SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Curriculum Learning for Transferable Deep-State Space Models

Tai Hoang

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Curriculum Learning for Transferable Deep-State Space Models

Curriculum-Lernen für übertragbare Deep-State-Space-Modelle

Author:	Tai Hoang
Supervisor:	apl. Prof. Dr. Georg Groh
Advisor:	Dr. Maximilian Karl
Submission Date:	17.04.2023

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 17.04.2023 Tai Hoang

Acknowledgments

I would like to express my appreciation and deepest gratitude to my advisor, Dr. Maximilian Karl, for his guidance, support, and mentorship throughout my thesis work. I feel really fortunate to work with you, more like a colleague than a student. Thank you for giving me the freedom to try out new ideas and all the insightful feedback. Without them, this thesis would not have been completed.

I would also like to extend my sincere thanks to apl. Prof. Dr. Georg Groh for his supervision and organisation. Special thanks go to Prof. Dr. Patrick van der Smagt for giving me such an exceptional opportunity to work at the Volkswagen Machine Learning Research Lab.

Last but not least, I would also like to thank all my lab mates from the Volkswagen Machine Learning Research Lab for their support, feedback and discussions and especially for creating such a great working environment.

Abstract

Deep state space model (DSSM) is used for representing the dynamics of a system. Having a good DSSM helps easing the complexity of the downstream tasks, e.g policy optimisation, planning or control. However, due to the rate-distortion theorem, learning a good DSSM is a notoriously hard problem. Finding the correct balance point that achieves an optimal trade-off between rate and distortion usually requires a huge amount of samples and long training time, which can be expensive in real-world robotic tasks. We therefore ask ourselves what if we have a pre-trained sequence model by design? Would it be possible to adapt this knowledge to a more difficult dynamical system? With that in mind, in this thesis, we develop a methodology based on Bayesian update principle that allows reusing the learned deep state space model for more challenging systems. Experiments on simulated robots show that a learned DSSM can be transferred from one environment to the others that have different observational inputs or even dynamical systems.

Contents

Acronyms

DMC DeepMind Control Suite. 20

DSSM Deep State Space Models. v, 1–3, 14, 16, 21–23, 33

DVBF Deep Variational Bayes Filters. 1, 2, 10, 15, 17, 21, 33–35

ELBO Evidence Lower bound. 1, 2, 5, 7, 8, 10, 23

LVM Latent Variable Models. 8–10, 18

MBAC Model-based Actor Critic. 14, 29, 30, 43

MDP Markov Decision Process. 10, 12

PI Policy Iteration. 12

POMDP Partial Observable Markov Decision Process. 10

REWO Reconstruction-error-based weighting of the objective function. 19, 21

SSM State Space Models. 1, 8–10, 17, 18

VAE Variational Auto-Encoders. 1, 6, 10

VI Value Iteration. 11

1. Introduction

The underlying dynamics of a robot is a mathematical expression used to describe its physical motion over a certain period given an initial condition. Indeed, all the robot's physical properties are encapsulated within a set of dynamical parameters, and depending on the environment and the robot itself, they can be stochastic in nature. A wrong value can lead to catastrophic outcomes: a robot can either harm the environment or be damaged itself. Therefore, identifying the system dynamics is considered a challenging problem.

Although this problem can be tackled by either engineering-based or learning-based methodologies, this thesis will put the focus on the learning-based approaches. To this end, we will heavily exploit the State Space Models (SSM). State-space models is a probabilistic generative model used to describe a sequence of states in a discrete-time setting. Transition model and observation model are the two main components of the SSM, and in robotics scenario, they correspond to the system dynamics and sensor modalities, respectively. DSSM is a specific type of SSM where both the transition and observation models are approximated by deep neural networks. Inference in SSM is a mathematical process to estimate the latent states based on sequences of observed data. At the time where deep neural networks have become popular, Kingma, et al. (2014) [1] has developed a method called Variational Auto-Encoders (VAE) based on variational inference principle to simultaneously do inference and learning. Later, Karl, et al. (2017) [2] extended this scheme for DSSM, and named their method as Deep Variational Bayes Filters (DVBF). In this thesis, we will discuss the problem of training DSSM on complex systems and propose a new approach based on curriculum learning fashion to expedite the training process while maintaining high performance.

1.1. Motivation

Under latent variable model setting, the likelihood is generally intractable to compute. Variational inference is a prevailing approach to resolve this issue. It works by introducing a new approximated posterior and then aiming to minimise the Kullback-Leibler divergence between this and the true posterior. This objective can be understood as the Evidence Lower bound (ELBO) of the true likelihood and consists of two terms: the reconstruction and recognition loss. The former measures the quality of the learned generative model and the other makes sure the approximated posterior does not diverge too far from the prior distribution. In most cases, there is a trade off between them, finding a good balance point therefore can be difficult. One way to resolve this issue is resorting to an additional hyperparameters β, which is introduced to rescale the importance of the two terms [3], [4]. However, in practice, finding a good β is also not an easy task. One can do annealing, i.e

in the beginning of the training process, we emphasise more on the reconstruction loss by setting a very small β, note that the objective is now the scaled ELBO. Then, we linearly increase it until the bar is hit i.e, $\beta = 1$ and we get back to the original ELBO. However, for more complex observations, e.g images, annealing does not help much. Recently, Rezende, et al. (2018) [5] formalised the ELBO maximisation as a constrained optimisation problem, and then later Klushyn, et al. (2021) [6] extended this for training DSSM. This formalisation helps avoiding hand-tuning the β, and was empirically and theoretically proven to give tighter ELBO and was reported to achieve good performance on images data.

Apart from the aforementioned issues, training DSSM also consumes a lot of time and data and gathering data is not always an easy task, particularly in robotics scenarios. It motivates us to seek an answer for the question "Is it possible to transfer knowledge from one DSSM to the others?". To be more specific, in DVBF inference model, all the network architectures including encoder, decoder and transition models are fixed. It is apparent that the DSSM trained on simulation data would not directly work in the real-world scenario. However, the latent manifolds in the two worlds should be in a similar shape as they are supposed to be generated from a similar system dynamics. Drawing from this idea, we will develop a training scheme based on Bayesian update principle that allows transferring the learned DSSM from one environment setup to the others, in a curricula fashion. The next two subsections will list several related works in this direction and show our main contributions in this thesis.

1.2. Related Work

Transfer learning Transferring knowledge from one environment to the others is a common way to accelerate the training progress. It can be categorised as lifelong learning [7, 8, 9], continual learning [10, 11, 12] or curriculum learning [13, 14, 15] in case the learner follows a specific curriculum during training which can be either pre-defined or learned. In general, transfer learning ideas are widely applicable in many domains, for example, in case of reinforcement learning, there are plenty of research tried to learn an agent that can work on multiple tasks and/or environments without starting from scratch [8, 9, 16, 17]. In the robotics domain, sim to real transfer [18, 19, 20] is a popular approach to alleviate the hardship of doing robotics control on real systems. Most of the successful sim-to-real stories rely on domain randomisation technique [21]: given a set of domains generated in simulation, it aims to learn a policy that is able to adapt to the changes of environment which afterwards can be transferred to the real robot.

Continual learning For deep neural networks, fine-tuning was one of the most popular approaches to do transfer learning [22, 23, 24]. The idea behind is very straightforward: instead of starting from scratch, one can use a pre-trained network and only retrain the essential part of the network, typically a few top layers while letting others be frozen. However, the transfer only happens in the initialisation step, which will result in a common problem called catastrophic forgetting. To overcome this, progressive networks [8] was

proposed: whenever a new task is added, the network architecture changes accordingly to have more capacity for it. Continual learning is another prevailing line of research, mainly for this issue. As opposed to the progressive networks, the architecture is fixed and instead a constraint is introduced to force the parameters stay close to the optimal pre-trained one [10, 11]. In principle, this formalisation can be interpreted as the Bayesian update, and there are several methods were proposed to solve this problem e.g, with second order posterior approximation [10, 12, 25], or via variational inference [11].

Bayesian update Bayesian update is a probabilistic framework proposed to do inference given a generative model. By following the Bayes' rule, the parameters are updated by maximising the posterior, which consists of both the current likelihood and the prior distribution. This prior can be changed as the training progresses, and be used as a representation for the distribution of the previous task. This formula can be interpreted as the Kullback-Lieber divergence between the two likelihood distributions conditioned on the new parameters and the old one (prior). This KL divergence form has been adopted in many frameworks, including model-free policy search [26, 27], model learning [28] and continual learning [10, 11, 12]. In this thesis, we will extend this principle for the DSSM inference problem and show that it is computationally efficient and more powerful than just pre-training and fine-tuning.

1.3. Contributions

The contributions of this thesis are outlined as follows:

- We propose an inference network architecture that can be reused for varying input dimensions, i.e, one DSSM can be shared for different environments, e.g pendulum, cart pole or even image pendulum.
- To train this newly introduced model, we follow the Bayesian update principle. In this setting, it can be interpreted as a constrained optimisation problem where apart from the ordinary likelihood, we also enforce the DSSM parameters stay close to the old pre-trained one.
- To show the benefits of reusing the pre-trained models, in the experiment section, we show the comparison between the pre-trained and non pre-trained models on the sequence model training task in both offline and online settings.

The rest of this thesis is organised as follows. In Chapter 2, we will recall the foundation knowledge. More specifically, in the first half, latent variables model and state-space model will be discussed, while the other half will be allocated for reinforcement learning. Chapter 3 describes in detail our proposed methodologies. The experimental setup and validation results will be presented in Chapter 4 and Chapter 5, respectively. Finally, we conclude the thesis in Chapter 6.

2. Foundations

In this chapter, the theoretical foundations will be discussed. The main topics are state-space models inference and reinforcement learning. In the first half, we start by introducing the latent variable-models, and then dive deeper to the most recent prevailing inference method - variational auto encoders and its extension on state-space models. Then, several reinforcement learning foundation algorithms will be recalled in the other half. In the end, we show how the two parts are linked together in the model-based actor-critic algorithm.

2.1. Latent Variable Models

In probabilistic generative modeling, given a set of observed data $\mathcal{D} = \{x_i\}_{i=1}^N$ that are identically independently distributed (i.i.d) from a true distribution $p^*(x)$. Our goal is to find an approximate distribution $p_\theta(x)$ parameterised by θ:

$$p_\theta(x) \approx p^*(x)$$

The true distribution can be arbitrarily complex and hence is not easy to approximate. To this end, by introducing a new latent variable z, which forms a hierarchical structure on the approximate distribution, we allow $p_\theta(x)$ to be more expressive. This variable is assumed to capture the essence of the data; however, is not observable. Overall, the distribution $p_\theta(x)$ becomes:

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz$$

The prior distribution $p(z)$ is normally simple and given, thus there is no need to encode a parameter to it. The other one, parameterised by θ, $p_\theta(x|z)$ is often referred to as the conditional likelihood distribution. Given those two distributions, data can be generated by the following process:

$$z \sim p(z) \longrightarrow x \sim p_\theta(x|z) \tag{2.1}$$

On the other hand, the learning process is the process of determining θ from the given $\mathcal{D}$. As the data is i.i.d, we can write the likelihood distribution as follows:

$$p_\theta(\mathcal{D}) = \prod_i p_\theta(x_i)$$

and the optimisation process to maximise the log-likelihood is:

$$\theta^* = \arg\max_\theta \log p_\theta(\mathcal{D}) = \arg\max_\theta \sum_i \log p_\theta(x_i) \tag{2.2}$$

Given the structure of latent variable models, the marginal log-likehihood is written as:

$$\sum_i \log p_\theta(x_i) = \sum_i \log \int p_\theta(x_i|z)p(z)dz \tag{2.3}$$

This Equation 2.3 is however often intractable to compute since it requires solving the integral on the domain that can be continuous for every data point. In the next section, we will introduce the prevailing methodology to find the optimal parameter θ^* in Equation 2.2 without the need to compute the intractable distribution $p_\theta(x)$.

2.2. Variational Auto Encoders

2.2.1. Variational Inference

Given the generative model in Equation 2.1, the process of finding the latent variable z from the observed data x is called "inference". Apply the Bayes's formula, we have:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)} = \frac{p_\theta(x|z)p(z)}{\int p_\theta(x|z)p(z)dz}$$

Due to the intractability of $p_\theta(x)$ introduced in the previous section, computing the posterior distribution $p_\theta(z|x)$ is thus also infeasible. In variational inference, the posterior is instead approximated by a variational distribution $q(z) \in \mathcal{Q}$ selected from the set of family distributions $\mathcal{Q}$. To this end, we first rewrite the Equation 2.3

$$\begin{aligned}
\log p_\theta(x) &= \log \int p_\theta(x|z)p(z)dz \\
&= \log \int p_\theta(x|z)p(z)\frac{q(z)}{q(z)}dz \\
&= \log \mathbb{E}_{z\sim q(z)}\left[\frac{p_\theta(x|z)p(z)}{q(z)}\right] \\
&\geq \mathbb{E}_{z\sim q(z)}\left[\log p_\theta(x|z)\right] - \mathrm{KL}(q(z)||p(z)).
\end{aligned} \tag{2.4}$$

In the final step 2.4, we applied the Jensen's inequality for the concave function $\log(\cdot)$, and the resulted equation is often referred to as the *ELBO*:

$$\mathcal{L}_{\mathrm{ELBO}} = \mathbb{E}_{z\sim q(z)}\left[\log p_\theta(x|z)\right] - \mathrm{KL}(q(z)||p(z)).$$

Put together, the final variational optimisation problem becomes:

$$q_i^*, \theta^* = \arg\max_{q_i,\theta} \mathbb{E}_{z\sim q_i(z)}\left[\log p_\theta(x_i|z)\right] - \mathrm{KL}(q_i(z)||p(z)). \tag{2.5}$$

2.2.2. Amortised Inference with Neural Networks

The optimisation problem in Equation 2.5 can be solved by an iterative fashion, i.e *Expectation-Maximisation* (EM) algorithm. However, every latent data point z_i inferred from x_i is generated by a corresponding individual distribution $q_i(z_i)$. This hinders us from doing inference on large datasets. Amortised inference is proposed as a solution to this problem, the main idea is, instead of modeling a set of individual distributions $q_i(z)$ for each data point x_i, we parameterise it with neural networks, e.g

$$q_\phi(z_i|x_i) = q(z_i = \mathrm{NN}_\phi(x_i)) \approx p_\theta(z_i|x_i).$$

The weights of the neural networks ϕ are shared for the entire dataset, this means the optimisation procedure will no longer depend on the size of the datasets. People often use this formulation due to its simplicity and especially in the case of Gaussian distributions, it is often referred to as the VAE [1]:

$$\mathcal{L}_{\mathrm{ELBO}} = \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right]}_{\text{reconstruction term}} - \underbrace{\mathrm{KL}(q_\phi(z|x)||p(z))}_{\text{regularisation term}}. \tag{2.6}$$

VAE comprises of two main components, encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$. The optimal distribution $q_\phi(z|x)$ from the loss in Equation 2.6 is expected to have a good reconstruction capability while being close to the prior distribution $p(z)$.

