Prediction and Planning in Dynamical Systems with Underlying Markov Decision Processes

by

SeyedErshad Banijamali

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Matthew E. Taylor, Associate Professor
Department of Computer Science
University of Alberta

Supervisor: Ali Ghodsi, Professor
Department of Statistics and Actuarial Science
University of Waterloo

Internal Member: Pascal Poupart, Professor
School of Computer Science
University of Waterloo

Jesse Hoey, Professor
School of Computer Science
University of Waterloo

Internal-External Member: Fakhri Karray, Professor
Department of Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Main part of this dissertation is based on some publications that I have co-authored. In particular, Chapter 2 is based on [4], Chapter 3 is based on [7, 8], Chapter 4 is based on [11, 9], and finally, Chapter 5 is based on [10].

Abstract

Predicting the future state of a scene with moving objects is a task that humans handle with ease. This is due to our understanding about the dynamics of the objects in the scene and the way they interact. However, teaching machines such understanding has always been a challenging task in machine learning. In recent years, with the abundance of data and enormous growth in computational power, there have been an outstanding progress in filling the gap between humans and machines perception and prediction. Deep learning, specifically, has been the main framework to address this problem.

Prediction models are not only crucial problems by themselves but also many downstream tasks in machine learning and robotics rely on the quality of output of these models. Model-based control and planning require an accurate modelling of the underlying dynamics of the systems. A common assumption about the underlying dynamics, which is also the main theme of this thesis, is that it can be expressed using Markov Decision Processes (MDPs). However, the major portion of the thesis is dedicated to the problems in which we do not have access to the actual underlying MDP and only observe some high-dimensional observations from the dynamical system. The objective is then to model the underlying dynamics from the data and built a model that can potentially be used for planning and control. We consider both single-agent and multi-agent systems and employ deep generative models for modelling the dynamics. For the single-agent problem we propose a model that maps the high-dimensional observations to a low-dimensional space in which the dynamics of the system is modelled by a locally-linear function. We find this mapping by a proper modelling of the variables using graphical models and show that the mapping is robust against dynamics noise and suitable for control. For the multi-agent problem we provide a formulation that describes the prediction problem in terms of the reaction of the environment to the action of one agent (ego-agent) and show that such formulation can improve the prediction accuracy as well as broaden the range of environment conditions. From a different perspective, we also consider the problem in which we have access to the MDP and would like to obtain the optimal policy. More specifically, given a set of base policies on the

MDP, we want to find the best policy in their convex hull. We show that this problem is NP-hard in general and provide an approximating algorithm with linear complexity, which outputs a policy that performs close to the optimal policy. This policy can be found under the condition that base policies have overlap in the occupancy measure space.

Acknowledgements

First and foremost, I want to express my gratitude to my supervisor, Ali Ghodsi, who has always encouraged me to explore the problems I am interested in. His immense knowledge in different research fields has inspired me to expand my horizons and develop new skills. I truly appreciate all his time, patience, ideas, discussions, and advises.

I would like to thank Mohammad Ghavamzadeh and Yasin Abbasi-Yadkori for their tremendous help in solving problems that form some parts of this thesis. I also thank Pascal Poupart for the discussions and his great technical advises.

During the years that I was in Waterloo I had been lucky to spend time with incredible friends. They always made every situation easier and more enjoyable.

Above all, I devote my deepest gratitude to my family for their unconditional love and support throughout my life and especially during this period. Their endless devotion and patience guided me in this journey.

Dedication

To my beloved parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Planning and sequential decision making is a crucial task in machine learning and more generally artificial intelligence with applications in many domains. Teaching machines how to plan and act independently in the field is probably the biggest step towards building intelligent agents. With the abundance of data and enormous resources devoted to the research in the area of machine learning, especially in the last two decades, there has been great advancements in developing algorithms and building systems that can carry out this task. However in many areas such as autonomous driving we are still far from being done.

There are two major approaches to solve planning problems: Model-based algorithms and model-free algorithm. In model-free algorithms, there is no explicit model for transition or reward, and policy is directly optimized by interaction with the environment. In model-based algorithms, on the other hand, a model of the environment is first learned that describes the evolution of the states, and then an algorithm is designed to optimize the agent's behaviour based on the learn model. The main benefit of model-based planning is that it requires much less data, and therefore much less in-

teractions with the environment. This is especially important for the problems in which interaction with environment is costly, e.g. autonomous driving.

Throughout this dissertation, we assume that the interaction between the agent and environment can be modelled using Markov decision processes (MDPs). We consider two problems. First we assume we have access to an MDP and would like to find an optimal policy over this MDP. In the second problem, we assume there is no explicit access to an MDP and we only observe data generated using an underlying MDP in the form of sequences of interactions between and agent and environment. The goal for the second problem is to approximate the transition probability distribution of the MDP using a prediction model that can be then used for model-based planning. Our focus, however, is on finding the so-called *world-model* [37, 66].

1.2 Research Questions

In this dissertation we address two research questions that are essentially related to two different aspects of planning but are relevant in the sense that the problems can be cast in the framework of MDPs.

1.2.1 RQ1: Policy optimization over a known MDP

In the first problem, we assume there is access to the actual MDP that defines our problem and we are given a set of limited number of policies on this MDP. The goal is then to find the optimal combination of these policies in the convex hull. This problem is of great interest in applications in which a number of reasonably good (or safe) policies are already known and we are interested in optimizing in their convex hull. In chapter 2 we address this problem.

1.2.2 RQ2: Finding a prediction model from high-dimensional observation of an unknown MDP

For the second problem, we assume that we only have access to sequences of data generated from an underlying MDP in the form high-dimensional observations, e.g. in pixel space. The goal is to find a probabilistic prediction model that serves as an approximation of the transition probability distribution of the MDP. In fact, this can be seen as solving a prediction task.

The probabilistic approach for solving the prediction task is formally defined as finding the probability distribution over the next state of the system, $\mathbf{s}_{t+1}$, given the history of the past states, $\mathbf{s}_{1:t}$, and action, $\mathbf{a}_{1:t}$, of an agent (actor), i.e. $p(\mathbf{s}_{t+1}|\mathbf{s}_{1:t}, \mathbf{a}_{1:t})$. Data usually comes in large sequences of state-action-state, $\{\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, ..., \mathbf{s}_N\}$. To derive a prediction model from large sequences of data, a common approach is maximizing the log-likelihood of the whole data sequences. In fact, by maximizing the log-likelihood we can discover the generative process that produces the dataset. When the sequence of data hold Markov property, the maximization can be written in the form of summations of $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, which makes the computations much easier. From another perspective, given the Markov property, maximizing the log-likelihood is equivalent to approximating the transition probability distribution of an MDP, which can be then used for learning a policy that can control the system. The better we model the generative process behind producing the data, the closer we get to the actual transition probability distribution.

With the dawn of deep learning and enormous improvement in computational power, we have been able to process complicated data for different application. Deep generative models (DGNs), such as variational autoencoders (VAEs) [48, 79], generative adversarial networks (GANs) [33], and flow-based models [78], are powerful tool to model the distribution of data. We will be using DGNs and specifically VAEs to tackle the prediction task.

Two sets of prediction problems are investigated in this dissertation: 1) High-dimensional

observation from a single dynamical system for which the prediction problem includes understanding the dynamics of the systems using the observation and representing that dynamics in a much lower-dimensional space that can be then used for planning and control. 2) High-dimensional observation of a multi-agent dynamical system for which the prediction problem includes capturing the dynamics of the objects as well as interactions among them. We particularly consider the problem of autonomous driving.

1.3 Summary of Contributions

- **Contribution 1: Combining a finite number of policies in MDPs:** We study the problem of finding an optimal policy in an MDP under a restricted policy class defined by the convex hull of a set of base policies. We prove that solving this problem is NP-hard. We then propose an efficient algorithm that finds a policy whose performance is almost as good as that of the best convex combination of the base policies, under the assumption that the occupancy measures of the base policies have a large overlap. The running time of the proposed algorithm is linear in the number of states and polynomial in the number of base policies. A distinct advantage of the proposed algorithm is that, apart from the computation of the occupancy measures of the base policies, it does not need to interact with the environment during the optimization process. This is especially important (i) in problems that due to concerns such as safety, we are restricted in interacting with the environment only through the (safe) base policies, and (ii) in complex systems where estimating the value of a policy can be a time consuming process. In practice, we demonstrate an efficient implementation for large state problems.

- **Contribution 2: A probabilistic model for robust controllable embedding based on VAEs:** We present a new model for learning *robust* locally-linear controllable embedding (RCE) for high-dimensional observations of an underlying MDP problem. Our model directly estimates the predictive conditional density of the future observation given the current one, while introducing the bottleneck [85] between

the current and future observations. Although the bottleneck provides a natural embedding candidate for control, our RCE model introduces additional specific structures in the generative graphical model so that the model dynamics can be robustly linearized. We also propose a principled variational approximation of the embedding posterior that takes the future observation into account, and thus, makes the variational approximation more robust against the noise. Experimental results demonstrate that RCE outperforms the existing embed to control model (E2C), and does so significantly in the regime where the underlying dynamics is noisy.

- **Contribution 3: A probabilistic model for action-conditional multi-step prediction for autonomous driving based on VAEs:** We propose "Prediction by Anticipation", a method for action-conditional environment prediction for self-driving cars where the environment is represented in the form of Occupancy Grid Map (OGM). Our motivation is that accurate modelling and prediction of the driving environment can efficiently improve path planning and navigation resulting in safe, comfortable and optimum paths in autonomous driving. Due to the importance of interactions between the objects in the scene, it is important to model and predict the driving environment based on the ego-actions to be able to predict the effect of our actions on other agents decisions and behaviours. We train our model in the framework of conditional varitional autoencoders (CVAEs) to maximize the evidence lower bound (ELBO) of the log-likelihood of a conditional observation distribution. An extension of our model is also presented that explicitly learns the difference between consecutive frames and is suitable for dense urban traffic. We evaluate our model on OGM sequences from two important autonomous driving benchmarks. The results show significant improvements of the prediction accuracy using our proposed architectures over the state-of-the-art.

- **Contribution 4: Employing VAE-based models for representation learning:** As mentioned in Contributions 2 and 3, the embedding of the high-dimensional observations in done using VAEs. In Chapter 3 of this thesis we provide an overview

of these models and show case their effectiveness in two different problem, which are formulated using VAEs:

- **Sufficient dimensionality reduction:** We consider the problem of sufficient dimensionality reduction (SDR), where the high-dimensional observation is transformed to a low-dimensional sub-space in which the information of the observations regarding the label variable is preserved. We propose DVSDR, a deep variational approach for sufficient dimensionality reduction. The deep structure in our model has a bottleneck that represent the low-dimensional embedding of the data. We explain the SDR problem using graphical models and use the framework of variational autoencoders to maximize the lower bound of the log-likelihood of the joint distribution of the observation and label. We show that such a maximization problem can be interpreted as solving the SDR problem. DVSDR can be easily adopted to semi-supervised learning setting. Experiment results show that DVSDR performs competitively on classification tasks while being able to generate novel data samples.
- **Disentangle representation for classification:** We present a novel method for disentangling factors of variation in data-scarce regimes. Specifically, we explore the application of feature disentangling for the problem of supervised classification in a setting where we have access to only a few labeled samples exist, and there are no unlabeled samples for use in unsupervised training. Instead, a similar datasets exists which shares at least one direction of variation with the sample-constrained datasets. We train our model end-to-end using the framework of variational autoencoders and are able to experimentally demonstrate that using an auxiliary dataset with similar variation factors contribute positively to classification performance, yielding competitive results with the state-of-the-art.

1.4 Document Organization

The rest of this thesis is organized as follows. In Chapter 2 we address the policy mixing problem. In Chapter 3, we provide a brief introduction about VAEs and showcase their effectiveness for discovering the underlying generative features using two representation learning problems, i.e. dimensionality reduction and disentangling representations. In Chapter 4 and 5 we use VAEs to solve the prediction task we mentioned in RQ2.

Chapter 2

Optimizing over a Restricted Policy Class in Markov Decision Processes

2.1 Contributions

In this chapter we address the problem of finding an optimal policy in a Markov decision process under a restricted policy class defined by the convex hull of a set of base policies. We first prove that solving this problem is NP-hard. We then propose an efficient algorithm that finds a policy whose performance is almost as good as that of the best convex combination of the base policies, under the assumption that the occupancy measures of the base policies have a large overlap. The running time of the proposed algorithm is linear in the number of states and polynomial in the number of base policies. In practice, we demonstrate an efficient implementation for large state problems. Specifically, we consider queue problems in our experiments.

2.2 Introduction

In many control and reinforcement learning problems, a number of reasonable (safe) base policies are known. For example, these policies might be provided by an expert. A natural question is whether a combination of these base policies can provide an improvement over a default policy. This problem is especially important when the number of states is large and the exact computation of the optimal policy is not feasible. One way to formulate the problem is to define a policy space that includes all mixtures of the base policies. A policy in this class samples a base policy at each state and acts according to that, as opposed to sampling a base policy at the initial state and running it until the end.

A popular method to optimize a parameterized policy is policy gradient, which typically employs a variant of the gradient descent/ascent method [104, 92, 13, 75, 14]. Although in some applications the quality of the solution is high, the policy gradient methods often converge to some local minima as the problem is highly non-convex. Further, computing a gradient estimate can be an expensive operation. For example, the finite difference method requires running a number of policies in each iteration and estimating the value of a policy in a complicated system might require a long running time.

Here, we show a number of results on the problem of policy optimization in a restricted class of mixture policies. First, we show that solving the optimization problem is NP-hard. The hardness result is obtained by a reduction from the INDEPENDENT-SET problem for graphs and an application of the Motzkin-Straus theorem for optimizing quadratic forms over the simplex [67]. This result is somewhat surprising, since the same problem is known to be easy (in the complexity class P), if the space of base policies includes all MDP policies (an exponentially large space!) [73]. The critical difference is that in the unconstrained case an optimal MDP policy is known to be deterministic, in which case linear programming or policy iteration are known to run in polynomial time [105], whereas in the restricted case an optimal policy may need to randomize.

Although this hardness result is somewhat disappointing, we show that an approximately optimal solution can be found in a reasonable time when the occupancy measures of the base policies have large overlap. We obtain this result by formulating the problem in the *dual space*. More specifically, instead of searching in the space of mixture policies, we construct a new search space that consists of linear combinations of the occupancy measures of the base policies. Each such linear combination is not an occupancy measure itself, but it defines a policy through a standard normalization. Importantly, this new policy space also contains the base policies and so finding a near optimal policy in this class also provides a policy improvement w.r.t. the initial base policies. The objective function in the dual space is still highly non-convex, but we can exploit the convex relaxation proposed by [1] to have an efficient algorithm with performance guarantees. [1] study the linear programming approach to dynamic programming in the dual space (space of occupancy measures) and propose a penalty method that minimizes the sum of the linear objective and a number of constraint violations.

To demonstrate the idea, consider the problem of controlling the service rate of a queue where jobs arrive at a certain rate and the cost is the sum of the queue length and the chosen service rate. Consider two policies, one that selects low and one that selects high service rates. The space of the mixture of these two policies is rich and is likely to contain a policy with low total cost. We can generate a wide range of service rates as a convex combination of these two base policies. Now let us consider the dual space. The occupancy measures of the first and second policies are concentrated at large and small queue lengths, respectively. It can be shown that the linear combination of occupancy measures can generate a limited set of policies that are either similar to the first policy or the second one. In short, although the space of mixture policies is a rich space (and hence the optimization is NP-hard in that space), the space of dual policies can be more limited. More crucially, if the base policies have some similarities so that their occupancy measures overlap, then we can generate non-trivial policies in the dual space that can be competitive with the mixture policies in the primal space. We show that this is indeed the case in our experiments in Section 4.5.

Let us compare our algorithm with the traditional policy gradient in the space of

mixture of policies. Although the space of the mixture of the base policies is rich and is likely to contain a policy with lower total cost than any policy that is a mixture of the occupancy measures of the base policies, our approach has several advantages. First, policy gradient is more computationally demanding. Gradient descent needs to perform several rollouts in each round to estimate the gradient direction. In a complicated system, the mixing times can be large, and thus, we might need to run a policy for a very long time before we can reliably estimate its gradient. In contrast, and as we will show, apart from the initial rollouts to estimate the occupancy measures of the base policies, the proposed method does not need to interact with the environment when optimizing in the dual space. Second, our approach is *safe*; during the optimization phase, we only need to execute the base policies that are assumed to be safe. In contrast, policy gradient in the primal space evaluates many policies in the intermediate steps that some of them may not be safe to be executed. Note that by a safe policy here we mean a policy that performs better than a threshold, in terms of the returning value. Furthermore, our method enjoys stronger theoretical guarantees than the policy gradient method.

2.2.1 Notation

Let $M_{i,:}$ and $M_{:,j}$ denote ith row and jth column of matrix M, respectively. We denote by I an identity matrix, and by 1_n and 0_n, n-dimensional all one and all zero vectors, respectively. We use 0_{mm} to denote the all-zero $m \times m$ matrix and e_i to denote the unit m-vector (1 at position i and 0 elsewhere). We also use $\mathbf{1}\{.\}$ to denote the indicator function, and $\wedge$ and $\vee$ to denote the minimum and maximum. We define $[v]_+ = v \vee 0$ and $[v]_- = v \wedge 0$. For vectors v and w, $v \leq w$ means element-wise inequality, i.e., $v_i \leq w_i$, for all i. We use $\Delta_{\mathcal{S}}$ to denote the space of probability distributions defined on the set $\mathcal{S}$. For positive integer m, we use $[m]$ to denote the set $\{1, 2, \ldots, m\}$.

2.3 Preliminaries

In this chapter, we study reinforcement learning (RL) problems in which the interaction between the agent and environment has been modeled as a discrete[1] discounted MDP. A discrete MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, c, P, \alpha, \gamma \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the sets of S states and A actions, respectively; $c : \mathcal{S} \to [0, 1]$ is the cost function; $P : \mathcal{S} \times \mathcal{A} \to \Delta_{\mathcal{S}}$ is the transition probability distribution that maps each state-action pair to a distribution over states $\Delta_{\mathcal{S}}$; $\alpha \in \Delta_{\mathcal{S}}$ is the initial state distribution; and $\gamma \in (0, 1)$ is the discount factor. We are primarily interested in the case where the number of states is large. We assume that the cost does not depend on the action, although a number of our results can be extended to action-dependent costs. With an abuse of notation, we also use c to denote a (SA)-dimensional vector such that $c(s, a) = c(s)$ for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The restriction that costs are bounded in $[0, 1]$ interval is to simplify the notation and can be relaxed to other bounded ranges. Since we consider discrete MDPs, all the MDP-related quantities can be written in vector and matrix forms.

We also need to specify the rule according to which the agent selects actions at each state. We assume that this rule does not depend explicitly on time. A stationary policy $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ is a probability distribution over actions, conditioned on the current state. The MDP controlled by a policy π induces a Markov chain with the transition probability P_π and cost function $c_\pi = c$. We denote by J_π the value function of policy π, i.e., the expected sum of discounted costs of following policy π, and by $\nu_\pi \in \Delta_{\mathcal{S}}$ and $\mu_\pi \in \Delta_{\mathcal{S}\times\mathcal{A}}$ the state and state-action occupancy measures under policy π and w.r.t. the starting distribution α, i.e.,

$$\nu_\pi(s) = (1 - \gamma) \sum_{s' \in \mathcal{S}} \alpha(s') \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s | S_0 = s'),$$

and $\mu_\pi(s, a) = \nu_\pi(s)\, \pi(a|s)$. Note that when $s_0 \sim \alpha$, we may write $\mathbb{P}(S_t|S_0) = \alpha^\top P_\pi^t$, and thus, $\nu_\pi^\top = (1 - \gamma)\alpha^\top (I - \gamma P_\pi)^{-1}$, where I is the identity matrix. Given a policy class Π,

[1] In a remark at the end of Section 2.5, we will discuss extension to continuous-state problems.

the goal is to find a policy $\pi \in \Pi$ that minimizes $J(\pi) = \alpha^\top J_\pi$. It is easy to show that $J(\pi) = \nu_\pi^\top c_\pi = \mu_\pi^\top c$.

In this work, we are interested in the policy optimization problem where the policy class Π is defined as a mixture of m base policies $\{\pi_1, \ldots, \pi_m\}$, i.e., $\Pi = \left\{\pi_w : \pi_w = \sum_{i=1}^m w_i \pi_i,\ w \in \Delta_{[m]}\right\}$. We assume that the transition dynamics of the base policies are known. This assumption is only used for estimating the stationary distributions and average costs of the base policies. The extension to the unknown dynamics and costs (the RL problem), and the extension to the continuous state problems are discussed in a remark at the end of Section 2.5. We call the policy space Π the primal space. Executing a policy $\pi_w \in \Pi$ amounts to sampling one of the m base policies from the distribution $w \in \Delta_{[m]}$ at each time step, and then acting according to this policy. Finding the best policy in Π requires solving the following optimization problem

$$\min_{w \in \Delta_{[m]}} J(\pi_w) = \min_{w \in \Delta_{[m]}} \alpha^\top J_{\pi_w} \,. \tag{2.1}$$

We denote by w^* the solution to (2.1) and use $\pi_* = \pi_{w^*}$. For a positive constant R, we define the dual space of Π as the space of linear combinations of the state-action occupancies of the base policies, i.e., $\Xi = \left\{\xi_\theta : \xi_\theta = \sum_{i=1}^m \theta_i \mu_{\pi_i},\ \theta \in \Theta\right\}$, where $\Theta = \left\{\theta \in \mathbb{R}^m : \sum_{i=1}^m \theta_i = 1,\ \|\theta\|_2 \leq R\right\}$. Here, R is the radius of the parameter space and restricts the size of the policy class. Note that given the definition of Θ, each $\xi_\theta \in \Xi$ is not necessarily a state-action occupancy measure. However, each $\xi_\theta \in \Xi$ corresponds to a policy π_θ, defined as

$$\pi_\theta(a|s) = \frac{[\xi_\theta(s,a)]_+}{\sum_{a' \in \mathcal{A}} [\xi_\theta(s,a')]_+} \,. \tag{2.2}$$

If $\xi_\theta(s,a) \leq 0$ for all $a \in \mathcal{A}$, we let $\pi_\theta(.|x)$ be the uniform distribution. If ξ_θ is the state-action occupancy measure of a policy π, it can be shown that $\pi_\theta = \pi$. This implies that the base policies are in the dual space. We denote by μ_{π_θ} the state-action occupancy of policy π_θ. The policy optimization problem in the dual space is defined as

$$\min_{\theta \in \Theta} J(\pi_\theta) = \min_{\theta \in \Theta} \alpha^\top J_{\pi_\theta} = \min_{\theta \in \Theta} \mu_{\pi_\theta}^\top c \,, \tag{2.3}$$

where π_θ is computed from θ using Equation 2.2.

2.4 Hardness Result

In this section, we show that the policy optimization problem in the primal space (Eq. 2.1) is NP-hard. At a high level, the proof involves designing a special MDP and m base policies such that solving Eq. 2.1 in polynomial time would imply P=NP.

Theorem 2.4.1. *Given a discounted MDP, a set of policies, and a target cost r, it is NP-hard to decide if there exists a mixture of these policies that has expected cost at most r.*

Proof. We reduce from the INDEPENDENT-SET problem. This problem asks, for a given (undirected and with no self-loops) graph $\mathcal{G}$ with vertex set V, and a positive integer $j \leq |V|$, whether $\mathcal{G}$ contains an independent set $V' \subseteq V$ having $|V'| \geq j$. This problem is NP-complete [30].

Let G be the $m \times m$ (symmetric, 0-1) adjacency matrix of an input graph $\mathcal{G}$, and let $A = I + G$, where I is the identity matrix. The reduction constructs a deterministic MDP with $m + 3$ states and $m + 3$ actions, and m deterministic policies π_i that induce corresponding chains P_{π_i}, for $i = 1, \ldots, m$, where each $(m+3) \times (m+3)$ matrix P_{π_i} reads

$$P_{\pi_i} = \begin{bmatrix} 0 & e_i^\top & 0 & 0 \\ 0_m & 0_{mm} & A_{:,i} & 1_m - A_{:,i} \\ 0 & 0_m^\top & 0 & 1 \\ 0 & 0_m^\top & 0 & 1 \end{bmatrix}. \tag{2.4}$$

A mixture π_w of the base policies, with weights w, induces the chain

$$P_{\pi_w} = \sum_{i=1}^{m} w_i P_{\pi_i} = \begin{bmatrix} 0 & w^\top & 0 & 0 \\ 0_m & 0_{mm} & Aw & 1_m - Aw \\ 0 & 0_m^\top & 0 & 1 \\ 0 & 0_m^\top & 0 & 1 \end{bmatrix}. \tag{2.5}$$

It is easy to see that, for each $k \geq 3$, the kth power of P_{π_w} is equal to $P_{\pi_w}^k = [\, 0_{(m+3)(m+2)},\, 1_{m+3}\,]$ (all zeros except for the last column that is all ones), while its square $P_{\pi_w}^2$ reveals the quadratic form $w^\top A w$ in position $(1, m+2)$. Hence, for initial state distribution $\alpha = [1, 0, \ldots, 0]^\top$ (all mass on the first state), state-only-dependent cost vector $c = [0, \ldots, 0, 1, 0]^\top$ (all zeros except for 1 in the $(m+2)$'th state), and discount factor $\gamma < 1$, the expected discounted cost of π_w is

$$\begin{aligned} J(\pi_w) &= (1-\gamma)\alpha^\top (I - \gamma P_{\pi_w})^{-1} c \\ &= (1-\gamma)\alpha^\top (I + \gamma P_{\pi_w} + \gamma^2 P_{\pi_w}^2 + \ldots) c \\ &= (1-\gamma)\gamma^2\, w^\top A w\ . \end{aligned} \tag{2.6}$$

For any graph $\mathcal{G}$ with m vertices and adjacency matrix G, the following holds [67]:

$$\frac{1}{\omega(\mathcal{G})} = \min_{y \in \Delta_{[m]}} y^\top (I + G) y\ , \tag{2.7}$$

where $\omega(\mathcal{G})$ is the size of the maximum independent set of $\mathcal{G}$. Let the target cost be $r = \frac{(1-\gamma)\gamma^2}{j}$, where j is the target integer of the INDEPENDENT-SET instance. Then the decision question $J(\pi_w) \leq r$ is equivalent to $w^\top (G + I) w \leq \frac{1}{j}$, where we used (2.6). Hence, it follows from (2.7) that the existence of a vector w that satisfies $J(\pi_w) \leq r$ would imply $\omega(\mathcal{G}) \geq j$, and thus, $|V'| \geq j$ for some independent set $V' \subseteq V$. This establishes that, deciding the MDP policy optimization problem in polynomial time would also decide the INDEPENDENT-SET problem in polynomial time, implying P=NP. □

Remark 1: The same technique can be used to show NP-hardness of the problem under an average cost criterion. This only requires changing the last two rows of the matrices P_{π_i}, by having the 1's in the first column instead of the last column. In that case, if $\nu_{\pi_w} = [\, x,\, v^\top,\, y,\, z\,]^\top$ is the stationary distribution of P_{π_w}, where v is an m-vector and x, y, z scalars, we can algebraically solve the eigensystem $\nu_{\pi_w}^\top = \nu_{\pi_w}^\top P_{\pi_w}$ (by elementary manipulations), to get $v = w$ and $y = w^\top A w$. Hence, for a cost vector $c = [0, \ldots, 0, 1, 0]^\top$, the average cost of the MDP under w is $w^\top A w$, and by choosing target cost $r = \frac{1}{j}$ the

Input: base policies $\{\pi_i\}_{i=1}^m$,; dual parameter space Θ; number of rounds T, learning rates $\{\beta_t\}_{t=1}^T$, penalty parameter H
Compute occupancy measures of the base policies, i.e., $\{\mu_{\pi_i}\}_{i=1}^m$
Initialize $\theta_1 = 0$
for $t := 1, 2, \ldots, T$ **do**
  Sample $i \in [m]$ and sample $(x_t, a_t) \sim \mu_{\pi_i}$
  Compute subgradient estimate $g_t(\theta_t)$ *(Eq. 2.9)*
  Update $\theta_{t+1} = \Pi_\Theta(\theta_t - \beta_t g_t)$ *(Π_Θ is the Euclidean projection onto Θ)*
end for
Compute $\widehat{\theta} = \frac{1}{T}\sum_{t=1}^T \theta_t$
Return policy $\pi_{\widehat{\theta}}$ *(Eq. 2.2)*

Figure 2.1: Stochastic Subgradient Method for MDPs.

Motzkin-Straus argument applies as above.

Remark 2: Optimizing over a restricted policy class essentially converts the MDP to a POMDP, for which related complexity results are known [73, 68, 99, 52].

2.5 Reduction to Convex Optimization

In this section, we first propose an algorithm to solve the policy optimization problem in the dual space (Eq. 2.3) and then prove a bound on the performance of the policy returned by this algorithm compared to the solution of the policy optimization problem in the primal space (Eq. 2.1).

Figure 2.1 contains the pseudocode of our proposed algorithm. The algorithm takes

the m base policies $\{\pi_i\}_{i=1}^m$ and the dual parameter space Θ as input. It first computes the state-action occupancy measures of the base policies, i.e., $\{\mu_{\pi_i}\}_{i=1}^m$. This is done using the recent results by [23] that show it is possible to compute the stationary distribution (occupancy measure) of a Markov chain in time *linear* to the size of the state space.[2] This guarantees that we can compute the state-action occupancy measures of the base policies efficiently. Alternatively, when the size of the state space is very large, these occupancy measures can be estimated by roll-outs.

Motivated by the approach of [1], we propose minimizing a convex surrogate function

$$\mathcal{L}(\theta) = c^\top \xi_\theta + H \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} |[\xi_\theta(x,a)]_-| \,, \tag{2.8}$$

where ξ_θ is a linear combination of the occupancy measures and H is a positive parameter that penalizes negative values in ξ_θ. At each time step t, our algorithm first computes an estimate of the sub-gradient $\nabla\mathcal{L}(\theta_t)$ and then feeds it to the projected sub-gradient method to update the policy parameter θ_t. In order to compute an estimate of the sub-gradient $\nabla\mathcal{L}(\theta_t)$, we first sample a state-action pair (x_t, a_t) from $(1/m)\sum_{i=1}^m \mu_{\pi_i}$, and then compute the function

$$g_t(\theta_t) = c^\top M - H\frac{M(x_t,a_t)\,\mathbf{1}\{\xi_{\theta_t}(x_t,a_t) < 0\}}{(1/m)\sum_{i=1}^m \mu_{\pi_i}(x_t,a_t)}, \tag{2.9}$$

where M is a $SA \times m$ matrix, whose j'th column is μ_{π_j}, and $M(x,a)$ is the row of M corresponding to the state-action pair (x,a). Notice that elements of $c^\top M$ are simply average costs of the base policies. To sample (x_t, a_t) from $(1/m)\sum_{i=1}^m \mu_{\pi_i}$, we first select a number $i \in [m]$ uniformly at random and then sample a state-action pair (x_t, a_t) from the state-action occupancy measure of the i'th base policy, i.e., $(x_t, a_t) \sim \mu_{\pi_i}$. If μ_{π_i} is approximated by the historical data generated under policy π_i, then this last sampling amounts to sampling one datapoint uniformly at random from historical observations under policy π_i.

