searched efficiently by utilising gradient information. An architectural overview of the π-machine is provided in Fig. 38.

5.4.1 *Program Optimisation*

5.4.1.1 *Functional Programs as Computational Graphs*

Neural networks are naturally expressed as computational graphs which are the most fundamental abstraction in computational deep learning frameworks [1, 21, 206]. Optimisation within a computational graph is usually performed by pushing the input through the entire graph in order to calculate the output (forward pass) and then backpropagating the error signal to update each parameter (backward pass). A key observation for the development of the π-machine is that computational graphs and functional programs are equivalent as both describe arbitrary compositions of pure functions applied to input data. For example, Fig. 39 shows how a logistic regression classifier can be represented both as a computational graph and as a functional program. Therefore, similarly to a computational graph, a functional program can also be optimised by executing the program (forward pass), measuring the error signal and then performing backpropagation to update the program (backward pass).



```
(sigm (+ (* W x) b))
```

Figure 39: A logistic regression classifier expressed as a computational graph (left) and a functional program (right).

5.4.1.2 *Forward Pass*

When a program is executed it is interpreted to a sequence of instructions $i_1, \dots, i_n \in \mathcal{I}$ which are executed by recursively calling $\varepsilon(\dots\varepsilon(\varepsilon(\mathcal{M}_1, i_1), i_2)\dots, i_n)$. $\mathcal{M}_1$ is the initial memory state containing the observed variables from s_1 and any induced parameters. The π-machine keeps a time counter t which is initialised to 1 and is automatically incremented whenever a primitive action instruction is executed.

If the instruction i_k is a primitive action, $i_k \in \mathcal{A}$, then the π-machine automatically sets $\hat{a}_t = i_k$ and invokes the error function $\sigma_{act}(\hat{a}_t, \hat{\theta}_t, a_t, \theta_t)$, where $\hat{\theta}_t$ has been calculated by previous instructions. If the error is above a certain threshold e_{max} the program execution is terminated and the backward pass is initiated. Otherwise, the time counter is incremented and the values of the variables in $\mathcal{M}_v$ are automatically updated to the new observed state. The π-machine, essentially, simulates the execution of each action reflecting any changes it has caused in the observed state.

Alternatively, if the currently executed instruction i_k is a function, $i_k \in \mathcal{F}$, then the resulting value is calculated and i_k, together with its arguments, is added to a detailed call trace χ maintained by the π-machine. Importantly, each function argument is either (i) the output of another function $g \in \mathcal{F}$, (ii) a parameter $p \in x_p$ or (iii) a variable $v \in x_v$ read from memory at time t. All this information is kept in χ which eventually contains the computational tree for the entire program.

5.4.1.3 *Backward Pass*

The gradients of the loss function $L(\rho)$ with respect to the program inputs x_v and x_p are required to perform a gradient descent step. Crucially, programs executed by the π-machine are automatically differentiated. The π-machine performs reverse-mode automatic differentiation, similarly to Autograd [138], by traversing the call trace χ, and post-multiplying Jacobian matrices. We assume that the Jacobian matrix with

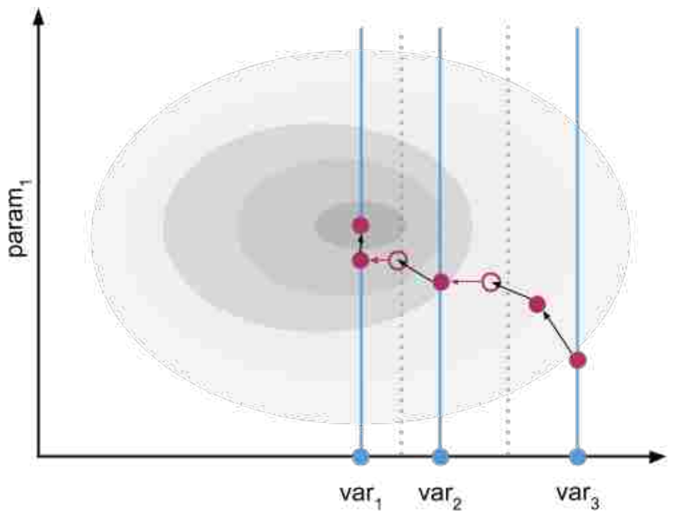


Figure 40: An example traversal of the parameter space containing both both continuous parameters (y-axis) and symbolic variables (x-axis). The dashed lines represent the partitioning induced by the corresponding KD-tree. Whenever the current solution enters a new partition it is immediately moved to the corresponding variable value. These switching points are denoted with outlined circles and red arrows.

respect to every input argument of any function $f \in \mathcal{F}$ or any specified error function σ_{act} is known a priori. Let $f \in \mathcal{F}$ be a function whose output needs to be differentiated with respect to the input arguments. There are three types of derivatives, which need to be considered in order to traverse backwards the entire tree of computations:

1. Let $g \in \mathcal{F}$, then $\frac{\partial f}{\partial g}$ is the Jacobian matrix of f with respect to the output of g and can be directly calculated.
2. Let $p \in x_p$, then the gradient $\frac{\partial f}{\partial p}$ is calculated by multiplying the corresponding Jacobian matrix of f with the value of p.
3. Let $v \in x_v$, then the gradient $\frac{\partial f}{\partial v}\Big|_{t=t_r}$ is calculated by multiplying the corresponding Jacobian of f with the value of the variable at the time it was read from memory t_r.

5.4.2 *Gradient Descent Step*

Once the gradient $\nabla_p L(\rho)$ of the loss function with respect to each input parameter $p \in x_p$ is calculated we utilise AdaGrad [47] to update the values of all parameters after each program execution. The gradient $\nabla_v L(\rho)$ with respect to each input variable $v \in x_v$ is also available. However, a variable cannot be simply updated in the direction of the gradient as it represents a symbol, not just a value. Variables can only take values from memory which are automatically updated according to the observation trace during execution. Nevertheless, the gradient provides important information about the direction of change which we utilise to find variables that minimise the loss. Whenever the memory state is automatically updated, a KD-tree is built for each type of variable stored in memory. We assume that the variables in memory are real vectors with different length. So, we represent the KD-tree which stores all D-dimensional variables in memory at time t as $\mathcal{K}_t^D$. If a d-dimensional variable v is to be optimised it is replaced with a temporary parameter p_{temp} initialised with v_t which is the value of v read from memory at the respective time step t. The temporary parameter p_{temp} is updated together with the other parameters in x_p using AdaGrad [47]. After each descent step, the nearest neighbour of the updated value p'_{temp} is determined by querying the KD-tree with $\mathcal{K}_t^d(p'_{temp})$. If the result of the query is a different d-dimensional variable u then the temporary parameter is immediately set to $p_{temp} = u_t$. As this often shifts the solution to a new region of the error space the gradient history for all parameters $p \in x_p$ is reset. Eventually, when a solution is to be returned, the temporary parameters are substituted with their closest variables according to the respective $\mathcal{K}_t^D$. A diagram illustrating the resulting parameter space traversal is shown in Fig. 40. Importantly, even though we are using AdaGrad, the augmentation which we described is optimiser agnostic and so can be applied to any gradient descent optimiser.

The forward and backward passes are repeated until the error is below the maximum error threshold e_{max} or a maximum number of iterations is reached. After that the optimised program ρ^* is scored according to its error and complexity, and pushed to a priority queue holding potential solutions.

5.4.3 *Structure Search*

We represent the space of possible program structures as a graph $G = (T^{AST}, E)$ where each node $T_i \in T^{AST}$ is a valid program abstract syntax tree (AST). There is an edge from T_i to T_j if and only if T_j can be obtained by replacing exactly one of the leaves in T_i with a subtree T_s of depth 1. The program induction procedure always starts with an empty program. We frame the search over program structure as a path finding problem and solve it through the use of an A* planner.

5.4.3.1 *Score Function*

The total cost function we use is

$$f_{total}(\rho) = C(\rho) + L(\rho) \tag{56}$$

where $L(\rho)$ is the loss function defined in equation (55) and $C(\rho)$ is a function which measures the complexity of the program ρ. $C(\rho)$ can be viewed as the cost to reach ρ and $L(\rho)$ as the distance to the desired goal. The complexity function $C(\rho)$ is the weighted sum of (i) maximum depth of the program AST; (ii) the number of free parameters; (iii) the number of variables used by the program; the weights of which we set to $w_C = [10, 5, 1]$. These choices have the effect that concise programs which maximally exploit the underlying dependencies between the symbols extracted from the observed data trace are preferred.

5.4.3.2 *Neighbours Expansion*

When the current best candidate solution is popped from the priority queue, we check if it matches the observation trace according to the input specification. If so, the candidate can be returned as the final solution, otherwise it is used as a seed to propose new candidate solutions. Typically in A* search, all neighbouring nodes are expanded and pushed to the priority queue, which is not feasible in our case due to the size of the search space. Therefore, we utilise the available gradients in order to perform a guided proposal selection. Each leaf in the abstract syntax tree T_ρ of a seed candidate solution ρ corresponds to a parameter or a variable. According to the definition of G we need to select exactly 1 leaf to be replaced with a subtree T_s of depth 1. We select leaf $l \in T_\rho$ according to:

$$l = \underset{x \in x_p \cup x_v}{\arg\max} \|\nabla_x L(\rho)\|_2 \tag{57}$$

After that, all possible replacement subtrees are constructed. An AST subtree T_s of depth 1 represents a function call. We prune the number of possible functions in $\mathcal{F}$ by ensuring type consistency. Each leaf of T_s can be a parameter or a variable. So, all possible combinations are considered. New variable leaves are initialised to a random variable with suitable type from memory, while new parameter leaves are sampled from the multivariate normal distribution $\mathcal{N}(0, 0.1)$. As a result, if n_f functions are type compatible with l and each function takes n_a arguments at most, then there are $2^{n_a} \cdot n_f$ replacement subtrees, resulting in that many new candidates. It is common to assume fixed upper limit of n_a such that the number of new candidates is contained in a reasonable range. All newly proposed candidates are optimised in parallel, scored by f_{total} and pushed to the priority queue.

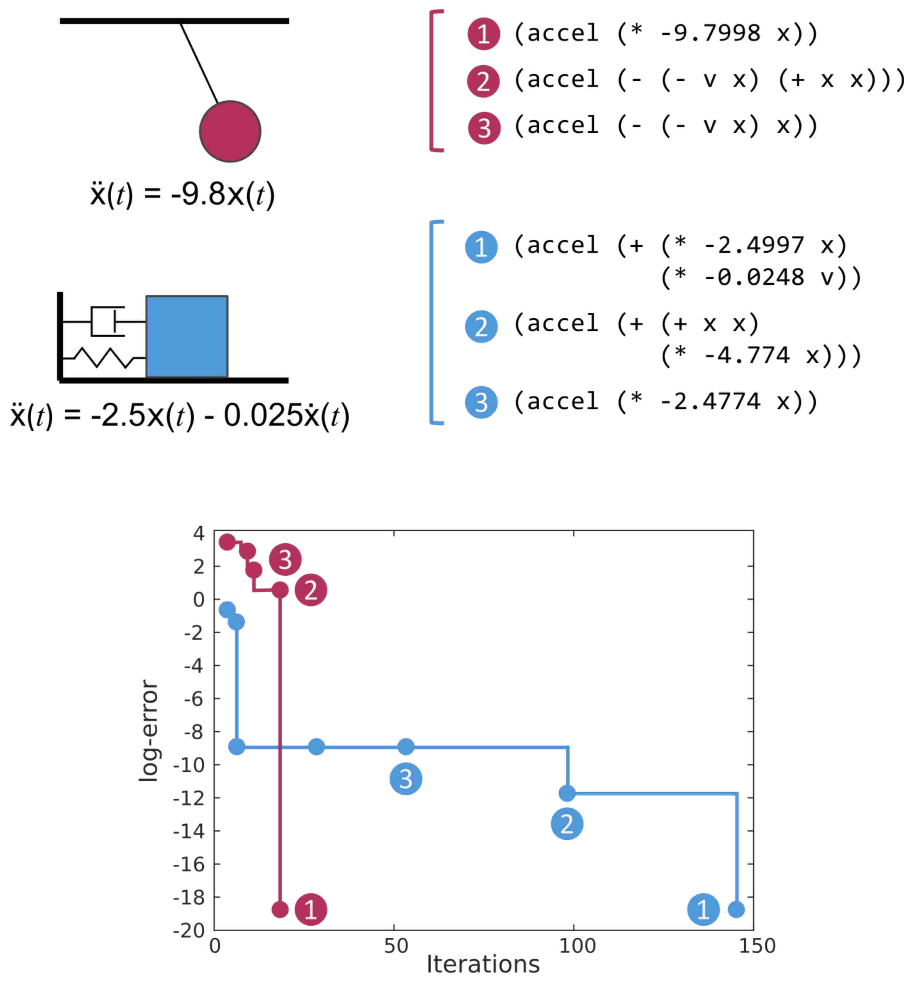
5.5 Experimental Results

In order to evaluate the effectiveness of the π-machine, we apply it to three different scenarios.