2.2.3. Reparameterisation Trick

Training θ, ϕ requires backpropagating through the loss in Equation 2.6 to get the corresponding gradients:

$$\theta^{(i+1)} = \theta^{(i)} + \gamma \nabla_\theta \mathcal{L}_{\mathrm{ELBO}}(\theta^{(i)}, \phi^{(i)})$$
$$\phi^{(i+1)} = \phi^{(i)} + \gamma \nabla_\phi \mathcal{L}_{\mathrm{ELBO}}(\theta^{(i)}, \phi^{(i)}).$$

The gradient $\nabla_\theta \mathcal{L}_{\mathrm{ELBO}}$ is not straightforward to compute as it requires to solve the integration; however, it can be approximated with the Monte-Carlo method:

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\mathrm{ELBO}} &= \nabla_\theta \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] \\
&= \nabla_\theta \int q_\phi(z|x) \log p_\theta(x|z) dz \\
&= \int q_\phi(z|x) \nabla_\theta \log p_\theta(x|z) dz \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\nabla_\theta \log p_\theta(x|z)\right] \\
&\approx \frac{1}{S}\sum_{i=1}^{S} \nabla_\theta \log p_\theta(x_i|z).
\end{aligned}$$

The gradient is thus the average of the individual gradients throughout the whole dataset. Unfortunately, we can not use the same trick to approximate the $\nabla_\phi \mathcal{L}_{\text{ELBO}}$ as

$$\nabla_\phi \int q_\phi(z|x) \log p_\theta(x|z) dz \neq \int q_\phi(z|x) \nabla_\phi \log p_\theta(x|z) dz.$$

Reparameterisation trick can be applied in this situation. In general, it leverages the change of variables principle and apply the deterministic transformation on a variable that is sampled from a trivial distribution $b(\epsilon)$ to our desired variable, i.e $z = T(\epsilon, \phi)$

$$\begin{aligned}
\mathbb{E}_{z\sim q_\phi(z|x)}[h_\phi(z)] &= \int q_\phi(z|x) h_\phi(z) dz \\
&= \int b(\epsilon) h_\phi(T(\epsilon, \phi)) d\epsilon \\
&= \mathbb{E}_{\epsilon\sim b(\epsilon)}[h_\phi(T(\epsilon, \phi))].
\end{aligned}$$

with $h_\phi(z) = \log p_\theta(x, z) - \log q_\phi(z)$. We can now apply the same Monte Carlo approximation as for $\nabla_\theta \mathcal{L}_{\text{ELBO}}$ to get $\nabla_\phi \mathcal{L}_{\text{ELBO}}$:

$$\begin{aligned}
\nabla_\phi \mathcal{L}_{\text{ELBO}} &= \nabla_\phi \mathbb{E}_{\epsilon\sim b(\epsilon)}[h_\phi(T(\epsilon, \phi))] \\
&= \nabla_\phi \int b(\epsilon) h_\phi(T(\epsilon, \phi)) d\epsilon \\
&= \int b(\epsilon) \nabla_\phi h_\phi(T(\epsilon, \phi)) d\epsilon \\
&= \mathbb{E}_{\epsilon\sim b(\epsilon)} \left[\nabla_\phi h_\phi(T(\epsilon, \phi))\right] \\
&\approx \frac{1}{S} \sum_{i=1}^{S} \nabla_\phi h_\phi(T(\epsilon_i, \phi)).
\end{aligned}$$

2.2.4. Constrained Optimisation

ELBO maximisation although resolves the intractable problem, comes with several caveats:

- Over-regularisation: if the regularisation term in Equation 2.6 is too small, the approximate posterior $q_\phi(z|x)$ will be non-informative regardless of the input data. This is difficult to recover during the training process.
- Local optimum: High ELBO does not necessarily mean good latent representation. According to the rate-distortion theorem, a latent representation with high ELBO can either be good at compression or reconstruction. Balancing between them is difficult to achieve without sufficient care.

To address these issues, people often put a scalar factor β on the regularisation term to avoid the posterior collapse and hope that it will converge to a region that is close to the balance point. This type of re-weighting is often called β-VAE:

$$\mathcal{L}^{\beta}_{\text{ELBO}} = \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z)\right]}_{\text{reconstruction term}} - \underbrace{\beta \text{KL}(q_\phi(z|x)||p(z))}_{\text{regularisation term}}.$$

Tuning the newly introduced hyperparameter β is also not that straightforward, as during the training process, we normally can only get the approximation of the gradient (via stochastic gradient descent). Therefore, instead of fixing the β, people often use an annealing schedule: starts from a small value, and then linearly increases it until $\beta = 1$ which will give us the correct ELBO in the end. However, with high dimensional and complex observation, finding the correct temperature for the annealing process is also difficult, Rezende and Viola (2018) proposed a solution to mitigate this issue by treating the ELBO maximisation as a constrained optimisation problem [5], i.e

$$\begin{aligned} \min_{\theta,\phi} \quad & \mathrm{KL}(q_\phi(z|x)||p(z)) \\ \text{s.t.} \quad & \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log p_\theta(x|z)\right] \leq \kappa^2. \end{aligned} \tag{2.7}$$

and the Lagrangian form

$$-\mathcal{L}^{\lambda}_{\mathrm{ELBO}} = \lambda\left(\mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log p_\theta(x|z)\right] - \kappa^2\right) + \mathrm{KL}(q_\phi(z|x)||p(z)).$$

By setting the Lagrangian multiplier $\lambda = 1/\beta$ and putting the negative sign in front we get back the $\mathcal{L}^{\beta}_{\mathrm{ELBO}}$. The nice thing about this formulation is that we now also optimise λ in concurrently with our desired parameters θ, ϕ in the dual optimisation step:

$$\lambda^{(i+1)} = \lambda^{(i)} \cdot \exp\left(\nu \cdot \left(\mathcal{C}^{(i+1)}_{\mathrm{ma}} - \kappa^2\right)\right).$$

with $\mathcal{C}^{(i+1)}_{\mathrm{ma}} = (1-\alpha)\mathcal{C}^{(i+1)} + \alpha\mathcal{C}^{(i)}$ is the moving average version of $\mathcal{C}^{(i+1)} = \mathbb{E}_z\left[-\log p_\theta(x|z)\right]$, our negative log conditional likelihood. However, as λ is automatically tuned, it might result to a different loss which is not the exact ELBO we want to optimise. To this end, Klushyn et al. (2021) proposed a new scheme for the λ update [6] to fulfill this mismatch:

$$\lambda^{(i+1)} = \lambda^{(i)} \cdot \exp\left(-\nu \cdot f_\lambda(\lambda^{(i)}, \mathcal{C}^{(i+1)}_{\mathrm{ma}} - \kappa^2; \tau_1, \tau_2) \cdot \left(\mathcal{C}^{(i+1)}_{\mathrm{ma}} - \kappa^2\right)\right).$$

where f_λ is specified:

$$f_\lambda(\lambda, \delta; \tau_1, \tau_2) = (1 - H(\delta)) \cdot \tanh(\tau_1 \cdot (1/\lambda - 1)) - \tau_2 \cdot H(\delta).$$

where H is the Heaviside function, and τ_1, τ_2 are additional hyperparameters to control the slope of the update function.

2.3. Deep Variational Bayes Filters

2.3.1. State Space Models

State-space model is used to describe a sequence of discrete-time data. It can be seen as an extension of the Latent Variable Models (LVM) previously introduced in Section 2.1. According to the graphical model illustrated in Figure 2.1, the latent states $z_{1:T}$ in SSM are

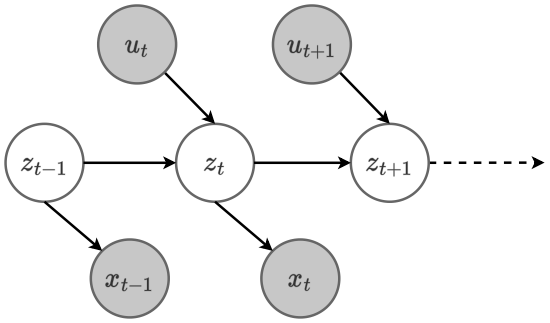


Figure 2.1.: State-space models. The true state z is unobserved and is conditionally dependent on the previous state and the current action. The observed variables are x and the action u.

explained by a sequence of observation variables $x_{1:T}$ and additional controls $u_{1:T}$ with $u_1 = 0$ (zero action in the first time step). The joint distribution is given as follows:

$$\begin{aligned} p(x_{1:T}|u_{1:T}) &= \prod_{t=1}^{T} p(x_t|x_{1:t-1}, u_{1:t}) \\ &= \int \prod_{t=1}^{T} p(x_t, z_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) dz_{1:T} \\ &= \int \prod_{t=1}^{T} p(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) p(z_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) dz_{1:T} \\ &= \int p(x_1|z_1) p(z_1) \prod_{t=2}^{T} p(x_t|z_t) p(z_t|z_{t-1}, u_t) dz_{1:T}. \end{aligned} \tag{2.8}$$

Two important properties have been applied to get the final form in Equation 2.8. One is the factorisation between time steps, and the other is the Markov's property - the current state and observation only depends on the last state and action:

$$\begin{aligned} p(z_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) &= p(z_t|z_{t-1}, u_t) \\ p(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) &= p(x_t|z_t). \end{aligned} \tag{2.9}$$

Similar to the original LVM, learning and doing posterior inference are the main problems we will consider. However, posterior inference in SSM is more complicated because now we need to do it in the whole sequence. Doing inference by conditioning on the whole trajectory we get *smoothing*, $p(z_t|x_{1:T}, u_{1:T})$, and in case we only consider the history, it is called *filtering*, $p(z_t|x_{1:t}, u_{1:t-1})$. Smoothing although can give you a more accurate posterior, can not be done in an online fashion while filtering does. We will discuss more in detail about Bayesian filters in the next section.

2.3.2. Bayesian Filters with Variational Auto Encoders

Similar to the LVM, due to the intractability of the integral, we can not maximise the likelihood directly but instead start with the ELBO:

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{z_{1:T}\sim q_\phi(\cdot)}\left[\log p(x_{1:T}|z_{1:T})\right] - \text{KL}(q_\phi(z_{1:T}|x_{1:T}, u_{1:T})||p(z_{1:T}|u_{1:T})). \tag{2.10}$$

As in VAE, the approximate posterior q_ϕ is also introduced and can be modeled directly by a neural network [2]: $q_\phi(z_t = \text{NN}_\phi(z_{t-1}, x_t, u_t))$. They named it *Deep Variational Bayes Filters (DVBF)*. However, there is another way to model it, inspired by the true SSM's posterior (Equation 2.9), Karl et al. (2019) [29] proposed another variant named *Fusion DVBF* based on the data fusion scheme:

$$q_\phi(z_t|z_{1:t-1}, x_{1:t}, u_{1:t}) = q_\phi(z_t|z_{t-1}, x_t, u_t) \propto q_{\text{inv\_meas}}(z_t|x_t) \times q_{\text{trans}}(z_t|z_{t-1}, u_t). \tag{2.11}$$

As mentioned earlier, the main focus in this work will be the filtering. Therefore, Markov's property has been applied to remove redundant information from history and the future. Intuitively speaking, the factorisation in Eq. 2.11 is similar to the data fusion scheme: the prior state from the transition model q_{trans} is corrected by the inverse measurement model $q_{\text{inv\_meas}}$ from the sensors. Another important point of DVBF is in practice [4], sharing the transition with prior distribution $q_{\text{trans}}(z_t|z_{t-1}, u_t)$ also helps increase the performance and reduce significantly the numbers of parameter:

$$p(z_{1:T}|u_{1:T}) = p(z_1)\prod_{t=2}^{T} p(z_t|z_{t-1}, u_t) = p(z_1)\prod_{t=2}^{T} q_{\text{trans}}(z_t|z_{t-1}, u_t).$$

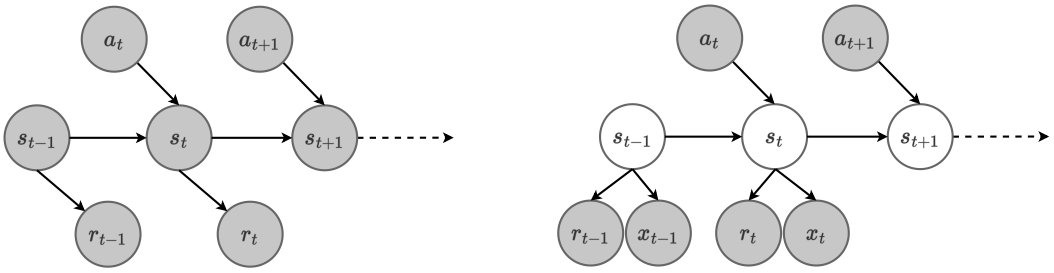
2.4. Reinforcement Learning

2.4.1. Markov Decision Process

Markov Decision Process (MDP) - Figure 2.2a is defined as a tuple of $\mathcal{M} = (S, A, P, R, \gamma)$ where S and A define the state and action space, respectively. P is the transition function which returns the next state given the current state and action: $P(s_{t+1}|s_t, a_t)$. R is the reward function, $R(s_t, a_t) = \mathbb{E}[r_t|s_t, a_t]$. Note that both P and R fulfill Markov's property, hence the name. The goal of MDP is to find the optimal policy $\pi^*(a_t|s_t)$ that maximise the expected return, i.e

$$\pi^*(a_t|s_t) = \arg\max_\pi \mathbb{E}\left[\sum_t \gamma^t r_t\right]. \tag{2.12}$$

where γ is the discounted factor and the expectation is taken over the P and R. Partial Observable Markov Decision Process (POMDP) - Figure 2.2b, is another variant of MDP where the state is not observed directly but only indirectly via the observation. Finding the optimal policy under POMDP is much more difficult due to the intractability introduced by the latent states, we will discuss more about this in the methodology section.