[2] These results can also be applied to the discounted case.

We now show that $g_t(\theta)$ in (2.9) is an unbiased estimate of $\nabla \mathcal{L}(\theta)$:

$$\begin{aligned}
&\mathbb{E}\left[\frac{M(x_t,a_t)\mathbf{1}\left\{\xi_\theta(x_t,a_t)<0\right\}}{(1/m)\sum_{i=1}^m \mu_{\pi_i}(x_t,a_t)}\right] \\
&= \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} \frac{M(x,a)\mathbb{P}\left(x_t=x,a_t=a\right)}{(1/m)\sum_{i=1}^m \mu_{\pi_i}(x,a)}\mathbf{1}\left\{\xi_\theta(x,a)<0\right\} \\
&= \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} M(x,a)\mathbf{1}\left\{\xi_\theta(x,a)<0\right\}.
\end{aligned}$$

After T rounds, the algorithm returns the average of the computed policy parameters $\{\theta_t\}_{t=1}^T$, i.e., $\widehat{\theta} = (1/T)\sum_{t=1}^T \theta_t$. The parameter $\widehat{\theta}$ defines an element $\xi_{\widehat{\theta}}$ of the dual space Ξ, which in turn defines a policy $\pi_{\widehat{\theta}}$ using Eq. 2.2. We denote by $\mu_{\pi_{\widehat{\theta}}}$ the state-action occupancy measure of this policy.

The approach of [1] involves minimizing an objective function with an additional term that measures the distance of ξ_θ from the space of occupancy measures. The transition matrix appears in this extra term. In our case, ξ_θ is a linear combination of occupancy measures and hence this extra term is zero, and the transition dynamics does not appear in the objective (2.8). This results in a simpler optimization problem and improved performance bounds. More importantly, we can use the resulting algorithm in the reinforcement learning setting as no *backward simulator* is needed.

Our main result shows that the policy returned by our algorithm, $\pi_{\widehat{\theta}}$, is near-optimal as long as the occupancy measures of the base policies have large overlap.

Theorem 2.5.1. *Let $\delta > 0$ be the probability of error, $H = 1/\eta$ be the constraints multiplier in the sub-gradient estimate (2.9), and $T = O\big(\frac{R^2}{\eta^2}\log(1/\delta)\big)$ be the number of rounds. Let $\widehat{\theta}$ be the output of the stochastic sub-gradient method after T rounds, the learning rate be $\beta_t = S/(G'\sqrt{T})$, where $G' = \sqrt{m}+Hm$, and $U(\theta) = \sum_{(x,a)\in\mathcal{X}\times\mathcal{A}} |[\xi_\theta(x,a)]_-|$. Then with probability at least $1-\delta$, we have the following bound on the performance of the policy $\pi_{\widehat{\theta}}$ returned by the*

algorithm in Figure 2.1:

$$\overbrace{\alpha^\top J_{\pi_{\widehat{\theta}}}}^{J(\pi_{\widehat{\theta}})} \le \min_{w \in \Delta_{[m]}} \overbrace{\alpha^\top J_{\pi_w}}^{J(\pi_w)} + O(\eta) \\ + \min_{\theta \in \Theta} \left(\sum_i \theta_i (\nu_{\pi_i} - \nu_{\pi_*})^\top c + \left(\frac{1}{\eta} + \frac{9}{1-\gamma} \right) U(\theta) \right) .$$

Moreover, the computational complexity of the algorithm is $O\big(poly(m)A\big)$ and the constants hidden in $O(\eta)$ are polynomials in R, m, and $1/(1-\gamma)$.

Before proving Theorem 2.5.1, we show the improvement using a simple example.

Example Let λ be such that for any $i, j \in [m]$ and any $(x, a) \in \mathcal{X} \times \mathcal{A}$,

$$\frac{\lambda}{1+\lambda} \mu_{\pi_j}(x,a) \le \mu_{\pi_i}(x,a) \le \frac{1+\lambda}{\lambda} \mu_{\pi_j}(x,a) . \tag{2.10}$$

A large value of λ indicates large overlap of occupancy measures. Let $\varepsilon = \|\mu_{\pi_1} - \mu_{\pi_*}\|_1$. Parameters λ and ε provide ℓ^∞ and ℓ^1 error bounds. We can obtain an easy but non-trivial bound under the condition (2.10) as follows. Let π_1 and π_m be the policies with the smallest and largest values. Choose $\theta_1 = 1 + \lambda$, $\theta_m = -\lambda$, and all other elements are zero. Then

$$\begin{aligned} U(\theta) &= \sum_{(x,a)} |[\xi_\theta(x,a)]_-| \\ &= \sum_{(x,a)} |[(1+\lambda)\mu_{\pi_1}(x,a) - \lambda \mu_{\pi_m} .(x,a)]_-| \ . \end{aligned}$$

For each term to be negative, we must have $\mu_{\pi_1}(x,a) \le \frac{\lambda}{1+\lambda} \mu_{\pi_m}(x,a)$, which contradicts our assumption, thus $U(\theta) = 0$. Let $\mathcal{E} = \sum_i \theta_i (\nu_{\pi_i} - \nu_{\pi_*})^\top c$. We get that

$$\begin{aligned} \mathcal{E} &= (1+\lambda)(\mu_{\pi_1} - \mu_{\pi_*})^\top c - \lambda (\mu_{\pi_m} - \mu_{\pi_*})^\top c \\ &= (\mu_{\pi_1} - \mu_{\pi_*})^\top c - \lambda (\alpha^\top J_{\pi_m} - \alpha^\top J_{\pi_1}) . \end{aligned}$$

Thus,

$$\alpha^\top J_{\pi_{\widehat{\theta}}} \leq \alpha^\top J_{\pi_*} + O(\eta) + \varepsilon - \lambda\alpha^\top (J_{\pi_m} - J_{\pi_1}) \,. \tag{2.11}$$

Let's compare the above result to the simple bound

$$\alpha^\top J_{\pi_1} \leq \alpha^\top J_{\pi_*} + \varepsilon \,. \tag{2.12}$$

The term $O(\eta)$ is due to the error of gradient descent in a convex problem and can be made arbitrarily small by increasing the number of iterations. Thus, the term $\lambda\alpha^\top (J_{\pi_m} - J_{\pi_1})$ in (2.11) shows the amount of improvement compared to the bound in (2.12). This term is positive since π_m has a larger value than π_1.

Next, we state a number of results that will be used to prove Theorem 2.5.1. Theorem. 1 of [1] adapted to the dual space Ξ implies that $\pi_{\widehat{\theta}}$ is near-optimal in the dual space Ξ.

Theorem 2.5.2 ([1]). *Let δ, T, H, and $\widehat{\theta}$ be defined as in Theorem 2.5.1. Then with probability at least $1-\delta$, we have*

$$J(\widehat{\theta}) = \alpha^\top J_{\pi_{\widehat{\theta}}} \leq \min_{\theta\in\Theta} \Bigg(\alpha^\top J_{\pi_\theta} + \frac{6\sqrt{m}CR\eta}{1-\gamma} + O(\eta) + \left(\frac{1}{\eta} + \frac{6}{1-\gamma}\right) \sum_{(x,a)} |[\xi_\theta(x,a)]_-| \Bigg) \,,$$

where the constants hidden in $O(\eta)$ are polynomials in R, m, and C.

Our main technical tool is the following lemma.

Lemma 2.5.3. *Let $\varepsilon = \max_{i,j\in[m]} \left\|\nu_{\pi_i} - \nu_{\pi_j}\right\|_1$. Then, for any $i \in [m]$ and any policy π_w in the primal space Π, we have $\left\|\nu_{\pi_i} - \nu_{\pi_w}\right\|_1 \leq \frac{\varepsilon(1+\gamma)}{1-\gamma}$.*

Proof. For any $i, j \in [m]$, there exists a vector $v_{i,j}$ with $\left\|v_{i,j}\right\|_1 \leq \varepsilon$, such that $\nu_{\pi_i} - \nu_{\pi_j} = v_{i,j}$. We may rewrite this equation as $\alpha^\top (I - \gamma P_{\pi_i})^{-1} = \alpha^\top (I - \gamma P_{\pi_j})^{-1} + v_{i,j}^\top$, and further as

$$\alpha^\top (I - \gamma P_{\pi_i})^{-1} (I - \gamma P_{\pi_j}) = \alpha^\top + v_{i,j}^\top (I - \gamma P_{\pi_j}) \,. \tag{2.13}$$

Now for a policy $\pi_w \in \Pi$ corresponding to the weight vector $w \in \Delta_{[m]}$, we may write

$$\begin{aligned}
&\alpha^\top (I - \gamma P_{\pi_i})^{-1} \Big(I - \gamma \sum_{k=1}^{m} w_k P_{\pi_k} \Big) \\
&\qquad = \sum_{k=1}^{m} w_k \alpha^\top (I - \gamma P_{\pi_i})^{-1} (I - \gamma P_{\pi_k}) \\
&\qquad \overset{\text{(a)}}{=} \alpha^\top \sum_{k=1}^{m} w_k + \sum_{k=1}^{m} w_k v_{i,k}^\top (I - \gamma P_{\pi_k}) \\
&\qquad \overset{\text{(b)}}{=} \alpha^\top + \sum_{k=1}^{m} w_k v_{i,k}^\top (I - \gamma P_{\pi_k}) \,, \qquad (2.14)
\end{aligned}$$

where **(a)** is from Eq. 2.13 and **(b)** is because $w \in \Delta_{[m]}$, and thus, $\sum_{k=1}^{m} w_k = 1$. From Eq. 2.14, we have

$$\begin{aligned}
\alpha^\top (I - \gamma P_{\pi_i})^{-1} &= \alpha^\top \Big(I - \gamma \sum_{k=1}^{m} w_k P_{\pi_k} \Big)^{-1} \\
&+ \sum_{k=1}^{m} w_k v_{i,k}^\top (I - \gamma P_{\pi_k}) \Big(I - \gamma \sum_{l=1}^{m} w_l P_{\pi_l} \Big)^{-1} . \qquad (2.15)
\end{aligned}$$

Since $P_{\pi_w} = \sum_{k=1}^{m} w_k P_{\pi_k}$, we may rewrite Eq. 2.15 as

$$\begin{aligned}
\nu_{\pi_i} - \nu_{\pi_w} &= \alpha^\top (I - \gamma P_{\pi_i})^{-1} - \alpha^\top (I - \gamma P_{\pi_w})^{-1} \\
&= \sum_{k=1}^{m} w_k v_{i,k}^\top (I - \gamma P_{\pi_k}) \Big(I - \gamma \sum_{l=1}^{m} w_l P_{\pi_l} \Big)^{-1} .
\end{aligned}$$

Let

$$\varepsilon' = \max_{i \in [m]} \left\| \sum_{k=1}^{m} w_k v_{i,k}^\top (I - \gamma P_{\pi_k}) \Big(I - \gamma \sum_{l=1}^{m} w_l P_{\pi_l} \Big)^{-1} \right\|_1 ,$$

$z_k = (I - \gamma P_{\pi_k})^\top v_{i,k}$, $Q = \sum_{l=1}^m w_l P_{\pi_l}$, and $M^{-1} = I - \gamma Q$. We may now write

$$\begin{aligned}
\|\nu_{\pi_i} - \nu_{\pi_w}\|_1 \leq \varepsilon' &\leq \left\| M^\top \sum_{k=1}^m w_k z_k \right\|_1 \\
&\leq \left\| M^\top \right\|_1 \left\| \sum_{k=1}^m w_k z_k \right\|_1 \\
&\leq \underbrace{\left\| I + \gamma Q^\top + \gamma^2 Q^{2\top} + \dots \right\|_1}_{\leq 1/(1-\gamma)} \\
&\qquad \times \sum_{k=1}^m w_k \overbrace{\left\| (I - \gamma P_{\pi_k})^\top \right\|_1}^{\leq (1+\gamma)} \overbrace{\|v_{i,k}\|_1}^{\leq \varepsilon} \\
&\leq \frac{\varepsilon(1+\gamma)}{1-\gamma} \,.
\end{aligned}$$

This concludes the proof. □

Theorem 2.5.1 follows immediately from the above two results.

Proof of Theorem 2.5.1. From Lemma 2 of [1], for any $\theta \in \Theta$, we have $\|\xi_\theta - \mu_{\pi_\theta}\|_1 \leq \frac{3U(\theta)}{1-\gamma}$. From Lemma 2.5.3, we have $\|\nu_{\pi_i} - \nu_{\pi_*}\|_1 = O(\varepsilon)$, for any $i \in [m]$ and $\pi^* = \pi_{w^*}$. Thus, for any $\theta \in \Theta$, we may write

$$\begin{aligned}
\alpha^\top J_{\pi_\theta} &= \alpha^\top J_{\pi_*} + \mu_{\pi_\theta}^\top c - \mu_{\pi_*}^\top c \\
&= \alpha^\top J_{\pi_*} + \xi_\theta^\top c - \mu_{\pi_*}^\top c + \mu_{\pi_\theta}^\top c - \xi_\theta^\top c \\
&\leq \alpha^\top J_{\pi_*} + (\xi_\theta - \mu_{\pi_*})^\top c + \frac{3U(\theta)}{1-\gamma} \\
&= \alpha^\top J_{\pi_*} + \sum_i \theta_i (\mu_{\pi_i} - \mu_{\pi_*})^\top c + \frac{3U(\theta)}{1-\gamma} \\
&= \alpha^\top J_{\pi_*} + \sum_i \theta_i (\nu_{\pi_i} - \nu_{\pi_*})^\top c + \frac{3U(\theta)}{1-\gamma} \,.
\end{aligned}$$

We used the fact that $c(x, a) = c(x)$ in the 5th step. This last inequality together with Theorem 2.5.2 gives us the desired result. □

The main theoretical contribution of this section is Lemma 2.5.3, which allows us to relate the primal and dual spaces. Although Theorem 2.5.2 is taken from [1], this particular presentation using stationary distributions as features is new and provides an interesting application of the previous result.

Remark: When the reward function and transition probabilities are unknown (the RL problem), we can run the base policies to estimate their occupancy measures and average costs. These rollouts introduce an estimation error that will also appear in our performance bounds. Nevertheless, as we will show in our experiments, the rollouts provide an effective way to approximate the occupancy measures. Alternatively, we might have access to historical data from base policies that can be used for this estimation.

To simplify the presentation, we assumed that the state and action spaces are finite. However, the proposed algorithm can be easily extended to continuous problems; the occupancy measures can still be approximated using rollouts along with state aggregation or other function approximation techniques.

2.6 Experiments

We compare the results of combining policies in the policy space (w) and occupancy measure space (θ), where the base policies have overlapping occupancy measures. We consider three different queuing problems: *1-Queue*, *4-Queue*, and *8-Queue systems*. These problems have been studied before by [25] with slightly different parameters. The results for *1-Queue*, *4-Queue systems*, and *8-Queue systems* are shown next.

In all the experiments, the proposed policy gradient in the dual space produces policies whose average costs are comparable with solutions of the policy gradient in the primal space. The notable difference is that the proposed policy gradient is significantly

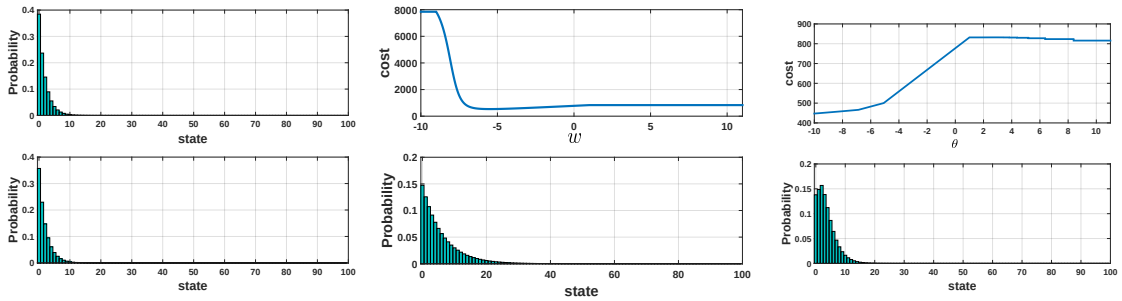
more resource efficient. In both the 4-queue and 8-queue problems, the time complexity of the proposed method is less than 0.001 of the time complexity of policy gradient in the primal space. In a RL setting, the proposed policy gradient would be similarly more sample efficient.

2.6.1 Queuing Problem: 1-Queue

Consider a queue of length L. We denote the state of the system at time t by x_t that shows the number of jobs in the queue. Jobs arrive at the queue with rate p. Action at each time, a_t, is chosen from the finite set $\{0.1625, 0.325, 0.4875, 0.65\}$ that represents the service rate or departure probability. The transition function of the system is then defined as: $x_{t+1} = x_t - 1$ w.p. a_t; $x_{t+1} = x_t + 1$ w.p. p; and $x_{t+1} = x_t$, otherwise. The system goes from state 0 to 1 w.p. p and stays in 0 w.p. $1 - p$. Also, transition from state L to $L-1$ has probability $a(L)$ and the system stays in L w.p. $1-a(L)$. The cost incurred by being in state x and taking action a is given by $c(x, a) = x^2 + 2500a^2$.

Consider a queue with $L = 99$ and two base policies that are independent of the states and have the following distributions over the set of actions: $\pi_1 = [0, 0, 0.50, 0.50]$ and $\pi_2 = [0, 0.1, 0.45, 0.45]$. Consider two scenarios: **1)** Mixing the original policies in the space of w, and **2)** Building a policy based on combining the corresponding occupancy measures (optimization in the space of θ).

Figure 2.2 shows the results of optimization in the two spaces. The costs associated with π_1 and π_2 are $J(\pi_1) = 831.91$ and $J(\pi_2) = 777.36$. Suppose $\pi_w = w\pi_1 + (1 - w)\pi_2$ is our mixture policy. Then for $0 \leq w \leq 1$, $J(\pi_w)$ is a monotonically increasing function of w. In other words, there is no gain in mixing the policies when w is between 0 and 1. For this experiment, we let w to take values outside this interval to examine the lowest possible cost. The best mixing weight is $w = -5.49$ and the associated cost is $J(\pi_w) = 533.60$. However, in the space of occupancy measures, we can build a better policy. Suppose $\xi_\theta = \theta\mu_1 + (1 - \theta)\mu_2$. Then the cost associated with the policy that is built based on ξ_θ decreases monotonically as we decrease θ and saturates at almost



(a) (b) (c)

Figure 2.2: (a) Stationary distribution of the two initial policies. (b) Top: Cost of the mixture policy versus w. Bottom: Stationary distribution of the best mixture policy found in the space of w. (c) Top: Cost of the mixture policy versus θ. Bottom: Stationary distribution of the best mixture policy found in the space of θ. Note that by letting θ to take negative values we enlarge the set of possible mixture policies that our algorithm returns. However, the resulting policy is not necessarily in the convex hull of the base policies.

$J(\pi_\theta) = 447$, which is much lower than the cost of the optimal policy mixing in the space of w.

2.6.2 Queuing Problem: 4-Queues

Consider a system with four queues, each with capacity $L = 9$, shown in Figure 2.3. This problem has been studied in several papers (e.g., [22, 53, 25]). There are two servers in the system: Server 1 serves queue 1 and 4 with rates r_1 and r_4, and Server 2 serves queue 2 and 3 with rates r_2 and r_3. Each server serves only one of its associated queues at each time. Jobs arrive at queue 1 and 3 with rate λ. A job leaves the systems after being served at either queue 1 and 2 or queue 3 and 4. The state of the system is denoted by a 4-dimensional vector $[x_1, x_2, x_3, x_4]$, where x_i represents the number of jobs in queue i at each time. A controller can choose a 4-dimensional action from $\{0, 1\}^4$ such that $a_1 + a_4 \leq 1$ and $a_2 + a_3 \leq 1$. The cost at each time is equal to the total number of jobs in the system: $c(x) = \sum_{i=1}^{4} x_i$.

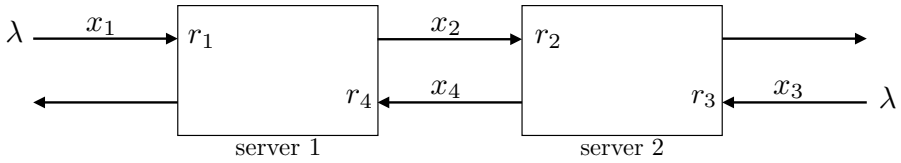


Figure 2.3: A system with four queues and two servers

index	p_1	p_2	cost
1	0.9	0.9	16.2950
2	0.9	0.7	17.3926
3	0.8	0.8	15.1535
4	0.7	0.9	13.6525
5	0.7	0.7	14.3266

Table 2.1: Base Policies for the 4-Queue problem.

Assume $r_1 = r_2 = 0.12$, $r_3 = r_4 = 0.28$, and $\lambda = 0.08$. We choose our base policies from a family of policies for which server i serves its longer queue w.p. p_i and its shorter queue w.p. $1 - p_i,\ i = \{1, 2\}$. Five base policies and their associated costs are shown in Table 2.1.

Solving this problem in the space of policies using policy gradient will result in the optimal weight vector $w^* = [0, 0, 0, 1, 0]$, i.e., giving all the weight to the policy with the lowest cost. Therefore, the cost of the mixture policy will be $J(\pi^*) = 13.6525$. Interestingly, if we solve the problem in the dual space (space of stationary distributions) using policy gradient and Eq. 2.2, after 200 iterations, the optimal resulting policy will have cost $J(\pi_\theta) = 12.84$ with $\theta = [-0.32, -0.68, -0.4, 2.16, 0.24]$ (note that based on the definition of Θ, this vector can have negative values). In either of the spaces, in order to approximate the gradient surface for the policy gradient method, we need to compute the cost of each mixed policy a couple of times (depending on the number of base policies) in each iteration. Computing the cost involves an eigen-decomposition of the transition matrix, which makes the whole process computationally costly. Using our stochastic sub-gradient method, we can approximate the cost very fast and efficiently.

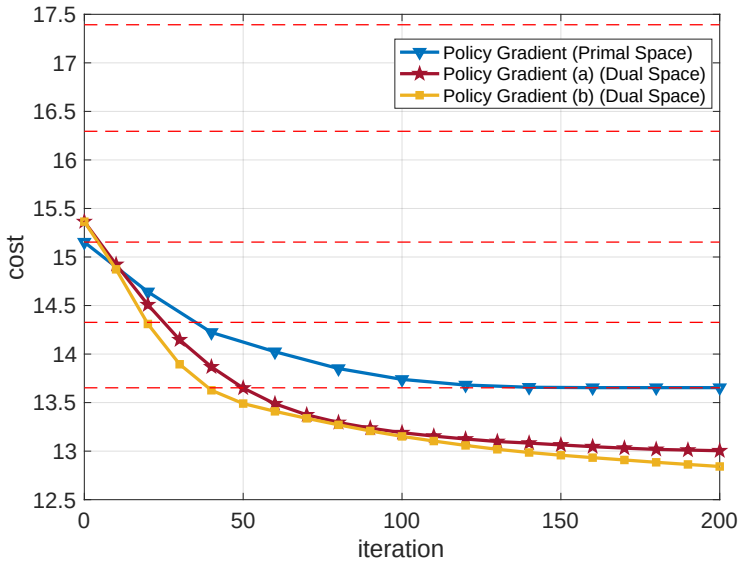


Figure 2.4: Cost per iteration for the primal and dual spaces. The policy gradient (b) for the dual space is the method described in Algorithm 2.1. Horizontal dashed lines are the costs of the base policies.

Our method needs only one eigen-decomposition for the final mixing weights. Therefore, the computational cost is much lower (in this particular example $1/(8 \times 200)$ of the policy gradient method, where 8 is the number of points we use for approximating the gradient surface and 200 is the number of iterations). The policy resulted from our method will have cost $J(\pi_\theta) = 13.00$ with $\theta = [-0.23, -1.28, 0.09, 1.54, 0.88]$. Figure 2.4 shows the difference between the costs of policy gradient and the stochastic sub-gradient method at each iteration.

2.6.3 Queuing Problem: 8-Queues

Consider a system with three servers and eight queues. Similar to 4-queue problem, each server is responsible of serving multiple queues. There are two pipelines. Jobs in the first pipeline enter the system from the first queue by rate λ_1 and exit the system from the third queue. Jobs in the second pipeline enter the system from the forth queue

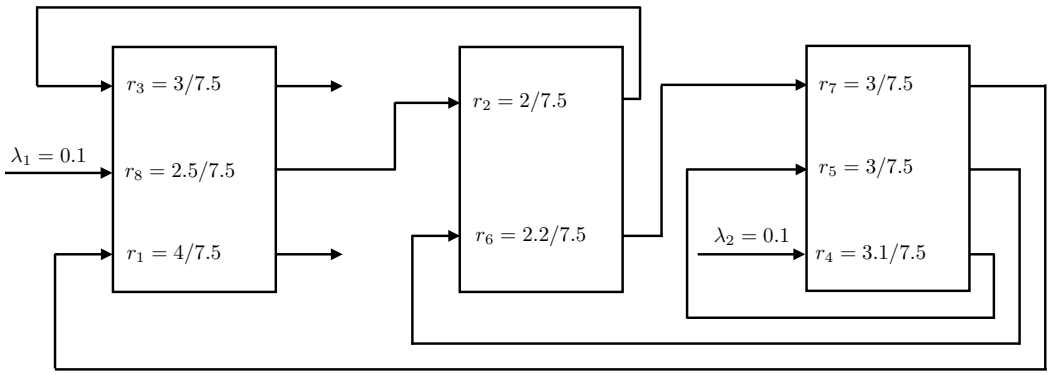


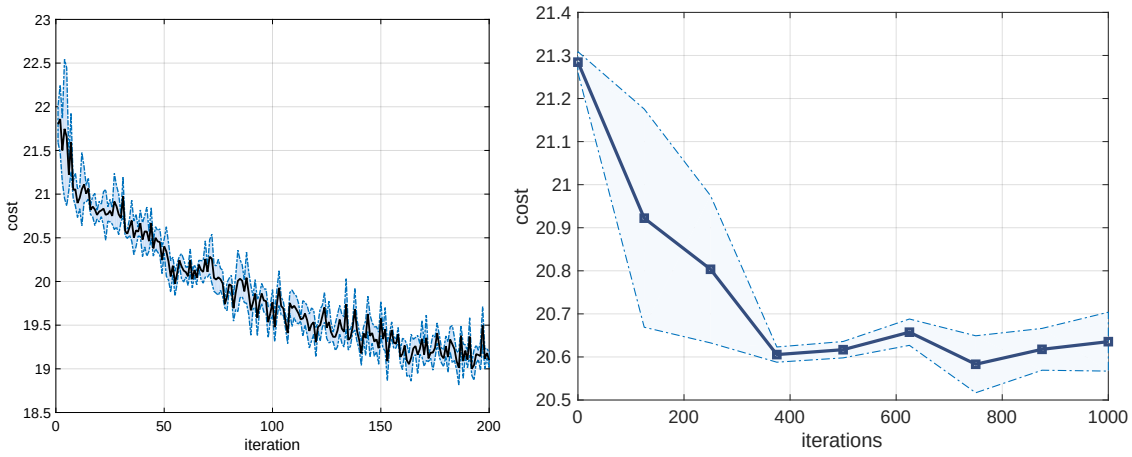
Figure 2.5: A system with 8 queues and 3 servers.

by rate λ_2 and exit the system from the eighth queue. Fig. 2.5 demonstrates the system. The service rate is denoted by r_i, $i \in \{1, 2, ..., 8\}$. There is no limit on the length of queues. State of the system is determined by an 8-dimensional vector that shows the number of jobs in each queue. Actions are also 8-dimensional binary vectors, where each dimension shows if the associated queue is served or not. The cost at each time step is equal to the total number of jobs in the system. Each server can serve only one queue at each time. We choose three base policies from a family of policies that is described below.

Family of base policies: The base policies are parameterized by three components: p_1, p_2, and p_3, which are associated to each server and satisfy the these properties: **1)** If all of the queues associated to server i are empty the server is idle. **2)** If only one of the queues for server i is non-empty, that queue is served with probability p_i. **3)** If server i has more than one non-empty queue, it serves the longest queue with probability p_i and the rest of the non-empty queues with probability $(1 - p_i)/(n_q - 1)$, where n_q is the number of non-empty queues.

The three base policies and their costs are shown in Table 2.2. The cost is computed by running each policy for 10,000,000 iterations, starting from empty system, and averaging over number of jobs at each iteration.

Solving the problem in the policy space: As opposed to the previous problems,



(a) (b)

Figure 2.6: Mean and standard deviation of cost per iteration in the eight queue problem. (a) Primal space. (b) Dual Space.

index	p_1	p_2	p_3	cost
1	0.8	0.5	0.5	21.78$\pm$ 0.16
2	0.5	0.8	0.5	22.47$\pm$0.08
3	0.5	0.5	0.8	22.98 $\pm$ 0.12

Table 2.2: Base Policies for eight queue problem

the number of states in this domain is not limited. We solve the problem in the primal space using policy gradient, and specifically a finite differences method. We start from a random initial weight. At each iteration, we approximate the gradient surface using four points around the current point and the weights are updated accordingly. For each point the cost of the mixed policy should be computed by running the policy for a long time, e.g. 10,0000,000 iterations. Therefore updating the weights in each iteration is computationally expensive and finding the optimum mixing weight in this space is very tedious. After 200 iterations the weight converges to $w^* = [0.35, 0.33, 0.32]$, which corresponds to $J(\pi^*) = 19.14$ (on average).

Solving the problem in the space of occupancy measures: To solve the problem

in this space we need to run the base policies for a long time (around 10,0000,000 iterations) only once and obtain the stationary distributions. From there, running the induced mixed policies to estimate their values are not needed and the optimal θ is obtained using the described algorithm. The optimal value of θ for this problem is: $\theta = [1.1246, -0.1143, -0.0102]$ and the cost of the induced policy is $J(\pi_\theta) = 20.59$. In fact, by mixing the policies in this space we obtain a lower cost compared to the base policies. Although, this cost is slightly worse than the cost of the policy obtained in the primal space, the gain in computation is extremely significant. In this specific example the optimization in the dual is almost 300 times faster than the optimization in the primal space. Figure 2.6 shows the cost per iteration in the two spaces.

2.7 Summary

In this chapter we showed that the problem of finding an optimal policy in convex hull space of some base policies in an MDP is an NP-hard problem to be solved exactly or approximated to arbitrary accuracy. We proposed, however, an efficient algorithm that finds an policy almost as good as optimal policy under the assumption that the base policies have large overlaps in their occupancy measure space. The proposed algorithm does not require interaction with the environment during its optimization, and therefore unsafe policies will not cause potentially catastrophic outcomes. We showed through our experiments for different queue problems that the proposed algorithm can efficiently output policies that have smaller costs compared to the base policies.

Chapter 3

Representation Learning using VAEs

In this chapter we first provide a short overview of variational autoencoder (VAE) as a deep model that is used for implementing variational inference. Then we show the effectiveness of using VAEs for two different applications: 1) Learning disentangled representations. 2) Sufficient dimensionality reduction. VAE is the main tool that we use for representation learning in the next two chapters.

3.1 VAE: An introduction

In machine learning, probabilistic models are used to model natural and artificial phenomena based on the data. The probabilistic models explain such phenomena in a mathematical way that can be useful for understanding the phenomena as well as predicting unknown future or automated decision making.

