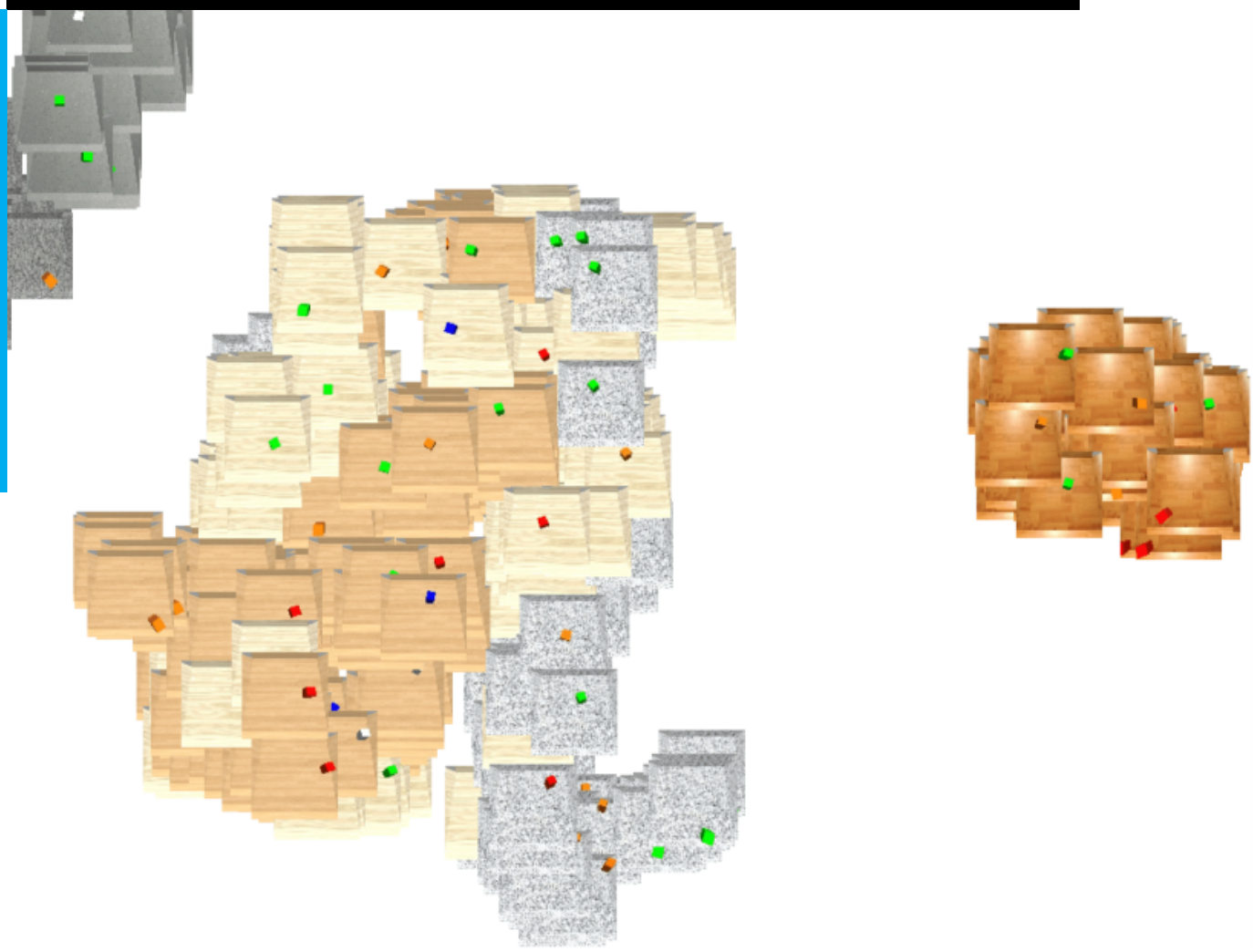
Learning State Representations for Robotic Control

Information Disentangling and Multi-modal Learning

Wuyang Duan

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Learning State Representations for Robotic Control

Information Disentangling and Multi-modal Learning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Wuyang Duan

October 23, 2017

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Copyright © Delft Center for Systems and Control (DCSC)
All rights reserved.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Learning State Representations for Robotic Control

by

Wuyang Duan

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: October 23, 2017

Supervisor(s):

Dr.-Ing. J. Kober

Reader(s):

Prof.dr.ir. M. Wisse

Dr.-Ing. J. Kober

Dr. G.C.H.E. de Croon

Ir. W.J. Wolfslag

Abstract

Representation learning is a central topic in the field of deep learning. It aims at extracting useful state representations directly from raw data. In deep learning, state representations are usually used for classification or inferences. For example, image embedding that provides similarity metrics can be used for face recognition. Recent success on deep learning has stimulated the interest of applying state representation learning to control. This problem is very different from deep learning in computer visions and language modelling. Control tasks usually require pose information about the task relevant object in the scene. Applying state representation learning to control requires such features to be embedded. This is a difficult problem since there is generally no explicit supervision for extracting pose information from the data.

A problem of state representation learning for control is that raw data can contain much irrelevant information for the control task. For example, colors, textures and shapes of task-relevant objects are not as important as poses of objects. This suggests that the appearance and pose information should be disentangled in learned representations and the pose representations should be used for the control task. Furthermore, we usually need a system dynamic model in order to perform model-based optimal control. The prediction ability of the model is crucial for planning since it has to evaluate future trajectories and optimizes the actions. Thus we need to learn a decent dynamic model on extracted representations.

To address these problems, we first propose to use Variational Auto-encoders to disentangle the pose and appearance representations. The benefit of preserving both appearance and pose representations is that we can predict pose representations conditioned on actions. From appearance and predicted pose representations, we can reconstruct and predict future image frames. For this purpose, we introduce the Long Short-Term Memory network to learn a prediction model on the pose representations. We further intend to apply multi-modal state representation learning for control. Leveraging the idea of sensor fusions, we want to verify whether multi-modal sensory data lead to better representations for control.

To test our hypothesis, an object poking simulation is prepared in the Gazebo simulator where training and testing data-sets are collected. We validate disentangled representations and prediction performances with a few baseline models. Experiment results show that information disentangling can be achieved without explicit supervisions. The prediction model can

effectively predict future frames a few steps ahead conditioned on poking actions. For multi-modal learning, we apply a Siamese network for inverse planning on the object poking task. We test multi-modal representations with the Siamese network and report improvements in online simulations.

Keywords

State representation learning, information disentangling, multi-modal learning, model-based control.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Dr.-Ing. Jens Kober for his guidance during the past ten months. His suggestions and questions have been constantly inspiring me to formulate and approach the problem differently. He has offered enough resources to conduct the research. He has always been responsible and encouraging even when I am frustrated. His detailed feedback on my thesis are valuable to me beyond researches.

I want to thank the committee members for evaluating this thesis. For experiments in this project, I am thankful for the GPU donation from Nvidia. I also want to thank Mr. John Stals. I will never finish this master without his help. I want to thank my grandfather who has always been there for me.

Finally, I want to express my gratitude to my friends here. I will not forget this two years of study.

Delft, University of Technology Wuyang Duan
October 23, 2017

Chapter 1

Introduction

Robotic control is a comprehensive and in-depth field. Due to its numerous and diversified application scenarios, the control strategy for different robots are usually tailored and highly customized. With specially designed control strategies, we want the robot to perceive the world, make decisions and interact with the environment. This is a complex and systematic problem. One of the most fundamental problem here is the state estimation. We want to efficiently find a representation of the environment as well as the internal status of the robot, which is important for any decision making problems. Nowadays we have collaborative robot arms and automatic aerial and ground vehicles. None of them can circumvent the difficulty of state representations.

1-1 State representations in deep learning

States that are crucial for control tasks are not always measurable. From an optimal control point of view, a typical approach is to estimate the state from observations. A milestone of this approach is the invention of Kalman filter [6]. The Kalman filter provides an optimal framework for state estimations and rigorously defines conditions of observable states. Many improvements have been made in the field of optimal control since then. The development of new sensors has greatly promoted the research of sensor fusions. Visual based approaches have also been proposed to directly estimate the state from a 3D geometry perspective [7]. These approaches usually require considerable amount of hand engineering and feature designs. With the growing need of reliable and efficient automatic robot control, the state representation problem has to be solved better.

In the meantime, applying machine learning approaches to state estimations is not new. In the machine learning world, a compact description of the data is usually referred to as a feature or a representation. This is the equivalent of states in optimal control. The Kalman filter is in essence a Bayesian filter. Bayesian inference type of methods are proved to be powerful in filtering problems. The neural network based reinforcement learning has been proposed a few years ago to play games from raw observations [8]. The benefit of applying deep learning

approaches to control is that they directly work on raw observation data. The powerful feature extraction and function approximation abilities of deep neural networks undermine the necessity of feature designs and representation extractions. Instead, it only needs to be trained on a large data-set where many scenarios are shown to the network. The network will extract the most relevant features for decision makings depending on the training objective and hopefully generalize to unseen situations.

Recently, we have seen end-to-end approaches achieving very impressive results [9] on robotic tasks. Applying end-to-end deep learning methods on robotic manipulation tasks is now a trending topic. While they achieve impressive performances on trained tasks, they seldomly generalize to unseen tasks. Generally a network has to be trained for each different task or every group of similar tasks. Although it succeeds at one task, the representation it has learned does not transfer to another task in general. Thus the prevalence of end to end methods does not make the representation learning problem less important.

1-1-1 Problems in representation learning for control

When end to end methods are applied on a certain task, a representation of the relevant data can be retrieved from the network. However, we usually need a suitable state representation before solving the task in many scenarios. In robotic settings, we actually want to extract the state with the least human intervention and in a unsupervised manner. In all, an unsupervised state representation learning approach is very useful because:

- It extracts representations automatically without feature designs leveraging deep networks.
- It does not require labeled data which is expensive in robotic settings.
- It is flexible for more general and broader tasks comparing to end to end supervised learning that hardly generalizes.

Ideally, such an approach should be able to infer the control relevant information (e.g., positions) in a self-supervised or unsupervised fashion. This problem is less studied comparing to general supervised learning problem like imitations and classifications. Therefore, we choose to attack the problem from an unsupervised perspective.

Another problem of representation learning for control is that the state of the environment should be observed from relevant data modalities. For instance, a robot arm might not know the exact position of a box that is to be pushed, but a camera mounted on the arm provides enough information for the arm to infer the relative position with respect to its own frame. In many other daily manipulation tasks like gripping, flipping and poking, it is reasonable to assume that such tasks can be achieved from information contained in image data. Moreover, depth and tactile data usually cover different aspects of the task that are helpful for control. Towards a more widely applicable state representation learning method, we choose to learn state representations purely based on visions. We also want to examine whether multi-modal sensory data are useful for control.

Learning from visions subjects to distractions and task irrelevant information. In many cases, the appearance information about the scene is not useful for the control tasks. For a pole

balancing task, the texture and color of the pole do not change the way that optimal actions are derived. Task relevant information like the angle of a pole in a balancing task and the pose of a box in a manipulation task should be separated and extracted from the data and used for control. Based on this motivation, we further propose to disentangle the representation of images into the pose and the appearance parts.

The fourth problem of robotic control is the ability of predictions into the future. For many model-based control methods like MPC and iLQG [10], accurate predictions of future representations are crucial. Multiple steps ahead predictions allow for a proper cost evaluation within a certain horizon. This is a necessary procedure for optimizations in many residing horizon control or reinforcement learning type of methods. In our case, we need to learn a prediction model from images without supervisions. Taking robot pushing as an example, multiple step predictions are useful for finding an optimal pushing trajectory and correcting from mistakes and model uncertainties. However, the prediction of pixels is generally considered as a challenging problem. An alternative is to learn a dynamic model in the latent space. Then the prediction in the data space can be used as a training cue in a self-supervision manner. Towards model-based optimal control, we propose to learn a multiple step ahead prediction model in the latent space.

1-2 Research questions

To summarize our motivations in the last section, we study the following questions in this thesis.

- Is it possible to learn useful representations from visions for control in a unsupervised manner?
- Does multi-modal sensory data generate a better representation for control?
- Is it possible to specifically disentangle the pose information from robot interaction images and use that for planning?
- Is it possible to learn the aforementioned representations with a multiple step ahead prediction model?

To further clarify the second question, we are interested in whether the classical sensor fusion idea would also work in deep learning for better classification or decision makings. Utilizing data like depth and tactile can possibly contributes to a more complete representation for tasks like object manipulations. The third question is our main focus. Representations from neural networks are entangled and not tangible to human. We want to examine whether it is possible to disentangle appearance and pose relevant information from images without any explicit supervisions. Studying this problem also provides a new perspective for human to understand the representations of deep networks given their black box nature.

To explore answers to these questions, we propose our own solutions and validate them in a elementary control task. We prepare an object poking task in the simulation that is used as a central test case through this thesis. From the object poking simulation, we collect a data-set for representation learning and validate our proposed solutions.

1-3 Outline

The content of this thesis is organized in the following chapters.

In **Chapter 2**, several unsupervised learning models that are directly relevant to our work are introduced.

In **Chapter 3**, we introduce the multi-modal state representation learning and incorporate it with the Siamese network.

In **Chapter 4**, we explain our proposed Long Short Term Memory based Variational Auto-encoders. We use the model to learn disentangled state representations and predict future images.

In **Chapter 5**, we briefly introduce a object poking data-set generated from Gazebo simulations and give the motivation of preparing this data-set.

In **Chapter 6**, we first report the result of multi-modal inverse planning in object poking simulations. Subsequently, several baseline models for image predictions are introduced. We validate all models on the test set and compare their performances. Furthermore, we examine whether learned representations are disentangled.

In **Chapter 7**, a conclusion is drawn from experiment results. Possible directions for future improvements are outlined.

More experiment details and results are attached in Appendices A and B.

Chapter 2

Background

Unsupervised learning is generally considered a harder task than supervised learning. For supervised learning, very impressive results on image classification have been achieved in the past few years [11]. With large data-set and powerful distributed computation resources, deep networks can usually model the labeled data-set very well. However, learning concepts and semantics from huge amount of labels is not entirely biologically plausible. Human can usually learn similarities, discriminate and locate objects by looking at a few images and quickly generalize to other similar images. Therefore learning representations in a semi-supervised or unsupervised way is still an open question. In robotic settings, this is especially a problem since labeled data are usually expensive. Efficient and automatic representation learning from unlabeled data is of great necessity comparing to hand engineering for every single problem.

Typical data like natural images, videos and audios are usually high dimensional and generated from very complicated distributions. It is hard to model them directly. In case that labels are expensive for such data, the auto-encoder is a good choice to extract representations. The research on auto-encoders has been part of the historical landscape of state representation learning. In general, they are used to extract representations with lower dimensions (or under-complete) of the input data. There are many variants of auto-encoders with different regularization and different probabilistic interpretations. In this chapter, a few different auto-encoders that are directly used in our work will be introduced. In Section 2-2, a few deterministic auto-encoders are explained with a specific consideration on how they capture the data manifold. A probabilistic auto-encoder is also reviewed in Section 2-3. Afterwards, we briefly explain how the convolutional and recurrent network structures can be applied in auto-encoders. Finally, a discussion on auto-encoders in control, auto-encoders with multi-modal learning and a conclusion are given.

2-1 Auto-encoders

Auto-encoders are a type of feedforward networks which reconstruct their inputs [12]. The simplest auto-encoder has only one hidden layer as shown in Figure 2-1. Auto-encoders learn

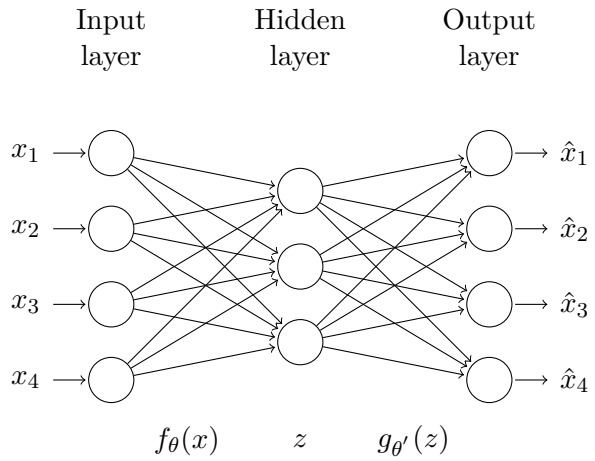


Figure 2-1: An auto-encoder with a hidden dimensionality of 3

to reconstruct inputs from the under-complete representations in the hidden layers. This process forces them to discard irrelevant information in the representations. The learning objective is to minimize the reconstruction loss as follow:

$$\mathcal{L}(x, \hat{x}) = \mathcal{L}(x, g_{\theta'}(f_\theta(x))), \tag{2-1}$$

The encoding function f_θ parameterized by θ maps the input into a representation z and the decoding function g parameterized by $\theta^{'}$ maps the representation back to the input. The parameter θ and $\theta^{'}$ can be optimized by minimizing the reconstruction loss. This is usually done by back-propagation [13]. We can denote the input data, representations (or latent variables) and reconstructions as random variables X, Z and $\hat{X}$. The reconstructed data $\hat{x}$ is usually not interpreted as the exact reconstruction, but parameterizing the distribution $p(X|\hat{X} = \hat{x})$. The loss in Equation (2-1) is then formulated as:

$$\mathcal{L}(x, g(f(x))) = \mathbb{E}_{q(X)}\left[-\log p(X|\hat{X})\right]. \tag{2-2}$$

Here $q(X)$ denotes the unknown data distribution and $p(X|\hat{X})$ denotes the distribution of X conditioned on the reconstruction $\hat{X}$. This is equivalent to minimizing the negative log likelihood $-\log p(X|Z)$ since $\hat{X}$ is generated by Z. From the information perspective, the expectation term in Equation (2-2) is actually a lower bound of the mutual information between X and Z [14]. Thus minimizing the loss pushes the lower bound and maximizes the mutual information between the representation and the data distribution.