(a) Markov Decision Process.

(b) Partial-Observable Markov Decision Process.

Figure 2.2.: MDP and POMDP models. The current state is influenced by only the last state and the current action. However, in MDP, while the state variables are fully observed, they are only partially observed from the observation x_t in POMDP.

2.4.2. Value-based Learning

The expected return introduced in Equation 2.12 is often referred to as the *Value function*:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_t \gamma^t r_t | s_0 = s\right]. \tag{2.13}$$

Under the case where S and A are finite, P is fully known (as a transition matrix), one can apply Dynamic Programming or Value Iteration (VI) to get the optimal value function, and solve Equation 2.12 to get the optimal policy. The value is iteratively updated via the Bellman backup operator T [30], $V^{(k)} = TV^{(k-1)}$ according to the following equation:

$$V^{(k)}(s) = \max_{a \in A}\left[R(s,a) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^{(k-1)}(s_{t+1})\right]. \tag{2.14}$$

It has been proven that under Lipschitz condition (with a factor γ), the operator T is guaranteed to converge. In other words, $V \to V^*$ as $k \to \infty$ where V^* is the fixed-point value function. State-action value function can also be defined similarly: $Q(s,a) = \mathbb{E}\left[\sum_t \gamma^t r_t | s_0 = s, a_0 = a\right]$. One can also get the value function by picking the best possible action, i.e

$$V(s) = \max_a Q(s,a).$$

In case the transition is not given but can be sampled from e.g, by rolling out from the environment, we can apply Monte-Carlo method to get the value function in Equation 2.14. By law of large numbers, we will eventually get the true expected return. However, Monte-Carlo is an unbiased, high variance estimator, which will require a lot of samples to get the correct value. One way to reduce the variance is introducing some bias, *Temporal Difference*

learning is such a method. It updates the value function immediately after observing the tuple (s_t, a_t, r_t, s_{t+1}) instead of waiting till the end of an episode like MC method:

$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha \left(\underbrace{[r_t + \gamma V^{\pi}(s_{t+1})]}_{\text{TD target}} - V^{\pi}(s_t) \right). \tag{2.15}$$

TD learning introduced the bias by bootstrapping from the previously estimated value function: $V^{\pi}(s_t) = \mathbb{E}[G_t] \approx \mathbb{E}[r_t + \gamma V^{\pi}(s_{t+1})]$ where G_t is the true return.

2.4.3. Policy-based Learning

Another prevailing approach to solving the MDP is *Policy Iteration (PI)*. It starts with an initial policy π, being evaluated via the Bellman backup operator:

$$V^{(k)}(s) = T^{\pi}V(s) = R^{\pi}(s,a) + \gamma \sum_{s_{t+1}} P^{\pi}(s_{t+1}|s_t, a_t) V^{(k-1)}(s_{t+1}).$$

and then improved via this equation:

$$\pi_{k+1}(s) = \arg\max_{a} R(s,a) + \gamma \sum_{s_{t+1}} P^{\pi}(s_{t+1}|s_t, a_t) V^{(k)}(s_{t+1}).$$

These two steps are being repeated alternately until convergence is observed. In practice, policy iteration converges much faster than value iteration although can be sub-optimal and more sample inefficient due to the high variance of the evaluation step. Like the value-based learning, without giving the transition function, we can use Monte-Carlo to get the approximate expected return. In the case where the value function is differentiable w.r.t policy parameters, *Policy Gradient* can be applied. To this end, we first parameterise the policy $\pi_\theta(s) = f_\theta(s)$, and then find the optimal parameter by maximising the value function:

$$\arg\max_{\theta} V(\theta) = \mathbb{E}_{\tau}\left[\sum_t \gamma^t r_t\right] \approx \arg\max_{\theta} \sum_{\tau} P(\tau;\theta) R(\tau).$$

where τ represents the whole trajectory in an episode. Gradient-based optimiser can be applied to solve this problem and therefore, it is essential to compute the gradient w.r.t θ:

$$\begin{aligned}
\nabla_\theta V(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) R(\tau) \nabla_\theta \log P(\tau;\theta) \\
&= \mathbb{E}_{\tau}\left[R(\tau) \nabla_\theta \log P(\tau;\theta)\right].
\end{aligned}$$

Note that in the last step we applied the likelihood ratio formula $\nabla_\theta \log P(\tau;\theta) = {}^{\nabla_\theta P(\tau;\theta)}/_{P(\tau;\theta)}$. The next interesting point is that no model knowledge is required to compute this term:

$$\begin{aligned}\nabla_\theta \log P(\tau_i;\theta) &= \nabla_\theta \log \left[P(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t) \right] \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t).\end{aligned}$$

Taking the gradient by applying MC directly over the whole set of trajectories is very noisy. To this end, the *Reinforce* algorithm leverages the temporal structure to reduce the variance:

$$\begin{aligned}\nabla_\theta V(\theta) &= \mathbb{E}\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \\ &= \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r_{t'} \right] \\ &= \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right].\end{aligned} \tag{2.16}$$

Now for every time step the gradient is updated as follows:

$$\theta = \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) G_t.$$

2.4.4. Model-based Actor-Critic

Actor-critic is a hybrid approach between value-based and policy-based, i.e it aims to jointly learn both the value function (critic) and the policy (actor). This combination has been empirically proven to be more stable than policy gradient methods [31, 32, 33]. To this end, we start by introducing a baseline $b(s)$ in Equation 2.16:

$$\nabla_\theta V(\theta) = \mathbb{E}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right].$$

The term inside can be seen as the *advantage estimate* $A_t = R_t - b(s_t)$, and if we extend it further by replacing $R_t = \sum_{t'=t}^{T-1} r_{t'} = r_t + \gamma V(s_{t+1})$ and $b(s_t) = V(s_t)$ as the baseline:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

The gradient is thus $\nabla_\theta \log \pi(a_t|s_t;\theta) A_t$, which requires us to compute the value function. As the optimal value function $V(\cdot)$ is not given, one can replace it with a parametric function $V_\phi(\cdot)$ and train it together with the policy.

On the other hand, all of the reinforcement learning algorithms introduced so far are categorised as model-free reinforcement learning. They have been widely applied in various domains and reported to have a great asymptotically convergence rate. However, the biggest

downside is that it requires a lot of samples, which can be up to millions [34, 35, 36]. In the domain where sampling is expensive, e.g robotics, model-free approaches are not preferable. *Sim2real* is a common approach to deal with this problem. In this case, a policy is learned with RL in the simulation and then will be transferred to the real world domain. This helps reduce the number of interaction on real robots during training. However, the gap between two worlds remains large, *sim2real* transferring is also challenging. Model-based RL [37, 38, 39, 40] is another approach to reduce the sample complexity, the idea is simple, the model will be learned together with the policy in an iterative fashion. Together with actor-critic [38, 39, 40] we have the following algorithm, which often referred to as Model-based Actor Critic (MBAC):

Algorithm 1 Model-based Actor-Critic

Initialise policy π_θ , value function V_ϕ, and dynamic f_ψ
Initialise a dataset $\mathcal{D} = \emptyset$
repeat
  Execute π_θ and collect $\{s_t, a_t, s_{t+1}\}_{t=0}^{T}$
  Add $\mathcal{D} = \mathcal{D} \cup \{s_t, a_t, s_{t+1}\}_{t=0}^{T}$
  Train model f_ψ using $\mathcal{D}$ {Model learning}
  repeat
    Collects N trajectories $\{\tau_i\}_{i=1}^{N}$ from dynamics f_ψ using π_θ
    Update policy π_θ on fictitious data $\{\tau_i\}_{i=1}^{N}$ {Policy learning}
    Update value function V_ϕ {Critic learning}
  until convergence
until task successfully learned

Model learning is an important part of the algorithm. In case the state is fully observed, one can just do supervised learning by minimising the mean square error between data and the prediction, i.e $\|s_{t+1} - f_\phi(s_t, a_t)\|_2$. However, in this thesis, we will put the focus on the situation where states are only observed partially, e.g via images. To this end, the DSSM inference method introduced in the Section 2.3 will be used and the details will be discussed in the methodology section.

3. Methodologies

3.1. Problem Statement

As discussed in Section 2.4, model-free reinforcement learning is one way to solve the MDP, although it is very sample inefficient. On the contrary, learning a model helps to restrict the search space and therefore can reduce the sample complexity. Under the MDP setting, directly minimising the mean square error is an obvious choice to obtain the desired approximate transition model. However, in the robotics domain, usually, only part of the information can be observed. Therefore, our main focus is doing control under partially observable MDP setup. Learning models under POMDP setting is similar to do inference on state space models, and thus the Deep Variational Bayes Filters (DVBF) presented in Section 2.3 can be applied. First, to strengthen the consistency, we reintroduce some of the terminologies:

Table 3.1.: Terminologies

Variables	Name	Descriptions
z_t	latent state	latent state at time t
x_t	observation	observation at time t, e.g images
u_t	control	action taken at time t
r_t	reward	reward at time t after executing an action
$\pi(u_t \vert z_t)$	policy	policy returns an action given a state z_t
$p(z_{t+1} \vert z_t, u_t)$	transition distribution	latent transition distribution

Let $q_{\text{trans}}(z_{t+1}|z_t, u_t) = \mathcal{N}(\mu, \Sigma)$ be the parameterized transition distribution where $\mu, \Sigma = \text{NN}(z_t, u_t)$ is represented by a deep neural network, the dynamic model can then be learned by optimising the following constrained optimisation problem:

$$\begin{aligned} \min_{\theta,\phi} \quad & KL(q_\phi(z_{1:T}|x_{1:T}, u_{1:T})||p(z_{1:T}|u_{1:T})) \\ \text{s.t.} \quad & \mathbb{E}_{z_{1:T}\sim q_\phi(\cdot)}\left[\log p(x_{1:T}|z_{1:T})\right] \leq \kappa^2. \end{aligned} \tag{3.1}$$

The Lagrangian form of Equation 3.1 can be seen as the typical scaled ELBO, please refer to Section 2.2.4 for more detail. Then, to learn the policy, the fictitious trajectories sampled from the learned transition can be used, there is no need to use the limited samples from the real world. Indeed, the two phases can be trained in an iterative fashion like the model-based actor-critic introduced in Section 2.4.4.

Given the aforementioned framework, an arbitrary policy can be efficiently learned by maximising a predefined reward $r(z_t, u_t)$ e.g, goal reaching. Typical reinforcement learning

algorithms can only guarantee to give an optimal policy for a specific task or an environment defined by that predefined reward. It means even in the case where there is only a single slight difference in the perception or the dynamics (mass changed), rerunning the whole algorithm might be the only option. One can argue that the true dynamic is able to deal with such a change. However, the learned transition function only captures the local behaviours (based on tasks) of the true dynamics [41, 42, 43]; therefore, it will fail if such a change occurred. Now, looking closer to the formula, it is apparent that the policy, transition, and value function are all defined in the latent space. To this end, we seek an answer to the question: can the latent space be reused and would it be possible to quickly adapt it to different kinds of tasks or even robots? The following sections will show our approach to address this problem.

3.2. Curriculum Learning for DSSM

Given a pre-trained state space models on a base task, our main goal in this research is to transfer it to a more difficult one, in a curricula fashion. This section will describe how we do it by first explaining the curriculum learning as a Bayesian update, then propose a new DSSM architecture that is transferable based on that learning mechanism.

