Learning Structured Representations for Perception and Control

by

Tejas Dattatraya Kulkarni

B.S., Purdue University (2010)

Submitted to the Department of Brain and Cognitive Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Signature redacted

Author . . . .
Department of Brain and Cognitive Science
August 12, 2016

Signature redacted

Certified by. . . . .
Joshua B. Tenenbaum
Paul E. Newton Career Development Professor of Cognitive Science and Computation
Thesis Supervisor

Signature redacted

Accepted by . . . .
Matthew A. Wilson
Sherman Fairchild Professor of Neuroscience and Picower Scholar
Director of Graduate Education for Brain and Cognitive Sciences

Learning Structured Representations for Perception and Control

by

Tejas Dattatraya Kulkarni

Submitted to the Department of Brain and Cognitive Science
on August 12, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

I argue that the intersection of deep learning, hierarchical reinforcement learning, and generative models provides a promising avenue towards building agents that learn to produce goal-directed behavior given sensations. I present models and algorithms that learn from raw observations and will emphasize on minimizing their sample complexity and number of training steps required for convergence. To this end, I introduce hierarchical variants of deep reinforcement learning algorithms, which produce and utilize temporally extended abstractions over actions. I also present a hybrid model-free and model-based deep reinforcement learning model, which can also be potentially used to automatically extract subgoals for bootstrapping temporal abstractions. I will then present a model-based approach for perception, which unifies deep learning and probabilistic models, to learn powerful representations of images without labeled data or external rewards.

Learning goal-directed behavior with sparse and delayed rewards is a fundamental challenge for reinforcement learning algorithms. The primary difficulty arises due to insufficient exploration, resulting in an agent being unable to learn robust value functions. I present the Deep Hierarchical Reinforcement Learning (h-DQN) approach, which integrates hierarchical value functions operating at different time scales, along with goal-driven intrinsically motivated behavior for efficient exploration. Intrinsically motivated agents can explore new behavior for its own sake rather than to directly solve problems. Such intrinsic behaviors could eventually help the agent solve tasks posed by the environment. h-DQN allows for flexible goal specifications, such as functions over entities and relations. This provides an efficient space for exploration in complicated environments. I will demonstrate h-DQN's ability to learn optimal behavior given raw pixels in environments with very sparse and delayed feedback.

I will then introduce the Deep Successor Reinforcement (DSR) learning approach. DSR is a hybrid model-free and model-based RL algorithm. It learns the value function of a state by taking the inner product between the state's expected future feature occupancy and the corresponding immediate rewards. This factorization of the value function has several appealing properties – increased sensitivity to changes in the reward structure and potentially the ability to automatically extract subgoals for learning temporal abstractions.

Finally, I argue for the need for better representations of images, both in reinforcement learning tasks and in general. Existing deep learning approaches learn useful representations given lots of labeled data or rewards. Moreover, they also lack the inductive biases needed to disentangle causal structure in images such as – objects, shape, pose and other intrinsic scene properties. I present generative models of vision, often referred to as *analysis-by-synthesis* approaches, by combining deep generative methods with probabilistic modeling. This approach aims to learn structured representations of images given raw observations. I argue that such intermediate representations will be crucial to scale-up deep reinforcement learning algorithms, and to bridge the gap between machine and human learning.

Thesis Supervisor: Joshua B. Tenenbaum
Title: Paul E. Newton Career Development Professor of Cognitive Science and Computation

Acknowledgments

This thesis is a culmination of many years of debates, disagreements, friendships, support, sacrifice and love.

I must begin with the person that taught me how to think. Everyone knows that Josh is one of the best thinkers of our time. But perhaps only a handful of students, friends and family have been lucky enough to be mentored by him. When I first came to MIT, I was profoundly lost. Josh's relentless enthusiasm for understanding the mind and his unwavering faith in himself was contagious. It is fair to say that I found new meaning in my research and life after being his student.

Next, it would have been difficult to continue my self-exploration without the guidance of Vikash Mansinghka. Vikash has always gone out of his way to help me. He introduced me to the world of AI and Machine Learning. He taught me how to be rigorous in my thinking, while avoiding pigeon holing myself into academic obscurity.

Without the love and sacrifice of my wife and parents, I wouldn't be able to write this thesis. I must begin with Nisha, the love of my life. She has been an unwavering source of energy and support. Academia can often make you feel excruciatingly isolated and lonely. Without her love and sacrifice, I wouldn't have been able to grow as much. Next my parents and younger sister Tanvi – they literally spent all their life savings to ship me to the US. I will be forever grateful for their support, sacrifice and love. For the last decade, Nisha's family has been a home away from home. I feel fortunate to be surrounded by such wonderful and supportive family.

I have been lucky to work with many collaborators. Without Karthik Rajagopal and Ardavan Saeedi, I would have never forayed into reinforcement learning. I will always cherish our late night discussions about the future of AI, intense gym sessions intermixed with philosophical discussions, and late night coding sessions to get our AI systems to converge. Sometime last year, a bunch of random students got together to take over an empty office space and called themselves the "AGI people". Will Whitney was the first one to join me. Will is one of those exceptional people with lots of raw intelligence. I learnt a lot from him and will always cherish our late night conversations about neural nets, information theory and physics. I also had the privilege to collaborate with some exception undergraduate researchers – Simanta Gautam, Michael Janner and Michael Chang. I must also thank Yura Perov, with whom I wrote my first paper in graduate school. Whenever I had any question about the brain, Ilker never hesitated to help out. Vlad has always been my go-to mathematician and I always learnt something new from talking to him. Sam Gershman's scientific work and his ability to effectively communicate ideas has always been inspiring. Without Pushmeet's optimism, it would have been difficult to push some of the earlier ideas in my academic career. Everyone has scientific and philosophical heroes, whose work inevitably end up shaping your own perspectives. Along with Josh, my perspective has been heavily influenced by – Rich Sutton, Geoff Hinton, Ray Solomonoff, Alan Turing.

My friends really made my MIT experience special. Pedro, Mikhil, Max KW, Max S, David, Rishi, You – I am forever grateful for your company. Pedro has been a very dear friend for the longest time. He always pushed and inspired me to lead a healthier life style. I will miss those endless conversations with him about life, philosophy, and the nature of happiness. Mikhil's curiosity has always struck me and every conversation with him has taught me something new. I hope at least some of us end up in the same city.

I am grateful to my committee members – Tommy Poggio, Ted Adelson and Bill Freeman. They gave me the utmost freedom to explore anything that I wanted to. I was also fortunate enough to be at MIT while the Center for Brains, Machines and Minds (CBMM) was being created. CBMM along with Siemens generously supported me for the final year of my PhD. I also want to acknowledge the Singleton and Leventhal fellowship for their generous support.

The research presented in chapters 2 and 3 was done in collaboration with Karthik Rajagopal, Ardavan Saeedi, Simanta Gautam, Sam Gershman and Joshua Tenenbaum. Finally, research in chapter 4&5 was done in collaboration with Will Whitney, Pushmeet Kohli, Vikash Mansinghka and Joshua Tenenbaum.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The science of intelligence is about the construction, analysis, and understanding of agents behaving in peculiar ways in different environments. Intelligence is multifaceted and encompasses many notions like induction, logic, perception, learning, problem solving, intrinsic motivation, knowledge representations and more. Ultimately, any theory of intelligence must formalize all of these facets within a coherent mathematical framework. Why does an agent need to posses or manifest all of these capabilities? It can be argued that it enables the agent to solve many goals in many environments.

The ability to perceive is one of the first capabilities that a human-like agent must possess. What are the models and algorithms that can learn state abstractions from raw sensations? Second, how can an agent learn higher levels of representations to solve many goals in a diverse set of environments? Existing learning algorithms are inferior than the way humans learn. Humans can learn autonomously, while using much less sample complexity [83], and they are also able to effectively transfer knowledge between tasks.

In this thesis, I take modest steps towards building more powerful perception and control systems that learn from experiences. I develop joint perception and control models, which learn spatio-temporal representations of pixels to produce goal-directed behavior. There has been remarkable progress in deep learning approaches, which learn abstract features given image, speech and text data [85]. There has also been lots of progress in learning goal-directed behavior using reinforcement learning algorithms [134]. These two disparate threads have been recently unified [49, 74, 94, 123] under the umbrella of so called deep reinforcement learning (deep RL) approaches. However, existing deep RL approaches struggle given sparse and delayed feedback, are slow to adapt to changes in rewards, and do not successfully make use of temporally extended abstractions over actions. In chapters 2&3, I present models to alleviate some these issues. See Figure 1-1a&b for an overview.

Finally, efficient goal learning mandates good representations of images. Although deep learning approaches such as Convolutional Neural Nets (CNNs) [84] provide a reasonable state abstractions, it is imperative to build systems that disentangle the causal structure in images to scale up deep RL approaches. In chapter 4&5, I present model-based approaches for perception, by combining ideas from unsupervised deep learning, probabilistic generative models, and computer graphics. See Figure 1-1c for an overview.

My working hypothesis is that by combining deep learning, hierarchical reinforcement learning and probabilistic generative modeling, it is possible to start building more human-like agents that learn to perceive and produce optimal behaviors towards solving goals.

1.2 Building Features, Theories or Explanations from Visual Observations

There is a long history of trying to understand what it means to "see" in philosophy and science. I take a pre-dominantly computational perspective and discuss two of the most appealing conceptual frameworks – (1)*Bottom-up Empirical Regressors*: sensory information passes through a series of computational processes to give rise to increasingly complex features and (2) Model or theory building (analysis-by-synthesis): a top-down computational process which iteratively refines itself to best explain in-

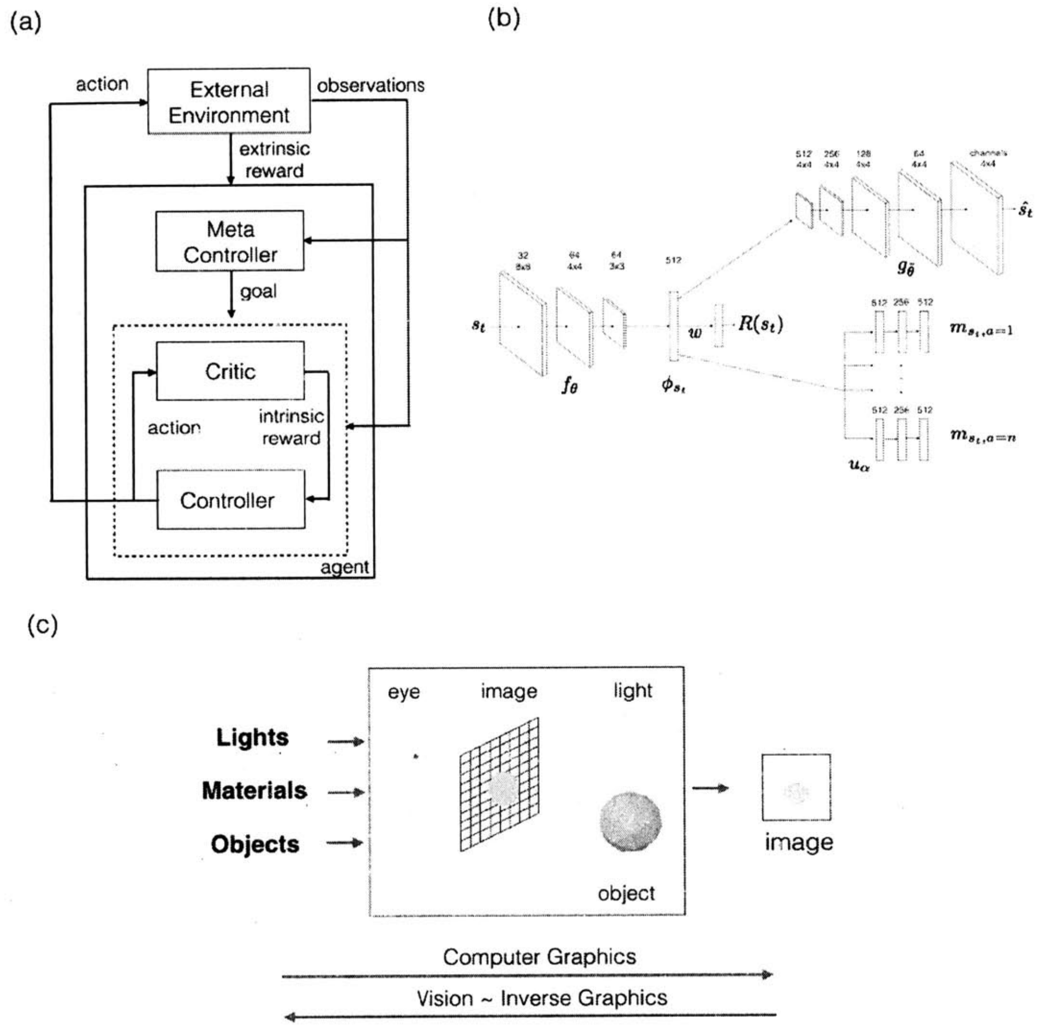


Figure 1-1: **Thesis Overview:** I present agents that learn to produce goal-directed behavior from pixels. This is achieved by algorithms that learn spatio-temporal representations from raw experiences. I also present model-based approaches for perception, which will be increasingly important for building goal seeking agents. **(a)** In chapter 2, I present the Deep Hierarchical Reinforcement learning approach (h-DQN), which integrates multiple value functions operating at different time scales, to enable efficient policy learning in complex environments. **(b)** In chapter 3, I present a hybrid model-free and model-based RL approach called the Deep Successor Reinforcement learning algorithm (DSR). DSR has several appealing properties – rapid sensitivity to distal changes in rewards and ability to extract subgoals from experiences. DSR could also be plugged directly into h-DQN for hierarchical RL. **(c)** Due to the sparse and delayed reward problem in RL, it can be difficult to learn features from images. Unsupervised deep learning, probabilistic generative models and their combinations can be utilized to build structured generative models that produce powerful representations of images. I present analysis-by-synthesis based approaches (also referred to as *inverse graphics*) for perception (chapter 4&5).

coming sensory information, often taking aid from bottom-up computational processes for inference.

1.2.1 Bottom-up Approach to Perception

Deep learning has led to remarkable breakthroughs in learning hierarchical representations of images. Models such as CNNs, Restricted Boltzmann Machines [53, 114], and Auto-encoders [10, 148] have been successfully applied to produce multiple layers of increasingly abstract feature representations. These techniques have been used to solve a variety of tasks including: object recognition [76], machine translation and reinforcement learning [94, 123, 93]. However, coming up with the best representation for any given task is still an open question, especially in the absence of labeled data or when the environment provides sparse and delayed rewards.

Various work [11, 17, 40, 107] has been done on the theory and practice of representation learning, and from this work a consistent set of desiderata for representations have emerged: invariance, interpretability, abstraction, and disentanglement. In particular, Bengio *et al.* [11] propose that a *disentangled* representation is one for which changes in the encoded data are sparse over real-world transformations; that is, changes in only a few latents at a time should be able to represent sequences which are likely to happen in the real world.

1.2.2 Analysis-by-Synthesis: Top-down Approach to Perception

Perhaps the oldest philosophical foundation for analysis-by-synthesis came from David Hume. Hume argued that ideas are mental images and minds only have indirect access to reality. Another early account was proposed by Hermann von Helmholtz in his book titled *Treatise on physiological optics* [149]. Helmholtz describes the nature of perception as –

"The general rule determining the ideas of vision that are formed whenever an impression is made on the eye, with or without the aid of optical instruments, is that

such objects are always imagined as being present in the field of vision as would have to be there in order to produce the same impression on the nervous mechanism, the eyes being used under ordinary normal conditions".

There is evidence that our own visual system seems to interpret sensory information via model building. The most notable evidence comes from optical illusions developed by Louis Necker [101], Friedrich Schumann [121], Gaetano Kanizsa [69] and others. A person watching the Necker cube can observe the two different 3D interpretations from the same 2D image. This suggests the presence of a top-down processes attempting to explain the incoming sensory information. In the Kanizsa triangle experiment, this top-down process is even more profound as the subject undergoing the experiment can hallucinate triangle edges that do not even exist in the original image.

The idea of *analysis-by-synthesis* was revived by Dayan et. al. [23] via a model termed as the *Helmholtz Machine.* This paper highlights the idea of self-supervised learning that relates the function of bottom-up and top-down cortical processing pathways. In this model, a feed-forward neural network takes in sensory data and produces a hierarchy of neural activations. A top-down generative network uses these hypothesized neural states to re-produce observations. In this way, the model uses iterative feed-forward passes and top-down synthesis to learn a compressed representation of data. After learning, the top layer of the model disentangles hidden factors of variations within data.

The *Helmholtz Machine* uses a neural network to model the top-down generative process. Although neural networks can approximate any computable continuous functions [19], there are no formal guarantees on the learnability of a network's weights for a specified task. Therefore, there is no a-priori reason to believe that a simple feed-forward decoder used in the *Helmholtz Machine* model can efficiently represent the generative process.

In order to create more flexible vision models, a powerful and principled conceptual framework is that of "vision as inverse graphics" (inverse graphics). Computer graphics depicts the process of going from structured description of scenes to image data. The task of vision can be though of as running this process backwards. Given observations, we are typically interested in obtaining the hidden properties of the scene including – number of objects, shape, texture, lighting, orientation, mass, etc. The field of computer graphics has seen rapid growth over the last few decades [62] and provides powerful and flexible graphics simulators that are very much relevant for the goal of inverse-graphics. Moreover, the idea of co-ordinate frames to relate whole-to-parts in graphics is crucial if we want our vision systems to have strong generalization properties [55]. There has been a lot of progress in parametric 3D shape models [4, 33, 13] and procedural graphics [112]. The insights from computer graphics will likely accelerate progress in analysis-by-synthesis, by providing new ideas for flexible model classes and constraints on the structure of generative models.

Computer vision began with approaches in a similar vein, especially the work of Larry Roberts who argued that "the perception of solid objects is a process which can be based on the properties of three-dimensional transformations and the laws of nature" [68]. More recently, probabilistic generative models for a range of image parsing tasks have been explored [144, 24, 145, 160, 153]. These provide an appealing avenue for integrating top-down constraints with bottom-up processing, and provide an inspiration for the inverse graphics approach. But like traditional bottom-up pipelines for vision, these approaches have relied on considerable problem specific engineering, chiefly to design and/or learn custom inference strategies, such as MCMC proposals [145, 160] that incorporate bottom up cues. Other combinations of top down knowledge with bottom up processing have been remarkably powerful [59]. For example, [58] has shown that global, 3D geometric information can significantly improve the performance of bottom-up object detectors. What's needed are automatic modeling and inference techniques to scale-up these approaches for richer visual reasoning tasks.

In chater 4, I present *Picture*, a probabilistic programming framework for visual scene perception. A picture program induces a probability distribution over visual scenes and applies a general-purpose inference algorithm for image scene interpretation. In order to scale up probabilistic inference, I develop inference amortization algorithms using deep neural networks.

Probabilistic generative models make it easy to express computations with compositional structure. However, this flexibility comes at a price – the programmer needs to express a lot of the structure and therefore it is harder to scale these approaches with increasing computation. In the final parts of my thesis, I develop models that combine the best of both worlds – take the compositional ideas from probabilistic generative models and use them to design structured deep generative models.

One of the first analysis-by-synthesis style neural network model was the Boltzmann machine [1], which was later constrained to layer-wise connections for tractability and termed as restricted boltzmann machines [56]. Since then, there have been plenty of papers on using encoder-decoder style architectures for unsupervised feature learning [108, 56]. In more recent work, structured deep generative models have been designed to disentangle multiple factors of variations such as – intrinsic images [139], objects [31, 61, 42], attention-based models to handle affine transformations [63], 3D transformations [82, 151, 156, 161].

In chapter 5, I present a new neural network based analysis-by-synthesis approach for learning interpretable features called the *Deep Convolutional Inverse Graphics Network* (DC-IGN) [82]. DC-IGN factors out latent factors such as 3D pose and lighting given contiguous frames of images. In the future, it will be crucial to use such structured generative models within deep RL for scaling it to handle more complex learning problems. Current deep RL agents spend most of their time on learning a visual system rather than higher level representations useful for learning policies. Deep generative models along with weakly supervised methods can greatly aid in pre-training a visual system before solving the reinforcement learning problem.

1.3 Learning Goal-directed Behavior

The ultimate objective of an agent could be formulated as the maximization of its expected utility function given observations. Utility theory dates back to work by Von Neumann and Morgenstern, where they defined rational actions for a decision-making agent. They made the observation that ideal decisions are actions that maximize an agent's expected utility. As utility and probability theory developed, probabilistic approaches dominated mainstream AI field in the late 1980s. This led to the development of probabilistic graphical models [106, 73], ultimately leading to several advances in approximate inference techniques, probabilistic programming, structure learning and probabilistic approaches for decision making under uncertainty. A historical overview of probabilistic approaches in decision making is summarized in [36, 38].

1.3.1 Reinforcement Learning

In a separate line of work, the field of reinforcement learning (RL) was taking shape motivated by the same questions but stemming from Psychology and Neuroscience. Some of the earliest work dates back to Thorndike's work on behavioral conditioning [141], Bellman's work on optimal control theory [9], Minsky's work on SNARCs (Stochastic Neural-Analog Reinforcement Calculators), and many others. Sutton & Barto's textbook on RL [134] provides a detailed historical overview of the field. Between 1970-80 [134], Sutton & Barto developed temporal-difference (TD) learning algorithms, inspired by the notion of secondary reinforcers. Watkin [150] then made the significant contribution of integrating TD learning with optimal control algorithms, to develop Q-learning.

From a theoretical perspective, a series of papers [110, 116] bridged reinforcement learning with universal induction as an idealized model of intelligence. One such idealized model is called AIXI; it maximizes the expected cumulative rewards received from the environment. At each time step, it checks every possible program, evaluates the possible reward achieved conditioned on an action, weighs the future reward by the program's Kolmogorov complexity (description length) and then selects the best action by averaging contributions from all the programs. However, AIXI is incomputable but there have been computable approximations of the model [147]. The constraint on description length of the program in AIXI effectively places an inductive bias to prefer programs with a lower Kolmogorov complexity. This is directly related to the problem of induction. Induction can be defined as the process of drawing 'best' conclusions from a set of observations [110]. Probabilistic generative models described in chapters 4&5 share the same goal - producing compact programs that can reproduce all of the given observations.

In 1990s, there was an explosion of work in combining linear function approximations with TD learning algorithms [67]. These ideas were generalized much later, when researchers started combining deep neural networks with reinforcement learning. One of the first empirical work was presented in [49], where the authors used evolutionary approaches to train a neural network to play Atari games. In a similar vein, [74] developed a reinforcement learning agent that learns to play a car racing games from raw pixels. Subsequently, [94] presented an impressive demonstration of deep reinforcement learning on Atari games, where the agent surpassed human-level performance on several of the games. More recently, many other approaches have emerged in the deep reinforcement learning literature including asynchronous actor critic methods [93], policy gradient methods [120, 52] . However, existing approaches suffer when the rewards are sparse and delayed. This makes it hard for the agent to efficiently explore the environment for learning robust value functions. These models also lack strong generalization properties across tasks (transfer learning) and their learning curves are significantly worse than humans [83].

1.3.2 Hierarchies in Reinforcement Learning

One of the main ideas to mitigate some of these problems was to use temporal abstractions in RL. Sutton et al.[136] proposed the *options* framework, which involves abstractions over the space of actions. At each step, the agent chooses either a one step "primitive" action or a "multi-step" action policy (option). Each option defines a policy over actions (either primitive or other options) and can be terminated according to a stochastic function. Thus, the traditional MDP setting can be extended to a semi-Markov decision process (SMDP) with the use of options. Recently, several methods have been proposed to learn options in real-time by using varying reward functions [138] or by composing existing options [128]. Value functions have also been generalized to consider goals along with states [115]. This universal value function

$V(s, g; \theta)$ provides an universal option that approximately represents optimal behavior towards the goal g.

Other related work for hierarchical formulations include the model of Dayan and Hinton [22] which consisted of "managers" taking decisions at various levels of granularity, percolating all the way down to atomic actions made by the agent. The MAXQ framework [27] built up on this work to decompose the value function of an MDP into combinations of value functions of smaller constituent MDPs, as did Guestrin et al.[47] in their factored MDP formulation.

Singh et al.[127] explored agents with intrinsic reward structures in order to learn generic options that can apply to a wide variety of tasks. Using a notion of "salient events" as sub-goals, the agent can learn options to get to such events. In the context of hierarchical RL, Goel and Huber [39] discuss a framework for subgoal discovery using the structural aspects of a learned policy model.

1.3.3 Intrinsic Motivation

The issue of sparse and delayed rewards is also related to the notion of intrinsic motivation in AI and Psychology. Intrinsically motivated agents can explore new behavior for its own sake rather than to directly solve problems. Such intrinsic behaviors could eventually help the agent solve tasks posed by the environment.

The nature and origin of supposedly good intrinsic reward functions is an open question in reinforcement learning. Oudeyer et al. categorized intrinsic motivation into three different models – knowledge based models, competence based models and morphological models [103].

Knowledge based models of intrinsic motivation relies on an error signal (value error, learning progress, reward error etc.) to build a pseudo-reward functions. Schmidhuber [117] provided a coherent formulation of knowledge based intrinsic motivation, which is measured by the improvements to a predictive world model made by the learning algorithm. Mohamed and Rezende [95] have recently proposed a notion of intrinsically motivated learning within the framework of mutual information maximization. Frank et al. [35] demonstrated the effectiveness of artificial curiosity using information gain maximization in a humanoid robot. Bellemare et al. [8] proposed a count-based model for training deep RL agents to facilitate deep exploration.

