UNIVERSITEIT LEIDEN

MATHEMATISCH INSTITUUT

MASTER THESIS

Deep State-Space Models in Multi-Agent Systems

Author:
Pararawendy Indarjo

Supervisors:
Dr. Michael Kaisers
(CWI)
Prof. Dr. Peter D. Grünwald
(MI Leiden / CWI)

Defended on July 11th, 2019

Abstract

Multi-agent interactions are driven by both game dynamics and joint behavior. This implies that any specific agent faces a task in which the optimal policy is the best response to the opponents' joint policy, which is not directly observable and must therefore be inferred from observations. Motivated by this setting, we propose a new formulation of deep state-space models (DSSMs) in multi-agent systems (MAS). This formulation results in models that can be used to represent the environment dynamics from an individual agent's point of view, and ultimately to predict the other agents' future actions. The models are built upon the framework of latent variable models, armed with the variational inference principle in regard to approximating the true (but intractable) posterior distributions. In addition, we explore extensions that make the model richer and thus more flexible by inducing covariance structures on the latent variable distributions. This leaves us with several models available, corresponding to the rank of the covariance inducing matrix. We then propose to choose the best-parsimonious model amongst them via the minimum description length (MDL) principle and illustrate the resulting performance on a given simple problem. Finally, we also demonstrate how the models explicitly affect the agent's learned policy in a multi-agent reinforcement learning task. From the empirical implementation, we found that the models perform reasonably well to infer both the environment dynamics and the opponent's policy. Overall, we believe that the proposed models are highly flexible, since the models can be used in any interaction setting of MAS (collaborative/competitive), partially observable environments, and any reinforcement learning paradigm (model-based/model-free), not to mention the models' advantageous generative characteristics.

Acknowledgement

There are a number of people to whom I would like to express my gratitude, due to their valuable support (in various ways) regarding this thesis writing.

The first and foremost person is Dr. Michael Kaisers. I would like to thank Michael for accepting me in this master thesis internship at CWI and being my supervisor, even though the original thesis project he offered was a bit off from the work I eventually carried out. Michael gave me complete independence, allowing me to pursue what is an interesting topic to be worked out. His remarkable words are like this, *"Since it will be your thesis, it is best to set you on the driver seat, do what you want to do"*. Also, thank you for introducing me to the fascinating field of reinforcement learning: a learning paradigm that is right in the middle of supervised and unsupervised learning.

The next person is my second supervisor, Prof. Peter D. Grünwald. I would like to say thank you to Peter, for his willingness to dedicate a part of his extremely tight schedules to supervise me, as well as for his commitment to ensuring that I can graduate on time.

Masoume, thank you for the earlier reviews on my thesis ideas, for the lengthy brainstorming phase in order me to find the best topic, as well as the proofreading and suggestions on the draft. Next, Majid, as one of a few friends of mine at CWI. Thank you for the good spirit and calming words you gave. Once, when I still did not find any topic, he said, *"Just keep calm and don't worry too much, there are many PhD students here, in their second, or even third year, they still don't have any clue on what to do"*, such a healing of anxiousness.

Of course, I would like to thank my parents: Ibu & Bapak; and my little family: Niken & Alfath, especially my wife Niken. Thank you for your countless *du'a*, for always believing me and making me strong, and for your sincere companion. Niken's words, *"You can do it! All difficulties shall pass! Everything is gonna be alright!"*, are always successful to boost up my spirit whenever I got stuck while writing this thesis.

Last but not least, I would like to express my gratitude to Kingdom of the Netherlands. My master study in this world-class university, as well as this vibrant country, would not be possible without StuNed Scholarship program you hosted.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many real-life applications can be represented as multi-agent systems (MAS). In robotics, MAS can be found when one designs several distinct robot units to solve problems with high complexity. Loosely speaking, any system comprised of several autonomous elements (called agents) that interact in the same environment constitutes a MAS (Bloembergen et al., 2015). We see that MAS are very flexible according to this loose definition. A broad range of use cases, from designing strong network security to building smart cities, thus forms a set of examples of MAS applications.

1.1 Problem statement

In MAS, the dynamics of the environment is governed by the joint policies of all agents in the system. In addition, each agent executes actions based on observations it perceives from the environment. In other words, each agent through its policy only has partial control over the environment change. Subsequently, both environment dynamics and agents' joint behavior drive the interaction between the agents. This implies that any specific agent faces a task in which the optimal policy is the best response to opponents' policy, which is not directly observed and must therefore be inferred from observations.

This thesis addresses the challenge above by modelling the environment dynamics as well as the other agents' policy using a formulation of Deep State-Space Models (DSSMs). The formulation incorporates self-introduced latent variables to help modelling the problem. Despite being very well suited for such a setting with the occurrence of latent variables in a sequential setting, the models presented in this thesis, to the best of our knowledge, is the first formulation of such approach.

1.2 Research questions

The research questions of this thesis are the following:

- Can we formulate deep state-space models in the context of MAS? [Chapter 3]
- Can we extend the models to become more flexible and subsequently choose the best-parsimonious models configuration via the minimum description length principle? [Chapter 4]
- In what (simple) problem the proposed models can be implemented? [Chapter 5]
- Can we utilize the models to explicitly affect the agent's learned policy in a multi-agent reinforcement learning task? [Chapter 6]

1.3 Related work

Study of constructing the models of other agents, also known as opponent modelling, has a long history in multi-agent learning (Hernandez-Leal et al., 2017; Albrecht and Stone, 2018). Amongst the earliest works, there is a method called fictitious play, which constructs a simple opponent model based on their empirical actions frequency (Brown, 1951). Another important development in this study stream is the proposal of joint action learners (JAL) modelling paradigm (Claus and Boutilier, 1998). Moreover, with respect to the modelling approach, there are methods built upon game-theoretic evolutionary dynamics (Bloembergen et al., 2015) and Bayesian approach (Chalkiadakis and Boutilier, 2003).

Regarding the method used in this thesis, variational inference principle has been extensively used in the field of reinforcement learning, especially in the single-agent case. Particularly, variational inference is naturally evoked when one views a sequential decision making under uncertainty like a Markov decision process (MDP) as a probabilistic inference (Levine, 2018). The technique is firstly utilized in reinforcement learning domain as an approach to model the environment dynamics (Furmston and Barber, 2010). The follow-up works include various variational inference base policy search algorithms (Neumann, 2011; Rawlik, Toussaint, and Vijayakumar, 2012; Levine and Koltun, 2013). In those works, a lower bound of the log likelihood of optimality is derived based on specific assumptions on the reinforcement learning task being considered.

Quite recently, there has been a growing interest to extend the use cases of variational inference principle in the multiple agents case of reinforcement learning. Several of the related studies are mentioned as follows. Firstly, there is a regularized opponent model with maximum entropy objective (ROMMEO), which formulates multi-agent reinforcement learning into Bayesian inference, resulting in a version of maximum entropy reinforcement learning (Tian et al., 2019). Unlike our method, which also models the environment dynamics, ROMMEO only models the agents' policies. Secondly, a method called probabilistic recursive reasoning captures how opponents believe about what the agent believes (Wen et al., 2019). The main difference compared to our method is the correlated-recursive assumption on the agents' joint policies it uses. Next, there is a method that utilizes variational inference in multi-step generative models. In spite of having a similar approach with ours, i.e. modelling the environment dynamics, the method is designed only for deterministic dynamics (Krupnik, Mordatch, and Tamar, 2019). Lastly, there is also a method based on graph-structured variational recurrent neural network (Graph-VRNN) (Sun et al., 2019). The method is similar to ours in two aspects: it models the transition dynamics and it can be used in the partially observable setting. Yet, there is a significant difference, since the model is discriminative rather than generative (as ours).

1.4 Outline

The outline of this thesis is arranged as follows. The next chapter (Chapter 2) presents the material of the building blocks used by the models proposed in this thesis: latent variable models, state-space models, and multi-agent systems. To mention earlier, materials of the first two topics largely follow the unified treatment of (Fraccaro,

2018). The follow up chapter (Chapter 3) is the core part of this thesis. There, we discuss the proposed models in detail. Next, in Chapter 4, we explore extensions to make the models richer and thus more flexible. Still in this chapter, we implement models selection via the Minimum Description Length (MDL) principle in order to pick the best-parsimonious configuration of the models. This is because we do not want the models to overfit the training data. The following chapter (Chapter 5) presents a practical implementation of the models on a simple problem. Right before the last chapter, in Chapter 6, we give one use case of the models in a multi-agent reinforcement learning task, in which we demonstrate how the model explicitly affects the agent's learned policy. Finally, in the last chapter (Chapter 7), conclusion and possible future work directions are presented.

Chapter 2

Background

This chapter presents the background materials that constitute the building blocks of the models we propose in Chapter 3. First, we discuss a class of generative models which incorporates self-introduced latent (unobserved) variables, namely latent variable models. Based on the way they are constructed, these models naturally suit with static type of data, i.e. data without temporal structure. Since the data that we want to model in this thesis is naturally sequential rather than static, we follow-up the section by presenting state-space models (SSMs). There, we can see that SSMs can be viewed as the extension of latent variable models in sequential data. To conclude this chapter, we discuss multi-agent systems (MAS), as the scope of the models proposed in this piece of work.

2.1 Latent variable models

In this section, we discuss two essential concepts related to latent variable models: the models introduction and how to perform posterior inference on such models. We also present variational auto-encoders as one important application of the models.