3.2.1. Curriculum Learning as the Bayesian Update

Consider the scenario where given a set of parameters ψ learned on task A, our objective is transferring it to a target task B. To this end, let D_A be the base task dataset and D_B be the target task dataset. Our optimal ψ^* is expected to work for both tasks A and B. Towards this direction, we frame this problem as the Bayesian update [44]:

$$\begin{aligned}\psi^* &= \arg\max_{\psi} p(\psi|D_A, D_B) \propto p(D_B|\psi)p(\psi|D_A) \\ &= \arg\max_{\psi} \log p(\psi|D_A, D_B) \propto \log p(D_B|\psi) + \log p(\psi|D_A).\end{aligned} \tag{3.2}$$

The main objective is maximising the joint posterior $p(\psi|D_A, D_B)$, which consists of the original likelihood $p(D_B|\psi)$ and the prior $p(\psi|D_A)$ on the previous task. The posterior $p(\psi|D_A)$ is intractable due to the integral of the normalization term $p(D_A)$ and similarly this also holds for the posterior $p(\psi|D_A, D_B)$ w.r.t the new task B. We can however approximate it with the Taylor approximation [25]:

$$-\log p(\psi|D_A) \approx \frac{1}{2}(\psi - \psi_A^*)^T H(\psi_A^*)(\psi - \psi_A^*)$$

where ψ_A^* is the optimal parameter for task A, i.e $\psi_A^* = \arg\min_\psi -\log p(\psi|D_A)$, hence $\nabla_{\psi=\psi_A^*}[-\log p(\psi|D_A)] = 0$, and $H(\psi_A^*)$ is the corresponding Hessian matrix. The expectation of the Hessian matrix can be seen as the Fisher information matrix (Section A.2):

$$\mathbb{E}_{p(\psi|D_A)}\left[H_{-\log p(\psi|D_A)}(\psi^*)\right] = F(\psi^*)$$

In most cases, the expectation is not available in closed form. However, it can be computed from data and referred to as the empirical Fisher information matrix:

$$\frac{1}{N_A}\sum_{x_i \in D_A} H_{-\log p(\psi|x_i)}(\psi^*) = F(\psi^*)$$
$$\sum_{x_i \in D_A} H_{-\log p(\psi|x_i)}(\psi^*) = N_A F(\psi^*)$$

Hence,

$$-\log p(\psi|D_A) \approx (\psi - \psi^*)^T \gamma F(\psi^*)(\psi - \psi^*)$$

where γ is introduced to replace N_A for better quality control of the approximation. However, computing the Fisher information matrix can be difficult, others ([10, 25, 45]) enforce the diagonal version instead. More interestingly, Fisher information matrix also has a nice connection to KL-divergence (Section A.3), that is:

$$\gamma \mathrm{KL}(p(D_A|\psi)\|p(D_A|\psi^*)) = \gamma(\psi - \psi^*)^T F(\psi^*)(\psi - \psi^*) + \mathcal{O}(\Delta\psi^3) \tag{3.3}$$

Thus, we arrive the final loss function:

$$\mathcal{L}(\psi) = -\log p(D_B|\psi) + \gamma \mathrm{KL}(p(D_A|\psi)\|p(D_A|\psi^*))$$

In this form, it can be intuitively seen that apart from maximising the original likelihood in the target task B, we also put a constraint to enforce the parameters to stay not too far from the optimal point trained on the base task A.

3.2.2. Transferable State Space Inference Models

In this research, we decided to employ the DVBF model introduced in Section 2.3 to do inference on SSM. Its architecture is illustrated in Figure 3.1

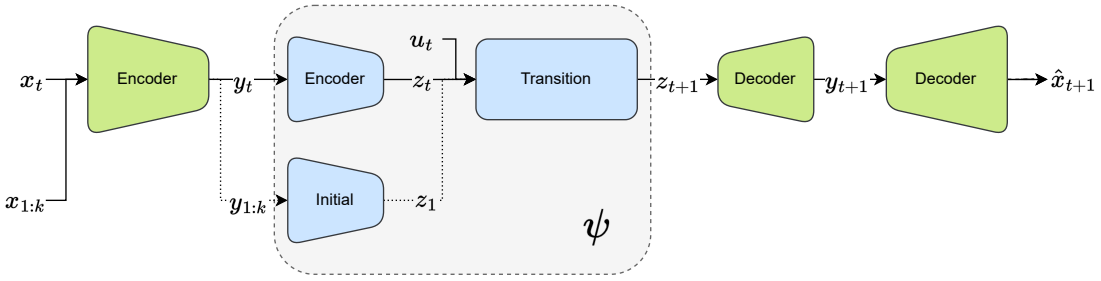


Figure 3.1.: Transferable state-space inference models. The parameters ψ described in the grey block can be transferred to the other environments. The overall structure is the multi-level encoding decoding scheme with two latent layers y and z.

The described model is extended to a multi-level encoding scheme: in addition to the latent variable z, a new latent variable y has been introduced. This design choice has several advantages compared to the original one:

- This multi-level architecture has been empirically proven to be powerful for high dimensional observation data, e.g images [46, 47, 48].
- The newly introduced auxiliary observation y helps prevent the bottleneck problem w.r.t the information loss. In fact, y can be seen as the compressed version of x where x can be in arbitrary space. This means the shared latent structure (blue blocks) can be fixed and reused on any types of input.

Formally speaking, let y be an auxiliary observation variable, the generative process is defined as follow:

$$z \sim p(z) \longrightarrow y \sim p_\theta(y|z) \longrightarrow x \sim p_\theta(x|y)$$

and similar to the LVM inference, under variational inference framework, only ELBO is available during the optimisation:

$$\begin{aligned}
\log p(x) &= \log \int\int p(x|y)p(y|z)p(z)\frac{q(y|x)}{q(y|x)}\frac{q(z|y)}{q(z|y)}dzdy \\
&\geq \mathbb{E}_{y\sim q(y|x)}\mathbb{E}_{z\sim q(z|y)}\left[\log p(x|y)\right] - \mathrm{KL}(q(z|y)||p(z)) - \mathrm{KL}(q(y|x)||p(y|z)) \\
&= \mathcal{L}_{\mathrm{ELBO}}.
\end{aligned}$$

As one can see, it is quite straightforward to extend the ELBO to the newly introduced latent variable y: similar to z, we also introduce the new approximate posterior $q(y|x)$ and apply the same derivation to arrive the final form above. Now, follow the extension version of LVM inference for the SSM inference as in Section 2.3, and putting everything together, we have the following constrained optimisation problem:

$$\begin{aligned}
\min_{\theta,\phi} \quad & \mathrm{KL}(q_\phi(z_{1:T}|y_{1:T}, u_{1:T})||p(z_{1:T}|u_{1:T})) + \mathrm{KL}(q_\phi(y_{1:T}|x_{1:T})||p(y_{1:T}|z_{1:T})) \\
\text{s.t.} \quad & \mathbb{E}_{y_{1:T}\sim q_\phi(\cdot)}\mathbb{E}_{z_{1:T}\sim q_\phi(\cdot)}\left[-\log p(x_{1:T}|y_{1:T})\right] \leq \xi_{\mathrm{rec}} \\
& \mathrm{KL}(p(\tilde{x}|\psi)\|p(\tilde{x}|\psi^*)) \leq \xi_{\mathrm{bu}}.
\end{aligned} \tag{3.4}$$

where $\tilde{x}$ is the observed data from the previous task. Apart from the reconstruction constraint, the curriculum constraint has also been integrated to enforce the shared components stay close to the optimal one learned from the based task. The shared parameters ψ consists of three main components: the z-encoder, transition function and the initial network as highlighted in blue in Figure 3.1. In practice, we will employ the approximation form based on Fisher information matrix introduced in Equation 3.3 as it can be computed after training the base task. This means no additional computational cost (w.r.t base task dataset) will be added during the training phase of the target task.

Algorithm 2 Sequence models learning with Bayesian Update

Initialise dual variables $\lambda_{\text{rec}}^{\text{init}}$, $\lambda_{\text{bu}}^{\text{init}}$
Initialise $init\_phase = True$
repeat
  Compute moving average of $rec\_constraint$
  Compute moving average of $cur\_constraint$
  if $rec\_constraint \leq \xi_{\text{rec}}$ **then**
    $init\_phase = False$
  end if
  if $init\_phase = True$ **then**
    Optimise the loss $rec\_error + \lambda_{\text{bu}}^{\text{init}} \cdot cur\_error$
  else
    Optimise the loss $(\lambda_{\text{rec}} \cdot rec\_error + KL) + \lambda_{\text{bu}} \cdot cur\_error$
    Update λ_{rec} based on $rec\_constraint$
    Update λ_{bu} based on $cur\_constraint$
  end if
until convergence

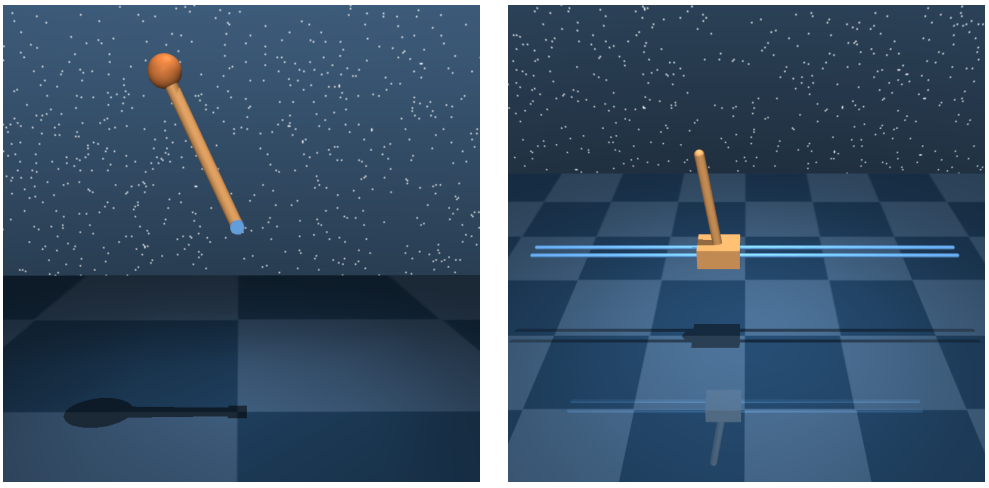
The training algorithm is described in Algorithm 2. We follow the Reconstruction-error-based weighting of the objective function (REWO) style introduced by Klushyn, et al. (2019, 2021) [49, 6], i.e the training scheme is divided into two phases: initial and main phase. In the initial phase, the regularised reconstruction constraint is being optimised with fixed coefficient $\lambda_{\text{bu}}^{\text{init}}$. Then, after the constraint is satisfied it starts to optimise the main objective function and together with the Lagrange dual variables for both the reconstruction and curriculum constraints. Note that there are some slight differences in the actual implementation as opposed to the above pseudo code. Instead of directly running the optimiser with a very high value of λ_{bu} and λ_{rec}, we do it in the inverse fashion, which means, in the beginning, we only allow a very small gradient from the *rec_error* flow through the shared parameters. We also scaled back the non-shared parameters in this initial phase, then once it was done, we used the same scale for all the parameters. This can be intuitively understood that in the beginning we focus on learning a good mapping from the observation to latent space while trying to make as little change as possible to all the latent dynamic parameters. Then, once a good representation is found, we can start to optimise the dynamics (putting the scale down on the encoder, decoder makes the training process clearer that we are focusing on the dynamics). An example is the pendulum $\rightarrow$ cart-pole curricula: initially, only the representation of the pole is extracted from the knowledge of the previously trained pendulum. Note that the cart position will be completely ignored in the latent space. Then, after reaching certain conditions, we start to optimise the dynamics, now the dynamic is allowed to have significant changes to better capture the true cart-pole dynamics.

4. Experimental Setup

This chapter provides in detail the experimental setup that will be used to demonstrate the benefits of our proposed methods. The first subsection will discuss the testing environments and data collection process. Meanwhile, in the second half, we will present a list of baseline models and describe the metrics that will be used to compare them.

4.1. Environments

To evaluate our proposed models, we decided to choose DeepMind Control Suite (DMC) [50] as our testing environments. DMC is a standard benchmark suite for continuous control tasks. It is developed based on Mujoco simulation [51], which is a popular, fast and efficient simulation for multi-body dynamical systems. More specifically, the two environments employed in this thesis are pendulum and cart pole (Figure 4.1).



(a) Pendulum (b) Cart pole

Figure 4.1.: DeepMind Control Suite environments [50].

4.1.1. Environmental Details

Pendulum The pendulum has only one controllable joint, which is located at the center of the scene and indicated by the blue circle in Figure 4.1a. The main task in this environment is to swing up the pendulum. The dynamics of the system is mainly determined by the length

of the pole and the mass of the system which is concentrated at the end-effector (indicated by the orange sphere in Figure 4.1a). It is a highly non-linear dynamical system and has been extensively studied in classical control theory.

Cart pole The cart pole system consists of two joints, one prismatic and one revolute joint. However, only the prismatic joint is controllable, which makes the system underactuated. The cart is free to move along the horizontal track (Figure 4.1b) while the pole, which is attached to the center of the moving cart, behaves like the single pendulum. The objective of this system is to balance the pole in the upright position on the moving cart.

4.1.2. Data Collection

For all of the following experiments, our pre-trained DSSM will be trained on the fully observable pendulum environment described in the previous section. Initially, we uniformly sample both the position and velocity of the system, then we applied random actions to get the whole trajectory, 3000 such trajectories will be collected in total. To further enrich the dataset, we use the sliding window method, which generates a set of trajectories by incrementally moving a fixed-size window over each collected trajectory. However, during the evaluation phase, for the image version only 1000 trajectories are used. Note that for this particular setup, the velocity information is implicitly removed, and make it a partially observable environment. We expect that the pre-trained DSSMs will help us quickly infer the hidden velocity information.

For the cart-pole, the data is generated by initially letting the pole be always in the upright position and moving the cart with random actions. Similar to the previous environment, 1000 trajectories will be collected and only positions are observed, both the joint velocities are removed. This two dimensional system is useful for demonstrating the evolution of a robotic system. We expect that the pre-trained latent space learned by a single pendulum can be transferred and adapted accordingly to the cart-pole environment.