Firstly, we consider the problem of system identification of a physical system and show that the π-machine successfully induces basic physical laws from observational data more efficiently than state of the art methods.

Secondly, we demonstrate how a program can be induced in order to explain the behaviour of a DQN agent. This experiment is based on our view that the core deep neural network based policy learner and the explaining program play complementary roles. As is well known by now, there are numerous advantages to performing end-to-end policy learning, such as DQN-learning from raw video. However, there is also the need to explain the behaviour of the learnt policy with respect to user-defined symbols of interest, hence a program defined in terms of user-defined detectors D_v.

Lastly, we apply the π-machine to a learning-by-demonstration task, where a demonstrated behaviour of physical object manipulation is explained through a functional task encoding program that captures the abstract structure of that planning task.



```
1 (accel (* -9.7998 x))
2 (accel (- (- v x) (+ x x)))
3 (accel (- (- v x) x))
```

```
1 (accel (+ (* -2.4997 x)
            (* -0.0248 v))
2 (accel (+ (+ x x)
            (* -4.774 x)))
3 (accel (* -2.4774 x))
```

Figure 41: The π-machine applied to explaining the behaviour of a pendulum (red) and a linear oscillator(blue). The 3 best programs obtained for each system are shown and annotated on the log-error plot (bottom).

All experiments are run on an Intel Core i7-4790 processor with 32GB RAM. The π-machine is implemented in Clojure, which is a LISP dialect supporting powerful data structures and homoiconic syntax (see Appendix A).

Given the nature of the experimental tasks, we use the following list of supported functions, $\mathcal{F}$, for all of our experiments: vector addition, subtraction and scaling.

5.5.1 *Physical Systems*

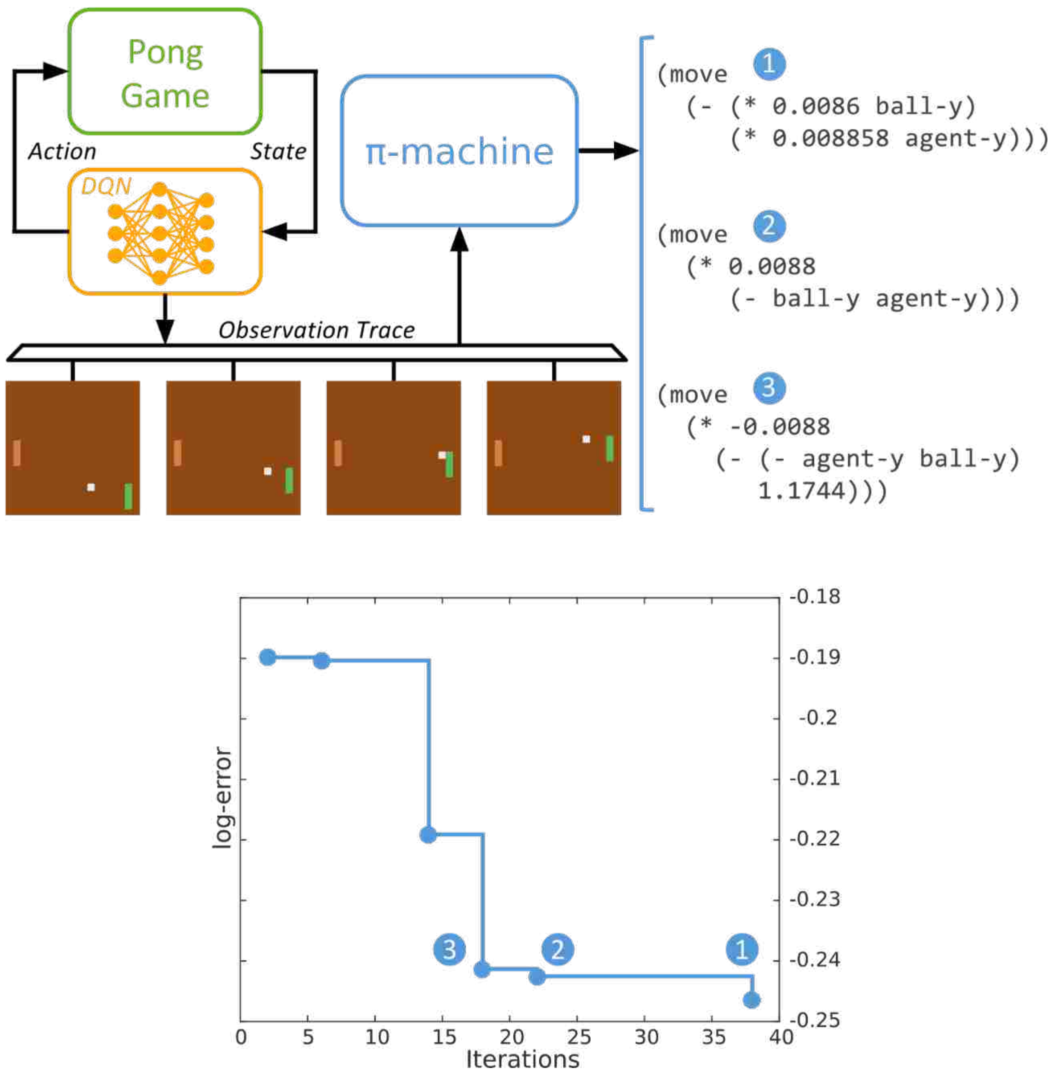
The transition dynamics of a second order dynamical system is written as $\ddot{\mathbf{x}}(t) = k_1\mathbf{x}(t) + k_2\dot{\mathbf{x}}(t)$, where $\mathbf{x}(t)$ is the state of the system at time t and k_1, k_2 are system coefficients. We have recreated an experiment described in Schmidt and Lipson

[188], where the authors show the learning of physical laws associated with classical mechanical systems including the simple pendulum and linear oscillator. A diagram of these two systems is shown in Fig. 41 (top left). We set $\mathcal{A} = \{accel(\theta)\}$ where $\theta \in \mathbb{R}$ for both experiments. The observation trace for each system is generated by simulating the dynamics for 1s at 100Hz. We specify the action error function as $\sigma_{act} = \|\hat{\theta} - \theta\|_2$ and set $\sigma_{len} = 0$. In the pendulum experiment, $\mathbf{x} \in \mathbb{R}$ is the angular position of the pendulum, while $\mathbf{v} = \dot{\mathbf{x}} \in \mathbb{R}$ is the angular velocity. In the linear oscillator experiment, $\mathbf{x}$ and $\mathbf{v}$ are the linear position and velocity, respectively.

The three best solutions found by the π-machine for each system are shown in Fig. 41 (top right). The best solution for each system correctly represents the underlying laws of motion. The program describing the behaviour of the pendulum was induced in 18 iterations, while the linear oscillator program needed 146 iterations. The total number of possible programs with AST depth of 2, given the described experimental setup, is approximately 1.7×10^4. The average duration of an entire iteration (propose new programs, optimise and evaluate) was $0.6s$. Schmidt and Lipson [188] achieve similar execution times, but distributed over 8 quad core computers (32 cores in total). The experimental results demonstrate that the π-machine can efficiently induce programs representing fundamental laws of physics.

5.5.2 *Deep Q-value Network*

We consider explaining the behaviour of a DQN agent trained to play the ATARI Pong game. We are interested in the question: how does the network control the position of the paddle in order to hit the ball when it is approaching. A diagram of the experimental setup is shown in Fig. 42 (top left). The behaviour of the DQN is observed during a *single* game. Since the environment is deterministic, the only component of the state transition function generating the observation trace is the policy $\pi(a|s)$ that the DQN agent has learnt. We would like to explain the behaviour of the DQN in terms of the position of the opponent, the ball and the DQN agent (not just in terms of RAM or pixel values for instance). Therefore, the observation trace contains those positions which are extracted from each frame by a predefined detector. We set $\mathcal{A} = \{move(\theta)\}$ where $\theta \in \mathbb{R}$ and represent the discrete actions of the network

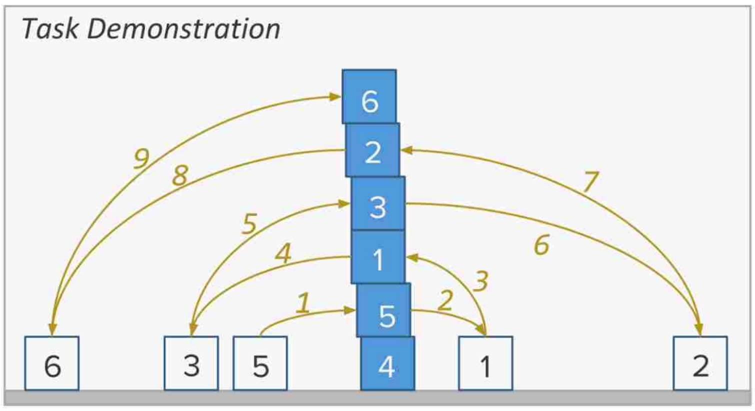


```
(move 1
  (- (* 0.0086 ball-y)
     (* 0.008858 agent-y)))

(move 2
  (* 0.0088
     (- ball-y agent-y)))

(move 3
  (* -0.0088
    (- (- agent-y ball-y)
       1.1744)))
```

Figure 42: The π-machine applied to explaining the behaviour of a DQN agent playing ATARI Pong. The 3 best programs obtained for each system are shown and annotated on the log-error plot (bottom).

`left`, `right`, `nop` as $move(1)$, $move(-1)$, $move(0)$ respectively. We specify the action error function as $\sigma_{act} = \|\hat{\theta} - \theta\|_2$ and set $\sigma_{len} = 0$.

The best 3 programs found by the π-machine are shown in Fig. 42 (top right), where it took 38 iterations for the best solution (average iteration duration 3.2s). By inspecting the second solution it becomes clear that the neural network behaviour can be explained as a proportional controller minimising the vertical distance between the agent and the ball. However, the best solution reveals even more structure in the behaviour of the DQN agent. The coefficient in front of the agent position is slightly larger than the one in front of the ball position which results in a small amount of damping in the motion of the paddle. Thus, it is evident that the DQN not only learns



```
Induced Program                              Interpretation
(let [param [-0.05 1.17]]
  (pick   cube-5-loc)                        0 (pick   cube-5)
  (place (+ cube-4-loc param))               1 (place (above cube-4))
  (pick   cube-1-loc)                        2 (pick   cube-1)
  (place (+ cube-5-loc param))               3 (place (above cube-5))
  (pick   cube-3-loc)                        4 (pick   cube-3)
  (place (+ cube-1-loc param)                5 (place (above cube-1))
  (pick   cube-2-loc)                        6 (pick   cube-2)
  (place (+ cube-3-loc param)                7 (place (above cube-3))
  (pick   cube-6-loc)                        8 (pick   cube-6)
  (place (+ cube-2-loc param)))              9 (place (above cube-2))
```

Figure 43: Program induced by the π-machine explaining a human demonstration of tower building (induced for 27 iterations and 7.2s).

the value of each game state, but also the underlying dynamics of controlling the paddle. Furthermore, we have tested the performance of an agent following a greedy policy defined by the induced program. In our experiments over 100 games this agent achieved a score of $11.1(\pm 0.17)$. This is not quite the score of $18.9(\pm 1.3)$ obtained by an optimised DQN, but it is better than human performance 9.3 [146]. This difference of course emanates from the predefined set of detectors not capturing all aspects of what the perceptual layers in the DQN have learnt, so improved detector choices should yield interpretable programs that also attain performance closer to the higher score of the black-box policy.