2.1.1 Latent variable models as generative models

Suppose we are exposed with some observation data x and interested in modelling its distribution $p(x)$. Latent variable models refer to statistical models which utilize a self-introduced unobserved latent variable z to help modelling $p(x)$. One such way to go is by defining both a prior distribution $p(z)$ and a conditional distribution $p(x|z)$. These two distributions are later used to define the joint distribution $p(x, z)$ through a designed factorization as can be found in the following.

$$p(x, z) = p(x|z)p(z) \tag{2.1}$$

In this way, latent variable models are an indirect modelling approach compared to when one models $p(x)$ directly. In this thesis, we assume that z is a continuous random variable. This latent variable z can be interpreted as somewhat similar to features of the observed data. For example, if our data x is a set of car pictures, then z could be a higher level characteristics of a car could have, such as car type (sedan, SUV, minivan, etc), color, and bumper installation. Nevertheless, recall that the variable z is truly unobserved (we just imaginary made it up), therefore please bear in mind that the aforementioned crystal clear interpretations are merely for illustration purpose.

It may first appear that introducing such latent variables only results in an additional modelling task we have at hand, and this is indeed true when the data distribution is relatively simple. Yet, if the data we want to model is complex (e.g. due to the high dimensionality of the data), we can gain a lot by introducing such latent variables (Pu et al., 2016). Because it allows us to represent the complex distribution $p(x)$ as marginalization of a more tractable joint distribution, whose components $p(x|z)$ and $p(z)$ are typically much simpler to define. More formally,

$$\begin{aligned} p(x) &= \int p(x,z)dz \\ &= \int p(x|z)p(z)dz. \end{aligned} \tag{2.2}$$

These latent variable models that learn joint probability distribution with a certain way of factorization are a class of generative models. Given the latent variable z, it can generate a new data observation x by drawing one from the conditional distribution $p(x|z)$, hence the name.

2.1.2 Posterior inference via variational principle

As we observe more data, we might want to refine our (prior) belief on the latent distribution of our model, which in turn leads to a better joint probability (2.1) we have at hand. We do this by computing the posterior distribution of latent variables, i.e. $p(z|x)$ using Bayes rule

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{2.3}$$

This posterior distribution represents our updated belief about the latent variables after having seen the data. However, in many cases, this distribution is intractable to compute. This may be due to the integral in (2.2) that appears in the denominator of (2.3), which may have no analytic solution or include exponential terms to compute.

In variational inference principle, the true posterior $p(z|x)$ is approximated by a newly introduced *variational distribution* $q(z)$. We specify a family $\mathcal{Q}$ of densities over the latent variables. Each $q(z) \in \mathcal{Q}$ is a candidate approximation to the exact posterior $p(z|x)$. Our goal is to find the best candidate, the one closest in KL divergence to the exact posterior. Posterior inference thus amounts to solving the following optimization problem (Blei, Kucukelbir, and McAuliffe, 2017).

$$q^*(z) = \arg\min_{q(z)\in\mathcal{Q}} \mathrm{KL}(q(z)||p(z|x)) \tag{2.4}$$

Once found, $q^*(\cdot)$ is the best approximation of the true posterior, within the family of $\mathcal{Q}$. However, this optimization is not computable since the right hand side of it includes the intractable term $p(x)$. To see this,

$$\begin{aligned} \mathrm{KL}(q(z)||p(z|x)) &= \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(z|x)] \\ &= \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(x|z)p(z)] + \log p(x) \\ &= \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(x,z)] + \log p(x) \end{aligned} \tag{2.5}$$

Or equivalently

$$\log p(x) - \mathrm{KL}(q(z)||p(z|x)) = \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log q(z)] \tag{2.6}$$

Since KL divergence is always non-negative, from here we see that maximizing the right hand side of (2.6), which is often called the ELBO (Evidence Lower BOund) approximately achieves two things at once:

1. Minimizing the objective of variational inference that we want to minimize in the first place, i.e. $\mathrm{KL}(q(z)||p(z|x))$
2. Maximizing the log likelihood of the data, i.e. $\log p(x)$ (note that ELBO is always smaller or equal to log of *evidence* $\log p(x)$, which explains its name)

In practice, $\mathcal{Q}$ is often restricted to a particular parametric family for which the ELBO is tractable or simple to compute (e.g Gaussian distributions), and the maximization with respect to $q(z)$ is therefore a maximization with respect to the parameters of the family. $\mathcal{Q}$ needs to be flexible enough to provide a good approximation to the posterior distribution, but simple enough to make the ELBO easy to compute (Fraccaro, 2018).

2.1.3 Variational auto-encoders

Variational auto-encoders (VAEs) are a class of latent variable models (Kingma and Welling, 2013; Rezende, Mohamed, and Wierstra, 2014). In its original setting, the goal is to fit a distribution over some static (non-temporal) data x with the help of unobserved continuous latent variable z, whose factorization coincides with (2.1). The original goal of maximizing $\log p(x)$ is replaced by instead maximizing the ELBO, resorting exactly to the approach presented in the Section 2.1.2.

A standard VAE would consists of two building blocks: a generative model and an inference model. The **generative model** comprises the factorization in (2.1), with the prior $p(z)$ typically set to a standard multivariate Gaussian. Whilst the conditional distribution $p(x|z)$, which is also called the *decoder*, depends on the data modelled. It can be Bernoulli if the data is binary, or Gaussian if the data is real valued. A graphical model example of a generative model with the decoder being a Gaussian with diagonal structured covariance matrix is given by the following:

$$\begin{aligned} p(z) &\sim \mathcal{N}(z; 0, I) \\ p_\theta(x|z) &\sim \mathcal{N}(x; \mu_\theta, \mathrm{diag}(\sigma_\theta^2)) \end{aligned} \tag{2.7}$$

This graphical model can guide us when we want to generate new data from the model. First, we draw z from a standard Gaussian, then we use it to fix the conditional distribution $p(x|z)$, by which we draw our new data x.

The second building block is the **inference model**, also known as the *encoder*. This block is the distinctive part of VAEs, which distinguishes it from the classical variational inference method as derived in Section 2.1.2. In VAEs, the variational distribution, whose main task is to approximate the true posterior distribution $p(z|x)$, is explicitly conditioned on data, i.e. $q(z|x)$. In practice, this distribution is commonly modelled as a Gaussian with diagonal structured covariance matrix.

$$q_\phi(z|x) \sim \mathcal{N}(z; \mu_\phi, \mathrm{diag}(\sigma_\phi^2)) \tag{2.8}$$

There is another distinctive feature of VAEs, namely all learned distributions, i.e. $p_\theta(x|z)$ and $q_\phi(z|x)$, are parameterized by neural networks. Note that with parameterization by neural networks we mean that the neural networks emit the parameters of the distribution. Therefore, the subscript θ and ϕ in (2.7) and (2.8) each represents the parameters of the corresponding neural networks. In other words, it means that μ_ϕ and σ^2_ϕ in (2.8) are the outputs of neural network with a set of parameters (weights and biases) ϕ and input x. This is analogue for (2.7).

To conclude this section, let us discuss the training procedure of VAEs. First, we rewrite the ELBO expression on the right hand side of (2.6) to better suit VAEs as follows.

$$\begin{aligned} ELBO_{\theta,\phi} &= \mathbb{E}_{q_\phi}[\log p_\theta(x,z)] - \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi}[\log p(z)] - \mathbb{E}_{q_\phi}[\log q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z)) \end{aligned} \tag{2.9}$$

In the training phase, the objective is to maximize (2.9). As both the decoder and encoder are defined by neural networks, we can efficiently compute gradients of the ELBO with respect to θ and ϕ using the back-propagation algorithm (Rumelhart, Hinton, and Williams, 1986). Importantly, both set of parameters can be updated jointly in a single optimization step. To be able to back-propagate through the latent variable z, a *reparametrization trick* on the latent variable is used (Kingma and Welling, 2013; Rezende, Mohamed, and Wierstra, 2014). The details of the trick on our models are presented in Chapter 3, specifically in Section 3.2.1.

2.2 State-space models

State-space models (SSMs) are a general and flexible methodology for sequential data modelling. In a SSM, we are given a sequence of T observations $x_{1:T} = [x_1, ..., x_T]$, that possibly depend on some inputs $u_{1:T} = [u_1, ..., u_T]$, and we are interested in modelling the distribution $p(x_{1:T}|u_{1:T})$ (Fraccaro, 2018). Notice that, by using $u_t = x_{t-1}$, we can resemble autoregressive models, i.e. a framework for predicting the next sequence entry based on the previous one. In SSMs, we introduce at each time step a *state* variable z_t that summarizes all the information coming from the past and determines the present and future evolution of the system. SSMs can then be seen as a temporal extension of the latent variable models discussed in Section 2.1.1, where the term *"state"* here exactly acts as the *latent variable* in latent variable models. The distinctive feature which also becomes the main assumption of SSMs is the prior over the latent variables z_t at each time step varies over time as it depends on the previous state z_{t-1} and possibly some inputs u_t to the model. In other words, the sequence of latent variables $z = [z_1, ..., z_T]$ in SSMs constitutes a first order Markov chain, whose graphical representation (Bayesian network) is given in Figure 2.1. Based on this assumption, we can prove the following properties of SSMs.

Proposition 2.2.1. *In a SSM, these properties hold*

- $p(x_t|z_{1:t}, x_{1:t-1}, u_{1:t}) = p(x_t|z_t)$
- $p(z_t|z_{1:t-1}, x_{1:t-1}, u_{1:t}) = p(z_t|z_{t-1}, u_t)$

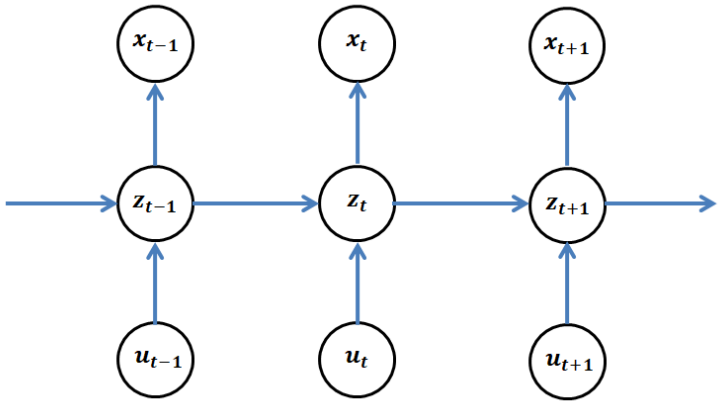

FIGURE 2.1: Graphical representation of a SSM

Proof.