Therefore we can formalize the concepts of knowledge and skill using probabilistic models. Probabilistic models in their complete form present all sorts of correlations and dependencies between variables in the model. Let's denote the observed variables by $\mathbf{x}$. One of the most important applications of probabilistic models is to model the unknown distribution of $\mathbf{x}$ denoted by $p(\mathbf{x})$. To do so a common approach is to discover the un-

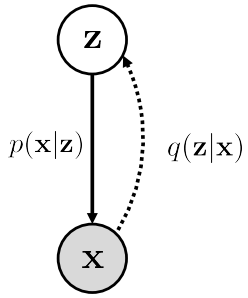


Figure 3.1: Graphical model

derlying process that generates the observations. It is usually assumed that the observations can be described by some features in a more simple (lower-dimensional) space. For example, a set of portraits can be represented in a very high-dimensional pixel space. However, each portrait can also be described using a much lower-dimensional variable with only few features, e.g. using few face landmarks, and we can change the portrait only by tweaking these few features. Since these features are not directly observed in original observation space (the pixel space), they belong to a *latent space* and the variable in this space is called a *latent variable*, which we denote it by $\mathbf{z}$. Inferring the latent variable $\mathbf{z}$ from the observed variable $\mathbf{x}$ is an important task in probabilistic modelling. It also leads to discovering the marginal likelihood of the observed data, $p(\mathbf{x})$.

In Bayesian statistics, the inference problem is formalized by finding the *posterior* distribution $p(\mathbf{z}|\mathbf{x})$, which is *intractable* in most cases. *Variational inference* is a set of techniques to approximate this *intractable posterior* analytically. Therefore, it can also be used to approximate the marginal log-likelihood of the observed data. In fact, variational inference provides a lower bound of this log-likelihood, which is called the evidence lower bound (ELBO).

Relations between the variables in the Bayesian inference are described using graphical models. Fig. 3.1 shows a graphical model with one observed variable, $\mathbf{x}$, and one latent variable, $\mathbf{z}$. The solid link shows the generative link $p(\mathbf{x}|\mathbf{z})$ and the dashed link shows the variational approximation link $q(\mathbf{z}|\mathbf{x})$.

Using this model, it can be shown that the ELBO has the following form:

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\big(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big)}_{\text{ELBO}}, \tag{3.1}$$

where $p(\mathbf{z})$ is the *prior* distribution over the latent variable. Both the prior and the variational posterior distributions are usually assumed to be from the exponential family, e.g. Gaussian. The prior is usually a standard Gaussian distribution, i.e. $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$.

Kingma and Welling in [48] and Rezende et al. in [79], proposed a method for performing variational inference using neural networks that scales to large datasets. The proposed method is based on an encoder-decoder structure, which is called variational autoencoder (VAE). The encoder in VAE implements the variational posterior distribution $q(\mathbf{z}|\mathbf{x})$, which given an observed data point outputs the parameters of the posterior distribution, e.g. mean and variance of a Gaussian distribution $q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x}))$. The decoder of VAE is a generative network that implements $p(\mathbf{x}|\mathbf{z})$. The decoder takes as input a sample from the variational posterior distribution and outputs a distribution over the corresponding values of the observed variable. The networks are trained using stochastic backpropagation to maximize the ELBO in Eg. 3.1. For the example that we assume both the prior and variational approximating ditributions are Gaussian, the ELBO has the following form.

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\big(\mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x})) \parallel \mathcal{N}(\mathbf{0}, \mathbf{1})\big). \tag{3.2}$$

Compared to a regular autoencoder objective, the first term in ELBO can be considered as a reconstruction error and the KL divergence is an additional regularization term. It is also worth mentioning that since sampling (from variational posterior) is not a differentiable process, in [48] the authors proposed a method called *reparameterization trick* to make this process differentiable. Please refer to that article for more details.

Since their introduction in 2013, VAEs found their ways in many theoretical and practical research areas inside and outside machine learning. In the next two sections we introduce two problems that are formulated in the framework of VAEs.

3.2 Deep Variational Sufficient Dimensionality Reduction

3.2.1 Sufficient Dimensionality Reduction

Dimensionality reduction is a long-standing problem in machine learning. The initial motivation behind dimensionality reduction was to visualize data and many unsupervised, supervised, and semi-supervised algorithms have been designed for this purpose. Recent advancements in the area of deep learning has pushed the boundaries of the state-of-the-art across a variety of fields including object classification [93, 50, 88], speech recognition [34, 41], sample synthesis [33, 76, 43], and clustering [5, 6], among others. Casting classical algorithms in statistical machine learning within the framework of deep learning has been transformative for the aforementioned applications, e.g. [19]. This inspired us to revisit the problem of sufficient dimensionality reduction and tackle it using the tools provided to us by deep learning.

Sufficient dimensionality reduction (SDR) [32] is a technique that aims to find a low-dimensional representation of data that retains predictive information regarding the label (response) variable. The original paper of SDR presents a way of quantifying this information using information theoretic notions and introduces an iterative algorithm for extracting the features that maximizes this information. Although SDR methods are typically applies to continuous target variables, there exist methods based on distance covariance that can estimate the central subspace, the intersection of all dimensionality reduction subspaces, for discrete target variables [84]. Here we consider the discrete target scenario.

3.2.2 Model description

In this section we first overview SDR and then present a graphical model interpretation of SDR and propose a deep model in the framework of variational autoencoder that approximates the posterior in the graphical model and learns the low-dimensional space efficiently [8].

Consider the frequently encountered goal of regression: predicting a future value for a univariate label $\mathbf{y} \in \mathbb{R}^D$ for a given observation $\mathbf{x} \in \mathbb{R}^p$. In large scale domains, traditional regression methods may require large amounts of training data to avoid overfitting. Therefore, there is a pertinent need for dimensionality reduction methods that replace the original covariate $\mathbf{x}$ with another variable $\mathbf{z} \in \mathbb{R}^d$ that retains most or all of the information and variation in $\mathbf{x}$. When $\mathbf{z}$ retains all the relevant information about $\mathbf{y}$, the dimensionality reduction is said to be *sufficient*: $p(\mathbf{y}|\mathbf{z}) = p(\mathbf{y}|\mathbf{x})$.

Alternatively, sufficient dimension reduction (SDR) techniques can be viewed as methods that find a low dimensional representation such that the remaining degrees of freedom become conditionally independent of the output values, i.e.:

$$\mathbf{y} \perp\!\!\!\perp \mathbf{x} \mid \mathbf{z} \tag{3.3}$$

In other words, $\mathbf{z}(\mathbf{x})$ carries all the information of $\mathbf{x}$ needed to predict a future value for $\mathbf{y}$.

SDR, a graphical model interpretation:

The goal in SDR, as stated in above, is to discover a latent space that can be used for classification. Therefore, compared to the unsupervised dimensionality reduction algorithms, SDR aims to find a latent variable with high predictive power. SDR problem can be explained by either of the graphical models in 3.2. Although the objective of SDR might be derived from 3.2b more intuitively, we argue that the latent space that is learned in Fig. 3.2a carries more information about $\mathbf{x}$ and therefore is more generalizable. In fact, deriving $p(\mathbf{y}|\mathbf{x})$ from the joint distribution $p(\mathbf{x}, \mathbf{y})$ yields a better representation in the latent space that is more robust against overfitting [86]. This is especially true when we parameterize these conditional probabilities using neural networks with hundreds of thousands of parameters. Another benefit of the model in Fig. 3.2a is that it can leverage unlabeled samples to build the latent space and therefore may be used for semi-supervised learning. Whereas the model in Fig. 3.2a can only use the labeled

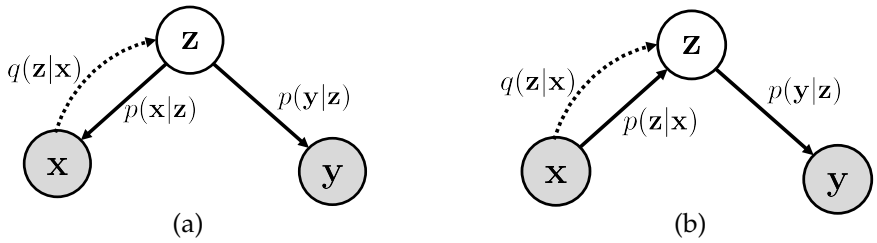


Figure 3.2: Candidate graphical models for sufficient dimensionality reduction

samples.

In fact, we assume that $\mathbf{z}$ can not only generate high-dimensional observations, but can also generate the target variable. Therefore, our objective is to find a latent variable that keeps the information of $\mathbf{X}$ and can be used for classification. In the graphical model of Fig. 3.2a, this is equivalent to maximizing the joint distribution of evidences, $p(\mathbf{X}, \mathbf{Y})$ for all pairs of $(\mathbf{x}_i, \mathbf{y}_i)$. One way to maximize this likelihood is using the variational inference. Based on this graphical model the variational lower bound of the log-likelihood of the joint distribution can be written as:

$$\log p(\mathbf{x}, \mathbf{y}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\big[\log p(\mathbf{x}|\mathbf{z}) + \log p(\mathbf{y}|\mathbf{z})\big] - \mathrm{KL}\big(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big) \tag{3.4}$$

We assume that the prior distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Our goal is to maximize this lower bound. In this model the posterior approximating distribution $q(\mathbf{z}|\mathbf{x})$ plays the role of a probabilistic dimensionality reduction function, $\mathbf{z}(\mathbf{x}) \sim q(\mathbf{z}|\mathbf{x})$.

Deep Variational Learning:

An *encoder* network parameter set φ models $q_\varphi(\mathbf{z}|\mathbf{x})$, which is transformation function that maps high-dimensional observation to the latent space. A *decoder* with parameter set θ models $p_\theta(\mathbf{x}|\mathbf{z})$, which is mapping from the latent space to observation space. And a *classifier* with parameter set ψ models $p_\theta(\mathbf{y}|\mathbf{z})$, which is a mapping from the latent space to the space of labels. Therefore the lower bound in 3.4, denoted by $\mathcal{L}(\mathbf{x}, \mathbf{y})$ can

be rewritten as:

$$\mathcal{L}_{\varphi,\theta,\psi}(\mathbf{x},\mathbf{y}) \quad = \mathbb{E}_{q_\varphi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x}|\mathbf{z})\big] + \mathbb{E}_{q_\varphi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{y}|\mathbf{z})\big] - \mathrm{KL}\big(q_\varphi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})\big). \quad (3.5)$$

Our approach to solve the SDR problem is to maximize this lower bound using deep variational learning, and we call it deep variational SDR (DVSDR). By maximizing $\mathbb{E}_{q_\varphi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{y}|\mathbf{z})\big]$ for each labeled sample, we build a latent representation that has a high predictive power and this is exactly what SDR aims for.

Semi-supervised DVSDR:

In many cases in practical problems, the label information is scarce and therefore a semi-supervised learning method should be exploited. DVSDR can be easily adopted to a semi-supervised learning setting. Suppose $\mathbf{X}_L \subset \mathbf{X}$ is the set of all labeled samples for which we have the label set $\mathbf{Y}$, and $\mathbf{X}_U = \mathbf{X} \backslash \mathbf{X}_L$ is the set of unlabeled samples. Since our latent variable is inferred purely based on our observation variable (and not label variable), we can split the objective function to two parts, $\mathcal{L}^{\ell}_{\varphi,\theta,\psi}(\mathbf{x},\mathbf{y})$ and $\mathcal{L}^{u}_{\varphi,\theta}(\mathbf{x})$, corresponding to labeled and unlabeled subsets of the data. Therefore:

$$\{\varphi^\star, \theta^\star, \psi^\star\} = \arg\max_{\varphi,\theta,\psi} \sum_{\mathbf{x}\in\mathbf{X}^\ell} \mathcal{L}^{\ell}_{\varphi,\theta,\psi}(\mathbf{x},\mathbf{y}) + \sum_{\mathbf{x}\in\mathbf{X}^u} \mathcal{L}^{u}_{\varphi,\theta}(\mathbf{x}) \quad (3.6)$$

In fact, in the semi-supervised setting, the low-dimensional latent variable for unsupervised samples only reconstructs observations, but for supervised samples generates both observations and labels.

3.2.3 Experiment Results

We evaluate the performance of DVSDR in two different tasks: Classification and sample generation.

Classification:

We compare the classification performance of the proposed model with Adversarial Autoencoder (AAE) [61] and semi-supervised learning with deep generative models [47] that are both deep generative models with bottleneck.

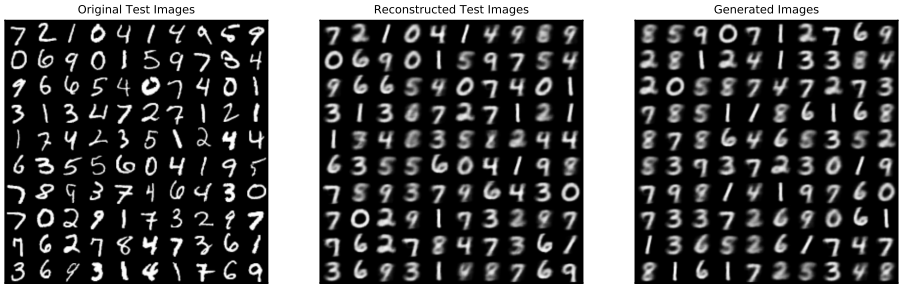
	MNIST (All labeled samples)	MNIST (1000 labeled samples)
VAE (M1+M2)	0.96 ± 0.03	2.40 ± 0.02
AAE	0.86 ± 0.03	$\mathbf{1.60 \pm 0.08}$
DVSDR	$\mathbf{0.80 \pm 0.04}$	2.1 ± 0.06

Table 3.1: Classification Error rate

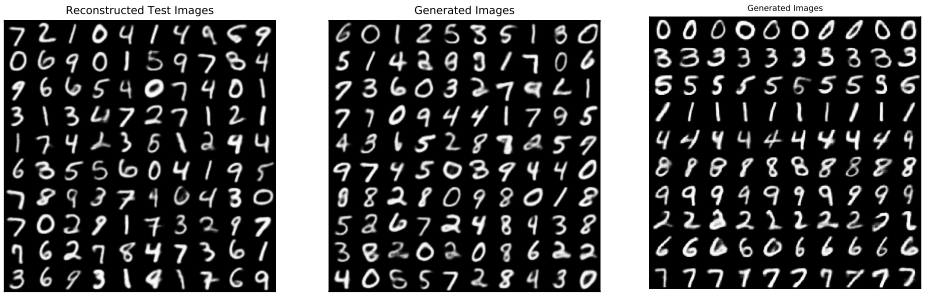
As we can see our model outperforms the other two models when we use all the label information, but when only 1000 labeled samples are used, AAE performs better. This is because AAE directly impose a supervised loss on the latent space by matching the distribution of the latent space with a categorical distribution.

Generation:

Sample generation is a bi-product of the DVSDR algorithm. Using the structure of DVSDR not only can we preserve the information in the observation set while reconstructing the input samples, but also we can generate novel data by sampling from the prior $p(\mathbf{z})$ and feeding it to the decoder. Fig. 3.3 and 3.4 in the appendix show the results of sample reconstruction/generation for MNIST and Fashion MNIST datasets. For MNIST we use DVSDR with latent dimension of 2 and 15 and for Fashion MNIST dimension of 10. We can generate high quality images using the proposed model. When we fit a mixture of Gaussian over the latent space of the model for the MNIST dataset, we can generate samples from different classes, this is because the representations of the points in each class in the latent space are very well separated.



(a) 2-dimensional latent space



(b) 15-dimensional latent space (c)

Figure 3.3: (a,b) Reconstructed and generated images using DVSDR with 2-D and 15-D latent space. (c) Fitting a mixture of Gaussian with 10 components on the latent space and sampling from each component

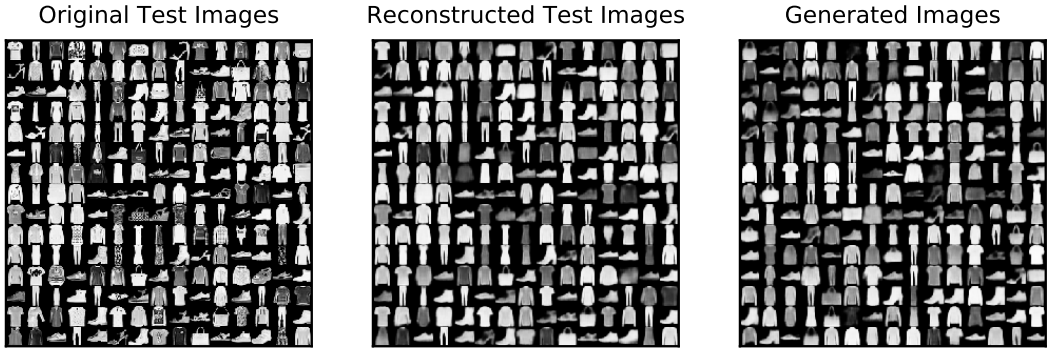


Figure 3.4: Fashion MNIST. 10-D latent space

3.3 Joint Autoencoders for Dis-Entanglement

The problem of feature disentanglement has been explored in the literature, for the purpose of image and video processing and text analysis [26, 96, 45, 40, 21, 2]. State-of-the-art methods for disentangling feature representations rely on the presence of many labeled samples. Here, we present a novel method for disentangling factors of variation in data-scarce regimes. Specifically, we explore the application of feature disentangling for the problem of supervised classification in a setting where few labeled samples exist, and there are no unlabeled samples for use in unsupervised training. Instead, a similar datasets exists which shares at least one direction of variation with the sample-constrained datasets. We train our model end-to-end using the framework of variational autoencoders and are able to experimentally demonstrate that using an auxiliary dataset with similar variation factors contribute positively to classification performance, yielding competitive results with the state-of-the-art in unsupervised learning.

3.3.1 Problem statement and prior works

In machine learning, samples in a dataset originate via complicated processes driven by a number of underlying factors. Individual factors lead to independent directions of variations in the observed samples, while the accumulation of factors give rise to the rich structure characteristic of these datasets. The underlying factors often interact in complicated and unpredictable ways, and appear tightly *entangled* in the raw data. Being able to tease apart the effect of underlying factors is a fundamental challenge in understanding these datasets.

For instance, a dataset containing images of natural scenery may be subject to variation in lighting conditions, camera elevation, and the appearance of the scene itself. Controlling and restraining variation at data acquisition time is difficult, and limits the number of acceptable samples in the dataset. On the other hand, capturing annotations for every direction of variation is time-consuming and infeasible. Therefore, designing methods that automatically learn to separate out underlying factors (known and

unknown) is relevant for many applications in machine learning.

One area that has enjoyed tremendous success for separating factors of variation is supervised learning. The representations learned here aim to satisfy a specific task that is driven by the explicit labels in the dataset. Therefore, these representations are invariant to factors of variation that are uninformative for solving the task at hand. For example, when identifying individuals in a school yearbook, the identity of the person is paramount compared to their facial expression. Hence, a simple method that simply discards the irrelevant variation in expression will perform really well. Learning invariant representations, however, require many samples and comes at the cost of needing to train a new model for a closely related task that depends on an alternative direction of variation. It would seem reasonable then to desire a strategy that captures all directions of variation in a single model in a *disentangled* manner allowing one to infer all factors for a given sample in the absence of labels for each factor.

Current state-of-the-art strategies for disentangling factors of variation mostly fall victim to the challenges in deep learning and rely on the presence of abundant data samples. In [51], the authors were able to accurately separate out lighting, pose, and shape while sampling seemingly unlimitedly from an auxiliary generative model that creates samples with different variations. The results presented in [77, 62] also build upon datasets containing often hundreds of thousands of samples. Whereas [47, 87] use very few samples in their training process, these methods are semi-supervised and have access to unlabeled samples from the same dataset following the same statistical distribution.

In this work, we explore classification in a data-scarce scenario where not only are there few labeled samples available, there are also no unlabeled samples from which one could perform semi-supervised training. These situations commonly arise in medical imaging datasets, e.g., pancreatic cancer MRI images are scarce whereas breast cancer MRI images are abundant ([35] and references therein). In such a situation, we ask whether one can employ a secondary dataset, with many samples, similar content, but different style, to improve the performance of a benchmark classification model. What

remains to be demonstrated is how to learn good intermediate representations that can be shared across tasks and use the disentanglement process of the secondary dataset to effectively disentangle the factors of variation in the primary dataset of interest. Essentially we are entangling together the feature disentangling of two similar datasets. This is the focus of the work below.

3.3.2 Model description

In this work [7], we consider a situation where we are given a labeled dataset, $\mathbf{x}$, with limited number of points. We denote the label variable by ℓ. We also have access to another dataset $\mathbf{y}$ with a larger number of points that share the same categories as $\mathbf{x}$. However, the underlying distribution of the datasets are different. Let us denote the distribution for $\mathbf{x}$ and $\mathbf{y}$ by $p(\mathbf{x})$ and $p(\mathbf{y})$, respectively. Suppose that our goal is to classify unseen data points that come from $p(\mathbf{x})$, i.e. to maximize $p(\ell|\mathbf{x})$. Building a classifier that simply uses $\mathbf{x}$ can lead to low accuracy and overfitting, due to its small size. Therefore we want to leverage the information of $\mathbf{y}$ about the label variable and build a model that can classify the points from $p(\mathbf{x})$ with higher accuracy.

Our approach to address this problem is to disentangle the features in $\mathbf{x}$ and $\mathbf{y}$ that contribute in predicting the label variable (i.e., content) from the features that contribute to the style of $\mathbf{x}$ and $\mathbf{y}$. Consider the graphical model in Fig. 3.5. We assume there are

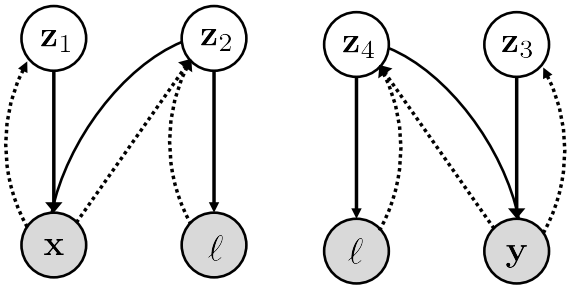


Figure 3.5: Graphical models of the method.

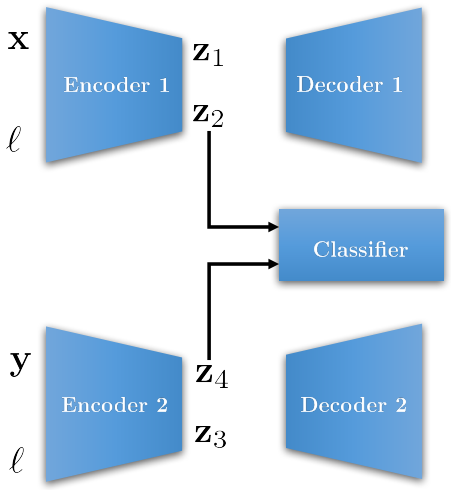


Figure 3.6: Network structure of the method

two pairs of latent variables that describe each of $\mathbf{x}$ and $\mathbf{y}$. Based on this figure, suppose that $\mathbf{z}_1$ and $\mathbf{z}_2$ generate samples in dataset $\mathbf{x}$ and $\mathbf{z}_3$ and $\mathbf{z}_4$ generated samples in dataset $\mathbf{y}$. If we assume that $\mathbf{z}_2$ and $\mathbf{z}_4$ are the latent variables that carry all the information about the label variable ℓ then $p(\ell|\mathbf{z}_2) = p(\ell|\mathbf{z}_4)$. Considering the same prior distributions over $\mathbf{z}_2$ and $\mathbf{z}_4$, i.e. $\mathcal{N}(0, I)$, we can guarantee the disentanglement of latent features by asserting that $p(\mathbf{z}_2|\ell) = p(\mathbf{z}_4|\ell)$. However, these posteriors are intractable. To approximate them we use the framework of variational inference where $p(\mathbf{z}_2|\ell)$ and $p(\mathbf{z}_4|\ell)$ are approximated by $q(\mathbf{z}_2|\mathbf{x}, \ell)$ and $q(\mathbf{z}_4|\mathbf{y}, \ell)$, respectively. Therefore, by matching these approximating distribution, we guarantee that only $\mathbf{z}_2$ and $\mathbf{z}_4$ carry information regarding the label ℓ (i.e., content) and therefore are disentangled from $\mathbf{z}_1$ and $\mathbf{z}_3$ respectively which represent style.

All the conditional distributions on the graphical models in Fig. 3.5 are parameterized by the neural networks depicted in Fig. 3.6. The joint model here builds on earlier work in [82] where an autoencoder and a discriminator were trained in the framework of contractive discriminative analysis for semi-supervised learning. Here, we use the variational autoencoding [48] approach to jointly train two networks that simultaneously extract shared discriminative features present in the primary and secondary

datasets. This architecture is reminiscent of Domain Separation Networks [16]. The proposed JADE model, however, focuses on a shared classifier for improved classification and joint disentanglement instead of a shared encoder and decoder.

The variational lower bound on the joint distribution of the observations is:

$$\begin{aligned}\log p(\mathbf{x},\ell) \geq \mathcal{L}(\mathbf{x},\ell) = \mathbb{E}_{\substack{q(\mathbf{z}_1|\mathbf{x}) \\ q(\mathbf{z}_2|\mathbf{x},\ell)}}\big[\log p(\mathbf{x}|\mathbf{z}_1,\mathbf{z}_2)\big] + \mathbb{E}_{q(\mathbf{z}_2|\mathbf{x},\ell)}\big[\log p(\ell|\mathbf{z}_2)\big] \\ -\mathrm{KL}\big(q(\mathbf{z}_1|\mathbf{x}) \parallel p(\mathbf{z}_1)\big) - \mathrm{KL}\big(q(\mathbf{z}_2|\mathbf{x},\ell) \parallel p(\mathbf{z}_2)\big)\big) \\ \log p(\mathbf{y},\ell) \geq \mathcal{L}(\mathbf{y},\ell) = \mathbb{E}_{\substack{q(\mathbf{z}_3|\mathbf{y}) \\ q(\mathbf{z}_4|\mathbf{y},\ell)}}\big[\log p(\mathbf{x}|\mathbf{z}_3,\mathbf{z}_4)\big] + \mathbb{E}_{q(\mathbf{z}_4|\mathbf{y},\ell)}\big[\log p(\ell|\mathbf{z}_4)\big] \\ -\mathrm{KL}\big(q(\mathbf{z}_3|\mathbf{x}) \parallel p(\mathbf{z}_3)\big) - \mathrm{KL}\big(q(\mathbf{z}_4|\mathbf{x},\ell) \parallel p(\mathbf{z}_4)\big)\big)\end{aligned} \tag{3.7}$$

We would like to maximize the sum over the above lower bounds. The approximating distributions are from exponential family (Gaussian) and to match them we assume that for the samples that are from the same class in the two datasets, we want to minimize $\mathrm{KL}\big(q(\mathbf{z}_2|\mathbf{x},\ell) \parallel q(\mathbf{z}_4|\mathbf{y},\ell)\big)$. Given this condition, the overall objective of the model is:

$$\max_{\Theta} \mathcal{L}(\mathbf{x},\ell) + \mathcal{L}(\mathbf{y},\ell) - \mathrm{KL}\big(q(\mathbf{z}_2|\mathbf{x},\ell) \parallel q(\mathbf{z}_4|\mathbf{y},\ell)\big) \tag{3.8}$$

where Θ represents the entire parameter set of neural networks.

3.3.3 Experiments

Datasets: Our framework addresses the problem of performing supervised classification in data-scarce regimes where there exists a secondary dataset that has at least one direction of variation in common with the primary sample-constrained dataset. In our experiments we emulate this scenario with commonly used datasets such as MNIST [55] and SVHN [69]. Because MNIST is relatively easier to learn, even with very few samples, we select SVHN as the sample-constrained primary dataset that is difficult to learn, and use the entirety of MNIST as the secondary dataset. These datasets differ in appearance and style: whereas MNIST is gray-scale and comes in 28×28 pixel images, SVHN has three color channels and comes in 32×32 pixel images. However, both

datasets represent the same content (i.e., digit values) across different styles. This similarity in content of both datasets is what makes MNIST a good secondary dataset to boost SVHN's supervised classification performance.

Model Comparison: To evaluate the performance of our framework, we first develop a benchmark for supervised classification of SVHN. Here, we choose a relatively powerful convolutional neural network (CNN) architecture combined with a multi-layer perceptron (MLP) as the supervised classification model. The CNN architecture comprises of 4 layers of 3×3 spatial convolutions ($\{64, 96, 64, 8\}$ filters respectively) followed by ReLU and interspersed with 3 layers of $2\times$ max-pooling. The MLP contains 3 blocks of 500-dimensional fully connected layers, followed by ReLU and Dropout ($p = 0.5$) layers [90]. A 10-dimensional bottleneck layer was placed in between the CNN and the MLP to encourage only important features from being retained. A final softmax layer is present at the end of the network for 10-way classification. The loss for this model is measured using categorical cross-entropy. This architecture is referred to as *single classifier* (i.e., benchmark).

A simple extension of above setup is a model that jointly trains SVHN and MNIST on a shared MLP classifiers using features extracted from separate CNN feature extractors, one per dataset. The CNN used for SVHN and the MLP follow the same architecture as the benchmark above. The CNN architecture for MNIST comprises of 3 layers of 3×3 spatial convolutions ($\{32, 32, 16\}$ filters respectively) followed by ReLU and interspersed with 3 layers of $2\times$ max-pooling. A 10-dimensional bottleneck layer was placed in between the CNN for MNIST and the shared MLP to capture the latent features of MNIST. Feature-extracted samples from both datasets are fed into the shared MLP in alternation and trained jointly. The loss of the system is the sum of the categorical cross-entropy losses for both datasets on the shared classifier. This setup is called *paired classifier*.

Finally, the proposed model (outlined in Fig. 3.6) extends upon the previous two methods by adding a decoder network to reconstruct the 10-dimensional latent representations from each of the CNN feature extractors. To encourage disentanglement of

Method	SVHN (1000 samples)	MNIST (45K samples)
VAE (M1+M2) [47]	36.02 ± 0.10	-
Siddharth et al. [87]	28.71 ± 2.38	-
Single Classifier (benchmark)	32.31 ± 1.56	-
Paired Classifier	30.17 ± 2.77	0.82 ± 0.05
JADE (proposed)	29.08 ± 0.92	0.72 ± 0.03

Table 3.2: Classification error rates for SVHN on limited data: 100 samples per each class. Error rates calculated using the entirety of SVHN's test set. Results of our experiments are averaged over 3 runs. We observe improved SVHN classification performance without sacrificing near state-of-the-art performance on MNIST.

features in the latent space, and to perform factor separation in a way that the MLP classifier is only given content-related features (i.e., digit values), we increase the size of the latent spaces from 10 to 20 dimensions. However, only 10 of the latent dimensions resulting from each CNN are passed into the shared MLP, essentially keeping consistent with the previous method in terms of classifier capacity. All 20 latent dimensions are used to reconstruct the inputs via a decoder that identically mirrors the corresponding CNN ($2\times$ up-sampling layers used in place of $2\times$ max-pooling). Losses are defined in Section 2. Due to the autoencoding structure of this model, we refer to it as *JADE: Joint Autoencoders for Dis-Entanglement*.