4.2. Evaluation Methodologies and Metrics

4.2.1. Evaluation Methodologies

For the sequence model training, the following methods will be compared:

- DVBF-REWO: the DVBF model with REWO training scheme. The whole training procedure will start from scratch.
- DVBF-REWO-PT: DVBF-REWO with pre-trained DSSM on the raw state pendulum. No bayesian update will be applied. However, in the initial phase, we apply a smaller weight to the learning rate of the pre-trained parameters to enforce the trainer to emphasise more on the non-pre-trained parameters.

- DVBF-REWO-BU: the DVBF-REWO will be used similar to the previous two models, however, the bayesian update formula proposed in Section 3.2.2 will be applied. The curriculum threshold ξ_{bu} is chosen as follow, for cart-pole $\xi_{\text{bu}} = 0.001$ while $\xi_{\text{bu}} = 0.3$ for pixel pendulum. The initial Lagrange are also different, i.e $\lambda_{\text{bu}}^{\text{init}} = 5$ for cart-pole and $\lambda_{\text{bu}}^{\text{init}} = 4$ for pixel-pendulum. Similar to the previous model, in the initial phase, we only want the bayesian update being applied on the pre-trained parameters. To this end, we also rescale the learning rate for non-pre-trained components by multiplying it by an amount of $\lambda_{\text{bu}}^{\text{init}}$.

4.2.2. Metrics

To compare the aforementioned methodologies, we use the following metrics:

- MSE-30: Cumulative mean square error from the first to the 30th steps, i.e mse-30 $= \sum_{t=1}^{T} \frac{1}{|D_x|} \sum_{x \in D_x} (x_t - \hat{x}_t)^2$, where x_t and $\hat{x}_t$ are the ground-truth and predicted observation, respectively and D_x is the entire observation trajectories collected in the dataset.
- NLL: Negative log-likelihood. This can be interpreted as the reconstruction loss, where the lower value indicates that the reconstructed observations close to the actual one.
- KL for z: KL divergence for latent variable z. This represents the divergence between the approximated posterior and the prior distribution. In practice, for DSSM, the lower value usually indicates the better quality of the rolled out prediction trajectories.

5. Results

The final results discussed in this chapter will answer the following questions:

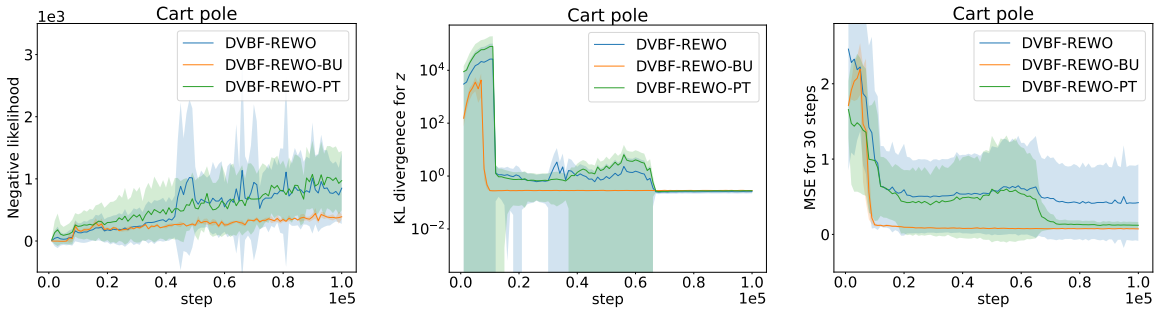
- Can a pre-trained DSSM be re-used for other environments, with different observational inputs, and/or different dynamical systems, but having one or more components that have the same movement?
- Training sequence models from scratch by maximising the ELBO is very time and data consuming. With a pre-trained model, is it possible to accelerate the sequence model training speed?
- In model-based actor-critic discussed in Section 2.4.4, the sequence model training is done in an online fashion. With a pre-trained model, does it help to reduce the sample complexity, i.e less numbers of interaction to the environment?

5.1. Sequence Model Learning

In this section, we validate our hypothesis on two different environments. The first one is cart-pole, which has a different kinematic structure, hence different dynamical system, but the movement of the pole is supposed to be similar to the single pendulum. In the second environment, the pixel pendulum is used, i.e the dynamic is exactly the same, but the observational input is a sequence of images without any velocity information.

5.1.1. Cart-pole - Different Dynamical Systems

In this experiment, for the purpose of showing the transferring capability of the proposed models on another dynamical system, we set the latent dimension to a low value, i.e $d_z = 3$. First, Figure 5.1 illustrates the comparison between pre-trained and non-pre-trained models. For the prediction capability, Figure 5.1c clearly shows that the pre-trained model achieve higher performance than its opponent, and the one with Bayesian update (DVBF-REWO-BU) converges much faster than the only pre-trained one (DVBF-REWO-PT). Meanwhile, there is not a big difference between DVBF-REWO-PT and DVBF-REWO (the non-pre-trained model) for the ELBO plots since both of them can not extract the underlying dynamics given only three dimension. Note that three dimension is not the optimal number of dimension for cart-pole system since its intrinsic dimension (ID) is four (two joint positions and two joint velocities). Cart velocity will be ignore in the learned latent space, that is also the main reason where the negative likelihood shown in Figure 5.1a is also not the optimal one, even with DVBF-REWO-BU.



(a) Negative likelihood (b) KL for z (c) MSE-30 for z

Figure 5.1.: Sequence model training results on cart pole environment. DVBF model with REWO training scheme described in Section 2.2.4 is used to compare between non-pre-trained model and pre-trained models with and without bayesian update. The pre-trained models are trained on the raw state pendulum environment. The latent dimensions are set to $d_y = 8$, $d_z = 3$.

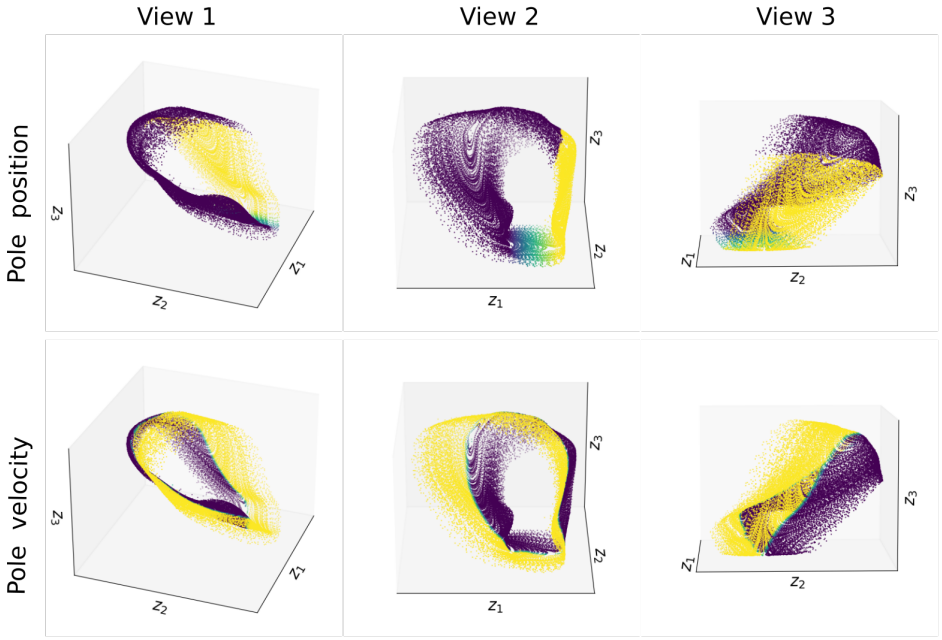


Figure 5.2.: Latent manifold for the pendulum. The color indicated by each rows is marked by the pole position and pole velocity in that order. Each column shows different views in the 3D space.

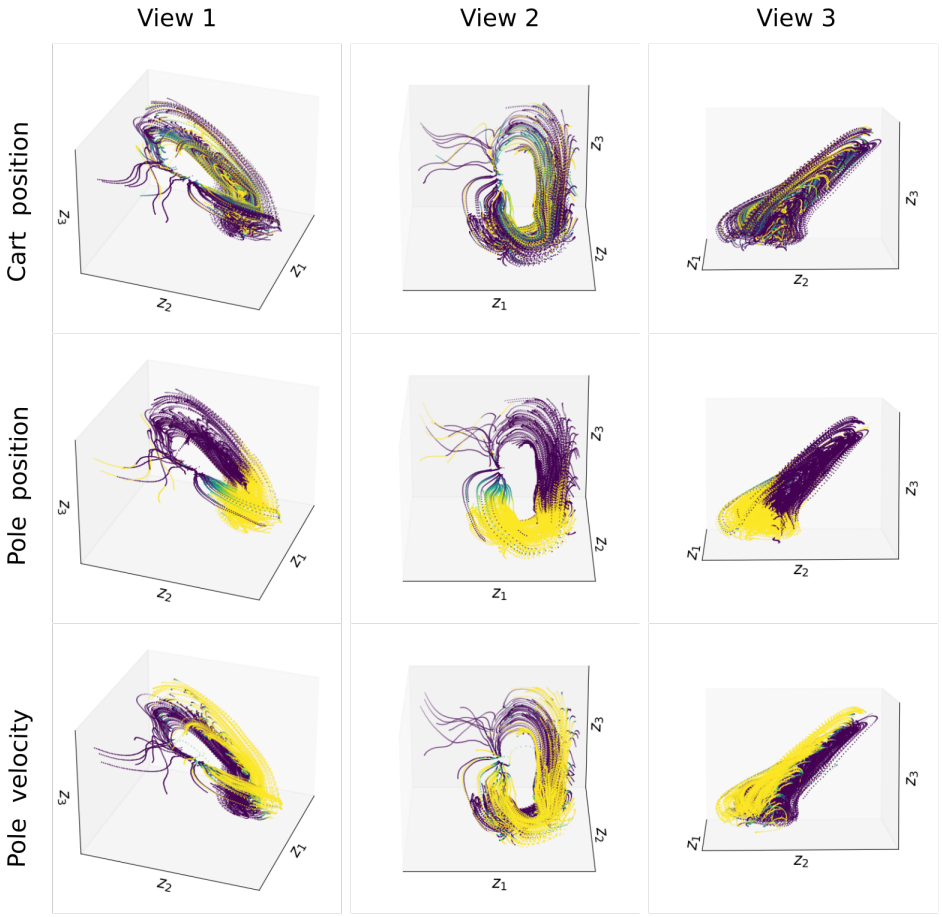
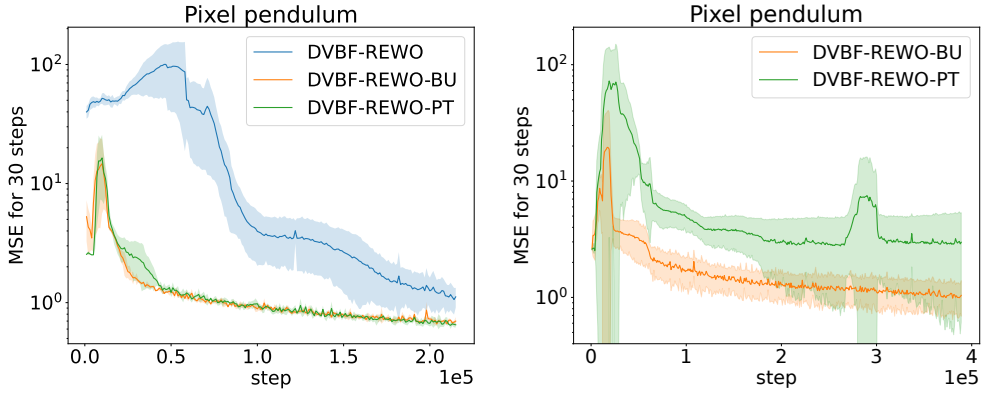


Figure 5.3.: Latent manifold for the cart pole. The color indicated by each rows is marked by the cart position, pole position and pole velocity in that order. Each column shows different views in the 3D space.

To better demonstrate the usefulness of having a pre-trained model, we also compare the latent manifold learned by the pendulum (Figure 5.2) and the cart-pole (Figure 5.3) using the DVBF-REWO-BU model. The plots show that the latent encoding scheme is preserved, i.e, the pendulum position and velocity are aligned by an empty barrel-like shape. Meanwhile, to encode the cart position, the latent manifold changed by stacking multiple barrel-like shapes in layers.

5.1.2. Pixel Pendulum - Different Observational Inputs



(a) MSE for 30 steps with $z = 6$

(b) MSE for 30 steps with $z = 4$

Figure 5.4.: Sequence model training results on pixel pendulum environment w.r.t MSE-30 metric. DVBF model with REWO training scheme described in Section 2.2.4 is used to compare between non-pre-trained model and pre-trained models with and without bayesian update. The pre-trained models are trained on the raw state pendulum environment. The latent dimensions are set to $d_y = 32$, $d_z = \{4, 6\}$.