- From Figure 2.1, we have $x_t \perp z_{1:t-1}, x_{1:t-1}, u_{1:t} | z_t$, i.e. x_t is conditionally independent with $z_{1:t-1}, x_{1:t-1}, u_{1:t}$ given z_t. In fact, given z_t, x_t is conditionally independent with all other variables in the model. Hence we have the first property.
- Again, from Figure 2.1, we can see that z_t only depends on z_{t-1} and u_t. Hence $z_t \perp z_{1:t-2}, x_{1:t-1}, u_{1:t-1} | z_{t-1}, u_t$, and the second property follows.

□

Using these properties, we can write the factorization of the joint distribution of a SSM as follows.

$$\begin{aligned} p(x_{1:T}, z_{1:T} | u_{1:T}) &= p(x_{1:T} | z_{1:T}) p(z_{1:T} | u_{1:T}) \\ &= \prod_{t=1}^{T} p(x_t | z_t) \cdot p(z_1) \prod_{t=2}^{T} p(z_t | z_{t-1}, u_t) \end{aligned} \tag{2.10}$$

The *emission distribution* $p(x_t|z_t)$ specifies how observation x_t depends on the latent state z_t. $p(z_t|z_{t-1}, u_t)$ is called the *transition distribution*, and represents the prior distribution for the state at each time step given the previous state and the current input to the model. This distribution fully determines the temporal evolution of the system. Then, the marginal distribution over the observations can be obtained by integrating out the states, i.e.

$$p(x_{1:T} | u_{1:T}) = \int p(x_{1:T}, z_{1:T} | u_{1:T}) dz_{1:T} \tag{2.11}$$

Here we have assumed that the $z_{1:T}$ are continuous variables, but the same ideas apply to the discrete version by replacing the integral with a summation. For the rest of this section, we present the counterpart of VAEs in SSMs, i.e. deep state-space models. Afterwards, we discuss how to deal with posterior inference in such models.

2.2.1 Deep state-space models

Inspired by VAEs, deep state-space models (DSSMs) are a class of SSMs which utilize neural networks to define probability distributions involved in their model. Here, there are three of them, namely transition distribution, emission distribution, and posterior of transition distribution. In DSSMs, the transition distribution can be Gaussian with diagonal structured covariance matrix, i.e. $p_\gamma(z_t|z_{t-1}, u_t) = \mathcal{N}(z_t; {}_\gamma\mu_t, \text{diag } {}_\gamma\sigma_t^2)$, where ${}_\gamma\mu_t$ and ${}_\gamma\sigma_t^2$ are a function (neural networks with parameters γ) of the previous latent state z_{t-1} and current input u_t. As for VAEs, depending on the type of observation, the emission distribution $p_\theta(x_t|z_t)$ is typically chosen to be either a Gaussian distribution (real-valued data) or a Bernoulli distribution (binary data). The parameters of both distributions are computed by neural networks with input z_t.

Because now we have explicitly mentioned the parameterization of the distributions, it might be better to rewrite the joint distribution of DSSMs as follows (notice the subscript of each probability function term, which represents the set of parameters of corresponding neural networks)

$$\begin{aligned} p_{\theta,\gamma}(x_{1:T}, z_{1:T}|u_{1:T}) &= p_\theta(x_{1:T}|z_{1:T})p_\gamma(z_{1:T}|u_{1:T}) \\ &= \prod_{t=1}^{T} p_\theta(x_t|z_t)p_\gamma(z_t|z_{t-1}, u_t) \end{aligned} \tag{2.12}$$

The above distribution is also called the generative model of SSMs. As the name explains, it specifies the generative process of the data.

2.2.2 Posterior inference on deep state-space models

When using SSMs to model a sequential data, generally there are two ways to perform posterior inference. Let us consider a simplified speech recognition example of trying to understand what a friend is saying in a very noisy bar. The observation x_t represents the noisy speech waveform at each time step, while z_t is the discrete variable that represents the corresponding word pronounced by the friend. At any point in time, we want of course to infer the word that the friend is saying, i.e. compute the posterior distribution $p(z_t|x_t)$. As the bar is noisy however, we may not be sure of which word was pronounced. At time t we also know what the friend said in $x_{1:t-1}$, and we can therefore condition even on the past observations, i.e. compute $p(z_t|x_{1:t})$ instead. The knowledge about what the friend was talking about at previous time steps can provide some context and help us better infer z_t. We call this task **filtering**, as it reduces the noise compared to only using the present observation x_t during inference. Despite this, due to the noise in the bar we may still be unsure on the inferred word. In this case, we can hope that while we keep listening to the friend, there will be one clue that will clarify the inferred state z_t. In this case we are therefore also using knowledge on the future during inference, i.e. we are considering the **smoothed** posterior $p(z_t|x_{1:T})$. Note that, filtering can be implemented in online settings. Whilst smoothing cannot, since it is conditioned not only on the past and present information, but also on the future one (Fraccaro, 2018).

Before we discuss further about each style of posterior inference above, observe that the true posterior, the one derived from Bayes rule below

$$p(z_{1:T}|x_{1:T}, u_{1:T}) = \frac{p(x_{1:T}|z_{1:T})p(z_{1:T}|u_{1:T})}{p(x_{1:T}|u_{1:T})} \tag{2.13}$$

is typically intractable, due to a similar reason as for VAEs (notice the denominator in (2.13) is even more complex than what in VAEs). See Example 2.2.2 below for a demonstration

Example 2.2.2
Suppose we consider a simple deep state-space model with $T = 2$ only. We then have the expression for the marginal distribution as follows.

$$\begin{aligned}
p(x_{1:2}|u_{1:2}) &= \int p(x_{1:2}, z_{1:2}|u_{1:2})dz_{1:2} \\
&= \int_{z_2}\int_{z_1} p(x_1, z_1, x_2, z_2|u_1, u_2)dz_1dz_2 \\
&= \int_{z_2}\int_{z_1} p(x_2, z_2|x_1, z_1, u_2)p(x_1, z_1|z_0, u_1)dz_1dz_2 \\
&= \int_{z_2}\int_{z_1} p(x_2, z_2|z_1, u_2)p(x_1, z_1|z_0, u_1)dz_1dz_2 \\
&= \int_{z_2}\int_{z_1} p(x_2|z_2)p(z_2|z_1, u_2)p(x_1|z_1)p(z_1|z_0, u_1)dz_1dz_2
\end{aligned} \tag{2.14}$$

Let us assume that we model any z_t and x_t to be multivariate Gaussian. Then, each term in (2.14) has the form as the following.

$$(2\pi)^{-\frac{k^{(\cdot)}}{2}} \det(\Sigma^{(\cdot)})^{-\frac{1}{2}} e^{-\frac{1}{2}(\cdot-\mu^{(\cdot)})'\text{inv}(\Sigma^{(\cdot)})(\cdot-\mu^{(\cdot)})} \tag{2.15}$$

where k, μ, and Σ are respectively the dimensionality, mean vector, and covariance matrix parameters, and a superscript $(\cdot)$ on any parameter denotes the corresponding random variable it represents. We see that the resulting integrand is complex, especially when the dimensionality of each variable is high.

Through this example, we have demonstrated that the integral in (2.14) is hardly tractable, so is the posterior in (2.13).

Therefore, we resort the variational principle method to approximate the posterior by introducing a distribution over the latent variables $q(z_t|\cdot)$. Although originally not advocated by the method of variational principle to make the distribution q becomes conditional on any variable, we might want to condition it to resemble the true posterior distribution structure. This is in order to have a better approximation, just like we do on VAEs. With respect to this, the simplified true posterior distribution structure for smoothing type inference is given in the following proposition.

Proposition 2.2.2. *In a SSM with fixed initial state z_0, the true posterior distribution factorizes as*

$$p(z_{1:T}|x_{1:T}, u_{1:T}, z_0) = \prod_{t=1}^{T} p(z_t|z_{t-1}, x_{t:T}, u_{t:T}) \tag{2.16}$$

Proof. From the graphical representation in Figure 2.1, we know the left hand side in (2.16) originally factorizes as

$$p(z_{1:T}|x_{1:T}, u_{1:T}, z_0) = \prod_{t=1}^{T} p(z_t|z_{0:t-1}, x_{1:T}, u_{1:T}) \tag{2.17}$$

Note that we can then write

$$\prod_{t=1}^{T}(z_t|z_{0:t-1}, x_{1:T}, u_{1:T}) = \prod_{t=1}^{T} p(z_t|z_{0:t-1}, x_{1:t-1}, u_{1:t}, x_{t:T}, u_{t+1:T}) \tag{2.18}$$

$$= \prod_{t=1}^{T} p(z_t|z_{t-1}, u_t, x_{t:T}, u_{t+1:T}) \tag{2.19}$$

$$= \prod_{t=1}^{T} p(z_t|z_{t-1}, x_{t:T}, u_{t:T}) \tag{2.20}$$

where we have used the second property in Proposition 2.2.1 in (2.19). □

This structure of true posterior is particularly useful when the offline setting is possible, namely the data is available/received in a complete sequence basis (instead of per entry basis). In such cases, one is encouraged to go with smoothing type of inference by making the variational distribution to mimic the structure of the true posterior, i.e.

$$q(z_{1:T}|x_{1:T}, u_{1:T}, z_0) = \prod_{t=1}^{T} q(z_t|z_{t-1}, x_{t:T}, u_{t:T}) \tag{2.21}$$

However, when the online learning environment is inevitable, e.g. on multi-agent systems setting, such an approximation structure as in (2.21) is not possible. Because at each time t in an online setting, we do not have the access to any variable with index later than t. Therefore, in this type of setting, we must resort to filtering type of inference and use instead the following simpler structure for variational distribution.

$$q(z_{1:T}|x_{1:T}, u_{1:T}, z_0) = \prod_{t=1}^{T} q(z_t|z_{t-1}, x_t, u_t) \tag{2.22}$$

To conclude this section, note that in DSSMs, the posterior distribution is also parameterized by neural networks. It means that in filtering type of inference, we can write each factor of distribution (2.22) as $q_\phi(z_t|z_{t-1}, x_t, u_t)$, where ϕ is the parameter set of the neural networks which take inputs z_{t-1}, x_t, and u_t at each time step t.