There has also been work on formulating intrinsic rewards from an evolutionary perspective. In another paper, Singh et al.[126] take an evolutionary perspective to optimize over the space of reward functions for the agent, leading to the notion of extrinsically and intrinsically motivated behavior.

The nature and origin of intrinsic rewards in humans is a thorny issue but there are some notable insights from existing literature, especially related to the space of competence based models of intrinsic motivation. There is converging evidence in developmental psychology that human infants, primates, children, and adults in diverse cultures base their core knowledge on certain cognitive systems including – entities, agents and their actions, numerical quantities, space, social-structures and intuitive theories [129, 83]. Even newborns and infants seem to represent the visual world in terms of coherent visual entities, centered around spatio-temporal principles of cohesion, continuity, and contact. They also seem to explicitly represent other agents, with the assumption that an agent's behavior is goal-directed and efficient. Infants can also discriminate relative sizes of objects, relative distances and higher order numerical relations such as the ratio of object sizes. During curiosity-driven activities, toddlers use this knowledge to generate intrinsic goals such as building physically stable block structures. In order to accomplish these goals, toddlers seem to construct sub-goals in the space of their core knowledge, such as – putting a heavier entity *on top of* (relation) a lighter entity in order to build tall blocks. I will primarily focus on hierarchical RL models of this nature. However, knowledge based and competence based models are complementary and should be combined.

1.3.4 Extracting subgoals to build options

As described in section , there are many ways to think about intrinsic rewards – subgoals in the space of $< object1, relation, object2 >$, decomposition of the SR representation, communication from other agents, goal embedding predicted from another value network, and more. Within a hierarchical RL framework, intrinsic rewards in the space of subgoals can then be defined and a lower level controller can learn to solve these subgoals. In Chapter 2, I present Deep Hierarchical Reinforcement Learning (h-DQN), which learns hierarchical values functions over subgoals to enable efficient goal driven exploration.

Knowledge of space can also be utilized to learn a hierarchical decomposition of spatial environments, where the bottlenecks between different spatial groupings correspond to sub-goals. This has been explored in Neuroscience with the successor representation, which represents a value function in terms of the expected future state occupancy. Decomposition of the successor representation (SR) yields reasonable sub-goals for spatial navigation problems [21, 37, 130]. Botvinick et al.[16] have written a general overview of hierarchical reinforcement learning in the context of cognitive science and neuroscience. In Chapter 3, I present a generalization of SR, called the Deep Successor Reinforcement Learning, which combines deep neural networks with SR to learn robust value functions.

I demonstrate that under a random policy, we can learn the SR representation and extract subgoals by using the normalized cut algorithm [122]. Random walk in the the environment induces a graph structure of states clustered with respect to their corresponding state abstractions. The bottleneck states connecting the cluster pair can be thought of as plausible subgoal candidates. One of the earliest work in automatic subgoal discovery is presented in *csimcsek2005identifying*. In this work, a collection of state trajectories were considered and subgoals were extracted using the normalized cuts algorithm [122].

1.4 Summary of Contributions

My objective was to explore two main questions: (1) How does an agent build compact descriptions of visual scenes from experiences, and (2) how can it effectively use and fine tune these representations, along with learning temporal abstractions for efficient goal driven exploration to produce behavior? My contributions can be summarized as follows:

1.4.1 Integrating Perception and Control

- **Deep Hierarchical Reinforcement Learning with Intrinsic Motivation:** Learning goal-directed behavior in environments with sparse feedback is a major challenge for reinforcement learning algorithms. The primary difficulty arises due to insufficient exploration, resulting in an agent being unable to learn robust value functions. Intrinsically motivated agents can explore new behavior for its own sake rather than to directly solve problems. Such intrinsic behaviors could eventually help the agent solve tasks posed by the environment.

  I present hierarchical-DQN (h-DQN), a framework to integrate hierarchical value functions, operating at different temporal scales, with competence based intrinsically motivated deep reinforcement learning. A top-level value function learns a policy over intrinsic goals, and a lower-level function learns a policy over atomic actions to satisfy the given goals. h-DQN allows for flexible goal specifications, such as functions over entities and relations. This provides an efficient space for exploration in complicated environments.

  Paper: Kulkarni, T.D., Narasimhan, K.R., Saeedi, A. and Tenenbaum, J.B., 2016. *Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation.* arXiv preprint arXiv:1604.06057.

- **Deep Successor Reinforcement Learning:** Learning robust value functions given raw observations and rewards is possible with model-free and model-based deep reinforcement learning algorithms. There is a third alternative, called Successor Representations (SR), which decomposes the value function into two components – a reward predictor and a successor map. The successor map represents the expected future state occupancy from any given state and the reward predictor maps states to scalar rewards. The value function of a state can be computed as the inner product between the successor map and the reward weights. This disentanglement of the value function separates knowledge about the world from that of goals, which is a crucial property for transfer learning of policies.

I present DSR, which generalizes SR within an end-to-end deep reinforcement learning framework. DSR has several appealing properties including: increased sensitivity to distal reward changes due to factorization of reward and world dynamics, and the ability to extract bottleneck states (subgoals) given successor maps trained under a random policy.

Paper: *Deep Successor Reinforcement Learning.* Under Submission. NIPS 2016.

1.4.2 Towards more structured models of perception

The papers that I published relevant to this section are – [78, 82, 79, 81, 151, 91].

- **Unified framework for analysis-by-synthesis:**
  I shall lay out a conceptual and theoretical framework to cast the problem of vision as the inverse of computer graphics. Inverse graphics is a conceptual framework for analysis-by-synthesis where graphics simulations define the generative process. I shall lay out the foundations for defining the scene model, approximate rendering, representation layers for comparing hypotheses with observations and a general-purpose inference layer for interpretation.

- **Design and implementation of a probabilistic programming language for scene perception:** I design and experiment with a general-purpose probabilistic language for visual scene perception. This language unifies the process of expressing generative vision models, probabilistic inference techniques, and data-driven techniques for scaling up inference. I demonstrate this system on a wide variety of tasks including: 3D face analysis, 3D body pose estimation, 3D shape perception, solving CAPTCHA's, 3D road-lane estimation, etc.

- **General-purpose inference algorithms:** Due to the high-dimensional latent space of typical probabilistic programs, inference is often intractable and requires task specific updates for reliable mixing and convergence. I develop a host of probabilistic inference algorithms to handle mixed discrete-continuous latent variables. I bridge the gap between neural networks and probabilistic programs by using neural networks as a general-purpose scheme to amortize inference. I also explore particle based techniques for variational inference to handle generative processes with discrete latent variables.

- **Learning structured deep generative models:** Deep Neural Networks have been successful at learning a mapping between input and target labels. However, there has been relatively less success in trying to learn structured generative models with neural networks. Moreover, the learnt representations of a neural networks are often not interpretable and disentangled with respect to factors of variations in the data. I present neural network architectures which learn disentangled and interpretable generative models form image sequences. I demonstrate the system for learning 3D generative models of faces and chairs.

Chapter 2

Deep Hierarchical Reinforcement Learning

2.1 Introduction

Learning goal-directed behavior with sparse feedback from complex environments is a fundamental challenge for artificial intelligence. Learning in this setting requires the agent to represent knowledge at multiple levels of spatio-temporal abstractions and to explore the environment efficiently. Recently, non-linear function approximators coupled with reinforcement learning [75, 94, 123] have made it possible to learn abstractions over high-dimensional state spaces, but the task of exploration with sparse feedback still remains a major challenge. Existing methods like Boltzmann exploration and Thomson sampling [131, 102] offer significant improvements over ϵ-greedy, but are limited due to the underlying models functioning at the level of basic actions. In this work, we propose a framework that integrates deep reinforcement learning with hierarchical value functions (h-DQN), where the agent is motivated to solve intrinsic goals (via learning options) to aid exploration. These goals provide for efficient exploration and help mitigate the sparse feedback problem. Additionally, we observe that goals defined in the space of entities and relations can help significantly constrain the exploration space for data-efficient learning in complex environments.

Reinforcement learning (RL) formalizes control problems as finding a policy π

that maximizes expected future rewards [134]. Value functions $V(s)$ are central to RL, and they cache the utility of any state s in achieving the agent's overall objective. Recently, value functions have also been generalized as $V(s, g)$ in order to represent the utility of state s for achieving a given goal $g \in G$ [135, 115]. When the environment provides delayed rewards, we adopt a strategy to first learn ways to achieve intrinsically generated goals, and subsequently learn an optimal policy to chain them together.

Each of the value functions $V(s, g)$ can be used to generate a policy that terminates when the agent reaches the goal state g. A collection of these policies can be hierarchically arranged with temporal dynamics for learning or planning within the framework of semi-Markov decision processes [136, 138]. In high-dimensional problems, these value functions can be approximated by neural networks as $V(s, g; \theta)$.

We propose a framework with hierarchically organized deep reinforcement learning modules working at different time-scales. The model takes decisions over two levels of hierarchy – (a) the top level module (*meta-controller*) takes in the state and picks a new goal, (b) the lower-level module (*controller*) uses both the state and the chosen goal to select actions either until the goal is reached or the episode is terminated. The *meta-controller* then chooses another goal and steps (a-b) repeat. We train our model using stochastic gradient descent at different temporal scales to optimize expected future intrinsic (*controller*) and extrinsic rewards (*meta-controller*). We demonstrate the strength of our approach on problems with long-range delayed feedback: (1) a discrete stochastic decision process with a long chain of states before receiving optimal extrinsic rewards and (2) a classic ATARI game ('Montezuma's Revenge') with even longer-range delayed rewards where most existing state-of-art deep reinforcement learning approaches fail to

2.2 Model

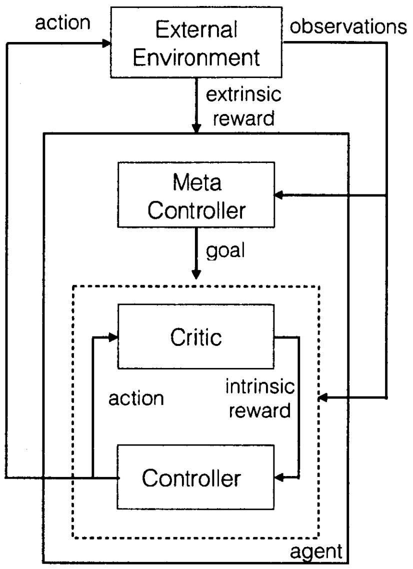
Consider a Markov decision process (MDP) represented by states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and transition function $\mathcal{T} : (s, a) \rightarrow s'$. An agent operating in this framework receives a state s from the external environment and can take an action a, which results in a new state s'. We define the extrinsic reward function as $\mathcal{F} : (s) \rightarrow \mathbb{R}$. The objective of the agent is to maximize this function over long periods of time. For example, this function can take the form of the agent's survival time or score in a game.

Agents Effective exploration in MDPs is a significant challenge in learning good control policies. Methods such as ϵ-greedy are useful for local exploration but fail to provide impetus for the agent to explore different areas of the state space. In order to tackle this, we utilize a notion of *goals* $g \in \mathcal{G}$, which provide intrinsic motivation for the agent. The agent focuses on setting and achieving sequences of goals in order to maximize cumulative extrinsic reward.

We use the temporal abstraction of *options* [136] to define policies π_g for each goal g. The agent learns these option policies simultaneously along with learning the optimal sequence of goals to follow. In order to learn each π_g, the agent also has a critic, which provides *intrinsic rewards*, based on whether the agent is able to achieve its goals (see Figure 5-1).

Temporal Abstractions As shown in Figure 5-1, the agent uses a two-stage hierarchy consisting of a *controller* and a *meta-controller*. The meta-controller receives state s_t and chooses a goal $g_t \in \mathcal{G}$, where $\mathcal{G}$ denotes the set of all possible current goals. The controller then selects an action a_t using s_t and g_t. The goal g_t remains in place for the next few time steps either until it is achieved or a terminal state is reached. The internal critic is responsible for evaluating whether a goal has been reached and providing an appropriate reward $r_t(g)$ to the controller. The objective function for the controller is to maximize cumulative intrinsic reward: $R_t(g) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}(g)$. Similarly, the objective of the meta-controller is to optimize the cumulative extrinsic reward $F_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} f_{t'}$, where f_t are reward signals received from the environment.

One can also view this setup as similar to optimizing over the space of optimal reward functions to maximize fitness [125]. In our case, the reward functions are dynamic and temporally dependent on the sequential history of goals. Figure 5-1 provides an illustration of the agent's use of the hierarchy over subsequent time steps.



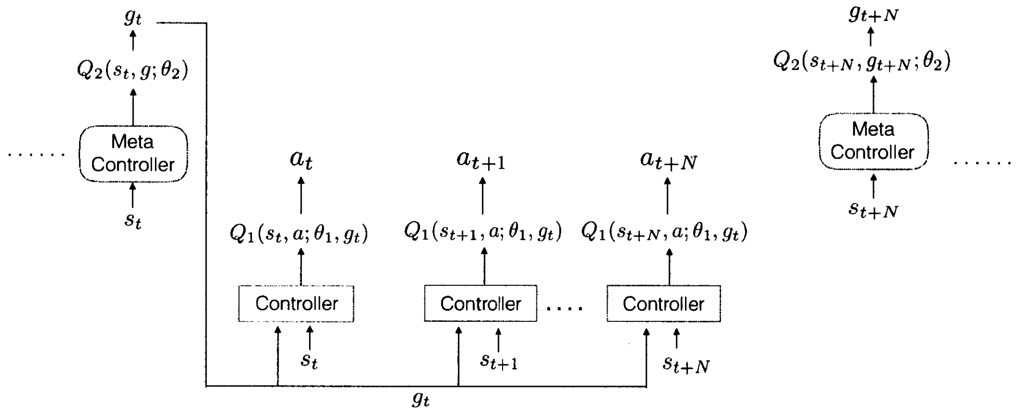


Figure 2-1: **Overview:** The agent produces actions and receives sensory observations. Separate deep-Q networks are used inside the *meta-controller* and *controller*. The meta-controller that looks at the raw states and produces a policy over goals by estimating the value function $Q_2(s_t, g_t; \theta_2)$ (by maximizing expected future extrinsic reward). The controller takes in states and the current goal, and produces a policy over actions by estimating the value function $Q_2(s_t, a_t; \theta_1, g_t)$ to solve the predicted goal (by maximizing expected future intrinsic reward). The internal critic checks if goal is reached and provides an appropriate intrinsic reward to the controller. The controller terminates either when the episode ends or when g is accomplished. The meta-controller then chooses a new g and the process repeats.

Deep Reinforcement Learning with Temporal Abstractions

We use the Deep Q-Learning framework [94] to learn policies for both the controller and the meta-controller. Specifically, the controller estimates the following Q-value function:

$$\begin{aligned} Q_1^*(s,a;g) &= \max_{\pi_{ag}} \mathrm{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \mid s_t = s, a_t = a, g_t = g, \pi_{ag}] \\ &= \max_{\pi_{ag}} \mathrm{E}[r_t + \gamma \ \max_{a_{t+1}} Q_1^*(s_{t+1}, a_{t+1}; g) \mid s_t = s, a_t = a, g_t = g, \pi_{ag}] \end{aligned} \tag{2.1}$$

where g is the agent's goal in state s and $\pi_{ag} = P(a|s,g)$ is the action policy.

Similarly, for the meta-controller, we have:

$$Q_2^*(s,g) = \max_{\pi_g} \mathrm{E}[\sum_{t'=t}^{t+N} f_{t'} + \gamma \ \max_{g'} Q_2^*(s_{t+N}, g') \mid s_t = s, g_t = g, \pi_g] \tag{2.2}$$

where N denotes the number of time steps until the controller halts given the current goal, g' is the agent's goal in state s_{t+N}, and $\pi_g = P(g|s)$ is the policy over goals. It is important to note that the transitions (s_t, g_t, f_t, s_{t+N}) generated by Q_2 run at a slower time-scale than the transitions $(s_t, a_t, g_t, r_t, s_{t+1})$ generated by Q_1.

We can represent $Q^*(s,g) \approx Q(s,g;\theta)$ using a non-linear function approximator with parameters θ, called a deep Q-network (DQN). Each $Q \in \{Q_1, Q_2\}$ can be trained by minimizing corresponding loss functions – $L_1(\theta_1)$ and $L_2(\theta_2)$. We store experiences (s_t, g_t, f_t, s_{t+N}) for Q_2 and $(s_t, a_t, g_t, r_t, s_{t+1})$ for Q_1 in disjoint memory spaces $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. The loss function for Q_1 can then be stated as:

$$L_1(\theta_{1,i}) = \mathrm{E}_{(s,a,g,r,s') \sim D_1}[(y_{1,i} - Q_1(s,a;\theta_{1,i},g))^2], \tag{2.3}$$

where i denotes the training iteration number and $y_{1,i} = r + \gamma \ \max_{a'} Q_1(s', a'; \theta_{1,i-1}, g)$.

Following [94], the parameters $\theta_{1,i-1}$ from the previous iteration are held fixed when optimising the loss function. The parameters θ_1 can be optimized using the gradient:

$$\nabla_{\theta_{1,i}} L_1(\theta_{1,i})$$

$$= \mathrm{E}_{(s,a,r,s' \sim D_1)} \left[\left(r + \gamma \ \max_{a'} Q_1(s', a'; \theta_{1,i-1}, g) - Q_1(s, a; \theta_{1,i}, g) \right) \nabla_{\theta_{1,i}} Q_1(s, a; \theta_{1,i}, g) \right) \right]$$

The loss function L_2 and its gradients can be derived using a similar procedure.

Learning Algorithm We learn the parameters of h-DQN using stochastic gradient descent at different time scales – experiences (or transitions) from the controller are collected at every time step but experiences from meta-controller are only collected when the controller terminates (i.e. when a goal is re-picked or the episode ends). Each new goal g is drawn in an ϵ-greedy fashion (Algorithms 4 & 2) with the exploration probability ϵ_2 annealed as learning proceeds (from a starting value of 1).

In the controller, at every time step, an action is drawn with a goal using the exploration probability $\epsilon_{1,g}$ which is dependent on the current empirical success rate of reaching g. The model parameters (θ_1, θ_2) are periodically updated by drawing experiences from replay memories $\mathcal{D}_1$ and $\mathcal{D}_2$), respectively (see Algorithm 3).

2.3 Experiments

We perform experiments on two different domains involving delayed rewards. The first is a discrete-state stochastic decision process with stochastic transitions, and the second is an ATARI 2600 game called 'Montezuma's Revenge'.

2.3.1 Stochastic Decision Process with Delayed Rewards

Game Setup We consider a stochastic decision process where the extrinsic reward depends on the history of visited states in addition to the current state. We selected this task in order to demonstrate the importance of intrinsic motivation for exploration in such environments.

There are 6 possible states and the agent always starts at s_2. The agent moves left deterministically when it chooses *left* action; but the action *right* only succeeds

Algorithm 1 Learning algorithm for h-DQN

```
1: Initialize experience replay memories $\{\mathcal{D}_1, \mathcal{D}_2\}$ and parameters $\{\theta_1, \theta_2\}$ for the controller and meta-controller respectively.
2: Initialize exploration probability $\epsilon_{1,g} = 1$ for the controller for all goals $g$ and $\epsilon_2 = 1$ for the meta-controller.
3: for $i = 1, num\_episodes$ do
4:     Initialize game and get start state description $s$
5:     $g \leftarrow$ EPSGREEDY$(s, \mathcal{G}, \epsilon_2, Q_2)$
6:     while $s$ is not terminal do
7:         $F \leftarrow 0$
8:         $s_0 \leftarrow s$
9:         while not ($s$ is terminal or goal $g$ reached) do
10:            $a \leftarrow$ EPSGREEDY$(\{s, g\}, \mathcal{A}, \epsilon_{1,g}, Q_1)$
11:            Execute $a$ and obtain next state $s'$ and extrinsic reward $f$ from environment
12:            Obtain intrinsic reward $r(s, a, s')$ from internal critic
13:            Store transition $(\{s, g\}, a, r, \{s', g\})$ in $\mathcal{D}_1$
14:            UPDATEPARAMS$(\mathcal{L}_1(\theta_{1,i}), \mathcal{D}_1)$
15:            UPDATEPARAMS$(\mathcal{L}_2(\theta_{2,i}), \mathcal{D}_2)$
16:            $F \leftarrow F + f$
17:            $s \leftarrow s'$
18:        end while
19:        Store transition $(s_0, g, F, s')$ in $\mathcal{D}_2$
20:        if $s$ is not terminal then
21:            $g \leftarrow$ EPSGREEDY$(s, \mathcal{G}, \epsilon_2, Q_2)$
22:        end if
23:    end while
24:    Anneal $\epsilon_2$ and adaptively anneal $\epsilon_{1,g}$ using average success rate of reaching goal $g$.
25: end for
```

Algorithm 2 : EPSGREEDY$(x, \mathcal{B}, \epsilon, Q)$

```
1: if random() < $\epsilon$ then
2:     return random element from set $\mathcal{B}$
3: else
4:     return $\text{argmax}_{m \in \mathcal{B}} Q(x, m)$
5: end if
```

Algorithm 3 : UPDATEPARAMS$(\mathcal{L}, \mathcal{D})$

```
1: Randomly sample mini-batches from $\mathcal{D}$
2: Perform gradient descent on loss $\mathcal{L}(\theta)$ (cf. (2.2))
```

50% of the time, resulting in a left move otherwise. The terminal state is s_1 and the agent receives the reward of 1 when it first visits s_6 and then s_1. The reward for going to s_1 without visiting s_6 is 0.01. This is a modified version of the MDP in [102], with the reward structure adding complexity to the task. The process is illustrated in Figure 2-2.

We consider each state as a possible goal for exploration. This encourages the agent to visit state s_6 (whenever it is chosen as a goal) and hence, learn the optimal policy. For each goal, the agent receives a positive intrinsic reward if and only if it reaches the corresponding state.

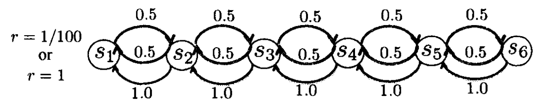


Figure 2-2: A stochastic decision process where the reward at the terminal state s_1 depends on whether s_6 is visited ($r = 1$) or not ($r = 1/100$).

Results We compare the performance of our approach (without the deep neural networks) with Q-Learning as a baseline (without intrinsic rewards) in terms of the average extrinsic reward gained in an episode. In our experiments, all ϵ parameters are annealed from 1 to 0.1 over 50,000 steps. The learning rate is set to 0.00025. Figure 2-3 plots the evolution of reward for both methods averaged over 10 different runs. As expected, we see that Q-Learning is unable to find the optimal policy even after 200 epochs, converging to a sub-optimal policy of reaching state s_1 directly to obtain a reward of 0.01. In contrast, our approach with hierarchical Q-estimators learns to choose goals s_4, s_5 or s_6, which statistically lead the agent to visit s_6 before going back to s_1. Therefore, the agent obtains a significantly higher average reward of around 0.13.

Figure 2-4 illustrates that the number of visits to states s_3, s_4, s_5, s_6 increases with episodes of training. Each data point shows the average number of visits for each state over the last 1000 episodes. This indicates that our model is choosing goals in a way so that it reaches the critical state s_6 more often.

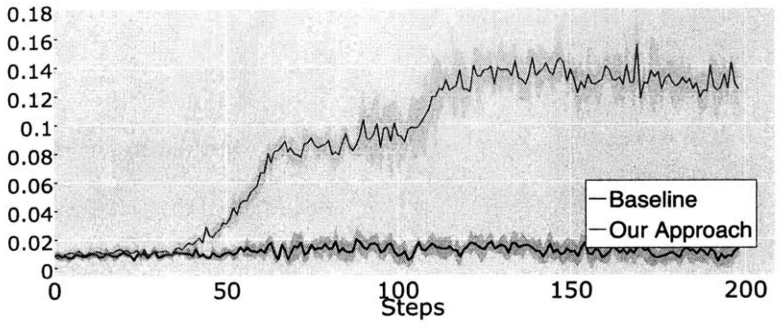


Figure 2-3: Average reward for 10 runs of our approach compared to Q-learning.

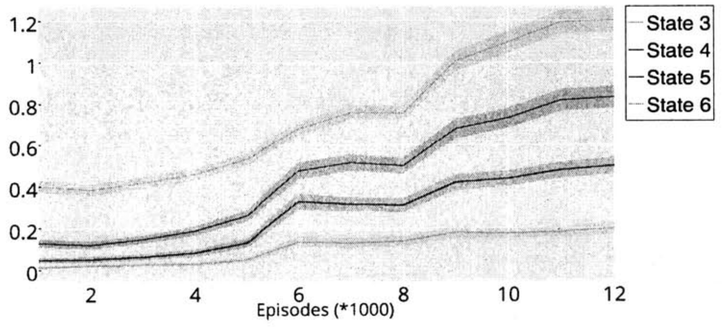
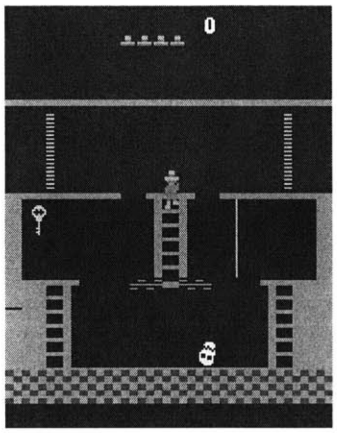


Figure 2-4: Number of visits (for states s_3 to s_6) averaged over 1000 episodes. The initial state is s_2 and the terminal state is s_1.

2.3.2 ATARI Game with Delayed Rewards

Game Description We consider 'Montezuma's Revenge', an ATARI game with sparse, delayed rewards. The game (Figure 2-5(a)) requires the player to navigate the explorer (in red) through several rooms while collecting treasures. In order to pass through doors (in the top right and top left corners of the figure), the player has to first pick up the key. The player has to then climb down the ladders on the right and move left towards the key, resulting in a long sequence of actions before receiving a reward (+100) for collecting the key. After this, navigating towards the door and opening it results in another reward (+300).