5.5.3 *Learning by Demonstration*

We consider a learning by demonstration scenario where a person demonstrates how to build a tower in a virtual simulated 2D environment. A typical demonstration is depicted in Fig. 43 (top). Our goal is to learn a program describing the demonstration such that it could later be utilised by an autonomous agent.

We are interested in how a person moves the cubes through the entire demonstration. So, we set $\mathcal{A} = \{pick(\theta), place(\theta)\}$, where $\theta \in \mathbb{R}^2$ is a 2D location. We specify the action and length error functions as

$$\sigma_{act}(\hat{a}, \hat{\theta}, a, \theta) = \begin{cases} \|\hat{\theta} - \theta\|_2 & \text{if } \hat{a} = a \\ d_{max} & \text{otherwise} \end{cases} \qquad \sigma_{len}(T', T) = (T' - T)^2 \tag{58}$$

where d_{max} is the maximum distance within the simulated 2D environment. The states in the observation trace contain the 2D position of every cube in the environment.

We tested the π-machine on 300 demonstrations and it successfully induced a program for each one of them. On average, 67 iterations (average iteration duration 0.4s) were needed per demonstration. The solution for one of the demonstrations is shown in Fig. 43. Closer inspection reveals that the program not only describes which cube was picked up and put where, but also what spatial relations are considered in the placement. The π-machine induced and optimised a free parameter which in combination with the function addition is used to encode the relation "above" as `(+ loc [-0.05 1.17])`. This type of abstract knowledge is crucial for efficient task learning from demonstration.

5.6 Discussion

Even though the π-machine can be applied to a relatively wide range of tasks, as we have shown so far, the symbols it can work with and the programs it can induce are not sufficiently general in order to tackle truly interesting problems, such as finding new algorithms, forming scientific theories or reverse engineering software components. These challenging tasks are likely to remain unsolved for decades, however the π-machine is a small, but exciting, step in this direction and so we outline several suggestions for further research.

Firstly, the π-machine assumes that the observed symbols in the execution trace can be represented in some real vector space, such that the gradient is well defined and continuous optimisation techniques can be directly applied. However, there certainly are symbols of interest that belong to discrete sets. For example, encoding whether a certain entity exists in the environment requires a boolean variable, whereas representing the count of something is naturally expressed with integers. A promising research direction is to integrate within the π-machine an integer optimisation method together with an SMT solver in order to efficiently search over discrete symbols.

Secondly, programs currently induced by the π-machine do not support branching because, even though gradients can flow through each individual branch, they cannot flow through the branching condition. Conditional branching requires methods for efficiently searching over possible conditions from an assumed class of expressions. Modern solvers for syntax-guided program synthesis [9] seem to be particularly well fitted for the purpose of finding a suitable branching condition.

Furthermore, the programs induced by the π-machine are equivalent to combinational logic as they can express arbitrary boolean expressions (e.g. by using modulo 2 arithmetic), but cannot carry any state information from one time step to the next one. Hence, an important step is to expand the class of supported programs by allowing them to use memory. Currently, programs can include parameters allocated on the stack (see Fig. 43), but they are treated as constants. An intuitive way of enabling state persistence across time is to induce a tail recursive call invoking the program for the next time step, instead of the π-machine explicitly calling the program at each time step. Such a differentiable program would effectively encode an RNN and so techniques such as backpropagation through time can still be applied to the program. Additionally, given the finite size of the input arguments, various methods for approximately solving Partially Observable Markov Decision Processs (POMDPs) can also be applied [27]. Another possibility is to induce programs which manipulate memory allocated on the heap, however this dramatically increases the size of the search space and so most likely techniques such as sketching would be needed [196].

Importantly, all of the aforementioned extensions are needed to reach the expressive power of a Turing machine. Assuming that branching is available, then the introduction of tail recursion would enable the π-machine to induce programs equivalent to a Finite State Machine (FSM), whereas general recursion, supported by a call stack, would result in programs equivalent to a Pushdown Automaton (PDA). Finally, providing the induced programs with the ability to randomly access memory (or work with a queue or a deque), together with branching, would result in programs equivalent to a Turing machine. In theory, Turing completeness is an important property, but in practice performing program induction within a Turing complete language is a challenging problem. Perlis [168] warns us exactly about this phenomena *"Beware of the Turing tar-pit in which everything is possible but nothing of interest is easy."*. Therefore, targeting richer DSL's, rather than aiming for Turing completeness, seems to be a more fruitful direction for the immediate future.

Finally, the Neural Turing Machine (NTM) [70, 71] can be viewed, in a way, as the end-to-end analogue of the π-machine as it induces a program that best fits the data. However, the program induced by the NTM is encoded in the weights of the RNN controller and cannot be inspected as the LISP-like programs induced by the π-machine. Nevertheless, the NTM successfully solves complex problems by purely learning from data, so it would certainly be interesting to extract symbolic programs from the NTM itself, in a fashion similar to explaining a DQN.

5.7 Concluding Remarks

The π-machine can also be viewed as a framework for automatic network architecture design [10, 93, 102, 149, 241], as different models can be expressed as concise LISP-like programs (see Fig. 39). Deep learning methods for limiting the search space of possible programs, which poses the greatest challenge, have been proposed [17, 128, 217], but how they can be applied to more generic settings where labelled training data is not available is an open question. The specification of variable detectors not only addresses this issue, but enables the user to make targeted and well grounded queries about the observed data trace.

In our view, the ability to induce simple interpretable models, which can explain observed data is crucial not only for the further advancement of machine learning and its applications, but also for the development of robust autonomous agents. Enabling robots to infer physical laws or the structure behind a human demonstration can speed up the process of learning immensely and potentially make zero shot learning tractable in realistic scenarios.

In conclusion, we proposed a novel architecture, the π-machine, for inducing LISP-like functional programs from observed data traces by utilising backpropagation, stochastic gradient descent and A* search. The experimental results demonstrate that the π-machine can efficiently and successfully induce interpretable programs from short data traces.

6 Future work and Conclusions

6.1 Key Ideas

This dissertation represents an attempt to bridge the gap between data driven machine learning methods and classic symbolic AI approaches in order to improve the capabilities of autonomous agents. There are 2 key ideas at the conceptual core of our work that allow us to think about highly structured symbolic systems and universal learners such as neural networks in the same framework.

Task decomposition

Any task that is to be executed by an embodied autonomous agent can be naturally decomposed into two stages - i) perceiving the world and ii) making decisions. This decomposition enables us to apply statistical machine learning methods to the perception problem in order to extract symbolic representations, i. e. semantically meaningful patterns, from data such that they can be processed by a formal symbolic system emulating the decision making process. Such a hybrid neuro-symbolic architecture combines the merits of both schools of thought, potentially providing capabilities that neither of them can achieve independently. Similar ideas have been proposed and studied before, however only in the past few years, with the advancement of deep learning, we have obtained general learners that can truly cope with challenging pattern recognition problems.

Task encoding programs

As we have demonstrated, programs are sufficiently expressive to represent a decision making procedure for various tasks given the right set of symbols is available. On one hand, task encoding programs can provide a strong inductive bias for the statistical machine learning based perception modules, and thus ensuring that the required set of symbols will be learnt. On the other hand, the available set of symbols shapes the space of possible programs and tasks that can be expressed. There is a bidirectional interaction, in the context of embodied autonomous agents, between learning task encoding programs and symbolic representations of the world, which we find intriguing and discuss further in the following section.

6.2 Future Work & Open Questions

6.2.1 *Scaling to Natural Tasks and Environments*

Scaling the proposed methods to natural tasks and environments poses a set of challenging scientific and engineering problems caused by the inevitable variations in the world and corresponding sensory signals. State of the art pattern recognition methods such as deep learning seem to perform well provided that vast amounts of labelled data is available. However, if autonomous agents are to truly work closely with people, then safety becomes a critical concern, hence the need for interpretable machine learning. Embedding programs into the decision making processes carried out by autonomous agents will allow for formal validation and verification techniques to be applied as demonstrated by some recent and seminal works [61, 170].

6.2.2 *Joint Learning of Tasks Encoding Programs and Symbolic World Representations*

As already mentioned, there are intricate bidirectional interactions between the available symbolic world representations and the possible task encoding programs that can be expressed. We have demonstrated that conditioning on any one of them enables the learning of the other. Is it possible, however, to learn both of them jointly? Our intuition is that simply applying techniques such as expectation maximisation would not yield satisfactory results due to the highly structured nature of the problem. A particular direction that we believe to be more fruitful is to find a way of sequencing tasks in a suitable curriculum leading to the gradual learning of more and more complex programs and symbols constantly alternating between one or the other.

6.2.3 *Lifelong Learning*

Lifelong learning is the only way to cope with constantly varying worlds and robustly adapt to changes. We find the idea of maintaining a pool of perceptors (as introduced in chapter 4), which are able to extract various symbolic representations from sensory data, particularly appealing for life long learning. Whenever a new symbol is to be recognised a corresponding perceptor can be added to the pool and kept up-to-date. The size of the pool can be bounded by removing obsolete perceptors or merging similar ones. It is not obvious, however, how to maintain a similar pool of learnt programs. Certainly, entire programs will be too specific to be useful and so key sub-programs will have to be extracted. Such an architecture could potentially store perceptual and procedural knowledge in a form easily transferable to new tasks and extendable from new experiences.

6.3 Concluding Remarks

In this dissertation we addressed the problem of learning structured task related abstractions. Firstly, we considered learning symbolic representations of the world assuming that a task encoding program is available and proposed a framework applicable to collaborative HRI scenarios together with a more general algorithm that supports *any* task encoding program. Our results clearly indicate that task encoding programs provide a strong inductive bias sufficient for learning relevant symbolic representations of the world. Secondly, we considered the complement problem of learning a task encoding program assuming that symbolic description of the environment is available. We proposed an efficient program induction procedure for learning task encoding programs from observation traces and demonstrated that they not only capture important structural features of the task, but also result in interpretable representations of black-box models such as deep neural networks. A pervasive theme throughout the entire thesis was the synergetic combination of the statistical machine learning and symbolic AI paradigms.

A

Probabilistic Programming with Anglican

A.1 Probabilistic Programming

The main goal of probabilistic programming is to bridge the gap between the theoretical desing and the actual implementation of probabilistic models by bringing together the fields of machine learning, statistics and programming languages as shown in Fig. 44. Probabilistic languages replace the common notion of a program variable with a random variable and so operations performed with the variables would typically require some sort of inference that is automatically performed by a built-in inference engine. There are numerous probabilistic programming languages [1] which can be classified by the type of supported probabilistic models and available inference engine.

In this tutorial we introduce Anglican [225] which is a functional high-order probabilistic language implemented as a DSL in Clojure. It supports more than 10 inference algorithms based on sampling, including ones that support massive parallelism, and a large number of probabilistic primitives. Additionally, it can cope with models that change their structure over time and have a varying number of latent variables. We have used Anglican to implement the main inference procedure in GLIDE (chapter 3) and Clojure to implement the π-machine (chapter 5). Besides the technical aspects, both Clojure and Anglican have influenced the conceptual core of this thesis by shaping the author's views about programs and their representational power. In

1 http://probabilistic-programming.org/wiki/Home

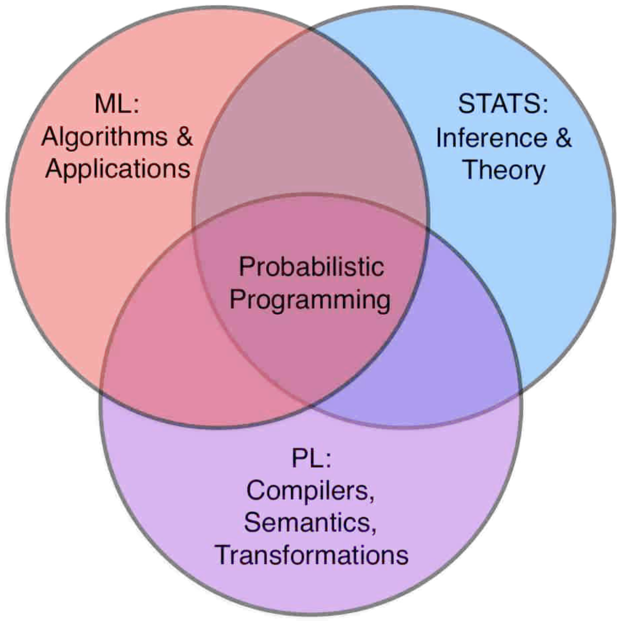


Figure 44: Probabilistic programming as a multidisciplinary field.

this short tutorial we first provide a brief introduction to Clojure and then demonstrate how several simple probabilistic models can be implemented in Anglican.