The actual form of the loss depends on the data to be modelled. If the data distribution X is real valued and assumed to be Gaussian, the reconstruction $\hat{x}$ parameterizes the mean of Gaussian distributed X (e.g., real valued image pixels). In this case, the negative log likelihood is equivalent to a squared error loss. For binary data like MNIST image pixels, they are usually modelled by multivariate Bernoulli distributions. Then the reconstruction error is the cross entropy between X and $\hat{X}$, which can be further decomposed into a Kullback-Leibler divergence between the distribution of X and $\hat{X}$ and an unknown constant entropy of X. The KL divergence D_{KL} between two distributions P and Q is defined as:

$$D_{KL}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{2-3}$$

It is also called the relative entropy which measures the amount of information lost when distribution Q is used to approximate P. Minimizing the KL divergence between Bernoulli distributed X and $\hat{X}$ pulls two distributions close to each other.

Auto-encoders are the basis of our work. We use auto-encoders to extract representations without supervisions. Moreover, the reconstruction ability from decoders will be useful if we want to make predictions in the data space. Unfortunately, plain auto-encoders without extra improvements do not work very well in practice. In the next section, two types of regularized auto-encoders are reviewed.

2-2 Auto-encoders with regularization

Although theoretically plausible, auto-encoders need to be regularized in practice. If an auto-encoder is allowed to have more neurons than its input size (over-complete) in its hidden layer, it can learn to copy the input even with linear activation functions [1]. An auto-encoder without regularization can learn arbitrary representations that are not useful. Adding regularization however expresses our prior knowledge about the data distribution or the preferred behavior of the model. It usually helps the model to generalize better to unseen data during the test phase. For example, dropout of neurons [15] helps to prevent over-fittings of the training data-set and batch normalization [16] improves the training stability and test accuracy. The following sections will give overviews of several commonly applied regularized auto-encoders.

2-2-1 Denoising auto-encoders

The denoising auto-encoder (DAE) reconstructs the original input data from artificially corrupted data. DAE has to capture the most important features of the data in order to recover from the corruption. By reconstructing from the corrupted data, DAE learns robust representations without any extra penalties [17]. The reconstruction of DAE actually matches the score of the data distribution implicitly.

The corruption process can be described by a conditional distribution $C(\tilde{X}|X)$. The corrupted random input variable can be denoted as $\tilde{X}$. Usually a Gaussian corruption process is applied on the input data. So $\tilde{x}$ can be sampled from $C(\tilde{X}|X = x)$ given a input point x. The reconstruction distribution $p(X|\hat{X})$ can also be written as $p(X|Z = f(\hat{X}))$. Using the same notation as the last section, the loss function for DAE is:

$$\mathcal{L}(x, g(f(\tilde{x}))) = -\mathbb{E}_{x\sim q(X)}\mathbb{E}_{\tilde{x}\sim C(\tilde{X}|X)} \log p(X|Z = f(\tilde{X}))\,, \tag{2-4}$$

The expectation over the data distribution q can be estimated by averaging over the training samples. Apart from the corruption process, there is no difference from plain auto-encoders. Although no explicit regularization is included, the denoising auto-encoder learns the manifold of the data distribution [17]. This can be visualized in Figure 2-2. Suppose that the high dimensional data distribution which we are trying to model concentrate on a low dimensional manifold, the manifold is marked as the black curve in the data space. The corruption process takes original samples away from the manifold, meaning that such corrupted data points will have low probabilities under the original data distribution. DAE thus learns to map low

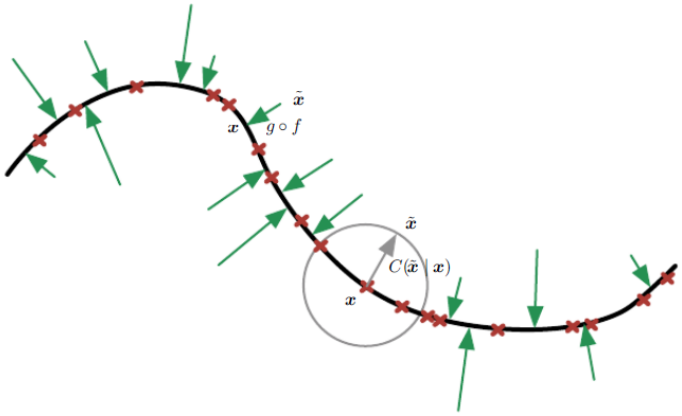


Figure 2-2: DAE learns to map the corrupted data points back to the manifold marked as green arrows. The corruption is marked as a gray circle [1].

probability $\hat{x}$ back to high probability x on the manifold, which is the same as maximizing the likelihood of $p(X|\hat{X})$. If DAE is under-complete, the representation actually captures the main variation directions in the data distribution. DAE has a very close connection to the score matching method [18], which is an alternative of maximum likelihood estimation methods. The score matching method estimates a probability density model $p_{SM}(x;\theta)$ of the data distribution. The gradient $\frac{\partial \log p_{SM}}{\partial x}$ is referred to as the score function $\psi(x;\theta)$. The model parameters θ are estimated by matching the score of true data distribution $q(x)$ with $\psi(x;\theta)$. A score match function is introduced to train DAE in [19]. The score matching loss looks like:

$$\mathbb{E}_{p_\sigma(X,\tilde{X})}\left[\frac{1}{2}|\psi(\tilde{X};\theta) - \frac{\partial \log C(\tilde{X}|X)}{\partial \tilde{X}}|^2\right], \tag{2-5}$$

The distribution $p_\sigma(X,\tilde{X}) = C(\tilde{X}|X)q_0(x)$ is the estimated joint distribution where q_0 is the empirical density function based on available training data. If a Gaussian corruption process is applied, the second term is:

$$\frac{\partial \log C(\tilde{X}|X)}{\partial \tilde{X}} = \frac{1}{\sigma^2}(X - \hat{X}) \tag{2-6}$$

where σ^2 is the variance of the corruption. Note that the score function term points from the corrupted variable back to the true variable, which should be the score of the estimated density function. In other word, DAE trained with such score matching criterion learns the score of the original data distribution.

It is further proved in [20] that the optimal reconstruction of DAE recover the score of the data distribution. We have:

$$\frac{\partial \log q(x)}{\partial x} = \frac{1}{\sigma^2}(r^*_\sigma(x) - x) + o(1) \quad \text{as} \quad \sigma \to 0\,. \tag{2-7}$$

where r^*_σ is the optimal reconstruction function. This relation holds when the noise variance approaches zero and training samples are infinite. Although without any explicit regularization, DAE matches the score of the data distribution.

2-2-2 Contractive auto-encoders

The contractive auto-encoder (CAE) has an explicit regularization term in its loss function which penalizes the Jacobian of the representation z with respect to the input x. By penalizing the Jacobian, the representation is trained to be less sensitive to the input variations. To examine the contraction of the Jacobian, bigger singular values of the Jacobian correspond to main directions of variations of the training data and the density drops slow. The contrary holds for smaller singular values. Intuitively, they correspond to tangent and orthogonal directions of the data manifold.

It is actually proved in [20] that DAE with small corruption noises and the squared error reconstruction loss is a special case of CAE. Although regularized differently, both DAE and CAE estimate the score to learn the data manifold. DAE matches the reconstruction with the score of the data distribution and CAE discovers the strong and weak directions of contractions. Both of them learn useful representations by counterbalancing the reconstruction and regularization. On the one hand, minimizing the reconstruction loss forces the representation to be sensitive along the data manifold in the tangent direction. On the other hand, minimizing the regularization loss forces the representation to be less sensitive along the manifold in the orthogonal direction.

DAE and CAE can usually learn the data manifold in practice. However, they learn deterministic representations from data points. They are not applicable for our purpose since we want to learn stochastic representations from the data distribution. Instead, we should use probabilistic auto-encoders as will be introduce in the next section. In our work, we apply DAE as baseline models for comparisons.

2-3 Variational auto-encoders

The variational auto-encoder (VAE) is inherently different from aforementioned auto-encoders. It is a type of generative model which approximates the true posterior of the hidden representation. After training, they are able to generate fake data which conform to the data distribution. Unlike deterministic auto-encoders, VAE maximize a variational lower bound of the data likelihood and learn an useful distribution of the hidden representation without extra regularization.

A VAE is a directed graph model [21]. Using the same X and Z notations as previous sections, the approximated posterior is modelled by a recognition model $q_\phi(Z|X)$ as a probabilistic encoder and the data generation is modeled by a generative model $p_\theta(X|Z)$ as a probabilistic decoder. A variational lower bound and a re-parameterization trick are introduce to enable direct posterior inference and gradient back-propagation. Suppose that the data distribution is $p_\theta(X)$, the log likelihood can be decomposed as:

$$\log p_\theta(X) = D_{KL}(q_\phi(Z|X)||p_\theta(Z|X)) + \mathcal{L}(\theta, \phi; X)\,.$$

The first term is the divergence between the approximated posterior $q_\phi(Z|X)$ and the unknown posterior $p_\theta(Z|X)$. The second term is the variational lower bound of the data likeli-

hood, which can be further decomposed as:

$$\mathcal{L}(\theta, \phi; X) = \mathbb{E}_{q_\phi(Z|X)}[-\log q_\phi(Z|X) + \log p_\theta(X, Z)] \tag{2-8}$$

$$= \mathbb{E}_{q_\phi(Z|X)}[\log p_\theta(X|Z)] - D_{KL}(q_\phi(Z|X)||p_\theta(Z))\,, \tag{2-9}$$

By maximizing the lower bound $\mathcal{L}$, the recognition model $q_\phi(Z|X)$ approximates the true posterior $p_\theta(Z|X)$ which also maximizes the data likelihood. The formulation of $\mathcal{L}$ is intuitive. Equation (2-8) is similar to Expectation Maximization methods. Maximizing the entropy of q_ϕ encourages the approximated posterior to have high variances over all possible representations z. The first term in Equation (2-9) is a reconstruction term just like in usual auto-encoders. The second term expresses our prior knowledge about the latent distribution. The approximated posterior is pulled close to the prior. In order to differentiate the lower bound $\mathcal{L}$ and back-propagate, we need to apply re-parameterization on the representation Z.

2-3-1 Re-parameterization

The hidden representation z can be sampled from the posterior $q_\phi(Z|X)$. A graph of VAE is given in Figure 2-3. Re-parameterization pushes the stochasticity of the latent random

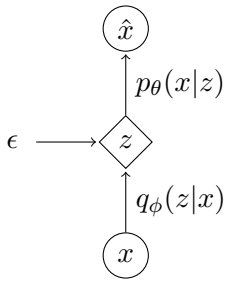


Figure 2-3: The directed model of a VAE with a recognition model (encoder) and a generative model (decoder). The diamond block shows that the parameters of distribution Z are outputs from encoding X. If the representation distribution is Gaussian, parameters are means and variances.

variable Z into a standard random variable ϵ which makes the generation of samples z easy. Now $z \sim q_\phi(Z|X)$ is replaced by:

$$z = g_\phi(\epsilon, x) \quad \text{with} \quad \epsilon \sim p(\epsilon)\,. \tag{2-10}$$

Based on different types of latent distributions, the deterministic function g_ϕ can be chosen accordingly. Since the cumulative probability on differential areas must be invariant under the change of variables, an integral of some function f_0 on Z can be transformed as:

$$\int q_\phi(z|x) f_0(z)\, dz = \int p(\epsilon) f_0(z)\, d\epsilon \simeq \frac{1}{L} \sum_{l=1}^{L} f_0(g_\phi(x, \epsilon^{(l)}))\,. \tag{2-11}$$

This is the proposed Stochastic Gradient Variational Bayers (SGVB) estimator, which can be used to estimate the variational lower bound $\mathcal{L}$. The second term in Equation (2-9) usually has closed-form solution. With a training data point $x^{(i)}$, the first reconstruction loss can be approximated using SGVB with L random samples of the latent variable:

$$\mathbb{E}_{q_\phi(Z|x^{(i)})}[\log p_\theta(x^{(i)}|z)] = \frac{1}{L} \sum_{l=1}^{L} \log p_\theta(x^{(i)}|z^{(i,l)})\,. \tag{2-12}$$

If $q_\phi(Z|X)$ is assumed to be Gaussian, the re-parameterization function is given as:

$$z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \epsilon^{(l)} \quad \text{with} \quad \epsilon \sim \mathcal{N}(0, I)\,. \tag{2-13}$$

where $\sigma^{(i)}$ and $\epsilon^{(l)}$ are multiplied element-wisely. For the recognition model, μ and σ here are outputs of the encoder, e.g., activation from the last layer of the encoder network. Thus, the variational loss $\mathcal{L}$ estimated by SGVB is differentiable. It is found out that an L (number of samples per training data point) of value one works well in practice. We only need to sample from the approximated posterior, perform stochastic gradient descent and update ϕ and θ jointly. The benefits of applying SGVB and re-parameterization are:

- Unbiased and differentiable estimator of the variational lower bound.
- Gradient back-propagation based on samples of Z.

In practice, the quality of learned representations is directly affected by the choice of the prior. Moreover, it is unclear how well VAE wil perform in control tasks. VAE learns a stationary distribution of the data-set, while the state representation in control tasks are usually not. Thus introducing a transition model in the latent space of VAE is necessary. The transition model can be approximated by recurrent structures in auto-encoders.

Auto-encoders take in pixel vectors of images and ignore 2D structures. Convolution operations on images however preserve local information of images by reusing filters across images. For our purpose of disentangling appearance and pose representations, it is crucial to use convolutional structures so that locations and appearances are preserved in convoluted feature maps. Specifically, we utilize convolutional VAE in our work. In the next section, we introduce convolutional auto-encoders with a focus on the decoder part.

2-4 Convolutional auto-encoders

Convolutional neural networks (CNNs) are inspired by animal visual cortices. Individual cortical neuron of cats is found to only respond to stimuli in a limited region on the visual field. These sub-regions are called local receptive fields which are core components of CNNs. The location and orientation selective biological mechanism can be mimicked by convolution operations in CNNs [22].

In CNNs, local stimuli is processed by the convolution between filters and image patches. Each filter strides over the visual field and produces a feature map. Feature maps from shallow layers are then processed by filters from deeper layers. Usually, filters from shallow layers can extract simple features like edges and corners in images. Deeper layers of CNNs will produce more abstract image features. This hierarchy is the key of CNNs to learn very high dimensional and non-linear mappings from images. In recent years, state-of-art image classification results are almost all achieved by CNNs [23]. Because of the powerful feature extraction ability, we can use convolutional auto-encoders to extract low-dimensional representations from images.

A convolutional auto-encoder has convolutional layers in place of fully connected layers. A convolutional encoder first transforms the input image into a set of down-sized feature maps. Optional fully connected layers can be applied on encoded feature maps. In order to reconstruct the input image, the convolutional decoder has the following key structures:

- Upsampling/unpooling:

  Upsampling and unpooling increase the size of feature maps. Unpooling is an inverse operation of pooling. An convolutional auto-encoder with unpooling layers is introduced in [2] to reconstruct larger feature maps from smaller feature maps. They use a set of so-called switch variables to store the pooled location indexes and recover the unpooled feature maps. This process is visualized in Figure 2-4. An alternative way is to always

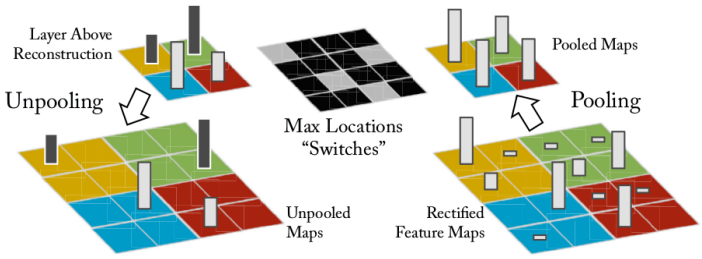


Figure 2-4: The pooling in the encoder is shown on the right. Indexes of preserved numbers in pooled feature maps are stored. The unpooling in the decoder is shown on the left. Stored location indexes are used to unpool down-sized feature maps to the original up-sized feature maps [2].

  put activation from previous decoder layers to fixed locations on unpooled feature maps, e.g., top left corners. This is usually referred to as upsampling. Other empty locations can be filled with zeros or same values optionally.

- Transpose convolution:

  After unpooling/upsampling, a transposed convolution needs to be applied on unpooled feature maps. This means the filters are flipped horizontally and vertically [24]. However this operation is usually miscalled deconvolution.

Apart from the convolutional structure, recurrent auto-encoders are also widely used in sequence to sequence learning type of tasks. Since we want to learn a prediction model on the disentangled representations, we need to additionally employ recurrent structures in hidden layers.

2-5 Recurrent auto-encoders

Recurrent neural networks (RNNs) are powerful tools for modeling sequential data. RNNs are a family of neural networks for modeling sequential data with recurrent neurons. Essentially, parameters of recurrent layers are shared across time steps. The input to hidden layers of RNNs is a combination of current inputs and previous hidden activation from the recurrent layer [25]. The parameter sharing ensures a fixed transition from hidden states and inputs at the current time step to hidden states at the next time step. Such networks can capture correlations within sequential data.

One of the main drawback of RNNs is the vanishing gradient problem when they are trained on long sequences [26]. To back-propagate the gradient through time (BPTT), a chain rule needs to be applied on inputs at each previous time step. The multiplication of Jacobians

on trainable weights can decay to zero exponentially quickly and the training will stop. This makes it difficult for RNNs to capture long term dependencies. However, many important tasks require long term predictions, e.g., machine translations. To address this problem, the long short-term memory network (LSTM) with a special memory structure is proposed in [27]. The main difference is that the LSTM cell maintains a memory called cell state. A illustration of LSTM cell is given in Figure 2-5. The writing and discarding of the cell state

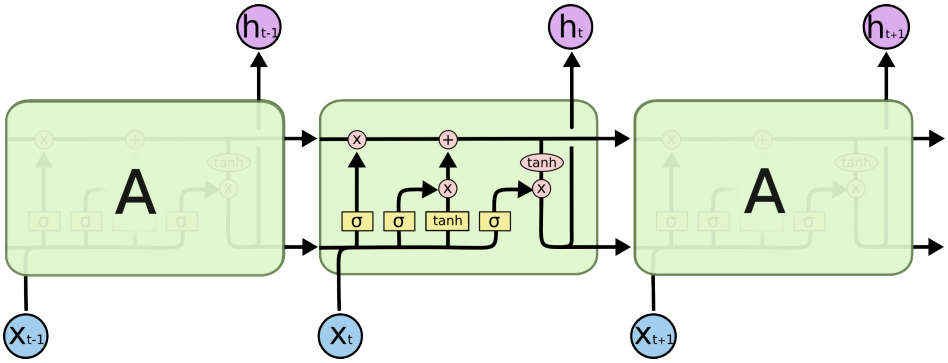


Figure 2-5: An LSTM cell that is unfolded by three time steps [3]. The input and the output hidden state at time step t are denoted as x_t and h_t. The output at time step t is usually a function of hidden state h_t. A cell state denoted as the long horizontal arrow in the cell is maintained across time steps.

are governed by gates that control what to forget and what to memorize. A longer memory kept by LSTM enables the model to look further into the past to make better predictions than RNNs. When using auto-encoders with RNNs, decoders reconstruct the inputs from the processed hidden states of RNNs. For example, a LSTM type of recurrent auto-encoder is introduced to model language translation data-set in [28]. Such model structure can also be applied on images if we first transform images into low dimensional features, for example by encoding.