Existing deep RL approaches fail to learn in this environment since the agent rarely reaches a state with non-zero reward. For instance, the basic DQN [94] achieves a score of 0 while even the best performing system, Gorila DQN [98], manages only 4.16 on average.

(a) 

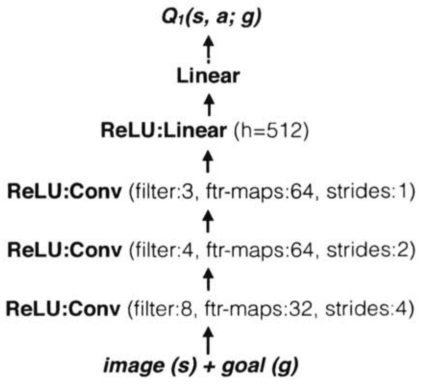
(b) 
Q1(s, a; g)
Linear
ReLU:Linear (h=512)
ReLU:Conv (filter:3, ftr-maps:64, strides:1)
ReLU:Conv (filter:4, ftr-maps:64, strides:2)
ReLU:Conv (filter:8, ftr-maps:32, strides:4)
image (s) + goal (g)

Figure 2-5: (a) A sample screen from the ATARI 2600 game called 'Montezuma's Revenge'. (b) **Architecture**: DQN architecture for the controller (Q_1). A similar architecture produces Q_2 for the meta-controller (without goal as input). In practice, both these networks could share lower level features but we do not enforce this.

Setup The agent needs intrinsic motivation to explore meaningful parts of the scene before it can learn about the advantage of getting the key for itself. Inspired by the developmental psychology literature [129] and object-oriented MDPs [28], we use entities or objects in the scene to parameterize goals in this environment. Unsupervised detection of objects in visual scenes is an open problem in computer vision, although there has been recent progress in obtaining objects directly from image or motion data [34, 31, 42]. In this work, we built a custom object detector that provides plausible object candidates. The controller and meta-controller are convolutional neural networks (see Figure 2-5(b)) that learn representations from raw pixel data. We use the Arcade Learning Environment [7] to perform experiments.

The internal critic is defined in the space of $\langle entity_1, relation, entity_2\rangle$, where $relation$ is a function over configurations of the entities. In our experiments, the agent is free to choose any $entity_2$. For instance, the agent is deemed to have completed a goal (and receives a reward) if the agent entity *reaches* another entity such as the *door*. Note that this notion of relational intrinsic rewards can be generalized to other settings. For instance, in the ATARI game 'Asteroids', the agent could be rewarded when the bullet **reaches** the asteroid or if simply the ship never **reaches** an asteroid. In the game of 'Pacman', the agent could be rewarded if the pellets on the screen are **reached**. In the most general case, we can potentially let the model evolve a parameterized intrinsic reward function given entities. We leave this for future work.

Model Architecture and Training As shown in Figure 2-5b, the model consists of stacked convolutional layers with rectified linear units (ReLU). The input to the meta-controller is a set of four consecutive images of size 84×84. To encode the goal output from the meta-controller, we append a binary mask of the goal location in image space along with the original 4 consecutive frames. This augmented input is passed to the controller. The experience replay memories $\mathcal{D}_1$ and $\mathcal{D}_2$ were set to be equal to 1E6 and 5E4 respectively. We set the learning rate to be 2.5E−4, with a discount rate of 0.99. We follow a two phase training procedure – (1) In the first phase, we set the exploration parameter ϵ_2 of the meta-controller to 1 and train the controller on actions. This effectively leads to pre-training the controller so that it can learn to solve a subset of the goals. (2) In the second phase, we jointly train the controller and meta-controller.

Results Figure 2-6(a) shows reward progress from the joint training phase from which it is evident that the model starts gradually learning to both reach the key and open the door to get a reward of around +400 per episode. As shown in Figure 2-6(b), the agent learns to choose the key more often as training proceeds and is also successful at reaching it. As training proceeds, we observe that the agent first learns to perform the simpler goals (such as reaching the right door or the middle ladder) and then slowly starts learning the 'harder' goals such as the key and the bottom ladders, which provide a path to higher rewards. Figure 2-6(c) shows the evolution of the success rate of goals that are picked. At the end of training, we can see that the 'key', 'bottom-left-ladder' and 'bottom-right-ladders' are chosen increasingly more often. In order to scale-up to solve the entire game, several key ingredients are missing such as – automatic discovery of objects from videos to aid goal parametrization we considered, a flexible short-term memory, ability to intermittently terminate ongoing options.

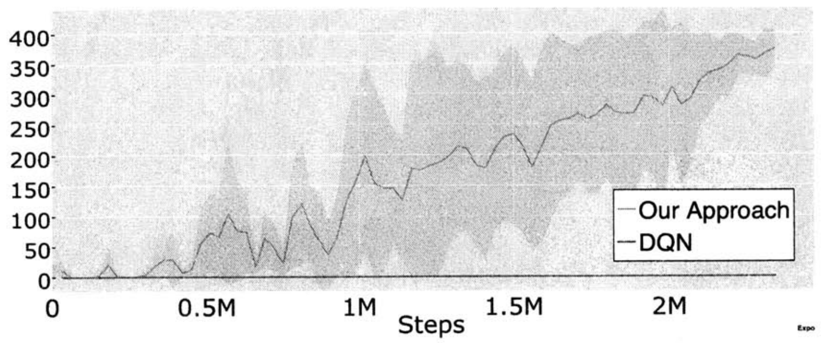
We also show some screen-shots from a test run with our agent (with epsilon set to 0.1) in Figure 2-7, as well as a sample animation of the run.[1]

2.4 Conclusion

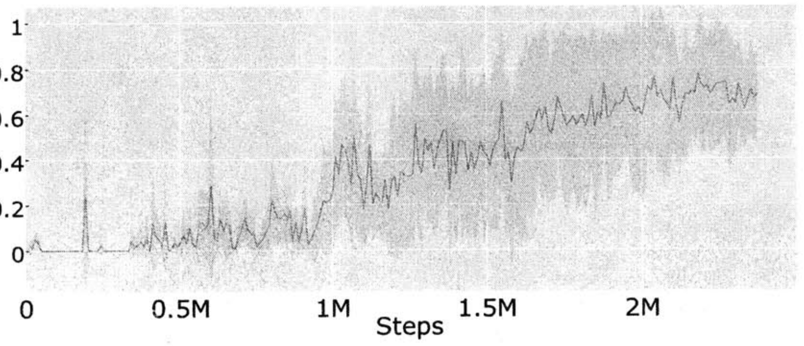
We have presented h-DQN, a framework consisting of hierarchical value functions operating at different time scales. Temporally decomposing the value function allows the agent to perform intrinsically motivated behavior, which in turn yields efficient exploration in environments with delayed rewards. We also observe that parameterizing intrinsic motivation in the space of entities and relations provides a promising avenue for building agents with temporally extended exploration. We also plan to explore alternative parameterizations of goals with h-DQN in the future.

The current framework has several missing components including automatically disentangling objects from raw pixels and a short-term memory. The state abstractions learnt by vanilla deep-Q-networks are not structured or sufficiently compositional. There has been recent work [31, 42, 111, 82, 151, 43, 61] in using deep generative models to disentangle multiple factors of variations (objects, pose, location, etc) from pixel data. We hope that our work motivates the combination of deep generative models of images with h-DQN. Additionally, in order to handle longer range dependencies, the agent needs to store a history of previous goals, actions and representations. There has been some recent work in using recurrent networks in conjunction with reinforcement learning [50, 99]. In order to scale-up our approach to harder non-Markovian settings, it will be necessary to incorporate a flexible episodic memory module.

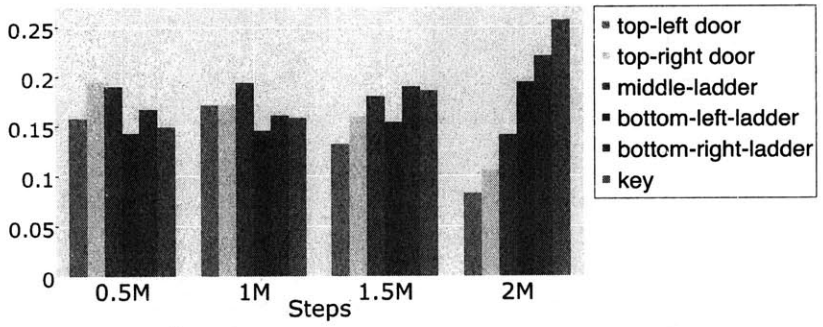
[1]Sample trajectory of a run on 'Montezuma's Revenge' – `https://goo.gl/3Z64Ji`



(a) Total extrinsic reward



(b) Success ratio for reaching the goal 'key'



(c) Success % of different goals over time

Figure 2-6: **Results on Montezuma's Revenge:** These plots depict the joint training phase of the model. As described in Section 2.3.2, the first training phase pre-trains the lower level controller for about 2.3 million steps. The joint training learns to consistently get high rewards after additional 2 million steps as shown in **(a)**. **(b) Goal success ratio:** The agent learns to choose the key more often as training proceeds and is successful at achieving it. **(c) Goal statistics:** During early phases of joint training, all goals are equally preferred due to high exploration but as training proceeds, the agent learns to select appropriate goals such as the key and bottom-left door.

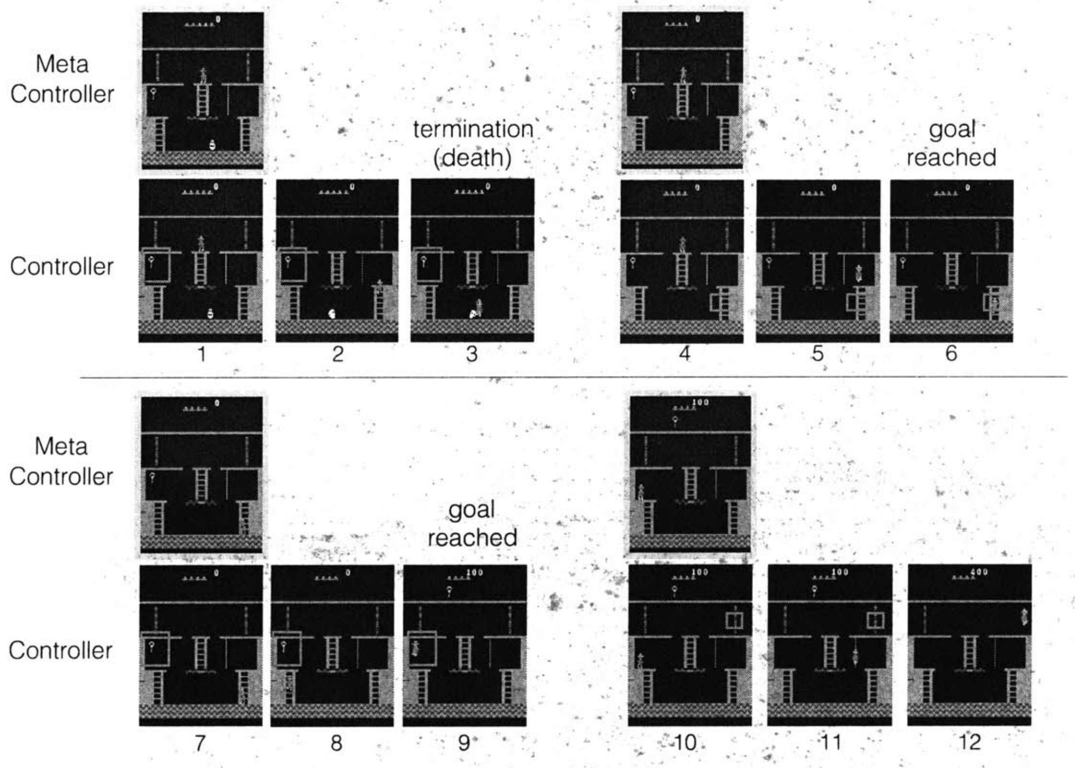


Figure 2-7: **Sample gameplay by our agent on Montezuma's Revenge:** The four quadrants are arranged in a temporally coherent manner (top-left, top-right, bottom-left and bottom-right). At the very beginning, the meta-controller chooses key as the goal (illustrated in *red*). The controller then tries to satisfy this goal by taking a series of low level actions (only a subset shown) but fails due to colliding with the skull (the episode terminates here). The meta-controller then chooses the bottom-right ladder as the next goal and the controller terminates after reaching it. Subsequently, the meta-controller chooses the key and the top-right door and the controller is able to successfully achieve both these goals.

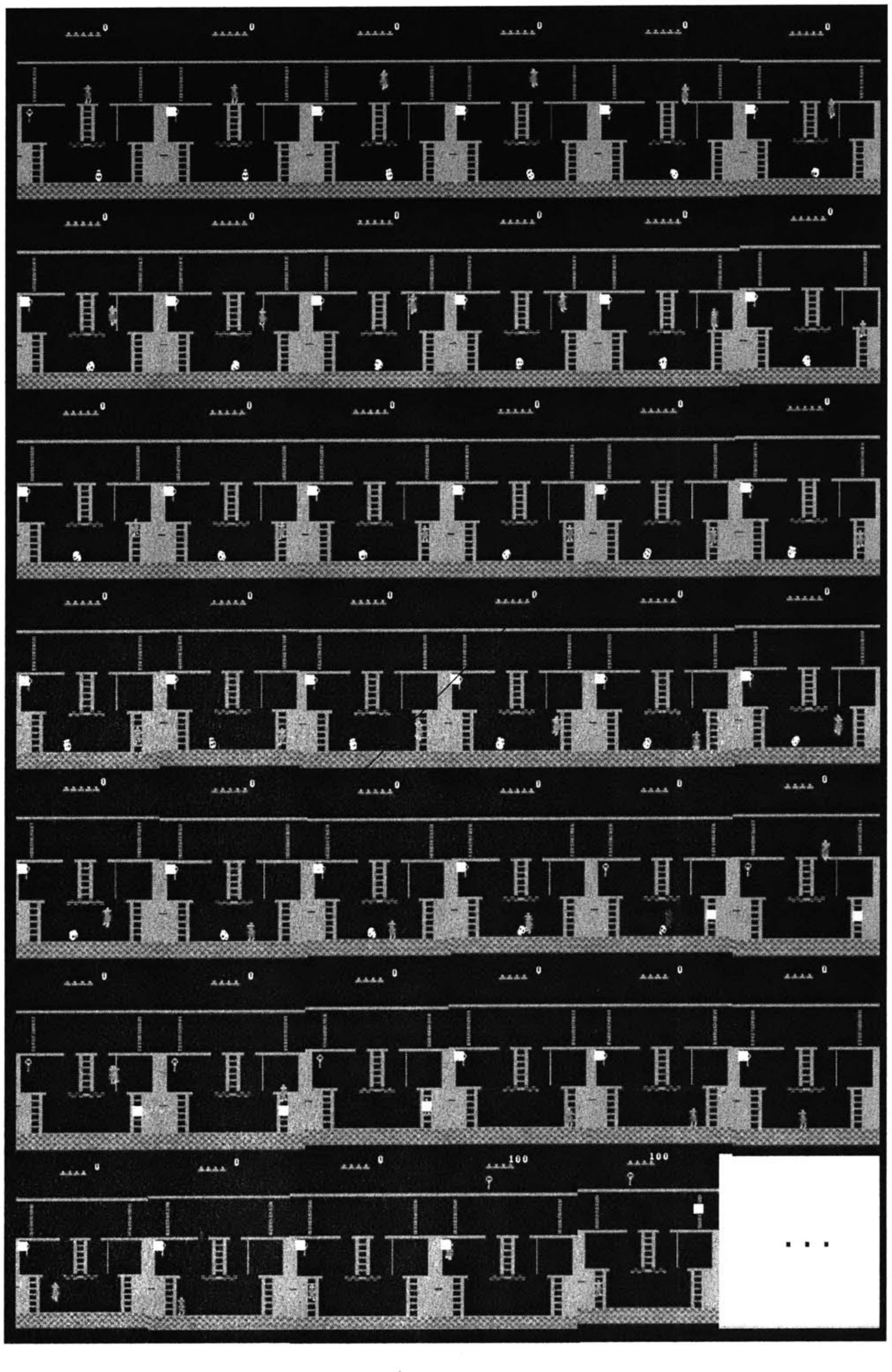


Figure 2-8: Sample gameplay by our agent on Montezuma's Revenge with finer time resolution

Chapter 3

Deep Successor Reinforcement Learning: A Hybrid Model-free and Model-based Approach

3.1 Introduction

Many learning problems involve inferring properties of temporally extended sequences given an objective function. For instance, in reinforcement learning (RL), the task is to find a policy that maximizes expected future discounted rewards (value). RL algorithms fall into two main classes: (1) model-free algorithms that learn cached value functions directly from sample trajectories, and (2) model-based algorithms that estimate transition and reward functions, from which values can be computed using tree-search or dynamic programming. However, there is a third class, based on the *successor representation* (SR), that factors the value function into a predictive representation and a reward function. Specifically, the value function at a state can be expressed as the dot product between the vector of expected discounted future state occupancies and the immediate reward in each of those successor states.

Representing the value function using the SR has several appealing properties. It combines computational efficiency comparable to model-free algorithms with some

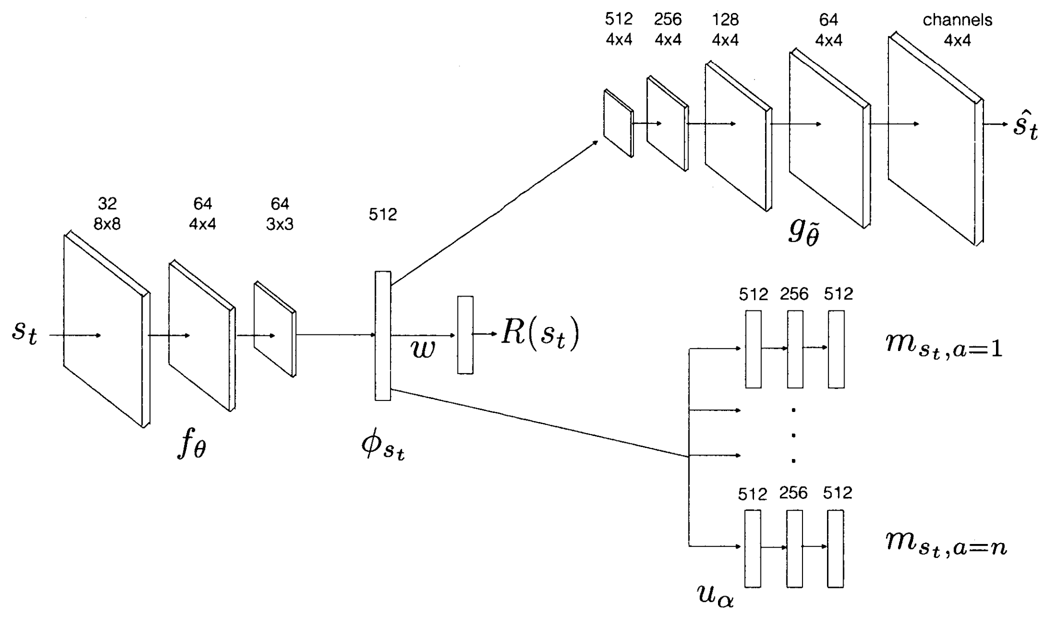


Figure 3-1: **Model Architecture:** DSR consists of: (1) feature branch f_θ (CNN) which takes in raw images and computes the features ϕ_{s_t}, (2) successor branch u_α which computes the SR $m_{s_t,a}$ for each possible action $a \in \mathcal{A}$, (3) a deep convolutional decoder which produces the input reconstruction $\hat{s}_t$ and (4) a linear regressor to predict instantaneous rewards at s_t. The Q-value function can be estimated by taking the inner-product of the SR with reward weights: $Q^\pi(s, a) \approx m_{sa} \cdot \mathbf{w}$.

of the flexibility of model-based algorithms. In particular, the SR can adapt quickly to changes in distal reward, unlike model-free algorithms. In this paper, we also highlight a feature of the SR that has been less well-investigated: the ability to extract bottleneck states (candidate subgoals) from the successor representation under a random policy [130]. These subgoals can then be used within a hierarchical RL framework. In this paper we develop a powerful function approximation algorithm and architecture for the SR using a deep neural network, which we call *Deep Successor Reinforcement Learning* (DSR). This enables learning the SR and reward function from raw sensory observations with end-to-end training.

The DSR consists of two sub-components: (1) a reward feature learning component, constructed as a deep neural network, predicts intrinsic and extrinsic rewards to learn useful features from raw observations; and (2) an SR component, constructed as a separate deep neural network, that estimates the expected future "feature occupancy" conditioned on the current state and averaged over all actions. The value function can then be estimated as the dot product between these two factored representations. We train DSR by sampling experience trajectories (state, next-state, action and reward) from an experience replay memory and apply stochastic gradient descent to optimize model parameters. To avoid instability in the learning algorithm, we interleave training of the successor and reward components.

We show the efficacy of our approach on two different domains: (1) learning to solve goals in grid-world domains using the *MazeBase* game engine and (2) learning to navigate a 3D maze to gather a resource using the *Doom* game engine. We show the empirical convergence results on several policy learning problems as well as sensitivity of the value estimator given distal reward changes. We also demonstrate the possibility of extracting plausible subgoals for hierarchical RL by performing normalized-cuts on the SR [122].

3.2 Related work

The SR has been used in neuroscience as a model for describing different cognitive phenomena. [37] showed that the temporal context model [60], a model of episodic memory, is in fact estimating the SR using the temporal difference algorithm. [18] introduced a model based on SR for preplay and rapid path planning in the CA3 region of the hippocampus. They interpret the SR as an an attractor network in a lowâĂŞdimensional space and show that if the network is stimulated with a goal location it can generate a path to the goal. [130] suggested a model for tying the problems of navigation and reward maximization in the brain. They claimed that the brain's spatial representations are designed to support the reward maximization problem (RL); they showed the behavior of the place cells and grid cells can be explained by finding the optimal spatial representation that can support RL. Based on their model they proposed a way for identifying reasonable subgoals from the spectral features of the SR. Other work (see for instance, [15, 20]) have also discussed utilizing the SR for subgoal and option discovery.

There are also models similar to the SR that have been been applied to other RL-related domains. [118] introduced a model for evaluating the positions in the game of Go; the model is reminiscent of SR as it predicts the fate of every position of the board instead of the overall game score. Another reward-independent model, universal option model (UOM), proposed in [138], uses state occupancy function to build a general model of options. They proved that UOM of an option, given a reward function, can construct a traditional option model.

Our model is also related to the literature on value function approximation using deep neural networks. The deep-Q learning model [94] and its variants [123] have been successful in learning Q-value functions from high-dimensional complex input states.

3.3 Model

3.3.1 Background

Consider an MDP with a set of states $\mathcal{S}$, set of actions $\mathcal{A}$, reward function $R : \mathcal{S} \to \mathbb{R}$, discount factor $\gamma \in [0, 1]$, and a transition distribution $T : \mathcal{S} \times \mathcal{A} \to [0, 1]$. Given a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, the Q-value function for selecting action a in state s is defined as the expected future discounted return:

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, a_0 = a\right], \tag{3.1}$$

where, s_t is the state visited at time t and the expectation is with respect to the policy and transition distribution. The agent's goal is to find the optimal policy Q^* which follows the *Bellman equation*:

$$Q^*(s, a) = R(s_t) + \gamma \max_{a'} \mathbb{E}\left[Q(s_{t+1}, a')\right]. \tag{3.2}$$

3.3.2 The successor representation

The SR can be used for calculating the Q-value function as follows. Given a state s, action a and future states s', SR is defined as the expected discounted future state occupancy:

$$M(s, s', a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[s_t = s'] | s_0 = s, a_0 = a\right],$$

where $\mathbb{1}[.] = 1$ when its argument is true and zero otherwise. This implicitly captures the state visitation count. Similar to the Bellman equation for the Q-value function (Eq. 3.2), we can express the SR in a recursive form:

$$M(s, s', a) = \mathbb{1}[s_t = s'] + \gamma \mathbb{E}[M(s_{t+1}, s', a_{t+1})]. \tag{3.3}$$

Given the SR, the Q-value for selecting action a in state s can be expressed as the inner product of the immediate reward and the SR [21]:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} M(s, s', a) R(s') \tag{3.4}$$

3.3.3 Deep successor representation

For large state spaces, representing and learning the SR can become intractable; hence, we appeal to non-linear function approximation. We represent each state s by a D-dimensional feature vector ϕ_s which is the output of a deep neural network $f_\theta : \mathcal{S} \to \mathbb{R}^D$ parameterized by θ.

For a feature vector ϕ_s, we define a feature-based SR as the expected future occupancy of the features and denote it by m_{sa}. We approximate m_{sa} by another deep neural network u_α parameterized by α: $m_{sa} \approx u_\alpha(\phi_s, a)$. We also approximate the immediate reward for state s as a linear function of the feature vector ϕ_S: $R(s) \approx \phi_s \cdot \mathbf{w}$, where $\mathbf{w} \in \mathbb{R}^D$ is a weight vector. Since reward values can be sparse, we can also train an intrinsic reward predictor $R_i(s) = g_{\tilde{\theta}}(\phi_s)$. A good intrinsic reward channel should give dense feedback signal and provide features that preserve latent factors of variations in the data (e.g. deep generative models that do reconstruction). Putting these two pieces together, the Q-value function can be approximated as (see 3.4 for closed form):

$$Q^{\pi}(s,a) \approx m_{sa} \cdot \mathbf{w}. \tag{3.5}$$

The SR for the optimal policy in the non-linear function approximation case can then be obtained from the following Bellman equation:

$$m_{sa} = \phi_s + \gamma \mathbb{E}\left[m_{s_{t+1}a'}\right] \tag{3.6}$$

where $a' = \text{argmax}_a m_{s_{t+1}a} \cdot \mathbf{w}$.