2.3 Multi-agent systems

Multi-agent systems (MAS) are systems composed of multiple interacting computing elements, known as agents (Woolridge, 2002). An agent itself is an entity which is placed in an environment and senses different parameters that are used to make decisions to accomplish a goal of the entity. Agents are capable of interacting with the environment and/or with other agents to learn new contexts and actions. Subsequently, agents use their knowledge to decide and perform an action on the environment to solve their allocated task (Dorri, Kanhere, and Jurda, 2018). In MAS, the environment is typically dynamic. That is, it is affected/changed by the actions

executed by all involved agents in the system. This results in that, each agent only has partial control over the environment changes. The recurrent diagram of a typical MAS is depicted in Figure 2.2. As the diagram shows, each of the agents performs an action to take based on their current observation of the environment, which together recurrently affect the environment.

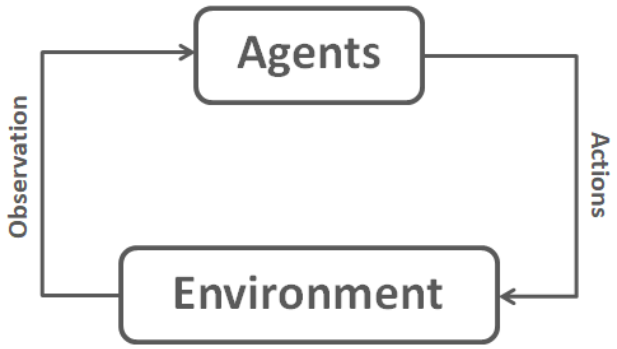

FIGURE 2.2: Agents-environment interaction in MAS

In MAS, the complete situation of the environment in the trajectory is called the *environment state*. Ideally, the agents choose their best actions according to this information sequentially. Unfortunately, it is often the case that the agents face the problem of *partial observability*, i.e. they only observe some facets of the environment, instead of the complete figure of it. In such cases, the agents need to infer the environment states based their partial observations, and later choose actions based on the inferred version of the environment states.

MAS can be applied to both cooperative and competitive settings. In a cooperative MAS, the agents share a common goal ad need to collaborate in order to accomplish the goal. Whilst, in the competitive setting, agents follow their individual interests/goals.

Below, we present how to look at MAS in a formal way, i.e. MAS abstraction. The section is then concluded by mentioning some challenges that exist in the field of MAS, as well as the benefits of modelling the opponents' policy.

2.3.1 Multi-agent systems abstraction

The abstraction of MAS in this section follows what presented in (Woolridge, 2002). Suppose there are n agents involved in our MAS. Let us denote the finite set of actions available (action space) to the agent $i \in \{1, 2, ..., n\}$ as $\mathcal{U}^i$. Next, let u_t^i denotes the action taken by agent i in time step t (note that $u_t^i \in \mathcal{U}^i$). Let c_t denotes the joint actions taken by all agents in time step t, i.e. $c_t \in \mathcal{U}_t^1 \times ... \times \mathcal{U}_t^n$. Moreover, let us denote the environment state in time step t as s_t. With all the above assumptions, the running of our MAS through time steps results in a trajectory τ as follows.

$$\tau : s_0 \xrightarrow{c_1} s_1 \xrightarrow{c_2} s_2 \xrightarrow{c_3} ... \xrightarrow{c_T} s_T \tag{2.23}$$

where s_0 is the initial state of the environment, and T is the maximum time index we consider.

Let us denote the set of possible environment states as S, along with

- Γ be the set of all possible finite sequence like (2.23);
- Γ^c be the subset of these that end with an action; and
- Γ^s be the subset of these that end with an environment state.

In order to represent the effect of the agents' actions on the environment state, we introduce a state transition function

$$\lambda : \Gamma^c \to \mathcal{S} \tag{2.24}$$

Thus, a state transition function maps a trajectory (assumed to end with the action of agents) to a set of possible environment states (state space). To wrap up, we say an environment Env is a triple $Env = \langle \mathcal{S}, s_0, \lambda \rangle$.

In addition, we model each agent's policy as a function which maps trajectory (assumed to end with an environment state) to an action. Thus for $i \in \{1, 2, ..., n\}$,

$$\pi : \Gamma^c \to \mathcal{U}^i \tag{2.25}$$

2.3.2 Challenges in multi-agent systems

In MAS, the dynamics of the environment is governed by the joint policies of all agents in the system. In addition, each agent executes actions based on observations it perceives from the environment. In other words, each agent through its policy only has partial control over the environment change. Subsequently, both environment dynamics and agents' joint behavior drive the interaction between the agents. This implies that any specific agent faces a task in which the optimal policy is the best response to opponents' policy, which is not directly observed and must therefore be inferred from observations. Among other things, this circumstance implies problems in the coordination among agents and the learning process (Dorri, Kanhere, and Jurda, 2018). For the first, coordination control is essential in a collaborative setting of MAS, to ensure that agents collaboratively reach their common goals. There are at least two sub-challenges entailing this aspect, namely controllability and synchronization. Controllability refers to the situation when MAS can be steered from an initial state to a specific state using certain regulations, this is a staple implementation in robotics. However, in non-deterministic environments the results of actions are not predictable, thus agents should decide on new actions after observing the result of their previous actions which incurs delay and limits the proactivity of agents. Whilst synchronization means the action performed by the agent is aligned in time with other agents' actions. This aspect focuses on ensuring correlated-in-time behavior among different agents policy to be harmonized together in order to achieve the common goal. On the other hand, with respect to the learning aspect, agents in MAS can leverage machine learning algorithms to discover and forecast the environment anticipated change and adapt to unforeseen situations. Again, the dynamicity of the environment in MAS increases the complexity of the learning process.

In light of the above discussed challenges, in this work we motivate the benefits of modelling other agents' policy in MAS. In general, by knowing others' policy, agents would better design their own policy through choosing actions which are best responses to those opponents' anticipated moves. We believe such modelling will bring benefits for the agent in both collaborative and competitive settings (Sen

and Weiss, 2001). In a collaborative MAS, it will enhance collaboration since it targets controllability and synchronization. By knowing other agents' next actions, the modelling agent can subsequently align better in the systems and eventually leads to a better performance of all agents achieving the shared goal. While in the opposite setting, such anticipated information can be used to choose a set of actions or strategy that outplays the opponents.

Chapter 3

Deep State-Space Models in Multi-Agent Systems

This chapter is the most important part of this thesis. Here, the proposed models are formulated and discussed in details. We start from the underlying assumptions of the models, which are followed by the models construction. Afterward, we provide guidelines on how to use the models. Then, this chapter is concluded by a discussion on the proposed models.

3.1 Problem scenario and assumptions

We consider multi-agent systems (MAS) that consist of a controllable agent and some other agents (can be either collaborators or opponents, or both) which interact with each other (see Section 2.3). The general assumptions on the systems are as follows:

- Observation in each time step, denoted by o_t is a noisy non-linear mapping from a true environment state $\tilde{s}_t$, which is unobserved (latent).
- The true environment state $\tilde{s}_t$ evolves over time and gets affected also by both controllable agent's action u_t, and other agents' actions a_t.
- Other agents' actions a_t also come from a noisy non-linear mapping of another true latent state representation $\tilde{z}_t$
- The true latent state representation $\tilde{z}_t$ also evolves over time and gets affected by the environment state s_t, which can be seen as the *summary* of all current observation and previous actions (both controllable agent's and other agents').

Note that the above assumptions are very general and suit with the natural characteristics of MAS. For each time t, $\tilde{s}_t$ may be interpreted as the ground truth *environment state* at time t. Likewise, we can think of $\tilde{z}_t$ as the other agents' *mental state* ground truth at time t. In the following sections, we propose a formulation of deep state-space models (DSSMs) over the considered systems. There are two main goals of this formulation. First is to model the environment dynamics. That is, the change on the environment as a result of all actions taken by the involved agents in the systems. Secondly, to predict the other agents' actions to be taken utilizing some results from the first modelling phase. Concluding this section, it is emphasized once again that we use tilde signs on environment and other agents' mental states to denote the ground truth values of these variables, which we assume are never observed. To already mention here, inference on those variables, i.e. finding their distributions and later using it to sample values s_t and z_t (notice the expressions' lack of tilde signs), are amongst the most essential part of the models constructed in the following sections.

3.2 The models

The key idea of the models is to perform two phases of DSSMs (see Section 2.2.1), each for modelling one of the two goals mentioned in the previous section. There are three sequences available (observed) in this problem: the observations $o_{1:T} = [o_1, ..., o_T]$, our controllable agent's actions $u_{1:T} = [u_1, ..., u_T]$ and other agents' actions $a_{1:T} = [a_1, ..., a_T]$ (recall that there can be many other agents, in such cases a_t would be multidimensional). Note that any action with time index $t > 1$ represents the action taken based on observation at time index $t - 1$. With this respect, let us assume that the agents execute their actions u_t and a_t immediately after the emergence of observation o_{t-1}. By denoting the concatenation of u_t and a_t as c_t, i.e. $c_t := [u_t, a_t]$, we can write all the three sequences as a single trajectory $\tau = (c_1, o_1, ..., o_{T-1}, c_T, o_T)$.