Another type of RNNs for predictions is proposed in [29], the output prediction of the last time step is used to parameterize the input for the next step. This is very similar to the Jordan/Elman type of networks [30]. The characteristic of the Jordan network is that the prediction is fed forward as the input to the next step to predict further into the future. A Jordan type of auto-encoders will have the structure as shown in Figure 2-6. In Section 6-2, we use the Jordan network shown in Figure 2-6 as a baseline model for comparisons.

On top of the convolutional VAE, we additionally apply LSTM in the latent space of VAE, thus a forward dynamical model can be learned for predictions. Specifically, we use LSTM to predict the disentangled pose representations. This will be further elaborated in Chapter 4.

2-6 AE for control

Because of the feature extraction ability of auto-encoders, quite a few methods have been proposed to apply auto-encoders in control tasks.

The Embed to Control (E2C) method [31] is proposed to employ VAE for control. Firstly, VAE is used to extract state representations from raw images. Usually, raw images can

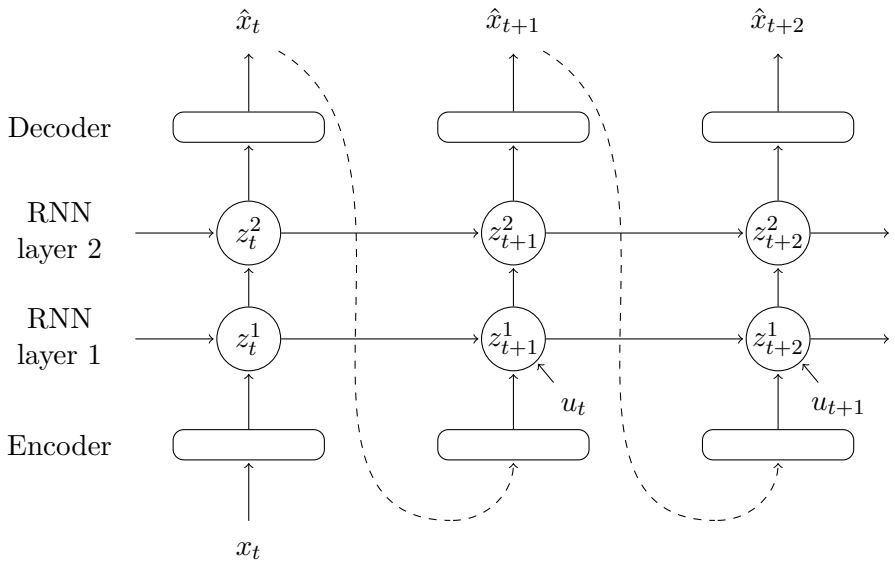


Figure 2-6: A Jordna/Elman like recurrent auto-encoder. The current prediction is fed into the next step as an input. Actions are directly fed into recurrent cells for transitions.

be regarded as highly non-linear observations of the system dynamics. It is much easier to perform control in the latent space than directly in the image space. The E2C method leverages this idea and learns a linear transition model in the latent space given consecutive images and actions. Specifically, the transited state representation given the previous image and action are regularized to be close to the representation of the current image. With a consistent latent space learned, a locally optimal control like iLQG [10] can be performed.

The E2C method directly learns a one-step ahead transition model in the latent space. In order to learn transitions with longer time steps in the latent space, it is beneficial to add recurrent structure as we mentioned.

The idea of performing control in the latent space is further explored in [32]. Instead of learning a linear model, the proposed Deep Dynamical Model (DDM) learns a non-linear auto-regressive exogenous (NARX) model in the latent space. So it looks back at a window of past representations and control inputs and predicts a future representation, which is further decoded into a future image frame. However, the transition is deterministic as it uses the deterministic auto-encoder rather than VAE. It neglects that the action outcome can be stochastic. With learned representations, a nonlinear model based predictive control (MPC) is performed online.

DDM is not fundamentally different from E2C. Both of them learn representations and perform control jointly in the latent space. They have been successful applied on a few simulation tasks. For real world tasks where the action space and the observation are much higher dimensional, more interaction data are probably required. This is the main drawback of E2C type of algorithms.

2-7 Multi-modal learning for control

By now, we have explained the auto-encoder type of models for representation learning and how these representations can be used for control. The other important ingredients for state representation learning are data themselves. When the dynamic process is observable, state representations can be estimated from data and possibly fused from different modalities of observations. Naturally, we can apply the same idea in representation learning.

Multi-modal learning is a representation learning technique that finds a joint representation from data modalities. Statistically, different modalities are assumed to be generated from the same set of latent variables. They are usually correlated with each other as they are different observations of the same random event. For example, sudden drops of depth values in depth images and sharp edges in RGB (red, green and blue channels) images are essentially describing the contrast of the scene differently.

To learn joint representations, a straightforward method is to use auto-encoders. In [33], a deep auto-encoder is applied to learn a shared representations between audio data of speeches and images of lips. The key idea is that higher order correlations among modalities are easier to find if shallow representations are first extracted from each modality. However, the contrary is suggested in [34], where learning directly from all modalities leads to more robust representations. In their experiments, they concatenate RGB and depth images and train the auto-encoder end-to-end to reconstruct both. From a information perspective, the variation of information loss among modalities is proposed in [35]. Variation of information penalizes the information distance between random variables (data modalities). A joint distribution that generates different modalities can be learned with this objective.

Aforementioned multi-modal learning methods mainly focus on modeling the shared data distribution. However, these methods are not specific to control tasks. To learn joint representations for control, different auto-encoders from previous sections can be applied to model multi-modal data and extract joint representations without much modifications.

2-8 Discussion

This chapter gives an overview of existing work on performing control tasks with auto-encoders. The structure of auto-encoders usually depends on the form of data. For tactile sensory data as introduced in [36], a feed-forward auto-encoder is enough. For image data, convolutional auto-encoders are suitable. This is also the focus of our work. However, it is less common to use recurrent networks with auto-encoders for control. RNNs can be a viable choice for modeling long time transitions in the latent space. A longer prediction horizon is in general useful for residing horizon control type of control strategies. This idea is also reflected in our work.

In terms of the interpretation of auto-encoders, VAE is more commonly considered as a powerful tool to infer the distribution of the hidden representations. Suppose we want to approximate a locally linear Gaussian dynamic model of any complex non-linear system, VAE is useful since we can learn a linear transition in the probabilistic latent space with a Gaussian prior.

A main issue of existing methods is that they can all be considered as general dimensionality reduction techniques. They neglect possible interpretations of the learned representations. A different point of view would be extracting only the task relevant information from the data and use it for control. For example, information of the pose is enough in pendulum balancing type of tasks and appearance of the scene is not relevant. If we can extract only the task relevant information, it is possible to learn a state space with a lower dimensionality. This can alleviate the curse of dimensionality problem. In case that information disentangling is possible with VAE, another appealing property is that we can sample the data with variations in only one disentangled aspect from the decoder.

As a conclusion from reviewing existing work, we decide to use VAE to learn disentangled state representations from image data. Additionally, we will use RNNs to facilitate predictions in the latent space. Then the decoded predictions in the data space can be used as a training objective. So the model can be trained in a self-supervision manner. Apart from the proper design of model structures, observation data themselves are also essential for state representation learning. We further decide to use multi-modal data for state representation learning.

Chapter 3

Multi-modal learning

Nowadays, prices and sizes of various sensors are decreasing. Data that reflect different aspects of the environment can be easily obtained without much extra costs. Though the nature of data from different modalities can be totally different, processing on these modalities together leads to more robust representations. In manipulation tasks, the tactile modality provides information about the pressure and contact information, the depth modality provides segmentation of the scene and the RGB modality provides more high frequency signals like sizes, shapes, orientations and so on. With different modalities available, the control task does not have to rely on any single modality to infer the state. Therefore multi-modal learning is especially useful when one modality is noisy and extra observations are needed. In this Chapter, we first explain our motivations for multi-modal learning on RGB-D data. A test case is then introduced that is suitable for our problem setting.

3-1 Multi-modal representation learning

We plan to learn multi-modal representations for control, for example, RGB images and depths images. With the assumption that different modalities share a common latent space, one modality can be thought of as soft labels for the other which is useful for unsupervised learning. Thus multi-modal learning can possibly lead to more robust and useful representations for control. Specifically to our tasks, we learn joint representations from RGB and depth modalities. Double modalities of RGB and depth images are useful since:

- Compared to RGB images, depth images provide clearer segmentations of the scene, which is useful for manipulations. Depth image are much more invariant to lighting changes, different image backgrounds and textures.
- Compared to depth images, RGB images are more useful for identifying object identities with different shapes, colors and textures. RGB images contain more high frequency information and are easier to collect than depth images.

From a sensor fusion perspective, RGB and depth information is supplementary to each other. States are fused and updated such that the posterior given observations from different modalities are maximized. Thus the idea of sensor fusion coincides with multi-modal learning. Multi-modal representations serve as better cues for control.

To verify this idea, we apply multi-modal learning to the object poking task. The object poking is a basic robotic manipulation task. Suppose that a robot arm is asked to poke an object in the scene to a target location, a typical pipeline would be segmenting the scene, estimating the pose of the object, calculating a viable poking trajectory of the end-effector and executing the poking trajectory. Instead of explicitly estimating poses of objects, a deep learning approach would be extracting a representation from image data and use that representation to predict the poking action.

There is a group of similar work that use CNNs to learn representations for poking [4][37]. We thus choose the object poking task as a test case. Moreover, object poking is also a proper test case for information disentangling and prediction model learning. In this thesis, we will validate all our models and hypothesis on the object poking task. We will further elaborate it in Chapter 5. In the next section, we will explain how we can apply the Siamese network for representation learning and incorporate multi-modal learning.

3-2 Multi-modal learning with the Siamese network

Among various network models, the Siamese network is suitable for extracting representations without reconstructing image data. The Siamese network is proposed in [38] to learn a similarity measure of image pairs. It is essentially two copies of one identical convolutional network that operate on comparable image pairs. Therefore two low dimensional feature representations can be extracted from an image pair. Since the parameters of two copies are the same, the difference of two embedded representations only depends the data pair. With a carefully designed training objective, Siamese networks can properly capture similarities and differences of varying data pairs.

In [4], a Siamese network is applied to learn state representations for the object poking task. Their experiment is in line with our experiment settings in Chapter 6. The difference is that they perform real world experiments using a Baxter robot while we collect all object data in simulations. Compared to our settings, they use real world objects that have more complex textures and shapes. However, we use objects that are significantly smaller. The camera is also farther away to the scene which gives a much bigger view. Effectively, objects in our setting have a larger moving range.

The Siamese network for object poking in [4] is shown in Figure 3-1. Given an image pair (I_t, I_{t+1}), both branches of the Siamese network first extract a state representation pair (x_t, x_{t+1}). From the state pair, subsequent layers have to predict the action that will cause the transition from x_t to x_{t+1}. Clearly, the Siamese network is modeling an inverse dynamics of the poking task. Given a dynamic system, an inverse model predicts the action that will bring the current system state into the desired state. In optimal control settings, inverse models are commonly used as feedforward controllers [39]. It is primarily useful in trajectory tracking type of tasks.

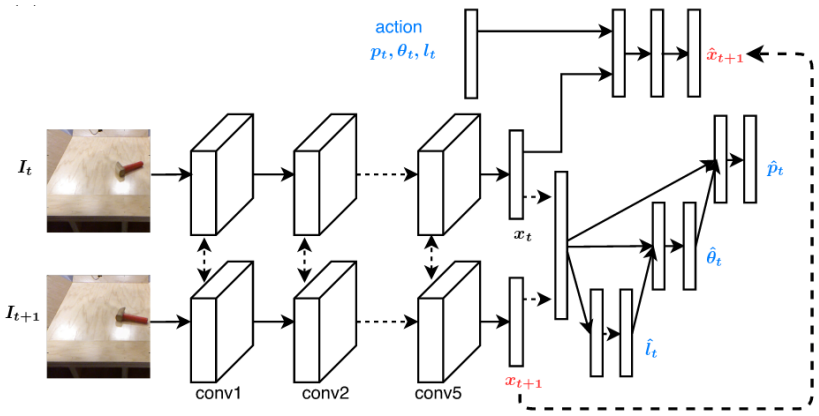


Figure 3-1: The Siamese network is trained to predict the poke action (p_t, θ_t, l_t) given the image pair (I_t, I_{t+1}) before and after the action [4]. Furthermore, an extra forward dynamic model is trained as a regularizer.

This model further includes a forward dynamic model on extracted representation pairs of the Siamese network. Given the representation x_t and action u_t, the next representation x_{t+1} should be predicted from a forward dynamic model. This is formulated as a l_1 (absolution values) regularizer in the total loss. It transforms the inverse modeling problem into a self-supervision problem, meaning that action data are enough for the supervised learning. However, the action space of poking in the image view is continuous and considerably large. To circumvent this problem, the action space can be discretized into certain bins. The self supervision problem is then further transformed into a classification problem which is well studied in researches of CNNs.

For the multi-modal learning, we apply the same Siamese network structure respectively on RGB and depth images. We further concatenate RGB and depth images into images with four channels and train the Siamese network from scratch. We evaluate two modality specific networks and one double modalities network on the object poking task in simulations. Experiment results are reported in Chapter 6.

3-3 Discussion

An important trick of training the above Siamese network is to additionally add a forward model as a regularizer. Learning a forward and an inverse model at the same time is beneficial. The forward model can be learned in the latent space of the inverse model, which is significantly easier than directly in the image space. Also, the training objective of the inverse model prevents from a degenerate solution because merely training the forward model will collapse both representations of the Siamese network to zero. However, the problem of such model is also significant because:

- The learned inverse model solves a classification problem on the current image pair and can only be used for greedy planning.
- The forward model is only trained as a l_1 regularizer rather than regressions. Thus in

can not be used for multiple step ahead predictions neither in the latent space nor in the data space.

- The planning is coarse since the action space is discretized into limited number of bins. The learned representations from the Siamese network are not interpretable by human and not disentangled.

To solve these problems, it is beneficial to learn a forward model on the representations directly. We propose to disentangle the appearance and the pose representations from convolutional auto-encoders and learn a multiple step ahead prediction model on the pose representations. The solution will be explained in the next chapter.

Chapter 4

Learning disentangled state representations

Although representation learning has been studied very well in the computer vision field, learning a representation for robotic control remains a challenging problem. The main difficulty lies in the fact that robotic tasks require pose information about the task relevant objects. In contrast, many state-of-art computer vision methods focus on semantics of images. Moreover, learned representations are not interpretable by human and do not contain explicit physical information. Considering the classical way of state estimations, pose information is directly estimated by filtering. This is not the case for control using deep networks. If we can disentangle the representation into a task-relevant part like the pose of objects and an irrelevant part like the colors and textures, we can learn a lower dimensional and more consistent state space. This promotes simpler and quicker model-based optimal control.

In this chapter, we will first introduce our approach from a image synthesis point of view. A basic convolutional VAE for information disentangling is then introduced. Afterwards, we explain how we can also use LSTM to enable long term predictions.

4-1 Image synthesis

Humans can interpret an image by understanding different aspects of it, like the lighting of the scene, the viewpoint of certain objects, the colors and textures of certain areas and the distances. Many recent vision algorithms have been proposed to understand the scene, such as scene segmentation [40] and object detection [41]. These algorithms aim at learning abstract representations that describe certain aspects of images only from image pixels. Such extraction process can be viewed as an inverse of image rendering. When a lighting condition, a viewpoint, a certain shape, texture and position of a 3D object and other factors are given, a scene can be rendered and projected on the camera image plane. This is a natural sampling procedure from the complex image distribution. If images can be rendered from various factors, it is certainly possible to deduce disentangled factors backwards from images

leveraging the ability of unsupervised representation learning. From an optimal control perspective, we should be able to answer how the scene or appearance will change with certain transformations applied on those extracted factors or with actions applied on objects. Being able to predict means being able to plan, which is the primary concern for representation learning in robotic control.

An convolutional auto-encoder combined with VAE is proposed in [42] for image synthesis. In order to disentangle the representation, they specifically encode the image into separate factors including the lighting, pose, shade and so on. However, the network will not disentangle them automatically. They have to organize the training data into mini batches where each batch only has one chosen variation, e.g., only changing the texture of the object. Then in the back-propagation phase, they introduce a special training procedure of updating weights only for that chosen texture representations. While their work succeeds at image synthesis after training, it is fully supervised and the difficulty of data-set preparation restricts it to limited applicable cases. Their work does not focus on disentangling the pose information of relevant objects, but on general image rendering. A convolutional auto-encoder structure specifically for learning 3D transformations is proposed in [5]. The auto-encoder splits the

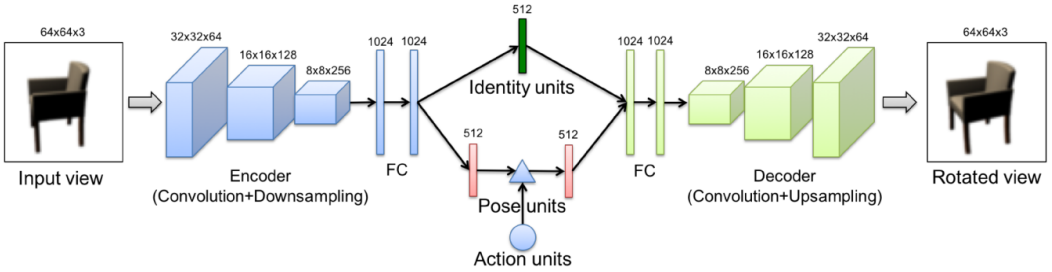


Figure 4-1: An convolution auto-encoder that learns 3D rotation on chair images [5].

hidden representation into the identity unit part and the pose unit part as shown in Figure 4-1. When certain transformations are applied on the pose unit, the auto-encoder is required to reconstruct the image after transformations. In terms of chair images, they use three scalar vectors to denote the clockwise rotation of 15 degrees, no rotation and counter-clockwise rotation of 15 degrees. The triangle shown above indicates tensor products between the pose units and the action units, generating the transformed pose units. So the action units select the the input pose units and transform them into the output pose units. In the training phase, the auto-encoder has to reconstruct the correct rotated chair images from original chair images. Essentially it models the local transformation in the nonlinear pose manifold.