3.3.4 Learning

The parameters $(\theta, \alpha, \mathbf{w}, \tilde{\theta})$ can be learned online through stochastic gradient descent.

The loss function for α is given by:

$$L_t^m(\alpha, \theta) = \mathbb{E}[(\phi(s_t) + \gamma u_{\alpha_{prev}}(\phi_{s_{t+1}}, a') - u_\alpha(\phi_{s_t}, a))^2],$$

where $a' = \text{argmax}_a u_\alpha(\phi_{s_{t+1}}, a) \cdot \mathbf{w}$ and the parameter α_{prev} denotes a previously cached parameter value, set periodically to α. This is essential for stable Q-learning with function approximations (see [94]).

For learning $\mathbf{w}$, the weights for the reward approximation function, we use the following squared loss function:

$$L_t^r(\mathbf{w}, \theta) = (R(s_t) - \phi_{s_t} \cdot \mathbf{w})^2 \tag{3.7}$$

Parameter θ is used for obtaining the $\phi(s)$, the shared feature representation for both reward prediction and SR approximation. An ideal $\phi(s)$ should be: 1) a good predictor for the immediate reward for that state and 2) a good discriminator for the states. The first condition can be handled by minimizing loss function L_t^r; however, we also need a loss function to help in the second condition. To this end, we use a deep convolutional auto-encoder to reconstruct images under an L2 loss function. This dense feedback signal can be interpreted as an intrinsic reward function. The loss function can be stated as:

$$L_t^a(\tilde{\theta}, \theta) = (g_{\tilde{\theta}}(\phi_{s_t}) - s_t)^2. \tag{3.8}$$

The composite loss function is the sum of the three loss functions given above:

$$L_t(\theta, \alpha, \mathbf{w}, \tilde{\theta}) = L_t^m(\alpha, \theta) + L_t^r(\mathbf{w}, \theta) + L_t^a(\tilde{\theta}, \theta) \tag{3.9}$$

For optimizing Eq. 3.9, with respect to the parameters $(\theta, \alpha, \mathbf{w}, \tilde{\theta})$, we iteratively update α and $(\theta, \mathbf{w}, \tilde{\theta})$. That is, we learn a feature representation by minimizing $L_t^r(\mathbf{w}) + L_t^a(\tilde{\theta})$; then given $(\theta^*, \mathbf{w}^*, \tilde{\theta}^*)$, we find the optimal α^*. This iteration is important to ensure that the successor branch does not back-propagate gradients to affect θ. We use experience replay memory $\mathcal{D}$ of size $1e^6$ to store transitions, and apply stochastic gradient descent with a learning rate of $2.5e^{-4}$, momentum of 0.95, a discount factor of 0.99 and the exploration parameter ϵ annealed from 1 to 0.1 as training progresses. Algorithm 1 highlights the learning algorithm in greater detail.

3.4 Automatic Subgoal Extraction

Learning policies given sparse or delayed rewards is a significant challenge for current reinforcement learning algorithms. This is mainly due to inefficient exploration schemes such as $\epsilon-$greedy. Existing methods like Boltzmann exploration and Thomson sampling [131, 102] offer significant improvements over ϵ-greedy, but are limited due to the underlying models functioning at the level of basic actions. Hierarchical reinforcement learning algorithms [6] such as the *options* framework [138, 136] provide a flexible framework to create temporal abstractions, which will enable exploration at different time-scales. Inspired by previous work in subgoal discovery from state trajectories [124] and the tabular SR [130], we use the learned SR to generate plausible subgoal candidates.

Algorithm 4 Learning algorithm for DSR

```
1: Initialize experience replay memory $\mathcal{D}$, parameters $\{\theta, \alpha, \mathbf{w}, \tilde{\theta}\}$ and exploration probability $\epsilon = 1$.
2: for $i = 1 : \#episodes$ do
3:     Initialize game and get start state description $s$
4:     while not terminal do
5:         $\phi_s = f_\theta(s)$
6:         With probability $\epsilon$, sample a random action $a$, otherwise choose $\max_a U_\alpha(\phi_s, a) \cdot \mathbf{w}$
7:         Execute $a$ and obtain next state $s'$ and reward $R(s')$ from environment
8:         Store transition $(s, a, R(s'), s')$ in $\mathcal{D}$
9:         Randomly sample mini-batches from $\mathcal{D}$
10:        Perform gradient descent on the loss $L^r(\mathbf{w}, \theta) + L^a(\tilde{\theta}, \theta)$ with respect to $\mathbf{w}$, $\theta$ and $\tilde{\theta}$.
11:        Fix $(\theta, \tilde{\theta}, \mathbf{w})$ and perform gradient descent on $L^m(\alpha, \theta)$ with respect to $\alpha$.
12:        $s \leftarrow s'$
13:    end while
14:    Anneal exploration variable $\epsilon$
15: end for
```

Given a random policy π_r ($\epsilon = 1$), we train the DSR until convergence and collect the SR for a large number of states $\mathcal{T} = \{m_{s_1,a_1}, m_{s_2,a_2}, ..., m_{s_n,a_n}\}$. Following [124, 122], we generate an affinity matrix W given $\mathcal{T}$, by applying a radial basis function (with Euclidean distance metric) for each pairwise entry $(m_{s_i,a_i}, m_{s_j,a_j})$ in $\mathcal{T}$ (to generate w_{ij}). Let D be a diagonal matrix with $D(i,i) = \sum_j w_{ij}$. Then as per [122], the second largest eigenvalue of the matrix $D^{-1}(D - W)$ gives an approximation of the minimum normalized cut value of the partition of $\mathcal{T}$. The states that lie on the end-points of the cut are plausible subgoal candidates, as they provide a path between a community of state groups. Given randomly sampled $\mathcal{T}$ from π_r, we can collect statistics of how many times a particular state lies along the cut. We pick the top-k states as the subgoals. Our experiments indicate that it is possible to extract useful subgoals from the DSR.

3.5 Experiments

In this section, we demonstrate the properties of our approach on MazeBase [133], a grid-world environment, and the Doom game engine [71]. In both environments, observations are presented as raw pixels to the agent. In the first experiment we show that our approach is comparable to DQN in two goal-reaching tasks. Next, we investigate the effect of modifying the distal reward on the initial Q-value. Finally, using normalized-cuts, we identify subgoals given the successor representations in the two environments.

3.5.1 Goal-directed Behavior

Solving a maze in MazeBase We learn the optimal policy in the maze shown in Figure 3-2 using the DSR and compare its performance to the DQN [94]. The cost of living or moving over water blocks is -0.5 and the reward value is 1. For this experiment, we set the discount rate to 0.99 and the learning rate to $2.5 \cdot 10^{-4}$. We anneal the ϵ from 1 to 0.1 over 20k steps; furthermore, for training the reward branch, we anneal the number of samples that we use, from 4000 to 1 by a factor of 0.5 after each training episode. For all experiments, we prioritize the reward training by keeping a database of non-zero rewards and sampling randomly from the replay buffer with a 0.8 probability and 0.2 from the database. Figure 3-3 shows the average trajectory (over 5 runs) of the rewards obtained over 100k episodes. As the plot suggests, DSR performs on par with DQN.

Finding a goal in a 3D environment We created a map with 4 rooms using the ViZDoom platform [71]. The map is shown in Figure 3-2. We share the same network architecture as in the case of MazeBase. The agent is spawned inside a room, and can explore any of the other three rooms. The agent gets a per-step penalty of -0.01 and a positive reward of 1.0 after collecting an item from one of the room (highlighted in *red* in Figure3-2). As shown in Figure3-3, the agent is able to successfully navigate the environment to obtain the reward, and is competitive with DQN.

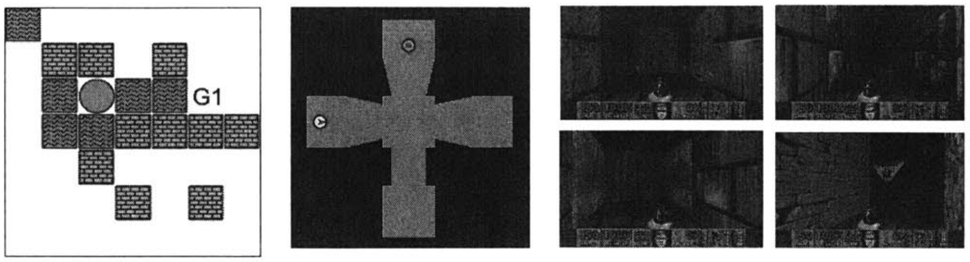


Figure 3-2: **Environments**: **(left)** MazeBase [133] map where the agent starts at an arbitrary location and needs to get to the goal state. The agent gets a penalty of -0.5 per-step, -1 to step on the water-block (blue) and +1 for reaching the goal state. The model observes raw pixel images during learning. **(center)** A *Doom* map using the VizDoom engine [71] where the agent starts in a room and has to get to another room to collect ammo (per-step penalty = -0.01, reward for reaching goal = +1). **(right)** Sample screen-shots of the agent exploring the 3D maze.

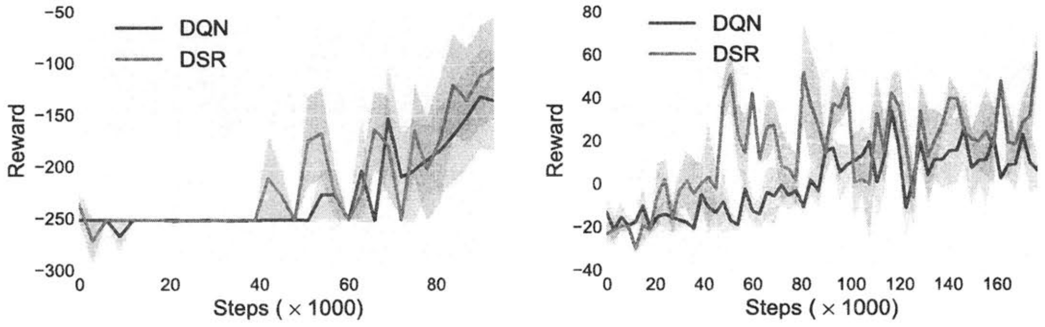


Figure 3-3: Average trajectory of the reward **(left)** over 100k steps for the grid-world maze. **(right)** over 180k steps for the Doom map over multiple runs.

3.5.2 Value function sensitivity to distal reward changes

The decomposition of value function into SR and immediate reward prediction allows DSR to rapidly adapt to changes in the reward function. In order to probe this, we performed experiments to measure the adaptability of the value function to distal reward changes. Given the grid-world map in Figure3-2, we can train the agent to solve the goal specified in the map as highlighted in section 3.5.1. Without changing the goal location, we can change the reward scalar value upon reaching the goal from 1.0 to 3.0. Our hypothesis is that due to the SR-based value decomposition, our value estimate converges to this change by just updating the reward weights **w** (SR remains same). As shown in Figure 3-4, we confirm that the DSR is able to quickly adapt to

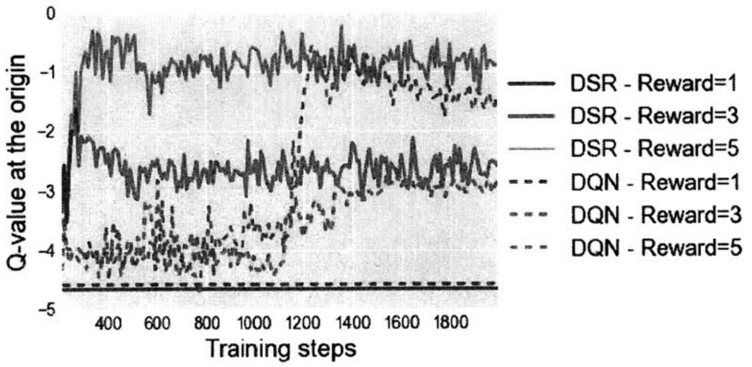


Figure 3-4: **Changing the value of the distal reward:** We train the model to learn the optimal policy on the maze shown in Figure 3-2. After convergence, we change the value of the distal reward and update the Q-value for the optimal action at the origin (bottom-left corner of the maze). In order for the value function to converge again, the model only needs to update the linear weights $\mathbf{w}$ given the new external rewards.

the new value function by just updating $\mathbf{w}$.

3.5.3 Extracting subgoals from the DSR

Following section 3.4, we can also extract subgoals from the SR. We collect $\mathcal{T}$ by running a random policy on both MazeBase and VizDoom. During learning, we only update SR (u_α) and the reconstruction branch ($g_{\tilde{\theta}}$), as the immediate reward at any state is zero (due to random policy).

As shown in Figures 3-5 and 3-6, our subgoal extraction scheme is able to capture useful subgoals and clusters the environment into reasonable segments. Such a scheme can be ran periodically within a hierarchical reinforcement learning framework to aid exploration. One inherent limitation of this approach is that due to the random policy, the subgoal candidates are often quite noisy. Future work should address this limitation and provide statistically robust ways to extract plausible candidates. Additionally, the subgoal extraction algorithm should be non-parametric to handle flexible number of subgoals.

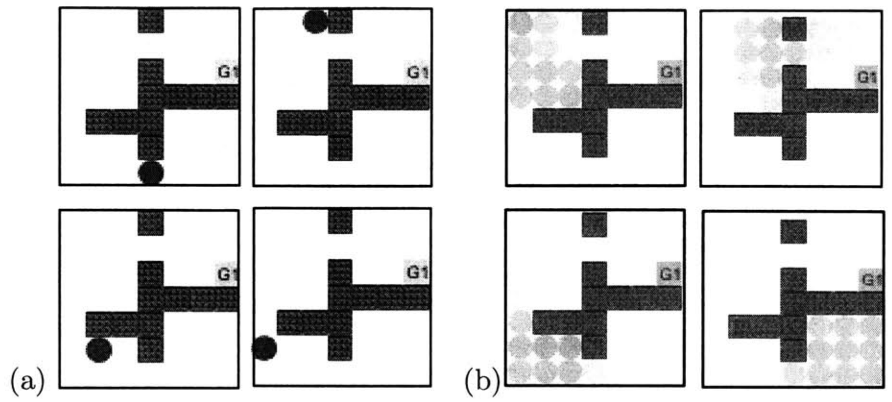


Figure 3-5: **Subgoal extraction on grid-world:** Given a random policy, we train DSR until convergence and collect a large number of sample transitions and their corresponding successor representations as described in section 3.4. We apply a normalized cut-based algorithm on the SRs to obtain a partition of the environment as well as the bottleneck states (which correspond to goals) **(a)** Subgoals are states which separate different partitions of the environments under the normalized-cut algorithm. Our approach is able to find reasonable subgoal candidates. **(b)** Partitions of the environment reflect latent structure in the environment.

3.6 Conclusion

We presented the DSR, a novel deep reinforcement learning framework to learn goal-directed behavior given raw sensory observations. The DSR estimates the value function by taking the inner product between the SR and immediate reward predictions. This factorization of the value function gives rise to several appealing properties over existing deep reinforcement learning methods—namely increased sensitivity of the value function to distal reward changes and the possibility of extracting subgoals from the SR under a random policy.

For future work, we plan to combine the DSR with hierarchical reinforcement learning. Learning goal-directed behavior with sparse rewards is a fundamental challenge for existing reinforcement learning algorithms. The DSR can enable efficient exploration by periodically extracting subgoals, learning policies to satisfy these intrinsic goals (skills), and subsequently learning hierarchical policy over these subgoals in an options framework [138, 80, 115]. One of the major issues with the DSR is learning discriminative features. In order to scale up our approach to more expressive

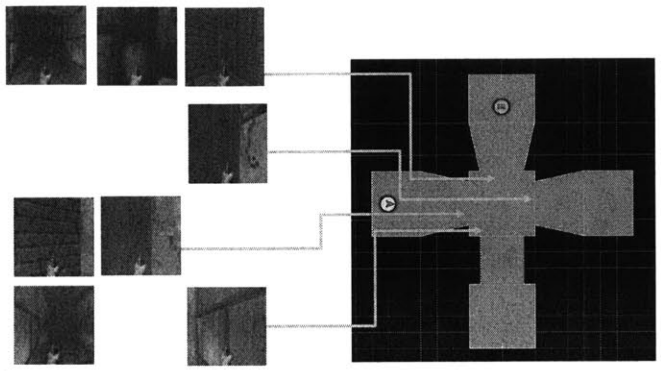


Figure 3-6: **Subgoal extraction on the Doom map** The subgoals are extracted using the normalized cut-based algorithm on the SR. The SR samples are collected based on a random policy. The subgoals mostly correspond to the rooms' entrances in the common area between the rooms. Due to random policy, we sometimes observe high variance in the subgoal quality. Future work should address robust statistical techniques to obtain subgoals, as well as non-parametric approaches to obtaining flexible number of subgoals.

environments, it will be critical to combine state-of-art deep generative models with our approach.

Chapter 4

Perception as analysis-by-synthesis

4.1 Introduction

In this section, I develop model-based techniques for representing visual scenes. So far, I have been effectively using CNNs for representation learning. However, in order to learn features in regimes with much less data and without human labels, it is important to go beyond feature learning and explore causal models of perception.

Probabilistic generative models of perception aim to produce high-probability descriptions of scenes conditioned on observed images or videos, typically either via discriminatively trained models or generative models in an "analysis by synthesis" framework. Discriminative approaches lend themselves to fast, bottom-up inference methods and relatively knowledge-free, data-intensive training regimes, and have been remarkably successful on many recognition problems [32, 76, 84, 90]. Generative approaches hold out the promise of analyzing complex scenes more richly and flexibly [44, 45, 160, 25, 64, 89, 91, 53, 66], but have been less widely embraced for two main reasons: Inference typically depends on slower forms of approximate inference, and both model-building and inference can involve considerable problem-specific engineering to obtain robust and reliable results. These factors make it difficult to develop simple variations on state-of-the-art models, to thoroughly explore the many possible combinations of modeling, representation, and inference strategies, or to richly integrate complementary discriminative and generative modeling approaches to the same problem. More generally, to handle increasingly realistic scenes, generative approaches have to scale not just with respect to data size but also with respect to model and scene complexity. This scaling arguably requires general-purpose frameworks to compose, extend and automatically perform inference in complex structured generative models – tools that for the most part do not yet exist.

Here we present *Picture*, a probabilistic programming language that aims to provide a common representation language and inference engine suitable for a broad class of generative scene perception problems. We see probabilistic programming as key to realizing the promise of "vision as inverse graphics". Generative models can be represented via stochastic code that samples hypothesized scenes and generates images given those scenes. Rich deterministic and stochastic data structures can express complex 3D scenes that are difficult to manually specify. Multiple representation and inference strategies are specifically designed to address the main perceived limitations of generative approaches to vision. Instead of requiring photo-realistic generative models with pixel-level matching to images, we can compare hypothesized scenes to observations using a hierarchy of more abstract image representations such as contours, discriminatively trained part-based skeletons, or deep neural network features. Available Markov Chain Monte Carlo (MCMC) inference algorithms include not only traditional Metropolis-Hastings, but also more advanced techniques for inference in high-dimensional continuous spaces, such as elliptical slice sampling, and Hamiltonian Monte Carlo which can exploit the gradients of automatically differentiable renderers. These top-down inference approaches are integrated with bottom-up and automatically constructed data-driven proposals, which can dramatically accelerate inference by eliminating most of the "burn in" time of traditional samplers and enabling rapid mode-switching.

We demonstrate *Picture* on three challenging vision problems: inferring the 3D shape and detailed appearance of faces, the 3D pose of articulated human bodies, and the 3D shape of medially-symmetric objects. The vast majority of code for image modeling and inference is reusable across these and many other tasks. We shows that *Picture* yields performance competitive with optimized baselines on each of these

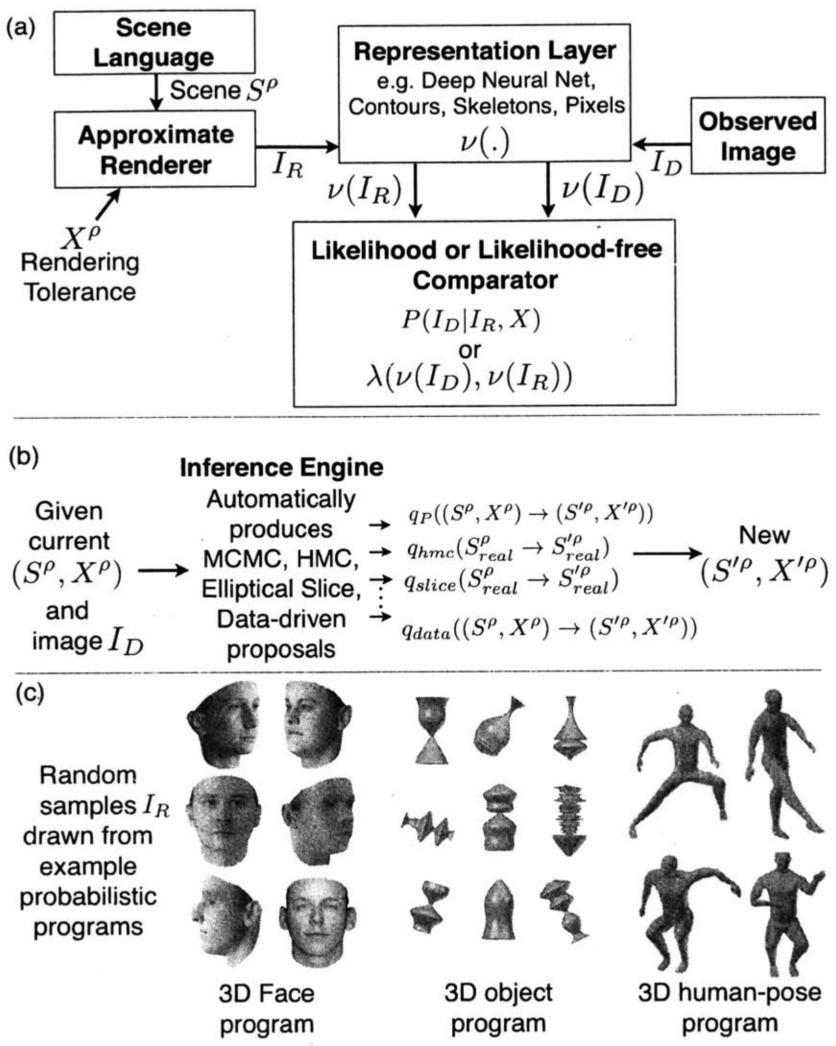

(a)
Scene Language
Scene S^ρ
Approximate Renderer
X^ρ
Rendering Tolerance
I_R
Representation Layer
e.g. Deep Neural Net, Contours, Skeletons, Pixels
$\nu(.)$
Observed Image
I_D
$\nu(I_R)$
$\nu(I_D)$
Likelihood or Likelihood-free Comparator
$P(I_D|I_R, X)$
or
$\lambda(\nu(I_D), \nu(I_R))$
(b)
Given current (S^ρ, X^ρ) and image I_D
Inference Engine
Automatically produces MCMC, HMC, Elliptical Slice, Data-driven proposals
$q_P((S^\rho, X^\rho) \to (S'^\rho, X'^\rho))$
$q_{hmc}(S^\rho_{real} \to S'^\rho_{real})$
$q_{slice}(S^\rho_{real} \to S'^\rho_{real})$
$q_{data}((S^\rho, X^\rho) \to (S'^\rho, X'^\rho))$
New (S'^ρ, X'^ρ)
(c)
Random samples I_R drawn from example probabilistic programs
3D Face program
3D object program
3D human-pose program

Figure 4-1: **Overview: (a)** All models share a common template; only the scene description S and image I_D changes across problems. Every probabilistic graphics program f defines a stochastic procedure that generates both a scene description and all the other information needed to render an approximation I_R of a given observed image I_D. The program f induces a joint probability distribution on these program traces ρ. Every *Picture* program has the following components. **Scene Language:** Describes 2D/3D scenes and generates particular scene related trace variables $S^\rho \in \rho$ during execution. **Approximate Renderer:** Produces graphics rendering I_R given S^ρ and latents X^ρ for controlling the fidelity or tolerance of rendering. **Representation Layer:** Transforms I_D or I_R into a hierarchy of coarse-to-fine image representations $\nu(I_D)$ and $\nu(I_R)$ (deep neural networks [82, 76], contours [29] and pixels). **Comparator:** During inference, I_R and I_D can be compared using a likelihood function or a distance metric λ (as in Approximate Bayesian Computation [152]). **(b) Inference Engine:** Automatically produces a variety of proposals and iteratively evolves the scene hypothesis S to reach a high probability state given I_D. **(c):** Representative random scenes drawn from probabilistic graphics programs for faces, objects, and bodies.

benchmark tasks.

4.2 *Picture* Language

Picture descends from our earlier work on generative probabilistic graphics programming (GPGP) [91], and also incorporates insights for inference from the Helmholtz machine [54, 23] and recent work on differentiable renderers [89] and informed samplers [64]. GPGP aimed to address the main challenges of generative vision by representing visual scenes as short probabilistic programs with random variables, and using a generic MCMC (single-site Metropolis-Hastings) method for inference. However, due to modeling limitations of earlier probabilistic programming languages, and the inefficiency of the Metropolis-Hastings sampler, GPGP was limited to working with low-dimensional scenes, restricted shapes, and low levels of appearance variability. Moreover, it did not support the integration of bottom-up discriminative models such as deep neural networks [76, 82] for data-driven proposal learning. Our current work extends the GPGP framework in all of these directions, letting us tackle a richer set of real-world 3D vision problems.