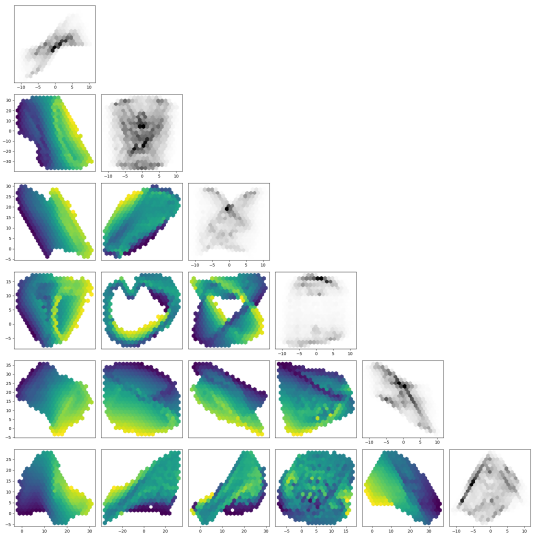
This experiment evaluates the pre-trained models on the pendulum system with images as the only observation inputs, i.e an RGB $32 \times 32 \times 3$ image. Encoding images usually requires more capacity compared to the raw state; therefore, we set the latent dimension to a higher value, i.e $d_z = \{4, 6\}$. In the left Figure 5.4, the pre-trained model clearly outperforms the non-pre-trained one. We highlight the results w.r.t the long horizon prediction error (MSE-30). For further insight, we refer the reader to the ELBO plots in the Appendix B.2. The non-pre-trained model DVBF-REWO took more than 50k steps to satisfy the reconstruction constraint (indicated by the drop in the KL divergence plot) while both of the pre-trained models only need less than 10k steps to finish the initial phase and then able to optimise the full ELBO. On the other hand, with $d_z = 6$, the difference between the one with Bayesian update (DVBF-REWO-BU) and the only pre-trained one (DVBF-REWO-PT) is negligible although DVBF-REWO-BU indeed performs slightly better than DVBF-REWO-PT in the MSE plot (from 20k steps to 50k steps). With only four dimensions, training becomes much more challenging, even for the pre-trained models (the non-pre-trained one was omitted since it did

not converge at all). However, the training curve of DVBF-REWO-BU appears to be smoother and perform better than DVBF-REWO-PT.

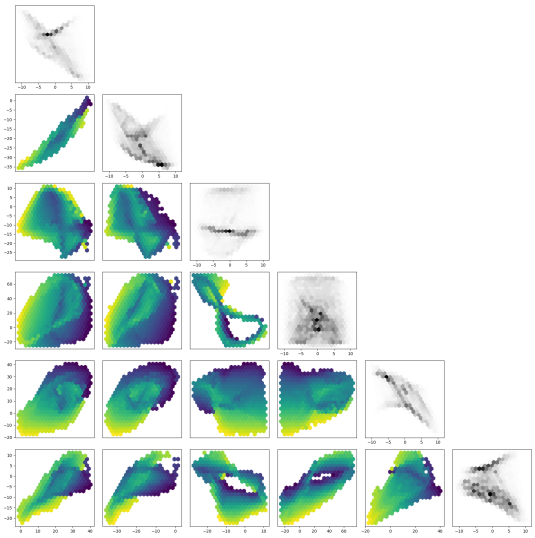
The reconstruction capability of the DVBF-REWO models are illustrated in Figure 5.5. Since there are almost no big difference between the pre-trained and non-pre-trained models in this sense, we choose DVBF-REWO-BU model to generate the plots. It clearly shows that at the converged state, the REWO scheme for DVBF model demonstrates a good capability of reconstructing images from the latent state. Note that in the prediction trajectory (the third row of Figrure 5.5), the first step is inferred by the initial distribution given the first three observations, i.e, the first prediction step corresponds to the third step of the ground-truth one.

Figure 5.5.: Rolled out trajectories of the pixel pendulum system generated by DVBF-REWO-BU model. The filtered and predicted trajectories are shown in the second and third rows, respectively. Ground-truth trajectory is plotted in the first row.

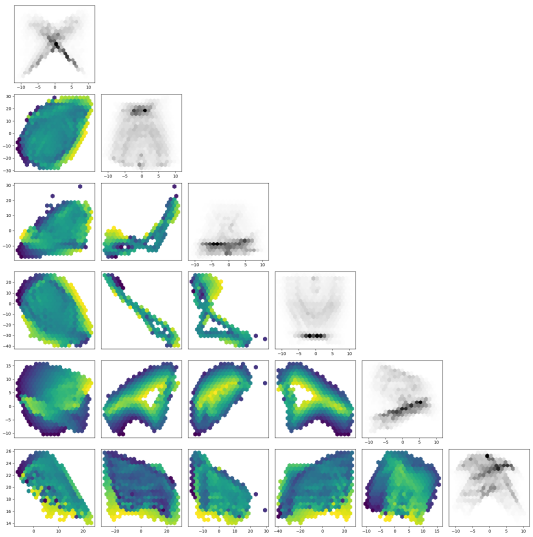
The inferred latent manifold from the validation dataset for all the models are also demonstrated in Figure 5.6. Due to the number of dimension exceeding three, we can only project the manifold onto 2D space by considering all possible pairs of dimensions, resulting in a 6×6 lower triangular matrix. The color is marked by the hidden velocity information. At the converged state, our results show that the constrained optimisation formula and the REWO updating scheme successfully infer the pendulum velocity, as indicated by looking at the color gradient.



(a) DVBF-REWO-BU



(b) DVBF-REWO-PT



(c) DVBF-REWO

Figure 5.6.: Latent manifold for the pixel pendulum. The images are projected onto 2D space for each pair of latent dimensions. The diagonal shows the correlation between the metric being used (pendulum velocity) and its corresponding latent dimension.

5.2. Model-based Reinforcement Learning

In this section, we evaluate our proposed models on the policy learning task. We follow the MBAC algorithm described in Section 2.4.4, i.e the sequence model and policy are learned together in an alternative fashion. Initially, a set of 50 trajectories is collected (each has 200 steps) by running an exploration policy with uniformly random actions. This data is used to train the sequence model during the warm-up phase in the first 40k steps. After that, the main phase is activated, the policy is optimised and being used to collect data at every 200 update steps. To show the efficiency of MBAC, only one trajectory is collected in each cycle and is stored in the replay buffer, which is the main source of data we used to update the sequence model.

Similar to the sequence model training experiments, we compare three methods: non-pre-trained model MBAC-DVBF-REWO, pre-trained model with bayesian update MBAC-DVBF-REWO-BU and without bayesian update MBAC-DVBF-REWO-PT. The pre-trained models were trained on the raw state pendulum environment while also learning the optimal policy. We also present the comparison between the pre-trained models with optimal and random policy in Figure B.2. To ensure a fair comparison, we decided to set the latent dimensions to a higher value, i.e $d_y = 32$ and $d_z = 8$. This is the best performing setting for the non-pre-trained model MBAC-DVBF-REWO.

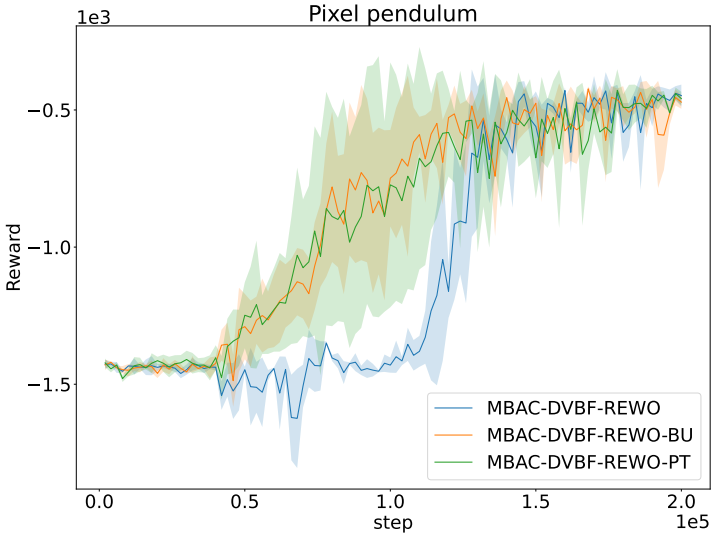
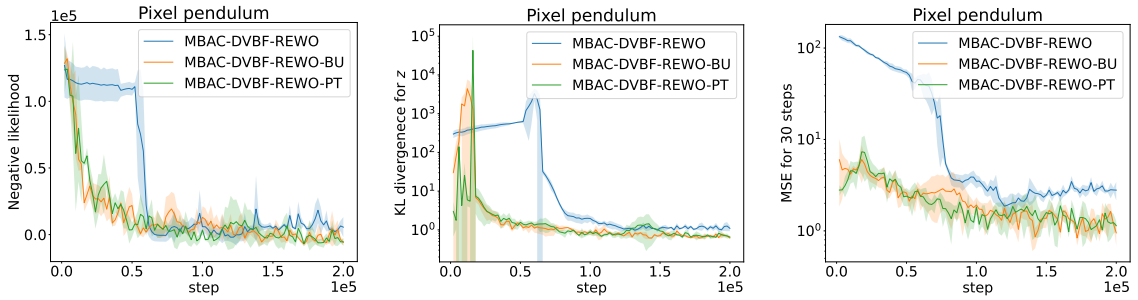


Figure 5.7.: Reward obtained by MBAC algorithm with different types of training the sequence models similar to the previous experiment (Section 5.1.1). The latent dimensions are set to $d_y = 32$, $d_z = 8$. The warm-up phase ends at the 40k-th step and after that the data is collected at every 200 steps.

The comparison between different model-based reinforcement learning methodologies in terms of reward is illustrated in Figure 5.7. As shown, the MBAC-DVBF-REWO takes longer training time and data to learn the policy. It only started to learn the policy after



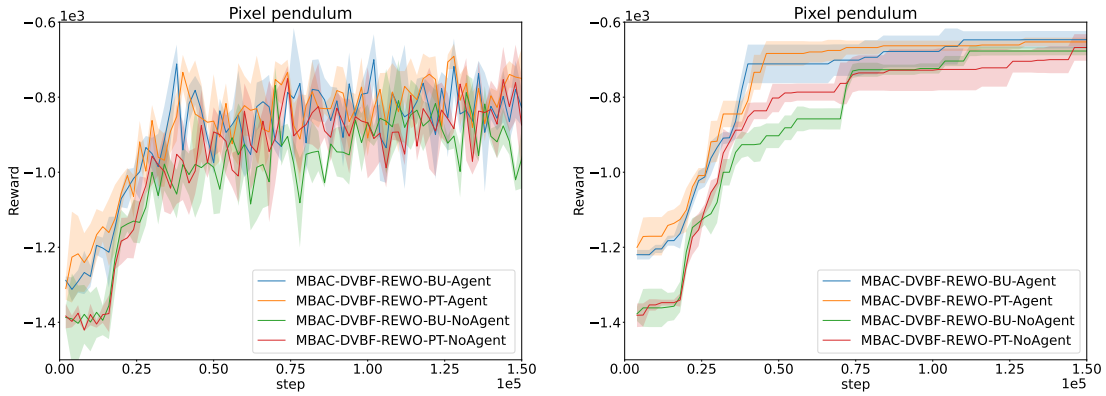
(a) Negative likelihood (b) KL for z (c) MSE for 30 steps

Figure 5.8.: Comparison between different sequence model learning methodologies (combined with MBAC) on pixel pendulum environment. The latent dimensions are set to $d_y = 32$, $d_z = 8$. The warm-up phase ends at the 40k-th step and after that the data is collected at every 200 steps.

100k steps. Meanwhile, the pre-trained ones immediately learn the policy right after the warm-up phase, and perform reasonably well at around 100k steps. To gain better insight, we also present the sequence model performance in Figure 5.8. While the pre-trained models almost immediately converge to a good sequence model, only after finishing the warm-up phase, MBAC-DVBF-REWO started to learn the sequence model and converge at 100k steps. However, the 30-steps prediction error of the non pre-trained model is much higher than the pre-trained one. This verifies that even without having a perfect sequence model, the policy is still able to converges to the optimal policy.

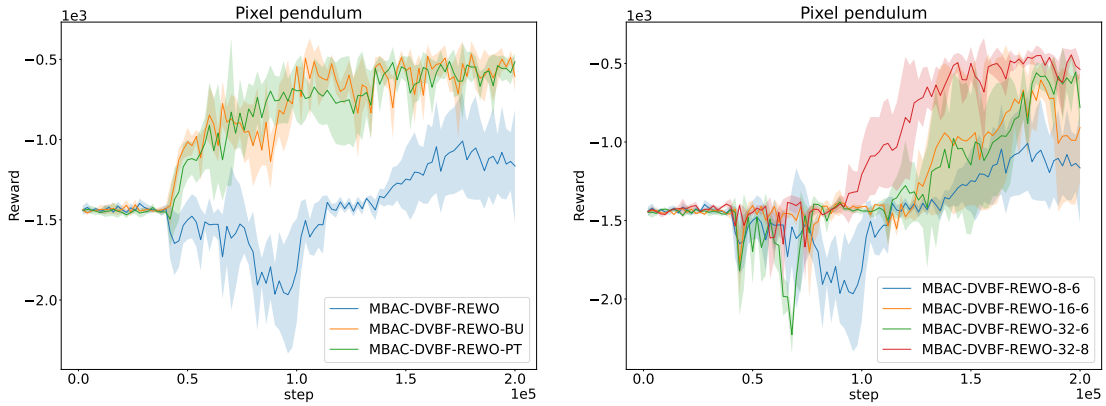
To further evaluate the capability of having pre-trained components, we try out the pre-trained models on an environment where the dynamics and the observation inputs are both different. To be more concrete, we slightly modify the pixel pendulum environment used before by doubling the mass and changing the background color. This change is inspired by the sim-to-real setting, where the real dynamic is not exactly the same as the simulated one, and obviously, the image input will also be different. To see the efficiency of our proposed models, we also reduce the numbers of interaction to the real-world in this experiment by increasing the cycle of the data collection from 200 steps to 500 steps. We compare the proposed models with and without pre-trained policy and critic models. The results in Figure 5.9 clearly indicate that with a pre-trained agent although on different dynamics and background, the training time is almost reduced by half, i.e the ones with pre-trained agent started to achieve a good reward at around 32k steps while it takes 75k steps for the opponents to get the same level. It is easier to witness the difference in the right figure as the average rewards reported in the left figure fluctuate significantly. The reason is, with doubled mass, the pendulum becomes more unstable in the upright position, making it more challenging to balance between exploration and exploitation.