4-2 Learning a disentangled probabilistic state space with VAE

Although the auto-encoder in [5] is able to disentangle the representation, the transformation is not continuous. The pose representations will easily fall off the manifold if we apply other transformations and the transformation will fail. Therefore it cannot achieve a rotation other than multiple of 15 degrees. Moreover, their model structure is deterministic. If we apply such a model structure on data generated from robotic control tasks, it does not reflect the fact that the outcome of applying actions is usually probabilistic conditioned on the actions

and the states. Therefore, we propose to incorporate the variational auto-encoder (VAE) as introduced in Section 2-3 to learn a disentangled state space that is both stochastic and continuous. The difference of our model is that:

- We use VAE to learn probabilistic representations.
- We learn a transition model that is stochastic rather than deterministic.
- We learn a continuous state space where the action space is much larger.

As introduced in Section 2-3, VAE is trained to approximate the true posterior of the hidden representation Z. In terms of disentangle representations, we assume that the representation can be split into two distributions Z_i and Z_p that are independent from each other. Considering a general control task (e.g., object pushing and pendulum swinging), the appearance of the scene usually stays identical. The colors, textures and shapes of the task relevant objects will not change much within a short action sequence. However, the position and orientation of the object can vary much more. The appearance representation that stays identical during the transitions is noted as z_i and the pose representation that transits when a continuous action is applied is noted as z_p. Thus we can use two separate recognition models for each part of the inferred representations. A forward model is incorporated to learn the forward dynamics on the pose representations.

An graph illustration of the proposed model is given in Figure 4-2.

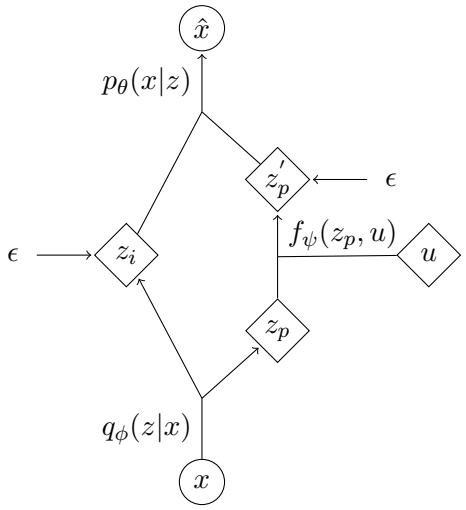


Figure 4-2: The representation z is splitted into identity z_i and pose z_p. An extra transition model is included to model the forward dynamics of the pose. Diamond blocks of z mark them as deterministic variables because of the re-parameterization trick applied.

The recognition model q_ϕ of VAE infers the representation z which is then split into z_i and z_p. Note that they are shown as deterministic variables in Figure 4-2 because of the usage of the re-parameterization trick, but they are actually stochastic. In practice, this means that z_i and z_p are outputs of the encoder that describe the approximated posterior, e.g., means and variances of the Gaussian. Given an control action u, the model also learns a forward dynamics f_ψ parameterized by ψ:

$$z_p^{'} = f_\psi(z_p, u) \tag{4-1}$$

Algorithm 1 Training of information disentangling VAE.

Require: m: number of mini-batchs; N: minibatch size; $i = 0$; $\mathcal{D}$: training data set.

$q_\phi, q_\psi, p_\theta \leftarrow$ Initialize parameters ϕ, ψ, θ.
while $i < m$ **do**
  Sample mini-batch M containing N tuples of $(x_i, x_i^{'}, u_i)$ from $\mathcal{D}$.
  Random sample from q_ϕ and q_ψ by sampling noise ϵ.
  $g \leftarrow \nabla_{\phi,\psi,\theta}\mathcal{L}_{total}(M)$
  $\phi, \psi, \theta \leftarrow$ Updates using gradient g with SGD like methods.
  $i+ = 1$
end while
return ϕ, θ, ψ

which predicts the transited pose representation $z_p^{'}$. Note that f_ψ can be both continuous and discrete and can be modeled as feed-forward or recurrent layers. Because of the extra transition in z_p, we denote the recognition model for $z_p^{'}$ as $q_\psi(Z_p^{'}|X, u)$ for simplicity. Then given an input point x and action u, we can sample from both approximated posteriors:

$$q_\phi(Z_i|X = x) \tag{4-2}$$

$$q_\psi(Z_p^{'}|X = x, u) \tag{4-3}$$

From the sampled representations, we can reconstruct the corresponding target image $\hat{x}^{'}$ using the generative model $p_\theta(X|Z)$ or $p_\theta(X|Z_i, Z_p)$. For an input data point x, a corresponding target data point $x^{'}$ and an action u, we maximize the the lower bound:

$$\mathcal{L}(\theta, \phi, \psi; x, x^{'}, u) = -D_{KL}(q_\phi(Z_i|x)\|p_\theta(Z_i)) - D_{KL}(q_\psi(Z_p^{'}|x, u)\|p_\theta(Z_p)) \tag{4-4}$$

$$+ \mathbb{E}_{z_i\sim q_\phi,\, z_p^{'}\sim q_\psi}[\log p_\theta(x^{'}|z_i, z_p^{'})] \tag{4-5}$$

Here $p_\theta(Z_i)$ and $p_\theta(Z_p)$ are priors for the identity and the pose representations. If no prior knowledge is known, both priors can be chosen as standard Gaussian distributions. Then the first two KL divergence terms have analytical forms. The third term is the expected negative reconstruction error which can be estimated by averaging over the samples of z. For a training data-set $\mathcal{D}$, the total loss is:

$$\mathcal{L}_{total}(\mathcal{D}) = \sum_{(x\,,x^{'},u)\in\mathcal{D}} \mathcal{L}(\theta, \phi, \psi; x, x^{'}, u) \tag{4-6}$$

In practice, the loss is usually summed up over a mini-batch of the training set. The disentangling VAE can be trained by following Algorithm 1. We use it to learn disentangled representations from images and examine if representations for the appearance and pose are indeed disentangled. We can also apply actions on the pose representations to see if the model gives the correct prediction. We will report the result in Chapter 6.

4-3 Incorporating recurrent structures

In the last section, a VAE like model is proposed to disentangle the appearance and the pose representations and learn a forward dynamic model on the pose representations. The

forward model however only predicts one step ahead. In model-based optimal control and reinforcement learning, it will be more useful if the model can predict the state a few steps in the future, e.g., for a residing horizon control scheme. For example, MPC optimize an action sequence within a future horizon. A necessary step is to evaluate the cost of the nominal trajectory given the initial state and the action sequence to be optimized. To achieve such long-term predictions, we need to introduce recurrent structure in our model. In [5], a recurrent layer is introduced to rotate chair images step by step. We use recurrent layers to transform the pose representations as well. However, we further use LSTM in our model rather than plain recurrent cells. We incorporate it with VAE to learn stochastic and continous transitions rather than deterministic and discrete as in [5]. The model structure is shown in Figure 4-3. The motivations of such model are as follow:

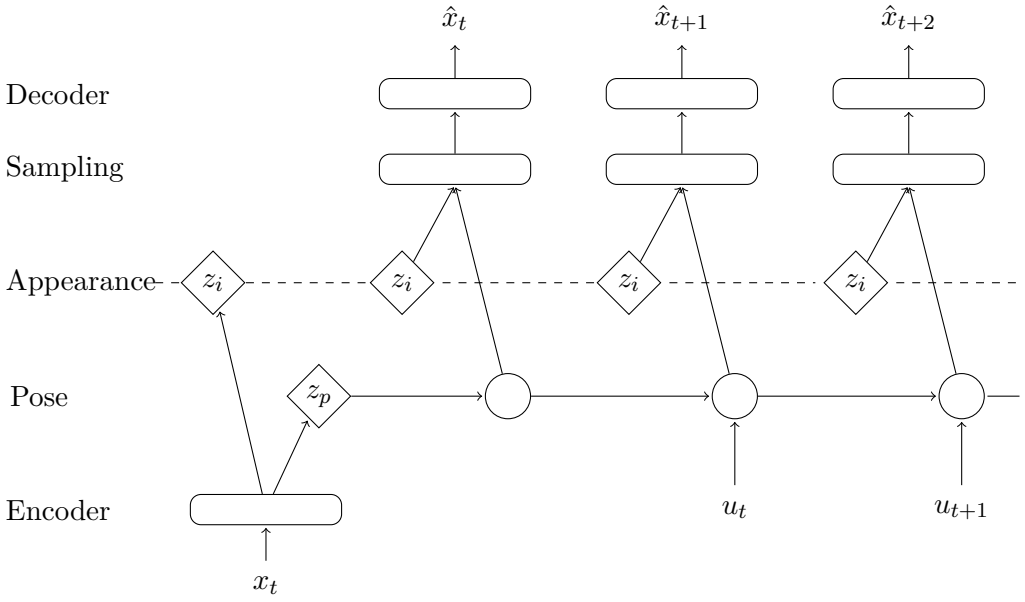


Figure 4-3: A recurrent version of information disentangling VAE. The network introduced in [5] has a similar recurrent structure. The model difference is explained above. The encoder and decoder networks are shown as round rectangles. The sampling layer is marked as the same for simplicity. The circles are unrolled LSTM cells. Pose representations are fed into the LSTM network to initialize the hidden states of LSTM cells.

- Reusing of the identity representations: For some control tasks, it is reasonable to assume that the appearance of the scene stays identical. Thus part of the representations are reused across the time sequence.
- Applying the LSTM structure: To facilitate long term predictions and mitigate the vanishing gradient problem of RNNs, we use long short-term memory networks.

As is the case for the non-recurrent version in Section 4-2, the recognition model first splits representations into the identity and pose parts. The pose representations are used to initialize the hidden states of the LSTM network. The LSTM cells also take in corresponding actions to learn a consistent forward dynamics on the pose representations. The identity representations

are however shared across multiple time steps. From transited pose representations and identity representations, we can take samples of the approximated posteriors. The decoder will then reconstruct the image at each time step from the corresponding sampled points.

To reconstruct the $\hat{x}_t$ from x_t, no transition happens on the pose representations. Thus we feed in a zero action to the LSTM network at the first time step. Note that LSTM layers can be stacked deeper although only one layer of LSTM is shown in Figure 4-3. In case that the forward dynamics is very nonlinear, it is necessary to use more than one hidden layer to learn a more complex transition function. In this case, the LSTM cell at each time step takes in the hidden state of the previous time step and the hidden state of the previous layer at the same time step as inputs and keeps a cell state within the same layer across time steps.

In practice, time steps that RNNs look back are usually truncated to a fixed number for stabilizing the training. Suppose that the truncated backpropagation through time is T, then the network is trained on mini-batches of time sequences. Each sequence is a series of transition tuples $S_{t_0:T} = \{(x_{t_0}, x_{t_0}, 0), (x_{t_0}, x_{t_0+1}, u_{t_0}), ..., (x_{t_0+T-2}, x_{t_0+T-1}, u_{t_0+T-2})\}$. The loss is simply a summation of losses from each transition tuple.

$$\mathcal{L}(\theta, \phi, \psi; S_{t_0:T}) = \sum_{i=0}^{T-2} \mathcal{L}(\theta, \phi, \psi; x_{t_0+i}, x_{t_0+i+1}, u_{t_0+i}) + \mathcal{L}(\theta, \phi, \psi; x_{t_0}, x_{t_0}, 0) \tag{4-7}$$

Note that parameters ψ of the forward dynamic model are now weights and biases of the LSTM cells that are shared over time steps. Following the steps in Algorithm 2, we can train the LSTM based VAE on batches of sequence data. In the sampling step, we can simply sample from the identity and transited pose representations separately and concatenate them together at each time step.

We now need to collect a suitable data-set for validating our model performance. Specifically, we need to verify the continuous transitions and information disentangling of our proposed model. Thus the control task should have a continuous action space and a stochastic system dynamics. Also the task should have separable appearances and poses of task relevant objects. The considerations and procedures of data preparations will be explained in the next chapter.

Algorithm 2 Training of LSTM based VAE.

Require: m: number of mini-batches; N: size of mini-batch; T: truncated back-propagation steps; $i = j = k = 0$; $\mathcal{D}$: the training data set.

$q_\phi, q_\psi, p_\theta \leftarrow$ Initialize parameters ϕ, ψ, θ.
while $i < m$ **do**
  Mini-batch $M \leftarrow$ Sample N sequences of $S_{t_0:T}$ from $\mathcal{D}$.
  while j<N **do**
    $S_{t_0:T} = M[j]$
    z_i, $z_p \leftarrow$ encode x_{t_0}
    $\{z'_{p,t_0}, ..., z'_{p,t_0+T-1}\} \leftarrow f_\psi(z_p, \{0, ..., u_{t_0+T-2}\})$ (LSTM network).
    while k<T **do**
      $z_{t_0+k} \leftarrow$ sample given z_i, z'_{p,t_0+k} and noise ϵ.
      $\hat{x}_{t_0+k} \leftarrow$ reconstruct from z_{t_0+k}.
      k+=1
    end while
    $\mathcal{L}_{total}(M) += \mathcal{L}(\theta, \phi, \psi; S_{t_0:T})$, given the ground truth and reconstructions.
    $j += 1$
  end while
  $g \leftarrow \nabla_{\phi,\psi,\theta}\mathcal{L}_{total}(M)$
  $\phi, \psi, \theta \leftarrow$ Updates using gradient g with SGD like methods.
  $i += 1$
end while
return ϕ, ψ, θ

Chapter 5

Data-set Collection

The sampling efficiency problem is general in deep learning researches. For image and language modelling tasks, huge amount of labeled data does not usually cause any practical issues. However for robot interaction data, continuous data collections on real robots can lead to fatigue and hardware problems. To quickly verify proposed algorithms and network structures, we can use data collected from physical simulations. They comply with physical laws but also neglect factors that are not of high priorities. For example, complex static frictions in object manipulations and momentum transfer in object collisions can be safely approximated with easier ODE models. Among various tools of robotic simulations, Gazebo is one of the most common used and flexible simulation environment.

If affordable hardware is available, real world data-set would be a better choice. The Baxter robot is used to collect object grasping data in [43] where random picking up actions are recorded together with scene images. In [4], the Baxter robot is also used to collect object poking data-set. However, we do not have similar robots at hand for instant and continuous experiments. We thus decide to build vision and control pipelines in ROS and conduct simulations in Gazebo. In this chapter, we introduce the motivation of our data-set. The data collection process and the simulation system is also briefly introduced.

5-1 Data-set

Among various robotic control tasks, the object poking is considered one of the most fundamental tasks in the sense that it is a basic manipulation for more complex manipulations. During the poking and pushing of the object, the appearance of the scene stays relatively identical while the position and orientation of the object changes constantly. So it is a suitable test case for learning disentangled representations. Apart from that, the pose transition of the poked object is actually very non-linear due to different choices of poke positions, directions and distances when the end-effector approaches the object. The action space is also considerably large. Therefore it will be useful to examine whether such continuous dynamics can be modelled by deep networks. Moreover, the object poking is also a good test case for

planning. As a staring point, we collect data of the object poking task and use them test our hypotheses.

The overall goal is to simulate a robot arm that randomly pushes a block on a table. The object poking process is repetitive. The data to be collected at time t is shown in Table 5-1.

Table 5-1: Data to collect

Images before poking:	RGB_t, $DEPTH_t$
Images after poking:	RGB_{t+1}, $DEPTH_{t+1}$
poking locations on image plane:	px_t, py_t
poking direction:	θ_t
poking length	l_t
object location	coordinates: (x_t, y_t, z_t), $(x_{t+1}, y_{t+1}, z_{t+1})$ quaternion: (a_t, b_t, c_t, d_t), $(a_{t+1}, b_{t+1}, c_{t+1}, d_{t+1})$

The total action is denoted as a vector $a_t = (px_t, py_t, \theta_t, l_t)$. In simulations, an action vector is randomly generated at every time step and then executed. In the meantime, the object locations and images are saved. To improve the efficiency of data collections, the poking location (px_t, py_t) is chosen randomly on one side of the block to prevent poking in the air.

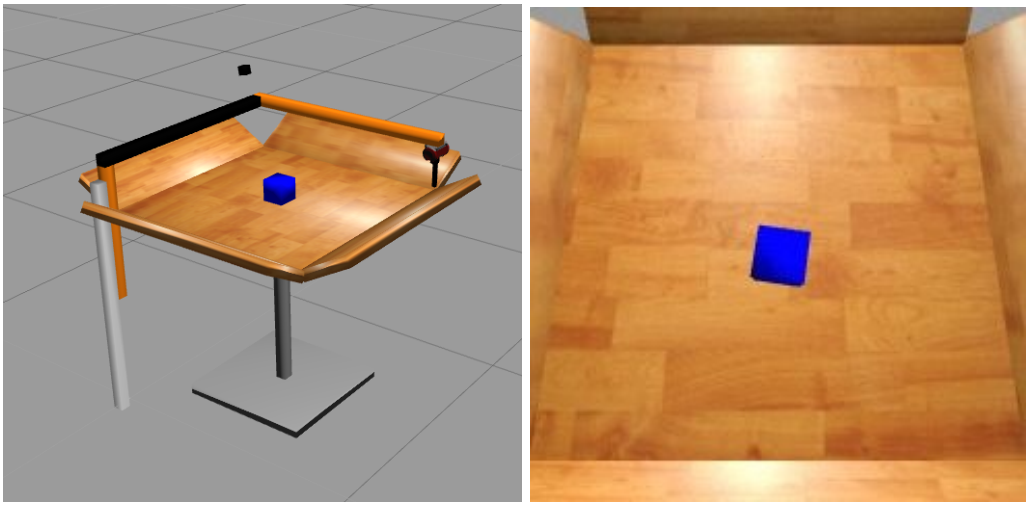


Figure 5-1: Poking simulation: Gazebo simulator (left); simulated camera view (right).