A.2 Introduction to Clojure

Clojure [2] is a functional language which runs on the Java virtual machine that can be best described as a modern LISP dialect with efficient data structures designed for parallelism. Why should one consider learning Clojure? After all, isn't it just another programming language? Clojure offers a unique and well balanced mixture between powerful abstract concepts (e. g. metaprogramming and transducers) and practical aspects (e. g. full compatibility with the Java ecosystem). In general, we find it disappointing, that functional programming languages are extremely underused for machine learning while some of their key features such as functions as first class citizens, immutable data structures and pure functions are particularly well suited for processing large amounts of data.

2 http://clojure.org/

A.2.1 *Basic Syntax*

Functions in Clojure are called by enclosing the function name, followed by the corresponding arguments if any, in brackets

```
(println "Hello world!")
```

Every single possible operation that one might want to perform in Clojure is a function. The arithmetic expression $\frac{10(2.1+4.3)}{2}$ is therefore calculated by

```
(/ (* 10 (+ 2.1 4.3)) 2)
>> 32.0
```

and logic operators are also functions that return a Boolean value

```
(and (= 3.14 3.14) (not= true false))
>> true
```

Conditional statements are not an exception either as they are functions that take 3 arguments: i) condition, ii) value to be returned if the condition is true and iii) value to be returned otherwise. Therefore, the following expressions

```
(if (<= 3 3) (+ 10 10) 0)
>> 20
(+ (if (< 4 5) 1 2) 3)
>> 4
```

are perfectly valid Clojure.

A.2.2 *Scoped Variables*

Variables can be defined within the scope of a `let` expression, which is a function that takes a list of variable definitions and a series of expressions and returns the value of the last one. For example, the following snippet evaluates to 12

```
(let [x 10
      y 2]
  (+ x y))
>> 12
```

and so does that

```
(let [x 10
      y 2]
  (* x 3)
  (+ x y))
>> 12
```

One typically uses expressions with side effects before the last one

```
(let [x 10
      y 2]
  (println "x times 3 =" (* x 3))
  (+ x y))
>> 12
```

A.2.3 *Functions*

Functions are first class citizens in Clojure and so can be defined as scoped variables. The following snippet defines the function $f(x, y) = 2x + y + 3$ and returns the value at $f(5, 10)$:

```
(let [f (fn [x y]
          (+ (* 2 x) y 3))]
  (f 5 10))
>> 23
```

Functions are the building blocks of Clojure programs, therefore there is an even shorter way to define the same $f(x, y)$ function

```
(let [f #(+ (* 2 %1) %2 3)]
  (f 5 10))
>> 23
```

If the function depends on a single argument one can even write

```
(let [square #(* % %)]
  (square 16))
>> 256
```

The following defines a function globally, not just in the scope of a `let` expression,

```
(defn f [x y]
  (+ (* 2 x) y 3))

(f 5 10)
>> 23
```

A function calculating $n!$ in Clojure can be written as

```
(defn factorial [n]
  (if (> n 1)
    (* n (factorial (- n 1)))
    1))

(factorial 5)
>> 120
```

These examples already start to demonstrate the succinctness of Clojure which is often found confusing by beginners, but enables the experienced programmer to focus entirely on thinking about the problem and spend minimal effort on dealing with syntax and language constructs.

A.2.4 *Data Structures*

Clojure implements the following immutable data structures:

- lists: `(1 2 3)`
- vectors: `[1 2 3]`
- hash maps: `{:a 1 :b 2 :c 3}`
- sets: `{1 2 3}`

Despite the fact that they are immutable, techniques such as path copying are internally used in order to reduce the computational burden to an absolute minimum. Technically speaking, the structures are classified as either sequences or collections, but any further details are out of the scope of this tutorial. The following code snippets show how some standard operations are performed on lists, vectors and hash maps.

A.2.4.1 *Lists*

```
(list 1 2 3)          ;; create a list
>> (1 2 3)
(first (list 1 2 3))  ;; extract the first element
>> 1
(rest (list 1 2 3))   ;; return all items after the first one
>> (2 3)
(count (list 1 2 3))  ;; return the length of a list
>> 3
(range 5)             ;; create a range of numbers starting from 0
>> (0 1 2 3 4)
(range 2 8)           ;; create a range of numbers
>> (2 3 4 5 6 7)
```

A.2.4.2 *Vectors*

```
[1 2 3]               ;; create a vector
>> [1 2 3]
(first [1 2 3])       ;; extract the first element
>> 1
(rest [1 2 3])        ;; return all items after the first one
>> [2 3]
(nth [1 2 3] 2)       ;; return the nth element
>> 3
(count [1 2 3])       ;; return the length of a list
>> 3
(vec (range 5))       ;; create a range of numbers starting from 0
>> [0 1 2 3 4]
```

```
(vec (range 2 8))     ;; create a range of numbers
>> [2 3 4 5 6 7]
```

A.2.4.3 *Hash Maps*

```
{:a 1 :b 2 :c 3}           ;; create a hash map
>> {:a 1 :b 2 :c 3}
(keyword "a")              ;; typically keywords are used as keys
>> :a
{:a 1 "b" 2 [3 4] 5}       ;; but any structure can be used as a key
>> {:a 1 "b" 2 [3 4] 5}
(get {:a 1 :b 2 :c 3} :a)  ;; query for a key
>> 1
(:a {:a 1 :b 2 :c 3})      ;; keywords are query functions
>> 1
```

Closer look into the data literals reveals that Clojure is a homoiconic language meaning that the Clojure source code itself is a data structure literal. This facilitates metaprogramming and the creation of various macroses.

A.2.5 *Higher-Order Functions*

Being a functional language, Clojure natively supports a large number of higher-order functions. Filtering a sequence with a given predicate is performed by

```
  (filter #(> % 0)           ;; predicate function
          [-1 1 -2 2 -3 3])  ;; sequence
  >> (1 2 3)
```

We can square all the numbers in a sequence, for example, using the `map` function

```
  (map #(* % %)         ;; mapping function
       (list 1 2 3 4))  ;; sequence
  >> (1 4 9 16)
```

The function used by `map` can take multiple arguments as well

```
  (map #(* %1 %2)       ;; mapping function
       (list 1 2 3 4)   ;; sequence 1
       (list 1 2 3 4))  ;; sequence 2
  >> (1 4 9 16)
```

The last important higher-order function is `reduce` with which we can, for example, sum all the elements of a list such as

```
  (reduce +         ;; reduction function
          0.0       ;; initial value
          [1 2 3])  ;; sequence
  >> 6.0
```

Finally, we demonstrate the standard filter-map-reduce composition by summing the squares of all even numbers in the range 1 to 100:

```
(reduce +                                ;; sum
        (map #(* % %)                    ;; square
             (filter #(mod % 2)          ;; even number?
                     (range 1 101))))    ;; sequence (1 ... 100)
>> 338350
```

If no initial value is provided to `reduce` then the reduction function is directly applied to the first two elements of the sequence. Programmers with procedural programming background would be often tempted to use loops while a filter-map-reduce operation will suffice. The benefit of using filter-map-reduce operations is that standard parallelisation techniques can be easily applied with Clojure. General loops in Clojure are usually implemented by tail recursion, however this is out of the scope of this tutorial.

A.3 Introduction to Anglican

We have covered sufficient amount of Clojure functionality such that in principle we can express numerous probabilistic models with Anglican. The Anglican website[3] is a great resource for learning the language in depth, with many examples and probabilistic models. In this tutorial we only present several simple ones in order to demonstrate what probabilistic programming is and how elegant Anglican makes it.

A.3.1 *Probabilistic Primitives*

Since the sole purpose of Anglican is to represent probabilistic models it offers and extensive range of probabilistic primitives. For example, we can sample various distributions with a single line:

```
(sample (uniform-continuous 3 10))  ;; U(x; a = 3, b = 10)
>> 4.659805616613525

(sample (normal 1 2))               ;; N(x; μ = 1, σ = 2)
>> -0.16795121075210417

(sample (flip 0.7))                 ;; Bern(x; μ = 0.7)
>> true

(sample (beta 2 3))                 ;; Beta(x; a = 2, b = 3)
>> 0.29186700108925917
```

3 http://www.robots.ox.ac.uk/~fwood/anglican/language/

```
(sample (binomial 10 0.4))           ;; Bin(x; n = 10, p = 0.4)
>> 6

(sample (discrete [0.3 0.2 0.5]))    ;; Cat(x; p1 = 0.3, p2 = 0.2, p3 = 0.5)
>> 2
```

A very useful high-order function for generating a sequence of samples is

```
(repeatedly 10 #(sample (normal 1 2)))
>> (-0.381 3.224 2.309 0.467 2.132 2.010 -0.046 2.970 2.468 -1.308)
```

which in this case generates a sequence of 10 samples from $\mathcal{N}(x; \mu = 1, \sigma = 2)$.

Similarly, we can easily estimate the log-pdf of any distribution using `observe` as

```
(observe (normal 0 1) 3)
>> -5.418938533204672
```

returns the value of $\log \mathcal{N}(3; 0, 1)$.

A.3.2 *Single Coin Flip*

Consider the outcome of a coin flip experiment

$$y \sim \text{Bern}(\theta) \tag{59}$$

where our belief about the bias of the coin is

$$\theta \sim \text{Beta}(5, 3) \tag{60}$$

and we are interested in the following query

$$p(\theta > 0.7 | y = \mathit{true}) = ? \tag{61}$$

We can easily compute the ground truth as

$$p(\theta > 0.7 | y = \mathit{true}) = 0.448 = 1 - \text{BetaCDF}(0.7 | 6, 3) \tag{62}$$

We can now “simulate” as many coin flip trials as we need by sampling the model and evaluating the likelihood of the sample with respect to the observed data and the specified query. Here is the Anglican description of the probabilistic model:

```
(defquery single-flip [y]
  (let [theta (sample (beta 5 3))]
    (observe (flip theta) y)
    (> theta 0.7)))
```

Probabilistic models written in Anglican are called queries and are defined using `defquery`. `defquery` looks a lot like a standard Clojure function definition, with the important difference that its contents is actually Anglican code that is a DSL written in Clojure designed to look like Clojure. The query is named `single-flip` and it takes a single argument `y`, which is the observed outcome of the coin flip experiment. `theta` is defined in the `let` expression as a random variable sampled according to our prior belief distribution `(beta 5 3)`. The `observe` statement asserts that we observe a value for the random variable `y` distributed according to `(flip theta)`. The final expression encodes the query of interest `(> theta 0.7)` and its value is returned once the query is executed. An Anglican query can be executed, or sampled, with `doquery`. In our case we would like to use importance sampling and generate 1000 samples of the answer to the problem query:

```
(let [samples (take 1000                   ;; number of samples
                    (doquery :importance   ;; sampling method
                             single-flip   ;; name of the query
                             [true]))]     ;; observed data
  (first samples))
>> {:log-weight -0.09819641371168224, :result true, :predicts []}
```

As you can see each sample is a hash map containing the log-weight of the sample as well as the return value from the query. The log-weight is automatically calculated by the inference engine by simply accumulating likelihood obtained from every call to the `observe` function. We can get the final answer by extracting the result from every sample and counting

```
(frequencies (map :result samples))
>> {true 376, false 624}
```

or by using the provided Anglican functions for analysing the obtained samples

```
(empirical-distribution (collect-results samples))
>> {true 0.47493018486959776, false 0.5250698151304022}
```

The resulting answer is $p(\theta > 0.7|y = true) \approx 0.474$ which is a relatively good estimate given that we used only 1000 samples.