Picture is an imperative programming language, where expressions can take on either deterministic or stochastic values. We use the transformational compilation technique [154] to implement *Picture*, which is a general method of transforming arbitrary programming languages into probabilistic programming languages. Compared to earlier formulations of GPGP, *Picture* is dynamically compiled at run-time (JIT-compilation) instead of interpreting, making program execution much faster.

A *Picture* program f defines a stochastic procedure that generates both a scene description and all other information needed to render an approximation image I_R for comparison with an observed image I_D. The program f induces a joint probability distribution on the program trace $\rho = \{\rho_i\}$, the set of all random choices i needed to specify the scene hypothesis S and render I_R. Each random choice ρ_i can belong to a familiar parametric or non-parametric family of distributions, such as *Multinomial*, *MvNormal*, *DiscreteUniform*, *Poisson*, or *Gaussian_Process*, but in being used to

```
function PROGRAM(MU, PC, EV, VERTEX_ORDER)
  # Scene Language: Stochastic Scene Gen
  face=Dict();shape = []; texture = [];
  for S in ["shape", "texture"]
   for p in ["nose", "eyes", "outline", "lips"]
     coeff = MvNormal(0,1,1,99)
     face[S][p] = MU[S][p]+PC[S][p].*(coeff.*EV[S][p])
   end
  end
  shape=face["shape"][:]; tex=face["texture"][:];
  camera = Uniform(-1,1,1,2); light = Uniform(-1,1,1,2)

  # Approximate Renderer
  rendered_img= MeshRenderer(shape,tex,light,camera)

  # Representation Layer
  ren_ftrs = getFeatures("CNN_Conv6", rendered_img)

  # Comparator
  #Using Pixel as Summary Statistics
  observe(MvNormal(0,0.01), rendered_img-obs_img)
  #Using CNN last conv layer as Summary Statistics
  observe(MvNormal(0,10), ren_ftrs-obs_cnn)
end

global obs_img = imread("test.png")
global obs_cnn = getFeatures("CNN_Conv6", img)
#Load args from file
TR = trace(PROGRAM,args=[MU,PC,EV,VERTEX_ORDER])
# Data-Driven Learning
learn_datadriven_proposals(TR,100000,"CNN_Conv6")
load_proposals(TR)
# Inference
infer(TR,CB,20,["DATA-DRIVEN"])
infer(TR,CB,200,["ELLIPTICAL"])
```

Figure 4-2: *Picture* code illustration for 3D face analysis: Modules from Figure 4-1a,b are highlighted in **bold**. Running the program unconditionally (by removing *observe's* in code) produces random faces as shown in Figure 4-1c. Running the program conditionally (keeping *observe's*) on I_D results in posterior inference as shown in Figure 4-3. The variables **MU**, **PC**, **EV** correspond to the mean shape/texture face, principal components, and eigenvectors respectively (see [105] for details). These arguments parametrize the prior on the learned shape and appearance of 3D faces. The argument **VERTEX_ORDER** denotes the ordered list of vertices to render triangle based meshes. The *observe* directive constrains the program execution based on both the pixel data and CNN features. The *infer* directive starts the inference engine with the specified set of inference schemes (takes the program trace, a callback function CB for debugging, number of iterations and inference schemes). In this example, data-driven proposals are run for a few iterations to initialize the sampler, followed by slice sampling moves to further refine the high dimensional scene latents.

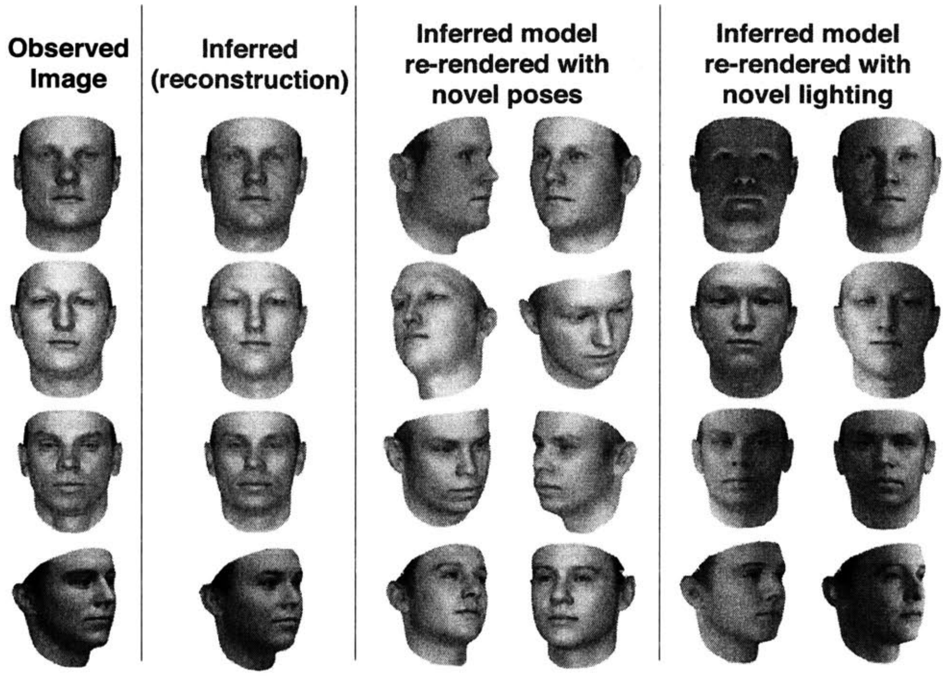

Observed Image
Inferred (reconstruction)
Inferred model re-rendered with novel poses
Inferred model re-rendered with novel lighting

Figure 4-3: **Inference on representative faces using *Picture*:** We tested our approach on a held-out dataset of 2D image projections of laser-scanned faces from [105]. Our short probabilistic program is applicable to non-frontal faces and provides reasonable parses as illustrated above using only general-purpose inference machinery. For quantitative metrics, refer to section 4.4.1.

specify the trace of a probabilistic graphics program, their effects can be combined much more richly than is typical for random variables in traditional statistical models.

Consider running the program in Figure 4-2 unconditionally (without observed data): as different ρ_i's are encountered (for e.g. *coeff*), random values are sampled w.r.t their underlying probability distribution and cached in the current state of the inference engine. Program execution outputs an image of a face with random shape, texture, camera and lighting parameters. Given image data I_D, inference in *Picture* programs amounts to iteratively sampling or evolving program trace ρ to a high probability state while respecting constraints imposed by the data (Figure 4-3). This constrained simulation can be achieved by using the *observe* language construct (see code in Figure 4-2), first proposed in Venture [92] and also used in [104, 155].

4.2.1 Architecture

In this section, we explain the essential architectural components highlighted in Figure 4-1 (see Figure 4-4 for a summary of notation used).

Scene Language: The scene language is used to describe 2D/3D visual scenes as probabilistic code. Visual scenes can be built out of several graphics primitives such as: description of 3D objects in the scene (e.g. mesh, z-map, volumetric), one or more lights, textures, and the camera information. It is important to note that scenes expressed as probabilistic code are more general than parametric prior density functions as is typical in generative vision models. The probabilistic programs we demonstrate in this paper embed ideas from computer-aided design (CAD) and nonparametric Bayesian statistics[109] to express variability in 3D shapes.

Approximate Renderer (AR): *Picture*'s AR layer takes in a scene representation trace S^ρ and tolerance variables X^ρ, and uses general-purpose graphics simulators (Blender[14] and OpenGL) to render 3D scenes. The rendering tolerance X^ρ defines a structured noise process over the rendering and is useful for the following purposes: (a) to make automatic inference more tractable or robust, analogous to simulated annealing (e.g. global or local blur variables in GPGP [91]), and (b) to soak up model mismatch between the true scene rendering I_D and the hypothesized rendering I_R. Inspired by the *differentiable renderer*[89], *Picture* also supports expressing AR's entire graphics pipeline as *Picture* code, enabling the language to express end-to-end differentiable generative models.

Representation Layer (RL): To avoid the need for photo-realistic rendering of complex scenes, which can be slow and modeling-intensive, or for pixel-wise comparison of hypothesized scenes and observed images, which can sometimes yield posteriors that are intractable for sampling-based inference, the RL supports comparison of generated and observed images in terms of a hierarchy of abstract features. The RL can be defined as a function ν which produces summary statistics given I_D or I_R, and may also have internal parameters θ_ν (e.g. weights of a deep neural net). For notational convenience, we denote $\nu(I_D; \theta_\nu)$ and $\nu(I_D; \theta_\nu)$ to be $\nu(I_D)$ and $\nu(I_R)$ respectively.

RL produces summary statistics (features) that are used in two scenarios: (a) to compare the hypothesis I_R with observed image I_D during inference (RL denoted by ν in this setting), and (b) as a dimensionality reduction technique for hashing learned data-driven proposals (RL denoted by ν_{dd} and its parameters $\theta_{\nu_{dd}}$). *Picture* supports a variety of summary statistic functions including raw pixels, contours [29] and supervised/unsupervised convolutional neural network (CNN) architectures[76, 82].

Likelihood and Likelihood-free Comparator: *Picture* supports likelihood $P(I_D|I_R)$ inference in a bayesian setting. However, in the presence of black-box rendering simulators, the likelihood function is often unavailable in closed form. Given an arbitrary distance function $\lambda(\nu(I_D), \nu(I_R))$ (e.g. L1 error), approximate bayesian computation (ABC) [152] can be used to perform likelihood-free inference.

4.3 Inference

We can formulate the task of image interpretation as approximately sampling mostly likely values of S^ρ given observed image I_D (L stands for $P(I_D|I_R, X^\rho)$):

$$P(S^\rho|I_D) \propto \int P(S^\rho)P(X^\rho)\delta_{render(S^\rho,X^\rho)}(I_R)\ L\ dX^\rho$$

Automatic inference in *Picture* programs can be especially hard due to a mix of discrete and continuous scene variables, which may be independent a priori but highly coupled in their posterior distributions ("explaining away"), and also because clutter, occlusion or noise can lead to local maxima of the scene posterior.

Given a program trace ρ, probabilistic inference amounts to updating (S^ρ, X^ρ) to $(S^{‘\rho}, X^{‘\rho})$ until convergence via proposal kernels $q((S^\rho, X^\rho) \rightarrow q(S^{‘\rho}, X^{‘\rho}))$. Let $K = |\{S^\rho\}|+|\{X^\rho\}|$ and $K' = |\{S^{‘\rho}\}|+|\{X^{‘\rho}\}|$ be the total number of random choices in the execution before and after applying the proposal kernels $q(.)$. Let the log-likelihoods of old and new trace be $L = P(I_D|I_R, X)$ and $L' = P(I_D|I'_R, X')$ respectively. Let us denote the probabilities of deleted and newly created random choices created in S^ρ to be $P(S^\rho_{del})$ and $P(S^\rho_{new})$ respectively. Let $q_{(S',X')\rightarrow(S,X)} := q((S^{‘\rho}, X^{‘\rho}) \rightarrow (S^\rho, X^\rho))$

Modules	Functional Description
Scene Representation S:	

```
light_source { <0, 199, 20>
                  color rgb<1.5,1.5,1.5> }
camera { location <30,48,-10> angle 40
         look_at <30,44,50> }

object{leg-right vertices ...
       trans  <32.7,43.6,9>}
object{arm-left vertices scale 0.2
       ... rotate x*0}
...
object{arm-left texture}
```

Program trace:	$\rho = \{\rho_i\}$
Rendering tolerance:	$X^{\rho} \in \rho$
Stochastic Scene:	$S^{\rho} \in \rho$
Approximate Rendering:	I_R
Approximate Renderer:	$render : (S, X) \to I_R$
Image data:	I_D
Data-driven Proposals:	$(f, T, \nu_{dd}, \theta_{\nu_{dd}}) \to q_{data}(.)$
Data representations:	$\nu(I_D)$ and $\nu(I_R)$
Comparator:	$\lambda : (\nu(I_D), \nu(I_R)) \to \mathbb{R}$
	$P(\nu(I_D) \vert \nu(I_R), X)$
Rendering Differentiator:	$\nabla_{S^{\rho}_{real}} : \rho \to$
	$grad\_model\_density(S_{real}; I_D)$

Figure 4-4: **Formal Summary:** The scene S can be conceptualized as a program that describes the structure of known or unknown number of objects, texture-maps, lighting and other scene variables. The symbol T denotes the number of times the program f is executed to generate data-driven proposals (see section 4.3.2 for details). The rendering differentiator produces gradients of the program density with respect to continuous variables S_{real} in the program.

and $q_{(S,X)\rightarrow(S',X')} := q((S^\rho, X^\rho) \rightarrow (S^{`\rho}, X^{`\rho}))$. The new trace $(S^{`\rho}, X^{`\rho})$ can now be accepted or rejected using the acceptance ratio:

4.3.1 Distance Metrics and Likelihood-free Inference

The likelihood function in closed form is often unavailable when integrating top-down automatic inference with bottom-up computational elements. Moreover, this issue is exacerbated when programs use black-box rendering simulators. Approximate bayesian computation (ABC) allows Bayesian inference in likelihood-free settings, where the basic idea is to use a summary statistic function $\nu(.)$, distance metric $\lambda(\nu(I_D), \nu(I_R))$ and tolerance variable X^ρ to approximately sample the posterior distribution [152].

Inference in likelihood-free settings can also be interpreted as a variant of the probabilistic approximate MCMC algorithm [152], which is similar to MCMC but with an additional tolerance parameter $\epsilon \in X^\rho$ on the observation model. We can interpret our approach as systematically reasoning about the model error arising due to the difference of generative model's "cartoon" view of the world with reality. Let Θ be the space of all possible renderings I_R that could be hypothesized and P_ϵ be the error model (e.g. Gaussian). The target stationary distribution that we wish to sample can be expressed as:

$$P(S^\rho|I_D) \propto \int_\Theta P(S^\rho)P_\epsilon(\nu(I_D) - \nu(I_R))P(\nu(I_R)|S^\rho)dI_R.$$

During inference, the updated scene $S^{`\rho}$ (assuming random choices remain unchanged, otherwise add terms relating to addition/deletion of random variables as in equation 1) can then be accepted with probability:

$$min\left(1, \frac{P_\epsilon(\nu(I_D) - \nu(I'_R))P(S^{`\rho})P(X^{`\rho})\ q_{(S',X')\rightarrow(S,X)}}{P_\epsilon(\nu(I_D) - \nu(I_R))P(S^\rho)P(X^\rho)\ q_{(S,X)\rightarrow(S',X')}}\right).$$

4.3.2 Proposal Kernels

In this section, we propose a variety of proposal kernels for scaling up *Picture* to complex 3D scenes.

Local and Blocked Proposals from Prior: Single site metropolis hastings moves on continuous variables and Gibbs moves on discrete variables can be useful in many cases. However, because the latent pose variables for objects in 3D scenes (e.g., positions and orientations) are often highly coupled, our inference library allows users to define arbitrary blocked proposals: $q_P((S^\rho, X^\rho) \rightarrow (S^{\prime\rho}, X^{\prime\rho})) = \prod_{\rho_i' \in (S^\rho, X^\rho)} P(\rho_i')$

Gradient Proposal: *Picture* inference supports automatic differentiation for a restricted class of programs (where each expression provides output and gradients w.r.t input). Therefore it is straightforward to obtain $\nabla_{S_{real}} \rho$ using reverse mode automatic differentiation, where $S_{real} \in S^\rho$ denotes all continuous variables. This enables us to automatically construct Hamiltonian Monte Carlo proposals[100, 153] $q_{hmc}(S^\rho_{real} \rightarrow S_{real}{}^{\prime\rho})$ (see supplementary material for a simple example).

Elliptical Slice Proposals: To adaptively propose changes to a large set of latent variables at once, our inference library supports elliptical slice moves with or without adaptive step sizes (see Figure 4-2 for an example)[12, 97]. For simplicity, assume $S_{real} \sim \mathcal{N}(0, \Sigma)$. We can generate a new sub-trace S'_{real} efficiently as follows: $S'_{real} = \sqrt{1 - \alpha^2} S_{real} + \alpha\theta$, where $\theta \sim \mathcal{N}(0, \Sigma)$ and $\alpha \sim Uniform(-1, 1)$.

Data-driven Proposals: The top-down nature of MCMC inference in generative models can be slow, due to the initial "burn-in" period and the cost of mixing among multiple posterior modes. However, vision problems often lend themselves to much faster bottom-up inference based on data-driven proposals [64, 146]. Arguably the most important inference innovation in *Picture* is the capacity for automatically constructing data-driven proposals by simple learning methods. Such techniques fall under the broader idea of amortizing or caching inference to improve speed and accuracy [132]. We have explored several approaches generally inspired by the *Helmholtz machine* [54, 23], and indeed Helmholtz's own proposals, including using deep learning to construct bottom-up predictors for all or a subset of the latent scene variables in ρ [158, 82]. Here we focus on a simple and general-purpose memory-based approach (similar in spirit to the *informed sampler* [64]) that can be summarized as follows: We "imagine" a large set of hypothetical scenes sampled from the generative model, store the imagined latents and corresponding rendered image data in memory, and build a fast bottom-up kernel density estimate proposer that samples variants of stored graphics program traces best matching the observed image data – where these bottom-up "matches" are determined using the same representation layer tools we introduced earlier for comparing top-down rendered and observed images. Figure 4-5 provides a qualitative description of the algorithm. More formally, we can construct data-driven proposals q_{data} as follows:

(1) Specify the number of times T to forward simulate (unconditional runs) the graphics program f.

(2) Draw T samples from f to create program traces ρ^t and approximate renderings I_R^t, where $\{1 \leq t \leq T\}$.

(3) Specify a summary statistic function ν_{dd} with model parameters $\theta_{\nu_{dd}}$. We can use the same representation layer tools introduced earlier to specify ν_{dd}, subject to the additional constraint that feature dimensionalities should be as small as possible to enable proposal learning and evaluation on massive datasets.

(4) Fine-tune parameters $\theta_{\nu_{dd}}$ of the representation layer ν_{dd} using supervised learning to best predict program traces $\{\rho^t\}_{t=1}^T$ from corresponding rendered images $\{I_R^t\}_{t=1}^T$. If labeled data is available for full or partial scene traces $\{S_p^\rho\}$ corresponding to actual observed images $\{I_D\}$, the parameters $\theta_{\nu_{dd}}$ can also be fine-tuned further to predict these labels. (see deep convolutional inverse graphics network [82] as an alternative ν_{dd}, which works in an weakly supervised setting.)

(5) Define a hash function $H : \nu(I_R^t) \rightarrow h^t$, where h^t denotes the hash value for $\nu(I_R^t)$. For instance, H can be defined in terms of K-nearest neighbors or a Dirichlet Process mixture model. Store triplets $\{\rho^t, \nu(I_R^t), h^t\}$ in a database $\mathcal{C}$.

(6) To generate data-driven proposals for an observed image I_D with hash value h_D, extract all triplets $\{\rho^j, \nu(I_R^j), h^j\}_{j=1}^N$ that have hash value equal to h_D. We can then estimate the data-driven proposal as:

$$q_{data}(S^{\rho} \rightarrow S^{\text{`}\rho}|\mathcal{C}, I_D) = P_{density}(\{\rho^j\}_{j=1}^N),$$

where $P_{density}$ is a density estimator such as the multivariate gaussian kernel in [64]).

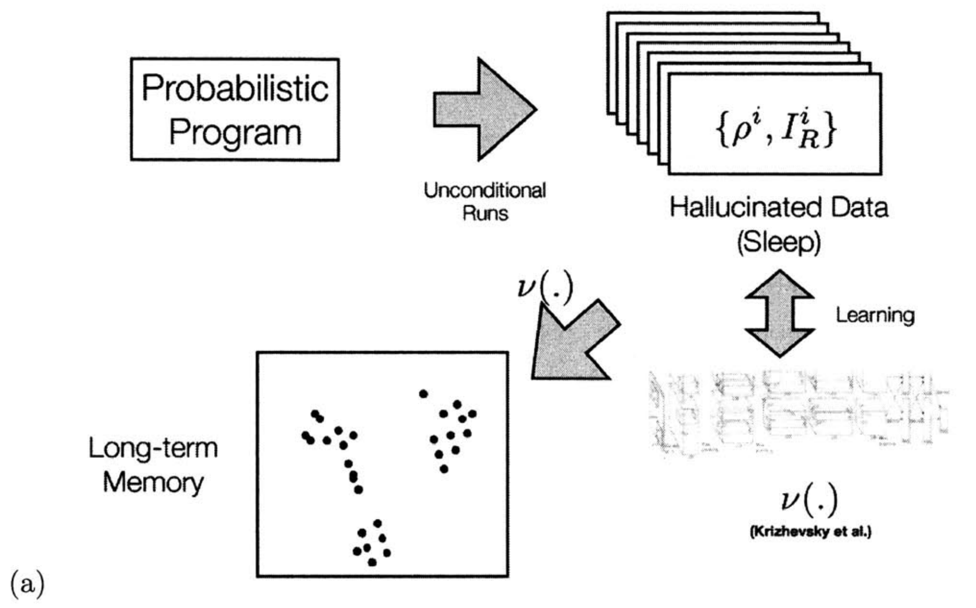
4.4 Example *Picture* Programs

To illustrate how *Picture* can be applied to a wide variety of 2D and 3D computer vision problems, we present three sample applications to the core vision tasks of 3D body pose estimation, 3D reconstruction of objects and 3D face analysis. Although additional steps could be employed to improve results for any of these tasks, and there may exist better fine-tuned baselines, our goal here is to show how to solve a broad class of problems efficiently and competitively with task-specific baseline systems, using only minimal problem-specific engineering.

See Appendix A for details about some of the experiments.

4.4.1 3D Analysis of Faces

We obtained a 3D deformable face model trained on laser scanned faces from Paysan *et al* [105]. After training with this dataset, the model generates a mean shape mesh and mean texture map, along with principal components and eigenvectors. A new face can be rendered by randomly choosing coefficients for the 3D model and running the program shown in Figure 4-2. The representation layer ν in this program used the top convolutional-layer features from the ImageNet CNN model[65] as well as raw pixels. (Even better results can be obtained using the deep convolutional inverse graphics network [82] instead of the CNN.) We evaluated the program on a held-out test set of 2D projected images of 3D laser scanned data (dataset from [105]). We additionally produced a dataset of about 30 images from the held-out set with different viewpoints and lighting conditions. In Figure 4-3, we show qualitative results of inference runs on the dataset.



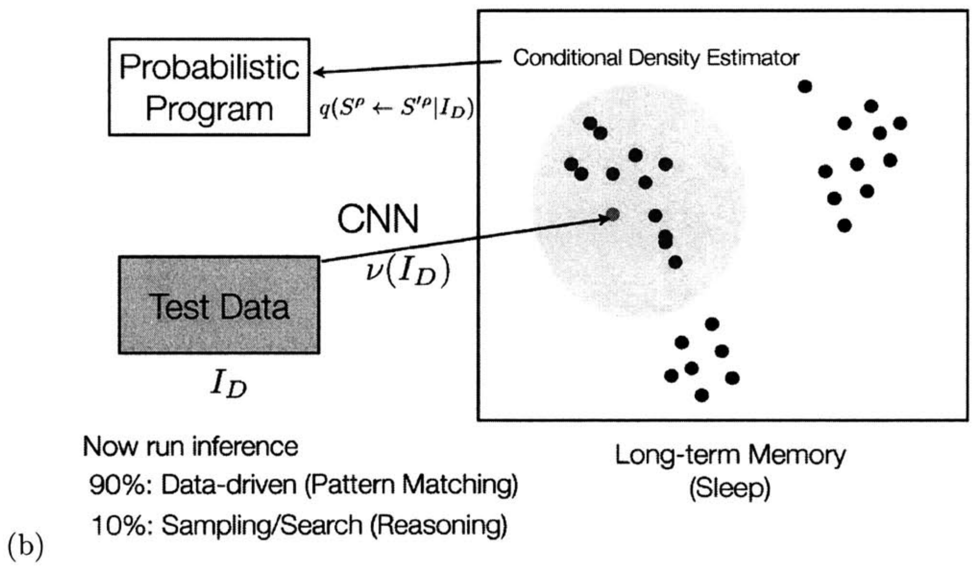

Probabilistic Program
Conditional Density Estimator
$q(S^\rho \leftarrow S'^\rho | I_D)$
CNN
$\nu(I_D)$
Test Data
I_D
Now run inference
90%: Data-driven (Pattern Matching)
10%: Sampling/Search (Reasoning)
Long-term Memory (Sleep)
(b)

Figure 4-5: Helmholtz Proposals. **(a)** Training Phase. **(b)** Testing Phase

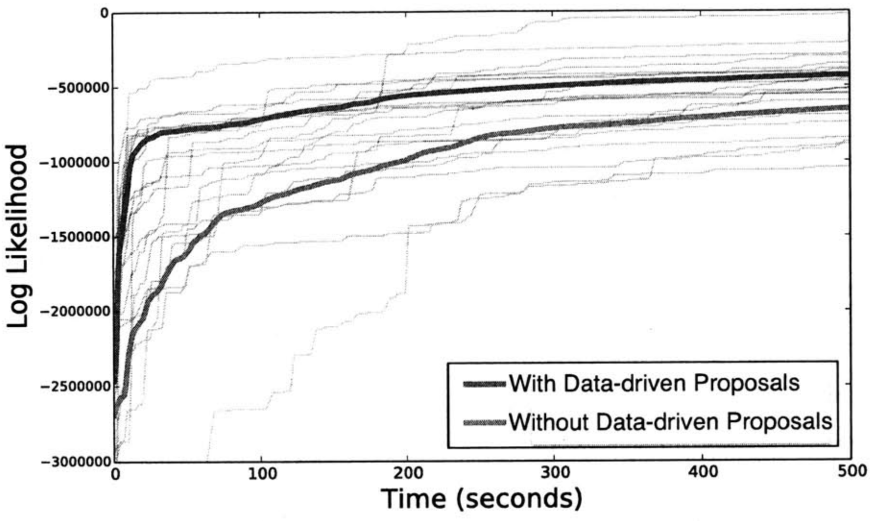


Figure 4-6: **The effect of adding data-driven proposals for 3D face program:** A mixture of automatically learned data-driven proposals and elliptical slice proposals significantly improves speed and accuracy of inference over a pure elliptical slice sampler. We ran 50 independent chains for both approaches and show a few sample trajectories as well as the mean trajectories (in bold).