5-2 Data collections

In ROS, a simulation system is built up for object poking simulations and data collections. The key idea is to use a series of service nodes to handle all data query requests. For example, a node for picking up poking point on the object and a node for projecting the poking location on the image plane are run in parallel to record the poking actions. More details about the ROS simulation are provided in Appendix A-1.

With all necessary services available, we only need a central node to perform the poking task, call services in between and save poking data. The simulated camera has a resolution of 240×240. Thus poking positions (px_t, py_t) take values in the range of $[0, 240]$. To make sure that objects move noticeable distances, we generate poking actions with lengths $l_t \in [0.04, 0.08]$ in meters. The poking angles θ_t are within $[0, 2\pi]$. A set of colored blocks and tables with different textures are prepared and randomly spawned into the simulation. Cubes have sizes of 8×8 cm and background tables are around 80×80 cm. We optionally change table colors and textures to introduce more variations to the data set. When a poking action is generated, the central node publishes the expected joint positions to joint controller command topics to execute the poking trajectory. The data collection process of the central node is shown in Algorithm 3.

Algorithm 3 Data collection process of the central node.

Require: m: number of collection runs; n: Number of pokings per run; $i = t = 0$: iterator.

Spawn the robot arm, randomly spawn an object O and a table.
while $i < m$ **do**
    while $t < n$ **do**
        Generate poking action (px_t, py_t), (θ_t, l_t)
        Save the object pose and poking action.
        Approach vertically and horizontally to object O.
        Poke object O at (px_t, py_t) with (θ_t, l_t).
        Retract the arm.
        Save $RGB_t, DEPTH_t$.
        $t += 1$
    end while
    Delete and re-spawn a table and an object O into the simulation.
    $i += 1$, $t = 0$
end while

After the data collection, some pre-processing procedures are performed.

5-3 Data processing

In simulations, ROS tries to synchronize different messages and services at prescribed frequencies. It is possible that a message is missing because of communication delays among nodes. This happens in our simulations. When a image request is sent, the service node only waits for the image message for three periods of the simulation for continuity of the experiment. If messages do not arrive, corresponding images are discarded. After data collection, missing images are sorted out and their corresponding poke actions are discarded as well.

The action data are processed based on training requirements of different network models. They are either discretized for classification or normalized to $[0, 1]$ as inputs.

5-4 Examples of collected data sequences.

Several example data sequences are given in Figures 5-2 and 5-3.

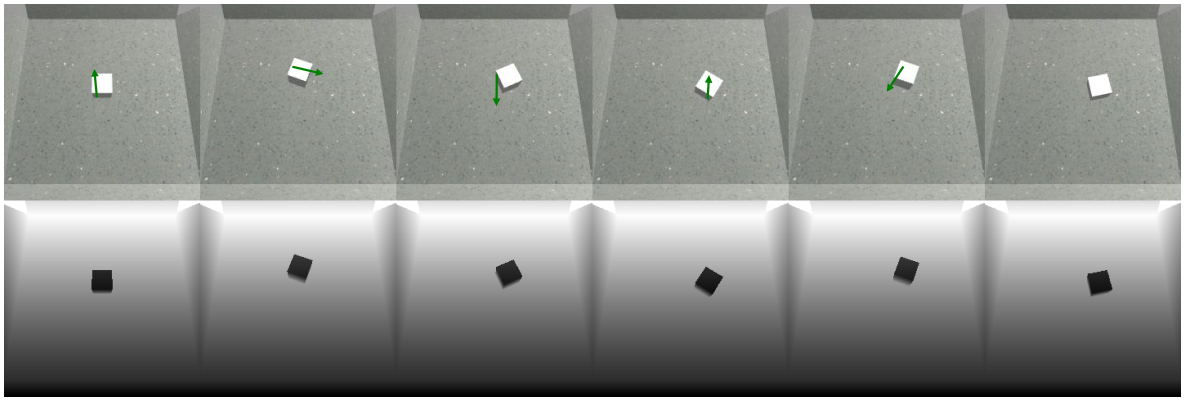


Figure 5-2: A sequence of six images. The first and the second rows are RGB and single channel depth images.

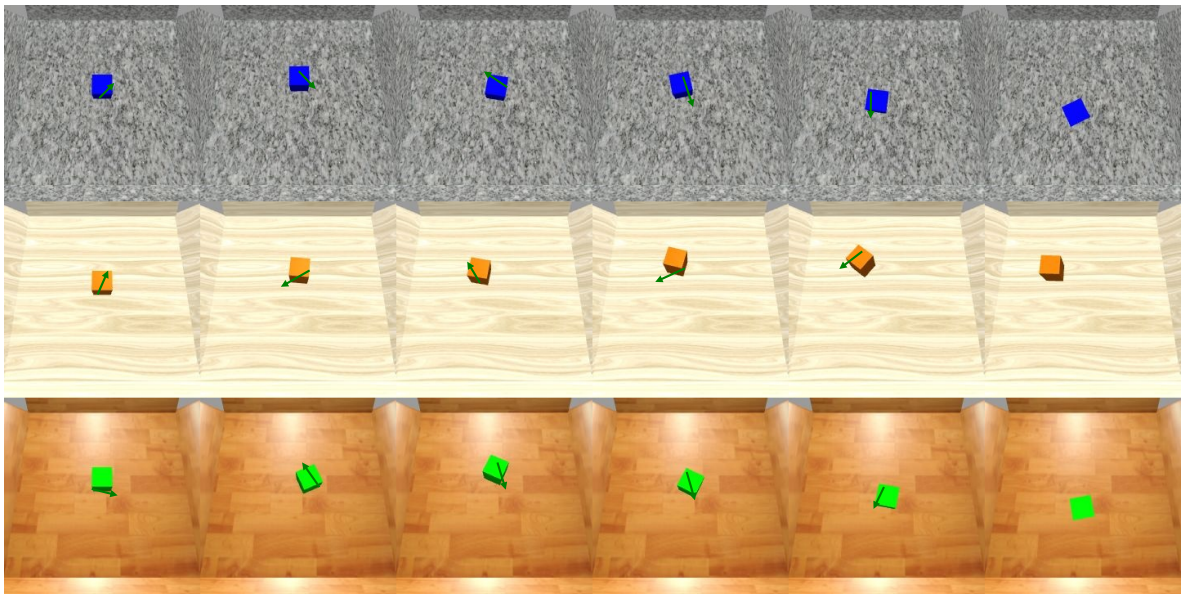


Figure 5-3: The poking actions are marked as green arrows in images. The anchor points of arrows are positions where the end-effector starts to poke. The lengths and angles are proportional to the actual poking actions.

Chapter 6

Experiments and comparisons

With the collected object poking data-set at hand, we can evaluate our hypotheses about information disentangling and multi-modal learning. We can either test trained models online in the Gazebo simulator or validate them on a pre-collected test set. In this chapter, we first report the result of multi-modal Siamese network in object poking simulations. For information disentangling and movement predictions of poked objects, we introduce a few baseline models that can predict future observations. In Section 6-3, we give some details about the common setting of baseline models. Then we compare them with the LSTM based VAE and evaluate the prediction ability of the learned forward dynamic model. In Section 6-5, we further examine whether or not the learned representations are disentangled.

6-1 Multi-modal learning

In this section, we give the poking simulation results with Siamese networks. The Siamese network proposed in [4] has an Alexnet [11] structure in each of its branch. Weights of the network are initialized from pre-trained weights of the Alexnet. Thus we use Alexnet like structure in the depth, RGB and RGB-D (RGB and Depth) Siamese networks. The only difference is that the RGB-D Siamese network accepts one more channel in its input layer. The depth Siamese network accepts single channel depth images as its input. Additionally, we artificially corrupt the depth data. The motivation is that depth images are quite symmetrical without textures and colors in our experiments. Injecting noises leads to more robust representations which has similar motivations as DAE in Section 2-2-1. For the RGB Siamese network, weights are initialized from pre-trained weight. We do not find existing work that trains a large Alexnet like network on depth images. So weights for the depth and RGB-D Siamese networks are randomly initialized and trained from scratch.

For classification on poking actions, depth and RGB images of size 240×240 are discretized into 20×20 bins which gives 400 classes of poking positions. The poking angle is discretized into 36 bins of 10° each. For our collected dataset, the poking distance within $[0.04, 0.08]$ is discretized in 10 bins. Thus the training on poking data is essentially a $400 \times 36 \times 10$

classification problem. We train three models with the same number of epochs and learning rate.

During the test phase, we validate each trained network online in simulations. Given an initial image and a target image of the object, the Siamese network makes a classification on what actions will poke the object from the initial image to the target image (position). Classified pixel locations are projected back to the 3D world using a simulated camera model. After one poking with predicted poking actions, new images are generated and used for the classification at the next time step. Note that the target image never changes and the model does not look into the future, it is essentially a greedy planning scheme.

For poking simulations, we fix the target location of the cube across all testings for different models. But we randomly choose the initial location within a certain range and randomly exchange different cubes and tables. All test runs are terminated after 15 steps of poking. After each run, pose errors between the target position and current locations after 15 steps of poking are retrieved from Gazebo and recorded. The final pose errors of each model are shown in Figure 6-1. For the evaluation on position errors, the depth Siamese network does

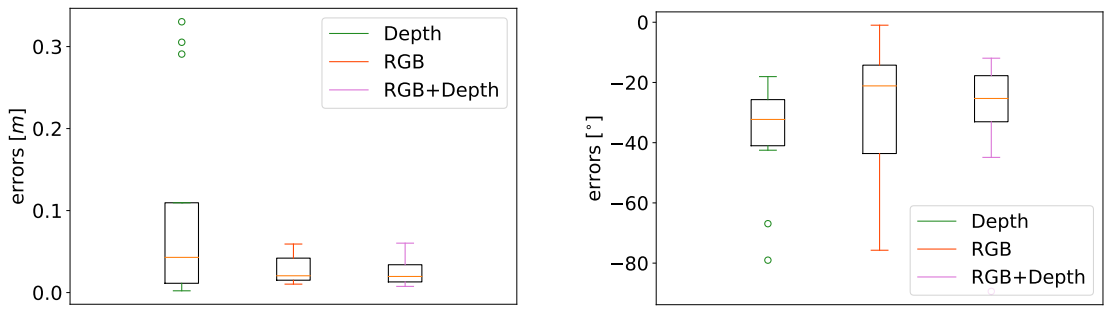


Figure 6-1: Left: position errors; Right: orientation errors.

not perform as well as the RGB and the RGB-D Siamese networks in terms of the final position errors. There are unsuccessful poking runs where the final position errors are larger than ten centimeters. Adding depth modality slightly improves the performance.

For the evaluation on orientation errors, we are only measuring the yaw difference since the cube never flips on the table. Networks with the depth modality have less variances on yaw errors, this suggests that the depth image is a better cue for object orientations in our experiments. Depth values alway have sharp contrasts on edges of cubes since they are closer to the camera. This does not hold for RGB values when the background color looks similar to the cube. Thus it verifies our motivation that multiple modalities act as supplements for each other. Overall, the Siamese network trained on RGB-D data-set yields the best performance. Incorporate the Siamese network with multi-modal learning leads to more accurate representations for control.

6-2 Baseline models for information disentangling VAE

To evaluate the performance of the LSTM based VAE, we introduce a few baseline models.

- Jordan network:

  We first choose the Jordan network based auto-encoder as explained in Section 2-5. In our scenario, the network will have the structure shown in Figure 2-6. Note that representations are not split. We use two recurrent layers to model the forward dynamics with either plain recurrent cells or LSTM cells in our experiments.

- RNN based DAE (Denoising auto-encoder)

  DAE learns robust representations by reconstructing from corrupted data points as mentioned in Section 2-2-1. The learned representations are deterministic although the corruption process is stochastic. We thus introduce DAE as a comparsion to VAE with the consideration that the forward dynamic model can be totally stochastic. Thus the outcome of taking an action is a conditional distribution on the action and previous state. Note that in the last chapter, the introduced forward model first predicts the future pose representations and then samples from that representation statistics. It naturally learns a probabilistic forward model. In DAE, the predicted pose representations can be interpreted as an average outcome of the action (e.g., mean of all possible predicted representations). During the training, we choose a corruption noise level, generate a noise mask and randomly mask out input image pixels.

- LSTM based DAE

  Since we can change the number of prediction steps, we can examine the prediction performance of plain RNN cells and LSTM cells which are introduced in Seciton 2-5. The model structure used in our experiments is shown in Figure 6-2. This model is introduced as an comparison to RNN based DAE and LSTM based VAE.

- RNN based VAE

  We also train a version of VAE with plain RNN cells. So the LSTM cells in Figure 4-3 are replaced with plain RNN cells. The loss and training procedures stay the same.

- LSTM based VAE

  This is our proposed information disentangling VAE with LSTM networks. Note that we use two RNN or LSTM layers in all models.

6-3 Model implementations

In all five different models, we use the same structure for the encoder and decoder. The encoder and decoder are also reused across the time sequence. Details of each layer are shown in Table 6-1. Training a convolutional network on images with size 240×240 usually requires five to ten layers to transform the feature map to a small enough size. Then the amount of parameters will be considerable and we will need more data to train. Instead, we resize the input images to 64×64 and a small CNN with around five convolutional layers will be

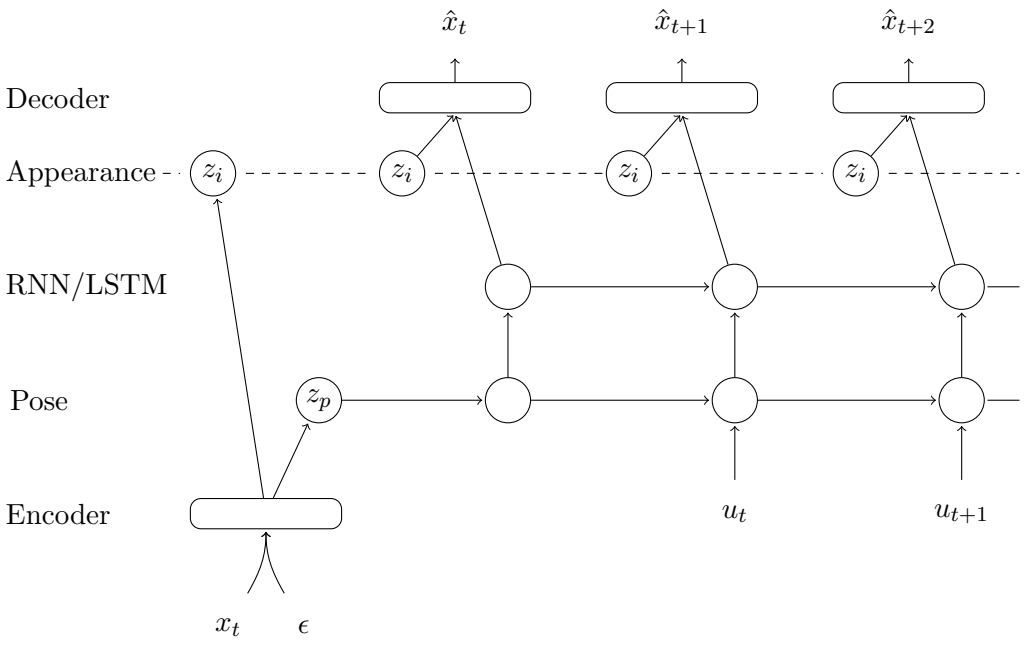


Figure 6-2: Structure of RNN or LSTM based DAE. Two layers of RNN or LSTM layers are applied in the models. The input data point x_t is corrupted with white noise ϵ.

Table 6-1: Common layer settings of the encoder and decoder.

Encoder	Decoder
5×5 3 conv, stride 1, ReLU	8×8×256 fully connected, BNorm, ReLU
5×5 64 conv, stride 2, BNorm, ReLU	upsampling stride 2
5×5 128 conv, stride 2, BNorm, ReLU	5×5 128 conv, stride 1, BNorm, ReLU
5×5 256 conv, stride 2, BNorm, ReLU	upsampling stride 2
1024 fully connected, BNorm, ReLU	5×5 64 conv, stride 1, BNorm, ReLU
512, 512 split (if not Jordan network)	upsampling stride 2
	5×5 3 conv stride 1, BNorm, ReLU
	5×5 3 conv stride 1, ReLU
RNN/LSTM (hidden state/cell state size 512)	
512 fully connected, tanh	
512 fully connected, tanh	

sufficient. We do not use large strides in the convolution layers. The reason is that we do not want to encourage translation invariance as the poked object in the image does not move very much.

In the intermediate layers of the encoder and decoder, we use the batch normalization [16] as an extra regularizer which normalizes the layer activation over each mini-batch of data. They are noted as BNorm in Table 6-1. With batch normalization layers, we can apply relatively larger learning rates in the training without decreasing the training stability. Note that data of actions in the collected poking data-set are in different ranges. We need to normalize the input actions into a range of $[0, 1]$. The normalized actions will then be close to the magnitude of layer activation in the intermediate layers with batch normalizations. For activation functions, we use the popular rectified linear unit (ReLU). The strength of ReLU is that the derivative of ReLU with respect to its input is a constant, making it less prone to the neuron saturation problem.

In convolutional layers, a stride of two halves the feature map size after the convolution. In the inner most layer of the encoder, feature maps are flattened and fully connected to a layer of size 1024. The outputs are split into two parts and the pose part is fed into the RNN/LSTM layers. The first decoder layer is fully connected and maps the activation from the encoder and RNN/LSTM into the correct size ($8 \times 8 \times 256$ in our case). The output activation from this layer can then be reshaped into the correct feature map size. In order to reconstruct the image, we need to gradually enlarge the feature maps. To do so, we can either use unpooling with switch variables as introduced in Section 2-4 or use upsampling directly. Since the poked object moves in the target reconstruction image, it does not make sense to unpool the feature map based on where it is pooled in the encoder. We do not use any pooling layers in our model. Instead, we use a fixed pattern of upsampling as introduced in [5] and [44].