A.3.3 *Multiple Coin Flips*

We now consider a modification of the coin flip experiment, whereby we observe a sequence of flips. The updated model is

$$\theta \sim \mathrm{Beta}(5,3) \tag{63}$$

$$y_i \sim \mathrm{Bern}(\theta) \tag{64}$$

and the query of interest is

$$p(\theta > 0.7|y_i) = ? \tag{65}$$

for some sequence of observations $\{y_i\}_{i=1}^{N}$. We modify the `single-flip` query by letting the input argument to be a sequence and use the high-order function `map` to `observe` all outcomes

```
(defquery many-flips [ys]
  (let [theta (sample (beta 5 3))
        outcome-dist (flip theta)]
    (map #(observe outcome-dist %) ys)
    (> theta 0.7)))
```

Only minor changes were needed in order to arrive at the `many-flips` query. The most important one is the fact that we now need to store the coin flip outcome distribution such that all data values can be observed with respect to it. We query the model by

```
(let [outcomes [true false false true]
      samples (take 1000 (doquery :importance many-flips [outcomes]))]
  (empirical-distribution (collect-results samples)))
>> {true 0.20849362899815674, false 0.7915063710018433}
```

resulting in $p(\theta > 0.7|y_i) \approx 0.21$.

A.3.4 *Curve Fitting with Model Selection*

The last problem which we sketch a solution for with Anglican demonstrates its capabilities to work with models with a varying number of latent variables. Curve fitting is usually posed as an optimization problem and most of us have fitted at least a line using methods such as least squares. However, we will take a probabilistic approach and show that we can not only fit a polynomial to the data, but also simultaneously infer the best polynomial degree for the purpose. Let's assume that we have a set of observed data points $\mathcal{D} = \{(x_1, y_1), \ldots (x_N, y_N)\}$ generated by

$$c(w, m) = w_m x^m + \ldots + w_1 x + w_0 \tag{66}$$

where

$$m \sim \text{Poisson}(2) \tag{67}$$

$$w|m \sim \mathcal{N}(w; 0, \mathbb{I}_{m+1}) \tag{68}$$

$$y|m, w \sim \mathcal{N}(y; c(w, m), 1) \tag{69}$$

$y \in \mathbb{R}$ is always a scalar, but the polynomial parameters $w \in \mathbb{R}^{m+1}$ change their dimensionality based on m. We have put a Poisson prior on the power of the polynomial m expecting a quadratic to be good fit.

We set the underlying function to be $f(x) = -4x^3 + 3x^2 - 2x + 5$ and generate the dataset $\mathcal{D}$ as follows

```
(defn f [x]
  (+ (* -4 x x x)
     (* 3 x x)
     (- 2 x)
     5))

(defn noisy-f [x]
  (+ (* 0.01 (f x))
     (sample (normal 0 4))))

(def xs (range -10 10 0.5))
(def ys (map noisy-f xs))
```

where `xs` and `ys` are global variables which we will use later.

A.3.4.1 *Linear Algebra Functions*

Since we are dealing with multivariate distributions, we provide a short list of relevant linear algebra functions in Clojure that we utilise:

```
(zero-vector 3)
>> [0.0 0.0 0.0]

(identity-matrix 3)
>> [[1.0 0.0 0.0]
    [0.0 1.0 0.0]
    [0.0 0.0 1.0]]
```

```
(shape (identity-matrix 3))
>> [3 3]

(dot [1 1 1] [1 1 1])
>> 3
```

Equipped with these functions we can now implement the curve equation by

```
(defn curve [w x]
  (let [m (first (shape w))
        phi (map #(pow %1 %2)
                 (vec (repeat m x))
                 (range 0 m 1))]
    (dot w phi)))
```

and the corresponding Anglican query is

```
(defquery fit-curve [xs ys]
  (let [m (sample (poisson 2))
        w (sample (mvn (zero-vector (+ m 1))
                       (identity-matrix (+ m 1))))]
    (map #(observe (normal (curve w x) 1.0) y)) xs ys)
    {:m m :w w}))
```

which returns a hash map with the sampled values for m and w.

We sample the model with

```
(let [samples (take 10000 (doquery :importance fit-curve [xs ys]))
  (first samples))
   {:log-weight -26084.717520229737,
    :result {:m 2, :w [-1.459,-0.262,-0.634]},
    :predicts []}
```

The technicalities around extracting the inferred value of m and w from the obtained samples are a bit involved, so we will omit them here, however they can be found in the author's repository [4]. The underlying function, the noisy data points, the sampled curves and the inferred curve are all shown in Fig. 45.

4 https://github.com/svepe/ppaml-tutorial

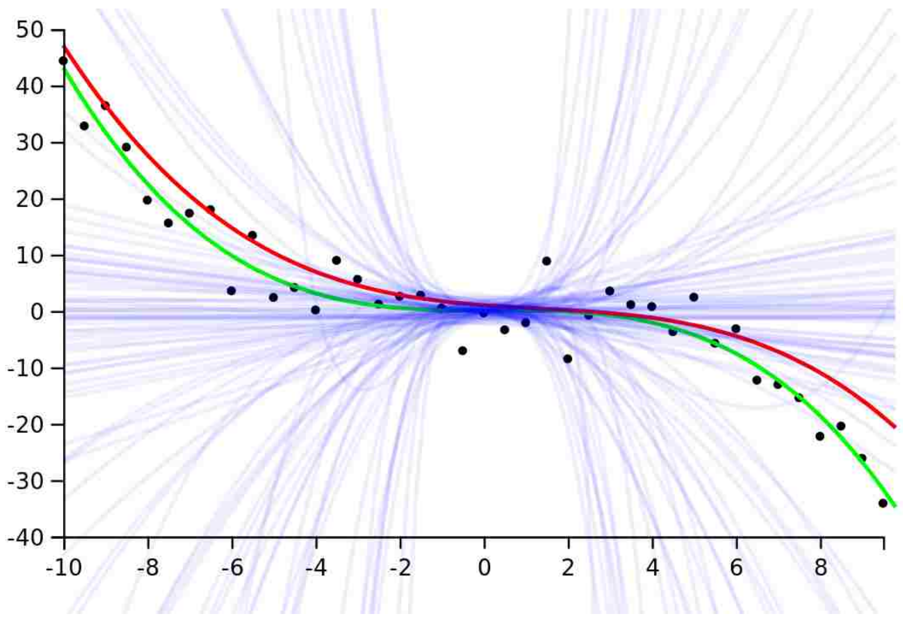


Figure 45: The underlying curve is in green and the observed noisy data points are shown as black dots. The fitted curve is in red and the generated samples are in semi-transparent blue.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin et al. 'Tensorflow: Large-scale machine learning on heterogeneous distributed systems'. In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Pieter Abbeel and Andrew Y Ng. 'Apprenticeship learning via inverse reinforcement learning'. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1.

[3] Pieter Abbeel, Adam Coates, Morgan Quigley and Andrew Y Ng. 'An application of reinforcement learning to aerobatic helicopter flight'. In: *Advances in neural information processing systems*. 2007, pp. 1–8.

[4] Nichola Abdo, Henrik Kretzschmar, Luciano Spinello and Cyrill Stachniss. 'Learning manipulation actions from a few demonstrations'. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1268–1275.

[5] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik and Sergey Levine. 'Learning to poke by poking: Experiential learning of intuitive physics'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 5074–5082.

[6] Samuel K Ainsworth, Nicholas J Foti, Adrian KC Lee and Emily B Fox. 'oi-VAE: Output Interpretable VAEs for Nonlinear Group Factor Analysis'. In: *International Conference on Machine Learning*. 2018, pp. 119–128.

[7] Eren Erdal Aksoy, Alexey Abramov, Johannes Dörr, Kejun Ning, Babette Dellen and Florentin Wörgötter. 'Learning the semantics of object–action relations by observation'. In: *The International Journal of Robotics Research* 30.10 (2011), pp. 1229–1249.

[8] Miltiadis Allamanis, Marc Brockschmidt and Mahmoud Khademi. 'Learning to represent programs with graphs'. In: *International Conference on Learning Representations (ICLR) Workshop Track* (2018).

[9] Rajeev Alur, Rishabh Singh, Dana Fisman and Armando Solar-Lezama. 'Search-based program synthesis'. In: *Communications of the ACM* 61.12 (2018), pp. 84–93.

[10] Jacob Andreas, Marcus Rohrbach, Trevor Darrell and Dan Klein. 'Learning to compose neural networks for question answering'. In: *arXiv preprint arXiv:1601.01705* (2016).

[11] Jacob Andreas, Marcus Rohrbach, Trevor Darrell and Dan Klein. 'Neural module networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 39–48.

[12] Brenna D Argall, Sonia Chernova, Manuela Veloso and Brett Browning. 'A survey of robot learning from demonstration'. In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[13] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage and Anil Anthony Bharath. 'Deep reinforcement learning: A brief survey'. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[14] Masataro Asai and Alex Fukunaga. 'Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary'. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[15] Christopher G Atkeson, BPW Babu, N Banerjee, D Berenson, CP Bove, X Cui, M DeDonato, R Du, S Feng, P Franklin et al. 'What happened at the DARPA Robotics Challenge, and why'. In: *submitted to the DRC Finals Special Issue of the Journal of Field Robotics* 1 (2016).

[16] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[17] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin and Daniel Tarlow. 'DeepCoder: Learning to Write Programs'. In: *arXiv preprint arXiv:1611.01989* (2016).

[18] Andrew G Barto and Sridhar Mahadevan. 'Recent advances in hierarchical reinforcement learning'. In: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41–77.

[19] Faisal I Bashir, Ashfaq A Khokhar and Dan Schonfeld. 'Object trajectory-based activity classification and recognition using hidden Markov models'. In: *IEEE transactions on Image Processing* 16.7 (2007), pp. 1912–1919.

[20] Yoshua Bengio, Aaron Courville and Pascal Vincent. 'Representation learning: A review and new perspectives'. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[21] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley and Yoshua Bengio. 'Theano: A CPU and GPU math compiler in Python'. In: *Proc. 9th Python in Science Conf.* 2010, pp. 1–7.

[22] Emmanuel Bezenac, Arthur Pajot and Patrick Gallinari. 'Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge'. In: *International Conference on Learning Representations*. 2018.

[23] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel and Urs Muller. 'Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car'. In: *arXiv preprint arXiv:1704.07911* (2017).

[24] Léon Bottou, Jonas Peters, Joaquin Quinonero Candela, Denis Xavier Charles, Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Y Simard and Ed Snelson. 'Counterfactual reasoning and learning systems: the example of computational advertising.' In: *Journal of Machine Learning Research* 14.1 (2013), pp. 3207–3260.

[25] Diane Bouchacourt, Ryota Tomioka and Sebastian Nowozin. 'Multi-level variational autoencoder: Learning disentangled representations from grouped observations'. In: *arXiv preprint arXiv:1705.08841* (2017).

[26] Matthew Brand. *Coupled hidden Markov models for modeling interacting processes*. 1997.

[27] Darius Braziunas. 'Pomdp solution methods'. In: *University of Toronto* (2003).

[28] William Brendel and Sinisa Todorovic. 'Learning spatiotemporal graphs of human activities'. In: *Computer vision (ICCV), 2011 IEEE international conference on*. IEEE. 2011, pp. 778–785.

[29] Marc Brockschmidt, Miltiadis Allamanis, Alexander L Gaunt and Oleksandr Polozov. 'Generative Code Modeling with Graphs'. In: *arXiv preprint arXiv:1805.08490* (2018).

[30] Rodney Brooks. 'Intelligence Without Representation'. In: *Artificial Intelligence* 47 (1991), pp. 139–159.

[31] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh and Pushmeet Kohli. 'Leveraging grammar and reinforcement learning for neural program synthesis'. In: (Apr. 2018).

[32] Jonathon Cai, Richard Shin and Dawn Song. 'Making neural programming architectures generalize via recursion'. In: ().

[33] Jonathon Cai, Richard Shin and Dawn Song. 'Making neural programming architectures generalize via Recursion'. In: *International Conference on Learning Representations (ICLR)*. Apr. 2017.

[34] Murray Campbell, A Joseph Hoane Jr and Feng-hsiung Hsu. 'Deep blue'. In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[35] Arun Chaganty, Aditya Nori and Sriram Rajamani. 'Efficiently sampling probabilistic programs via program analysis'. In: *Artificial Intelligence and Statistics*. 2013, pp. 153–160.