During experimentation, we discovered that since the number of latent variables is large (8 sets of 100 dimensional continuous coupled variables), elliptical slice moves are significantly more efficient than Metropolis-Hastings proposals (see supplementary Figure 2 for quantitative results). We also found that adding learned data-driven proposals significantly outperforms using only the elliptical slice proposals in terms of both speed and accuracy. We trained the data-driven proposals from around 100k program traces drawn from unconditional runs. The summary statistic function ν_{dd} used were the top convolutional-layer features from the pre-trained ImageNet CNN model[65]. The conditional proposal density $P_{density}$ was a multivariate kernel density function over cached latents with a Gaussian Kernel (0.01 bandwidth). Figure 4-6 shows the gains in inference from use of a mixture kernel of these data-driven proposals (0.1 probability) and elliptical slice proposals (0.9 probability), relative to a pure elliptical slice sampler.

Many other academic researchers have used 3D deformable face models in an analysis-by-synthesis based approach [88, 70, 2]. However, *Picture* is the only system to solve this as well as many other unrelated computer vision problems using a general-purpose system. Moreover, the data-driven proposals and abstract summary statistics (top convolutional-layer activations) allow us to tackle the problem without explicitly using 2D face landmarks as compared to traditional approaches.

4.4.2 3D Human Pose Estimation

We developed a *Picture* program for parsing 3D pose of articulated humans from single images. There has been notable work in model-based approaches [46, 87] for 3D human pose estimation, which served as an inspiration for the program we describe in this section. However, in contrast to *Picture*, existing approaches typically require custom inference strategies and significant task-specific model engineering. The probabilistic code (see supplementary Figure 4) consists of latent variables denoting bone and joints of an articulated 3D base mesh of a body. In our probabilistic code, we use an existing base mesh of a human body, defined priors over bone location and joints, and enable the armature skin-modifier API via *Picture*'s Blender engine API. The latent scene S^ρ in this program can be visualized as a tree with the root node around the center of the mesh, and consists of bone location variables, bone rotation variables and camera parameters. The representation layer ν in this program uses fine-grained image contours [29] and the comparator is expressed as the probabilistic chamfer distance [140].

We evaluated our program on a dataset of humans performing a variety of poses, which was aggregated from KTH [119] and LabelMe [113] images with significant occlusion in the "person sitting"(around 50 total images). This dataset was chosen to highlight the distinctive value of a graphics model-based approach, emphasizing certain dimensions of task difficulty while minimizing others: While graphics simulators for articulated bodies can represent arbitrarily complex body configurations, they are limited with respect to fine-grained appearance (e.g., skin and clothing), and fast methods for fine-grained contour detection currently work well only in low clutter environments. We initially used only single-site MH proposals, although blocked proposals or HMC can somewhat accelerate inference.

We compared this approach with the discriminatively trained Deformable Parts

Model (DPM) for pose estimation [157] (referred as DPM-pose), which is notably a 2D pose model. As shown in Figure 4-7b, images with people sitting and heavy occlusion are very hard for the discriminative model to get right – mainly due to "missing" observation signal – while our model-based approach can handle these reasonably if we constrain the knee parameters to bend only in natural ways in the prior. Most of our model's failure cases, as shown in Figure 4-7b, are in inferring the arm position; this is typically due to noisy and low quality feature maps around the arm area due to its small size.

In order to quantitatively compare results, we project the 3D pose obtained from our model to 2D key-points. As shown in Figure 4-7a, our system localizes these key-points significantly better than DPM-pose on this dataset. However, DPM-pose is a much faster bottom-up method, and we explored ways to combine its strengths with our model-based approach, by using it as the basis for learning data-driven proposals. We generated around 500k program traces by unconditionally running the body pose program. We used a pre-trained DPM pose model [157] as the function ν_{dd}, and used a similar density function $P_{density}$ as in the face example. As shown in Figure 4-8, inference using a mixture kernel of data-driven proposals (0.1 probability) and single-site MH (0.9 probability) consistently outperformed pure top-down MH inference in both speed and accuracy. We see this as representative of many ways that top-down inference in model-based approaches could be profitably combined with fast bottom-up methods like DPM-pose to solve richer scene parsing problems more quickly.

4.4.3 3D Shape Program

Lathing and casting is a useful representation to express CAD models and inspires our approach to modeling medially-symmetric 3D objects. It is straightforward to generate random CAD object models using a probabilistic program, as shown in supplementary Figure 3. However, the distribution induced by such a program may be quite complex. Given object boundaries in $\mathcal{B} \in \mathcal{R}^2$ space, we can lathe an object by taking a cross section of points (fixed for this program), defining a medial axis for the cross section and sweeping the cross section across the medial axis by continuously perturbing with respect to $\mathcal{B}$. Capturing the full range of 3D shape variability in real objects requires a very large space of possible boundaries $\mathcal{B}$. To this end, *Picture* allows flexible non-parametric priors over object profiles: here we generate $\mathcal{B}$ from a Gaussian Process [109] (GP). The probabilistic shape program produces an intermediate mesh of all or part of the 3D object (soft-constrained to be in the middle of the scene), which then gets rendered to an image I_R by a deterministic camera re-projection function. The representation layer and the comparator used in this program were same as those used for the 3D human pose example. The proposal kernel we used during inference consisted of blocked MCMC proposals on all the coupled continuous variables as described in the supplementary material. (For more details of the program and inference summarized here, refer to supplementary Section 1.)

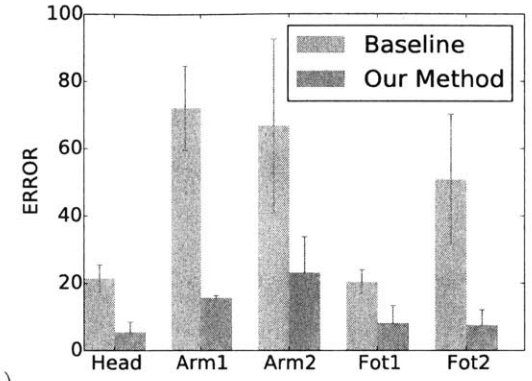
We evaluate this program on an RGB image dataset of 3D objects with large shape variability. We asked CAD experts to manually generate CAD model fits to these images in Blender, and evaluated our approach in comparison to a state-of-the-art 3D surface reconstruction algorithm from [5](SIRFS). To judge quantitative performance, we calculated two metrics: (a) Z-MAE – Shift-invariant surface mean-squared error and (b) N-MSE – mean-squared error over normals[5]. As shown in Figure 4-9, inference using our probabilistic shape program has a lower Z-MAE and N-MSE score than SIRFS [5], and we also obtain qualitatively better reconstruction results. However, it is important to note that SIRFS predominantly utilizes only low level shape priors such as piece-wise smoothness, in contrast to the high-level shape priors we assume, and SIRFS solves a more general and harder problem of inferring full intrinsic images (shape, illumination and reflectance). In the future, we hope to combine the best of SIRFS-style approaches and our probabilistic CAD programs to reconstruct rich 3D shape and appearance models for generic object classes, robustly and efficiently.

4.5 Discussion

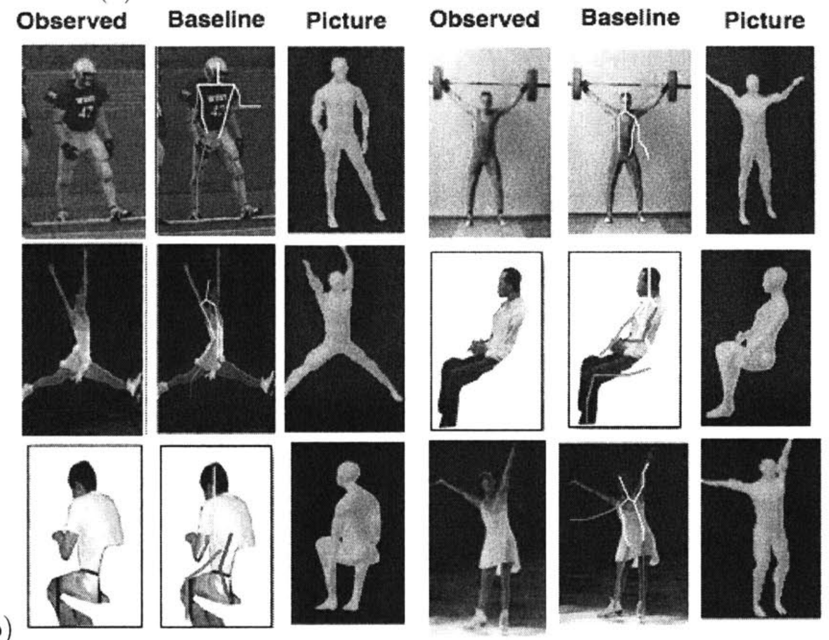
There are many promising directions for future research in probabilistic graphics programming. Introducing a dependency tracking mechanism could let us exploit the many conditional independencies in rendering for more efficient parallel inference. Automatic particle-filter based inference schemes [155, 81] could extend the approach to image sequences. Better illumination [162], texture and shading models could let us work with more natural scenes. Procedural graphics techniques [4, 33] would support far more complex object and scene models [159, 48, 25, 51]. Flexible scene generator libraries will be essential in scaling up to the full range of scenes people can interpret.

We are also interested in extending *Picture* by taking insights from learning based "analysis-by-synthesis" approaches such as transforming auto-encoders [55], capsule networks [143] and deep convolutional inverse graphics network [82]. These models learn an implicit graphics engine in an encoder-decoder style architecture. With probabilistic programming, the space of decoders need not be restricted to neural networks and could consist of arbitrary probabilistic graphics programs with internal parameters.

The recent renewal of interest in inverse graphics approaches to vision has motivated a number of new modeling and inference tools. Each addresses a different facet of the general problem. Earlier formulations of probabilistic graphics programming provided compositional languages for scene modeling and a flexible template for automatic inference. Differentiable renderers make it easier to fine-tune the numerical parameters of high-dimensional scene models. Data-driven proposal schemes suggest a way to rapidly identify plausible scene elements, avoiding the slow burn-in and mixing times of top-down MCMC-based inference in generative models. Deep neural networks, deformable parts models and other discriminative learning methods can be used to automatically construct good representation layers or similarity metrics for comparing hypothesized scenes to observed images. Here we show that by integrating all of these ideas into a single probabilistic language and inference framework, it may be feasible to begin scaling up inverse graphics to a range of real-world vision problems.



(a)



(b)

Figure 4-7: **Quantitative and qualitative results for 3D human pose program:** Refer to supplementary Figure 4 for the probabilistic program. We quantitatively evaluate the pose program on a dataset collected from various sources such as KTH [119], LabelMe [113] images with significant occlusion in the "person sitting" category and the Internet. On the given dataset, as shown in the error histogram in **(a)**, our model is more accurate on average than just using the DPM based human pose detector [157]. The histogram shows average error for all methods considered over the entire dataset separated over each body part.

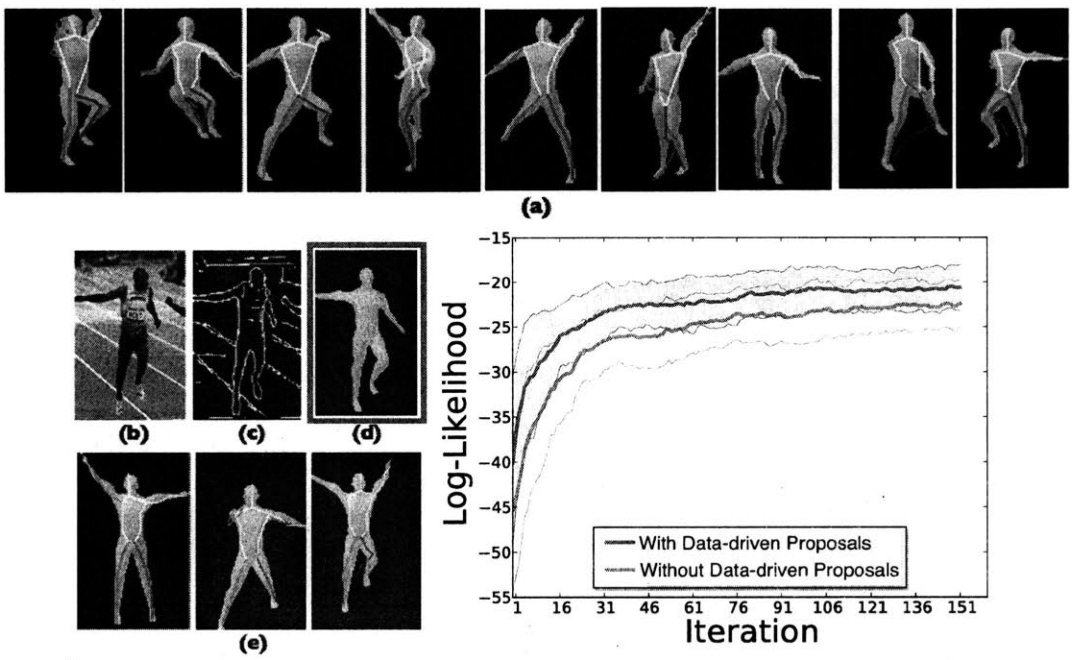


Figure 4-8: **Illustration of data-driven proposal learning for 3D human-pose program:** (**a**) Random program traces sampled from the prior during training. The colored stick figures are the results of applying DPM pose model on the *hallucinated* data from the program. (**b**) Representative test image. (**c**) Visualization of the representation layer $\nu(I_D)$. (**d**) Result after inference. (**e**) Samples drawn from the learned bottom-up proposals conditioned on the test image are semantically close to the test image and results are fine-tuned by top-down inference to close the gap. As shown on the log-l plot, we run about 100 independent chains with and without the learned proposal. Inference with a mixture kernel of learned bottom-up proposals and single-site MH consistently outperforms baseline in terms of both speed and accuracy.

to reconstruct input images. Our work is similar in spirit to these works but has some key differences: (a) It uses a very generic convolutional architecture in the encoder and decoder networks to enable efficient learning on large datasets and image sizes; (b) it can handle single static frames as opposed to pair of images required in [55]; and (c) it is generative.

5.3 Model

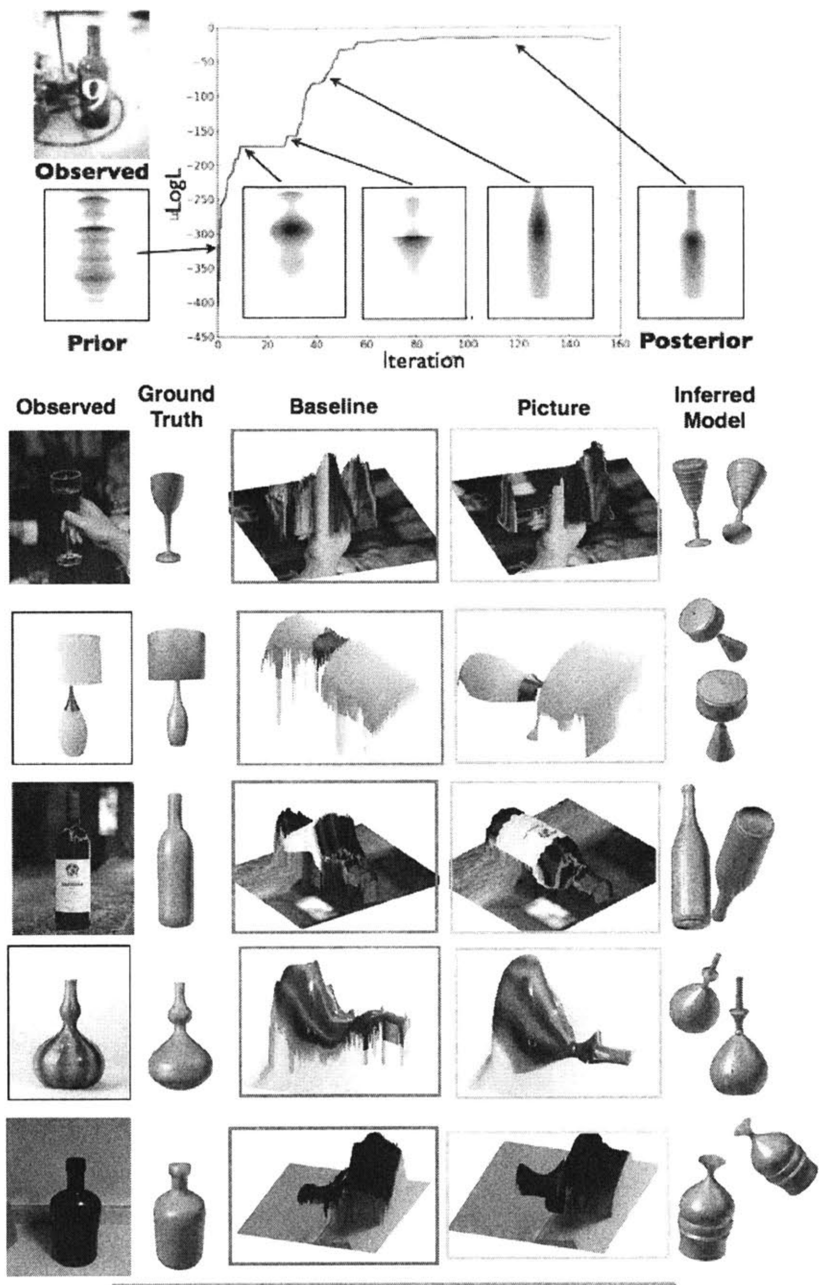
As shown in Figure 5-1, the basic structure of the Deep Convolutional Inverse Graphics Network (DC-IGN) consists of two parts: an encoder network which captures a distribution over *graphics codes* Z given data x and a decoder network which learns a conditional distribution to produce an approximation $\hat{x}$ given Z. Z can be a disentangled representation containing a factored set of latent variables $z_i \in Z$ such as pose, light and shape. This is important in learning a meaningful approximation of a 3D graphics engine and helps tease apart the generalization capability of the model with respect to different types of transformations.

Let us denote the encoder output of DC-IGN to be $y_e = encoder(x)$. The encoder output is used to parametrize the variational approximation $Q(z_i|y_e)$, where Q is chosen to be a multivariate normal distribution. There are two reasons for using this parametrization: (1) Gradients of samples with respect to parameters θ of Q can be easily obtained using the reparametrization trick proposed in [72], and (2) Various statistical shape models trained on 3D scanner data such as faces have the same multivariate normal latent distribution [105]. Given that model parameters W_e connect y_e and z_i, the distribution parameters $\theta = (\mu_{z_i}, \Sigma_{z_i})$ and latents Z can then be expressed as:

$$\mu_z = W_e y_e, \; \Sigma_z = \text{diag}(\exp(W_e y_e)) \tag{5.1}$$

$$\forall i, z_i \sim \mathcal{N}(\mu_{z_i}, \Sigma_{z_i}) \tag{5.2}$$

We present a novel training procedure which allows networks to be trained to have



Quantitative Metrics		
METHOD	Z-MAE	N-MSE
Barron et al.[2]	15.19	2.407×10^{-3}
Picture	11.40	1.704×10^{-3}

Figure 4-9: **Qualitative and quantitative results of 3D object reconstruction program**: Refer to supplementary Figure 3 for the probabilistic program. **Top:** We illustrate a typical inference trajectory of the sampler from prior to the posterior on a representative real world image. **Middle:** Qualitative results on representative images. **Bottom:** Quantitative results in comparison to [5]. For details about the scoring metrics, refer to section 4.4.3.

Chapter 5

Learning analysis-by-synthesis models

5.1 Introduction

Neural Networks can be thought of as differentiable programs or differentiable probabilistic programs. Under this view, we can define any computational process, discriminative or generative, and do learning or inference conditioned on data. Many modern deep learning systems are designed with an explicit structure for richer inductive biases – Neural Turing Machines [41] with explicit read/write memory heads, differentiable attention modules for selective read/write in images [63], transforming autoencoders [55], etc. This new conceptual paradigm can allow us to express structured probabilistic and non-probabilistic models, and apply efficient gradient based learning algorithms for training. In this section, I propose a purely learning based approach to inverse graphics, while preserving insights about the causal structure within images.

The desiderata for good representations of images can be summarized as follows: invariance, interpretability, abstraction, and disentanglement. The inverse graphics paradigm suggests a representation for images which provides these features. Computer graphics consists of a function to go from compact descriptions of scenes (the *graphics code*) to images, and this graphics code is typically disentangled to allow for rendering scenes with fine-grained control over transformations such as object location, pose, lighting, texture, and shape. This encoding is designed to easily and interpretably represent sequences of real data so that common transformations may be compactly represented in software code.

Recent work in inverse graphics [91, 79, 78] follows a general strategy of defining a probabilistic with latent parameters, then using an inference algorithm to find the most appropriate set of latent parameters given the observations. Recently, Tieleman *et al.* [143] moved beyond this two-stage pipeline by using a generic encoder network and a domain-specific decoder network to approximate a 2D rendering function. However, none of these approaches have been shown to automatically produce a semantically-interpretable graphics code and to learn a 3D rendering engine to reproduce images.

In this section, I present an approach which attempts to learn interpretable *graphics codes* for complex transformations such as out-of-plane rotations and lighting variations. Given a set of images, we use a hybrid encoder-decoder model to learn a representation that is disentangled with respect to various transformations such as object out-of-plane rotations and lighting variations. We employ a deep directed graphical model with many layers of convolution and de-convolution operators that is trained using the Stochastic Gradient Variational Bayes (SGVB) algorithm [72].

I propose a training procedure to encourage each group of neurons in the *graphics code* layer to distinctly represent a specific transformation. To learn a disentangled representation, we train using data where each mini-batch has a set of active and inactive transformations, but we do not provide target values as in supervised learning; the objective function remains reconstruction quality. For example, a nodding face would have the 3D elevation transformation active but its shape, texture and other transformations would be inactive. We exploit this type of training data to force chosen neurons in the *graphics code* layer to specifically represent active transformations, thereby automatically creating a disentangled representation. Given a single face image, our model can re-generate the input image with a different pose and lighting. We present qualitative and quantitative results of the model's efficacy at learning a 3D rendering engine.

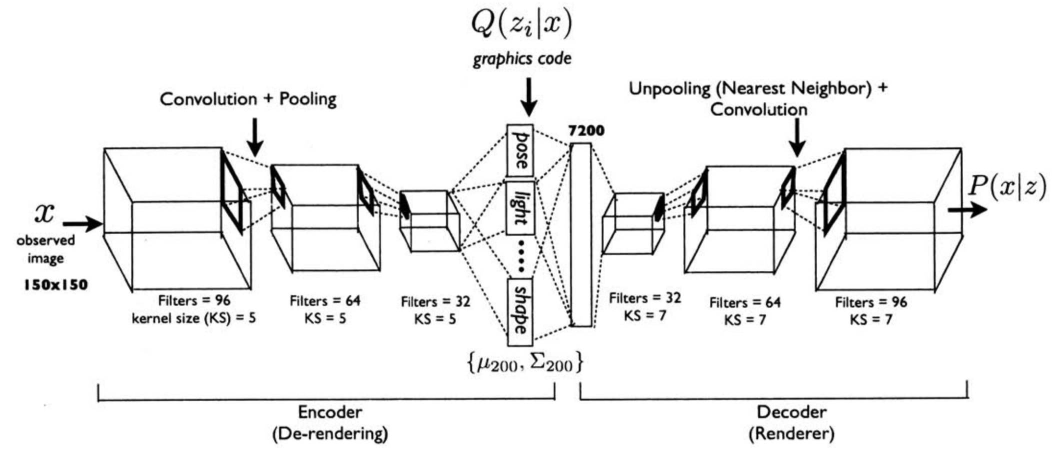


Figure 5-1: **Model Architecture:** Deep Convolutional Inverse Graphics Network (DC-IGN) has an encoder and a decoder. We follow the variational autoencoder [72] architecture with variations. The encoder consists of several layers of convolutions followed by max-pooling and the decoder has several layers of unpooling (upsampling using nearest neighbors) followed by convolution. (a) During training, data x is passed through the encoder to produce the posterior approximation $Q(z_i|x)$, where z_i consists of scene latent variables such as pose, light, texture or shape. In order to learn parameters in DC-IGN, gradients are back-propagated using stochastic gradient descent using the following variational object function: $-log(P(x|z_i)) + KL(Q(z_i|x)||P(z_i))$ for every z_i. We can force DC-IGN to learn a disentangled representation by showing mini-batches with a set of inactive and active transformations (e.g. face rotating, light sweeping in some direction etc). (b) During test, data x can be passed through the encoder to get latents z_i. Images can be re-rendered to different viewpoints, lighting conditions, shape variations, etc by setting the appropriate graphics code group (z_i), which is how one would manipulate an off-the-shelf 3D graphics engine.

5.2 Related Work

As mentioned previously, a number of generative models have been proposed in the literature to obtain abstract visual representations. Unlike most RBM-based models [53, 114, 86], our approach is trained using back-propagation with objective function consisting of data reconstruction and the variational bound.