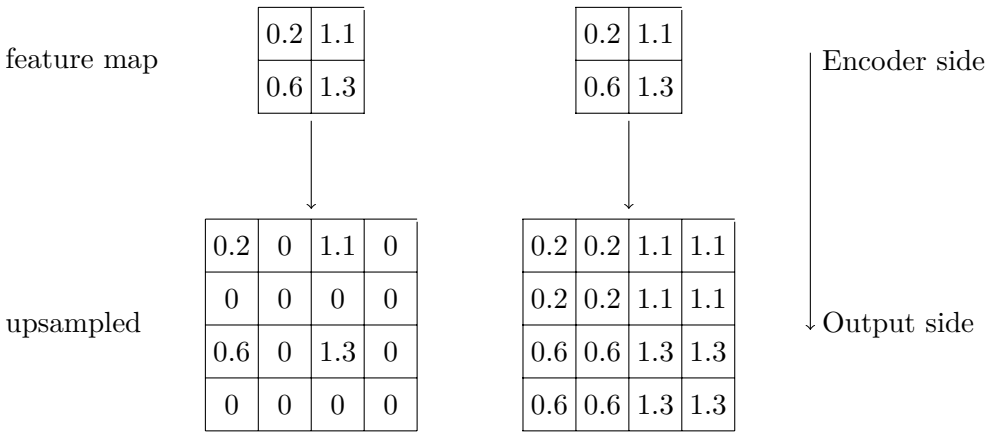


Figure 6-3: Upsampling with a stride of 2. Left: always put the elements from the feature map on predefined locations (e.g., top left). Right: copy the elements over the upsampled map.

During the training, we find that the optimization is slightly quicker with the "zero" sampling. Therefore we use zero sampling in our models. An upsampling of two enlarges the feature map by twice. In the encoder, the original image is halved three times to a size of 8×8. Thus feature maps can be transformed to the original size with three upsampling layers as shown in Table 6-1.

In our experiments, we have tried pose representations with dimensionalities of 128, 256 and 512. Experiments results do not indicate significant differences on performances. Therefore we report results where both the pose and appearance representations have a dimensionality of 512. In all five models, the hidden state size of RNNs/LSTM is 512. The LSTM cell has an extra cell state of size 512. During the training, we use the Adam optimizer [45] with a same learning rate and training epochs on all different models. We monitor the training process of DAE and VAE as shown in Figure 6-4. Both DAE and VAE are fitting the data-set. Note that the loss for VAE has a larger magnitude only because the loss is averaged over mini-batches while the loss is averaged over pixels for DAE.

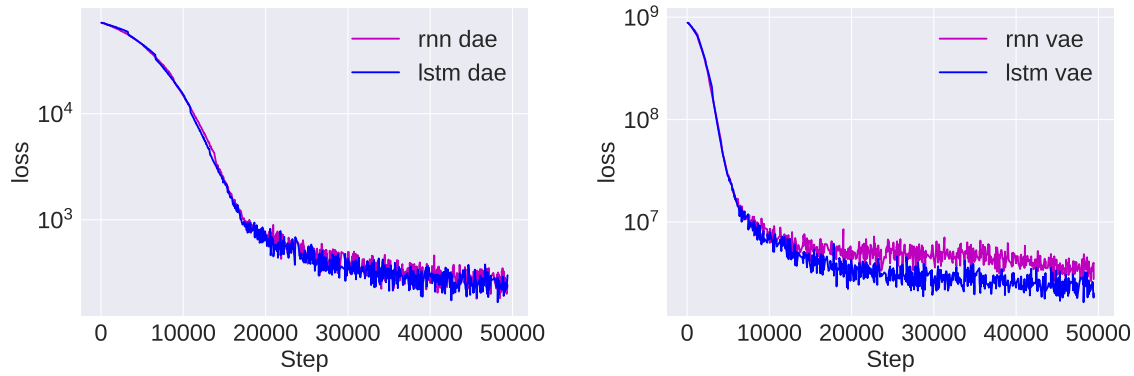


Figure 6-4: Left: Loss of DAE; Right: Loss of VAE.

6-4 Validation on the test set

We collect a test data-set separately which contains 20 runs with 1000 data tuples in total. Thus each run consists 50 consecutive data tuples that are collected sequentially. Each data tuple consists of an image pair and a poking action. Depending on the length of time steps of RNNs in our models, we can sample consecutive sequences of data tuples from the test set. All five trained models are validated on this test set. An example prediction of the cube movement is shown in Figure 6-5. Since predictions are in the image space, a suitable evaluation procedure is to check the pixel error around the ground truth location of the object. In Figure 6-5, we mark the ground truth object location in the current image as a red rectangle. The difference of RGB pixel values from the red rectangle patches are calculated as a prediction accuracy measure. During the test time, we query the ground truth position of the cube from the Gazebo simulator, choose a rectangle region of 10×10 pixels around the cube position and compare rectangle patches of the ground truth and the predicted images. We evaluate patch pixel errors over the test set and compare the mean and variance of prediction errors.

As a sanity check, a reasonable prediction of the cube movement is no movement at all since the cube does not move much after the poking. We first compare the validation results of Jordan type of networks with no movement predictions. Note that Jordan networks are all trained on sequential data of three steps. The box plot is given in Figure 6-6. It is clear that

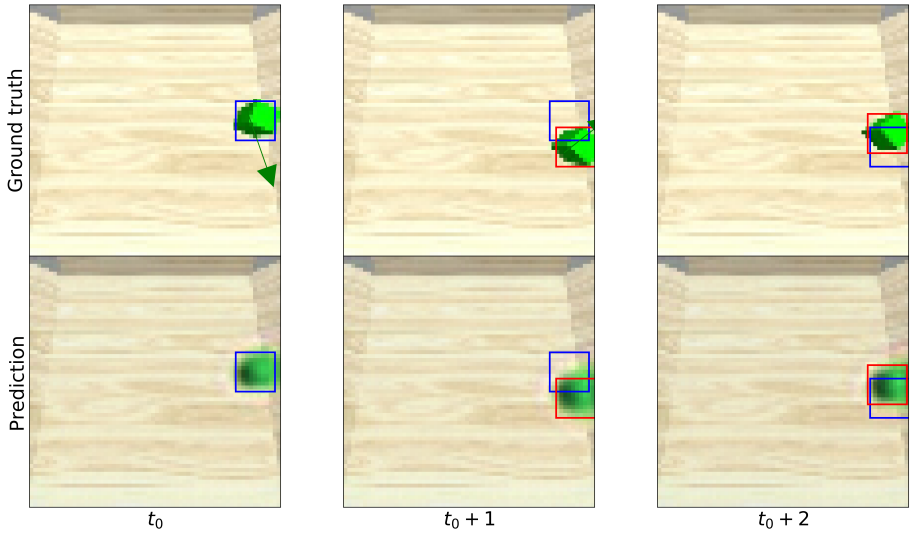


Figure 6-5: Top: ground truth; Bottom: movement predictions by the LSTM based VAE. Poking actions are marked as green arrows. Blue and red rectangles mark the ground truth cube positions at the previous and current time steps. When poked to the right, the block will fall back because of the table baffle. This nonlinear behavior is well captured by the network.

the Jordan type of networks do not produce satisfying predictions. This behavior is expected because that the prediction errors accumulate in the feed-forward process of the prediction. If we can reuse the encoded representation from the ground truth image, there is simply no need to encode new representations from reconstructed images.

We further compare the other four models where the pose and appearance representations are split. Models are trained and validated on sequential data of three time steps. As seen in Figure 6-7, models with LSTM cells predict better than models with plain RNN cells. This coincides with our motivation of applying the LSTM structure. The LSTM based VAE indeed yields the best prediction performance with the lowest mean and variance of pixel prediction errors. More prediction examples like Figure 6-5 can be found in the Appendix B-1.

Having compared LSTM based VAE with other baseline models, we further train it with longer prediction horizons. An example of 6 steps ahead prediction on the test data-set is given in Figure 6-8. By checking predicted image frames, we find that the cube movement predictions become more blurry as time steps grow bigger. Therefore it is hard to tell the orientations of the predicted cubes. This is a known problem with the l_2 loss for pixel reconstructions [46]. If the probability distribution of a predicted pixel has two equally possible modes p_1 and p_2, a value of $(p_1 + p_2)/2$ will minimize the reconstruction error. As time grows, the action outcomes become more uncertain and the blurry problem becomes more severe. This can be observed in the box plot of six steps ahead prediction errors in Figure 6-9. The negative reconstruction error term for VAE in Equation (4-5) amounts to squared errors in case of RGB images. This is the root cause that the prediction becomes more blurry as time grows. In general, it is hard to find a better loss than l_2 because it is easy to optimize with stochastic gradient descent methods. On the other hand, l_1 loss helps to generate sharper images but

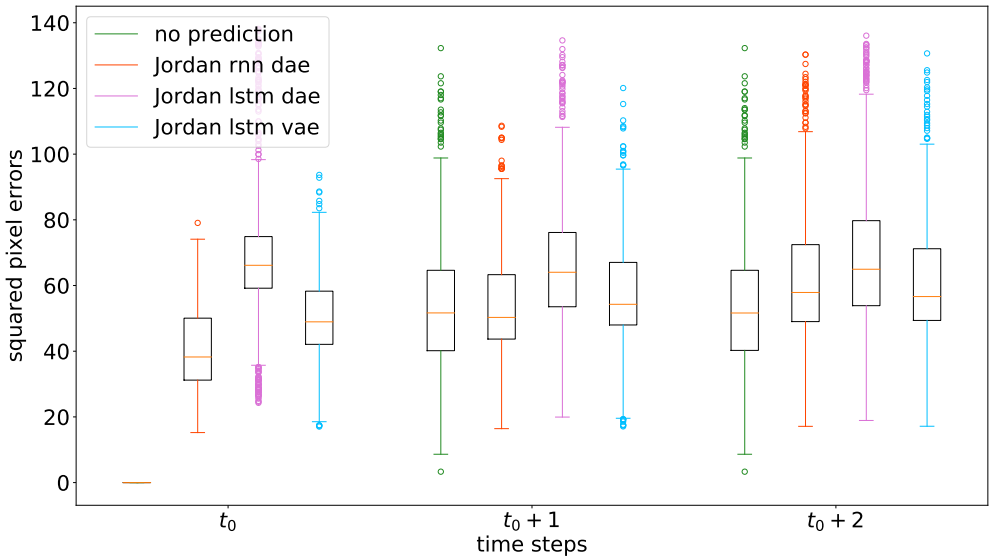


Figure 6-6: Prediction performances by Jordan type of networks.

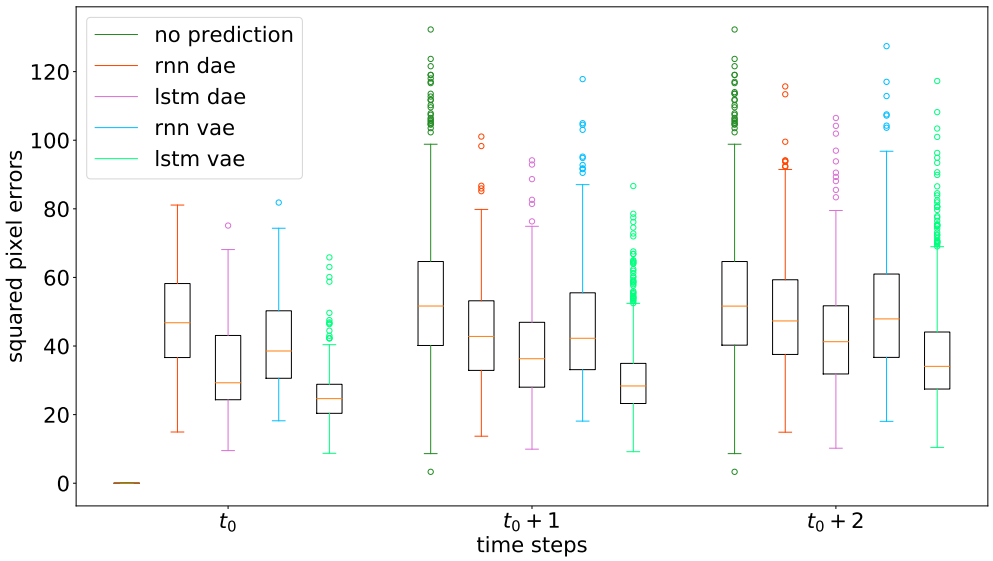


Figure 6-7: Prediction performances by models of which representations are split.

it is hard to optimize stably. Minimizing the mean squared error on pixel values produces over-smoothed reconstructions. Thus the orientation information expressed by cube edges is lost. We can only notice the prediction of positions.

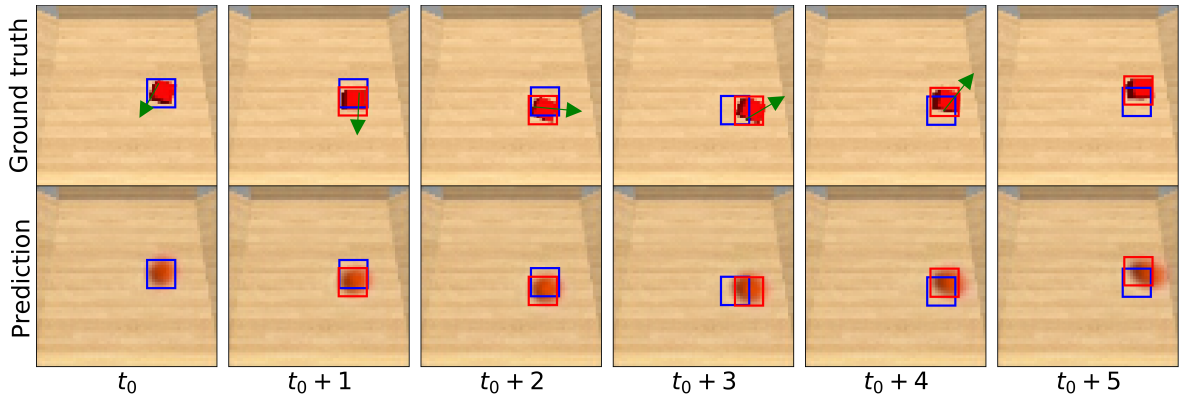


Figure 6-8: A six steps prediction given by the LSTM based VAE.

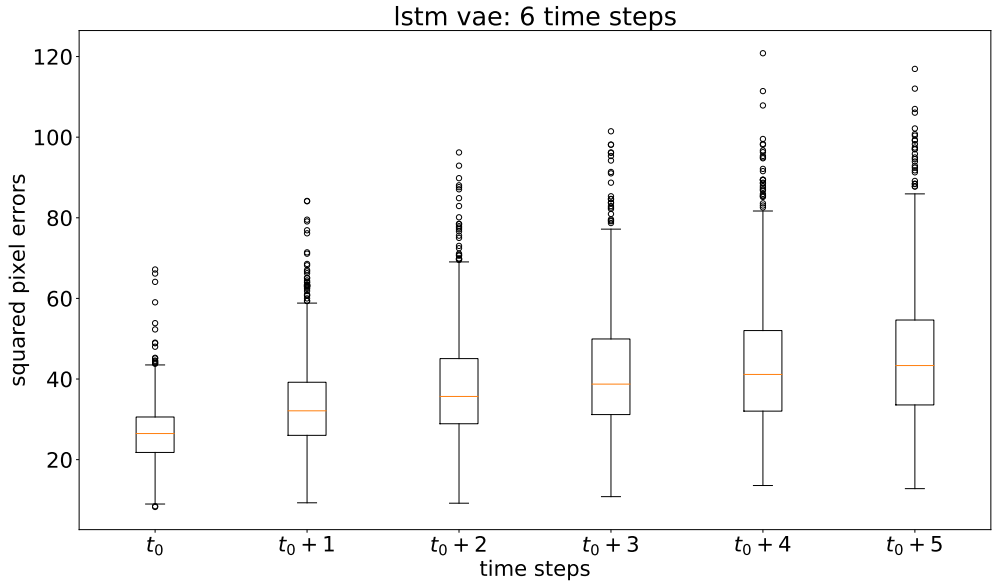


Figure 6-9: Six steps prediction errors by the LSTM based VAE.

6-5 Examination of information disentangling

After the training, we can sample from the posterior of the encoder separately for the appearance and the pose representation part. Based on the sampled codes, we can examine if the representations are disentangled.

6-5-1 Partial sampling from the encoder

Given an input image, our proposed LSTM based VAE encodes it into the appearance and pose representation z_i and z_p. Since we are using Gaussian priors, z_i is interpreted as a total vector of mean and variance of the representation as well as z_p. So we can perform partial sampling on images and input actions from the test set. Firstly, we sample from z_i but replace the sample from z_p with zeros. From the concatenation of samples and input actions, we reconstruct and predict the input images. We also replace the sample z_i with zeros but keep the sample from z_p. The reconstruction results are given in Figure 6-10. If the sample

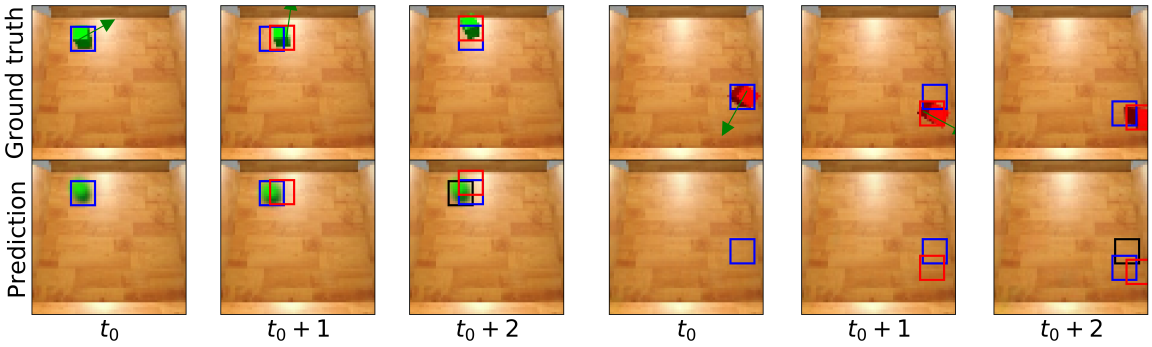


Figure 6-10: Left: only the sample from the appearance representation is kept. Right: only the sample from the pose representation is kept. Black rectangles at the last images mark the starting position of the time series and blue rectangle marks the position at the previous time step.

from the pose representation is discarded, the predicted cube does not move. If the sample from the appearance representation is discarded, the cube disappears in the image as seen on the right. This supports our hypothesis that the appearance representation contains the appearance information which is necessary for movement predictions.