[36] Tian Qi Chen, Xuechen Li, Roger Grosse and David Duvenaud. 'Isolating Sources of Disentanglement in Variational Autoencoders'. In: *arXiv preprint arXiv:1802.04942* (2018).

[37] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever and Pieter Abbeel. 'Infogan: Interpretable representation learning by information maximizing generative adversarial nets'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2172–2180.

[38] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg and Dario Amodei. 'Deep Reinforcement Learning from Human Preferences'. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Curran Associates, Inc., 2017, pp. 4299–4307. url: http://papers.nips.cc/paper/7017-deep-reinforcement-learning-from-human-preferences.pdf.

[39] Michael W Cole, Patryk Laurent and Andrea Stocco. 'Rapid instructed task learning: A new window into the human brain's unique capacity for flexible cognitive control'. In: *Cognitive, Affective, & Behavioral Neuroscience* 13.1 (2013), pp. 1–22.

[40] Michael W Cole, Joset A Etzel, Jeffrey M Zacks, Walter Schneider and Todd S Braver. 'Rapid transfer of abstract rules to novel contexts in human lateral prefrontal cortex'. In: *Frontiers in human neuroscience* 5 (2011), p. 142.

[41] *Computer Go Games*. http://games-photos-blog.blogspot.com/2015/12/computer-go-games.html. Accessed: 2018-07-07.

[42] George Cybenko. 'Approximation by superpositions of a sigmoidal function'. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[43] Emily L Denton et al. 'Unsupervised learning of disentangled representations from video'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 4417–4426.

[44] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdelrahman Mohamed and Pushmeet Kohli. 'RobustFill: Neural Program Learning under Noisy I/O'. In: *arXiv preprint arXiv:1703.07469* (2017).

[45] James Diebel and Sebastian Thrun. 'An application of markov random fields to range sensing'. In: *Advances in neural information processing systems*. 2006, pp. 291–298.

[46] Yan Duan, Xi Chen, Rein Houthooft, John Schulman and Pieter Abbeel. 'Benchmarking deep reinforcement learning for continuous control'. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[47] John Duchi, Elad Hazan and Yoram Singer. 'Adaptive subgradient methods for online learning and stochastic optimization'. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[48] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann and Pushmeet Kohli. 'A dual approach to scalable verification of deep networks'. In: *Uncertainty in Artificial Intelligence, Conference on*. 2018.

[49] Kevin Ellis, Armando Solar-Lezama and Josh Tenenbaum. 'Sampling for Bayesian program learning'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 1297–1305.

[50] Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama and Josh Tenenbaum. 'Learning Libraries of Subroutines for Neurally–Guided Bayesian Program Induction'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 7816–7826.

[51] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama and Josh Tenenbaum. 'Learning to infer graphics programs from hand-drawn images'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 6062–6071.

[52] Mahmoud Elmezain, Ayoub Al-Hamadi, Jorg Appenrodt and Bernd Michaelis. 'A hidden markov model-based continuous gesture recognition system for hand motion trajectory'. In: *2008 19th International Conference on Pattern Recognition*. IEEE. 2008, pp. 1–4.

[53] S. M. Ali Eslami et al. 'Neural scene representation and rendering'. In: *Science* 360.6394 (2018), pp. 1204–1210.

[54] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton et al. 'Attend, infer, repeat: Fast scene understanding with generative models'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3225–3233.

[55] Richard Evans and Edward Grefenstette. 'Learning explanatory rules from noisy data'. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 1–64.

[56] Shai Fine, Yoram Singer and Naftali Tishby. 'The hierarchical hidden Markov model: Analysis and applications'. In: *Machine learning* 32.1 (1998), pp. 41–62.

[57] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine and Pieter Abbeel. 'Deep spatial autoencoders for visuomotor learning'. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 512–519.

[58] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel and Sergey Levine. 'One-Shot Visual Imitation Learning via Meta-Learning'. In: *Conference on Robot Learning*. 2017, pp. 357–368.

[59] Tom Foulsham, Craig Chapman, Eleni Nasiopoulos and Alan Kingstone. 'Top-down and bottom-up aspects of active search in a real-world environment.' In: *Canadian journal of experimental psychology = Revue canadienne de psychologie expérimentale* 68.1 (Mar. 2014), pp. 8–19.

[60] Marco Fraccaro, Simon Kamronn, Ulrich Paquet and Ole Winther. 'A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3604–3613.

[61] Nathan Fulton and André Platzer. 'Safe Reinforcement Learning via Formal Methods'. In: (2018).

[62] Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, SM Eslami and Oriol Vinyals. 'Synthesizing programs for images using reinforced adversarial learning'. In: *arXiv preprint arXiv:1804.01118* (2018).

[63] Artur d'Avila Garcez, Aimore Resende Riquetti Dutra and Eduardo Alonso. 'Towards Symbolic Reinforcement Learning with Common Sense'. In: *arXiv preprint arXiv:1804.08597* (2018).

[64] Marta Garnelo, Kai Arulkumaran and Murray Shanahan. 'Towards deep symbolic reinforcement learning'. In: *arXiv preprint arXiv:1609.05518* (2016).

[65] Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman and Daniel Tarlow. 'Differentiable Programs with Neural Libraries'. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 1213–1222.

[66] Tobias Gerstenberg, Matthew F Peterson, Noah D Goodman, David A Lagnado and Joshua B Tenenbaum. 'Eye-tracking causality'. In: *Psychological science* 28.12 (2017), pp. 1731–1744.

[67] Zoubin Ghahramani and Michael I Jordan. 'Factorial hidden Markov models'. In: *Advances in Neural Information Processing Systems*. 1996, pp. 472–478.

[68] Paul Glimcher. *Foundations of Neuroeconomic Analysis*. Oxford University Press, 2011.

[69] 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*. 2014, pp. 2672–2680.

[70] Alex Graves, Greg Wayne and Ivo Danihelka. 'Neural turing machines'. In: *arXiv preprint arXiv:1410.5401* (2014).

[71] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou et al. 'Hybrid computing using a neural network with dynamic external memory'. In: *Nature* 538.7626 (2016), pp. 471–476.

[72] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman and Phil Blunsom. 'Learning to transduce with unbounded memory'. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1828–1836.

[73] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman and Phil Blunsom. 'Learning to transduce with unbounded memory'. In: *Advances in Neural Information Processing Systems*. 2015, pp. 1828–1836.

[74] Junxia Gu, Xiaoqing Ding, Shengjin Wang and Youshou Wu. 'Action and gait recognition from recovered 3-D human joints'. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 40.4 (2010), pp. 1021–1033.

[75] Sumit Gulwani. 'Automating string processing in spreadsheets using input-output examples'. In: *ACM SIGPLAN Notices*. Vol. 46. 1. ACM. 2011, pp. 317–330.

[76] Sumit Gulwani, William R Harris and Rishabh Singh. 'Spreadsheet data manipulation using examples'. In: *Communications of the ACM* 55.8 (2012), pp. 97–105.

[77] Sumit Gulwani and Mark Marron. 'Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation'. In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM. 2014, pp. 803–814.

[78] Sumit Gulwani, Oleksandr Polozov and Rishabh Singh. 'Program Synthesis'. In: *Foundations and Trends in Programming Languages* 4.1-2 (2017), pp. 1–119.

[79] David Ha and Jürgen Schmidhuber. 'World Models'. In: *arXiv preprint arXiv:1803.10122* (2018).

[80] Stevan Harnad. 'The symbol grounding problem'. In: *Physica D: Nonlinear Phenomena* 42.1 (1990), pp. 335 –346.

[81] Brent Harrison, Upol Ehsan and Mark O Riedl. 'Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations'. In: *arXiv preprint arXiv:1702.07826* (2017).

[82] John Haugeland. *Artificial intelligence: The very idea*. MIT press, 1989.

[83] MM Hayhoe and A Shrivastava. 'Visual memory and motor planning in a natural task'. In: *Journal of Vision* (2003), pp. 49–63.

[84] Mary Hayhoe and Dana Ballard. 'Eye movements in natural behavior.' In: *Trends in cognitive sciences* 9.4 (2005), pp. 188–94.

[85] 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'. In: *arXiv preprint arXiv:1707.02286* (2017).

[86] J Henderson. 'Human gaze control during real-world scene perception'. In: *Trends in Cognitive Sciences* 7.11 (Nov. 2003), pp. 498–504.

[87] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele and Trevor Darrell. 'Generating visual explanations'. In: *European Conference on Computer Vision*. Springer. 2016, pp. 3–19.

[88] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar and David Silver. 'Rainbow: Combining Improvements in Deep Reinforcement Learning'. In: *arXiv preprint arXiv:1710.02298* (2017).

[89] Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell and Alexander Lerchner. 'Darla: Improving zero-shot transfer in reinforcement learning'. In: *arXiv preprint arXiv:1707.08475* (2017).

[90] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed and Alexander Lerchner. 'β-VAE: Learning basic visual concepts with a constrained variational framework'. In: (2017).

[91] Irina Higgins, Nicolas Sonnerat, Loic Matthey, Arka Pal, Christopher P Burgess, Matthew Botvinick, Demis Hassabis and Alexander Lerchner. 'SCAN: learning abstract hierarchical compositional visual concepts'. In: *International Conference on Learning Representations (ICLR)* (2018).

[92] Yordan Hristov, Svetlin Penkov, Alex Lascarides and Subramanian Ramamoorthy. 'Grounding Symbols in Multi-Modal Instructions'. In: *Proceedings of the ACL 17 workshop Language Grounding for Robotics*. Vancouver, Aug. 2017.

[93] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell and Kate Saenko. 'Learning to reason: End-to-end module networks for visual question answering'. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 804–813.

[94] Ronghang Hu, Jacob Andreas, Trevor Darrell and Kate Saenko. 'Explainable neural computation via stack neural module networks'. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 53–69.

[95] Weiming Hu, Guodong Tian, Yongxin Kang, Chunfeng Yuan and Stephen Maybank. 'Dual sticky hierarchical Dirichlet process hidden Markov model and its application to natural language description of motions'. In: *IEEE transactions on pattern analysis and machine intelligence* 40.10 (2018), pp. 2355–2373.

[96] Chien-Ming Huang and Bilge Mutlu. 'Anticipatory Robot Control for Efficient Human-Robot Collaboration'. In: *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*. 2016, pp. 83–90.

[97] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan and Chrisina Jayne. 'Imitation learning: A survey of learning methods'. In: *ACM Computing Surveys (CSUR)* 50.2 (2017), p. 21.

[98] Andrew Irish, Iraj Mantegh and Farrokh Janabi-Sharifi. 'A PbD approach for learning pseudo-periodic robot trajectories over curved surfaces'. In: *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE. 2010, pp. 1425–1432.

[99] Raban Iten, Tony Metger, Henrik Wilming, Lidia del Rio and Renato Renner. 'Discovering physical concepts with neural networks'. In: *arXiv preprint arXiv:1807.10300* (2018).

[100] Eric Jang, Shixiang Gu and Ben Poole. 'Categorical reparameterization with gumbel-softmax'. In: *arXiv preprint arXiv:1611.01144* (2016).

[101] Roland S. Johansson, Göran Westling, Anders Bäckström and J. Randall Flanagan. 'Eye–Hand Coordination in Object Manipulation'. In: *The Journal of Neuroscience* 21.17 (2001), pp. 6917–6932.

[102] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick and Ross Girshick. 'Inferring and executing programs for visual reasoning'. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2989–2998.

[103] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix and Dileep George. 'Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics'. In: *International Conference on Machine Learning*. 2017, pp. 1809–1818.

[104] Neel Kant. 'Recent Advances in Neural Program Synthesis'. In: *arXiv preprint arXiv:1802.02353* (2018).

[105] Russell Kaplan, Christopher Sauer and Alexander Sosa. 'Beating atari with natural language guided reinforcement learning'. In: *arXiv preprint arXiv:1704.05539* (2017).

[106] Maximilian Karl, Maximilian Soelch, Justin Bayer and Patrick van der Smagt. 'Deep variational bayes filters: Unsupervised learning of state space models from raw data'. In: *International Conference on Learning Representations*. 2017.