Phase 1 of the models is devoted to model the sequence observations, conditioned on all actions in place (our controllable agent's and the other agents'). The main goal is to perform inference on the sequence of environment states $s_{1:T} = [s_1, ..., s_T]$ from the observations $o_{1:T}$, which is also affected by both agents' actions $c_{1:T}$. As environment states, we can think of $s_{1:T}$ as a sequence of summaries that contains all the information of the system in a more compact way. After we have $s_{1:T}$ at hand, we go to Phase 2 of the models, namely to model the sequence of other agents' actions $a_{1:T}$, conditioned by the system summaries $s_{1:T}$.

3.2.1 Phase 1

Model construction

In this phase, the goal is originally to fit a generative model for the sequence of observations conditioned on the sequence of all actions in place, i.e. $p(o_{1:T}|c_{1:T})$. As for the discussion in Section 2.2, we will achieve the goal by incorporating self-introduced latent variables (here they are known as the environment states) $s_{1:T}$. Similar to (2.12), the joint distribution (or equivalently the generative model) in this phase is thus

$$\begin{aligned} p_{\theta,\gamma}(o_{1:T}, s_{1:T}|c_{1:T}) &= p_\theta(o_{1:T}|s_{1:T}) p_\gamma(s_{1:T}|c_{1:T}) \\ &= \prod_{t=1}^{T} p_\theta(o_t|s_t) p_\gamma(s_t|s_{t-1}, c_t) \end{aligned} \tag{3.1}$$

Along with this, we subsequently introduce a variational distribution $q_\phi(s_{1:T}|o_{1:T}, c_{1:T})$ to approximate the true (but intractable) posterior $p_\gamma(s_{1:T}|o_{1:T}, c_{1:T})$. Therefore, we can write the log of marginal distribution $p(o_{1:T}|c_{1:T})$ as follows.

$$\begin{aligned}
\log p(o_{1:T}|c_{1:T}) &= \log \int p(o_{1:T}, s_{1:T}|c_{1:T}) ds_{1:T} \\
&= \log \int \frac{p(o_{1:T}, s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} q_\phi(s_{1:T}|o_{1:T}, c_{1:T}) ds_{1:T} \\
&\geq \int q_\phi(s_{1:T}|o_{1:T}, c_{1:T}) \log \frac{p(o_{1:T}, s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} ds_{1:T} \qquad (3.2) \\
&= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} \left[\log \frac{p(o_{1:T}, s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} \right] \qquad (3.3)
\end{aligned}$$

where the inequality in (3.2) is due to Jensen inequality (note that log is a concav function). The expression in (3.3) is the ELBO in Phase 1.

Now we have to define the exact parameterization of the factor of the variational distribution $q_\phi(s_{1:T}|o_{1:T}, c_{1:T})$ for each time step t. Since the systems we consider in this problem (i.e. MAS) assert an online setting, we perform filtering type of inference on those approximate posterior factors. As dictated in (2.22), we can factorize the variational distribution as follows.

$$q_\phi(s_{1:T}|o_{1:T}, c_{1:T}) = \prod_{t=1}^{T} q_\phi(s_t|s_{t-1}, c_t, o_t) \qquad (3.4)$$

Observe that, both the joint distribution in (3.1) and the posterior approximation in (3.4) factorize over time steps, we therefore can rewrite the ELBO in (3.3) as a sum over T terms as follows.

$$\begin{aligned}
\text{ELBO} &= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} \left[\log \frac{p(o_{1:T}, s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} \right] \\
&= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T}, c_{1:T})} \left[\sum_{t=1}^{T} \log \frac{p_\theta(o_t|s_t) p_\gamma(s_t|s_{t-1}, c_t)}{q_\phi(s_t|s_{t-1}, c_t, o_t)} \right] \\
&= \sum_{t=1}^{T} \mathbb{E}_{q_\phi^*(s_{t-1})} [\mathbb{E}_{q_\phi(s_t|s_{t-1}, c_t, o_t)} [\log p_\theta(o_t|s_t) \\
&\qquad + \log p_\gamma(s_t|s_{t-1}, c_t) - \log q_\phi(s_t|s_{t-1}, c_t, o_t)]] \qquad (3.5)
\end{aligned}$$

where $q_\phi^*(s_{t-1})$ denotes the marginal distribution of s_{t-1} in the variational approximation to the posterior $q_\phi(s_{1:t-1}|o_{1:t}, c_{1:t})$ given by

$$\begin{aligned}
q_\phi^*(s_{t-1}) &= \int q_\phi(s_{1:t-1}|o_{1:t}, c_{1:t}) ds_{1:t-2} \\
&= \mathbb{E}_{q_\phi^*(s_{t-2})} [q_\phi(s_{t-1}|s_{t-2}, c_{t-1}, o_{t-1})] \qquad (3.6)
\end{aligned}$$

Therefore, we can interpret the ELBO in (3.5) as having a VAE at each time step with a time-varying prior that depends on the previous state.

In this thesis, we assume all unobserved latent variables are multivariate Gaussian distributions, while the distribution of the observation depends on the data specification. It can be modelled as Gaussian (real-valued), binary, or categorical

distribution. Concluding this construction, an example of graphical form of the generative model in this phase is given in the following.

$$s_0 \sim p_0(s_0) = \mathcal{N}(s_0; 0, I) \tag{3.7}$$
$$s_t \sim p_\gamma(s_t|s_{t-1}, c_{t-1}) = \mathcal{N}(s_t; {}_\gamma\mu_t, \text{diag}({}_\gamma\sigma_t^2)) \tag{3.8}$$
$$o_t \sim p_\theta(o_t|s_t) = \mathcal{N}(o_t; {}_\theta\mu_t, \text{diag}({}_\theta\sigma_t^2)) \tag{3.9}$$

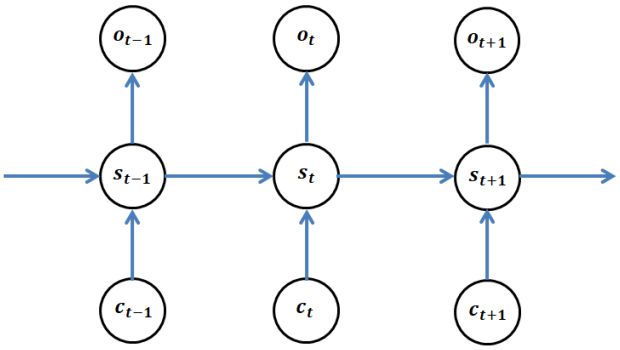

FIGURE 3.1: Graphical representation of the generative model in Phase 1

We parameterize each distribution p_θ, p_γ and q_ϕ by neural networks, such that θ, γ and ϕ are the parameters of neural networks p_θ, p_γ and q_ϕ respectively. These neural networks are called *emission net*, *transition net*, and *inference net*, respectively. Note that, with parameterization by neural networks we mean that the neural networks emit the parameters of the distribution, e.g. the mean and variance values (hence the output layer consists of two blocks in this case) for Gaussian distribution.

Model training

As for VAEs, the goal of our model in Phase 1 is to maximize the ELBO in (3.5). Before we proceed, note that the expression in (3.5) represents the ELBO over a single complete trajectory (the ELBO is sequence based). Since we parameterize all the distributions using neural networks, ELBO maximization is now carried out through jointly tune all the neural networks parameters θ, γ, and ϕ. We train all the neural networks using backpropagation algorithm (Rumelhart, Hinton, and Williams, 1986). As the algorithm dictates, the neural networks' parameters are adjusted according to the gradients of the objective function (here is the ELBO in (3.5)) with respect to the corresponding neural weights. Therefore, in our case, we have to compute the gradients with respect to each parameter sets θ, γ, and ϕ. Regarding this task, for the emission and transition net we have

$$\begin{aligned}\nabla_\theta \text{ELBO} &= \nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{q_\phi^*(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)}[\log p_\theta(o_t|s_t) \\ &\qquad + \log p_\gamma(s_t|s_{t-1}, c_t) - \log q_\phi(s_t|s_{t-1}, c_t, o_t)]] \\ &= \sum_{t=1}^{T} \mathbb{E}_{q_\phi^*(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)} \nabla_\theta \log p_\theta(o_t|s_t)]\end{aligned} \tag{3.10}$$

$$\begin{aligned}\nabla_\gamma \text{ELBO} &= \nabla_\gamma \sum_{t=1}^{T} \mathbb{E}_{q^*_\phi(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)}[\log p_\theta(o_t|s_t) \\ &\qquad + \log p_\gamma(s_t|s_{t-1},c_t) - \log q_\phi(s_t|s_{t-1},c_t,o_t)]] \\ &= \sum_{t=1}^{T} \mathbb{E}_{q^*_\phi(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)} \nabla_\gamma \log p_\gamma(s_t|s_{t-1},c_t)] \end{aligned} \tag{3.11}$$

By moving the gradient operator inside the expectation as we do in the above, later in the training section we can form Monte Carlo estimators to approximate the corresponding ELBO gradients, provided a sample of latent variable s_t for each time step t.