6-5-2 Visualizations by t-SNE

To better interpret learned representations in a human readable way, we further utilize t-SNE [47] to visualize the appearance and pose representations separately. t-SNE is an optimization based dimensionality reduction technique. It aims at mapping the high dimensional data points to a lower dimensional space and preserving the similarities among data points as well as possible. Specifically, it minimizes the similarity distributions in both high and low dimensional spaces for each data point. For the visualization purpose, similar data points in the high dimensional space will also be close to each other in the low dimensional space. We apply t-SNE on the embedding of our information disentangling VAE.

For the LSTM based VAE explained in Section 6-4, the appearance and pose representations both have a dimensionality of 512. Note that this is the size before sampling. We visualize the appearance and pose representations separately by t-SNE. Firstly, we generate the appearance and pose representations by passing the test data-set into the trained network. We then apply PCA (Principal Component Analysis) on saved representations to reduce the dimensionality to 50. Subsequently, we use t-SNE on the principal components of the representation to

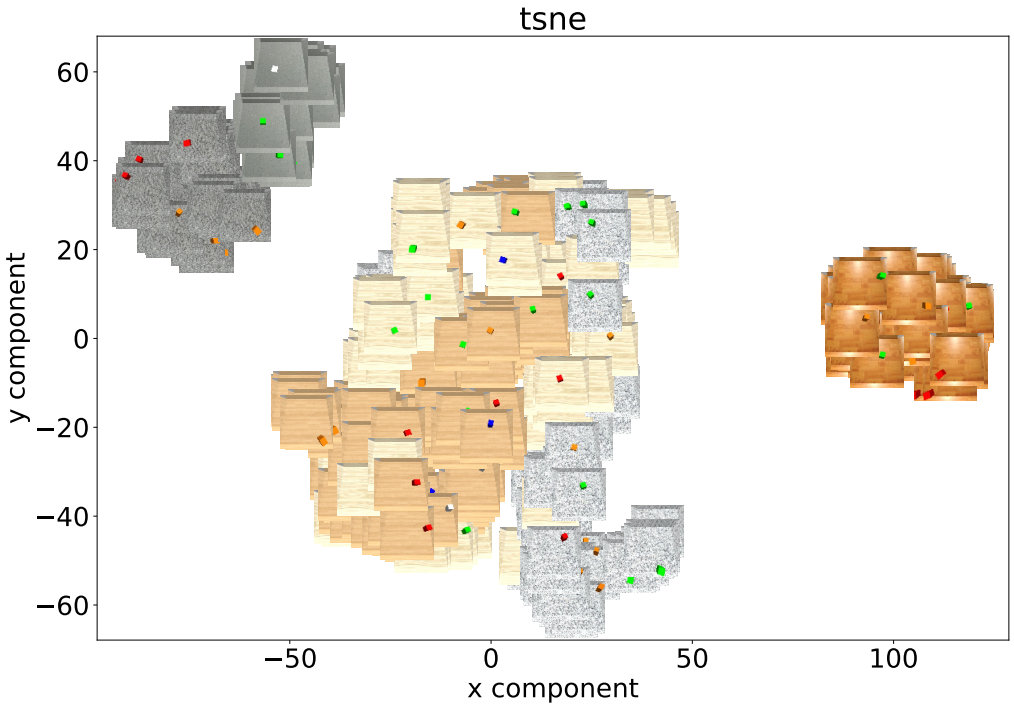


Figure 6-11: t-SNE visualization of the appearance representations.

reduce the dimensionality to 2. As a final step, we compute t-SNE components over the test set, retrieve test images and anchor them to their locations in the two dimensional space. The t-SNE visualization of the appearance representations is shown in Figure 6-11. Images are nicely grouped by their appearance, meaning that they are similar to each other by appearance in the data space. And this similarity is well captured by the appearance representations.

The t-SNE visualization of the pose representations is shown in Figure 6-12. For better visualizations, we do not include all images with different background colors and textures as the visualization looks cluttered. However, we visualize images with only different cube colors and positions. On the negative side of x component axis, cubes generally locate to the left of individual images. Cube positions gradually shift to the right as the x components grow bigger. Also, images with blue cubes and oranges cubes are grouped together at the top left of Figure 6-12 since cubes all locate close to the left boarder of individual images.

The test data-set is collected the same way as the training set. So cubes gradually move and do not jump around in images. As a result, t-SNE components are not well separated with enough gaps in between, we need to zoom in to see more grouping details. A complete visualization of various images and zoomed in details are given in Appendix B-2.

By visualizations, we can conclude that representations for appearances and cube poses in our cube poking task are indeed disentangled. Moreover, LSTM layers learn a forward dynamic model on the disentangled pose representations. The forward dynamic model is proven effective in predicting the cube movements.

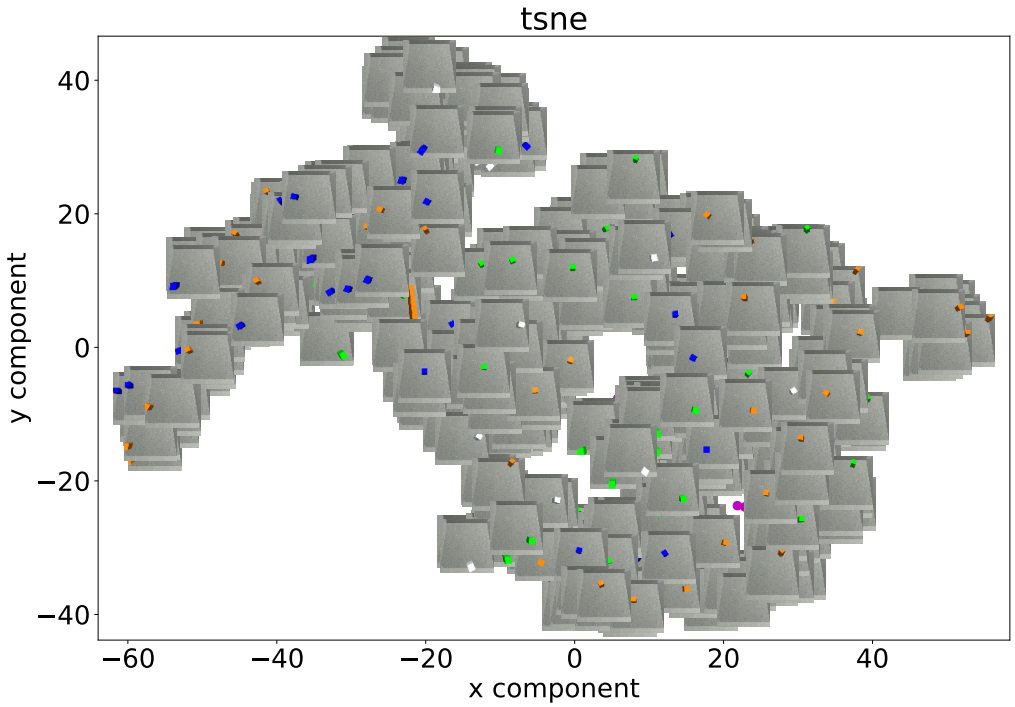


Figure 6-12: t-SNE visualizaion of the pose representations.

6-6 Discussion

Apart from experiment results shown here, we have taken a few measures to improve the prediction and planning results from different angles.

To address the blurry prediction problem, we tried to incorporate adversarial training which is known to be good at generating sharp images. The main idea is to add an extra discriminator to determine the ground truth and decoded images. Instead of using the l_2 loss, the difference of high level activation of the discriminator is utilized as an adversarial loss to the decoder (generator) and discriminator [48]. However, generative adversarial networks are also hard to train. We further used Wasserstein information distance measure with GAN as introduced in [49]. This measure however did not solve the problem. In generated images, objects are always missing. We suspect that objects are too small compared to the image canvas. The discriminator converges too fast to determine the images without colored cubes as fake before the generator can catch up.

We also try extra regularizer like the image gradient loss [46] as well as training on difference images [50]. In data collection phase, the camera model we use in ROS randomly injects white noises to images for more realistic data. Therefore gradient type of losses do not work very well on these data.

From the learned LSTM model, we planned to do a forward model planning as a comparison with the Siamese network based inverse model planning. Particularly, we tried iLQG[10] and cross entropy method (CEM) [51]. But planners do not yield reasonable poking actions. For

optimization in iLQG, approximated derivatives of the nonlinear LSTM model are required. This means the learned LSTM should be perturbed around some nominal trajectories. For CEM, the cost should be evaluated in the latent space by feeding in sampled actions from a multivariate Gaussian distribution into the learned LSTM. The problem is that the network is only trained on valid poke data for efficiency, which means that poking pixel locations (px_t, py_t) are always on cubes. The network never sees data that the cube stays static after poking in the air. Thus when we perturb or sample around a nominal trajectory, the behavior of the learned LSTM is unknown since the picked poking point are possibly not on the cube. In this case, the approximated derivatives or evaluated costs are not reliable. This issue should be solved by occasionally showing the network with invalid poking actions and static image predictions. Another issue is that representations in the latent space are not necessarily far away from each other when cubes locate far away in the data space. This makes the forward planning in the latent space difficult. This problem can potentially be solved by adding auxiliary tasks like regressing the pose of the cubes.

In summary, the first part of this chapter reports the result of multi-modal learning on the object poking task. We use trained Siamese networks to generate poke actions in simulations and compare pose errors. We find out that the multi-modal Siamese network trained on RGB-D data yields the best overall performance comparing object pose errors. The second part of this chapter aims at validating disentangled representations and predictions. A few baseline models for the LSTM based VAE are firstly introduced. Important implementation details are given for proper comparisons. We verify that the disentangled prediction model in the latent space enables multiple steps ahead predictions into the future. From comparisons, LSTM based VAE yields the best prediction result. We also specifically monitor the learned representations and confirm that they are indeed disentangled. These training results can all be all obtained in a self-supervision manner.

Chapter 7

Conclusions and recommendations

From experiment results on proposed solutions, this chapter concludes the study of state representation learning for robotic control. Section 7-1 summarizes the results and contributions and Section 7-2 gives some recommendations for future work.

7-1 Conclusions

In this thesis, we aim at learning state representations from image data for control. Specifically, we want to disentangle pose information from images and learn joint representations from RGB and depth images. Moreover, we intend to learn a prediction model on the disentangled pose representations. The difficulty lies in the fact that there is no explicit training objectives for information disentangling and all representations and prediction models should be learned without supervisions. Regarding these challenges, we split the representations of VAE and introduce LSTM for predictions. Predictions in the latent space are reflected in the data space by decoding which enable self-supervision type of training.

We can summarize this thesis as follow:

- A multi-modal planning scheme is introduced as a comparison to purely RGB based Siamese network for inverse planning. Two modalities specific Siamese networks and one cross modality Siamese network are trained with similar structures. By validating three trained models in poking simulations, we find that the RGB-D Siamese network performs the best. In particular, the extra modality of depth images helps the network to better differentiate orientations of task relevant objects. Thus the final orientation errors are smaller and less varied.
- A LSTM based VAE is proposed to learn disentangled representations from images. By treating the latent space as two separate set of variables, the model can effectively disentangle the pose and appearance information on our data-set. Specifically, the variations of appearances like background colors and textures are embedded in the appearance

representations and the poses of task relevant objects are embedded in the pose representations. The LSTM layers further learn a continuous and probabilistic transition model on the disentangled pose representations. During the testing, the learned LSTM is able to yield reasonable object movement predictions of image sequences. Such a continuous forward dynamic model enables multiple steps ahead predictions into the future. By comparing it with a few baseline models, we confirm that it outperforms other baseline models.

7-2 Future work

- A natural extension to our work is to apply LSTM based VAE on multi-modal data. Based on our experiment results, the extra depth modality can lead to a better representation of the object orientation. Adding extra depth modality, the predicted objects might have sharper edges. Moreover, it will be interesting to investigate cross modality predictions, e.g., predicting future depth frame from RGB representations and vice versa.

- Due to the time constraint, we have not been able to apply the same LSTM based VAE on more cluttered images and real world images. In our collected data set, there is alway only one cube in the scene. We should introduce more distractor objects in the scene and more variations on textures of objects to check how the network performs.

- We use the l_2 loss for image predictions because it is easy to optimize and more robust to different model structures. However, auto-encoders trained with l_2 loss reconstruct blurry pixels. The orientation information is lost in predictions of the network as reported. We have tried many methods to mitigate this problem. A different approach will be only modeling the pixel motions and reuse the appearance information. This suggests that we can apply the idea of optical flow in deep learning and explicitly model the flow of pixels. In this way, it is possible to get sharper predictions. However, we also want to claim that predictions yield by LSTM based VAE are stochastic. The outcome of a certain poking action is not deterministic either in our simulations. Thus the blurry predictions can be interpreted as all possible poses composited together as action outcomes.

- An important improvement is to show the network with invalid poking actions. So the network can predict the pose of the object to be static when the poking position is not on the object. This ensures that the transition model is consistent over the continuous state space. Another problem is that the poking angles of our poking data-set are randomly generated. The data-set thus contains lots of sequences in which cubes move back and forth without large displacements. A better way is to always poke in one direction and randomly change the direction. So the object moves a larger distance and the learned state space can be interpolated when objects are far away from each other. Another possible solution is to regress the object pose as an auxiliary task. These additional details can possibly improve the learned transition model such that it can be used for planning.

Appendix A

Data collections in ROS

A-1 Simulation system

The poking simulation is executed in Gazebo with a few ROS packages. An overview of the ROS graph is given in Figure A-1. A complete collection process is given in Algorithm 4.

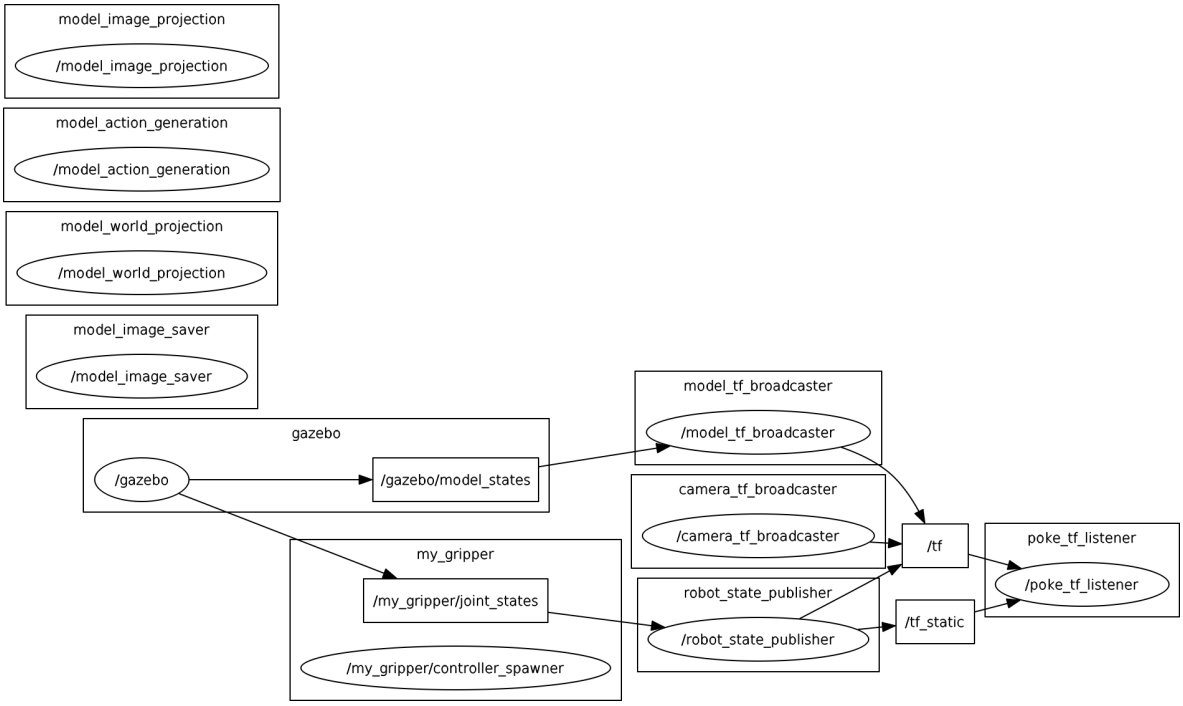


Figure A-1: Node graph of block poking simulation

- spawn_table, spawn_block:

Spawning randomly a table and a block from a block set and a table set into the simulation.

- spawn_urdf:

  Spawning the robot arm described in urdf (Unified Robot Description Format) format into the Gazebo simulation. The urdf file contains information about links, joints, coordinate frames and actuation.

  Since all poking actions are translations, an arm of three prismatic (sliding) joints is built to achieve translations in three directions of Euclidean coordinate frames. Moreover, a two-fingered gripper from Baxter Robot Simulator is added with a rotating joint to change the poking and gripping pose of the gripper. A illustration of the simulation and camera view is given in Figure 5-1. A simple arm like this can be easily controlled by four PID controllers (three prismatic joints and one rotating joint) to any Euclidean positions with a given poking pose. Thus it is unnecessary to apply an inverse kinematic solver in the joint space. This helps us to get a quicker simulation without bothering with the reachable work space of the robot. These nodes are stopped after spawning.

- camera_tf_broadcaster:

  The pose of the camera in world frame should be known before hand in order to perform any coordinate projections. A point in world frame should be first rotated to the camera frame and then projected onto the image plane based on a simulated pinhole camera model. Since the camera is fixed, the camera pose is published statically to ROS tf (transformation) tree at 20 HZ.

- model_tf_broadcaster:

  To keep track of the block pose, the pose should be constantly published. Moreover, we need to know the block position in order to generate meaningful poking actions. In Gazebo, poses of all models are published in topic */gazebo/model_states* as a big tuple. So model_tf_broadcaster subscribes to the topic, picks out the pose of the block and republishes it at the same frequency as the Gazebo topic.

- poke_tf_listener:

  The transformation of all coordinate frames are governed by the ROS package tf. In order to maintain a transformation tree of dynamic frames at any time instant, a listener should listen to all published transformations. It provides coordinate transformation service when a transformation request is sent.