Discussion about the results: The results of our experiments have been presented in Table 3.2. Here we compare the results of the single classifier (i.e., benchmark model), paired classifier, and proposed model (JADE) alongside those from Kingma et al. [47] and Siddharth et al. [87]. It is worth pointing out that the former 3 models are trained only on 1000 labeled sample from SVHN, whereas the cited models use the remainder of the SVHN training dataset in an unsupervised fashion. We, on the other hand, use all of the MNIST dataset to train the paired classifier in JADE.

These results demonstrate that when dealing with sample-constrained regimes without unlabeled samples, one can use a similar dataset with at least one shared direction of variation to improve classification performance. This can be seen when comparing the performance of a single classifier ($32.31/01.56$) with that of a paired classifier ($30.17/02.77$). On top of this, we see that the JADE model which learns to jointly disentangle SVHN and MNIST features performs even better than the former methods,

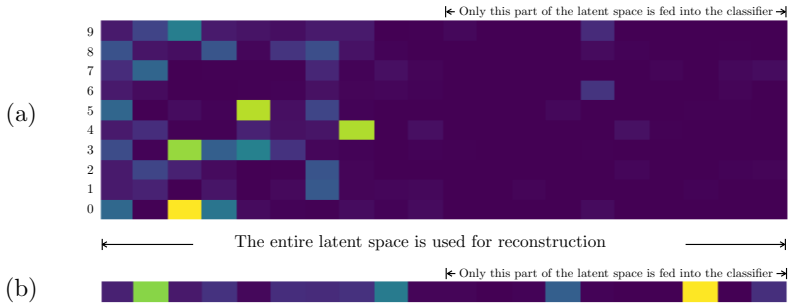


Figure 3.7: **(a)** variance normalized activations of latent space parameters, averaged over 500 random samples from each of 10 classes in SVHN; when content is fixed, the part of the latent space that feeds into the classifier exhibits weaker variance in activations compared to the part of the latent space that seemingly represents style over the 500 samples. **(b)** variance normalized activations of latent space parameters for 2500 random samples from SVHN spanning various style and content; all 20 latent space parameters fire for random splits of the data.

sitting at $29.08/00.92$. This is in line with our hypothesis that only the directions of variation shared between MNIST and SVHN (i.e., content) will contribute positively to classification performance on SVHN, and other factors of variation should be disentangled.

We hypothesize that actively attempting to disentangle variation factors (i.e., in JADE) is better than allowing the network to attempt to discard uninformative factors (i.e., paired classifier) given the sample-constrained regime. To assert that the JADE setup is indeed disentangling variation factors, we conduct the following simple experiment: observe the variation in latent space values as different types of samples are passed into the network. In Fig. 2a, we have shown how latent activations change when the SVHN CNN is fed with 500 samples from the same class (i.e., same content but varying style). These activations are shown for the 20 latent parameters (of which only 10 are passed into the MLP classifier, and all used for reconstruction) across 10 classes of digits in MNIST. We observe that in this setup where content is fixed, the normalized variance of the latent variables that are fed into the MLP classifier is much lower than the variance of latent variables that are solely used for reconstruction. In Fig. 2b, we observe an interesting and complementary phenomena when we pass in 2500 randomly selected test samples into the SVHN CNN. Here, both the style and the content vary between

input samples, and we observe that all 20 latent parameters are active given the varying input. These observations suggest that JADE is able to successfully disentanglement content and style in low-data SVHN using the help of MNIST as an auxiliary similar dataset.

3.4 Summary

We presented a brief introduction to VAEs in this chapter and investigated two problems in this framework. We considered the SDR problem, where the high-dimensional observation is transformed to a low-dimensional sub-space in which the information of the observations regarding the label variable is preserved. We proposed DVSDR, a deep variational approach for sufficient dimensionality reduction, which can be adopted to semi-supervised learning setting. We showed in our experiments that DVSDR performs competitively on classification tasks while being able to generate novel data samples.

We also explored the application of feature disentangling for the problem of supervised classification in a setting where few labeled samples exist, and there are no unlabeled samples for use in unsupervised training. Instead, a similar datasets exists which shares at least one direction of variation with the sample-constrained datasets. We train our model end-to-end using the framework of variational autoencoders and experimentally demonstrated that using a secondary dataset with similar content to SVHN leads to improvements in supervised classification performance.

Chapter 4

Robust Locally-Linear Controllable Embedding

4.1 Contributions

Embed-to-control (E2C) [103] is a model for solving high-dimensional optimal control problems by combining variational auto-encoders with locally-optimal controllers. However, the current E2C model suffers from two major drawbacks: **1)** its objective function does not correspond to the likelihood of the data sequence and **2)** the variational encoder used for embedding typically has large variational approximation error, especially when there is noise in the system dynamics. In this chapter, we present a model for learning locally-linear controllable embedding (RCE) that is robust against the noise in the dynamics of the system. Our model has a bottleneck encoder-decoder structure and estimates the predictive conditional density of the future observation given the current one and the action. Although the bottleneck provides a natural embedding candidate for control, our RCE model introduces additional specific structures in the generative graphical model so that the model dynamics can be robustly linearized. Our experiment results show that RCE outperforms the existing E2C model, especially in the regime where the underlying dynamics is noisy.

4.2 Problem statement and prior work

Model-based locally optimal control algorithms are popular in controlling non-linear dynamical systems with continuous state and action spaces. Algorithms from this class such as differential dynamic programming (DDP) [44], iterative linear quadratic regulator (iLQR) [58], and iterative linear quadratic Gaussian (iLQG) [95] have been successfully applied to a variety of complex control problems [3, 94, 57, 72]. The general idea of these methods is to iteratively linearize the non-linear dynamics around the current trajectory and then use linear quadratic methodology to derive Riccati-like equations to improve the trajectory. However, these methods assume that the model of the system is known and need relatively low-dimensional state representations. These requirements limit their usage in control of dynamical systems from raw sensory data (e.g., image and audio), a scenario often seen in modern reinforcement learning (RL) systems.

Although both model-based RL and methods to find low-dimensional representations that are appropriate for control (see e.g., [15]) have a long history, they have recently witnessed major improvements due to the advances in the field of deep learning. Deep autoencoders [54, 100] have been used to obtain low-dimensional representations for control, and deep generative models have been used to develop new model-based RL algorithms. However, what is desirable in model-based locally optimal control algorithms is a representation that can be used for learning a model of the dynamical system and can also be systematically incorporated into the existing tools for planning and control. One such model is embed to control (E2C) [103]. E2C turns the problem of locally optimal control in high-dimensional non-linear systems into one of identifying a low-dimensional latent space in which we can easily perform locally optimal control. The low-dimensional latent space is learned using a model based on variational autoencoders (VAEs) [49, 79] and the iLQG algorithm [95] is used for locally optimal control.

While the idea of E2C is intriguing, it suffers from two major statistical deficiencies. **Firstly**, to induce the lower-dimensional embedding, at each time step t, E2C models the pair-marginal distribution of two adjacent observations $(\mathbf{x}_t, \mathbf{x}_{t+1})$. As a result, its

loss function effectively is the sum over the pair-marginals, which is clearly not the data likelihood for the entire trajectory. Moreover, at every time step t, E2C needs to enforce the consistency between the posterior of the embedding and the predictive distribution of the future embedding by minimizing their KL divergence. These all indicate that the E2C loss is not a lower bound of the likelihood of the data. The practice of modeling the pair-marginal of $(\mathbf{x}_t, \mathbf{x}_{t+1})$ using a latent variable model also imposes a Gaussian prior on the embedding space, which might be in conflict with the locally-linear constraint that we would like to impose. **Secondly**, the variational inference scheme in E2C attempts to approximate the posterior of the latent embedding via a recognition model that does not depend on the future observation $\mathbf{x}_{t+1}$. We believe that this is done out of necessity, so that the locally-linear dynamics can be encoded as a constraint in the original E2C model. In an environment where the future is uncertain (e.g., in the presence of noise or other unknown factors), the future observation carries significant information about the true posterior of the latent embedding. Thus, a variational approximation family that does not take future observation into account, while approximating the posterior, will result in a large variational approximation error, leading to the learning of a sub-optimal model that underperforms, especially when the dynamics is noisy.

To address these issues, we take a more systematic view of the problem. Instead of mechanically applying VAE to model the pair-marginal density, we build on the recent bottleneck conditional density estimator (BCDE) [85] and directly model the predictive conditional density $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$. The BCDE model introduces a bottleneck random variable $\mathbf{z}_t$ in the middle of the information flow from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$. While this bottleneck provides a natural embedding candidate for control, these embeddings need to be structured in a way to respect the locally linear constraint of the dynamics. Our proposed model, *robust controllable embedding* (RCE), provides a rigorous answer to this question in the form of a generative graphical model. A key idea is to explicitly treat the reference linearization point in the locally-linear model as an additional random variable. We also propose a principled variational approximation of the embedding posterior that takes the future observation into account and optimizes a variational lower bound of the likelihood of the data sequence. This allows our framework to provide a clean

separation of the generative graphical model and the amortized variational inference mechanism (e.g., the recognition model).

After a brief overview of locally linear control and E2C in Section 4.3, we present our proposed RCE model in Section 4.4. Unlike E2C, RCE directly models the conditional density of the next observation given the current one via a form of bottleneck conditional density estimators [85]. In Section 4.4, we first describe the RCE's graphical model in details and then present the proposed variational approximation of the embedding's posterior. In Section 4.5, we apply RCE to four RL benchmarks from [103] and show that it consistently outperforms E2C in both prediction and planning. Crucially, we demonstrate the robustness of RCE: as the dynamics becomes more noisy, RCE continues to perform reasonably well while E2C's performance degrades sharply.

4.3 Preliminaries

In this section, we first define the non-linear control problem that we are interested to solve, and then provide a brief overview of stochastic locally optimal control and the E2C model. We also motivate our proposed robust controllable embedding (RCE) model that will be presented in Section 4.4.

4.3.1 Problem Formulation

We are interested in controlling the non-linear dynamical systems of the form

$$\mathbf{s}_{t+1} = f_{\mathcal{S}}(\mathbf{s}_t, \mathbf{u}_t) + \mathbf{n}^{\mathcal{S}}, \tag{4.1}$$

where $\mathbf{s}_t \in \mathbb{R}^{n_s}$ and $\mathbf{u}_t \in \mathbb{R}^{n_u}$ denote the state and action of the system at time step t, $\mathbf{n}^{\mathcal{S}} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\mathbf{n}^{\mathcal{S}}})$ is the Gaussian system noise, and $f_{\mathcal{S}}$ is a smooth non-linear system dynamics. Note that in this case $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{u}_t)$ would be the multivariate Gaussian distribution $\mathcal{N}\big(f_{\mathcal{S}}(\mathbf{s}_t, \mathbf{u}_t), \mathbf{\Sigma}_{\mathbf{n}^{\mathcal{S}}}\big)$. We assume that we only have access to the high-dimensional

observation $\mathbf{x}_t \in \mathbb{R}^{n_x}$ of each state $\mathbf{s}_t$ ($n_s \ll n_x$) and our goal is to learn a low-dimensional latent state space $\mathcal{Z} \subset \mathbb{R}^{n_z}$ ($n_z \ll n_x$) in which we perform optimal control.

4.3.2 Stochastic Locally Optimal Control

Stochastic locally optimal control (SLOC) is based on the idea of controlling the non-linear system (4.1), along a reference trajectory $\{\bar{\mathbf{s}}_1, \bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{s}}_H, \bar{\mathbf{u}}_H, \bar{\mathbf{s}}_{H+1}\}$, by transforming it to a time-varying linear quadratic regulator (LQR) problem

$$\begin{aligned} \min_{\mathbf{u}_{1:T}} \mathbb{E}\left[\sum_{t=1}^{T} \left((\mathbf{s}_t - \mathbf{s}^f)^\top \mathbf{Q}(\mathbf{s}_t - \mathbf{s}^f) + \mathbf{u}_t^\top \mathbf{R}\mathbf{u}_t\right)\right] \\ \text{s.t} \quad \mathbf{y}_{t+1} = \mathbf{A}_t\mathbf{y}_t + \mathbf{B}_t\mathbf{v}_t, \end{aligned} \tag{4.2}$$

where $\mathbf{s}^f$ is the final (goal) state, $\mathbf{Q}$ and $\mathbf{R}$ are cost weighting matrices, $\mathbf{y}_t = \mathbf{s}_t - \bar{\mathbf{s}}_t$, $\mathbf{v}_t = \mathbf{u}_t - \bar{\mathbf{u}}_t$, $\bar{\mathbf{s}}_{t+1} = f_{\mathcal{S}}(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)$, $\mathbf{A}_t = \frac{\partial f_{\mathcal{S}}}{\partial \mathbf{s}}(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)$, and $\mathbf{B}_t = \frac{\partial f_{\mathcal{S}}}{\partial \mathbf{u}}(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)$. Eq. 4.2 indicates that at each time step t, the non-linear system has been locally approximated with a linear system around the reference point $(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)$ as

$$\begin{aligned} \mathbf{s}_{t+1} \approx f_{\mathcal{S}}(\mathbf{s}_t, \mathbf{u}_t) + \left[\frac{\partial f_{\mathcal{S}}}{\partial \mathbf{s}}(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)\right](\mathbf{s}_t - \bar{\mathbf{s}}_t) \\ + \left[\frac{\partial f_{\mathcal{S}}}{\partial \mathbf{u}}(\bar{\mathbf{s}}_t, \bar{\mathbf{u}}_t)\right](\mathbf{u}_t - \bar{\mathbf{u}}_t). \end{aligned} \tag{4.3}$$

The RHS of Eq. 4.2 sometimes contains an offset $\mathbf{c}_t$ resulted from the linear approximation and/or noise

$$\mathbf{y}_{t+1} = \mathbf{A}_t\mathbf{y}_t + \mathbf{B}_t\mathbf{v}_t + \mathbf{c}_t. \tag{4.4}$$

Eq. 4.4 can be seen as

$$\begin{bmatrix} \mathbf{y}_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_t & \mathbf{c}_t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{y}_t \\ 1 \end{bmatrix} + \begin{bmatrix} \mathbf{B}_t \\ 0 \end{bmatrix} \mathbf{v}_t,$$

and thus, can be easily transformed to the standard from (4.2) by adding an extra dimension to the state as

$$\mathbf{y}'_t = \begin{bmatrix} \mathbf{y}_t \\ 1 \end{bmatrix}, \quad \mathbf{A}'_t = \begin{bmatrix} \mathbf{A}_t & \mathbf{c}_t \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B}'_t = \begin{bmatrix} \mathbf{B}_t \\ 0 \end{bmatrix}.$$

Locally optimal actions in Eq. 4.2 can be computed in closed-form by solving the local LQRs (4.3) using the value iteration algorithm.

Since the quality of the control depends on the quality of the reference trajectory, SLOC algorithms are usually iterative (e.g., iLQR and iLQG), and at each iteration generate a better reference trajectory. At the abstract level, a SLOC algorithm operates as follows: at each iteration k, a reference trajectory is generated using the current policy $\pi^{(k)}$, the LQR approximation of the non-linear system is computed around this reference trajectory, and finally the next policy $\pi^{(k+1)}$ is computed by solving this LQR. The algorithm stops after a fixed number of iterations, e.g., 100.

As mentioned in Section 4.3.1, since we do not have access to the true state $\mathbf{s}$, we perform the optimal control in the low-dimensional latent space $\mathbf{z}$ learned from the observations $\mathbf{x}$. Thus, all the $\mathbf{s}$'s in this section should be replaced by $\mathbf{z}$ in the following sections.

4.3.3 The Embed to Control (E2C) Model

We now return to the assumption that we only observe a finite number of high-dimensional sensory data (e.g., images) $\mathbf{x}_t \in \mathbb{R}^{n_x}$ from the system. We denote the high-dimensional observation sequence by $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$. Note that the observations are selected such that the sequence $\mathbf{X}$ is Markovian. For example, $\mathbf{x}$ could be a set of buffered observed images of the system that encodes all the information about the past. Depending on the system, this set may have only one or multiple images.

It is clear that direct control in $\mathbb{R}^{n_x}$ is complicated because of its high-dimensional nature. However, when the true underlying state space is low-dimensional, it would

be possible to embed the high-dimensional observations in a low-dimensional latent space $\mathcal{Z}$, in a way that the dynamics of the system can be captured by a much simpler model, which can then be used for optimal control. This general strategy is known as embed to control (E2C) [103]. Note that a suitable embedding function is sufficient for model-based control, we do not need to recover the true state $\mathbf{s}_t$.

We denote by $\mathbf{z}_t$ the low-dimensional embedding of $\mathbf{x}_t$. E2C first introduces a new variable $\hat{\mathbf{z}}_{t+1}$ as the result of applying $\mathbf{u}_t$ to the latent dynamics $f_{\mathcal{Z}}$, i.e.,

$$\hat{\mathbf{z}}_{t+1} = f_{\mathcal{Z}}(\mathbf{z}_t, \mathbf{u}_t) + \mathbf{n}_t^{\mathcal{Z}}, \tag{4.5}$$

where $\mathbf{n}_t^{\mathcal{Z}}$ denotes the transition noise in the latent space. E2C employs the pair $(\mathbf{z}_t, \hat{\mathbf{z}}_{t+1})$ as the latent variables that model the pair-marginal $p(\mathbf{x}_t, \mathbf{x}_{t+1})$. It uses the variational recognition network $q(\mathbf{z}_t|\mathbf{x}_t)$, while forcing $q(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \mathbf{u}_t)$ to be the generative dynamics of Eq. 4.5. This leads to the following lower bound of the pair-marginal

$$\begin{aligned} \log p(\mathbf{x}_t, \mathbf{x}_{t+1}|\mathbf{u}_t) &\geq \mathcal{L}_t^{\text{bound}}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}) \\ &= \mathbb{E}_{q(\mathbf{z}_t|\mathbf{x}_t)q(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \mathbf{u}_t)}\big[\log p(\mathbf{x}_t|\mathbf{z}_t) + \log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}) - \mathrm{KL}(q(\mathbf{z}_t|\mathbf{x}_t) \parallel p(\mathbf{z}_t))\big] \end{aligned} \tag{4.6}$$

Local linearization of the dynamics is enforced inside the recognition model $q(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \mathbf{u}_t)$, where mapping from a linearization point $\bar{\mathbf{z}}_t$ to the linearization matrices are estimated via neural networks.

Finally, we want $\mathbf{z}_{t+1}$ to be both the embedding of $\mathbf{x}_{t+1}$ and the result of applying $\mathbf{u}_t$ to $\mathbf{z}_t$. E2C attempts to enforce this temporal consistency criterion by encouraging the distributions of $\hat{\mathbf{z}}_{t+1}$ and the next step embedding $\mathbf{z}_{t+1}$ to be similar (in the KL sense). Enforcing the temporal consistency leads to the modified objective

$$\mathcal{L}_t = -\mathcal{L}_t^{\text{bound}} + \lambda \mathrm{KL}\big(q(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \mathbf{u}_t) \parallel q(\mathbf{z}_{t+1}|\mathbf{x}_{t+1})\big), \tag{4.7}$$

where λ is an additional hyperparameter of the model. We note that neither of the two objectives $\sum_t \mathcal{L}_t^{\text{bound}}$ and $\sum_t \mathcal{L}_t$ is a lower bound of the data likelihood $p(\mathbf{X})$. The fact that E2C does not optimize a proper lower bound of the data likelihood has also been

observed by [46].

Compared to E2C, our method is based on introducing a graphical model that clearly separates the generative model from the variational recognition model. This enables us to handle noise in the system and avoid heuristic terms in the objective functions that need extra hyperparameter tuning. Furthermore, we can optimize a lower bound on the likelihood of the data sequence using a better-designed recognition model more robust w.r.t. noise. Note that our goal is not to purely obtain the best predictive power as in [46], but to design a predictive model that yields a suitable embedding representation for locally optimal control. Unlike [46] which does not report control performance, our experiments focus on the performance of the controller under various noise regime. In the next section, we describe our proposed RCE model and demonstrate how it addresses the aforementioned issues of E2C.

4.4 Model Description

In this section, we first introduce our graphical model that represents the relation between the observations and latent variables in our model. We then derive a lower bound on the likelihood of the observation sequence. The objective of training in our model is to maximize this lower bound. Finally, we describe the details of the method we use for planning in the latent space learned by our model.

4.4.1 Graphical Model

We propose to learn an action-conditional density model of the observations $\mathbf{x}_{1:N}$. Similar to E2C, we assume that the observation sequence is Markovian. Thus, optimizing the likelihood $p(\mathbf{x}_{1:N}|\mathbf{u}_{1:N})$ reduces to learning an action-conditional generative model that can be trained via maximum likelihood, i.e.,

$$\max_{\theta} \ \log p_\theta(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t), \tag{4.8}$$

where the prediction of the next observation $\mathbf{x}_{t+1}$ depends only on the current $\mathbf{x}_t$ and action $\mathbf{u}_t$. Note that our generative model is parameterized by θ. For notational simplicity, we shall omit θ in our presentation.

We first discuss how to learn a low-dimensional representation of $\mathbf{x}$ that adheres to globally linear dynamics by incorporating several constraints into the structure of our generative model. First, we introduce the latent variables $\mathbf{z}_t$ and $\hat{\mathbf{z}}_{t+1}$ that serve as information bottlenecks between $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$, such that

$$p(\mathbf{x}_{t+1}, \mathbf{z}_t, \hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{z}_t | \mathbf{x}_t) p(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_t, \mathbf{u}_t) p(\mathbf{x}_{t+1} | \hat{\mathbf{z}}_{t+1}). \tag{4.9}$$

Intuitively, it is natural to interpret $\mathbf{z}_t$ and $\hat{\mathbf{z}}_{t+1}$ to be stochastic embeddings of $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$, respectively.

Next, we enforce global linearity of $p(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_t, \mathbf{u}_t)$ by restricting it to be a deterministic, linear transition function of the form

$$\hat{\mathbf{z}}_{t+1} = \mathbf{A}\mathbf{z}_t + \mathbf{B}\mathbf{u}_t + \mathbf{c}, \tag{4.10}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{c}$ are matrices that respectively define the state dynamics, control dynamics, and the offset. To emphasize the deterministic nature of this transition, we replace all the subsequent mentions of deterministic $p(\cdot|\cdot)$ transitions with $\delta(\cdot|\cdot)$.

In order to learn more expressive transition dynamics, we relax the global linearity constraint to a local one. Unlike global linearity, local linearity requires a linearization point. To account for this, we introduce an additional variable $\bar{\mathbf{z}}_t$ to serve as the linearization point, which results in a new generative model (see the black arrows in Fig. 4.1),

$$p(\mathbf{x}_{t+1}, \mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{z}_t | \mathbf{x}_t) p(\bar{\mathbf{z}}_t | \mathbf{x}_t) \delta(\hat{\mathbf{z}}_{t+1} | \mathbf{z}_t, \bar{\mathbf{z}}_t, \mathbf{u}_t) p(\mathbf{x}_{t+1} | \hat{\mathbf{z}}_{t+1}), \tag{4.11}$$

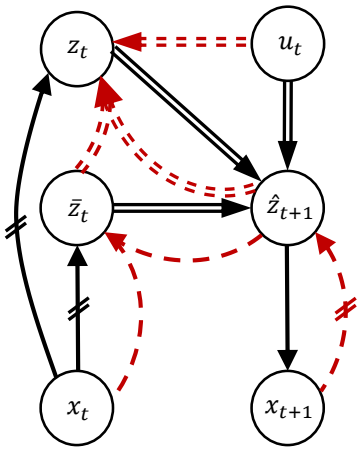


Figure 4.1: RCE graphical model. Black arrows show the generative links and dashed red arrows show the recognition model. Parallel lines mean deterministic links, while single lines mean stochastic links (a link that involves in sampling). $\mathbf{z}_t$ and $\bar{\mathbf{z}}_t$ are two samples from $p(\mathbf{z}|\mathbf{x})$. We use a single network (the encoder network) to model the conditional probability of the links with the hatch marks.

whose corresponding deterministic transition function for $\delta(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \bar{\mathbf{z}}_t, \mathbf{u}_t)$ is

$$\hat{\mathbf{z}}_{t+1} = \mathbf{A}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t)\mathbf{z}_t + \mathbf{B}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t)\mathbf{u}_t + \mathbf{c}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t). \tag{4.12}$$

Here, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{c}$ are functions of $(\bar{\mathbf{z}}_t, \mathbf{u}_t)$, and can be parameterized by neural networks. The offset vector $\mathbf{c}$ plays an important role here. In fact, if a system cannot be modelled using the above locally-linear function, the offset vector will have high variance. Therefore $\mathbf{c}$ can seen as an indicator of the type of systems that can be handled using our model. Since the linearization point $\bar{\mathbf{z}}_t$ is not known in advance, we treat $\bar{\mathbf{z}}_t$ as a random variable with distribution $p(\bar{\mathbf{z}}_t|\mathbf{x}_t)$. A natural consideration for $p(\bar{\mathbf{z}}_t|\mathbf{x}_t)$ is to set it to be identical to $p(\mathbf{z}_t|\mathbf{x}_t)$ *a priori*. This has the effect of making the iLQR controller robust to stochastic sampling of z_t during planning.

4.4.2 Deep Variational Learning

Training the generative model in Eq. 4.11 using maximum likelihood is intractable, since it requires the marginalization of the latent variables. Therefore, we propose to use deep variational inference [49, 79] and maximize the variational lower bound of the log-likelihood, instead. The variational lower bound requires us to define a variational approximation to the true posterior

$$q(\mathbf{z}, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t) \approx p(\mathbf{z}, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t).$$

In adherence to the interpretation of $\mathbf{z}_t$ and $\hat{\mathbf{z}}_{t+1}$ as stochastic embeddings of $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$, it is important to enforce consistency between $p(\hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t)$ and the next step probability of the embedding given the observation $p(\mathbf{z}_{t+1} | \mathbf{x}_{t+1})$. Since we do not have access to $p(\hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t)$, we instead encourage this consistency through posterior regularization by explicitly setting

$$q_\varphi(\hat{\mathbf{z}}_{t+1} | \mathbf{x}_{t+1}) = p(\mathbf{z}_{t+1} | \mathbf{x}_{t+1}). \tag{4.13}$$

Next, we propose a novel factorization of the full variational posterior as

$$q(\mathbf{z}, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1} | \mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t) = q_\varphi(\hat{\mathbf{z}}_{t+1} | \mathbf{x}_{t+1}) q_\varphi(\bar{\mathbf{z}}_t | \mathbf{x}_t, \hat{\mathbf{z}}_{t+1}) \delta(\mathbf{z}_t | \hat{\mathbf{z}}_{t+1}, \bar{\mathbf{z}}_t, \mathbf{u}_t),$$

where $q_\varphi(\bar{\mathbf{z}}_t | \mathbf{x}_t, \hat{\mathbf{z}}_{t+1})$ is the *backward encoder* and $\delta(\mathbf{z}_t | \hat{\mathbf{z}}_{t+1}, \bar{\mathbf{z}}_t, \mathbf{u}_t)$ is the deterministic *reverse transition*. Our choice of factorization results in a recognition model that contrasts sharply with that in E2C. First, our recognition model properly conditions the inference of $\hat{\mathbf{z}}_{t+1}$ on the observation $\mathbf{x}_{t+1}$. Second, our recognition model explicitly accounts for the deterministic transition in the generative model; inference of the deterministic

transition can be performed in closed-form using a deterministic reverse transition that recovers $\mathbf{z}_t$ as a function of $\bar{\mathbf{z}}_t, \mathbf{u}_t$ and $\hat{\mathbf{z}}_{t+1}$. To be consistent with Eq. 4.12, we require that

$$\mathbf{z}_t = \mathbf{A}_t^{-1}(\bar{\mathbf{z}}_t, \mathbf{u}_t)\big(\hat{\mathbf{z}}_{t+1} - \mathbf{B}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t)\mathbf{u}_t - \mathbf{c}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t)\big). \tag{4.14}$$

During the training of the generative model, we only need to access the inverse of $\mathbf{A}_t$. As such, we propose to directly train a network that outputs its inverse $\mathbf{M}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t) = \mathbf{A}_t^{-1}(\bar{\mathbf{z}}_t, \mathbf{u}_t)$. To make sure that $\mathbf{M}_t$ is an invertible matrix and to enable efficient estimation, we restrict $\mathbf{M}_t$ to be a rank-one perturbation of the identity matrix, i.e.,

$$\mathbf{M}_t = \mathbf{I}_{n_z} + \mathbf{w}_t(\bar{\mathbf{z}}_t, \mathbf{u}_t)\mathbf{r}_t^\top(\bar{\mathbf{z}}_t, \mathbf{u}_t), \tag{4.15}$$

where $\mathbf{I}_{n_z}$ is the identity matrix of size n_z, and $\mathbf{w}_t$ and $\mathbf{r}_t$ are two column vectors in $\mathbb{R}^{n_z}$. We constraint these vectors to be non-negative using a non-negative activation at their corresponding output layers.

We now formally the give the RCE loss and its lower bound property.

Lemma 4.4.1. *Let $\mathcal{L}_t^{RCE}$ be defined as*

$$\begin{aligned}\mathcal{L}_t^{RCE} = &\mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})\big] \\ &-\mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[KL\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t)\big)\big] \\ &+H\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big) + \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log p(\mathbf{z}_t|\mathbf{x}_t)\big].\end{aligned} \tag{4.16}$$

Subject to the constraints we explicitly impose on q, $\mathcal{L}_t^{RCE}$ is a lower bound of the conditional log-likelihood $\log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$.

Proof. See Appendix A.1. □

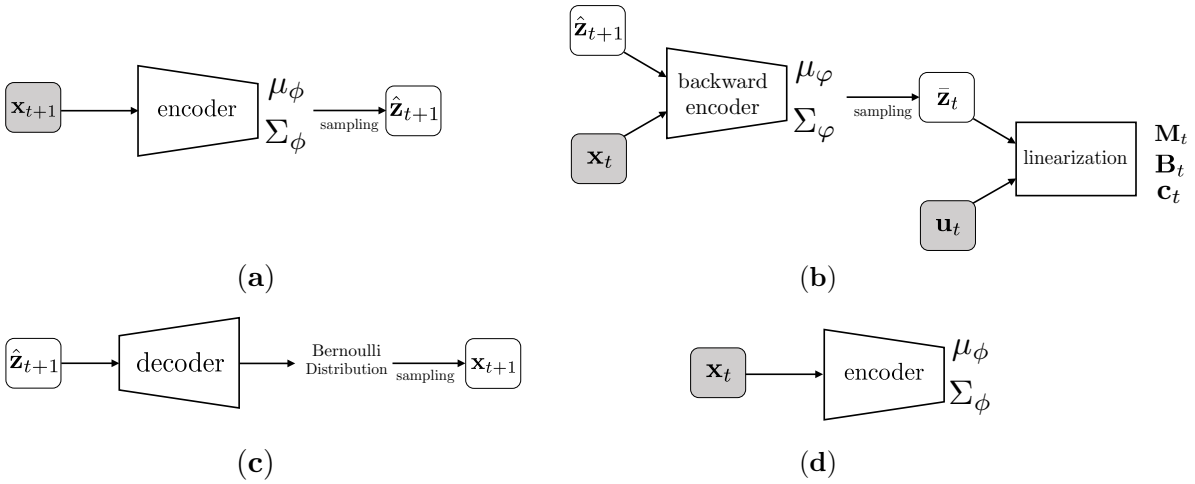


Figure 4.2: Schematic of the networks that are used for modeling the probabilities in our model. The gray boxes contain input (observable) variables. *(a)* Encoder network that models $q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) = \mathcal{N}\big(\mu_\varphi(\mathbf{x}_{t+1}), \mathbf{\Sigma}_\varphi(\mathbf{x}_{t+1})\big)$. *(b)* Transition network that contains two parts. One part, denoted by "backward encoder", models $q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1}) = \mathcal{N}\big(\mu_\varphi(\mathbf{x}_t, \hat{\mathbf{z}}_{t+1}), \mathbf{\Sigma}_\varphi(\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})\big)$, and the other part, denoted by "linearization", is used to obtain $\mathbf{M}_t$, $\mathbf{B}_t$, and $\mathbf{c}_t$, which are the parameters of the locally linear model in the latent space. *(c)* Decoder network that models $p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})$. In our experiments we assume that this distribution is Bernoulli. Therefore, we use sigmoid nonlinearity at the last layer of the decoder. $\bar{\mathbf{x}}_{t+1}$ is the reconstructed version of $\mathbf{x}_{t+1}$. *(d)* The network that models $p(\mathbf{z}_t|\mathbf{x}_t)$. According to Eq. 4.13, since $p(\mathbf{z}_t|\mathbf{x}_t) = q_\varphi(\mathbf{z}_t|\mathbf{x}_t)$ and therefore we tie the parameters of this network with the encoder network, $p(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mu_\varphi(\mathbf{x}_t), \mathbf{\Sigma}_\varphi(\mathbf{x}_t))$. Note that $p(\mathbf{z}_t|\mathbf{x}_t)$ is the same as $p(\bar{\mathbf{z}}_t|\mathbf{x}_t)$. Thus, the KL term in (4.21) can be written as $\mathrm{KL}\big(\mathcal{N}(\mu_\varphi, \mathbf{\Sigma}_\varphi) \parallel \mathcal{N}(\mu_\varphi(\mathbf{x}_t), \mathbf{\Sigma}_\varphi(\mathbf{x}_t))\big)$.