However, for gradients with respect to the inference net's parameters ϕ, the circumstance is a bit different.

$$\begin{aligned}\nabla_\phi \text{ELBO} = \nabla_\phi \sum_{t=1}^{T} \mathbb{E}_{q^*_\phi(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)}[\log p_\theta(o_t|s_t) \\ + \log p_\gamma(s_t|s_{t-1},c_t) - \log q_\phi(s_t|s_{t-1},c_t,o_t)]] \end{aligned} \tag{3.12}$$

Ideally, for the same reason as the above (forming Monte Carlo estimators), we want to have a similar situation as in the previous cases, i.e. moving the gradient operator inside the expectation and make it only affects the term of $-\log q_\phi(s_t|s_{t-1},c_t,o_t)$. Yet, we cannot do this since the distribution with respect to which the expectation is taken in (3.12) depends on the parameters ϕ. In other words, the sample s_t needed to form the Monte Carlo estimators depends on ϕ directly.

Following (Kingma and Welling, 2013), this issue is resolved by expressing the latent-random variable $s_t \sim q_\phi(s_t|s_{t-1},c_t,o_t)$ as a differentiable and invertible transformation of another random variable (called noise) ϵ_t, given the outputs of the inference net at time t: ${}_\phi\mu_t, {}_\phi\sigma^2{}_t$, denoted $g(\cdot)$ as follows.

$$s_t = g(\epsilon_t, {}_\phi\mu_t, {}_\phi\sigma^2{}_t) \tag{3.13}$$

Given such a parameterization, expectations in ELBO expression can be rewritten with respect to the noise distribution $p(\epsilon_t)$. In other words, the sample s_t does not (directly) depend on ϕ anymore. Hence, we can move the gradient operator inside the expecation, i.e.

$$\begin{aligned}\nabla_\phi \text{ELBO} &= \nabla_\phi \sum_{t=1}^{T} \mathbb{E}_{p(\epsilon_{t-1})}[\mathbb{E}_{p(\epsilon_t)}[-\log q_\phi(s_t)]] \\ &= \sum_{t=1}^{T} \mathbb{E}_{p(\epsilon_{t-1})}[\mathbb{E}_{p(\epsilon_t)}[\nabla_\phi - \log q_\phi(s_t)]] \end{aligned} \tag{3.14}$$

where s_t in (3.14) should really be viewed as the output of function $g(\epsilon_t, {}_\phi\mu_t, {}_\phi\sigma^2{}_t)$.

In training section, we estimate the ELBO by its simple Monte Carlo estimators. That is, the one that only based on a single s_t sample (drawn from reparameterization function $g(\epsilon_t, {}_\phi\mu_t, {}_\phi{\sigma^2}_t)$) at each time step t and compute the gradients accordingly. Thus we have

$$\begin{aligned}\text{ELBO} &\simeq \sum_{t=1}^{T}(\log p_\theta(o_t|s_t) + \log p_\gamma(s_t|s_{t-1}, c_t) - \log q_\phi(s_t|s_{t-1}, c_t, o_t)) \\ &= \tilde{\text{ELBO}} \end{aligned} \tag{3.15}$$

Proposition 3.2.1. *By carrying out reparameterization trick with noise distribution $p(\epsilon_t)$, gradients with respect to all parameters θ, γ, and ϕ taken from ELBO in (3.15) are unbiased estimators of the exact ELBO gradients.*

Proof.

$$\begin{aligned}\mathbb{E}_{p(\epsilon_t)}[\nabla_{\theta,\gamma,\phi}\tilde{\text{ELBO}}] &= \mathbb{E}_{p(\epsilon_t)}[\nabla_{\theta,\gamma,\phi}\sum_{t=1}^{T}(\log p_\theta(o_t|s_t) + \log p_\gamma(s_t|s_{t-1}, c_t) \\ &\qquad - \log q_\phi(s_t|s_{t-1}, c_t, o_t))] \\ &= \nabla_{\theta,\gamma,\phi}\mathbb{E}_{p(\epsilon_t)}[\sum_{t=1}^{T}(\log p_\theta(o_t|s_t) + \log p_\gamma(s_t|s_{t-1}, c_t) \\ &\qquad - \log q_\phi(s_t|s_{t-1}, c_t, o_t))] \\ &= \nabla_{\theta,\gamma,\phi}[\sum_{t=1}^{T}\mathbb{E}_{q^*_\phi(s_{t-1})}[\mathbb{E}_{q_\phi(s_t|s_{t-1},c_t,o_t)}[\log p_\theta(o_t|s_t) \\ &\qquad + \log p_\gamma(s_t|s_{t-1}, c_t) - \log q_\phi(s_t|s_{t-1}, c_t, o_t)]]] \end{aligned} \tag{3.16}$$

□

In this thesis, whenever the latent variables are modelled as Gaussian distributions with diagonal structured covariance matrix, $p(\epsilon_t)$ in parameterization trick is set to be standard Gaussian $\mathcal{N}(0, I)$, and

$$\begin{aligned} s_t &= g(\epsilon_t, {}_\phi\mu_t, {}_\phi{\sigma^2}_t) \\ &= {}_\phi\mu_t + {}_\phi\sigma_t \odot \epsilon_t \end{aligned} \tag{3.17}$$

where ${}_\phi\mu_t$ and ${}_\phi\sigma_t$ are the mean and square root of variance of s_t, emitted by the inference net with input s_{t-1}, c_t and o_t. Moreover, $\odot$ denotes pointwise multiplication operator.

Finally, the training protocols of Phase 1 model are given in Algorithm 1 as follows.

Result: tuned neural networks parameters θ, γ, ϕ
Initialize $\theta, \gamma, \phi, s_0$;
while *SGD not converged* **do**
  Gather a (mini)batch of trajectories $\mathcal{M}$;
  for $t \in$ *trajectory length* **do**
    $\epsilon_t \sim p(\epsilon_t)$
    $s_t \leftarrow g(\epsilon_t, \phi, s_{t-1}, c_t, o_t)$
  end
  Compute (sequence based) ELB̃O and its gradients $\nabla_{\theta,\gamma,\phi}$ELB̃O over $\mathcal{M}$;
  Update θ, γ, ϕ using SGD optimizer
end

Algorithm 1: Training Phase 1 Model

As a final remark, note that if the transition model $p_\gamma(s_t|s_{t-1}, c_t)$ and inference model $q_\phi(s_t|s_{t-1}, c_t, o_t)$ are both modeled as Gaussian distributions (as is in our implementation), we can simplify the last two terms in (3.5) since an analytic expression of the KL divergence is available as the following.

$$KL(q_\phi||p_\gamma) = \frac{1}{2}\left(\text{tr}(\Sigma_\gamma^{-1}\Sigma_\phi) + (\mu_\gamma - \mu_\phi)^T\Sigma_\gamma^{-1}(\mu_\gamma - \mu_\phi) - k + \log\left(\frac{\det\Sigma_\gamma}{\det\Sigma_\phi}\right)\right) \tag{3.18}$$

where Σ_γ and Σ_ϕ are the covariance matrix of transition and inference model, respectively. Moreover, k is the dimension of latent variables s, tr denotes matrix trace operation, and T denotes transpose operator.

3.2.2 Phase 2

After we finish training all neural networks in Phase 1, we are ready to work out Phase 2. In this phase, the ingredients are the sequence of environment states $s_{1:T} = [s_1, ..., s_T]$ and other agents' actions $a_{1:T} = [a_1, ..., a_T]$. Note that we can gather the sequence $s_{1:T}$ by utilizing the inference net from Phase 1. Before we continue, let us reside for a moment and think about the sequence $s_{1:T}$. As depicted in Figure 3.1, note that we can interpret the environment state s_t as a summary of the system up to the current time step (including the current observation and actions occurred). Since we assume that the other agents might consider the current observation and possibly some memories about actions taken in the past, it makes sense to replace those things with the current summary s_t. That is, making the summary s_t to become their consideration for taking the next actions a_{t+1}. In Phase 2, we therefore want to model the sequence of other agents' actions $a_{1:T}$, conditioned on the sequence of system summary $s_{1:T}$. Similar to Phase 1, we approach this by auxiliary modelling self-introduced latent variables $z_{1:T}$ that might be interpreted as *the mental state* of the other agents. Again, following variational principle, we introduce a variational distribution $q_\phi(z_{1:T}|a_{1:T}, s_{1:T})$ to approximate the true (but intractable) posterior $p_\gamma(z_{1:T}|a_{1:T}, s_{1:T})$. By an analogous derivation as Phase 1, we then have

$$\begin{aligned}
\log p(a_{1:T}|s_{1:T}) &= \log \int p(a_{1:T}, z_{1:T}|s_{1:T}) dz_{1:T} \\
&= \log \int \frac{p(a_{1:T}, z_{1:T}|s_{1:T})}{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} q_\phi(z_{1:T}|a_{1:T}, s_{1:T}) dz_{1:T} \\
&\geq \int q_\phi(z_{1:T}|a_{1:T}, s_{1:T}) \log \frac{p(a_{1:T}, z_{1:T}|s_{1:T})}{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} dz_{1:T} \\
&= \mathbb{E}_{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} \left[\log \frac{p(a_{1:T}, z_{1:T}|s_{1:T})}{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} \right] \qquad (3.19)
\end{aligned}$$

The generative graphical model in this Phase 2 is the following:

$$z_0 \sim p_0(z_0) = \mathcal{N}(z_0; 0, I) \qquad (3.20)$$

$$z_t \sim p_\gamma(z_t|z_{t-1}, s_t) = \mathcal{N}(z_t; {}_\gamma\mu_t, \text{diag}({}_\gamma\sigma_t^2)) \qquad (3.21)$$

$$a_{t+1} \sim p_\theta(a_{t+1}|z_t) = \text{Categorical}(a_{t+1}; {}_\theta\eta_{t+1}) \qquad (3.22)$$

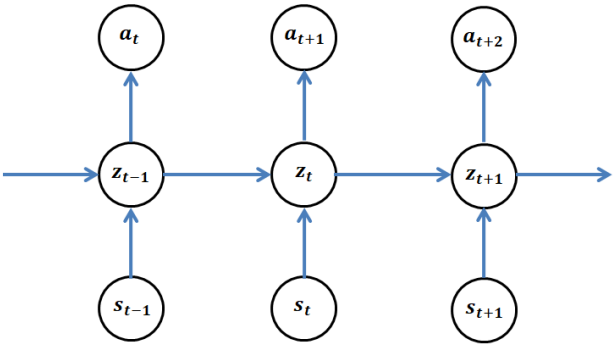

FIGURE 3.2: Graphical representation of the generative model in Phase 2

Note that in this phase we have one categorical distribution to model the other agents' actions (3.22). As in Phase 1, we parameterize each distribution p_θ, p_γ and q_ϕ by neural networks, such that θ, γ and ϕ are the parameters of neural networks p_θ, p_γ and q_ϕ respectively.