- model_image_saver:

  To save images before and after poking, this service node should be ready to subscribe image topics and wait for exact one image message when a request is sent. This node waits for one message each from topic */my_gripper/camera1/rgb/image_raw/compressed* and */my_gripper/camera1/depth/image_raw*, transforms them into Opencv data types and saves them upon request. The image topics are published by ROS camera plugin. An extra image_transport package is used to compress the rgb image to shrink the file size.

- model_image_projection:

  To generate poke action position (px_t, py_t), the poking coordinates in the camera frame are projected onto the image plane utilizing a pinhole camera model. This node returns the projected pixel location. This service is also called when ground truth block positions on image plane is generated.

- model_action_generation:

  This node provides service for generating poking action on a block. In the block coordinate frame, a point is randomly picked on a random side of the block. Because the pose of the block is constantly published by /model_tf_broadcaster, the point can be rotated into camera frame by calling service from poke_tf_listener. The rotated coordinates are then projected as (px_t, py_t) by calling the service from model_image_projection. For poking angle and length (θ_t, l_t), they are randomly chosen within a certain range.

- model_world_projection:

  For the transformation from image frame to world frame, this node generates a vector pointing from the camera center to the pixel location in the world frame. This is mainly useful in validation. For example, the pixel location predicted by a trained deep network can be projected back to the world frame for further usage. This node generates a uniform vector in camera frame, call the transformation service from poke_tf_listener and transforms it into world frame.

Algorithm 4 Data collection process of the central node.

Require: m: number of collection runs; n: Number of pokings per run; B: A set of objects; A: A set of tables with different appearances; $i = t = 0$: iterator; p: data collection path.

Spawn the robot arm.
Spawn an object O, a table T randomly chosen from B, A.
Initialize p.
RGB_0, $DEPTH_0 \leftarrow$ Call */model_image_saver*(p, 0).
while $i < m$ **do**
  while $t < n$ **do**
    Poking position (xp_t, yp_t, zp_t), $(\theta_t, l_t) \leftarrow$ Call */model_action_generation*(O).
    $(px_t, py_t) \leftarrow$ Call */model_image_projection*(xp_t, yp_t, zp_t)
    (x_t, y_t, z_t), $(a_t, b_t, c_t, d_t) \leftarrow$ Call */model_tf_broadcaster*()
    Save: action $(px_t, py_t, \theta_t, l_t)$, ground truth (x_t, y_t, z_t), (a_t, b_t, c_t, d_t).
    Approach vertically and horizontally to object O.
    Poke object O at (xp_t, yp_t, zp_t) with (θ_t, l_t).
    Retract the arm.
    $RGB_t, DEPTH_t \leftarrow$ Call */model_image_saver*(p, t)
    $t+ = 1$
  end while
  Delete O and T from simulation.
  Spawn O and T randomly chosen from B and A.
  RGB_0, $DEPTH_0 \leftarrow$ Call */model_image_saver*(p, 0).
  $i+ = 1$, $t = 0$
end while

A-2 Services and messages

A summary of service and message type of above nodes are given in Table A-1.

Table A-1: Service and message types of important nodes.

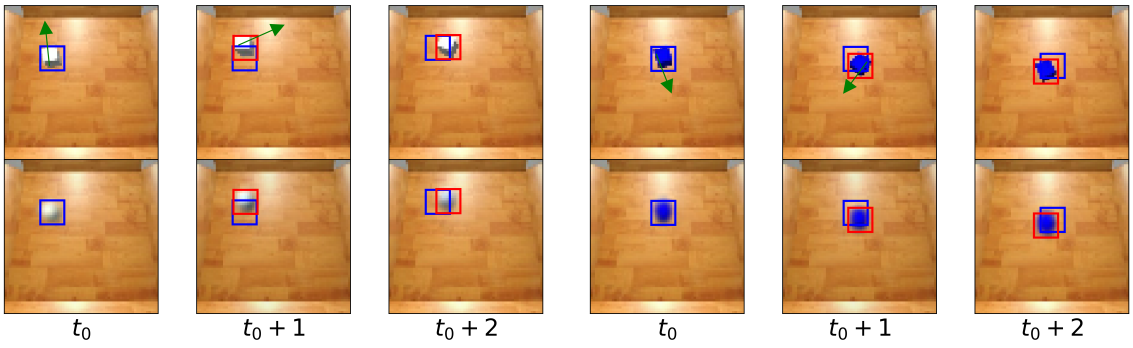
model_tf_broadcaster	
Type	Subscriber
Topic	*/gazebo/model_states*
poke_tf_listener	
Type	Service: Transform pointIn to pointOut in frame targetFrame.
name	/poke_tf_listener/coordinates_transform
Request message	geometry_msgs/PointStamped pointIn
	string targetFrame
Response message	geometry_msgs/PointStamped pointOut
model_image_saver	
Type	Service: Save image with number ImgNumber to path SavePath.
Service name	/model_image_saver/image_save
Request message	string SavePath
	int64 ImgNumber
Response message	bool success
model_image_projection	
Type	Service: Project point (x, y, z) onto image plane with pixel units (x, y).
Service name	/model_image_projection/image_projection
Request message	float64 x, float64 y, float64 z
Response message	float64 x, float64 y
model_action_generation	
Type	Service: Generate a poking position pointOut, angle θ and length l.
Service name	/model_action_generation/action_generation
Request message	string ObjectName
Response message	geometry_msgs/PointStamped pointOut
	float32 theta, float32 l
model_world_projection	
Type	Service: Generate a directional vector (x, y, z) from pixel (x, y).
Service name:	/model_world_projection/world_projection
Request message	float64 x, float64 y
Response message	float64 x, float64 y, float64 z

Appendix B

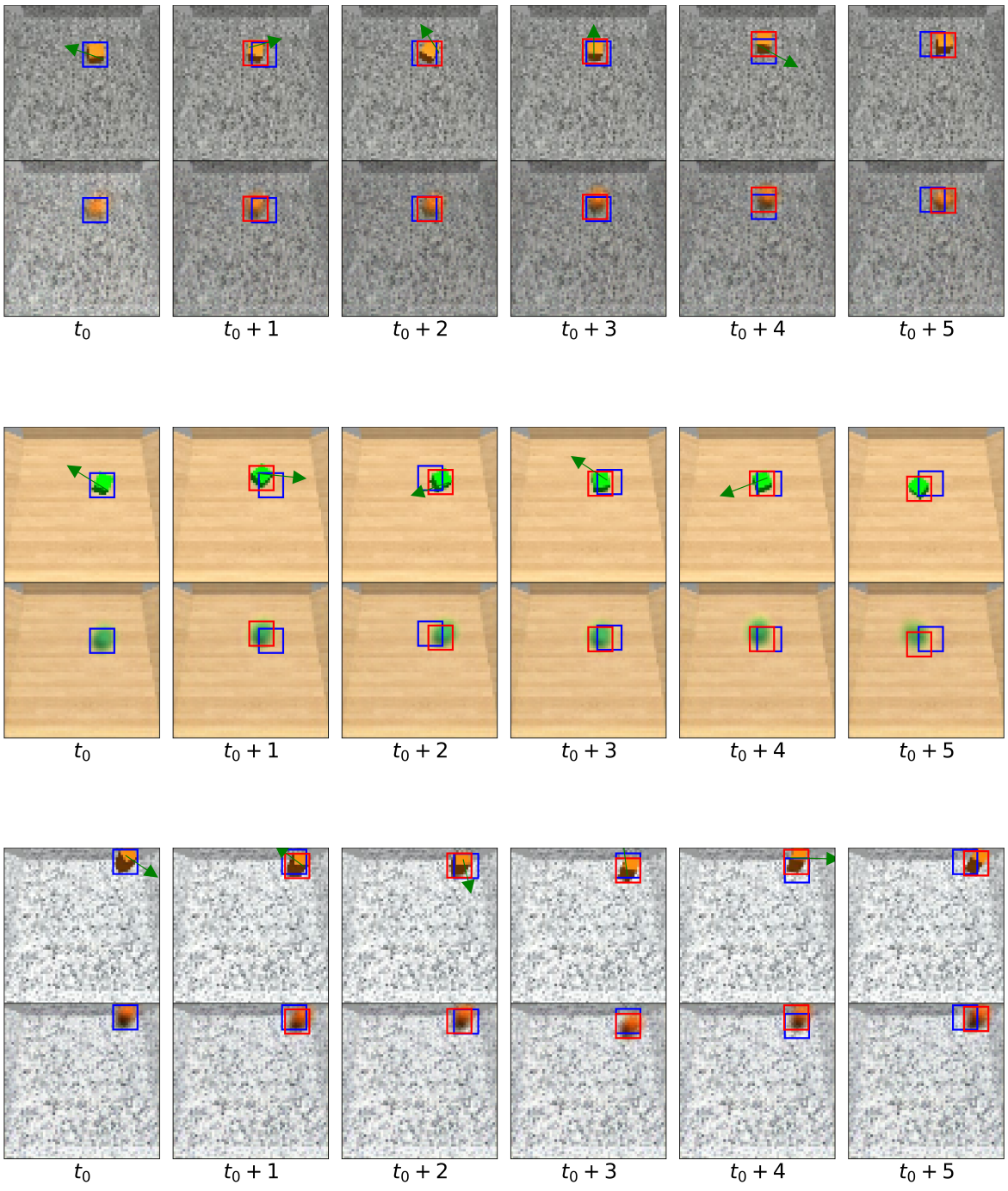
Validations of trained models

B-1 More prediction results

Several three steps ahead prediction results of LSTM based VAE are given here. Ground truth images and predictions are shown at top and bottom rows. Red rectangles highlight the ground truth positions at current time steps. Except for the first time step, blue rectangles highlight the ground truth positions of previous time steps.



Six steps ahead prediction results are given as follow:



B-2 t-SNE visualizations

Figure B-3 shows a complete t-SNE visualization of pose representations of all images from the test set. Images are not grouped by the background color and texture but by the position

of cubes. In Figure B-4, images that are grouped together have similar locations of cubes. But the background textures and cube colors differ from each other.

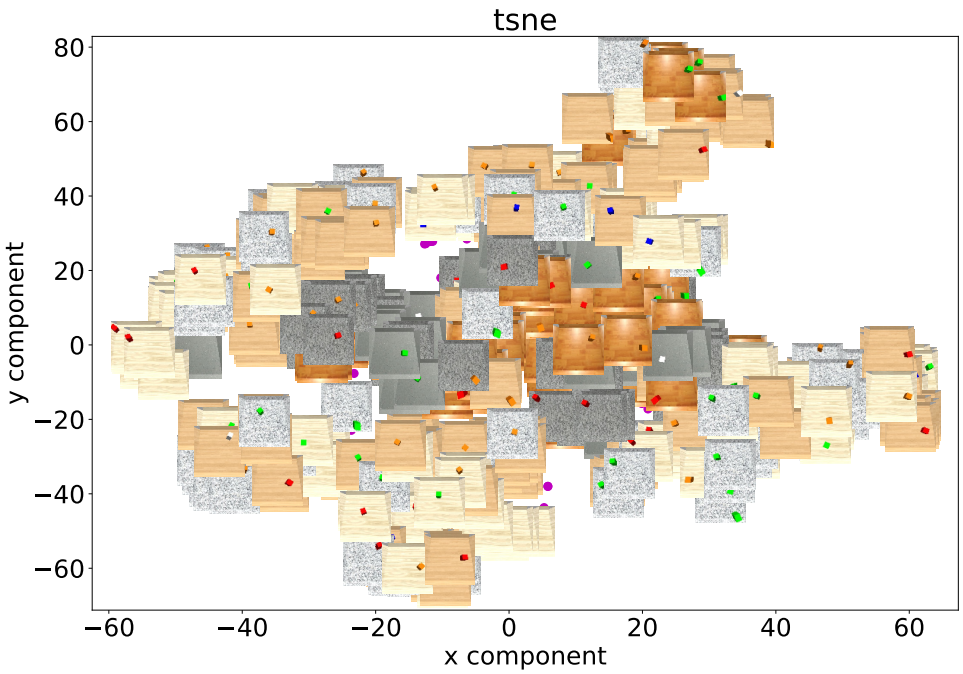


Figure B-3: A complete t-SNE visualization on pose representations from the test set.

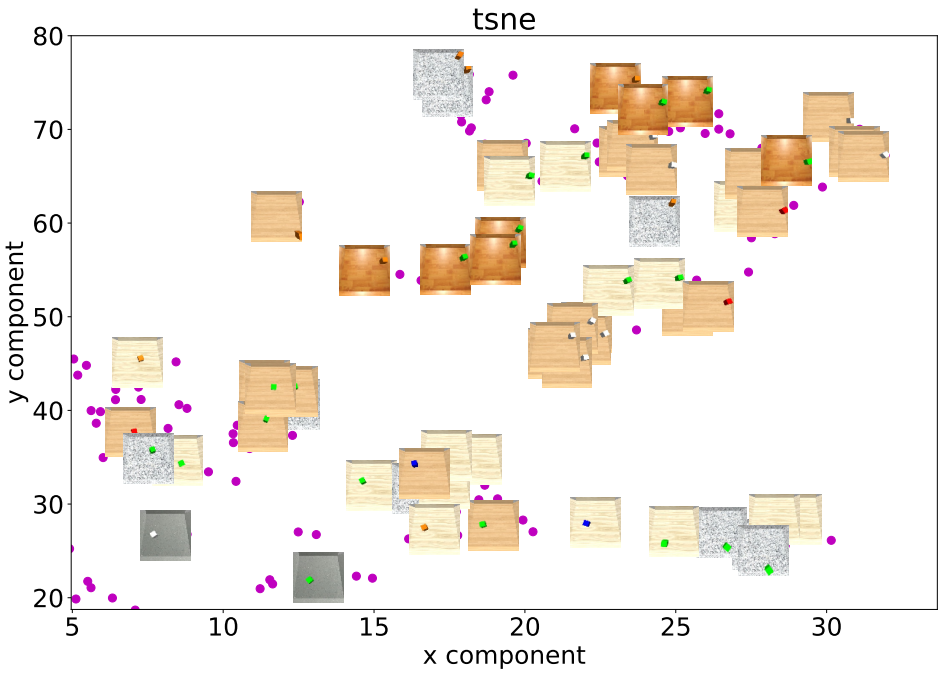


Figure B-4: Top right part of Figure B-3. Purple dots are individual data points of which the source images are not shown.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[3] C. Olah, "Understanding lstm networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[4] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016.

[5] J. Yang, S. E. Reed, M.-H. Yang, and H. Lee, "Weakly-supervised disentangling with recurrent transformations for 3d view synthesis," in *Advances in Neural Information Processing Systems*, pp. 1099–1107, 2015.

[6] R. E. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[7] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[8] G. Tesauro, "Td-gammon: A self-teaching backgammon program," in *Applications of Neural Networks*, pp. 267–285, Springer, 1995.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[10] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *American Control Conference, 2005. Proceedings of the 2005*, pp. 300–306, IEEE, 2005.

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

[12] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 59, no. 4, pp. 291–294, 1988.

[13] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1986.

[14] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural computation*, vol. 7, no. 6, pp. 1129–1159, 1995.

[15] 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*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456, 2015.

[17] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

[18] A. Hyvärinen, "Estimation of non-normalized statistical models by score matching," *Journal of Machine Learning Research*, vol. 6, no. Apr, pp. 695–709, 2005.

[19] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation*, vol. 23, no. 7, pp. 1661–1674, 2011.

[20] G. Alain and Y. Bengio, "What regularized auto-encoders learn from the data-generating distribution," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3563–3593, 2014.

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

[22] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*, 1995.

[23] 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*, pp. 1–9, 2015.

[24] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535, IEEE, 2010.

[25] M. Bodén, "A guide to recurrent neural networks and backpropagation," in *IN THE DALLAS PROJECT, SICS TECHNICAL REPORT T2002: 03, SICS*, Citeseer, 2002.

[26] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[27] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[28] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[29] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[30] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[31] 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*, pp. 2746–2754, 2015.

[32] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: Policy learning with deep dynamical models," *arXiv preprint arXiv:1502.02251*, 2015.

[33] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 689–696, 2011.

[34] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[35] K. Sohn, W. Shang, and H. Lee, "Improved multimodal deep learning with variation of information," in *Advances in Neural Information Processing Systems*, pp. 2141–2149, 2014.

[36] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 3928–3934, IEEE, 2016.

[37] L. Pinto and A. Gupta, "Learning to push by grasping: Using multiple tasks for effective learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2161–2168, IEEE, 2017.

[38] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 539–546, IEEE, 2005.

[39] M. Kawato, "Internal models for motor control and trajectory planning," *Current opinion in neurobiology*, vol. 9, no. 6, pp. 718–727, 1999.

[40] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *arXiv preprint arXiv:1511.00561*, 2015.

[41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

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

[43] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 3406–3413, IEEE, 2016.

[44] J. Zhao, M. Mathieu, R. Goroshin, and Y. LeCun, "Stacked What-Where Auto-encoders," *ArXiv e-prints*, June 2015.

[45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[47] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[48] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv preprint arXiv:1512.09300*, 2015.

[49] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[50] T. Xue, J. Wu, K. Bouman, and B. Freeman, "Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks," in *Advances in Neural Information Processing Systems*, pp. 91–99, 2016.

[51] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

Glossary

List of Acronyms

AE	Auto-encoder
DAE	Denoising Auto-encoder
CAE	Contractive Auto-encoder
VAE	Variational Auto-encoder
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks
LSTM	Long Short-term Memory
NARX	Non-linear Auto-regressive Exogenous
MPC	Model Predictive Control

List of Symbols

μ	The mean of a Gaussian
ϕ, θ, ψ	Parameters of the recognition, generation and forward transition models
σ	The standard deviation of a Gaussian
$\hat{X}$	Reconstruction variable
$\tilde{X}$	Corrupted data variable
$C(\tilde{X}\|X)$	Conditional distribution of data corruptions
D_{KL}	Kullback-Leibler divergence
f_ψ	The forward model in the latent space
$p_\theta(X\|Z)$	The generative model of VAE

$q(X)$	Data distribution
$q_\phi(Z\|X)$	The recognition model of VAE, an approximated posterior
X	Observation variable in the data space
Z	Latent variable
Z_i	Latent variable of the appearance representation
Z_p	Latent -variable of the pose representation