Relatively recently, Kingma *et al.* [72] proposed the SGVB algorithm to learn generative models with continuous latent variables. In this work, a feed-forward neural network (encoder) is used to approximate the posterior distribution and a decoder network serves to enable stochastic reconstruction of observations. In order to handle fine-grained geometry of faces, we work with relatively large scale images (150×150

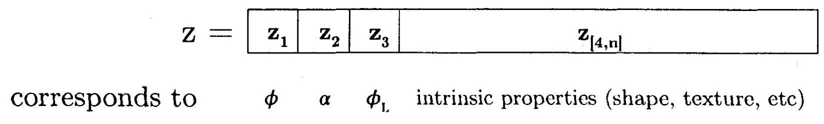


Figure 5-2: **Structure of the representation vector.** ϕ is the azimuth of the face, α is the elevation of the face with respect to the camera, and ϕ_L is the azimuth of the light source.

pixels). Our approach extends and applies the SGVB algorithm to jointly train and utilize many layers of convolution and de-convolution operators for the encoder and decoder network respectively. The decoder network is a function that transform a compact *graphics code* (200 dimensions) to a 150×150 image. We propose using unpooling (nearest neighbor sampling) followed by convolution to handle the massive increase in dimensionality with a manageable number of parameters.

Recently, [30] proposed using CNNs to generate images given object-specific parameters in a supervised setting. As their approach requires ground-truth labels for the *graphics code* layer, it cannot be directly applied to image interpretation tasks. Our work is similar to Ranzato *et al.* [108], whose work was amongst the first to use a generic encoder-decoder architecture for feature learning. However, in comparison to our proposal their model was trained layer-wise, the intermediate representations were not disentangled like a *graphics code*, and their approach does not use the variational auto-encoder loss to approximate the posterior distribution. Our work is also similar in spirit to [139], but in comparison our model does not assume a Lambertian reflectance model and implicitly constructs the 3D representations. Another piece of related work is Desjardins *et al.* [26], who used a spike and slab prior to factorize representations in a generative deep network.

In comparison to existing approaches, it is important to note that our encoder network produces the interpretable and disentangled representations necessary to learn a meaningful 3D graphics engine. A number of inverse-graphics inspired methods have recently been proposed in the literature [91]. However, most such methods rely on hand-crafted rendering engines. The exception to this is work by Hinton *et al.* [55] and Tieleman [143] on *transforming autoencoders* which use a domain-specific decoder

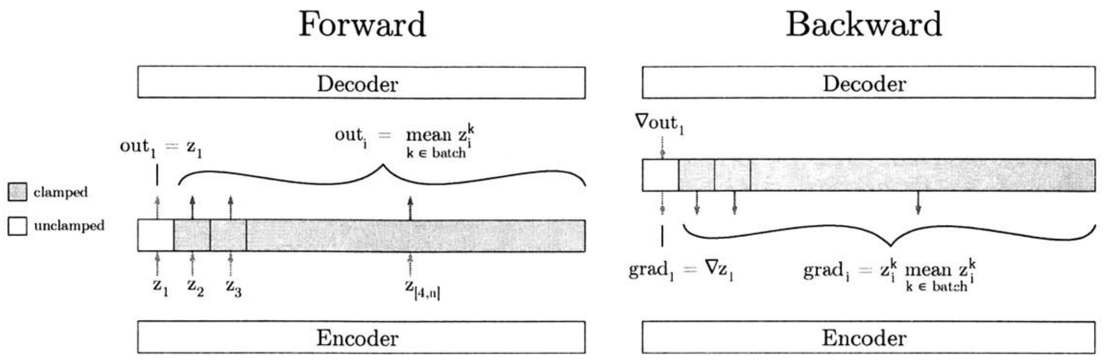

Forward
Backward
Decoder
Decoder
$\text{out}_1 = z_1$
$\text{out}_i = \underset{k \in \text{batch}}{\text{mean}}\, z_i^k$
∇out_1
clamped
unclamped
z_1 z_2 z_3
$z_{[4,n]}$
$\text{grad}_1 = \nabla z_1$
$\text{grad}_i = z_i^k\ \underset{k \in \text{batch}}{\text{mean}}\, z_i^k$
Encoder
Encoder

Figure 5-3: **Training on a minibatch in which only ϕ, the azimuth angle of the face, changes.** During the forward step, the output from each component $z_i \neq z_1$ of the encoder is altered to be the same for each sample in the batch. This reflects the fact that the generating variables of the image (e.g. the identity of the face) which correspond to the desired values of these latents are unchanged throughout the batch. By holding these outputs constant throughout the batch, the single neuron z_1 is forced to explain all the variance within the batch, i.e. the full range of changes to the image caused by changing ϕ. During the backward step z_1 is the only neuron which receives a gradient signal from the attempted reconstruction, and all $z_i \neq z_1$ receive a signal which nudges them to be closer to their respective averages over the batch. During the complete training process, after this batch, another batch is selected at random; it likewise contains variations of only one of $\phi, \alpha, \phi_L, intrinsic$; all neurons which do not correspond to the selected latent are clamped; and the training proceeds.

disentangled and interpretable representations.

5.3.1 Training with Specific Transformations

The main goal of this work is to learn a representation of the data which consists of disentangled and semantically interpretable latent variables. We would like only a small subset of the latent variables to change for sequences of inputs corresponding to real-world events.

One natural choice of target representation for information about scenes is that already designed for use in graphics engines. If we can deconstruct a face image by splitting it into variables for pose, light, and shape, we can trivially represent the same transformations that these variables are used for in graphics applications. Figure 5-2 depicts the representation which we attempt to learn.

With this goal in mind, we perform a training procedure which directly targets this definition of disentanglement. We organize our data into mini-batches corresponding to changes in only a single scene variable (azimuth angle, elevation angle, azimuth angle of the light source); these are transformations which might occur in the real world. We term these the *extrinsic* variables, and they are represented by the components $z_{1,2,3}$ of the encoding.

We also generate mini-batches in which the three extrinsic scene variables are held fixed but all other properties of the face change. That is, these batches consist of many different faces under the same viewing conditions and pose. These *intrinsic* properties of the model, which describe identity, shape, expression, etc., are represented by the remainder of the latent variables $z_{[4,200]}$. These mini-batches varying intrinsic properties are interspersed stochastically with those varying the extrinsic properties.

We train this representation using SGVB, but we make some key adjustments to the outputs of the encoder and the gradients which train it. The procedure (Figure 5-3) is as follows.

1. Select at random a latent variable z_{train} which we wish to correspond to one of {azimuth angle, elevation angle, azimuth of light source, intrinsic properties}.

2. Select at random a mini-batch in which that only that variable changes.

3. Show the network each example in the minibatch and capture its latent representation for that example z^k.

4. Calculate the average of those representation vectors over the entire batch.

5. Before putting the encoder's output into the decoder, replace the values $z_i \neq z_{train}$ with their averages over the entire batch. These outputs are "clamped".

6. Calculate reconstruction error and backpropagate as per SGVB in the decoder.

7. Replace the gradients for the latents $z_i \neq z_{train}$ (the clamped neurons) with their difference from the mean (see Section 5.3.2). The gradient at z_{train} is passed through unchanged.

8. Continue backpropagation through the encoder using the modified gradient.

Since the intrinsic representation is much higher-dimensional than the extrinsic ones, it requires more training. Accordingly we select the type of batch to use in a ratio of about 1:1:1:10, azimuth : elevation : lighting : intrinsic; we arrived at this ratio after extensive testing, and it works well for both of our datasets.

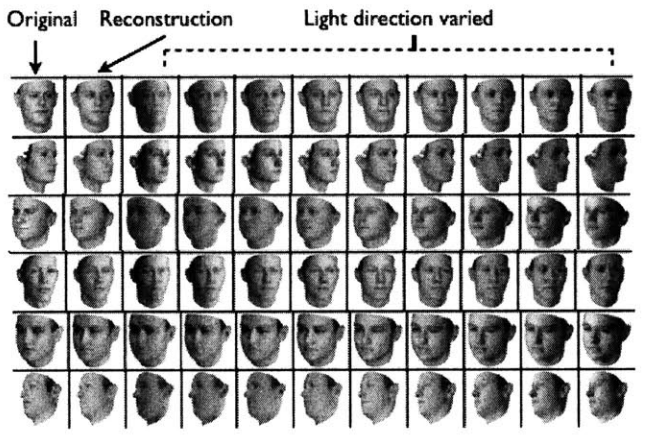
This training procedure works to train both the encoder and decoder to represent certain properties of the data in a specific neuron. By clamping the output of all but one of the neurons, we force the decoder to recreate all the variation in that batch using only the changes in that one neuron's value. By clamping the gradients, we train the encoder to put all the information about the variations in the batch into one output neuron.

This training method leads to networks whose latent variables have a strong *equivariance* with the corresponding generating parameters, as shown in Figure 5-5. This allows the value of the true generating parameter (e.g. the true angle of the face) to be trivially extracted from the encoder.

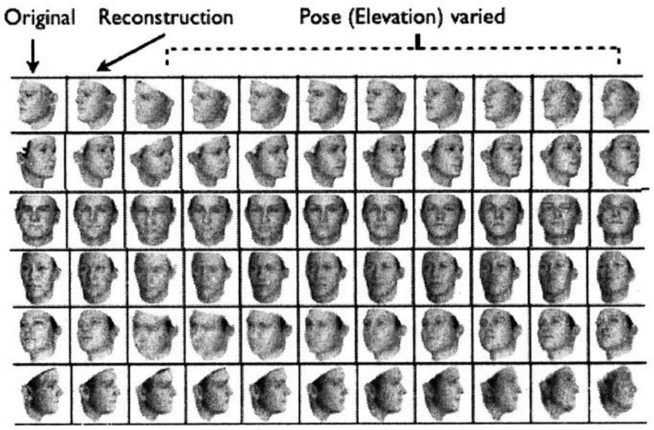
5.3.2 Invariance Targeting

By training with only one transformation at a time, we are encouraging certain neurons to contain specific information; this is equivariance. But we also wish to explicitly *discourage* them from having *other* information; that is, we want them to be invariant to other transformations. Since our mini-batches of training data consist of only one transformation per batch, then this goal corresponds to having all but one of the output neurons of the encoder give the same output for every image in the batch.

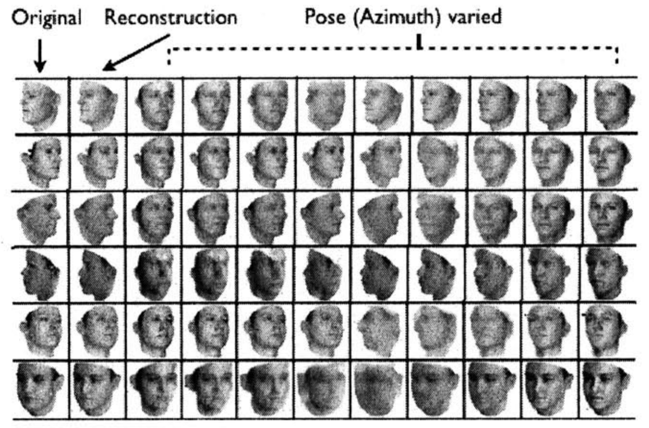
To encourage this property of the DC-IGN, we train all the neurons which correspond to the inactive transformations with an error gradient equal to their difference from the mean. It is simplest to think about this gradient as acting on the set of subvectors $z_{inactive}$ from the encoder for each input in the batch. Each of these $z_{inactive}$'s will be pointing to a close-together but not identical point in a high-dimensional space; the invariance training signal will push them all closer together. We don't care where they are; the network can represent the face shown in this batch however it likes. We only care that the network always represents it as still being the same face, no



(a)



(b)



(c)

Figure 5-4: **Manipulating light and elevation variables:** Qualitative results showing the generalization capability of the learned DC-IGN decoder to re-render a single input image with different pose directions. **(a)** We change the latent z_{light} smoothly leaving all 199 other latents unchanged. **(b)** We change the latent $z_{elevation}$ smoothly leaving all 199 other latents unchanged. (c) The latent neuron $z_{azimuth}$ is changed to random values but all other latents are clamped.

matter which way it's facing. This regularizing force needs to be scaled to be much smaller than the true training signal, otherwise it can overwhelm the reconstruction goal. Empirically, a factor of 1/100 works well.

5.4 Experiments

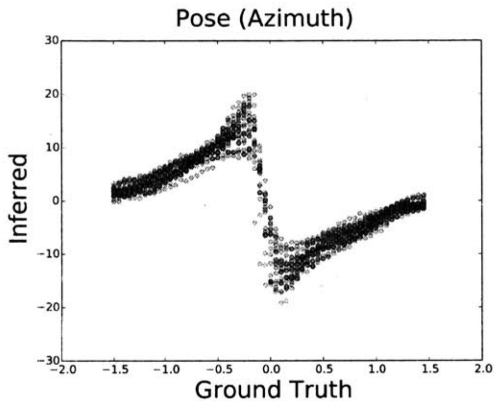
We trained our model on about 12,000 batches of faces generated from a 3D face model obtained from Paysan *et al.* [105], where each batch consists of 20 faces with random variations on face identity variables (shape/texture), pose, or lighting. We used the *rmsprop* [142] learning algorithm during training and set the meta learning rate equal to 0.0005, the momentum decay to 0.1 and weight decay to 0.01.

To ensure that these techniques work on other types of data, we also trained networks to perform reconstruction on images of widely varied 3D chairs from many perspectives derived from the Pascal Visual Object Classes dataset as extracted by Aubry *et al.* [96, 3]. This task tests the ability of the DC-IGN to learn a rendering function for a dataset with high variation between the elements of the set; the chairs vary from office chairs to wicker to modern designs, and viewpoints span 360 degrees and two elevations. These networks were trained with the same methods and parameters as the ones above.

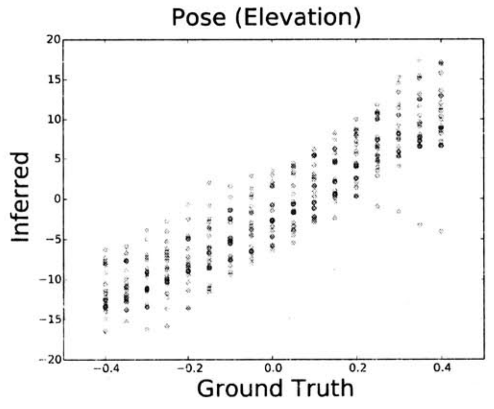
5.4.1 3D Face Dataset

The decoder network learns an approximate rendering engine as shown in Figures (5-4,5-6). Given a static test image, the encoder network produces the latents Z depicting scene variables such as light, pose, shape etc. Similar to an off-the-shelf rendering engine, we can independently control these to generate new images with the decoder. For example, as shown in Figure 5-6, given the original test image, we can vary the lighting of an image by keeping all the other latents constant and varying z_{light}. It is perhaps surprising that the fully-trained decoder network is able to function as a 3D rendering engine.

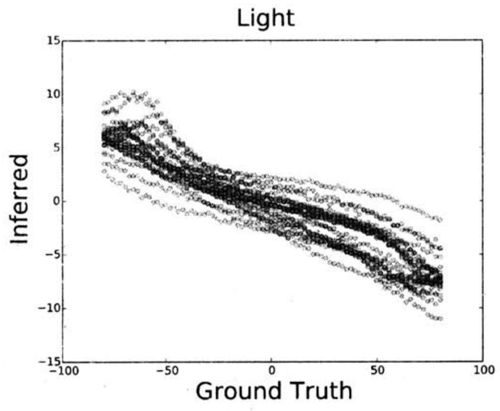
We also quantitatively illustrate the network's ability to represent pose and light



(a)



(b)



(c)

Figure 5-5: **Generalization of decoder to render images in novel viewpoints and lighting conditions:** We generated several datasets by varying light, azimuth and elevation, and tested the invariance properties of DC-IGN's representation Z. We show quantitative performance on three network configurations as described in section 5.4.1. (a,b,c) All DC-IGN encoder networks reasonably predicts transformations from static test images. Interestingly, as seen in (a), the encoder network seems to have learnt a *switch* node to separately process azimuth on left and right profile side of the face.

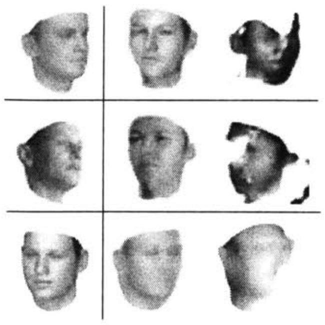


Figure 5-6: **Entangled versus disentangled representations. First column:** Original images. **Second column:** transformed image using DC-IGN. **Third column:** transformed image using normally-trained network.

on a smooth linear manifold as shown in Figure 5-5, which directly demonstrates our training algorithm's ability to disentangle complex transformations. In these plots, the inferred and ground-truth transformation values are plotted for a random subset of the test set. Interestingly, as shown in Figure 5-5(a), the encoder network's representation of azimuth has a discontinuity at $0°$ (facing straight forward).

Comparison with Entangled Representations

To explore how much of a difference the DC-IGN training procedure makes, we compare the novel-view reconstruction performance of networks with entangled representations (baseline) versus disentangled representations (DC-IGN). The baseline network is identical in every way to the DC-IGN, but was trained with SGVB without using our proposed training procedure. As in Figure 5-4, we feed each network a single input image, then attempt to use the decoder to re-render this image at different azimuth angles. To do this, we first must figure out which latent of the entangled representation most closely corresponds to the azimuth. This we do rather simply. First, we encode all images in an azimuth-varied batch using the baseline's encoder. Then we calculate the variance of each of the latents over this batch. The latent with the largest variance is then the one most closely associated with the azimuth of the face, and we call it $z_{azimuth}$. Once that is found, the latent $z_{azimuth}$ is varied for both the models to render a novel view of the face given a single image of that face. Figure

5-6 shows that explicit disentanglement is critical for novel-view reconstruction.

5.4.2 Chair Dataset

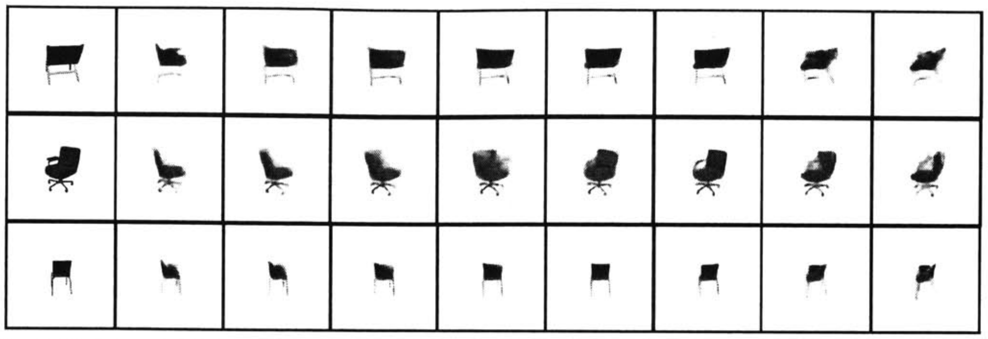
We performed a similar set of experiments on the 3D chairs dataset described above. This dataset contains still images rendered from 3D CAD models of 1357 different chairs, each model skinned with the photographic texture of the real chair. Each of these models is rendered in 60 different poses; at each of two elevations, there are 30 images taken from 360 degrees around the model. We used approximately 1200 of these chairs in the training set and the remaining 150 in the test set; as such, the networks had never seen the chairs in the test set from any angle, so the tests explore the networks' ability to generalize to arbitrary chairs. We resized the images to 150×150 pixels and made them grayscale to match our face dataset.

We trained these networks with the azimuth (flat rotation) of the chair as a disentangled variable represented by a single node z_1; all other variation between images is undifferentiated and represented by $z_{[2,200]}$. The DC-IGN network succeeded in achieving a mean-squared error (MSE) of reconstruction of 2.7722×10^{-4} on the test set. Each image has grayscale values in the range $[0, 1]$ and is 150×150 pixels.

In Figure 5-7 we have included examples of the network's ability to re-render previously-unseen chairs at different angles given a single image. For some chairs it is able to render fairly smooth transitions, showing the chair at many intermediate poses, while for others it seems to only capture a sort of "keyframes" representation, only having distinct outputs for a few angles. Interestingly, the task of rotating a chair seen only from one angle requires speculation about unseen components; the chair might have arms, or not; a curved seat or a flat one; etc.

5.5 Discussion

We have shown that it is possible to train a deep convolutional inverse graphics network with a fairly disentangled, interpretable graphics code layer representation from static images. By utilizing a deep convolution and de-convolution architecture


(a)

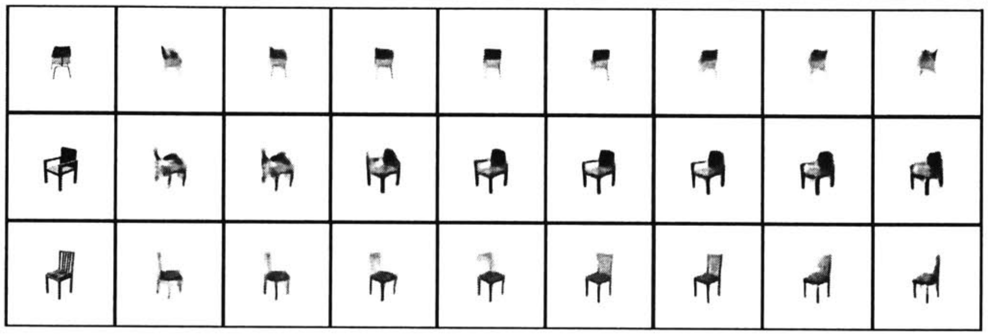

(b)

Figure 5-7: **Manipulating rotation:** Each row was generated by encoding the input image (leftmost) with the encoder, then changing the value of a single latent and putting this modified encoding through the decoder. The network has never seen these chairs before at any orientation. **(a)** Some positive examples. Note that the DC-IGN is making a conjecture about any components of the chair it cannot see; in particular, it guesses that the chair in the top row has arms, because it can't see that it doesn't. **(b)** Examples in which the network extrapolates to new viewpoints less accurately.

within a variational autoencoder formulation, our model can be trained end-to-end using back-propagation on the stochastic variational objective function [72]. We proposed a training procedure to force the network to learn disentangled and interpretable representations. Using 3D face and chair analysis as a working example, we have demonstrated the invariant and equivariant characteristics of the learned representations.

Chapter 6

Conclusion

6.1 Closing Remarks

By combining ideas from deep learning, reinforcement learning and probabilistic generative models, I have presented models and algorithms that produce compact descriptions of visual scenes and learn control policies from raw experiences. Fully autonomous agents that learn to solve many goals in many environments is the holy grail of AI research. There are several technical and conceptual challenges that lie ahead - the principles of unsupervised learning, origin of goals and intrinsic motivation, and the key principles underlying learning and representations in humans.

A first step towards this goal is to build machines than learn to perceive and use their new found capabilities to control their environment. As I described throughout this thesis, there are two broad frameworks to think about perception – a top-down view which emphasizes causal models of vision and a bottom-up view, which emphasizes learning feature based representations under some object function(s). As shown in Chapters 4&5, these two views can also be unified via the use of bottom-up models to amortize inference in top-down models.

For all upstream cognitive tasks such as policy learning, scene understanding, recognition, language-grounding, it seems essential to build systems that learn compositional representations. [83]. Top-down models make it easy to express arbitrary computational structures. This allows the programmer to express flexible prior knowledge into these systems. This is one the biggest strengths of top-down approaches, leading to the probabilistic renaissance in AI since the 1980s. However, this flexibility and the need for human in the loop comes at a price – once you define a model, it is very hard for the agent to manage and further improves it's representations for future learning.

In the cognitive science literature, the use of bayesian non-parametrics and probabilistic programming was motivated by similar concerns. These approaches advocate for non-parametric approaches that grow their structures with observations. However, inference in such models is often intractable. It is also very difficult to specify a universal probabilistic program and perform inference to learn in arbitrary environments. In my view, research in deep neural networks tries to address both these problems – the ability to express arbitrary computations and efficient credit-assignment due to differentiability. However, it is very difficult to specify prior knowledge in this setting and learning is often done from scratch for every new tasks. This seems like an important direction for deep learning research.

We are far from building general reinforcement learning agents that effectively share and generalize knowledge across tasks. It is hard to predict when and how AI will arrive. However, the history of machine learning, AI and my graduate career made me realize an important lesson – any promising path towards intelligence must scale with computation. Unifying deep learning, reinforcement learning and generative models might stand a real chance to modestly contribute towards artificial general intelligence, as they often seem to scale gracefully with more computation [123, 77, 94].

6.2 Future Directions

Existing learning systems face two fundamental challenges: continually creating temporal abstractions to scale up exploration in reinforcement learning approaches and learning useful representations of experiences without hand-crafted labels or rewards.

I finish by laying out some concrete problems to make progress in this direction:

6.2.1 Scaling up object-based subgoal discovery

Competence based intrinsic motivation can be scaled up by taking help from computer vision. In particular, to scale up goal-driven exploration, visual representations such as objects could facilitate progress in this direction. One obvious path is to train a vision system to predict many different targets on many different datasets and environments – object localization, categorization, segmentation, depth prediction, motion estimation, etc. This might enable us to abstract away unseen environments due to the universality of such targets and construct pseudo-rewards in this space of representations.

In conjunction, we should also develop object based deep generative models. Regardless of the underlying object model, an algorithm to train such a system would consist of the following steps:

(1) Initialize h-DQN's hyper-parameters.

(2) Collect frames using the current policy.

(3) Use a motion segmentation algorithms to get plausible object candidates. An alternative is to use deep generative models that produce object-based representations [31]. Another alternative is to train object localizers on different datasets or environments.

(4) Take the object proposals and train a CNN to predict an entity detector. Pick positive examples as the object candidates and random patches as the 'non-entity' class.

(5) Apply CNN on current frame obtained from the environment to obtain a set of entities.