For completeness, we also run the algorithms with lower latent dimension $d_y = 8$ and $d_z = 6$. The results are illustrated in Figure 5.10a. With only six dimensions, the non-pre-trained model is not able to converge within 200k steps. The performance of this model under



(a) Reward (b) Best reward

Figure 5.9.: Comparison between pre-trained sequence models with (*-Agent) and without (*-NoAgent) pretrained-policies (loaded from the ordinary pixel pendulum environment) for the pixel pendulum environment with double mass and different visual background. The warm-up phase ends at the 15k-th step and after that the data is collected at every 500 steps.



(a) Comparison between different MBAC with $d_y = 8$ and $d_z = 6$ (b) Ablation study for different latent dimensions

Figure 5.10.: Ablation study with different latent dimensions for MBAC-DVBF-REWO models.

different latent dimensions settings is also studied in Figure 5.10b. Overall, performance increases with higher latent dimensions. On the other hand, even with lower dimensions, the pre-trained models (MBAC-DVBF-REWO-BU and MBAC-DVBF-REWO-PT) still converge relatively quickly, and the difference for pre-trained-models between $y = 8, z = 6$ and $y = 32, z = 8$ in Figure 5.7 is negligible.

6. Conclusion

This thesis presents a methodology that enables one to transfer a learned DSSM from one environment to the others in a pre-defined curriculum. Training DSSM from scratch is challenging as due to the rate-distortion theorem, which makes it time and data consuming to find a solution that balances between rate and distortion trade-off. To this end, a new inference DSSM architecture was introduced to accept varying observational inputs. To be more specific, this structure is an extension of DVBF model with a multi-level encoding-decoding scheme. In the transition phase between two environments, part of the architecture including z-encoder, transition and initial networks are loaded and trained from the last optimal checkpoint; others, which may have different structures, are trained from scratch. Next, inspired by the Bayesian update principle, an additional constraint is introduced to enforce these shared parameters stay close to the pre-trained ones. The entire optimisation problem is then solved by following the constrained optimisation formulation. Experimental results validate that a learned DSSM from a simple raw state pendulum can be reused to accelerate the training speed of different systems that have different observational inputs (pixel pendulum) or even dynamical systems (cart pole). The proposed pre-training procedure works for both offline and online with reinforcement learning settings.

In the case where only limited samples are available, this thesis opens up new possibilities for leveraging pre-trained DSSM knowledge to tackle more challenging systems. One such possibility is extending beyond simulations, where the learned DSSM can be reused on real-world robots, thus reducing the number of interactions on the real robots.

A. Derivations

In this chapter we will give the detail derivations for both the DVBF model with multi-level structure and the bayesian update rule.

A.1. Deep Variational Bayes Filters with Multi Level Structure

We begin this section with the generative model:

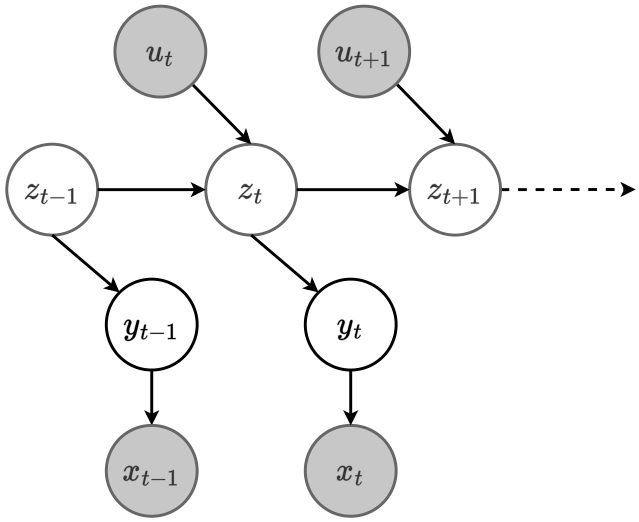


Figure A.1.: State-space models with multi-level structure.

In addition to the latent variable z, we introduce a so called auxiliary observation variable y. However, this variable is unobservable and need to be infered in a similar fashion to z.

Comparison with Klushyn, et al. (2021) [6] In this work, we do not specify any structure for the latent variable z. Meawhile, Klushyn, et al. (2021) [6] explicitly allocates part of the latent variable for velocity information via an encoding matrix H. This makes our framework become more generic, ones can put extra information in the observation space without paying attention on H.

The joint likelihood is as follow:

$$\begin{aligned}
p(x_{1:T}|u_{1:T}) &= \prod_{t=1}^{T} p(x_t|x_{1:t-1}, u_{1:t}) \\
&= \int\int \prod_{t=1}^{T} p(x_t, y_t, z_t|x_{1:t-1}, y_{1:t-1}, z_{1:t-1}, u_{1:t}) dz_{1:T} dy_{1:T} \\
&= \int\int \prod_{t=1}^{T} p(x_t|x_{1:t-1}, y_{1:t}, z_{1:t}, u_{1:t}) p(y_t|x_{1:t-1}, y_{1:t-1}, z_{1:t-1}, u_{1:t}) \\
&\qquad\qquad p(z_t|x_{1:t-1}, y_{1:t-1}, z_{1:t-1}, u_{1:t}) dz_{1:T} dy_{1:T} \\
&= \int\int p(x_1|y_1) p(y_1|z_1) p(z_1) \prod_{t=2}^{T} p(x_t|y_t) p(y_t|z_t) p(z_t|z_{t-1}, u_t) dz_{1:T} dy_{1:T}.
\end{aligned}$$

Similar to the SSMs derivation, Markov's property has been applied:

$$\begin{aligned}
p(z_t|x_{1:t-1}, y_{1:t-1}, z_{1:t-1}, u_{1:t}) &= p(z_t|z_{t-1}, u_t) \\
p(y_t|x_{1:t-1}, y_{1:t-1}, z_{1:t-1}, u_{1:t}) &= p(y_t|z_t) \\
p(x_t|x_{1:t-1}, y_{1:t}, z_{1:t}, u_{1:t}) &= p(x_t|y_t).
\end{aligned}$$

Due to the intractability, we can only optimise the following ELBO:

$$\begin{aligned}
\log p(x_{1:T}|u_{1:T}) &= \log \int\int p(x_1|y_1) p(y_1|z_1) p(z_1) \frac{q_\phi(y_1|x_1)}{q_\phi(y_1|x_1)} \frac{q_\phi(z_1|y_1)}{q_\phi(z_1|y_1)} \\
&\qquad \prod_{t=2}^{T} p(x_t|y_t) p(y_t|z_t) p(z_t|z_{t-1}, u_t) \frac{q_\phi(y_t|x_t)}{q_\phi(y_t|x_t)} \frac{q_\phi(z_t|y_t, z_{t-1}, u_t)}{q_\phi(z_t|y_t, z_{t-1}, u_t)} dz_{1:T} dy_{1:T} \\
&\geq \mathbb{E}_{y_1 \sim q_\phi(y_1|x_1)} \mathbb{E}_{z_1 \sim q_\phi(z_1|y_1)} \left[\log p(x_1|y_1)\right] \\
&\quad - \mathrm{KL}(q_\phi(z_1|y_1)||p(z_1)) - \mathrm{KL}(q_\phi(y_1|x_1)||p(y_1|z_1)) \\
&\quad + \sum_{t=2}^{T} \mathbb{E}_{y_t \sim q_\phi(y_t|x_t)} \mathbb{E}_{z_t \sim q_\phi(z_t|y_t, z_{t-1}, u_t)} \left[\log p(x_t|y_t)\right] \\
&\qquad - \mathrm{KL}(q_\phi(z_t|y_t, z_{t-1}, u_t)||p(z_t|z_{t-1}, u_t)) - \mathrm{KL}(q_\phi(y_t|x_t)||p(y_t|z_t)) \\
&= \mathcal{L}_{\mathrm{ELBO}}.
\end{aligned}$$

Similar to DVBF, we factorise the posterior as follow:

$$q_\phi(z_t|y_t, z_{t-1}, u_t) \propto q_{\text{inv\_meas}}(z_t|y_t) \times q_{\text{trans}}(z_t|z_{t-1}, u_t).$$

By following the constrained optimisation formalisation [5, 6], we can rewrite the ELBO as follow:

$$\begin{aligned}
\min_{\theta,\phi} \quad & \mathrm{KL}(q_\phi(z_{1:T}|y_{1:T}, u_{1:T})||p(z_{1:T}|u_{1:T})) + \mathrm{KL}(q_\phi(y_{1:T}|x_{1:T})||p(y_{1:T}|z_{1:T})) \\
\text{s.t.} \quad & \mathbb{E}_{y_{1:T} \sim q_\phi(\cdot)} \mathbb{E}_{z_{1:T} \sim q_\phi(\cdot)} \left[-\log p(x_{1:T}|y_{1:T})\right] \leq \xi_{\text{rec}}.
\end{aligned} \tag{A.1}$$

A.2. Relation between Hessian and Fisher Information Matrix

To begin this section, we recall the Hessian matrix under posterior distribution $\log p(\psi|x)$:

$$H_{\log p(\psi|x)}(\psi^*) = H_{\log p(x|\psi)}(\psi^*) + H_{\log p(\psi)}(\psi^*)$$

The following derivation will only put the focus on the log-likelihood term since the other can be later seen as the regularisation matrix (if $H_{\log p(\psi)}(\psi^*) = I$ it will be equivalent to L_2 regularisation). Then, by applying set of derivative rules and omitting the function parameter ψ^*, we get:

$$\begin{aligned}
H_{\log p(x|\psi)} &= J(\nabla_\psi \log\ p(x|\psi)) \\
&= J\left[\frac{\nabla_\psi p(x|\psi)}{p(x|\psi)}\right] \\
&= \frac{H_{p(x|\psi)} p(x|\psi) - \nabla_\psi p(x|\psi) \nabla_\psi p(x|\psi)^T}{p(x|\psi) p(x|\psi)} \\
&= \frac{H_{p(x|\psi)}}{p(x|\psi)} - \frac{\nabla_\psi p(x|\psi)}{p(x|\psi)} \frac{\nabla_\psi p(x|\psi)^T}{p(x|\psi)}
\end{aligned}$$

and by taking the expectation of the Hessian, we arrive:

$$\begin{aligned}
\mathbb{E}_{p(x|\psi)}[H_{\log p(x|\psi)}] &= \mathbb{E}_{p(x|\psi)}\left[\frac{H_{p(x|\psi)}}{p(x|\psi)} - \frac{\nabla_\psi p(x|\psi)}{p(x|\psi)} \frac{\nabla_\psi p(x|\psi)^T}{p(x|\psi)}\right] \\
&= \int \frac{H_{p(x|\psi)}}{p(x|\psi)} p(x|\psi) dx - \mathbb{E}_{p(x|\psi)}\left[\frac{\nabla_\psi p(x|\psi)}{p(x|\psi)} \frac{\nabla_\psi p(x|\psi)^T}{p(x|\psi)}\right] \\
&= H_{\int p(x|\psi) dx} - \mathbb{E}_{p(x|\psi)}[\nabla_\psi \log p(x|\psi) \nabla_\psi \log p(x|\psi)^T] \\
&= -F.
\end{aligned}$$

A.3. Relation between Fisher Information Matrix and KL-divergence

We first recall the definition of KL divergence:

$$\mathrm{KL}[p(x|\psi)||p(x|\psi^*)] = \mathbb{E}_{p(x|\psi)}[\log p(x|\psi)] - \mathbb{E}_{p(x|\psi)}[\log p(x|\psi^*)].$$

Taking the first derivative w.r.t ψ^*, we get:

$$\begin{aligned}
\nabla_{\psi^*} \mathrm{KL}[p(x|\psi)||p(x|\psi^*)] &= \nabla_{\psi^*} \mathbb{E}_{p(x|\psi)}[\log p(x|\psi)] - \nabla_{\psi^*} \mathbb{E}_{p(x|\psi)}[\log p(x|\psi^*)] \\
&= -\mathbb{E}_{p(x|\psi)}[\nabla_{\psi^*} \log p(x|\psi^*)] \\
&= -\int p(x|\psi) \nabla_{\psi^*} \log p(x|\psi^*) dx.
\end{aligned}$$

The Hessian is thus:

$$\begin{aligned} H_{\mathrm{KL}[p(x|\psi)||p(x|\psi^*)]} &= \nabla^2_{\psi^*} \mathrm{KL}[p(x|\psi)||p(x|\psi^*)] \\ &= -\int p(x|\psi)\, \nabla^2_{\psi^*} \log p(x|\psi^*)|_{\psi^*=\psi} dx \\ &= -\int_x p(x|\psi) H_{\log p(x|\psi)} dx \\ &= -\mathbb{E}_{p(x|\psi)}[H_{\log p(x|\psi)}] \\ &= F. \end{aligned}$$

Using Taylor approximation, the KL divergence can be re-written as follow:

$$\begin{aligned} \mathrm{KL}(p(x|\psi)\|p(x|\psi^*)) &= \mathrm{KL}(p(x|\psi)\|p(x|\psi)) + \nabla_{\psi^*=\psi}\mathrm{KL}[p(x|\psi)||p(x|\psi^*)]^T(\psi-\psi^*) \\ &\quad + (\psi-\psi^*)^T F(\psi^*)(\psi-\psi^*) + \mathcal{O}(\Delta\psi^3) \\ &= -\mathbb{E}_{p(x|\psi)}[\nabla_{\psi^*=\psi} \log p(x|\psi^*)]^T(\psi-\psi^*) \\ &\quad + (\psi-\psi^*)^T F(\psi^*)(\psi-\psi^*) + \mathcal{O}(\Delta\psi^3) \\ &= (\psi-\psi^*)^T F(\psi^*)(\psi-\psi^*) + \mathcal{O}(\Delta\psi^3). \end{aligned}$$

where we assume $\psi^* = \arg\min_\psi -\log p(x|\psi)$, hence $\nabla_{\psi=\psi^*}[-\log p(x|\psi)] = 0$.