[107] Hyunjik Kim and Andriy Mnih. 'Disentangling by factorising'. In: *arXiv preprint arXiv:1802.05983* (2018).

[108] Jinkyu Kim and John Canny. 'Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention'. In: *arXiv preprint arXiv:1703.10631* (2017).

[109] Diederik P Kingma and Max Welling. 'Auto-encoding variational bayes'. In: *International Conference on Learning Representations*. 2014.

[110] Stephen Cole Kleene, NG de Bruijn, J de Groot and Adriaan Cornelis Zaanen. *Introduction to metamathematics*. Vol. 483. van Nostrand New York, 1952.

[111] Ronald Kline. 'Cybernetics, automata studies, and the Dartmouth conference on artificial intelligence'. In: *IEEE Annals of the History of Computing* 33.4 (2011), pp. 5–16.

[112] Thomas Kollar, Stefanie Tellex, Matthew R Walter, Albert Huang, Abraham Bachrach, Sachi Hemachandra, Emma Brunskill, Ashis Banerjee, Deb Roy, Seth Teller et al. 'Generalized grounding graphs: A probabilistic framework for understanding grounded language'. In: *Journal of Artificial Intelligence Research* (2013).

[113] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[114] Soheil Kolouri, Mohammad Rostami, Yuri Owechko and Kyungnam Kim. 'Joint dictionaries for zero-shot learning'. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[115] John R Koza. 'Genetic programming as a means for programming computers by natural selection'. In: *Statistics and computing* 4.2 (1994), pp. 87–112.

[116] John Kser, Marc Brockschmidt, Alexander Gaunt and Daniel Tarlow. 'Differentiable Functional Program Interpreters'. In: *arXiv preprint arXiv:1611.01988v2* (2017).

[117] T Kulkarni, Ilker Yildirim, Pushmeet Kohli, W Freiwald and Joshua B Tenenbaum. 'Deep generative vision as approximate bayesian computation'. In: *NIPS 2014 ABC Workshop*. 2014.

[118] Tejas D Kulkarni, Vikash K Mansinghka, Pushmeet Kohli and Joshua B Tenenbaum. 'Inverse graphics with probabilistic CAD models'. In: *arXiv preprint arXiv:1407.1339* (2014).

[119] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli and Josh Tenenbaum. 'Deep convolutional inverse graphics network'. In: *Advances in neural information processing systems*. 2015, pp. 2539–2547.

[120] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum and Vikash Mansinghka. ‘Picture: A probabilistic programming language for scene perception’. In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2015, pp. 4390–4399.

[121] Karol Kurach, Marcin Andrychowicz and Ilya Sutskever. ‘Neural random-access machines’. In: *arXiv preprint arXiv:1511.06392* (2015).

[122] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart Russell and Pieter Abbeel. ‘Learning Plannable Representations with Causal InfoGAN’. In: *arXiv preprint arXiv:1807.09341* (2018).

[123] Brenden M. Lake, Ruslan Salakhutdinov and Joshua B. Tenenbaum. ‘Human-level concept learning through probabilistic program induction’. In: *Science* 350.6266 (2015), pp. 1332–1338.

[124] Johnny Lam. *Control of an Inverted Pendulum*. 2004.

[125] Michael F Land. ‘Eye movements and the control of actions in everyday life.’ In: *Progress in retinal and eye research* 25.3 (May 2006).

[126] Michael Land, Neil Mennie and Jennifer Rusted. ‘The roles of vision and eye movements in the control of activities of daily living’. In: *Perception* 28.11 (1999), pp. 1311–1328.

[127] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. ‘Deep learning’. In: *Nature* 521.7553 (2015), p. 436.

[128] Woosuk Lee, Kihong Heo, Rajeev Alur and Mayur Naik. ‘Accelerating search-based program synthesis using learned probabilistic models’. In: *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM. 2018, pp. 436–449.

[129] Tao Lei, Regina Barzilay and Tommi Jaakkola. ‘Rationalizing neural predictions’. In: *arXiv preprint arXiv:1606.04155* (2016).

[130] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan et al. ‘Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model’. In: *The Annals of Applied Statistics* 9.3 (2015), pp. 1350–1371.

[131] Hector J Levesque. ‘Planning with loops’. In: *IJCAI*. 2005, pp. 509–515.

[132] Sergey Levine, Chelsea Finn, Trevor Darrell and Pieter Abbeel. 'End-to-end training of deep visuomotor policies'. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[133] Sergey Levine, Peter Pastor, Alex Krizhevsky and Deirdre Quillen. 'Learning hand-eye coordination for robotic grasping with large-scale data collection'. In: *International Symposium on Experimental Robotics*. Springer. 2016, pp. 173–184.

[134] Chunyuan Li, Heerad Farkhoor, Rosanne Liu and Jason Yosinski. 'Measuring the Intrinsic Dimension of Objective Landscapes'. In: *International Conference on Learning Representations*. 2018.

[135] 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'. In: *arXiv preprint arXiv:1509.02971* (2015).

[136] Wang Ling, Dani Yogatama, Chris Dyer and Phil Blunsom. 'Program Induction by Rationale Generation:Learning to Solve and Explain Algebraic Word Problems'. In: *arXiv preprint arXiv:1705.04146* (2017).

[137] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[138] Dougal Maclaurin, David Duvenaud and Ryan P Adams. 'Autograd: Effortless gradients in numpy'. In: *ICML 2015 AutoML Workshop*. 2015.

[139] Alireza Makhzani. 'Implicit Autoencoders'. In: *arXiv preprint arXiv:1805.09804* (2018).

[140] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly and Ian Goodfellow. 'Adversarial Autoencoders'. In: *International Conference on Learning Representations*. 2016. URL: http://arxiv.org/abs/1511.05644.

[141] Zohar Manna and Richard Waldinger. 'A deductive approach to program synthesis'. In: *Readings in artificial intelligence and software engineering*. Elsevier, 1986, pp. 3–34.

[142] Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov and Josh Tenenbaum. 'Approximate bayesian image interpretation using generative probabilistic graphics programs'. In: *Advances in Neural Information Processing Systems*. 2013, pp. 1520–1528.

[143] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer and Dieter Fox. 'Learning to parse natural language commands to a robot control system'. In: *Experimental Robotics*. Springer. 2013, pp. 403–415.

[144] Cynthia Matuszek, Liefeng Bo, Luke Zettlemoyer and Dieter Fox. 'Learning from Unscripted Deictic Gesture and Language for Human-Robot Interactions.' In: *AAAI*. 2014, pp. 2556–2563.

[145] Matthew Mirman, Timon Gehr and Martin Vechev. 'Differentiable Abstract Interpretation for Provably Robust Neural Networks'. In: *International Conference on Machine Learning*. 2018, pp. 3575–3583.

[146] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski et al. 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540 (2015), pp. 529–533.

[147] Grégoire Montavon, Wojciech Samek and Klaus-Robert Müller. 'Methods for interpreting and understanding deep neural networks'. In: *Digital Signal Processing* (2017).

[148] Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós. 'ORB-SLAM: a Versatile and Accurate Monocular SLAM System'. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. doi: 10.1109/TRO.2015.2463671.

[149] Renato Negrinho and Geoff Gordon. 'DeepArchitect: Automatically Designing and Training Deep Architectures'. In: *arXiv preprint arXiv:1704.08792* (2017).

[150] Allen Newell and Herbert A. Simon. 'Computer Science As Empirical Inquiry: Symbols and Search'. In: *Commun. ACM* 19.3 (Mar. 1976), pp. 113–126. issn: 0001-0782.

[151] Nam Thanh Nguyen, Dinh Q Phung, Svetha Venkatesh and Hung Bui. 'Learning and detecting activities from movement trajectories using the hierarchical hidden Markov model'. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. IEEE. 2005, pp. 955–960.

[152] Aditya V Nori, Chung-Kil Hur, Sriram K Rajamani and Selva Samuel. 'R2: An Efficient MCMC Sampler for Probabilistic Programs.' In: *AAAI*. 2014, pp. 2476–2482.

[153] Junhyuk Oh, Satinder Singh, Honglak Lee and Pushmeet Kohli. 'Zero-shot task generalization with multi-task deep reinforcement learning'. In: *arXiv preprint arXiv:1706.05064* (2017).

[154] Aude Oliva and Antonio Torralba. 'The role of context in object recognition'. In: *Trends in cognitive sciences* 11.12 (2007), pp. 520–527.

[155] Edwin Olson. 'AprilTag: A robust and flexible visual fiducial system'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 3400–3407.

[156] Nikhil R Pal and Sankar K Pal. 'A review on image segmentation techniques'. In: *Pattern recognition* 26.9 (1993), pp. 1277–1294.

[157] Lucas Paletta, Katrin Santner, Gerald Fritz, Heinz Mayer and Johann Schrammel. '3D attention: measurement of visual saliency using eye tracking glasses'. In: *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. ACM. 2013, pp. 199–204.

[158] Sinno Jialin Pan, Qiang Yang et al. 'A survey on transfer learning'. In: *IEEE Transactions on knowledge and data engineering* 22.10 (2010), pp. 1345–1359.

[159] D Papadopoulos, A Clarke, F Keller and V Ferrari. 'Training object class detectors from eye tracking data'. In: *European Conference on Computer Vision (ECCV)*. 2014.

[160] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou and Pushmeet Kohli. 'Neuro-symbolic program synthesis'. In: *International Conference on Learning Representations (ICLR)*. Apr. 2017.

[161] Derek Partridge. 'The scope and limitations of first generation expert systems'. In: *Future Generation Computer Systems* 3.1 (1987), pp. 1–10.

[162] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra and Peter Battaglia. 'Learning model-based planning from scratch'. In: *arXiv preprint arXiv:1707.06170* (2017).

[163] Xue Bin Peng, Glen Berseth, KangKang Yin and Michiel Van De Panne. 'Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning'. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), p. 41.

[164] Svetlin Penkov, Alejandro Bordallo and Subramanian Ramamoorthy. 'Inverse Eye Tracking for Intention Inference and Symbol Grounding in Human-Robot Collaboration'. In: *Robotics Science and Systems (RSS), Workshop on Planning for Human-Robot Interaction* (June 2016).

[165] Svetlin Penkov, Alejandro Bordallo and Subramanian Ramamoorthy. 'Physical Symbol Grounding and Instance Learning through Demonstration and Eye Tracking'. In: *Robotics and Automation, 2017 IEEE International Conference on*. Singapore, June 2017.

[166] Svetlin Penkov and Subramanian Ramamoorthy. 'Using program induction to interpret transition system dynamics'. In: *Proceedings of ICML Workshop on Human Interpretability in Machine Learning*. Sydney, Aug. 2017.

[167] Svetlin Penkov and Subramanian Ramamoorthy. 'Learning Programmatically Structured Representations with Perceptor Gradients'. In: *International Conference on Learning Representations (ICLR)*. 2019.

[168] Alan Perlis. *Epigrapms in Programming*. http://www.cs.yale.edu/homes/perlis-alan/quotes.html. Accessed: 2019-03-06. 1982.

[169] Thies Pfeiffer, Patrick Renner and Nadine Pfeiffer-Leßmann. 'EyeSee3D 2.0: Model-based Real-time Analysis of Mobile Eye-tracking in Static and Dynamic Three-dimensional Scenes'. In: *Proceedings of the Ninth Biennial Symposium on Eye Tracking Research & Applications*. ACM, 2016, pp. 189–196.

[170] André Platzer. *Logical Foundations of Cyber-Physical Systems*. 2018.

[171] Illia Polosukhin and Alexander Skidanov. 'Neural Program Search: Solving Programming Tasks from Description and Examples'. In: *arXiv preprint arXiv:1802.04335* (2018).

[172] Oleksandr Polozov and Sumit Gulwani. 'FlashMeta: a framework for inductive program synthesis'. In: *ACM SIGPLAN Notices*. Vol. 50. 10. ACM. 2015, pp. 107–126.

[173] Christopher Poultney, Sumit Chopra, Yann L Cun et al. 'Efficient learning of sparse representations with an energy-based model'. In: *Advances in neural information processing systems*. 2007, pp. 1137–1144.