(6) *Causal Inference:* Collect all moving entities. For each of these entities, do a causality test on which of these entities is controllable by the provided actions. This is the *Self*.

(7) Given the *Self* entity and all other entities in the current scene, the agent now has access to many subgoal candidates. Define intrinsic rewards in this space.

(8) Train h-DQN using algorithm 4. **Goto (1)**.

6.2.2 Deep Hierarchical Successor Reinforcement Learning

A promising research direction is to combine DSR with h-DQN. As demonstrated in chapter 3, it is possible to extract subgoals after learning the SR representation. Every so often, a few candidate subgoals can be extracted and stored into a short term memory M. The meta-controller in h-DQN can choose any of the following actions: (1) choose a subgoal from M, (2) delete entries from M. The meta-controller can also output a probability indicating whether the current option should terminate or continue to execute, similar to intra-options [137]. To further minimize sample complexity during training, DSR can be extended to incorporate explicit model-based reasoning. In principle, a simple modification to the existing architecture could enable this – let the decoder network predict the next frames along with the current frame. However, learning deep generative models of images is an open problem in deep learning. Any advances in this research topic will create a big impact on reinforcement learning algorithms.

Another interesting possibility is to build a deep hierarchy of successor based Q-networks. Extrinsic rewards come at the highest level, where the network outputs a reward vector as an action for the downstream successor Q-network to solve. The bottom most Q-network outputs actions. This entire hierarchy can be trained end-to-end in a manner similar to h-DQN. Every reward vector outputted in the intermediate stages of the hierarchy can be thought of as subgoals. However, the semantics of the subgoals are not pre-specified and would be learned automatically by the network.

6.2.3 Deep Temporal Compressor

A good representation of the visual world can often make the upstream task of policy learning relatively more easy. I end by attempting to sketch an unsupervised learning algorithm to obtain representations of videos. The first few frames that an agent ever sees are highly informative due to new information. Subsequent few frames would carry much less information under a spatio-temporal continuity assumption. This is a very reasonable assumption of our world (in the local context). Modern video compression algorithms like MPEG already make use of this inductive bias. In fact, Zhou et al. [161] used a similar assumption for multi-view synthesis of objects.

A desirable deep generative model would share a lot of characteristics with video compression algorithms like MPEG. Given the current x_t, we can predict the next frame x_{t+1} by copying as many pixels as possible and generating only the residual error. Zhou et al.'s model does not generate new pixels and it is important to do this for dynamic scenes. However, we must ensure that the generator and copy operator are weighted proportionally. We expect the copy operator to do most of the work and the generator should only need to explain partial pixels under the spatio-temporal continuity assumption. We can enforce this by putting an information content penalty on both the channels so that most of the generator pixels turn out to be zero. Starting with such inductive biases seems like a promising start towards building more human like vision systems.

Appendix A

A.1 3D medially-symmetric object reconstruction program

As illustrated in Figure A-1, the height H of the object is sampled from a uniform distribution. 3D Objects can consist of several sub-parts. Without the loss of generality and for simplicity, we study objects with up-to two sub-parts and circular cross-section. We sample a cut C along the medial axis of the object using the beta distribution, resulting in two independent GPs spanning the cut proportions. Since the smoothness and profile of 3D objects is a priori unknown, we need to do hyper-parameter inference on the bandwidths L_1 and L_2 of the covariance kernel of the GPs. The resulting points $f_{\mathcal{GP}}(x)$ from GPs are passed to the graphics simulator for lathing based mesh generation, which results in the generation of I_R. During inference, reconstructing 3D objects amounts to calculating the posterior $P(S = \{H, C, L_1, L_2\}|I_D)$. While collecting results, we found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses. For visualizing the underlying stochastic process, refer to supplemental video.

The 3D shape program in Figure A-3 could then be roughly formalized as follows:

$$H \sim Uniform(a_0, b_0) \text{ and } C \sim a_c + b_c * Beta(1, H)$$

$$L_1 \sim a_1 + b_1 Beta(2, 5),\; x_1 = [a_0, C]$$

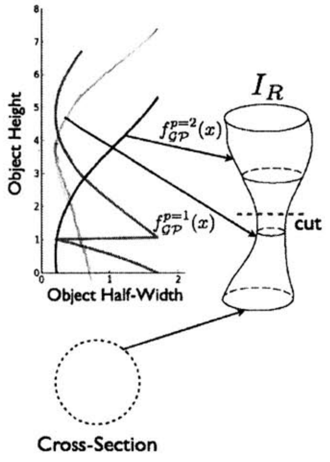


Figure A-1: **3D Shape Program Visualization**

$$L_2 \sim a_1 + b_1 Beta(2,5),\ x_2 = [C+1, b_0]$$

$$k(x_p^i, x_p^j) = \exp\Big(-\frac{(x_p^i - x_p^j)}{2L_1^2}\Big)$$

where $p \in \{1,2\}$ denote parts and

$$\min(x_p) \le (i,j) \le \max(x_p)$$

$$f_{\mathcal{GP}}^{p}(x) = \begin{cases} \text{normalize}(\mathcal{GP}(0, k(x_p^i, x_p^{i'}))) & 0 \le \text{length}(x_p) \\ 0 & \text{otherwise} \end{cases}$$

$$I_R = g_{LATHE}(\{f_{\mathcal{GP}}^1(x_1^m)\}, \{f_{\mathcal{GP}}^2(x_2^n)\})$$

where $min(x_1) \le m \le max(x_1)$ and $min(x_2) \le n \le max(x_2)$.

A.2 3D Human Pose Program

It is important to note that this particular program faces a lot of difficulty under clutter mainly due to lack of robust bottom-up features. Moreover, the graphics program could be significantly improved by using BlendSCAPE[57] model along with approximately reasoning about people's clothes for more accurate body-part localization. Please refer to Figure A-4 to view the probabilistic program. While collecting results, we

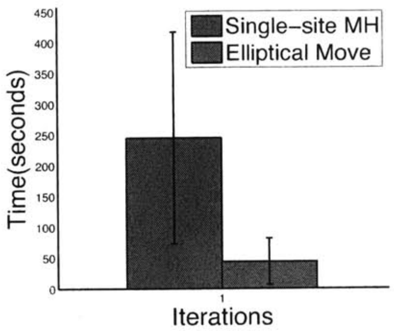


Figure A-2: **Inference run-time comparison for face program:** We ran 30 independent inference runs each by toggling the inference scheme between single-site metropolis hastings and elliptical slice proposals. Elliptical moves give significant speedup to reach a certain level of score, which is expected as single-site updates will scale linearly with dimensionality of latents.

found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses.

A.3 Generative Face Program

Since the dimensionality of the face program is high (8 sets of 100 dimensional continuous coupled latent variables), single-site metropolis hastings algorithm is highly inefficient as the number of simulation updates scale linearly with dimensionality of latents. Picture allows us to easily swap inference scheme, which enabled us to quickly discover that elliptical slice moves are significantly more efficient than Metropolis-Hastings proposals (see Figure A-2). While collecting results, we found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses.

```
function GP(xs,L)
  cov = zeros(length(xs),length(xs))
  mu = zeros(length(xs))
  for i=1:length(xs)
    for j=1:length(xs)
      cov = exp(-(xs[i]-xs[j])^2/(2*l^2))
    end
  end
  return mu,cov
end

function PROGRAM(observation_dt)
  RES = 120
  height = Uniform(5,7.8,1,1)
  xs = [0:height/RES:height]
  samples = zeros(length(xs))

  cut_var = Beta(1,5,1,1)
  cut1,cut2 = sample_cut(xs,height,cut_var,RES)
  l1 = 5*Beta(2,5,1,1)
  l2 = 5*Beta(2,5,1,1)

  if length(cut1)>0
    mu,cov = GP(xs[cut1],l1)
    samples[cut1]=MvNormal(mu, cov)
    samples[cut1] = normalize(samples[cut1])
  end

  if length(cut2)>0
    mu,cov = GP(xs[cut2],l2)
    samples[cut2]=MvNormal(mu, cov)
    samples[cut2] = normalize(samples[cut2])
  end
  #camera:[scale,rotation,translation]
  camera = [Normal(0.98,0.1,1,3), Normal(0,5,1,3),
           Uniform(-1,1,1,3)]
  #Call Blender API
  render = render("profile", samples, "cross-sec", xs, "camera", camera)
  edgemap = canny(rendering,1.0);
  valid_indxs = np.where(edgemap>0)
  D = np.multiply(observation_distance_transform, rendering)
  observe(0,Normal(0,0.35),D)
end

global observation_distance_transform
edge_img = canny(imread("test.png"),1.0)
observation_distance_transform = scipy.distance_transform_bf(edge_img)

TR = trace(PROGRAM,[])
infer(TR, debug_callback,100,"MCMC")
```

Figure A-3: **Picture code for 3D Object Reconstruction via Lathing:** Gaussian Process based 3D reconstruction program of lathe objects. This program samples 3D shapes with two independent sub-parts. We used probabilistic chamfer distance as the stochastic comparator.

```
function render(hip_location,...,camera)
 args = [hip_location,...,camera]
 blender.println("cmd:skin-modifier")
 rendered_image = blender.println(args)
 return rendered_image
end

function PROGRAM()
 sigma_0 = Uniform(10,50,1,1)
 hip_location = Uniform(-0.35,0,1,3)
 elbowR_rotation = Normal(0,sigma_0,1,3); elbowR_location = Uniform(-1,1,1,3)
 elbowL_rotation = Normal(0,sigma_0,1,3); elbowL_location = Uniform(-1,1,1,3)
 heelL_location = Uniform(-0.1,0.45,1,3); heelR_location = Uniform(-0.1,0.45,1,3)

 #camera:[scale, rotation, translation]
 camera = [Normal(0.98,0.1,1,1), Normal(0,5,1,3), Uniform(-1,1,1,2)]

 #Call Blender API
 rendering = render("bone-id0", hip_location, ... ,...,"camera", camera)

 edgemap = canny(rendering,1.0);
 valid_indxs = np.where(edgemap>0)
 D = np.multiply(observation_distance_transform[valid_indxs], rendering[valid_indxs])
 observe(0,Normal(0,0.35),D)
end

global observation_distance_transform
edge_img = canny(imread("test.png"),1.0)
observation_distance_transform = scipy.distance_transform_bf(edge_img)

TR = trace(PROGRAM,[])
infer(TR, debug_callback,100,"MCMC")
```

Figure A-4: **Picture code for 3D Human Pose:** This program use an existing base mesh of a human body, defines priors over bone location and joints, and enables armature skin-modifier[14] via Picture's Blender engine API. We used probabilistic chamfer distance as the comparator.

Bibliography

[1] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive science*, 9(1):147–169, 1985.

[2] Oswald Aldrian and William AP Smith. Inverse rendering of faces with a 3d morphable model. *PAMI*, 2013.

[3] Mathieu Aubry, Daniel Maturana, Alexei Efros, Bryan Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014.

[4] Melinos Averkiou, Vladimir G Kim, Youyi Zheng, and Niloy J Mitra. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. In *Computer Graphics Forum*, volume 33, pages 125–134. Wiley Online Library, 2014.

[5] Jonathan Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. Technical report, Berkeley Tech Report, 2013.

[6] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[7] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

[8] Marc G Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *arXiv preprint arXiv:1606.01868*, 2016.

[9] Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[10] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

[12] JM Bernardo, JO Berger, AP Dawid, AFM Smith, et al. Regression and classification using gaussian process priors. In *Bayesian Statistics 6: Proceedings of the sixth Valencia international meeting*, volume 6, page 475, 1998.

[13] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194. ACM Press/Addison-Wesley Publishing Co., 1999.

[14] Blender Online Community. *Blender - a 3D modelling and rendering package.* Blender Foundation, Blender Institute, Amsterdam,

[15] Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1655):20130480, 2014.

[16] Matthew M Botvinick, Yael Niv, and Andrew C Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

[17] Taco Cohen and Max Welling. Learning the irreducible representations of commutative lie groups. *arXiv preprint arXiv:1402.4437*, 2014.

[18] Dane S Corneil and Wulfram Gerstner. Attractor network dynamics enable preplay and rapid path planning in maze–like environments. In *Advances in Neural Information Processing Systems*, pages 1675–1683, 2015.

[19] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24, 2001.

[20] Nathaniel D Daw and Peter Dayan. The algorithmic anatomy of model-based evaluation. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 369(1655):20130478, 2014.

[21] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

[22] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–271. Morgan Kaufmann Publishers, 1993.

[23] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

[24] L Del Pero, J Bowdish, Daniel Fried, B Kermgard, E Hartley, and Kobus Barnard. Bayesian geometric modeling of indoor scenes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2719–2726. IEEE, 2012.

[25] Luca Del Pero, Joshua Bowdish, Bonnie Kermgard, Emily Hartley, and Kobus Barnard. Understanding bayesian rooms using composite 3d object models. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 153–160. IEEE, 2013.

[26] Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Disentangling factors of variation via generative entangling. *arXiv preprint arXiv:1210.5474*, 2012.

[27] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)*, 13:227–303, 2000.

[28] Carlos Diuk, Andre Cohen, and Michael L Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 240–247. ACM, 2008.

[29] Piotr Dollár and C Lawrence Zitnick. Structured forests for fast edge detection. 2013.

[30] A. Dosovitskiy, J. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *arXiv:1411.5928*, 2015.

[31] SM Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.

[32] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *PAMI*, 2010.

[33] Noa Fish, Melinos Averkiou, Oliver Van Kaick, Olga Sorkine-Hornung, Daniel Cohen-Or, and Niloy J Mitra. Meta-representation of shape families. In *Computer Graphics Forum*, volume 32, pages 189–200, 2013.

[34] Katerina Fragkiadaki, Pablo Arbelaez, Panna Felsen, and Jitendra Malik. Learning to segment moving objects in videos. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4083–4090. IEEE, 2015.

[35] Mikhail Frank, Jürgen Leitner, Marijn Stollenga, Alexander Förster, and Jürgen Schmidhuber. Curiosity driven reinforcement learning for motion planning on humanoids. *Intrinsic motivations and open-ended development in animals, humans, and robots*, page 245, 2015.

[36] Samuel J Gershman, Eric J Horvitz, and Joshua B Tenenbaum. Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245):273–278, 2015.

[37] Samuel J Gershman, Christopher D Moore, Michael T Todd, Kenneth A Norman, and Per B Sederberg. The successor representation and temporal context. *Neural Computation*, 24(6):1553–1568, 2012.

[38] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[39] Sandeep Goel and Manfred Huber. Subgoal discovery for hierarchical reinforcement learning using learned policies. In *FLAIRS conference*, pages 346–350, 2003.

[40] Ian Goodfellow, Honglak Lee, Quoc V Le, Andrew Saxe, and Andrew Y Ng. Measuring invariances in deep networks. In *Advances in neural information processing systems*, pages 646–654, 2009.

[41] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[42] Klaus Greff, Rupesh Kumar Srivastava, and Jürgen Schmidhuber. Binding via reconstruction clustering. *arXiv preprint arXiv:1511.06418*, 2015.

[43] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[44] Ulf Grenander. *General pattern theory-A mathematical study of regular structures.* Clarendon Press, 1993.

[45] Ulf Grenander, Yun-shyong Chow, and Daniel M Keenan. *Hands: A pattern theoretic study of biological shapes.* Springer-Verlag New York, Inc., 1991.

[46] Peng Guan, Alexander Weiss, Alexandru O Balan, and Michael J Black. Estimating human shape and pose from a single image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1381–1388. IEEE, 2009.

[47] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, pages 399–468, 2003.

[48] Abhinav Gupta, Alexei A Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *Computer Vision–ECCV 2010*, pages 482–496. Springer, 2010.

[49] Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general atari game playing. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(4):355–366, 2014.

[50] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[51] Varsha Hedau, Derek Hoiem, and David Forsyth. Thinking inside the box: Using appearance models and context based on room geometry. In *Computer Vision–ECCV 2010*, pages 224–237. Springer, 2010.

[52] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[53] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[54] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[55] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 44–51. Springer, 2011.

[56] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[57] David A Hirshberg, Matthew Loper, Eric Rachlin, and Michael J Black. Coregistration: Simultaneous alignment and modeling of articulated 3d shape. In *ECCV*. 2012.

[58] Derek Hoiem, Alexei A Efros, and Martial Hebert. Putting objects in perspective. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2137–2144. IEEE, 2006.

[59] Derek Hoiem, Alexei A Efros, and Martial Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75(1):151–172, 2007.

[60] Marc W Howard and Michael J Kahana. A distributed representation of temporal context. *Journal of Mathematical Psychology*, 46(3):269–299, 2002.

[61] Jonathan Huang and Kevin Murphy. Efficient inference in occlusion-aware generative models of images. *arXiv preprint arXiv:1511.06362*, 2015.

[62] John F Hughes, Andries Van Dam, James D Foley, and Steven K Feiner. *Computer graphics: principles and practice*. Pearson Education, 2013.

[63] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2008–2016, 2015.

[64] Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *arXiv preprint arXiv:1402.0859*, 2014.

[65] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[66] Ya Jin and Stuart Geman. Context and hierarchy in a probabilistic image model. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2145–2152. IEEE, 2006.

[67] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[68] Takeo Kanade. Recovery of the three-dimensional shape of an object from a single view. *Artificial intelligence*, 17(1):409–460, 1981.

[69] Gaetano Kanizsa. Subjective contours. *Scientific American*, 234(4):48–52, 1976.

[70] Ira Kemelmacher-Shlizerman and Ronen Basri. 3d face reconstruction from a single image using a single reference face shape. *PAMI*, 2011.

[71] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.

[72] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

[74] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.

[75] Jan Koutník, Jürgen Schmidhuber, and Faustino Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 541–548. ACM, 2014.

[76] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[77] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[78] 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*, pages 4390–4399, 2015.

[79] Tejas D Kulkarni, Vikash K Mansinghka, Pushmeet Kohli, and Joshua B Tenenbaum. Inverse graphics with probabilistic cad models. *arXiv preprint arXiv:1407.1339*, 2014.

[80] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057*, 2016.

[81] Tejas D Kulkarni, Ardavan Saeedi, and Samuel Gershman. Variational particle approximations. *arXiv preprint arXiv:1402.5715*, 2014.

[82] Tejas D Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B Tenenbaum. Deep convolutional inverse graphics network. *arXiv preprint arXiv:1503.03167*, 2015.

[83] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*, 2016.

[84] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[85] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[86] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[87] Mun Wai Lee and Isaac Cohen. A model-based approach for estimating human 3d poses in static images. *PAMI*, 2006.

[88] Martin D Levine and Yingfeng Yu. State-of-the-art of 3d facial reconstruction methods for face recognition based on a single 2d training image per person. *Pattern Recognition Letters*, 30(10):908–913, 2009.

[89] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *ECCV 2014*. 2014.

[90] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[91] Vikash 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*, pages 1520–1528, 2013.

[92] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

[93] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[94] 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. *Nature*, 518(7540):529–533, 2015.

[95] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2116–2124, 2015.

[96] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[97] Iain Murray, Ryan Prescott Adams, and David JC MacKay. Elliptical slice sampling. *arXiv preprint arXiv:1001.0175*, 2009.

[98] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[99] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.

[100] Radford Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

[101] Louis Albert Necker. Lxi. observations on some remarkable optical phænomena seen in switzerland; and on an optical phænomenon which occurs on viewing a figure of a crystal or geometrical solid. 1832.

[102] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*, 2016.

[103] Pierre-Yves Oudeyer, Frederic Kaplan, et al. How can we define intrinsic motivation. In *Proc. 8th Int. Conf. Epigenetic Robot.: Modeling Cogn. Develop. Robot. Syst*, 2008.

[104] Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. *arXiv preprint arXiv:1403.0504*, 2014.

[105] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter. A 3d face model for pose and illumination invariant face recognition. Genova, Italy, 2009. IEEE.

[106] Judea Pearl. Graphical models for probabilistic and causal reasoning. In *Quantified Representation of Uncertainty and Imprecision*, pages 367–389. Springer, 1998.

[107] Tomaso Poggio, Jim Mutch, Joel Leibo, Lorenzo Rosasco, and Andrea Tacchetti. The computational magic of the ventral stream: sketch of a theory (and why some deep architectures work). 2012.

[108] M Ranzato, Fu Jie Huang, Y-L Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[109] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[110] Samuel Rathmanner and Marcus Hutter. A philosophical treatise of universal induction. *Entropy*, 13(6):1076–1136, 2011.

[111] Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. *arXiv preprint arXiv:1603.05106*, 2016.

[112] Daniel Ritchie, Ben Mildenhall, Noah D. Goodman, and Pat Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM Trans. Graph.*, 34(4):105:1–105:11, July 2015.

[113] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. Labelme: a database and web-based tool for image annotation. *IJCV*, 2008.

[114] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[115] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1312–1320, 2015.

[116] Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence*, pages 199–226. Springer, 2007.

[117] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *Autonomous Mental Development, IEEE Transactions on*, 2(3):230–247, 2010.

[118] Nicol N Schraudolph, Peter Dayan, and Terrence J Sejnowski. Temporal difference learning of position evaluation in the game of go. *Advances in Neural Information Processing Systems*, pages 817–817, 1994.

[119] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *ICPR*, 2004.

[120] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

[121] F Schumann. Beitrlge zur analyse der gesichtswahrnehmungen. erste abhandlung. einige beobachtungen ilber die zusammenfassung von gesichtseindriicken zu einheiten. *Z. Psychol*, 23:1–32, 1900.

[122] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[123] 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. *Nature*, 529(7587):484–489, 2016.

[124] Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pages 816–823. ACM, 2005.

[125] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606, 2009.

[126] Satinder Singh, Richard L Lewis, Andrew G Barto, and Jonathan Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *Autonomous Mental Development, IEEE Transactions on*, 2(2):70–82, 2010.

[127] Satinder P Singh, Andrew G Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2004.

[128] Jonathan Sorg and Satinder Singh. Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 31–38, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[129] Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

[130] Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. Design principles of the hippocampal cognitive map. In *Advances in neural information processing systems*, pages 2528–2536, 2014.

[131] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[132] Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. In *Advances in neural information processing systems*, pages 3048–3056, 2013.

[133] Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

[134] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning.* MIT Press, 1998.

[135] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[136] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[137] Richard S Sutton, Doina Precup, and Satinder P Singh. Intra-option learning about temporally abstract actions. In *ICML*, volume 98, pages 556–564, 1998.

[138] Csaba Szepesvari, Richard S Sutton, Joseph Modayil, Shalabh Bhatnagar, et al. Universal option models. In *Advances in Neural Information Processing Systems*, pages 990–998, 2014.

[139] Yichuan Tang, Ruslan Salakhutdinov, and Geoffrey Hinton. Deep lambertian networks. *arXiv preprint arXiv:1206.6445*, 2012.

[140] Arasanathan Thayananthan, Bjoern Stenger, Philip HS Torr, and Roberto Cipolla. Shape context and chamfer matching in cluttered scenes. In *CVPR*, 2003.

[141] Edward L Thorndike. The law of effect. *The American Journal of Psychology*, 39(1/4):212–222, 1927.

[142] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning. 2012.

[143] Tijmen Tieleman. *Optimizing Neural Networks that Generate Images.* PhD thesis, University of Toronto, 2014.

[144] Zhuowen Tu, Xiangrong Chen, Alan L Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *IJCV*, 2005.

[145] Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5), May 2002.

[146] Zhuowen Tu and Song-Chun Zhu. Image segmentation by data-driven markov chain monte carlo. *PAMI*, 24(5):657–673, 2002.

[147] Joel Veness, Kee Siong Ng, Marcus Hutter, and David Silver. Reinforcement learning via aixi approximation. *arXiv preprint arXiv:1007.2049*, 2010.

[148] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[149] Hermann von Helmholtz and James Powell Cocke Southall. *Treatise on physiological optics*, volume 3. Courier Corporation, 2005.

[150] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[151] William F Whitney, Michael Chang, Tejas Kulkarni, and Joshua B Tenenbaum. Understanding visual concepts with continuation learning. *arXiv preprint arXiv:1602.06822*, 2016.

[152] Richard David Wilkinson. Approximate bayesian computation (abc) gives exact results under the assumption of model error. *Statistical applications in genetics and molecular biology*, 12(2):129–141, 2013.

[153] David Wingate, Noah D Goodman, A Stuhlmueller, and J Siskind. Nonstandard interpretations of probabilistic programs for efficient inference. *NIPS*, 23, 2011.

[154] David Wingate, Andreas Stuhlmueller, and Noah D Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *International Conference on Artificial Intelligence and Statistics*, pages 770–778, 2011.

[155] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *AISTATS*, 2014.

[156] Jimei Yang, Scott E Reed, Ming-Hsuan Yang, and Honglak Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Advances in Neural Information Processing Systems*, pages 1099–1107, 2015.

[157] Yi Yang and Deva Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR*, 2011.

[158] Ilker Yildirim, Tejas D Kulkarni, Winrich A Freiwald, and Joshua B Tenenbaum. Efficient and robust analysis-by-synthesis in vision: A computational framework, behavioral tests, and modeling neuronal representations.

[159] Yinda Zhang, Shuran Song, Ping Tan, and Jianxiong Xiao. Panocontext: A whole-room 3d context model for panoramic scene understanding. In *Computer Vision–ECCV 2014*, pages 668–686. Springer, 2014.

[160] Yibiao Zhao and Song-Chun Zhu. Image parsing via stochastic scene grammar. In *NIPS*, 2011.

[161] Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. *arXiv preprint arXiv:1605.03557*, 2016.

[162] Jasenko Zivanov, Andreas Forster, Sandro Schonborn, and Thomas Vetter. Human face shape analysis under spherical harmonics illumination considering self occlusion. In *Biometrics (ICB), 2013 International Conference on*, pages 1–8. IEEE, 2013.