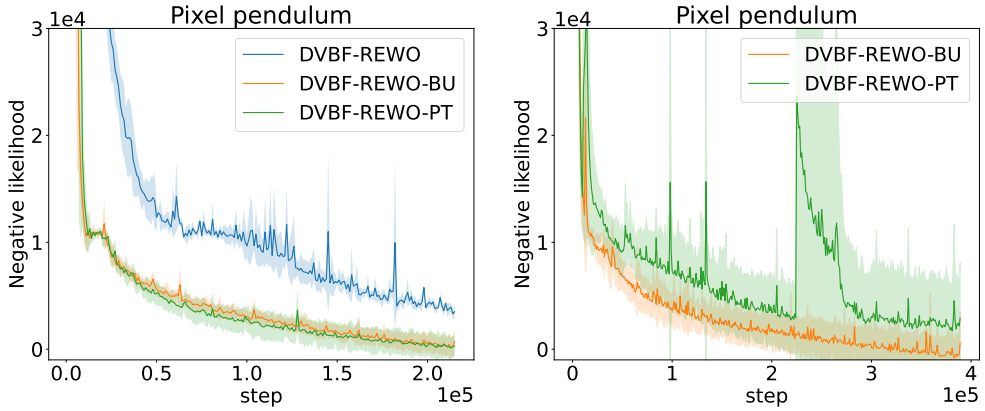
B. Supplementary for Experiments

B.1. Heuristic for Choosing Bayesian Update Hyperparameters

In this section, we demonstrate the applied heuristic methods for selecting curriculum related hyperparameters for training sequence models with bayesian update.

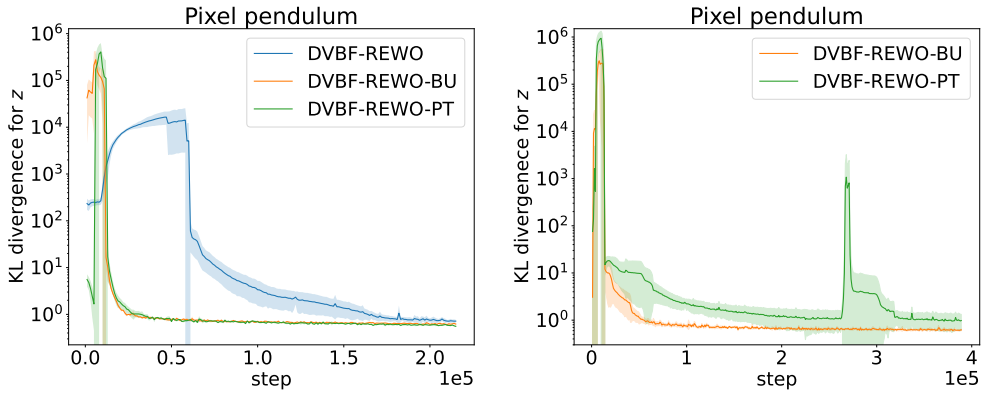
- Initial Lagrange $\lambda_{\text{bu}}^{\text{init}}$: this can be chosen by doing grid search, a good value results in a steep training curve in the initial phase. If there is a huge gap between two environments, too big value will slow down the training progress as we expect the current parameters to stay really close to the previous pre-trained one.
- Curriculum threshold ξ_{bu}: we also perform grid search for this parameter. It has a similar effect as the previous one, too small value will restrict the parameter space significantly and therefore slowing down the training progress. In contrast, For a too large value, in theory, the bayesian update will have no effect, and will perform similarly to the only pre-trained model.

B.2. Additional Experimental Results



(a) Negative likelihood with $z = 6$

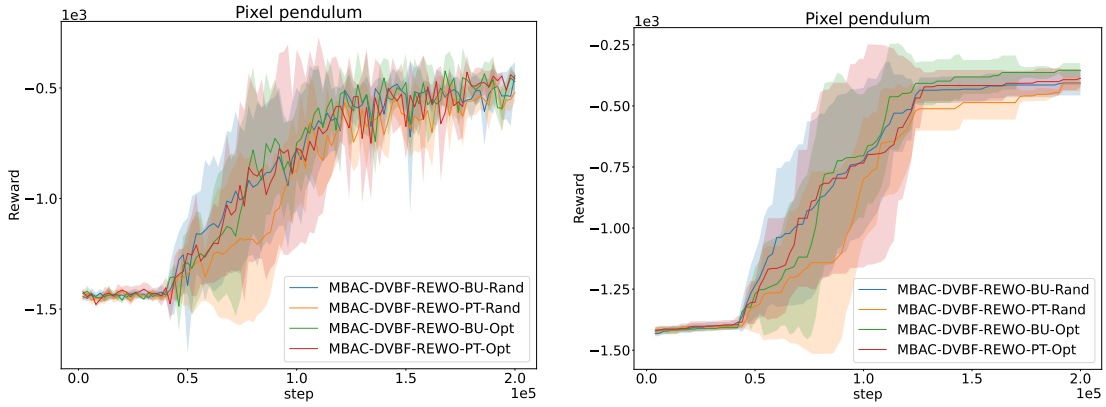
(b) Negative likelihood with $z = 4$



(c) KL for z with $z = 6$

(d) KL for z with $z = 4$

Figure B.1.: Sequence model training results on pixel pendulum environment w.r.t ELBO metric. DVBF model with REWO training scheme described in Section 2.2.4 is used to compare between non-pre-trained model and pre-trained models with and without bayesian update. The pre-trained models are trained on the raw state pendulum environment. The latent dimensions are set to $d_y = 32$, $d_z = \{4, 6\}$.



(a) Reward

(b) Best reward

Figure B.2.: Comparison between pre-trained models (on the raw state pendulum environment) with optimal policy (*-Opt) and random policy (*-Rand) for the pixel pendulum environment.

B.3. Hyperparameters Table

Table B.1.: Hyperparameters for Cart-pole Sequence Model Learning

Parameters Name	Value
Observations d_x	3 [cart_pos, sin(pole_pos), cos(pole_pos)]
Time steps	8
Actions d_u	1
Auxiliary variables d_y	8
Latents	3
Initial observations	3
$q(y_t\|x_t)$	FC 512, 512, ReLU activation, output Tanh activation
$p(x_t\|y_t)$	FC 512, 512, ReLU activation
$q(z_t\|y_t)$	FC 256, 256, ReLU activation
$p(y_t\|z_t)$	FC 256, 256, ReLU activation
$\lambda_{\text{rec}}^{init}$	10
ν_{rec}	1
ξ_{rec}	-2
$\lambda_{\text{bu}}^{init}$	5
ν_{bu}	100
ξ_{bu}	0.001
Batch size	512
Optimiser	Adam with lr $=$ 3e$-$4

Table B.2.: Hyperparameters for Pixel Pendulum Sequence Model Learning

Parameters Name	Value
Observations d_x	$32 \times 32 \times 3$
Time steps	40
Actions d_u	1
Auxiliary variables d_y	8
Latents	4 or 6
Initial observations	3
$q(y_t\|x_t)$	Conv $32 \times 5 \times 5$ (stride 2), $64 \times 5 \times 5$ (stride 2), $128 \times 5 \times 5$ (stride 2), ReLU activation, output Tanh activation
$p(x_t\|y_t)$	Deconv in reverse order
$q(z_t\|y_t)$	FC 256, 256, ReLU activation
$p(y_t\|z_t)$	FC 256, 256, ReLU activation
$\lambda_{\text{rec}}^{init}$	100
ν_{rec}	1
ξ_{rec}	3
$\lambda_{\text{bu}}^{init}$	4
ν_{bu}	100
ξ_{bu}	0.3
Batch size	128
Optimiser	Adam with lr $= 3\text{e}{-4}$

Table B.3.: Hyperparameters for MBAC-DVBF

Parameters Name	Value
Observations d_x	$32 \times 32 \times 3$
Trajectory steps	200
Data collection frequency	200
Warm-up steps	40000
Replay buffer initial data	4000 steps
Replay buffer size	150000
Reward network	FC 256, 256, ReLU activation
Policy network	FC 128, 128, ReLU activation
Critic network	FC 256, 256, ReLU activation
Batch size	128
Policy optimiser	Adam with lr $= 1\text{e}{-4}$
Critic optimiser	Adam with lr $= 3\text{e}{-4}$
Sequence model optimiser	Adam with lr $= 5\text{e}{-4}$

List of Figures

List of Tables

Bibliography

[1] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2014.

[2] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data". In: *International Conference on Learning Representations*. 2017.

[3] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International conference on learning representations*. 2017.

[4] N. Das, M. Karl, P. Becker-Ehmck, and P. van der Smagt. *Beta DVBF: Learning State-Space Models for Control from High Dimensional Observations*. 2019. arXiv: 1911.00756.

[5] D. J. Rezende and F. Viola. *Taming VAEs*. 2018. arXiv: 1810.00597.

[6] A. Klushyn, R. Kurle, M. Soelch, B. Cseke, and P. van der Smagt. "Latent Matters: Learning Deep State-Space Models". In: *Advances in Neural Information Processing Systems*. 2021.

[7] D. L. Silver, Q. Yang, and L. Li. "Lifelong machine learning systems: Beyond learning algorithms". In: *2013 AAAI spring symposium series*. 2013.

[8] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[9] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor. "A deep hierarchical approach to lifelong learning in minecraft". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[10] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.

[11] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. "Variational continual learning". In: *arXiv preprint arXiv:1710.10628* (2017).

[12] F. Zenke, B. Poole, and S. Ganguli. "Continual learning through synaptic intelligence". In: *International conference on machine learning*. PMLR. 2017, pp. 3987–3995.

[13] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[14] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. "Automated curriculum learning for neural networks". In: *international conference on machine learning*. PMLR. 2017, pp. 1311–1320.

[15] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. "Teacher–student curriculum learning". In: *IEEE transactions on neural networks and learning systems* 31.9 (2019), pp. 3732–3740.

[16] M. E. Taylor and P. Stone. "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7 (2009).

[17] E. Parisotto, J. L. Ba, and R. Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342* (2015).

[18] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. "Sim-to-real transfer of robotic control with dynamics randomization". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[19] A. Kumar, Z. Fu, D. Pathak, and J. Malik. "Rma: Rapid motor adaptation for legged robots". In: *arXiv preprint arXiv:2107.04034* (2021).

[20] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[21] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[22] G. E. Hinton and R. R. Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[23] Y. Bengio. "Deep learning of representations for unsupervised and transfer learning". In: *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings. 2012, pp. 17–36.

[24] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, et al. "Unsupervised and transfer learning challenge: a deep learning approach". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. JMLR Workshop and Conference Proceedings. 2012, pp. 97–110.

[25] F. Huszár. "Note on the quadratic penalties in elastic weight consolidation". In: *Proceedings of the National Academy of Sciences* 115.11 (2018), E2496–E2497.

[26] J. Peters, K. Mulling, and Y. Altun. "Relative entropy policy search". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[27] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[28] T. Hoang and N. A. Vien. "Bayes-Adaptive Deep Model-Based Policy Optimisation". In: *arXiv preprint arXiv:2010.15948* (2020).

[29] M. Karl, M. Soelch, P. Becker-Ehmck, D. Benbouzid, P. van der Smagt, and J. Bayer. *Unsupervised Real-Time Control through Variational Empowerment*. 2017. arXiv: 1710.05101.

[30] R. Bellman. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515.

[31] V. Konda and J. Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[32] J. Peters and S. Schaal. "Natural actor-critic". In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190.

[33] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[35] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[36] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[37] M. Deisenroth and C. E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 2011, pp. 465–472.

[38] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. "Dream to control: Learning behaviors by latent imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[39] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. "Learning to fly via deep model-based reinforcement learning". In: *arXiv preprint arXiv:2003.08876* (2020).

[40] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. "Mastering Diverse Domains through World Models". In: *arXiv preprint arXiv:2301.04104* (2023).

[41] S. Levine and V. Koltun. "Guided policy search". In: *International conference on machine learning*. PMLR. 2013, pp. 1–9.

[42] S. Levine and P. Abbeel. "Learning neural network policies with guided policy search under unknown dynamics". In: *Advances in neural information processing systems* 27 (2014).

[43] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[44] M. Opper. "A Bayesian Approach to On-Line Learning". In: *On-Line Learning in Neural Networks*. USA: Cambridge University Press, 1999, pp. 363–378. ISBN: 0521652634.

[45] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. "Reply to Huszár: The elastic weight consolidation penalty is empirically valid". In: *Proceedings of the National Academy of Sciences* 115.11 (2018), E2498–E2498.

[46] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. "Ladder variational autoencoders". In: *Advances in neural information processing systems* 29 (2016).

[47] B. Chen, K. Huang, S. Raghupathi, I. Chandratreya, Q. Du, and H. Lipson. "Discovering state variables hidden in experimental data". In: *arXiv preprint arXiv:2112.10755* (2021).

[48] A. Vahdat and J. Kautz. "NVAE: A deep hierarchical variational autoencoder". In: *Advances in neural information processing systems* 33 (2020), pp. 19667–19679.

[49] A. Klushyn, N. Chen, R. Kurle, B. Cseke, and P. van der Smagt. "Learning hierarchical priors in vaes". In: *Advances in neural information processing systems* 32 (2019).

[50] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[51] E. Todorov, T. Erez, and Y. Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.