Figure 4.1 contains a graphical representation of our model. We report a complete derivation of Eq. 4.21 in Appendix A.1. It is important to note that unlike the E2C encoder (Eq. 8 in [103]), our recognition model takes the future observation $\mathbf{x}_{t+1}$ as input. In the case of noisy dynamics, the future state heavily influences the posterior. Thus, E2C's failure to incorporate the future state into the variational approximation of the posterior could be detrimental to the performance of the system in the noisy regime. We clearly demonstrate this phenomenon in our experiments.

4.4.3 Network Structure

For the four problems used in our experiments in Section 4.5, we use feedforward networks for encoding, decoding, and transition. Depending on the input image size, the encoder and decoder can have fully-connected layers or convolutional layers. The transition networks always have fully-connected layers. According to Eq. 4.21, we need to model four different conditional probabilities: $p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})$, $q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})$, $q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t)$, and $p(\mathbf{z}_t|\mathbf{x}_t)$. Figure 4.2 shows the high-level depiction of the networks and the connection between different variables used in these probabilities.

4.4.4 Planning

We use the iLQR algorithm to plan in the latent space $\mathcal{Z}$. A good latent space representation should allow us not only to reconstruct and predict the images accurately, but also to plan well in this space using $f_\mathcal{Z}$.

The inputs to the planning algorithm are the two high-dimensional observations $\mathbf{x}^i$ and $\mathbf{x}^f$, corresponding to the initial and final (goal) states $\mathbf{s}^i$ and $\mathbf{s}^f$. We encode these two high-dimensional observations to the latent space observations $\mathbf{z}^i$ and $\mathbf{z}^f$. We sample a random set of H actions $\mathbf{u}_{1:H}$ and apply them to the system, starting from the initial state $\mathbf{s}^i$ (represented in the latent space by $\mathbf{z}^i$). This generates a reference trajectory $\{\bar{\mathbf{z}}_1 = \mathbf{z}^i, \bar{\mathbf{u}}_1 = \mathbf{u}_1, \bar{\mathbf{z}}_2, \bar{\mathbf{u}}_2 = \mathbf{u}_2, \ldots, \bar{\mathbf{z}}_H, \bar{\mathbf{u}}_H = \mathbf{u}_H, \bar{\mathbf{z}}_{H+1}\}$ of size H. We pass this reference trajectory to iLQR and it returns the set of actions $\mathbf{u}^*_{1:H}$ that has been iteratively optimized to minimize a quadratic cost similar to (4.2) in the latent space $\mathcal{Z}$. We apply $\mathbf{u}^*_1$ to the dynamical system, observe the next state's observation $\mathbf{x}_2$, and encode it to the latent space observation $\mathbf{z}_2$. We then generate another reference trajectory by starting from $\mathbf{z}_2$ and applying the sequence of H actions $\{\mathbf{u}^*_2, \ldots, \mathbf{u}^*_H, \mathbf{u}_{H+1}\}$, where $\mathbf{u}_{H+1}$ is a random action. We then run iLQR with this trajectory and apply the first action in the set of H actions it returns to the system. We continue this process for T (the planning horizon) steps.

4.5 Experiments

In this section, we compare the performance of our proposed RCE model with that of E2C in terms of both prediction and planning in the four domains of [103]. To generate our training and test sets, each consists of triples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$, we first sample a state $\mathbf{s}_t$ and generate its corresponding observation $\mathbf{x}_t$. We then take an action $\mathbf{u}_t$ and add a Gaussian noise with covariance $\mathbf{\Sigma}_{\mathbf{n}^s}$ according to Eq. 4.1 to obtain the next state $\mathbf{s}_{t+1}$, which is used to generate the next observation $\mathbf{x}_{t+1}$. We consider both deterministic ($\mathbf{\Sigma}_{\mathbf{n}^s} = \mathbf{0}$) and stochastic scenarios. In the stochastic case, we add noise to the system with different values of $\mathbf{\Sigma}_{\mathbf{n}^s}$ and evaluate the models performance under noise.

In each of the four domains used in our experiments, we compare the performance of RCE and that of E2C in terms of four different factors (see Tables 4.5– 4.4). **1)** *Reconstruction Loss* is the loss in reconstructing $\mathbf{x}_t$ using the encoder and decoder. **2)** *Prediction Loss* is the loss in predicting $\mathbf{x}_{t+1}$, given $\mathbf{x}_t$ and $\mathbf{u}_t$, using the encoder, decoder, and transition network. **3)** *Planning Loss* is computed based on the following quadratic loss:

$$J = \sum_{t=1}^{T} (\mathbf{s}_t - \mathbf{s}^f)^\top \mathbf{Q} (\mathbf{s}_t - \mathbf{s}^f) + \mathbf{u}_t^\top \mathbf{R} \mathbf{u}_t. \tag{4.17}$$

Note that matrices $\mathbf{Q}$ and $\mathbf{R}$ are in the form of $\alpha\mathbf{I}$ ($\mathbf{I}$ is the identity matrix). Therefore the planning loss function is convex and its lower bound is zero. We apply the sequence of actions returned by iLQR to the dynamical system and report the value of the loss in Eq. 4.17. **4)** *Success Rate* shows the number of times the agents reaches the goal within the planning horizon T, and remains near the goal in case it reaches it in less than T steps. For each of the domains, all the results are averaged over 20 runs. The details of our implementations, including the network's structure, the size of the latent space, and the planning horizon are specified in Appendix A.2. We also include the number of data points used for training our model in each of the experiments. During training, we observed that increasing the size of training data results in a better performance of

our model in terms of planning loss. This is an expected phenomenon in model-based algorithms. However, we did not perform experiments on the effect of size of training data on the performance of our model.

4.5.1 Planar System

Consider an agent in a surrounded area, whose goal is to navigate from a corner to the opposite one, while avoiding the six obstacles in this area. The system is observed through a set of 40×40 pixel images taken from the top, which specify the agent's location in the area. Actions are two-dimensional and specify the direction of the agent's movement.

Table 4.5 shows that RCE outperforms E2C in both prediction/reconstruction and planning in this domain. The Gaussian noise we add to the system has a diagonal covariance matrix with equal variance in all dimensions. The values mentioned in the table for $\mathbf{\Sigma_n}s$ are the standard deviation in each dimension.

Figure 4.3 shows the latent space representation of data points in the planar system dataset for both RCE and E2C models. RCE has clearly a more robust representation against the noise and is able to predict the defined trajectory with a much higher quality.

$\mathbf{\Sigma_n}s$	Algorithm	Reconstruction Loss	Prediction Loss	Planning Loss	Success Rate
0	**RCE**	3.6 ± 1.7	6.2 ± 2.8	21.4 ± 2.9	100 %
	E2C	7.4 ± 1.7	9.3 ± 2.8	26.3 ± 4.9	100 %
1	**RCE**	8.3 ± 5.5	10.1 ± 6.2	25.4 ± 3.6	100 %
	E2C	19.2 ± 5.1	28.3 ± 10.2	34.1 ± 9.5	95 %
2	**RCE**	12.3 ± 4.9	17.3 ± 6.2	36.4 ± 10.3	95 %
	E2C	37.1 ± 10.5	45.8 ± 13.1	63.7 ± 16.3	75 %
5	**RCE**	25.2 ± 6.1	27.3 ± 8.2	50.3 ± 14.5	85 %
	E2C	60.3 ± 18.2	78.3 ± 15.0	112.4 ± 30.2	45 %

Table 4.1: RCE and E2C Comparison – Planar System

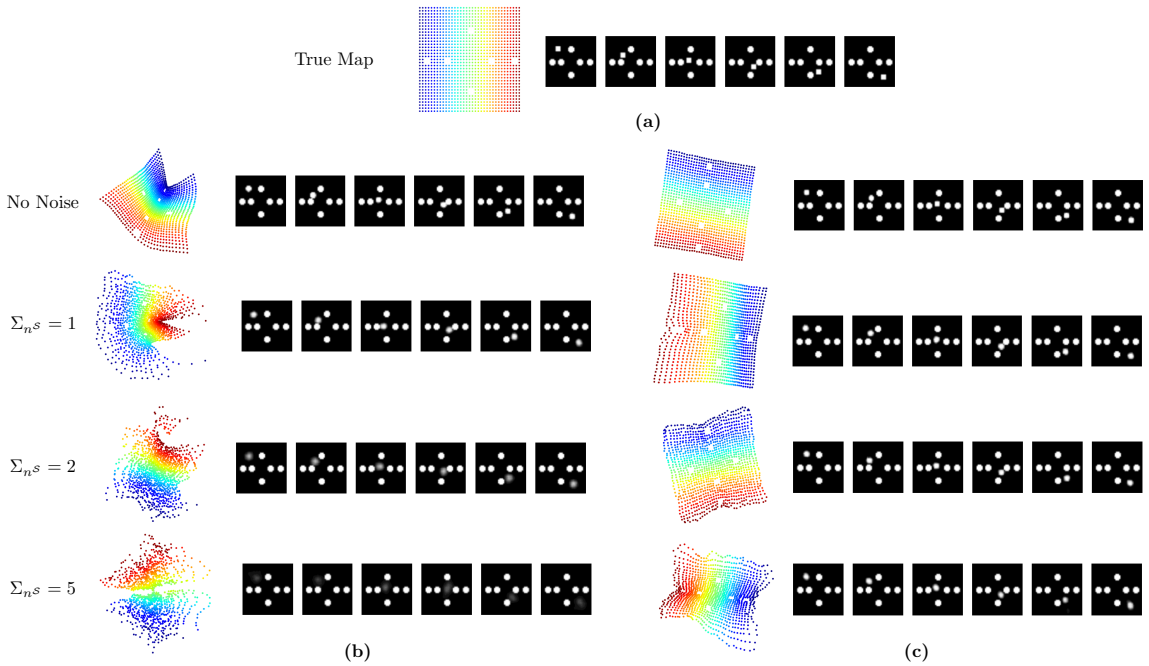


Figure 4.3: *(a) Left:* The true state space of the planar system. Each point on the map corresponds to one image in the dataset. *(a) Right:* A random trajectory. Each image is 40×40 black and white. The circles show the obstacles and the square is the agent in the domain. *(b)* Reconstructed map and predicted trajectory in the latent space of the E2C model for different noise levels. *(c)* Reconstructed map and predicted trajectory in the latent space of the RCE model for different noise levels.

4.5.2 Inverted Pendulum (Acrobat)

This is the classic problem of controlling an inverted pendulum [101] from 48×48 pixel images. The goal in this task is to swing up and balance an underactuated pendulum from a resting position (pendulum hanging down). The true state space of the system $\mathcal{S}$ has two dimensions: angle and angular velocity. To keep the Markovian property in the observation space, we need to have two images in each observation $\mathbf{x}_t$, since each image shows only position of the pendulum and does not have any information about its velocity.

Table 4.2 contains our results of comparing RCE and E2C models in this task. Learn-

ing the dynamics in this problem is harder than reconstructing the images. Therefore, at the beginning of the training we set the weights of the two middle terms in Eq. 4.21 to 10, and eventually decrease them to 1. The results show that RCE outperforms than E2C, and the difference is significant under noisy conditions.

$\Sigma_n s$	Algorithm	Reconstruction Loss	Prediction Loss	Planning Loss	Success Rate
0	**RCE**	43.1 ± 13.2	48.1 ± 17.2	14.2 ± 4.6	95 %
	E2C	73.2 ± 20.1	79.6 ± 32.6	16.1 ± 2.9	90 %
1	**RCE**	61.1 ± 16.2	70.2 ± 36.1	17.3 ± 7.1	85 %
	E2C	97.1 ± 34.1	109.7 ± 58.2	29.9 ± 9.2	60 %
2	**RCE**	92.11 ± 35.4	106.4 ± 53.2	27.5 ± 6.6	70 %
	E2C	140.2 ± 47.1	179.5 ± 61.1	40.7 ± 11.8	40 %

Table 4.2: RCE and E2C Comparison – Inverted Pendulum (Acrobat)

4.5.3 Cart-pole Balancing

This is the visual version of the classic task of controlling a cart-pole system [91]. The goal in this task is to balance a pole on a moving cart, while the cart avoids hitting the left and right boundaries. The control (action) is 1-dimensional and is the force applied to the cart. The original state of the system $\mathbf{s}_t$ is 4-dimensional. The observation $\mathbf{x}_t$ is a history of two 80×80 pixel images (to maintain the Markovian property). Due to the relatively large size of the images, we use convolutional layers in encoder and decoder. To make a fair comparison with E2C, we also set the dimension of the latent space $\mathcal{Z}$ to 8.

Table 4.3 contains our results of comparing RCE and E2C models in this task. We again observe a similar trend: RCE outperforms E2C in both noiseless and noisy settings and is significantly more robust.

4.5.4 Three-link Robot Arm

The goal in this task is to move a three-link planar robot arm from an initial position to a final position (both chosen randomly). The real state of the system $\mathcal{S}$ is 6-dimensional

Σ_{n^s}	Algorithm	Reconstruction Loss	Prediction Loss	Planning Loss	Success Rate
0	**RCE**	33.2 ± 15.6	42.1 ± 26.9	21.2 ± 6.3	90 %
	E2C	44.9 ± 17.0	57.3 ± 22.9	25.3 ± 4.8	85 %
1	**RCE**	52.1 ± 20.3	63.3 ± 27.2	28.4 ± 5.5	80 %
	E2C	70.2 ± 23.7	90.5 ± 42.4	39.8 ± 5.2	70 %
2	**RCE**	77.6 ± 30.2	88.4 ± 38.3	42.2 ± 8.3	70 %
	E2C	112.6 ± 39.2	133.0 ± 56.5	67.2 ± 9.3	40 %

Table 4.3: RCE and E2C Comparison – Cart-pole Balancing

and the actions are 3-dimensional, representing the force applied to each joint of the arm. We use two 128×128 pixel images of the arm as observation $\mathbf{x}$. To be consistent with the E2C model, we choose the latent space $\mathcal{Z}$ to be 8-dimensional.

Table 4.4 contains our results of comparing RCE and E2C models in this task. Similar to the other domains, our results show that the RCE model is more robust to noise than E2C.

Σ_{n^s}	Algorithm	Reconstruction Loss	Prediction Loss	Planning Loss	Success Rate
0	**RCE**	60.5 ± 27.1	69.9 ± 32.2	81.3 ± 35.5	90 %
	E2C	71.3 ± 19.5	83.4 ± 28.6	90.23 ± 47.38	90%
1	**RCE**	96.5 ± 34.4	112.6 ± 42.2	106.2 ± 50.8	80 %
	E2C	138.1 ± 42.5	172.2 ± 58.3	155.2 ± 70.1	65 %

Table 4.4: RCE and E2C Comparison – Robot Arm

4.6 Disentangling Dynamics and Content

Learning disentangled features has various applications in image and video processing and text analysis and has been studied in different works [62, 98]. Inspired by our observation about the effectiveness of using VAEs for learning disentangled representations in Chapter 3 we tackle the following planning problem using the framework of RCE [11, 9].

4.6.1 Problem Statement

Suppose we have different sets of high-dimensional observations from the states of dynamical systems where the underlying dynamics of the systems is the same. For now, let us assume that we only have one dynamical system and there are just two observation sets from this system from different angles. We make this assumption just for the sake of simplicity in notations, but it can be easily relaxed. The two observation sets are denoted by X and Y that belong to the observation spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively. We assume both observation sets are full depictions of all variables in the actual state space of the system.

Suppose set X consists of triples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$, i.e. observation of the system at time t, action that is applied to the system at time t, and the next observation after applying $\mathbf{u}_t$ to the system, respectively. Therefore, we know how the actions change our observations in X. We also assume that the observations in this set have Markov property. Set Y also has some observations of the system from a different point of view. However, there is no information about the actions and the effect of the actions on our observation in this set. We denote the observations in this set by $\mathbf{y}_t$. Note that $\mathbf{x}_t$ and $\mathbf{y}_t$ are two different observations of the state $\mathbf{s}_t$. Since X and Y, are observations from one system, the underlying dynamics is the same. Suppose that our goal is to do planning and long-term prediction in $\mathcal{Y}$. Our approach to achieve this goal is to extract the dynamics information from X and leverage this information to build a model for Y.

4.6.2 Model description

The RCE model is based on introducing a graphical model for the problem that describes the relation between pairs of observations and their embedded representations. Using deep variational learning, the lower bound of the conditional distribution of the observations is maximized.

We build our model up on RCE . However, instead of using only one latent variable, we assume that there are two independent variables in the latent space. One of these

variables is related to the dynamics of the system and the other one is related to the content of the observation. Therefore we aim to disentangle the dynamics and content in the latent space. We follow the same framework of JADE introduced in Chapter 3. Such disentanglement allows us to model the dynamics of the observations, even though the content of them might be very different. Consider the graphical models in Fig. 5.4. Fig. 4.4a shows the model for X. In this figure, $\mathbf{z}_t$ and $\mathbf{w}_\mathbf{x}$ are the two latent variables that we want to represent the dynamics and content information, respectively. Similar to RCE, we want to have locally-linear dynamics in the latent space, i.e.:

$$\hat{\mathbf{z}}_{t+1} = \mathbf{A}_t \mathbf{z}_t + \mathbf{B}_t \mathbf{u}_t + \mathbf{c}_t \tag{4.18}$$

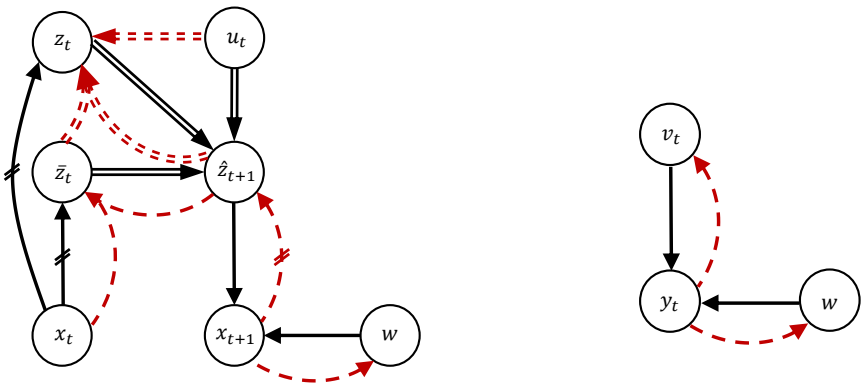
where $\mathbf{A}_t$, $\mathbf{B}_t$, and $\mathbf{c}_t$ are matrices that are learned during training the model. Building this locally-linear model will allow us to use iLQR method for control. We use $\mathbf{z}$ and $\hat{\mathbf{z}}$ to distinguish between encoding of $\mathbf{x}$ and the variable after transition. Fig. 4.4b shows the model for Y. This set is encoded with two latent variables $\mathbf{v}_t$ and $\mathbf{w}_\mathbf{y}$, representing dynamics and content, respectively. We would like to have a locally-linear dynamics similar to Eq. 4.18 for $\mathbf{v}$. All of the conditional distribution on these graphical models are parameterized by neural networks.

The goal in this work can be interpreted as maximizing the likelihood of observations, while imposing a further constraint that if $\mathbf{x}_t$ and $\mathbf{y}_t$ are two high-dimensional observations of the same state of the dynamical system(s), then we want $q(\mathbf{z}_t|\mathbf{x}_t)$ and $q(\mathbf{v}_t|\mathbf{y}_t)$ be close to each other, e.g. have small KL divergence.

Suppose $q^\star = q(\mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}, \mathbf{w}_\mathbf{x}|\mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t)$ and $q^\dagger = q(\mathbf{v}_t, \mathbf{w}_\mathbf{y}|\mathbf{y}_t)$. Based on the graphical model we can consider these factorizations for $q^\star$ and $q^\dagger$:

$$\begin{aligned} q^\star &= q_\varphi(\mathbf{w}_\mathbf{x}|\mathbf{x}_{t+1}) q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \delta(\mathbf{z}_t|\hat{\mathbf{z}}_{t+1}, \bar{\mathbf{z}}_t, \mathbf{u}_t) \\ q^\dagger &= q_\varphi(\mathbf{w}_\mathbf{y}|\mathbf{y}_t) q_\varphi(\mathbf{v}|\mathbf{y}_t) \end{aligned} \tag{4.19}$$

where φ and φ stand for encoder and transition network parameters, respectively. We



(a) (b)

Figure 4.4: Graphical models. The black arrows are generative links and the red dashed ones are recognition links. The parallel lines show the deterministic links. **(a)** Graphical model for set X. $\bar{\mathbf{z}}_t$ and $\mathbf{z}_t$ are two samples from $p(\mathbf{z}_t|\mathbf{x}_t)$. The neural networks that parameterize the links with hatch marks are hard tied, i.e. $p(\mathbf{z}_t|\mathbf{x}_t) = p(\bar{\mathbf{z}}_t|\mathbf{x}_t) = q(\hat{\mathbf{z}}_t|\mathbf{x}_t)$. **(b)** Graphical model for Y

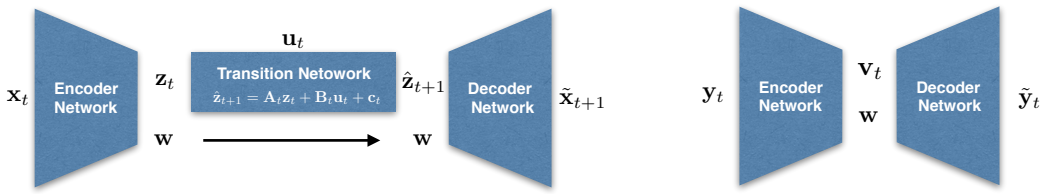
also have the following factorization for the generative links in the graphical model:

$$p(\mathbf{x}_{t+1}, \mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}, \mathbf{w}_\mathbf{x}|\mathbf{x}_t, \mathbf{u}_t) = p(\bar{\mathbf{z}}_t|\mathbf{x}_t)p(\mathbf{z}_t|\mathbf{x}_t)\delta(\hat{\mathbf{z}}_{t+1}|\bar{\mathbf{z}}_t, \mathbf{z}_t, \mathbf{u}_t)p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}, \mathbf{w}_\mathbf{x})p(\mathbf{w}_\mathbf{x}) \tag{4.20}$$

Lemma 4.6.1. *Let $\mathcal{L}_t^{DDC}$ be defined as*

$$\begin{aligned}\mathcal{L}_t^{DDC_R CE} = \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\mathbf{w}|\mathbf{x}_{t+1})}}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}, \mathbf{w})\big] - \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[KL\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t)\big)\big] \\ + H\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big) + \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log p(\mathbf{z}_t|\mathbf{x}_t)\big] - KL\big(q_\varphi(\mathbf{w}|\mathbf{x}_t) \parallel p(\mathbf{w})\big) \\ + \mathbb{E}_{q^\dagger}\big[\log p(\mathbf{y_t}|\mathbf{v}_t, \mathbf{w})\big] - KL\big(q_\varphi(\mathbf{v}_t|\mathbf{y}_t) \parallel p(\mathbf{v}_t)\big) - KL\big(q_\varphi(\mathbf{w}|\mathbf{y}_t) \parallel p(\mathbf{w})\big)\end{aligned} \tag{4.21}$$

Subject to the constraints we explicitly impose on q, $\mathcal{L}_t^{DDC}$ is a lower bound of the conditional



(a) (b)

Figure 4.5: Networks of the model

log-likelihood $\log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) + \log p(\mathbf{y}_t)$.

Proof. See Appendix A.1. □

As we mentioned above, for the observation from the same state we want to minimize the KL divergence between $q(\mathbf{z}_t|\mathbf{x}_t)$ and $q(\mathbf{v}_t|\mathbf{y}_t)$. We can easily impose this constraint by considering $q(\mathbf{z}_t|\mathbf{x}_t)$ as the prior for $p(\mathbf{v}_t)$, i.e.:

$$p(\mathbf{v}_t) = q(\mathbf{z}_t|\mathbf{x}_t) \tag{4.22}$$

Fig. 4.5 shows the high-level depiction of the networks in our model.

4.6.3 Experiment Result

To evaluate the effectiveness of the proposed model, we consider the planar system domain. Consider an agent in a surrounded area, whose goal is to navigate from a corner to the opposite one, while avoiding the six obstacles in this area. The system is observed through a set of 40×40 pixel images taken from the top, which specify the agent's location in the area. Actions are two-dimensional and specify the direction of the agent's movement. Suppose that the difference between the two observation sets from this system is in the shape of the agent, as shown in Fig. 4.6. We use the same encoder and decoder for the two observation sets. We used 8000 samples (triples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$) in the set X and only 2000 samples in set Y.

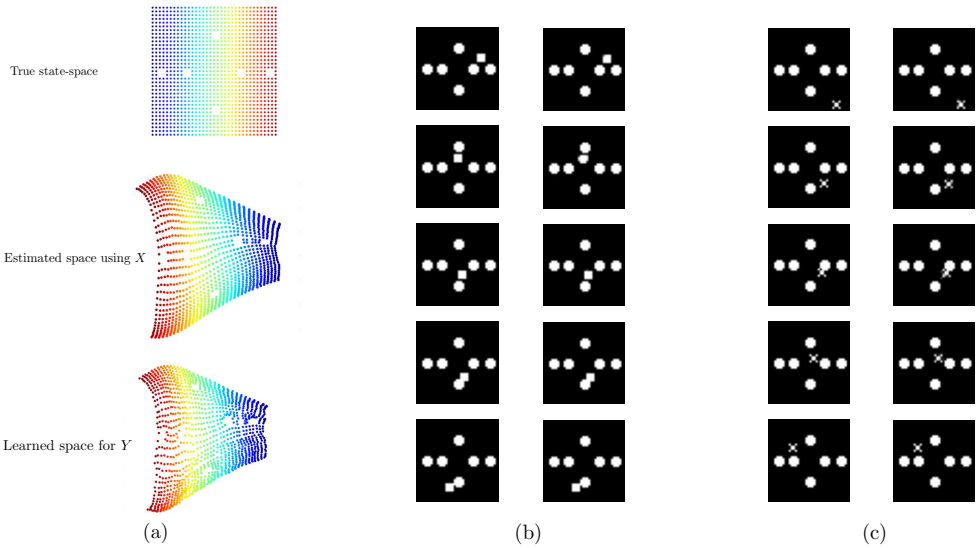


Figure 4.6: **(a)** Top: The true state space of the system. Middle: estimated locally-linear latent space from set X. Bottom: The hidden space learned for set Y. **(b)**: Left: An initial observation from X on top and its next observations after applying four random actions Right: Reconstruction of the initial state and prediction of the next observations. **(c)**: Left: An initial observation from Y on top and its next observations after applying four random actions Right: Reconstruction of the initial state and prediction of the next observations

Dataset	Reconstruction Loss	Prediction Loss	Planning Loss	Success Rate
with action (X)	3.6 ± 1.7	6.2 ± 2.8	21.4 ± 2.9	100 %
without action (Y)	3.9 ± 2.2	6.3 ± 3.0	22.0 ± 2.4	100 %

Table 4.5: Planar System

Fig. 4.6 shows the true map of the state-space of this system and the maps that are estimated using the model for the two observation sets. As we can see, the map that has been discovered using the information in X is very well preserved for the set Y. In this figure we can also see some predictions of the position of the agent for both sets given some actions versus the true position of the agent after applying those action. This shows that the model is successful in learning the dynamics for Y even though we did not have any information about the dynamics in this set.

To evaluate the performance of the model in planning, we provide different sets of initial and final observations in $\mathcal{X}$ and $\mathcal{Y}$, and use the learned models to find the policy that leads the agent to reach the final observation within T steps. We present the performance of the model in table 4.5.

4.7 Summary

We proposed a new method to embed the high-dimensional observations of an MDP in such a way that both the embeddings and locally optimal controllers are robust w.r.t. the noise in the system's dynamics. Our RCE model enjoys a clean separation between the generative graphical model and its recognition model. The RCE's generative model explicitly treats the unknown linearization points as random variables, while the recognition model is factorized in reverse direction to take into account the future observation as well as exploiting determinism in the transition dynamics. Our experimental results demonstrate that the RCE's predictive and planning performance are better and significantly more robust than that of E2C in all the four benchmarks where E2C performance has been measured [103].

Chapter 5

A Multi-step Action-based Prediction Method for Autonomous Driving

5.1 Contributions

In autonomous driving, being able to understand, model and predict the evolution of the surrounding environment is one of the most important and challenging tasks. This is due to the complexity of the driving environments, dynamic models of different types of objects and more importantly the complexity of the interaction between the objects. We propose *Prediction by Anticipation*, a method for action-conditional environment prediction for self-driving cars where the environment is represented in the form of Occupancy Grid Map (OGM). Our motivation is that accurate modelling and prediction of the driving environment can efficiently improve path planning and navigation resulting in safe, comfortable and optimum paths in autonomous driving.

Due to the importance of interactions between the objects in the scene, it is important to model and predict the driving environment based on the ego-actions to be able to predict the effect of our actions on other agents decisions and behaviours. We train our

model in the framework of conditional varitional autoencoders (CVAEs) to maximize the evidence lower bound (ELBO) of the log-likelihood of a conditional observation distribution. An extension of our model is also presented that explicitly learns the difference between consecutive frames and is suitable for dense urban traffic. We evaluate our model on OGM sequences from **NGSIM** and **Argoverse** dataset. The results show significant improvements of the prediction accuracy using our proposed architectures over the state-of-the-art.