Training goes very similar as Phase 1, we want to maximize the following ELBO expression.

$$\begin{aligned}
\text{ELBO} &= \mathbb{E}_{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} \left[\log \frac{p(a_{1:T}, z_{1:T}|s_{1:T})}{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} \right] \\
&= \mathbb{E}_{q_\phi(z_{1:T}|a_{1:T}, s_{1:T})} \left[\sum_{t=1}^{T-1} \log \frac{p_\theta(a_{t+1}|z_t) p_\gamma(z_t|z_{t-1}, s_t)}{q_\phi(z_t|z_{t-1}, s_t, a_{t+1})} \right] \\
&= \sum_{t=1}^{T-1} \mathbb{E}_{q_\phi^*(z_{t-1})} [\mathbb{E}_{q_\phi(z_t|z_{t-1}, s_t, a_{t+1})} [\log p_\theta(a_{t+1}|z_t) \\
&\qquad + \log p_\gamma(z_t|z_{t-1}, s_t) - \log q_\phi(z_t|z_{t-1}, s_t, a_{t+1})]] \qquad (3.23)
\end{aligned}$$

we omit further details for brevity.

In so far as there are significant differences, these would be caused by neural networks architecture choices. Note that a_t is a discrete random variable, specifically it follows a categorical distribution. Therefore, the output layer of p_θ has to emit only one block of vector η_t which defines the probability mass of each possible output values of a_t.

3.3 Using the models

To begin this section, let us introduce a new index $h \in \{1, 2\}$ to denote which phase is being referred for a particular probability distribution (or equally neural networks) mentioned. Therefore, once we are done training the two phases above, we have six well tuned neural networks. From Phase 1 we have $p_\theta^{(1)}, p_\gamma^{(1)}$ and $q_\phi^{(1)}$, while from the second we have $p_\theta^{(2)}, p_\gamma^{(2)}$ and $q_\phi^{(2)}$. Moreover, let us distinguish the variable a_t (other agents's actions) based on their two possible sources, namely from observation (written as plain a_t) and from prediction (written as $\hat{a}_t$). Similar treatment applies for observation o_t. In the following, we discuss how to implement the models after their training phase is finished. In the following, recall that any neural net really emits the distribution parameters instead of a specific value point. Therefore, any specific value point of latent variable that used as part of the inputs of any emission net is obtained by sampling from the corresponding distribution, which is defined by those parameters.

3.3.1 Predicting the environment anticipated change via Phase 1

As already mentioned in the beginning of this chapter discussion, the goal of Phase 1 is to fit a generative model for the observation sequence. Using trained $p_\theta^{(1)}, p_\gamma^{(1)}$ and $q_\phi^{(1)}$, we can predict the environment anticipated change, i.e. the next observation o_{t+1}. To demonstrate, starting from $t = 1$, we pass $(s_0, c_1 = [u_1, a_1], o_1)$ to the inference net $q_\phi^{(1)}$, this provides us s_1. Then, pass this s_1, along with u_2 and a_2, to the transition net $p_\gamma^{(1)}$ to get s_2. Finally, we get the prediction of $\hat{o}_2$ by passing s_2 to the emission net $p_\theta^{(1)}$. Identical procedures apply for other time step t. The graphical illustration of the procedures is depicted in Figure 3.3.

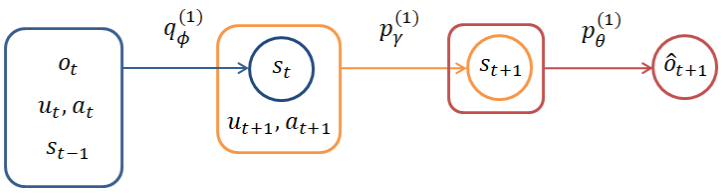

FIGURE 3.3: Using Phase 1 to predict the next observation

3.3.2 Predicting other agents' actions via Phase 1 and Phase 2

Standard implementation

This is a straightforward application of our established models: Phase 1 and Phase 2, to predict other agents' actions. We use the models as an *informant* who tells us (our controllable agent) the anticipated other agents' next actions $\hat{a}_{t+1}$, using current observation o_t and real observed actions $c_t = [u_t, a_t]$. Starting at $t = 1$, $q_\phi^{(1)}$ consumes $(o_1, c_1 = [u_1, a_1], s_0)$ to yield s_1. Then (s_1, z_0) is passed to $p_\gamma^{(2)}$ to get z_1. Finally, for this time step, $p_\theta^{(2)}$ will give us the prediction $\hat{a}_2$ based on z_1. In the beginning of $t = 2$, our agent has observed the **real** actions taken by the other agents a_2 (note there is no hat in the expression). At the end of $t = 2$, using $(o_2, c_2 = [u_2, a_2], s_1)$, $q_\phi^{(1)}$ will give us the value s_2. Together with z_1, it will provide us z_2 by passing them through $p_\gamma^{(2)}$. Next, $p_\theta^{(2)}$ will give us the prediction $\hat{a}_3$ based on z_2. The similar process goes on for the larger time step t. The illustration of the process is given in Figure 3.4

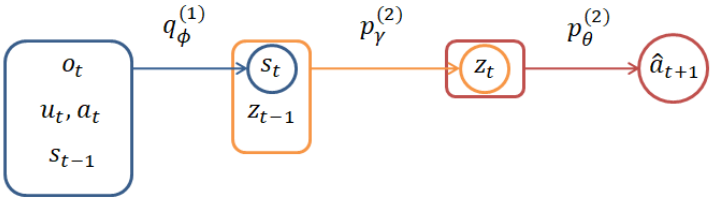

FIGURE 3.4: Standard implementation to predict other agents' actions

Pure imagination

Observe that, thanks to its generative nature, it is possible to utilize the models in a *fully imaginary* setting. By the last phrase we mean the models are only fed with the first observation o_1 and the controllable agent's action u_t at each time step. Our models still can perform the prediction of the sequence $\hat{a}_{2:T} = [\hat{a}_2, ..., \hat{a}_T]$. This is achieved by utilizing the transition model $p_\gamma^{(1)}$ that gives us the sequence $s_{1:T} = [s_1, ..., s_T]$, without the need to observe o_t other than the first one o_1. This can be done by passing the current **prediction** $\hat{a}_t$ (full imaginary), instead of the true observed a_t to $p_\gamma^{(1)}$. Such an implementation process is best described in a picture (see Figure 3.5). Note that, this style of implementation is particularly useful when the communication/interaction between agents in MAS is not reliable, or even worse: does not exist at all, in the test time.

3.4 Models discussion

In this chapter, we have proposed a new formulation of DSSMs in the field of MAS. The introduced models belong to the class of generative sequential models. Our complete models consist of two sub-models: Phase 1 and Phase 2. The ultimate use of the proposed models is to implement it as *an informant* who tells the controllable agent the anticipated action taken by the other agents at each time step in a MAS. Nevertheless, the model can be used as an alternative tool to model the environment

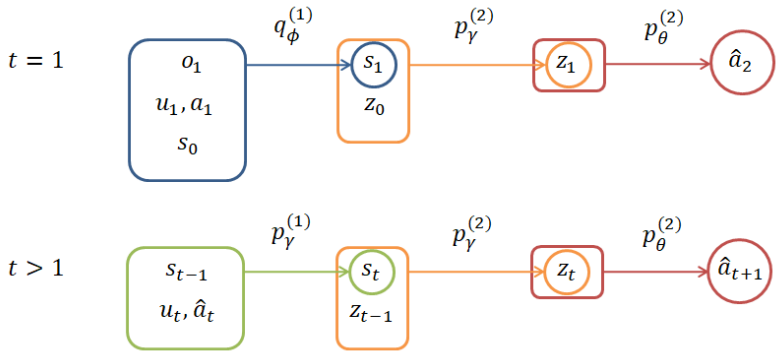

FIGURE 3.5: Pure imagination implementation to predict other agents' actions

dynamics. This is particularly true when one just merely use Phase 1 of the models.

Another models extension of application can be found in the field of reinforcement learning, particularly multi-agent reinforcement learning (MARL). It is almost immediate to use the model in MARL setting, since the model adjustment needed is minimal. That is, the only adjustment needs to be made is incorporating the reward values as one of the observed variables. This can be done by concatenating the reward with the environment observation at each time t. Using the model, the controllable agent in MARL can then design a better policy, which in turn leads to a higher total reward (or generally objective function) gained by the agent. The application of the model on this respect is presented in Chapter 6. Moreover, thanks to its generative nature, the model can be used to *imagine* any behavior of the other agents (not necessarily suggested by their–previously seen–original behavior) and later evaluate the effect it yielded to the system.

Nevertheless, there are still many ways to make the model presented in this chapter better. One of such alternatives will be discussed in the next Chapter 4. There, we induce covariance structures on the latent variable distribution, as opposed to the diagonal structured covariance matrix assumption in this chapter. The motivation of doing this takes root on tightening the ELBO, which in turn may lead to a better overall model obtained at hand.