[174] Antonin Raffin, Sebastian Höfer, Rico Jonschkowski, Oliver Brock and Freek Stulp. 'Unsupervised learning of state representations for multiple tasks'. In: *International Conference on Learning Representations (ICLR)* (2017).

[175] Muhammad Asif Rana, Mustafa Mukadam, S Reza Ahmadzadeh, Sonia Chernova and Byron Boots. 'Towards robust skill generalization: Unifying learning from demonstration and motion planning'. In: *Proceedings of the 2017 Conference on Robot Learning (CoRL)*. 2017.

[176] Keith Rayner. 'Eye movements and attention in reading, scene perception, and visual search.' In: *Quarterly journal of experimental psychology* 62.8 (Aug. 2009), pp. 1457–506.

[177] Scott Reed and Nando De Freitas. 'Neural programmer-interpreters'. In: *arXiv preprint arXiv:1511.06279* (2015).

[178] Carlo Reverberi, Kai Görgen and John-Dylan Haynes. 'Compositionality of rule representations in human prefrontal cortex'. In: *Cerebral Cortex* 22.6 (2012), pp. 1237–1246.

[179] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg and Nicolas Heess. 'Unsupervised learning of 3d structure from images'. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4996–5004.

[180] Marco Ribeiro, Sameer Singh and Carlos Guestrin. 'Why Should I Trust You?: Explaining the Predictions of Any Classifier'. In: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2016.

[181] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot and Yoshua Bengio. 'Contractive auto-encoders: Explicit invariance during feature extraction'. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress. 2011, pp. 833–840.

[182] Daniel Ritchie, Ben Mildenhall, Noah D Goodman and Pat Hanrahan. 'Controlling procedural modeling programs with stochastically-ordered sequential Monte Carlo'. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 105.

[183] Lukasz Romaszko, Christopher KI Williams, Pol Moreno, Pushmeet Kohli, Jan Czarnowski, Stefan Leutenegger, Andrew J Davison, Renata Khasanova, Pascal Frossard, Iaroslav Melekhov et al. 'Vision-As-Inverse-Graphics: obtaining a rich 3d explanation of a scene from a single image'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2017, pp. 851–859.

[184] Stéphane Ross, Geoffrey Gordon and Drew Bagnell. 'A reduction of imitation learning and structured prediction to no-regret online learning'. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[185] Constantin A Rothkopf, Dana H Ballard and Mary M Hayhoe. 'Task and context determine where you look.' In: *Journal of vision* 7 (2007).

[186] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia and Tim Lillicrap. 'A simple neural network module for relational reasoning'. In: *Advances in neural information processing systems*. 2017, pp. 4974–4983.

[187] Ute Schmid. *Inductive Program Synthesis*. https://pdfs.semanticscholar.org/presentation/5c78/b38f6ae25b690333c9b7632cd7a47c199561.pdf. Accessed: 2018-08-15. 2010.

[188] Michael Schmidt and Hod Lipson. 'Distilling free-form natural laws from experimental data'. In: *Science* 324.5923 (2009), pp. 81–85.

[189] John Schulman, Xi Chen and Pieter Abbeel. 'Equivalence between policy gradients and soft q-learning'. In: *arXiv preprint arXiv:1704.06440* (2017).

[190] John R. Searle. 'Minds, brains, and programs'. In: *Behavioral and Brain Sciences* 3 (03 Sept. 1980), pp. 417–424.

[191] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal and Sergey Levine. 'Time-Contrastive Networks: Self-Supervised Learning from Video'. In: *arXiv preprint arXiv:1704.06888* (2017).

[192] Amirhossein Shantia, Rik Timmers, Lambert Schomaker and Marco Wiering. 'Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment'. In: *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE. 2015, pp. 1–7.

[193] Ehud Y. Shapiro. *Algorithmic Program Debugging*. Cambridge, MA, USA: MIT Press, 1983.

[194] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot et al. 'Mastering the game of Go with deep neural networks and tree search'. In: *nature* 529.7587 (2016), p. 484.

[195] Armando Solar-Lezama and Rastislav Bodik. *Program synthesis by sketching*. Citeseer, 2008.

[196] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia and Vijay Saraswat. 'Combinatorial sketching for finite programs'. In: *ACM SIGOPS Operating Systems Review* 40.5 (2006), pp. 404–415.

[197] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. 'Dropout: a simple way to prevent neural networks from overfitting'. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[198] Richard S Sutton, Andrew G Barto et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[199] *Symposium: Beautiful new world of work*. https://www.logistik-heute.de/Logistik-News-Logistik-Nachrichten/Markt-News/13286/IFF-Wissenschaftstage-beschaeftigten-sich-mit-den-Auswirkungen-der-Digitalis. Accessed: 2018-07-07.

[200] Mariarosaria Taddeo and Luciano Floridi. 'Solving the symbol grounding problem: a critical review of fifteen years of research'. In: *Journal of Experimental & Theoretical Artificial Intelligence* 17.4 (2005), pp. 419–445.

[201] Benjamin W. Tatler, Nicholas J. Wade, Hoi Kwan, John M. Findlay and Boris M. Velichkovsky. 'Yarbus, eye movements, and vision'. In: *i-Perception* 1.1 (2010), pp. 7–27.

[202] Yee Whye Teh, Dilan Görür and Zoubin Ghahramani. 'Stick-breaking Construction for the Indian Buffet Process.' In: *AISTATS*. 2007, pp. 556–563.

[203] Stefanie Tellex, Pratiksha Thaker, Joshua Joseph and Nicholas Roy. 'Learning perceptually grounded word meanings from unaligned parallel data'. In: *Ma-*

chine Learning Journal (Special Issue on Learning Semantics in Machine Learning) 94.2 (2014), pp. 151–167.

[204] Garrett Thomas, Melissa Chien, Aviv Tamar, Juan Aparicio Ojea and Pieter Abbeel. 'Learning Robotic Assembly from CAD'. In: *arXiv preprint arXiv:1803.07635* (2018).

[205] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum and Jiajun Wu. 'Learning to Infer and Execute 3D Shape Programs'. In: ().

[206] Seiya Tokui, Kenta Oono, Shohei Hido and Justin Clayton. 'Chainer: a next-generation open source framework for deep learning'. In: *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*. 2015.

[207] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly and Bernhard Schoelkopf. 'Wasserstein auto-encoders'. In: *International Conference on Learning Representations (ICLR)*. 2017.

[208] Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer and Phil Blunsom. 'Neural arithmetic logic units'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 8046–8055.

[209] Jonathan Tremblay, Thang To, Artem Molchanov, Stephen Tyree, Jan Kautz and Stan Birchfield. 'Synthetically Trained Neural Networks for Learning Human-Readable Plans from Real-World Demonstrations'. In: *arXiv preprint arXiv:1805.07054* (2018).

[210] Kewei Tu, Maria Pavlovskaia and Song-Chun Zhu. 'Unsupervised structure learning of stochastic and-or grammars'. In: *Advances in neural information processing systems*. 2013, pp. 1322–1330.

[211] Alan Mathison Turing. 'On computable numbers, with an application to the Entscheidungsproblem'. In: *Proceedings of the London mathematical society* 2.1 (1937), pp. 230–265.

[212] Frits Vaandrager. 'Model Learning'. In: *Commun. ACM* 60.2 (Jan. 2017), pp. 86–95.

[213] Aleksandar Vakanski, Iraj Mantegh, Andrew Irish and Farrokh Janabi-Sharifi. 'Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping'. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.4 (2012), pp. 1039–1052.

[214] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton and Swarat Chaudhuri. 'HOUDINI: Lifelong Learning as Program Synthesis'. In: *Advances in Neural Information Processing Systems*. 2018, pp. 8701–8712.

[215] D. Vasquez, B. Okal and K. O. Arras. 'Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison'. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 1341–1346.

[216] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli and Swarat Chaudhuri. 'Programmatically Interpretable Reinforcement Learning'. In: *arXiv preprint arXiv:1804.02477* (2018).

[217] Ashwin J Vijayakumar, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain and Sumit Gulwani. 'Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples'. In: *International Conference on Learning Representations (ICLR)*. 2018.

[218] Pascal Vincent, Hugo Larochelle, Yoshua Bengio and Pierre-Antoine Manzagol. 'Extracting and composing robust features with denoising autoencoders'. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[219] Paul Vogt. 'The physical symbol grounding problem'. In: *Cognitive Systems Research* 3.3 (2002), pp. 429–457.

[220] Manuel Watter, Jost Springenberg, 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*. 2015, pp. 2746–2754.

[221] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li et al. 'Imagination-augmented agents for deep reinforcement learning'. In: *arXiv preprint arXiv:1707.06203* (2017).

[222] D. Whitney, M. Eldon, J. Oberlin and S. Tellex. 'Interpreting multimodal referring expressions in real time'. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 3331–3338. doi: 10.1109/ICRA.2016.7487507.

[223] Ronald J Williams. 'Simple statistical gradient-following algorithms for connectionist reinforcement learning'. In: *Reinforcement Learning*. Springer, 1992, pp. 5–32.

[224] Ludwig Wittgenstein. *Philosophical investigations*. Macmillan Publishing Company, 1953.

[225] Frank Wood, Jan Willem van de Meent and Vikash Mansinghka. 'A New Approach to Probabilistic Programming Inference'. In: *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*. 2014, pp. 1024–1032.

[226] Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman and Josh Tenenbaum. 'Learning to see physics via visual de-animation'. In: *Advances in Neural Information Processing Systems*. 2017, pp. 153–164.

[227] Caiming Xiong, Nishant Shukla, Wenlong Xiong and Song-Chun Zhu. 'Robot learning with a spatial, temporal, and causal and-or graph'. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 2144–2151.

[228] Hongyu Yang, Cynthia Rudin and Margo Seltzer. 'Scalable Bayesian rule lists'. In: *arXiv preprint arXiv:1602.08610* (2016).

[229] Jie Yang, Yangsheng Xu and Chiou S Chen. 'Human action learning via hidden Markov model'. In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27.1 (1997), pp. 34–44.

[230] AlfredL. Yarbus. 'Eye Movements During Perception of Complex Objects'. English. In: *Eye Movements and Vision*. Springer US, 1967, pp. 171–211. isbn: 978-1-4899-5381-0.

[231] Weilie Yi and Dana Ballard. 'Recognizing Behavor in Hand-Eye Coordination Patterns'. In: *International Journal of Humanoid Robotics* 06.03 (2009), pp. 337–359.

[232] Pengcheng Yin and Graham Neubig. 'A Syntactic Neural Model for General-Purpose Code Generation'. In: *arXiv preprint arXiv:1704.01696* (2017).

[233] Halley Young, Osbert Bastani and Mayur Naik. 'Learning Neurosymbolic Generative Models via Program Synthesis'. In: *arXiv preprint arXiv:1901.08565* (2019).

[234] Tianhe Yu, Chelsea Finn, Sudeep Dasari, Annie Xie, Tianhao Zhang, Pieter Abbeel and Sergey Levine. 'One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning'. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018.

[235] Tom Zahavy, Nir Ben-Zrihem and Shie Mannor. 'Graying the black box: Understanding DQNs'. In: *arXiv preprint arXiv:1602.02658* (2016).

[236] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart et al. 'Relational Deep Reinforcement Learning'. In: *arXiv preprint arXiv:1806.01830* (2018).

[237] Matthew D Zeiler and Rob Fergus. 'Visualizing and understanding convolutional networks'. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[238] Lisa Zhang, Gregory Rosenblatt, Ethan Fetaya, Renjie Liao, William E Byrd, Raquel Urtasun and Richard Zemel. 'Leveraging Constraint Logic Programming for Neural Guided Program Synthesis'. In: *International Conference on Learning Representations (ICLR) Workshop Track*. 2018.

[239] Kemin Zhou, John Comstock Doyle, Keith Glover et al. *Robust and optimal control*. Vol. 40. Prentice hall New Jersey, 1996.

[240] Zuyuan Zhu and Huosheng Hu. 'Robot Learning from Demonstration in Robotic Assembly: A Survey'. In: *Robotics* 7.2 (2018), p. 17.

[241] Barret Zoph and Quoc Le. 'Neural Architecture Search with Reinforcement Learning'. In: *International Conference on Learning Representations (ICLR)*. Apr. 2017.