5.2 Problem statement

In this chapter, we are interested in solving the prediction task for multi-agent systems with interacting agents. We propose a new probabilistic approach for action-conditional prediction. The action-conditional prediction task is defined as finding the next observation from the scene given a sequence of observations, $\mathbf{x}_{1:t}$, and an action for an agent (ego agent), $\mathbf{a}_t$, i.e. optimizing $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t)$. Our approach is based on the idea that sequences of data that include heavy interactions come from a probabilistic generative process wherein some reacting agents move partly in response to the *anticipated* motion of some acting agents. In fact we split the observation into two sets of features: ego-features, $\mathbf{x}_t^{ego}$, which contains the features related to the ego agent, e.g. it's position, and environment-features, $\mathbf{x}_t^{env}$, which contains all other features than the ego-features including other agents or fixed objects in the scene. The action has an effect on both of these feature sets. In most of the previous works $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t)$ is learned directly. This means predicting ego- and environment-features simultaneously, which is difficult due to interactions among the agents. However, the effect on the ego-features is usually much easier to model and often does not need to be learned. Hence, we can decompose $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t)$ into two steps: learning $p(\mathbf{x}_{t+1}^{ego}|\mathbf{x}_{1:t}, \mathbf{a}_t)$ and then learning $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$, i.e. we first learn how the action changes the ego-features and then learn how the environment reacts to this change. We can often use domain knowledge to fix $p(\mathbf{x}_{t+1}^{ego}|\mathbf{x}_{1:t}, \mathbf{a}_t)$ and then we can learn $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$ from data. Learning $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$ is much easier than learning $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t)$ since we only have to learn to predict $\mathbf{x}_{t+1}^{env}$ based on

the *effect* of action $\mathbf{a}_t$ on the ego-features $\mathbf{x}_{t+1}^{ego}$. Conditioning on $\mathbf{a}_t$ or $\mathbf{x}_{t+1}^{ego}$ is equivalent from an information theoretic perspective, but conditioning on $\mathbf{x}_{t+1}^{ego}$ allows the model to reason about interactions more easily since the ego effect is already anticipated. Concretely, at each step of training, we apply the motion of an acting agent to the input observation, i.e. *anticipating* the acting agent in its next position, and train a conditional density estimation model [86] to predict where the other agents accordingly should be in the target observation, i.e. where they should move to *partly in reaction to* the anticipated motion of the acting agent.

In particular we consider the prediction task in the challenging framework of autonomous driving (AD), where there is a large number of agents in the scene and they have complicated interactions with each other. By thus recovering the latent generative process, our model is capable of achieving a higher capacity for prediction, i.e. handling a wider range of actions and driving situations.

To minimize preprocessing of data that can be costly and time-consuming at training and inference time, we adopt input and target representations in the form of occupancy grid maps (OGMs). The anticipating can be done through editing the original input OGM based on motion tracking of the acting vehicle. Working with OGMs also enables us to easily extend our model to predict the *difference* between the input and target frames, rather than predict the target frame directly. We show that such difference learning works very well when the ego vehicle moves slowly, such as in dense urban traffic. Our contributions in this chapter include:

- A novel modular model for multi-step action-conditional prediction is proposed, which factorizes historical sequence data into samples drawn according to an underlying action-conditional distribution that covers a wider range of actions (including extreme actions) for predictions better than current state-of-the-art models. The method requires no labeling and is scalable with data.
- An extension of the model for difference learning that outperforms the state-of-the-art prediction models in dense traffic.
- Experiment results on two prominent AD datasets (NGSIM I-80 and Argoverse)

with different interactions among vehicles, i.e. highway and urban area, demonstrating effective coverage of a wide variety of driving situations.

5.3 Related Work

A large body of literature on prediction tasks in AD is dedicated to prediction in low-dimensional space, i.e. position of the cars in the xy coordinates [56, 27, 60, 17, 24, 12, 106, 83, 42, 74, 80, 81, 18, 107, 59]. In most of these works the prediction task is done by finding the most probable paths for the objects in the environment using generative models. However, all of these methods need object detection and tracking (at least at training time), which is computationally expensive and requires labeled data. Moreover, any error in the object detection and tracking can affect the whole system and result in catastrophic failure.

Unsupervised prediction of OGMs [29, 97] has also been studied recently. These methods do not model the effect of action in prediction and thus fail to capture the interactions. In [63] and its extension [28] recurrent neural network (RNN)-based models were employed for OGM prediction. But, both models need data labeling and object detection. Authors in [65] proposed a model for multi-step prediction of OGMs that produces state-of-the-art results on the KITTI dataset[31]. By *removing* the ego-motion from the whole sequence, all the frames are mapped to a reference frame in which the ego-vehicle is frozen, i.e. the global location of the car is fixed. This way, only moving objects in the scene change their locations. There are two major differences between this work and ours. First of all, their predictions are not action-conditional. Secondly, since we *compensate* the ego-motion step-by-step the global location of the ego car is not fixed. Therefore, in contrast to [65], our model can predict for much longer horizons.

In [70, 71, 36] OGM prediction is used for path planning. However, in [70, 71] only one object type (human) is considered. [36] relies on object detection and the OGMs are updated using object models. Model-predictive policy with uncertainty regularization (MPUR) [39], is a state-of-the-art prediction and planning approach in this area.

Although the model is successful in predicting the effect of existing actions in the training data, in the case of extreme actions it fails to predict a valid OGM. Alternatively, in [12] synthetic extreme actions are added to the training data in order to handle rare scenarios. However, since this is not a theoretically principled way for generalization, the performance of the algorithm is still limited by actions directly observed in the data. Moreover, addition of random actions not grounded in real interaction contexts can result in invalid scenarios that do not happen in real life, as the other cars do not react to the augmented actions. Finally, the method is an object tracking method with the aforementioned problems.

5.4 Prediction by Anticipation

The prediction task is described as follows. Given a set of t observations from the scene, denoted by $\mathbf{x}_{1:t}$, and a set of k actions denoted by $\mathbf{a}_{t:t+k-1}$, predict the future k observations, $\mathbf{x}_{t+1:t+k}$. We try to solve this task by maximizing the conditional likelihood $p(\mathbf{x}_{t+1:t+k}|\mathbf{x}_{1:t}, \mathbf{a}_{t:t+k-1})$. The observations include (a) A bird's-eye view (BEV) image, denoted by $\mathbf{i}_t$ for time step t, in the form of an OGM with fixed position, e.g. in the middle of the image, for the ego-vehicle. (b) Position and velocity of the ego-vehicle in each direction, which are denoted by $\mathbf{p}_t$ and $\mathbf{v}_t$, respectively, and are referred to as *measurements*. These two parts of the observation are both sensory data from the vehicle. However, we focus on predicting the images, as the position and velocity can be deterministically computed given the actions, as described in the next sections.

5.4.1 Base model

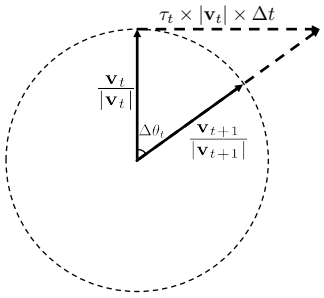
Let's assume $\mathbf{x}_t = \{\mathbf{x}_t^{ego}, \mathbf{x}_t^{env}\}$, where $\mathbf{x}_t^{ego}$ includes the features related to the ego-agent (ego-vehicle), i.e. $\mathbf{p}_t$, $\mathbf{v}_t$, and parts of the OGM related to the ego-vehicle, denoted by $\mathbf{i}_t^{ego}$. Therefore $\mathbf{x}_t^{ego} = \{\mathbf{p}_t, \mathbf{v_t}, \mathbf{i}_t^{ego}\}$. The environment-features, $\mathbf{x}_t^{env}$, include all other features in the observation than the ego-features. In our application this includes parts of the

OGM that represent other objects in the scene, e.g. other agents, maps, static objects, etc. Therefore, $\mathbf{x}_t^{env} = \{\mathbf{i}_t^{env}\}$. The actions of the ego-vehicle (ego-actions) affect both ego- and environment-features. The first-order effect of the action is on the ego-features, which can be determined using our *prior knowledge* about the dynamics of the ego-vehicle. The second-order effect is on the environment-feature that should be learned from data. Thus, to maximize the conditional log-likehihood of $\log p(\mathbf{x}_{t+1:t+k}|\mathbf{x}_{1:t}, \mathbf{a}_{t:t+k-1})$, we first split it into k autoregressive steps where at each time step the previous prediction is fed-back to the model. Then for each time step we factorize the log-likelihood into two terms:

$$\log p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t) = \log p(\mathbf{x}_{t+1}^{ego}|\mathbf{x}_t, \mathbf{a}_t) + \log p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego}). \tag{5.1}$$

Note that to determine the next ego-features, we only need the current observation and action. For the second term, we assume $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego}, \mathbf{a}_t) = p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$. This assumption is based on the fact that the environment does not observe $\mathbf{a}_t$ directly. In other words, the other agents only observe the *effect* of the ego-action and not the actual action itself. Also, it is worth mentioning that learning $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$ does not mean that we assume causality between $\mathbf{x}_{t+1}^{env}$ and $\mathbf{x}_{t+1}^{ego}$. We simply learn a distribution where there is a non-causal correlation between $\mathbf{x}_{t+1}^{env}$ and $\mathbf{x}_{t+1}^{ego}$. The idea is that other agents anticipate $\mathbf{x}_{t+1}^{ego}$ and act accordingly. The conditional distribution $p(\mathbf{x}_{t+1}^{env}|\mathbf{x}_{1:t}, \mathbf{x}_{t+1}^{ego})$ models the noise in this anticipation. But since there is a strong correlation between $\mathbf{x}_{t+1}^{env}$ and $\mathbf{x}_{t+1}^{ego}$, then conditioning on $\mathbf{x}_{t+1}^{ego}$ will be very informative.

To implement this idea, we design a modular model that consists of rule-based and learning-based modules. The rule-based modules implement the deterministic parts of Eq. 5.1, which are based on our prior knowledge and geometry of the problem. The learning-based module, which we call the prediction module, performs the prediction task by learning the interactions among the agents.



$$\Delta\mathbf{p}_t = \mathbf{v}_t \Delta t$$
$$\mathbf{p}_{t+1} = \mathbf{p}_t + \Delta\mathbf{p}_t$$
$$|\mathbf{v}_{t+1}| = \alpha_t \Delta t + |\mathbf{v}_t|$$
$$\Delta\theta_t = \arctan(\tau_t |\mathbf{v}_t| \Delta t)$$

Figure 5.1: Computing the effect of actions on the future position, velocity and change in the direction of the car.

Rule-based modules

These modules are responsible for factorization according to the presumed underlying generative process. Both parts of the observations are changed to account for the effect of the action. This is done using deterministic functions for measurements and deterministic transformations (rotation and translation) for the OGMs.

Measurements estimator module: Given the actions and measurements at each time step, this module computes the next measurements according to the update rule shown in Fig. 5.1. Actions are two-dimensional, which include acceleration, α, and rotation of the steering wheel, τ, $\mathbf{a}_t = [\alpha_t, \tau_t]$. This module also provides elements of translation and rotation matrices for the image processing modules:

$$\mathbf{p}_{t+1}, \mathbf{v}_{t+1}, \Delta\mathbf{p}_t, \Delta\theta_t = f_m(\mathbf{p}_t, \mathbf{v}_t, \mathbf{a}_t). \tag{5.2}$$

Input OGM transformation modules (IOTs): In order to apply the first-order effect of the action in the OGM, we change the position of the ego-vehicle in the current OGM, $\mathbf{i}_t$. This transformation takes the ego-vehicle to its *anticipated* position at time $t+1$ based on the action. We denote the module that performs this transformation by IOT1, and

the output is denoted by $\mathbf{j}_{t+1}^{ego}$:

$$\mathbf{j}_{t+1}^{ego} = \text{IOT1}(\mathbf{i}_t, \Delta\mathbf{p}_t, \Delta\theta_t). \tag{5.3}$$

Moreover, at each time step of training we preprocess the target OGM, $\mathbf{i}_{t+1}$, to account for the second-order effect of the action. That is, we transform the whole target OGM in a way that the ego-vehicle has the same position as in $\mathbf{j}_{t+1}^{ego}$. We denote the module for this transformation and its output by IOT2 and $\mathbf{j}_{t+1}^{env}$, accordingly.

$$\mathbf{j}_{t+1}^{env} = \text{IOT2}(\mathbf{i}_{t+1}, \Delta\mathbf{p}_t, \Delta\theta_t). \tag{5.4}$$

Fig. 5.2 shows the output of these two modules for a sequence of current and target OGMs. Note that after these two transformations, only the position of the moving objects will be different in $\mathbf{j}_{t+1}^{ego}$ and $\mathbf{j}_{t+1}^{env}$, while map information and structure of the fixed objects in the OGM, e.g. buildings, tree, parked cars, etc., will be the same. Both $\mathbf{j}_{t+1}^{ego}$ and $\mathbf{j}_{t+1}^{env}$ are fed to the prediction module during training.

We should clarify here that $\mathbf{j}_{t+1}^{ego}$ and $\mathbf{j}_{t+1}^{env}$ notations are used to emphasize the change after applying the motion of the ego-vehicle. More generally, we will use $\mathbf{j}$ to denote OGMs resulting from deterministic transformations and $\mathbf{i}$ to denote the ego-centred OGMs. In fact, our original goal of optimizing for the stochastic mapping $p(\mathbf{i}_{t+1}^{env}|\mathbf{o}_{\mathbf{1:t}}, \mathbf{i}_{t+1}^{ego})$

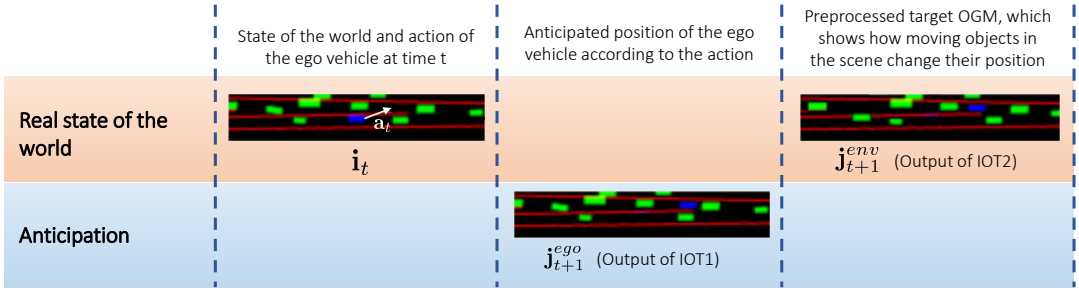


Figure 5.2: Applying the effect of action on the OGMs: Left: OGM $\mathbf{i}_t$ and corresponding action at time t. Middle: the output of IOT1, $\mathbf{j}_{t+1}^{ego}$, after applying transformation on the ego-features of $\mathbf{i}_t$, . Right: the output of IOT2, $\mathbf{j}_{t+1}^{env}$, after applying transformation on $\mathbf{i}_{t+1}$.

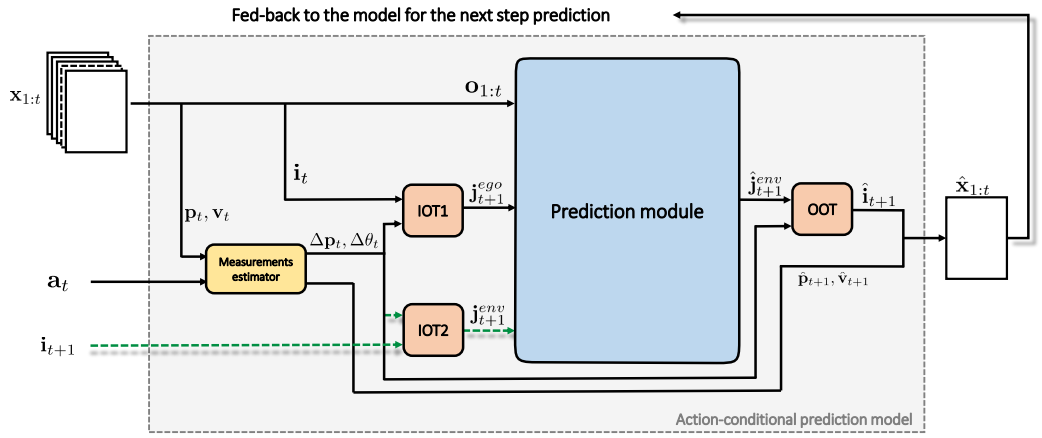


Figure 5.3: The prediction model with all of its components. The measurement estimation module updates the position and velocity of the ego-vehicle and provides transformation parameters for OGM transformer modules. IOT1 and IOT2 provide information about the first and second-order effect of action on the OGM, respectively. Prediction module is trained to minimize the cost in Eq. 5.6.

is equivalent to optimizing for $p(\mathbf{j}^{env}_{t+1}|\mathbf{o_{1:t}}, \mathbf{j}^{ego}_{t+1})$, up to some deterministic transformations.

Output OGM transformation module (OOT): This module takes the predicted frame and transforms the whole frame using $\Delta\theta_t$, and $\Delta\mathbf{p}_t$ such that the ego-vehicle goes back to its fixed position in $\mathbf{i}$ OGMs and environment-features change accordingly. The output should ideally be $\mathbf{i}_{t+1}$, which is fed back to the model for the next step prediction. This module is necessary to close the loop for *multi-step* prediction.

Learning-based module: Prediction module

The prediction module is the core of our model that predicts how the environment partially reacts to the ego-action, i.e. learns $p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$. Using the IOT1 and IOT2 modules the geometry of the target frame, $\mathbf{j}^{env}_{t+1}$, remains the same as the input frame, $\mathbf{j}^{ego}_{t+1}$, at each time step, regardless of the ego-action. Therefore $p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$ is a smoother function than the original objective function, $p(\mathbf{x}_{t+1}|\mathbf{x}_{1:t}, \mathbf{a}_t)$. Thus, $p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$ is in-

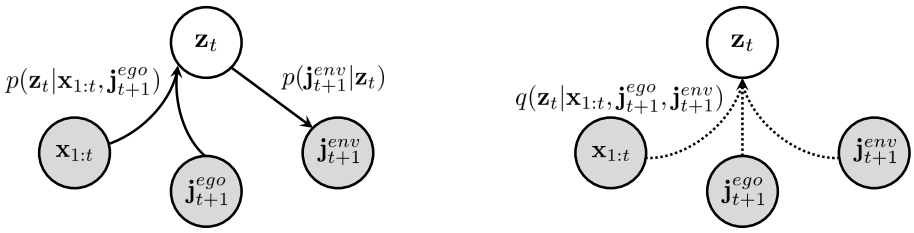


Figure 5.4: Graphical model at time t: Left: Generative links, $p(.)$. Right: Variational links, $q(.)$. Observable variables are gray.

tuitively easier to learn. Despite this simplification, the two objectives have the same optimum point, i.e. maximizing one leads to maximizing the other. The first term in Eq. 5.1 is deterministic and can be removed from the optimization. Also, $\mathbf{j}^{env}_{t+1}$ is uniquely determined by the action $\mathbf{a}_t$ (given $\mathbf{i}_{t+1}$). Consequently, we can re-write the second term of Eq. 5.1 as $\log p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$.

Bottleneck conditional density estimation: We maximize the conditional log-likelihood $\log p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$ in the framework of variational Bayes. We build our model upon a Bottleneck Conditional Density Estimation (BCDE) [86] model, a special variant of the conditional variational autoencoders (CVAEs) [89]. The latent code in BCDE acts as bottleneck of information and not just a source of randomness. The prior on the latent variable in BCDE is conditioned on the input. Such conditioning makes the model less prone to overfitting as it allows learning the distribution of the latent code conditioned on input, which is especially helpful for prediction with large horizon. We consider the graphical model in Fig. 5.4 at each time step for this prediction task. According to our definition of the approximating variational distribution in the graphical model, and also considering $\mathbf{z}_t$ as an information bottleneck between the input, $\mathbf{x}_{1:t}$ and $\mathbf{j}^{ego}_{t+1}$, and the target, $\mathbf{j}^{env}_{t+1}$, the ELBO to be maximized will have the following form:

$$\log p(\mathbf{j}^{env}_{t+1}|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1}) \geq \mathbb{E}_{q^*(\mathbf{z}_t)}[\log p(\mathbf{j}^{env}_{t+1}|\mathbf{z}_t)] - \mathrm{KL}\big(q^*(\mathbf{z}_t)||p(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})\big), \tag{5.5}$$

where $q^*(\mathbf{z}_t) = q(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1}, \mathbf{j}^{env}_{t+1})$. We implement each of the conditional probability distributions in Eq. 5.5 using a neural network and denote the parameters of $p_\psi(.)$ and

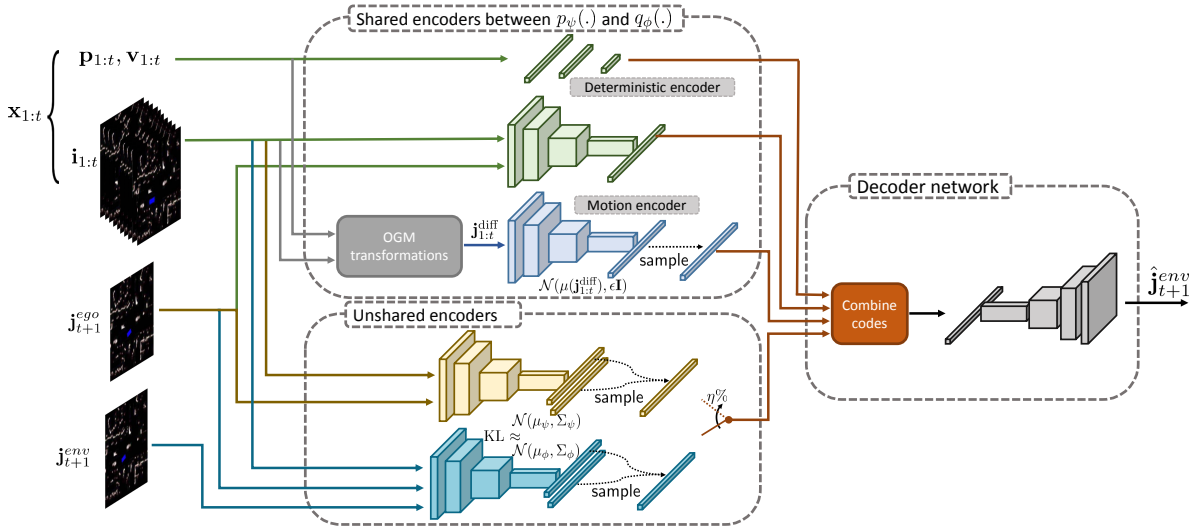


Figure 5.5: The prediction module at the training time. The measurements are encoded using fully-connected networks, while we use convolutional neural networks to encode and decode OGMs.

$q_\varphi(.)$ by ψ and φ, respectively.

Reconstruction loss and structural similarity: The first term in the ELBO in Eq. 5.5, can be interpreted as a reconstruction loss in the pixel space that measures the difference between the target OGM, $\mathbf{j}^{env}_{t+1}$, and predicted OGM, $\hat{\mathbf{j}}^{env}_{t+1}$, and we denote it by $\mathcal{D}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1})$. Depending on the type of values in the OGM being continuous or binary we consider Gaussian (with identity covariance matrix) or Bernoulli distributions for the output and replace $\mathcal{D}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1})$ with the mean squared error (MSE) or the cross entropy (CE), respectively. We also add an auxiliary term to our reconstruction loss that computes the Structural Similarity Index (SSIM) [102] loss between prediction and target. The experiments show the effectiveness of adding this term in improving the quality of the predicted frames. The weight of the SSIM term, λ, is set using the validation set.

Code splitting and sampling from prior: The second term, is a KL divergence regularization that minimizes the distance between the output distributions of $p_\psi(.)$ and $q_\varphi(.)$ encoders. Since the observations are highly dynamic with many objects in the scene, merely minimizing the KL divergence does not provide a proper training for the $p_\psi(.)$

encoder. Therefore we employ two ideas to better match these two distributions.

1) We split the latent code $\mathbf{z}_t$ into two parts with two different sets of encoders. For the first part we do not use the target frame as the input of the $q_\varphi(.)$ encoder and therefore its parameters can be shared with the $p_\psi(.)$ encoder, which guarantees the minimization of KL divergence. We call this part, *shared code* and since it only encodes the previous and current observations we assume it is partly deterministic. For the second part, called *unshared code*, we assume Gaussian distributions for both the conditional prior $p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$ and the variational posterior $q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1}, \mathbf{j}^{env}_{t+1})$ and minimize their KL divergence. This part of $\mathbf{z}_t$ encodes information about the target, $\mathbf{j}^{env}_{t+1}$, and its stochasticity represents the uncertainty about the future. By combining this splitting idea with the BCDE model we introduce another important difference with the vanilla CVAE model, i.e. instead of concatenating the stochastic code with our high-dimensional input, we concatenate it with an encoded version of the input that keeps only useful information. This makes the structure of the decoder simpler with far fewer parameters, and therefore easier to train.

2) To make sure that the encoder of $p_\psi(.)$ for the unshared code is trained properly, we randomly switch between the stochastic samples of the $p_\psi(.)$ and $q_\varphi(.)$ encoders, i.e. $\eta\%$ of the time the samples are drawn from $p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})$ instead of $q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1}, \mathbf{j}^{env}_{t+1})$. This way the $p_\psi(.)$ encoder is trained by backpropagating both errors of the KL term and the reconstruction term. In our experiments we set $\eta = 10$. The final training objective to be minimized is:

$$\begin{aligned}\mathcal{L}_t = &\underbrace{\mathcal{D}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1}) + \lambda\big(1 - \text{SSIM}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1})\big)}_{\mathcal{L}^{\text{rec.}}_t} \\ &+ \underbrace{\text{KL}\big(q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1}, \mathbf{j}^{env}_{t+1})||p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}^{ego}_{t+1})\big)}_{\mathcal{L}^{\text{KL}}_t}. \end{aligned} \tag{5.6}$$

For a multi-step prediction with horizon k a summation over $\mathcal{L}_t$ is minimized: $\min_{\psi,\varphi} \sum_{j=0}^{k-1} \mathcal{L}_{t+j}$.

5.4.2 Difference Learning (DL)

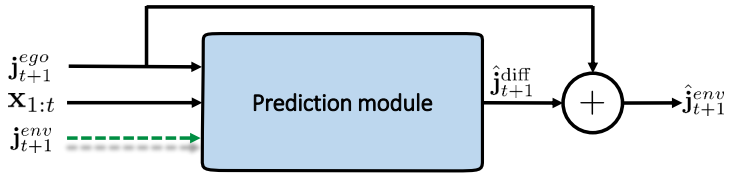


Figure 5.6: Difference learning module

After the OGM transformations, the difference between $\mathbf{j}^{ego}_{t+1}$ and $\mathbf{j}^{env}_{t+1}$ is only in the position of the moving objects. Therefore we also propose a variant of our model that explicitly learns the difference between the two OGMs, denoted by $\mathbf{j}^{\text{diff}}_{t+1} = \mathbf{j}^{env}_{t+1} - \mathbf{j}^{ego}_{t+1}$. In fact, $\mathbf{j}^{\text{diff}}_{t+1}$ represents the motion of other agents in the pixel space. Fig. 5.6 shows the difference learning module. One issue with this model is that, for incorrect predictions during training, adding $\hat{\mathbf{j}}^{\text{diff}}_{t+1}$ to the input frame $\mathbf{j}^{ego}_{t+1}$ causes $\hat{\mathbf{j}}^{env}_{t+1}$ to go out of the range of the input. This is especially problematic for multi-step prediction when this error is accumulated. A similar structure has been suggested in [65] for predicting binary OGMs, where a classifier layer is used after the summation to take the result within the input range. Although this can potentially resolve the issue, adding such classifier makes the learning process very slow. This is because the classifier ideally acts as a (sigmoid-shaped) clipper and the derivative of the clipper for the out of range values, caused by the actual error, is very small. In our model, we do not use these additional layers. The first term of our reconstruction loss remains the same as before and for the SSIM part we simply clip the prediction $\hat{\mathbf{j}}^{env}_{t+1}$ and then compute the SSIM:

$$\mathcal{L}^{\text{rec.}}_{t} = \mathcal{D}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1}) + \lambda\big(1 - \text{SSIM}(\mathbf{j}^{env}_{t+1}, \mathit{clip}(\hat{\mathbf{j}}^{env}_{t+1}))\big). \tag{5.7}$$

This enables us to backpropagate the incorrect difference predictions through the $\mathcal{D}(\mathbf{j}^{env}_{t+1}, \hat{\mathbf{j}}^{env}_{t+1})$ term and also to make use of the SSIM term. For future time-steps the clipped output is fed back to the network.

Motion Encoding: To further enhance the predictive power of our model we capture the past motion of other agents in the scene by encoding the difference between consecutive OGMs in the input sequence. In fact, we built a sequence of $\mathbf{j}^{\text{diff}}_{1:t}$ from the input

observations and encode it as a part of the shared code. While $\mathbf{i}_{1:t}$ contain information about motion of other agents relative to the ego-vehicle, $\mathbf{j}^{\text{diff}}_{1:t}$ represent their absolute motion and therefore encoding $\mathbf{j}^{\text{diff}}_{1:t}$ allows reasoning about higher level motion features such as intention of other agents. We model these features by a Gaussian distribution $\mathcal{N}(\mu(\mathbf{j}^{\text{diff}}_{1:t}), \epsilon\mathbf{I})$, where $\mu(\mathbf{j}^{\text{diff}}_{1:t})$ is a neural network and ϵ is a constant ($\epsilon = 0.5$ in the experiments). Motion encoding is used in both the base model and the DL variant.

Fig. 5.5 provides a high level architecture of the model. The implementation details of the prediction module are provided in the supplementary material.

5.5 Experiments

In this section, we evaluate the performance of our model, i.e. prediction by anticipation and its difference learning extension, referred to as PA and PA-DL, respectively.

Baselines: The proposed model is an OGM-in OGM-out model and therefore we compare its performance with two similar prediction models, which are, to the best of our knowledge, the state-of-the-art for this unsupervised prediction task:

- **Forward Model in Model-Predictive Policy Learning with Uncertainty Regularization (FM-MPUR)** [39]: FM-MPUR is a CVAE-based model, which aims to directly maximize the log-likelihood $\log p(\mathbf{x}_{t+1:t+k}|\mathbf{x}_{1:t}, \mathbf{a}_{t:t+k-1})$. Latent code of the FM-MPUR model has an unconditioned prior. Therefore, latent samples are independent of the input frames. This can potentially hurt the prediction accuracy for longer horizons.
- **RNN-based model with Difference Learning component (RNN-Diff)** [65]: RNN-Diff is an encoder-decoder structure that uses an RNN in the code space. Encoding and decoding are done using convolutional layers. The main idea in RNN-Diff is removing the ego-actions for a whole sequence of frames as if the ego-vehicle does not move and the scene is observed by a fixed observer for the whole sequence. Therefore they can just focus on predicting the movement of dynamic objects in

a fixed scene by learning the difference between consecutive frames. We use the best architecture of their model, named RNN-Diff2.1, for comparison.

FM-MPUR takes 20 frames as its input. However, we found 10 input frames to be as rich as 20 in terms of information about the past, i.e. the Markov property of the sequence is preserved by 10 frames. Therefore a sum over the conditional log-likelihood can result in a log-likelihood of the whole training set. The RNN-Diff2.1 model also uses 10 input frames.

Metrics: For real-valued OGMs we report *MSE*. For binary OGMs we assign class 1 (positive) to occupied pixel and class 0 (negative) to free pixels and report the results in terms of classification scores, i.e. *true positive (TP) and true negative (TN)*. This enable us to distinguish between the accuracy of predicting occupied and free pixels. TP is the more important metric for safe driving as it shows how well a model predicts the obstacles in the environment. We also report the results in terms of *average log-likelihood (ALL)*. In fact, we use kernel density estimation (KDE) by approximating the pdf of the training data using 10K training samples and then evaluating the approximated pdf on the predicted frames of the test sequences with different prediction horizons. We use Gaussian kernel with $\sigma = 0.1$.

5.5.1 Prediction under different driving situations

We ran our experiments on two complimentary datasets, one high-speed highway traffic and the other low-speed dense urban traffic. These two datasets cover a large set of real-life driving scenarios.

NGSIM I-80 dataset: The Next Generation Simulation program's Interstate 80 (NGSIM I-80) [38] dataset consists of 3 batches of 15-minute of recordings from traffic cameras mounted over a stretch of a highway in the US. Driving behaviours are complex with complicated interactions between vehicles moving at high-speed. This makes the future state difficult to predict. We follow the same preprocessing proposed in [39] to make the datasets. The images are real-valued RGB with size 117×24. The ego-vehicle is in the

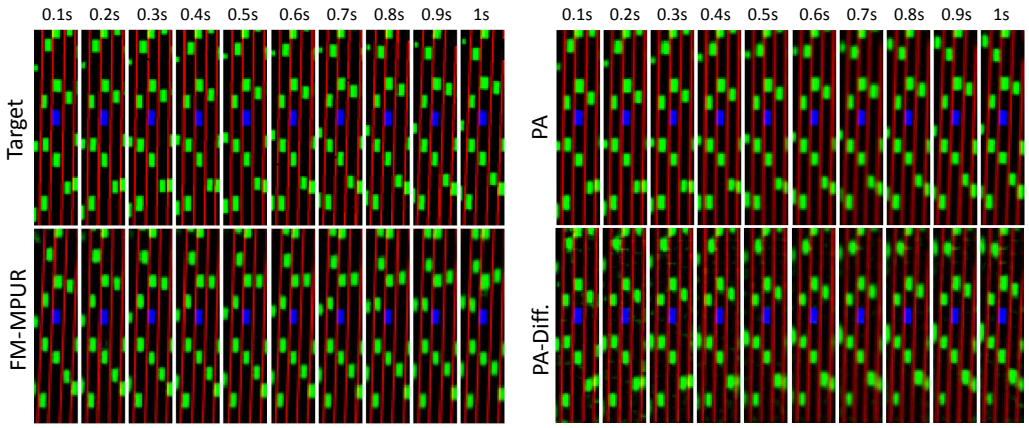


Figure 5.7: Predictions of 10 frames (1 sec) by different models.

center of the blue channel. Other (social) vehicles are in the green channel, which can also be interpreted as the OGM. The red channel has the map information, e.g. lanes.

We train each model using two batches of the 15-minute recording and test it on the third batch and repeat this process three times to cover all combinations. For both PA and PA-DL, we put $\lambda = 0.05$ in the reconstruction cost. Results are shown in Tables 5.1 and 5.5. Since the speed of the ego-vehicle is high in this dataset, removing the ego-motion for the whole sequence to train the RNN-Diff2.1, significantly reduces the size of meaningful pixels in the input and target frames and make them practically unusable for training a multi-step model. Therefore the reported results are from a model trained for single-step prediction. This is why its performance dramatically drops for larger values of k. As we can see, PA and PA-DL outperform FM-MPUR. Due to high-speed driving of agents in this dataset, their positions have dramatic changes from one frame to the next one, in many cases. Consequently, PA-DL performs worse than the base model. Fig. 5.7 shows a sequence of predictions for different models to allow us to see PA, PA-DL, and FM-MPUR performances qualitatively.

Argoverse dataset: For the urban area driving, the OGM sequences are obtained from the Argoverse raw dataset [20]. The dataset contains many different actions and maneuvers, e.g. stops and turns, in slow pace. The LiDAR point-clouds, collected at 10Hz, are converted to BEV 256×256 binary OGMs using ground removal proposed in

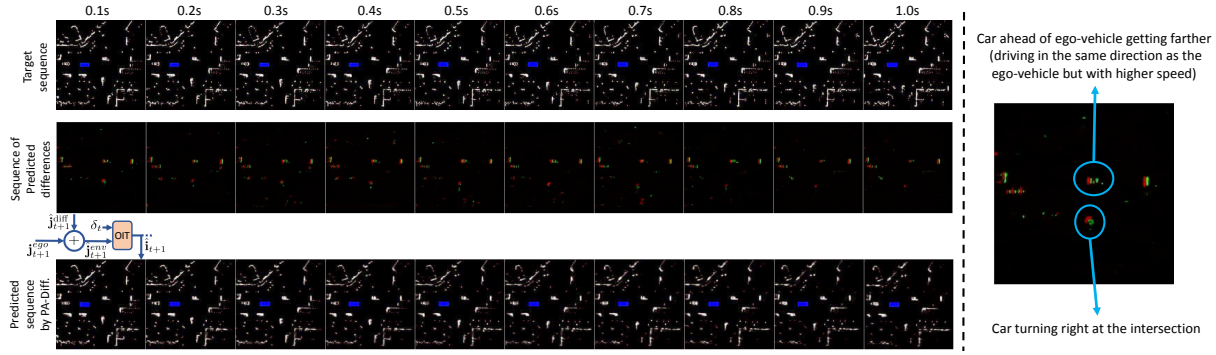


Figure 5.8: Left: OGM prediction of PA-DL for the Argoverse dataset. Top row shows the target sequence. Middle row shows the sequence of predicted differences, learned by the model, where red areas (negative values) are erased from the frame and green areas (positive values) are added to build the next frame. Bottom row shows the final predicted frame. We demonstrate the mechanism to build the predictions for the first time step. Right: Zoomed-in first predicted difference. Difference learning allows reasoning about the motion of other agents.

[64]. We compare the performance of the models in terms of TP/TN and ALL in Tables 5.1 and 5.5, respectively. Argoverse dataset has more complicated OGM structures and, unlike NGSIM I-80, the ego-motion is usually small and consecutive frames have slight differences. Therefore, PA-DL outperforms PA, and both PA and PA-DL outperform FM-MPUR significantly.

PA-DL and RNN-Diff2.1 perform closely for short-horizon predictions. Again, since in RNN-Diff2.1 the environment is observed from a fixed point, when k is large the dynamic objects eventually leave the scene and the predictions deviate from the actual ground truth. In PA-DL, the difference learning is done step-by-step. As we can see, the performance gap between RNN-Diff2.1 and PA-DL enlarges as k grows. Fig. 5.8 shows a sample sequence of the Argoverse dataset as well as the outputs of the PA-DL model.

The results of Tables 5.1 and 5.5 show that the proposed models outperform the two baselines with a significant margin. Moreover, each of the baselines fails in one of the two datasets, while PA and PA-DL perform the prediction task successfully for both datsets, i.e. both driving situations.

Dataset →	NGSIM I-80			
Method	MSE			
	$k=1$	$k=5$	$k=10$	$k=20$
FM-MPUR	3.4 ± 0.1	4.7 ± 0.2	5.2 ± 0.2	7.8 ± 0.1
RNN-Diff2.1	4.2 ± 0.2	14.5 ± 0.3	36.6 ± 1.1	80.2 ± 2.0
PA	$\mathbf{2.9 \pm 0.2}$	$\mathbf{4.1 \pm 0.1}$	$\mathbf{4.7 \pm 0.1}$	$\mathbf{6.2 \pm 0.2}$
PA-DL	3.3 ± 0.1	4.6 ± 0.1	5.1 ± 0.2	6.7 ± 0.3

Dataset →	Argoverse							
Method	TP	TN	TP	TN	TP	TN	TP	TN
	$k=1$		$k=5$		$k=10$		$k=20$	
FM-MPUR	96.24	99.68	88.44	97.12	74.62	94.21	65.60	85.31
RNN-Diff2.1	99.21	**99.91**	93.19	**99.85**	87.23	99.27	80.55	93.92
PA	99.18	99.89	94.12	99.76	90.14	99.48	83.17	96.87
PA-DL	**99.40**	**99.91**	**97.13**	99.83	**92.55**	**99.71**	**87.98**	**98.02**

Table 5.1: Comparison of different models in terms of MSE for NGSIM I-80 and TP/TN for Argoverse. For this table predictions for all methods except RNN-Diff2.1 are generated using the mean value for the latent code.

Dataset →	NGSIM I-80			
Method	$k=1$	$k=5$	$k=10$	$k=20$
FM-MPUR	212.6 ± 5.3	207.1 ± 6.9	201.4 ± 5.1	187.3 ± 7.5
RNN-Diff2.1	194.2 ± 7.1	141.3 ± 3.6	71.1 ± 3.4	26.6 ± 1.1
PA	$\mathbf{236.9 \pm 2.7}$	$\mathbf{232.66 \pm 3.9}$	$\mathbf{225.4 \pm 2.4}$	$\mathbf{211.2 \pm 5.2}$
PA-DL	221.7 ± 1.7	217.5 ± 3.1	210.2 ± 4.8	202.9 ± 5.1

Dataset →	Argoverse			
Method	$k=1$	$k=5$	$k=10$	$k=20$
FM-MPUR	624.3 ± 8.1	609.5 ± 5.6	538.4 ± 6.2	461.7 ± 4.5
RNN-Diff2.1	656.2 ± 2.9	631.0 ± 3.0	603.1 ± 5.6	545.6 ± 5.8
PA	661.1 ± 6.7	657.1 ± 7.9	635.2 ± 5.3	590.7 ± 4.2
PA-DL	$\mathbf{674.6 \pm 2.2}$	$\mathbf{660.2 \pm 2.2}$	$\mathbf{642.2 \pm 3.9}$	$\mathbf{603.4 \pm 3.7}$

Table 5.2: Comparison of different models in terms of ALL.

5.5.2 Prediction for rare actions

In this section we study the performance of the PA and PA-DL algorithms in the presence of rare actions and investigate the effectiveness of employing prior knowledge in providing robustness against the actions that are rare in the training data. Specifically we consider the NGSIM I-80 dataset.

Method	ALL			
	$k = 1$	$k = 5$	$k = 10$	$k = 20$
FM-MPUR	197.3 ± 6.2	175.7 ± 5.1	112.1 ± 7.7	76.4 ± 4.1
PA	$\mathbf{226.1 \pm 2.3}$	$\mathbf{218.6 \pm 4.1}$	$\mathbf{202.2 \pm 3.8}$	$\mathbf{180.4 \pm 6.6}$
PA-DL	213.4 ± 6.8	200.4 ± 4.9	192.9 ± 3.5	174.8 ± 7.9

Table 5.3: Comparison of predictions of PA, PA-DL, and FM-MPUR using rare actions.

We use the trained models with each of the batches of 15-minute recordings and apply actions that are rarely seen in the training set but are still in the maneuverability range of vehicles. We use the distributions shown in Fig. 5.9 to sample these actions and apply them to randomly selected sequences of the test set and predict for different prediction horizons. For comparison, we use the FM-MPUR algorithm. Since there is no ground truth, we only report the ALL results.

Table 5.6 summarizes the evaluation results. It shows that our models significantly outperform FM-MPUR for this task, especially for longer prediction horizons. This suggests that learning environment-features based on the anticipated ego-features makes the prediction task easier to learn for our model, which supports our initial intuition. In fact, by applying the extreme actions, the OGMs change dramatically from one time step to the next one. However, we can compensate this change by applying the anticipated modifications to the target OGM. Fig. 5.10 shows the result of applying rare actions to the same input sequence in Fig. 5.7. We apply $\mathbf{a} = [-25, 0]$ for 20 consecutive steps, which can be identified as a very low-probable action sequence according to the distributions in Fig. 5.9. This is equivalent to a hard brake in the middle of the road. As we can see our model can predict almost perfectly, while the FM-MPUR model fails after a few predictions. Compared to the predictions that correspond to the original actions in Fig. 5.7, location of the nearby social vehicles show that the model has learned the dynamics of the traffic: as the ego-vehicle brakes hard, other vehicles continue to move normally except for the one behind it, which is forced to slow down significantly.

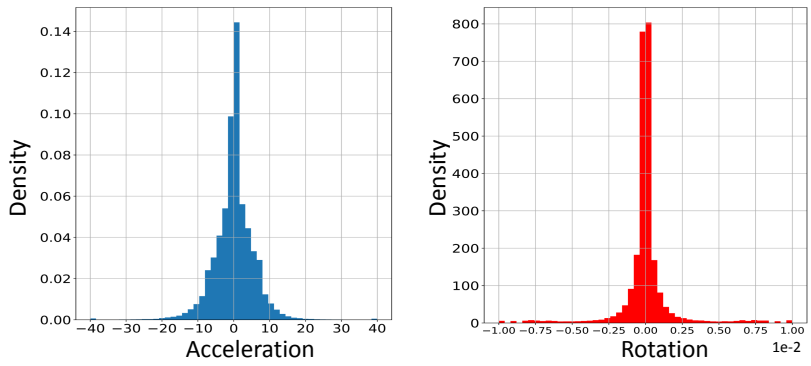


Figure 5.9: Distribution of actions.

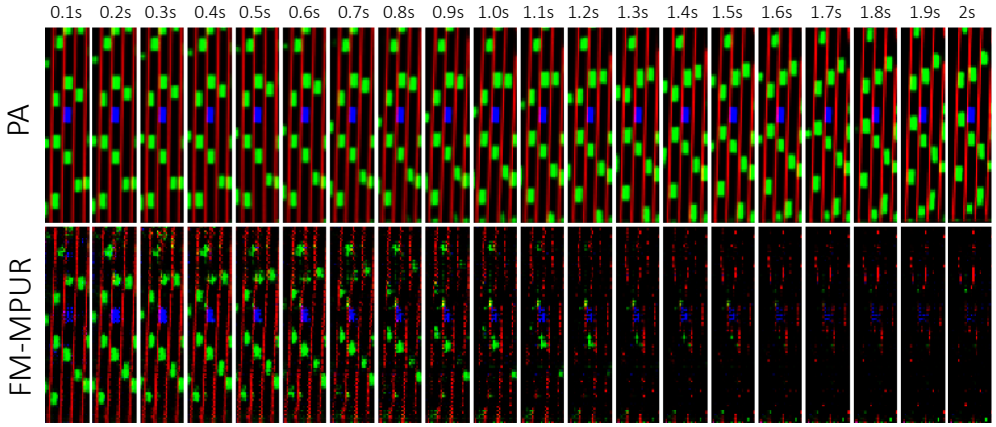


Figure 5.10: Effect of constantly applying rare actions on the prediction of PA and FM-MPUR models.

5.5.3 Ablation study

There are three main factors contributing to the better performance of our model compared to the baselines: 1) Employing prior knowledge using rule-based modules to set up the *anticipation-interaction* training. 2) Employing the bottleneck model that conditions the prior of the latent code on input. 3) Encoding *absolute* motion of other agents. We conduct an ablative study on each of these factors for both PA and PA-DL models and provide the results in Table 5.4 based on MSE and TP/TN . Comparing the results of this table with Table 5.1, we can see that while all factors are contributing, the rule-based

modules contribute more. Since the difference learning is a byproduct of our main idea of, removing the rule-based modules degrades the performance of PA-DL significantly. Also motion encoding plays a slightly more important role than the BCDE (conditioned prior) model. The effect of using conditioned prior becomes more apparent for larger values of k.

Dataset →	NGSIM I-80			
Method	MSE			
	$k=1$	$k=5$	$k=10$	$k=20$
PA (no RBM)	3.2 ± 0.2	4.5 ± 0.2	5.0 ± 0.1	7.2 ± 0.4
PA (no BCDE)	3.1 ± 0.3	4.2 ± 0.4	4.7 ± 0.2	6.9 ± 0.3
PA (no ME)	3.1 ± 0.2	4.3 ± 0.4	4.8 ± 0.3	6.7 ± 0.2
PA-DL (no RBM)	4.1 ± 0.4	6.9 ± 0.3	7.3 ± 0.1	10.8 ± 0.2
PA-DL (no BCDE)	3.4 ± 0.2	4.7 ± 0.1	5.6 ± 0.2	8.4 ± 0.2
PA-DL (no ME)	3.6 ± 0.2	4.7 ± 0.1	5.4 ± 0.2	8.1 ± 0.3

Dataset →	Argoverse							
Method	TP	TN	TP	TN	TP	TN	TP	TN
	$k=1$		$k=5$		$k=10$		$k=20$	
PA (no RBM)	97.03	99.74	90.15	97.98	80.12	95.23	72.12	88.63
PA (no BCDE)	99.01	99.84	93.20	99.64	88.18	99.22	80.14	92.20
PA (no ME)	98.65	99.75	92.51	98.50	86.19	98.98	81.35	92.71
PA-DL (no RBM)	95.51	99.83	91.94	97.72	81.25	95.74	70.55	86.30
PA-DL (no BCDE)	99.20	99.88	96.95	99.70	90.62	99.51	83.24	91.95
PA-DL (no ME)	99.05	99.80	96.54	99.71	91.14	99.59	84.64	94.34

Table 5.4: Results of ablative study on the contributing factors to the performance of our models. no RBM: a model without rule-based modules. no BCDE: a CVAE-based model with unconditioned prior for the latent code. no ME: a module without motion encoding.

5.5.4 SSIM term and ablation

The SSIM term helps to generate more clear images. Using the validation set the optimal weights for this term, denoted by λ^*, are $\lambda^* = 0.05$ for the NGSIM I-80 dataset and $\lambda^* = 0.1$ for the Argoverse dataset. This holds for both PA and PA-DL models.

The SSIM loss is an auxiliary term in our loss function that does not necessary appear in ELBO. Therefore, we present an ablation study on this term for each of the datasets and report the result in terms of MSE and TP/TN for the case we don't use SSIM in

Dataset →	NGSIM I-80			
Method	$k=1$	$k=5$	$k=10$	$k=20$
PA (no RBM)	219.4 ± 4.8	214.5 ± 5.4	207.0 ± 6.6	196.7 ± 7.3
PA (no BCDE)	232.5 ± 5.5	228.4 ± 4.9	218.9 ± 6.2	200.1 ± 6.0
PA (no ME)	230.4 ± 4.5	224.3 ± 6.9	217.6± 8.0	203.9 ± 5.2
PA-DL (no RBM)	210.5 ± 3.6	205.8 ± 5.8	196.3 ± 6.8	180.5 ± 7.1
PA-DL (no BCDE)	218.5 ± 3.1	214.2 ± 5.1	204.6 ± 6.6	193.4 ± 6.4
PA-DL (no ME)	218.2 ± 5.0	211.5 ± 7.2	205.8 ± 4.2	195.3 ± 7.5
Dataset →	Argoverse			
Method	$k=1$	$k=5$	$k=10$	$k=20$
PA (no RBM)	632.7 ± 9.2	614.5 ± 4.7	589.9 ± 7.2	535.9 ± 3.9
PA (no BCDE)	657.9 ± 4.5	650.4 ± 5.6	619.8 ± 6.9	573.6 ± 7.0
PA (no ME)	651.2 ± 3.4	636.2 ± 7.2	608.3 ± 3.2	555.1 ± 6.7
PA-DL (no RBM)	645.6 ± 6.2	623.3 ± 5.2	601.4 ± 6.5	557.2 ± 4.8
PA-DL (no BCDE)	668.2 ± 6.1	653.2 ± 4.4	633.1 ± 7.4	585.4 ± 7.8
PA-DL (no ME)	661.2 ± 5.5	641.8 ± 7.6	620.7 ± 5.4	577.1 ± 5.0

Table 5.5: Ablative study in terms of ALL for regular actions.

Method	ALL			
	$k=1$	$k=5$	$k=10$	$k=20$
PA (no RBM)	203.5 ± 4.5	182.9 ± 6.3	134.7 ± 3.4	99.7 ± 6.2
PA (no BCDE)	221.2 ± 6.2	211.6 ± 7.5	190.9 ± 5.0	161.2 ± 7.3
PA (no ME)	219.7 ± 6.2	205.2 ± 6.8	188.2 ± 7.2	163.8 ± 9.2
PA-DL (no RBM)	198.6 ± 5.4	172.4 ± 6.8	125.5 ± 6.2	82.3 ± 4.4
PA-DL (no BCDE)	208.9 ± 3.9	194.6 ± 5.6	178.5 ± 4.9	142.5 ± 6.8
PA-DL (no ME)	209.5 ± 4.3	191.5 ± 5.8	180.6 ± 5.0	149.8 ± 6.9

Table 5.6: Ablative study in terms of ALL for low-probable actions in NGSIM I-80 dataset.

the loss function. A comparison between $\lambda = 0$, i.e. not using the SSIM term, and λ^* is presented in Table 5.7 .

As we can see in both cases, adding the the SSIM term is effective, especially for larger values of k. This is due to the fact that the error of the prediction is accumulative.

It is worth mentioning that the effect of using a conditioned prior for the latent (BCDE model) is more significant for low-probable actions, as the model with conditioned code has access to its previous predictions when estimating the parameter of the latent distribution.

Dataset →	NGSIM I-80			
Method	MSE			
	$k = 1$	$k = 5$	$k = 10$	$k = 20$
PA (λ^*)	2.9 ± 0.2	4.1 ± 0.3	4.7 ± 0.3	6.2 ± 0.6
PA (λ = 0)	2.9 ± 0.3	4.0 ± 0.3	4.9 ± 0.2	6.5 ± 0.4
PA-DL (λ^*)	3.3 ± 0.2	4.6 ± 0.3	5.1 ± 0.4	6.7 ± 0.6
PA-DL (λ = 0)	3.4 ± 0.2	4.6 ± 0.2	5.2 ± 0.4	6.9 ± 0.4

Dataset →	Argoverse							
Method	TP	TN	TP	TN	TP	TN	TP	TN
$k = 1$	$k = 5$		$k = 10$		$k = 20$			
PA (λ^*)	99.18	99.89	94.12	99.76	90.14	99.48	83.17	96.87
PA (λ = 0)	99.09	99.88	94.03	99.64	87.56	98.19	82.25	94.03
PA-DL (λ^*)	99.40	99.91	97.13	99.83	92.55	99.71	87.98	98.02
PA-DL (λ = 0)	99.04	99.84	96.35	99.51	90.99	99.47	85.15	96.13

Table 5.7: Effect of the SSIM term in terms of MSE for NGSIM I-80 and TP/TN for Argoverse.

5.6 Summary

We proposed that an observed interaction sequence can be explained by an underlying generative process wherein some agents act partly in response to the anticipated action of other agents. Based on this view, we factorized the interaction sequence into anticipated action and anticipated partial reaction, thereby setting up an action-conditional distribution. We designed a bottleneck conditional density estimation model to learn the distribution. In comparison to the baselines, our model achieves a higher capacity for prediction: it reaches higher accuracy, it handles rare actions much better, it is able to perform well under different driving situations, including high-speed highway driving and complicated urban navigation. While our experiments are limited to vehicle-vehicle interaction, insofar as the understanding generative process is pertinent, our method may also generalize well to other tasks, such as prediction of pedestrian-vehicle interaction, or to other multi-agent domains. Finally, because our model is action conditional, it can serve as a world model for many downstream tasks.

It is also worth mentioning that in this work we did not need to worry about ego and social vehicles interactions. This is due to two main reasons: 1) During training we use the ground truth actions in the dataset. Since these actions come from a human

(expert) driver, they already reflect the previous observations and interaction with other agents. 2) We *do not* intend to learn the *optimal* action in this work. We want to have a model that learns how the environment changes based on the expert's action during the training and then use that model to predict $\mathbf{x}_{t+1}^{env}$ based on other actions, which are not necessarily optimal or sub-optimal. As we showed in the experiments our model successfully performs this task even for rare (extreme) actions, e.g. a hard brake in a highway.

One drawback of prediction in high-dimensional space is the gradual degradation of the predicted frames' quality. This can potentially limit the capability of such models to be used in application like building driving simulators.

Chapter 6

Conclusion and Future Work

Here we recap the contributions and discuss possible some future research directions.

6.1 Contributions

In this dissertation we presented several algorithms to tackle different aspects of planning and sequential decision making. Here we revisit the research questions raised at the beginning and provide a summary of our proposed solutions.

6.1.1 RQ1: Policy optimization over a known MDP

We proved that the problem of finding an optimal policy in an MDP under a restricted policy class defined by the convex hull of a set of base policies is an NP-hard problem to be solved exactly as well as approximated to arbitrary accuracy [4]. We show that under the condition that the corresponding occupancy measures of the base policies have large overlaps, there is an efficient approximating algorithm that works in the dual space and can output an policy almost as good as the optimal policy in the convex hull. The algorithm has linear running time in the number of states and polynomial

running time in the number of base policies. We also showed that due the fact that the algorithm works completely in the occupancy measure space, there is no need to run the output of the algorithm in the middle of training to obtain the value of the policies and therefore we will not have unsafe interactions with the environment.

6.1.2 RQ2: Finding a prediction model from high-dimensional observation of an unknown MDP

Here we looked at two different problems:

- **High-dimensional observations of a single dynamical system:** For this problem we proposed a model that uses VAEs to encode the high-dimensional observations into much lower-dimensional in which the transition function can be defined using a locally-linear function [11]. We also showed that this representation can be efficiently used for locally-linear control, and in particular we used iLQR algorithm for planning in the low-dimensional space. We also showed that due to our proper definition of the recognition model in the VAE, the model is much more robust to the noise in the dynamical system, compared to its rival models.

  We also showed that using feature disentanglement generalize our model to solve the problems that have similar underlying dynamics and only differ in terms of content of the observation.

- **High-dimensional observations of a multi-agent system:** We considered the problem of finding a prediction model for autonomous driving, where we have access to sequence of high-dimensional observations of the system. We proposed a model that employs human-level prior knowledge about the problem and splits the action-based prediction problem into two steps: first using prior knowledge we apply the first order effect of the action on the ego-agent by changing the ego-related features in our observations. Then we learn from the data how the environment reacts to this change [10]. For such learning we again used a VAE-based

framework. The probabilistic nature of this framework helps us capture our uncertainty about environment reaction to the ego-action. We showed in our experiments that exploiting the prior knowledge makes the prediction task more robust to the ego-action, and therefore we can handle a wider range of actions and driving situations using our prediction model. The proposed prediction model can act as a *world model* for downstream tasks, such as planning.

6.2 Future work

There are some possible research directions that can extend the proposed methods in this dissertations.

Combining dissimilar policies in MDPs: In Chapter 2 we proposed an efficient algorithm for combining a set of base policies under the condition that they have large overlaps in the occupancy measure space. This condition implies the assumption that the base policies are similar in the sense that they have similar values. This assumption might be too strong in many situations. It would be interesting to investigate this problem when such assumption is relaxed and design an approximating algorithm for combining the base policies.

Modelling dynamical systems from high-dimensional observations: In Chapter 4 and Chapter 5 of this dissertation we used maximum likelihood approach for modelling dynamical systems. In fact we used an encoder-decoder structure in which the dynamics and interactions are learned in a low-dimensional space (explicitly in Chapter 4 and implicitly 5). We showed that such modelling can in fact helps learning the dynamics, however, this might not be the optimal solution for planning. In other words, combining maximum likelihood objective with additional objective that directly consider the needs of the planner algorithm can potentially improve this line of work. For example, adding the iLQR algorithm's objective to the ELBO term during the training of the RCE algorithm seems to be a rational next step.

Disentangling dynamics and content: In Chapter 4 we showcased how an RCE-based

model can be used for disentangling dynamics- and content-related features of the observations. This model has potential applications in autonomous driving. Self-driving cars use different sensors to observe the surrounding environment including cameras, LiDAR, GPS, etc. Transferring knowledge among the domains of these observation can potentially results in using more accurate modelling of the environment and/or employing fewer sensors.

References

[1] Y. Abbasi-Yadkori, P. Bartlett, and A. Malek. Linear programming for large-scale Markov decision problems. In *International Conference on Machine Learning (ICML)*, 2014.

[2] A. Achille, T. Eccles, L. Matthey, C. P. Burgess, N. Watters, A. Lerchner, and I. Higgins. Life-long disentangled representation learning with cross-domain latent homologies. *arXiv preprint arXiv:1808.06508*, 2018.

[3] C. Atkeson and J. Murimoto. Non-parametric representations of policies and value functions: A trajectory-based approach. In *Advances in Neural Information Processing Systems*, 2002.

[4] E. Banijamali, Y. Abbasi-Yadkori, M. Ghavamzadeh, and N. Vlassis. Optimizing over a restricted policy class in mdps. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3042–3050. PMLR, 2019.

[5] E. Banijamali and A. Ghodsi. Fast spectral clustering using autoencoders and landmarks. In *International Conference Image Analysis and Recognition*, pages 380–388. Springer, 2017.

[6] E. Banijamali, A. Ghodsi, and P. Popuart. Generative mixture of networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3753–3760. IEEE, 2017.

[7] E. Banijamali, A. Karimi, A. Wong, and A. Ghodsi. Jade: Joint autoencoders for dis-entanglement. In *Learning Disentangled Representations, NIPS Workshop*, 2017.

[8] E. Banijamali, A.-H. Karimi, and A. Ghodsi. Deep variational sufficient dimensionality reduction. In *Bayesian Deep Learning, NIPS Workshop*, 2018.

[9] E. Banijamali, A. Khajenezhad, A. Ghodsi, and M. Ghavamzadeh. Disentangling dynamics and content for control and planning. In *Learning Disentangled Representations, NIPS Workshop*, 2017.

[10] E. Banijamali, M. Rohani, E. Amirloo, J. Luo, and P. Poupart. Prediction by anticipation: An action-conditional prediction method based on interaction learning. In *IEEE/CVF International Conference on Computer Vision. ©[2021] IEEE. Reprinted, with permission*, 2021.

[11] E. Banijamali, R. Shu, H. Bui, A. Ghodsi, et al. Robust locally-linear controllable embedding. In *International Conference on Artificial Intelligence and Statistics*, pages 1751–1759. PMLR, 2018.

[12] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[13] J. Baxter and P. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[14] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

[15] W. Böhmer, J. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer. Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations. *Künstliche Intelligenz*, 29(4):353–362, 2015.

[16] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[17] S. Casas, W. Luo, and R. Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *Conference on Robot Learning*, pages 947–956, 2018.

[18] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *arXiv preprint arXiv:1910.05449*, 2019.

[19] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12):5017–5032, 2015.

[20] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[21] A. Chartsias, T. Joyce, G. Papanastasiou, S. Semple, M. Williams, D. E. Newby, R. Dharmakumar, and S. A. Tsaftaris. Disentangled representation learning in cardiac image analysis. *Medical image analysis*, 58:101535, 2019.

[22] R.-R. Chen and S. Meyn. Value iteration and optimization of multiclass queueing networks. *Queueing Systems*, 32(1-3):65–97, 1999.

[23] M. B. Cohen, J. Kelner, J. Peebles, R. Peng, A. B. Rao, A. Sidford, and A. Vladu. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *STOC*, 2017.

[24] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, and N. Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019.

[25] D. P. de Farias and B. Van Roy. The linear programming approach to approximate dynamic programming. *Operations research*, 51(6):850–865, 2003.

[26] E. Denton and V. Birodkar. Unsupervised learning of disentangled representations from video. *arXiv preprint arXiv:1705.10915*, 2017.

[27] N. Deo and M. M. Trivedi. Convolutional social pooling for vehicle trajectory prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1468–1476, 2018.

[28] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 37(4-5):492–512, 2018.

[29] A. Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. *arXiv preprint arXiv:1304.1098*, 2013.

[30] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[31] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[32] A. Globerson and N. Tishby. Sufficient dimensionality reduction. *Journal of Machine Learning Research*, 3(Mar):1307–1331, 2003.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[34] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[35] H. Greenspan, B. van Ginneken, and R. M. Summers. Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. *IEEE Transactions on Medical Imaging*, 35(5):1153–1159, 2016.

[36] O. K. Gupta and R. A. Jarvis. Optimal global path planning in time varying environments based on a cost evaluation function. In *Australasian Joint Conference on Artificial Intelligence*, pages 150–156. Springer, 2008.

[37] D. Ha and J. Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[38] J. Halkias and J. Colyar. Ngsim interstate 80 freeway dataset. *US Federal Highway Administration, FHWA-HRT-06-137, Washington, DC, USA*, 2006.

[39] M. Henaff, A. Canziani, and Y. LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *arXiv preprint arXiv:1901.02705*, 2019.

[40] I. Higgins, D. Amos, D. Pfau, S. Racaniere, L. Matthey, D. Rezende, and A. Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

[41] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[42] Y. Huang, H. Bi, Z. Li, T. Mao, and Z. Wang. Stgat: Modeling spatial-temporal interactions for human trajectory prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6272–6281, 2019.

[43] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

[44] D. Jacobson and D. Mayne. *Differential Dynamic Programming*. American Elsevier, 1970.

[45] V. John, L. Mou, H. Bahuleyan, and O. Vechtomova. Disentangled representation learning for non-parallel text style transfer. *arXiv preprint arXiv:1808.04339*, 2018.

[46] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. In *Proceedings of ICLR*, 2017.

[47] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

[48] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[49] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

[52] A. Kumar and S. Zilberstein. History-based controller design and optimization for partially observable MDPs. In *In International Conference on Automated Planning and Scheduling (ICAPS)*, 2015.

[53] P. Kumar and T. I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3):289–298, 1990.

[54] S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2010.

[55] Y. LeCun, C. Cortes, and C. J. Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[56] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[57] S. Levine and V. Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems*, 2013.

[58] W. Li and E. Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of ICINCO*, pages 222–229, 2004.

[59] M. Liang, B. Yang, W. Zeng, Y. Chen, R. Hu, S. Casas, and R. Urtasun. Pnpnet: End-to-end perception and prediction with tracking in the loop. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11553–11562, 2020.

[60] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6120–6127, 2019.

[61] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[62] M. F. Mathieu, J. J. Zhao, J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. Disentangling factors of variation in deep representation using adversarial training. In *Advances in Neural Information Processing Systems*, pages 5040–5048, 2016.

[63] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler. Online multi-target tracking using recurrent neural networks. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[64] I. Miller and M. Campbell. A mixture-model based algorithm for real-time terrain estimation. *Journal of Field Robotics*, 23(9):755–775, 2006.

[65] N. Mohajerin and M. Rohani. Multi-step prediction of occupancy grid maps with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10600–10608, 2019.

[66] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

[67] T. S. Motzkin and E. G. Straus. Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics*, 17:533–540, 1965.

[68] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. *Journal of ACM*, 47:681–720, 2000.

[69] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[70] H. Noguchi, T. Yamada, T. Mori, and T. Sato. Mobile robot path planning using human prediction model based on massive trajectories. In *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, pages 1–7. IEEE, 2012.

[71] T. Ohki, K. Nagatani, and K. Yoshida. Collision avoidance method for mobile robot considering motion and personal spaces of evacuees. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1819–1824. IEEE, 2010.

[72] Y. Pan and E. Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, 2014.

[73] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[74] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi. Sequence-to-sequence prediction of vehicle trajectory via lstm encoder-decoder architecture. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1672–1678. IEEE, 2018.

[75] J. Peters, S. Vijayakumar, and S. Schaal. Natural actor-critic. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 280–291, 2005.

[76] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[77] S. Reed, K. Sohn, Y. Zhang, and H. Lee. Learning to disentangle factors of variation with manifold interaction. In *International Conference on Machine Learning*, pages 1431–1439, 2014.

[78] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

[79] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1278–1286, 2014.

[80] N. Rhinehart, K. M. Kitani, and P. Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 772–788, 2018.

[81] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2821–2830, 2019.

[82] S. Rifai, Y. Bengio, A. Courville, P. Vincent, and M. Mirza. Disentangling factors of variation for facial expression recognition. *Computer Vision–ECCV 2012*, pages 808–822, 2012.

[83] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control. *arXiv preprint arXiv:2001.03093*, 2020.

[84] W. Sheng and X. Yin. Sufficient dimension reduction via distance covariance. *Journal of Computational and Graphical Statistics*, 25(1):91–104, 2016.

[85] R. Shu, H. Bui, and M. Ghavamzadeh. Bottleneck conditional density estimation. In *Proceedings of the International Conference on Machine Learning*, 2017.

[86] R. Shu, H. H. Bui, and M. Ghavamzadeh. Bottleneck conditional density estimation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3164–3172. JMLR. org, 2017.

[87] N. Siddharth, B. Paige, V. de Meent, A. Desmaison, F. Wood, N. D. Goodman, P. Kohli, P. H. Torr, et al. Learning disentangled representations with semi-supervised deep generative models. *arXiv preprint arXiv:1706.00400*, 2017.

[88] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[89] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

[90] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[91] R. Sutton and A. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

[92] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 2000.

[93] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[94] Y. Tassa, T. Erez, and W. Smart. Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems*, 2008.

[95] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained non-linear stochastic systems. In *Proceedings of the American Control Conference*, 2005.

[96] L. Tran, X. Yin, and X. Liu. Disentangled representation learning gan for pose-invariant face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1415–1424, 2017.

[97] E. Tsardoulias, A. Iliakopoulou, A. Kargakos, and L. Petrou. A review of global path planning methods for occupancy grid maps regardless of obstacle density. *Journal of Intelligent & Robotic Systems*, 84(1-4):829–858, 2016.

[98] S. Tulyakov, M.-Y. Liu, X. Yang, and J. Kautz. Mocogan: Decomposing motion and content for video generation. *arXiv preprint arXiv:1707.04993*, 2017.

[99] N. Vlassis, M. L. Littman, and D. Barber. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Transactions on Computation Theory*, 4(4):12, 2012.

[100] N. Wahlström, T. Schön, and M. Desienroth. From pixels to torques: Policy learning with deep dynamical models. In *arXiv preprint arXiv:1502.02251*, 2015.

[101] H. Wang, K. Tanaka, and M. Griffin. An approach to fuzzy control of nonlinear systems; stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1), 1996.

[102] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[103] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

[104] R. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[105] Y. Ye. A new complexity result on solving the Markov decision problem. *Mathematics of Operations Research*, 30:733–749, 2005.

[106] R. S. Yichuan Charlie Tang. Multiple futures prediction. *arXiv preprint arXiv:1911.00997*, 2015.

[107] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019.

Appendix

Appendix A

Robust Locally-Linear Controllable Embedding

A.1 Objective Function

Proof of Lemma 4.4.1. Suppose $q^\star = q(\mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}|\mathbf{x}_t, \mathbf{x}_{t+1})$. Consider the factorization of $q^\star$ based on Eq. 4.14 and also the factorization of $p(\mathbf{x}_{t+1}, \mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ based on Eq. 4.11. The variational lower bound on the conditional probability distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ can be derived as following:

$$\begin{aligned}
\log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) &\geq \mathbb{E}_{q^\star}\big[\log p(\mathbf{x}_{t+1}, \mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) - \log q^\star)\big] \\
&= \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})\big] - \mathbb{E}_{q^\star}\big[\log q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) + \log q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \\
&\quad + \log \delta(\mathbf{z}_t|\bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}, \mathbf{u}_t) - \log p(\mathbf{z}_t|\mathbf{x}_t) - \log p(\bar{\mathbf{z}}_t|\mathbf{x}_t) - \log \delta(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \bar{\mathbf{z}}_t, \mathbf{u}_t)\big]
\end{aligned} \tag{A.1}$$

We can simply ignore the $\delta(.|.)$ terms, because the cross entropy for these terms are

zero. Therefore the lower bound can be written as:

$$\begin{aligned}
\log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \geq\ & \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})\big] - \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log \frac{q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t)}{p(\bar{\mathbf{z}}_t|\mathbf{x}_t)}\big] \\
& - \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\log q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big] - \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log p(\mathbf{z}_t|\mathbf{x}_t)\big] \\
& = \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1})\big] \\
& - \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})}\big[\mathrm{KL}\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t)\big)\big] \\
& + \mathrm{H}\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big) + \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log p(\mathbf{z}_t|\mathbf{x}_t)\big] = \mathcal{L}_t^{RCE}
\end{aligned} \tag{A.2}$$

□

Proof of Lemma 4.6.1.

$$\begin{aligned}
& \log p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) + \log p(\mathbf{y}_t) \\
& \geq \mathbb{E}_{q^\star}\big[\log p(\mathbf{x}_{t+1}, \mathbf{z}_t, \bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}, \mathbf{w}|\mathbf{x}_t, \mathbf{u}_t) - \log q^\star\big] + \mathbb{E}_{q^\dagger}\big[\log p(\mathbf{y}_t, \mathbf{v}_t, \mathbf{w}) - \log q^\dagger\big] \\
& = \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\mathbf{w}|\mathbf{x}_{t+1})}}\big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}, \mathbf{w})\big] - \mathbb{E}_{q^\star}\big[\log q_\varphi(\mathbf{w}|\mathbf{x}_{t+1}) + \log q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\
& \quad + \log q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) + \log \delta(\mathbf{z}_t|\bar{\mathbf{z}}_t, \hat{\mathbf{z}}_{t+1}) - \log p(\mathbf{z}_t|\mathbf{x}_t) - \log p(\bar{\mathbf{z}}_t|\mathbf{x}_t) \\
& \quad - \log \delta(\hat{\mathbf{z}}_{t+1}|\mathbf{z}_t, \bar{\mathbf{z}}_t) - \log p(\mathbf{w})\big] \\
& \quad + \mathbb{E}_{q^\dagger}\big[\log p(\mathbf{y_t}|\mathbf{v}_t, \mathbf{w}) + \log p(\mathbf{v}_t) + \log p(\mathbf{w}) - \log q_\varphi(\mathbf{v}_t|\mathbf{y}_t) - \log q_\varphi(\mathbf{w}|\mathbf{y}_t)\big]
\end{aligned} \tag{A.3}$$

$$
\begin{aligned}
&= \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\mathbf{w}|\mathbf{x}_{t+1})}} \big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}, \mathbf{w}) \big] - \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}} \big[\log \frac{q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t)}{p(\bar{\mathbf{z}}_t|\mathbf{x}_t)} \big] \\
&\quad - \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})} \big[\log q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \big] - \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}} \big[\log p(\mathbf{z}_t|\mathbf{x}_t) \big] \\
&\quad - \mathbb{E}_{q_\varphi(\mathbf{w}|\mathbf{x}_t)} \big[\log q_\varphi(\mathbf{w}|\mathbf{x}_t) - \log p(\mathbf{w}) \big] + \mathbb{E}_{q^\dagger} \big[\log p(\mathbf{y_t}|\mathbf{v}_t, \mathbf{w}) \big] \\
&\quad - \mathbb{E}_{q_\varphi(\mathbf{v}_t|\mathbf{y}_t)} [\log q_\varphi(\mathbf{v}_t|\mathbf{y}_t) - \log p(\mathbf{v}_t)] \big] - \mathbb{E}_{q_\varphi(\mathbf{w}|\mathbf{y}_t)} \big[\log q_\varphi(\mathbf{w}|\mathbf{y}_t) - \log p(\mathbf{w}) \big] \\
&= \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\mathbf{w}|\mathbf{x}_{t+1})}} \big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}, \mathbf{w}) \big] - \mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})} \big[\mathrm{KL}\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t) \big) \big] \\
&\quad + \mathrm{H}\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \big) + \mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}} \big[\log p(\mathbf{z}_t|\mathbf{x}_t) \big] - \mathrm{KL}\big(q_\varphi(\mathbf{w}|\mathbf{x}_t) \parallel p(\mathbf{w}) \big) \\
&\quad + \mathbb{E}_{q^\dagger} \big[\log p(\mathbf{y_t}|\mathbf{v}_t, \mathbf{w}) \big] - \mathrm{KL}\big(q_\varphi(\mathbf{v}_t|\mathbf{y}_t) \parallel p(\mathbf{v}_t) \big) - \mathrm{KL}\big(q_\varphi(\mathbf{w}|\mathbf{y}_t) \parallel p(\mathbf{w}) \big) = \mathcal{L}^{DDC}
\end{aligned}
\tag{A.4}
$$

□

The terms in $\mathcal{L}_t^{RCE}$ and $\mathcal{L}_t^{DDC}$ can be written in closed forms:

1. $\mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})} \big[\log p(\mathbf{x}_{t+1}|\hat{\mathbf{z}}_{t+1}) \big]$

   Using the reparameterization trick [49], we should first sample from $\mathcal{N}(\mu_\varphi(\mathbf{x}_{t+1}), \mathbf{\Sigma}_\varphi(\mathbf{x}_{t+1}))$. Considering a Bernoulli distribution for the posterior of $\mathbf{x}_{t+1}$, the term inside the expectation is a binary cross entropy.

2. $\mathbb{E}_{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})} \Big[\mathrm{KL}\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t) \big) \Big]$

   Again, we first need to sample from $\mathcal{N}(\mu_\varphi(\mathbf{x}_{t+1}), \mathbf{\Sigma}_\varphi(\mathbf{x}_{t+1}))$. Note that $p(\bar{\mathbf{z}}_t|\mathbf{x}_t) = p(\mathbf{z}_t|\mathbf{x}_t)$ and $p(\mathbf{z}_t|\mathbf{x}_t) = q(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mu_\varphi(\mathbf{x}_t), \mathbf{\Sigma}_\varphi(\mathbf{x}_t))$. For the q_φ network, which is

the transition network in our model, we have $q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) = \mathcal{N}(\mu_\varphi, \mathbf{\Sigma}_\varphi)$. The KL term can be written as

$$\begin{aligned}\mathrm{KL}\big(q_\varphi(\bar{\mathbf{z}}_t|\hat{\mathbf{z}}_{t+1}, \mathbf{x}_t) \parallel p(\bar{\mathbf{z}}_t|\mathbf{x}_t)\big) = \frac{1}{2}\big(&\mathrm{Tr}(\mathbf{\Sigma}_\varphi(\mathbf{x}_t)^{-1}\mathbf{\Sigma}_\varphi) \\ &+ (\mu_\varphi(\mathbf{x}_t) - \mu_\varphi)^\top \mathbf{\Sigma}_\varphi(\mathbf{x}_t)^{-1}(\mu_\varphi(\mathbf{x}_t) - \mu_\varphi) \\ &+ \log(\frac{|\mathbf{\Sigma}_\varphi(\mathbf{x}_t)|}{|\mathbf{\Sigma}_\varphi|}) - n_z\big)\end{aligned} \tag{A.5}$$

3. $\mathrm{H}\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big)$

   The entropy term for the encoding network can be easily written as

$$\mathrm{H}\big(q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1})\big) = \frac{1}{2}\log\left((2\pi e)^{n_z}|\mathbf{\Sigma}_\varphi(\mathbf{x}_{t+1})|\right) \tag{A.6}$$

4. $\mathbb{E}_{\substack{q_\varphi(\hat{\mathbf{z}}_{t+1}|\mathbf{x}_{t+1}) \\ q_\varphi(\bar{\mathbf{z}}_t|\mathbf{x}_t, \hat{\mathbf{z}}_{t+1})}}\big[\log p(\mathbf{z}_t|\mathbf{x}_t)\big]$

   Here we first need to sample from $\mathcal{N}(\mu_\varphi(\mathbf{x}_{t+1}), \mathbf{\Sigma}_\varphi(\mathbf{x}_{t+1}))$ and $\mathcal{N}(\mu_\varphi, \mathbf{\Sigma}_\varphi)$. Given that $p(\mathbf{z}_t|\mathbf{x}_t) = \mathcal{N}(\mu_\varphi(\mathbf{x}_t), \mathbf{\Sigma}_\varphi(\mathbf{x}_t))$, the log term inside the expectation means that we want the output of transition network to be close to the mean of its distribution, up to some constant.

$$\log p(\mathbf{z}_t|\mathbf{x}_t) = -\frac{1}{2}\Big(\log\left((2\pi e)^{n_z}|\mathbf{\Sigma}_\varphi(\mathbf{x}_t)|\right) + (\mathbf{z}_t - \mu_\varphi(\mathbf{x}_t))^\top \mathbf{\Sigma}_\varphi(\mathbf{x}_t)^{-1}(\mathbf{z}_t - \mu_\varphi(\mathbf{x}_t))\Big) \tag{A.7}$$

A.2 Implementation

Transition model structure: $\mathbf{x}_t$ goes through one hidden layer with ℓ_1 units and $\hat{\mathbf{z}}_{t+1}$ goes through one hidden layer with ℓ_2 units. The outputs of the two hidden layers are concatenated and go through a network with two hidden layers of size ℓ_3 and ℓ_4,

respectively, to build μ_φ and $\mathbf{\Sigma}_\varphi$. $\bar{\mathbf{z}}_t$ is sampled from this distribution and is concatenated by the action. The result goes through a three-layer network with ℓ_5, ℓ_6, and ℓ_7 units to build $\mathbf{M}_t$, $\mathbf{B}_t$, and $\mathbf{c}_t$.

In the following we will specify the values for ℓ_i's for each of the four tasks used in our experiments.

A.2.1 Planar system

Input: 40×40 images (1600 dimensions). 2-dimensional actions. 5000 training samples of the form $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$

Latent space: 2-dimensional

Encoder: 3 Layers: 300 units- 300 units- 4 units (2 for mean and 2 for the variance of the Gaussian distribution)

Decoder: 3 Layers: 300 units- 300 units- 1600 units

Transition: $\ell_1 = 100$- $\ell_2 = 5$- $\ell_3 = 100$- $\ell_4 = 4$- $\ell_5 = 20$- $\ell_6 = 20$- $\ell_7 = 10$

Number of control actions: or the planning horizon $T = 40$

A.2.2 Inverted Pendulum

Input: Two 48×48 images (4608 dimensions). 1-dimensional actions. 5000 training samples of the form $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$

Latent space: 3-dimensional

Encoder: 3 Layers: 500 units- 500 units- 6 units (3 for mean and 3 for the variance of the Gaussian distribution)

Decoder: 3 Layers: 500 units- 500 units- 4608 units

Transition: $\ell_1 = 200$- $\ell_2 = 10$- $\ell_3 = 200$- $\ell_4 = 6$- $\ell_5 = 30$- $\ell_6 = 30$- $\ell_7 = 12$

Number of control actions: or the planning horizon $T = 100$

A.2.3 Cart-pole Balancing

Input: Two 80×80 images (12800 dimensions). 1-dimensional actions. 15000 training samples of the form $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$

Latent space: 8-dimensional

Encoder: 6 Layers: convolutional layer: $32 \times 5 \times 5$; stride (1,1) - convolutional layer: $32 \times 5 \times 5$; stride (2,2) - convolutional layer: $32 \times 5 \times 5$; stride (2,2) -convolutional layer: $10 \times 5 \times 5$; stride (2,2) - 200 units- 16 units (8 for mean and 8 for the variance of the Gaussian distribution)

Decoder: 6 Layers: 200 units- 1000 units- convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)- convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)- convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)- convolutional layer: $2 \times 5 \times 5$; stride (1,1)

Transition: $\ell_1 = 300$- $\ell_2 = 10$- $\ell_3 = 300$- $\ell_4 = 16$- $\ell_5 = 40$- $\ell_6 = 40$- $\ell_7 = 32$

Number of control actions: or the planning horizon $T = 100$

A.2.4 Three-Link Robot Arm

Input: Two 128×128 images (32768 dimensions). 3-dimensional actions. 30000 training samples of the form $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$

Latent space: 8-dimensional

Encoder: 6 Layers: convolutional layer: $64 \times 5 \times 5$; stride (1,1) - convolutional layer: $32 \times 5 \times 5$; stride (2,2) - convolutional layer: $32 \times 5 \times 5$; stride (2,2) -convolutional layer: $10 \times 5 \times 5$; stride (2,2) - 500 units- 16 units (8 for mean and 8 for the variance of the Gaussian distribution)

Decoder: 6 Layers: 500 units- 2560 units- convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)- convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)-

convolutional layer: $32 \times 5 \times 5$; stride (1,1)- Upsampling (2,2)- convolutional layer: $2 \times 5 \times 5$; stride (1,1)

Transition: $\ell_1 = 400$- $\ell_2 = 10$- $\ell_3 = 400$- $\ell_4 = 6$- $\ell_5 = 40$- $\ell_6 = 40$- $\ell_7 = 48$

Number of control actions: or the planning horizon $T = 100$

Appendix B

Prediction by Anticipation

B.1 Terms in the loss function

We re-write the loss function in Eq. 5.6 here:

$$\mathcal{L}_t = \underbrace{\mathcal{D}(\mathbf{j}_{t+1}^{env}, \hat{\mathbf{j}}_{t+1}^{env}) + \lambda\big(1 - \text{SSIM}(\mathbf{j}_{t+1}^{env}, \hat{\mathbf{j}}_{t+1}^{env})\big)}_{\mathcal{L}_t^{\text{rec.}}} + \underbrace{\text{KL}\big(q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego}, \mathbf{j}_{t+1}^{env})||p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego})\big)}_{\mathcal{L}_t^{\text{KL}}}.$$

B.1.1 The ELBO term

Minimizing the first and last terms is equivalent to maximizing the ELBO in Eq. 5.5. Considering $q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego}, \mathbf{j}_{t+1}^{env}) = \mathcal{N}(\mu_\varphi, \Sigma_\varphi)$ and $p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego}) = \mathcal{N}(\mu_\psi, \Sigma_\psi)$ as Gaussian distributions, the KL term is simply differentiable with the following terms:

$$\begin{aligned}&\text{KL}\big(q_\varphi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego}, \mathbf{j}_{t+1}^{env}))||p_\psi(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{j}_{t+1}^{ego})\big) \\ &\quad = \frac{1}{2}\Big(\text{Tr}\big({\Sigma_\psi}^{-1}\Sigma_\varphi\big) + \big(\mu_\psi - \mu_\varphi\big)^\top {\Sigma_\psi}^{-1}\big(\mu_\psi - \mu_\varphi\big) + \log(\frac{|\Sigma_\psi|}{|\Sigma_\varphi|}) - d\Big),\end{aligned}$$

where d is the dimensionality of the stochastic latent code.

B.2 Implementation details of the prediction module

Figure B.1 shows a detailed implementation of the prediction module. The structure of the networks based on this figure are explained below. Convolutional neural networks, denoted by CNN, have convolutional layers as their core but can also include one or two fully-connected layers. MLP networks only contain fully-connected layers.

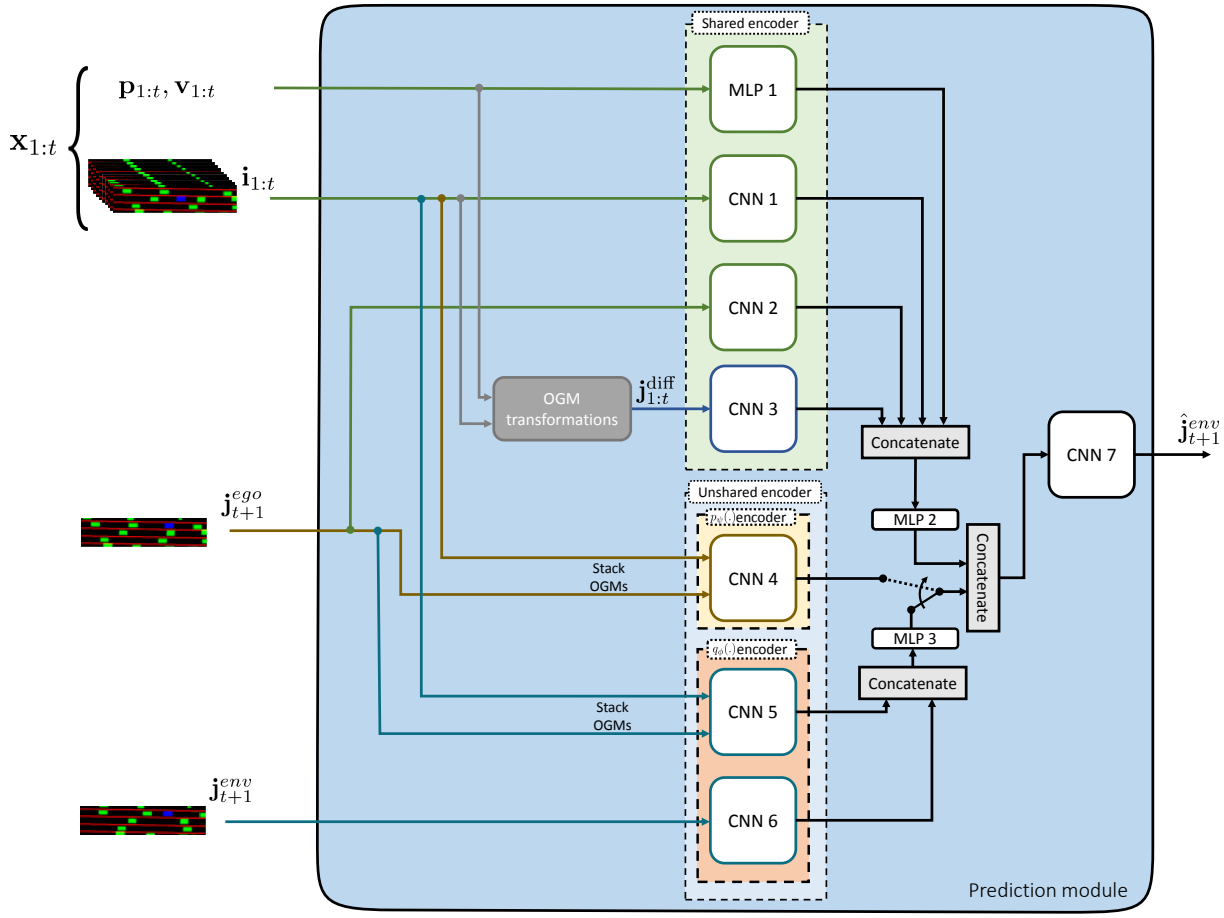


Figure B.1: Detailed structure of the prediction module.

B.2.1 NGSIM I-80 dataset

For the NGSIM I-80 dataset, we use the actions extracted by the authors in [39]. As stated in the experiment section, the images are $117 \times 24 \times 3$. For the IOTs we zero pad the images before feeding them to the modules with the padding size of 20 in each dimension and remove the padding from the output image after processing.

Network	Detail of structure	
MLP 1	2 layers	1. 256 units- leaky ReLU activation 2. 25 units- ReLU activation
CNN 1	4 layers	1. 64 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 128 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 768 units, ReLU activation
CNN 2	4 layers	1. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 768 units, ReLU activation
CNN 3	4 layers	1. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 32 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 32 units with linear activation as the mean, $\mu(\mathbf{j}^{\text{diff}}_{1:t})$.
CNN 4	5 layers	1. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 16 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 768 units, linear activation 5. Two branches of fully-connected layers with 32 units each, as the mean and variance , μ_ψ and Σ_ψ, with linear activation
CNN 5	4 layers	1. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation

		3. 16 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 768 units, linear activation
CNN 6	4 layers	1. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 16 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 4. Fully-connected layer with 768 units, linear activation
MLP 2	1 layer	1 .256 units, leaky ReLU activation
MLP 3	1 layer	1. Two branches of fully-connected layers with 32 units each, as the mean and variance , μ_φ and Σ_φ, with linear activation
CNN 7	5 layers	1. Fully-connected layer with 768 units, leaky ReLU activation 2. Fully-connected layer with 6144 units, leaky ReLU activation. Followed by reshaping to a $(12, 2, 256)$ tensor 3. Deconv. layer with 128 kernels of size 5×3, stride size = 2, leaky ReLU activation 4. Deconv layer with 64 kernels of size 6×4, stride size = 2, leaky ReLU activation 5. Deconv layer with 3 kernels of size 3×2, stride size = 2, sigmoid activation

Table B.1: Detail of the prediction module for the NGSIM I-80 dataset. The coefficient for the leaky ReLU activation functions is 0.2.

For difference learning model, we use the exact same structure of the networks. However, since the output of the module is the difference between two consecutive frames, it can get any values between -1 and 1. Therefore, we use *tanh* activation for the last layer of network CNN 7.

B.2.2 Argoverse dataset

The images in this dataset are $256 \times 256 \times 2$. They have two channels. One channel for OGM and the other one for the ego-vehicle. The ego-vehicle is more towards the left side of the image. Therefore we first zero pad the left side of the image so that the ego-vehicle is centered and then zero pad the whole image to apply the functions in the IOTs. To extract the actions, we apply inverse of the functions in Fig. 5.1 to compute the actions using the xy coordinate of the car at each time. Table B.2 shows the details of the difference learning model, i.e. the model with the best performance for this dataset.

Network	Detail of structure	
MLP 1	2 layers	1. 85 units- leaky ReLU activation 2. 25 units- ReLU activation
CNN 1	6 layers	1. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 32 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 64 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 128 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 128 units, ReLU activation
CNN 2	6 layers	1. 2 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 128 units, ReLU activation
CNN 3	6 layers	1. 2 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 32 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 32 units with linear activation as the mean , $\mu(\mathbf{j}^{\text{diff}}_{1:t})$.

CNN 4	7 layers	1. 2 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 128 units, linear activation 7. Two branches of fully-connected layers with 32 units each, as the mean and variance , μ_ψ and Σ_ψ, with linear activation
CNN 5	6 layers	1. 2 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 128 units, linear activation
CNN 6	6 layers	1. 2 kernels of size 4×4, stride size = 2, leaky ReLU activation 2. 4 kernels of size 4×4, stride size = 2, leaky ReLU activation 3. 8 kernels of size 4×4, stride size = 2, leaky ReLU activation 4. 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 5. 256 kernels of size 4×4, stride size = 2, ReLU activation. Followed by flattening 6. Fully-connected layer with 128 units, linear activation
MLP 2	1 layer	1 .256 units, leaky ReLU activation
MLP 3	1 layer	1. Two branches of fully-connected layers with 32 units each, as the mean and variance , μ_φ and Σ_φ, with linear activation
CNN 7	7 layers	1. Fully-connected layer with 128 units, leaky ReLU activation 2. Fully-connected layer with 9216 units, leaky ReLU activation. Followed by reshaping to a $(6, 6, 256)$ 3. Deconv. layer with 128 kernels of size 4×4, stride size = 2, leaky ReLU activation. 4. Deconv layer with 64 kernels of size 4×4, stride size = 2, leaky ReLU activation

		5. Deconv. layer with 32 kernels of size 4×4, stride size = 2, leaky ReLU activation. 6. Deconv layer with 16 kernels of size 4×4, stride size = 2, leaky ReLU activation 7. Deconv layer with 2 kernels of size 3×2, stride size = 2, tanh activation

Table B.2: Detail of the prediction module for the Argoverse dataset as a part of difference learning module. The coefficient for the leaky ReLU activation functions is 0.2.