Chapter 4

Models Refinement and Selection

To begin this chapter, recall that in Chapter 3 we set the distributions of latent variables such that there is no correlation across the dimensionality of those variables. As already mentioned in the last part of the previous chapter, we can extend the models to become richer and thus more flexible by inducing covariance structures on the distributions. This chapter presents the way to do the extension. The procedure leaves us with several models available, corresponding to the rank of the covariance inducing matrix. We then propose to choose the best-parsimonious model amongst them via the minimum description length (MDL) principle.

4.1 Towards a more flexible model

Given n data points, $x_1, ..., x_n$, a distribution over the whole data space, $\mathcal{X}_1 \times ... \times \mathcal{X}_n$, with independent assumption across the data dimensionality is less expressive compared to the one that allows non-zero correlations amongst them. As a result, if we fit the data that actually do have covariance structures across the dimensionality using the first distribution, we would end up with less goodness of fit compared if we use the latter distribution. With respect to our models, this is represented by the tightness of the ELBO approximation by the introduced variational distribution. This section is devoted to discuss this matter in details, as well as to present the way to refine the proposed models.

4.1.1 Tightness of the ELBO

Consider the models proposed in Chapter 3. Note that, in Phase 1 and Phase 2, we originally want to model $p(o_{1:T}|c_{1:T})$ and $p(a_{1:T}|s_{1:T})$, respectively. Using variational inference principle (see Section 2.1.2), we approximate these two models by *asking for help* from some self-introduced latent variables. This results in dependency of the obtained models with the quality of approximate posterior distributions $q_\phi^{(1)}$ and $q_\phi^{(2)}$. To better comprehend this relation, let us take a look on another way of deriving the ELBO for Phase 1 (notice the similarity of the derivation result with equation (2.6)). For any choice of posterior inference model $q_\phi(s_{1:T}|o_{1:T}, c_{1:T})$, including the choice of variational parameters ϕ, we have:

$$
\begin{aligned}
\log p(o_{1:T}|c_{1:T}) &= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\left[\log p(o_{1:T}|c_{1:T})\right] \\
&= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\left[\log \frac{p(o_{1:T},s_{1:T}|c_{1:T})}{p(s_{1:T}|o_{1:T},c_{1:T})}\right] \\
&= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\left[\log \frac{p(o_{1:T},s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\frac{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}{p(s_{1:T}|o_{1:T},c_{1:T})}\right] \\
&= \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\left[\log \frac{p(o_{1:T},s_{1:T}|c_{1:T})}{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\right] \\
&\qquad + \mathbb{E}_{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}\left[\log \frac{q_\phi(s_{1:T}|o_{1:T},c_{1:T})}{p(s_{1:T}|o_{1:T},c_{1:T})}\right] \\
&= ELBO_{q_\phi}(o_{1:T}|c_{1:T}) + KL(q_\phi(s_{1:T}|o_{1:T},c_{1:T})||p(s_{1:T}|o_{1:T},c_{1:T})) \quad (4.1)
\end{aligned}
$$

Note that, KL divergence is always non-negative. In particular, we have the KL term in (4.1) equals to zero if and only if $q_\phi(s_{1:T}|o_{1:T},c_{1:T})$ equals the true posterior $p(s_{1:T}|o_{1:T},c_{1:T})$. Therefore, the quality of our modelling approach (maximizing the ELBO instead of $\log p(o_{1:T}|c_{1:T})$) is determined by the closeness (in KL sense) between both distributions. Thus, ideally we want $q_\phi(s_{1:T}|o_{1:T},c_{1:T})$ to be expressive enough to approximate $p(s_{1:T}|o_{1:T},c_{1:T})$ in a tight fashion. However, in the initial work presented in Chapter 3, we model each factor of $q_\phi(s_{1:T}|o_{1:T},c_{1:T})$ as a multivariate Gaussian with diagonal structured covariance matrix, which is potentially not expressive enough (since omitting covariance structure between the dimension of latent variables) and thus could lead to a less tight ELBO. Therefore, by inducing covariance structures, it is hoped that the ELBO would be tighter approximating the log marginal probability $\log p(o_{1:T}|c_{1:T})$, and this in turn could lead to a better overall model obtained at hand.

4.1.2 Model with covariance structure

We can induce covariance structure across the dimensionality of the latent variables by modeling the covariance matrix as a low rank plus diagonal matrix (Miller, Foti, and Adams, 2017). It is particularly the low rank matrix, with rank $= r$, which induces the covariance structure. The reason why low rank matrix (instead of always going with full rank) structure could possibly be used to approximate the true covariance is, it might be unnecessary to model it precisely using a full rank matrix anyway. This is especially the case when some of the dimensions of the latent variable are close to independent. Suppose we denote the dimension of the latent variable as D. Using this method, we can represent the covariance matrix Σ as

$$\Sigma = FF^T + \text{diag}(\sigma^2) \quad (4.2)$$

where $F \in \mathbb{R}^{D\times r}$ is a low rank matrix inducing covariance structure, and $\sigma^2 \in \mathbb{R}^D$ is the diagonal component.

With respect to our models setting, this procedure affects two things. First, the neural networks used to parameterize the latent variable distribution will change in architecture. Specifically, the output layer of the inference net q_ϕ and transition net p_γ will consist of three blocks instead of two (recall Section 3.2.1), since the additional need to emit the entries of matrix F. Second, the new procedure to get a sample of latent variable s_t on the reparameterization trick step becomes the following.

$$\epsilon^{(lo)} \sim \mathcal{N}(0, I_r) \tag{4.3}$$
$$\epsilon^{(hi)} \sim \mathcal{N}(0, I_D) \tag{4.4}$$
$$s_t = F\epsilon^{(lo)} + {}_{\phi}\mu_t + {}_{\phi}\sigma_t \odot \epsilon^{(hi)} \tag{4.5}$$

where $\epsilon^{(lo)}$ generates the randomness due to the low-rank structure, and $\epsilon^{(hi)}$ generates the randomness due to the diagonal structure. Moreover, F, ${}_{\phi}\mu_t$ and ${}_{\phi}\sigma_t$ are respectively the low rank matrix, mean, and square root of variance of s_t, emitted by the inference net q_ϕ at time t.

4.2 Choosing the best-parsimonious model through MDL

Observe that, if we induce covariance structures through the method explained in Section 4.1.2, then we will end up at multiple models with different r (rank of the matrix F). In fact, we will have exactly D (the dimension of latent variables) models, i.e. model with $r = 0$ (our original model as in Chapter 3–no covariance) and $r = 2, 3, ..., D$. Exposed with these models, we might naturally want to choose one model that best of them. Of course, the *"best"* status must be based on some criteria. In this thesis, the criterion is set to be the one that does not overfit the data, i.e. performing equally good in both training and test data. This criterion is often in resemblance with choosing the best-parsimonious model, i.e. the model that is simple, but does perform satisfyingly good already.

Minimum Description Length (MDL) principle is just the framework of choosing a model with such a criterion. In other words, given comparable performance on training data, MDL would prefer the parsimonious model over the others. This fact allows us to have a continuation of a justification of using a low rank inducing covariance matrix, instead of a higher rank. It is not just because the true rank may be low, but also because even the true rank is actually higher, for small samples we may get better performace (less overfitting) if we use a small rank. MDL with small samples may select an overly simple model because it will generalize better than a model with the right (larger) number of parameters. This is due to lacking information to learn all these parameters (Grünwald, 2007).

As the name suggests, in MDL, we associate models with their corresponding codelength. Of course, we should have settled on some code first before talking about any codelength. To make the term code precise, below is given the definition, taken from (Grünwald, 2007).

Definition 4.2.1 (Coding Systems, Codes, Description Methods)**.** *Let $\mathcal{A}$ be some alphabet and $\mathbb{B}^* = \cup_{k\geq 1}\mathbb{B}^k$, where $\mathbb{B}$ is the binary alphabet, $\mathbb{B} = \{0, 1\}$, to encode the data from $\mathcal{A}$.*
Coding system: *A coding system C is a relation between $\mathcal{A}$ and $\mathbb{B}^*$. If $(a, b) \in C$, we say b is a code word for source sequence a. Moreover, we call a coding system singular if there exist $a, a' \in \mathcal{A}, a \neq a'$, that share a code word (i.e. there exist b such that $(a, b) \in C$ and $(a', b) \in C$).*
Description method: *A description method is a nonsingular coding system, that is, a coding system such that each $b \in \mathbb{B}^*$ is associated with at most one $a \in \mathcal{A}$.*

Code: *A code is a description method such that each $a \in \mathcal{A}$ is associated with at most one $b \in \mathbb{B}^*$.*

Motivated from viewing descriptions as messages, it is then necessary to have codes that are guaranteed to remain nonsingular under concatenation. *Prefix-free codes* are a class of code that satisfy this property. They are the codes with the property that no extension of a code word can itself be a code word. Because of this, in MDL, every term *code* always refers to prefix-free code without further clarification. Next, we present the central foundation of MDL framework, in the name of Kraft Inequality, see (Grünwald, 2007) for a proof.

Theorem 4.2.1 (Kraft Inequality)**.** *For any prefix-free code C for finite alphabet $\mathcal{A} = \{1, ..., m\}$, the code word lengths $L_C(1), ..., L_C(m)$ must satisfy the inequality*

$$\sum_{x \in \mathcal{A}} 2^{-L_C(x)} \leq 1$$

Conversely, given a set of code word lengths that satisfy this inequality, there exist a prefix-free code with these code word lengths.

Kraft inequality states that there is one to one correspondence between codelength and probability distribution. Thanks to the theorem, we can think of any probability distribution p as defining a prefix-free code such that for each outcome x, $-\log p(x)$ is the codelength of this probability distribution on the outcome. There is another term to be introduced before we can go to the model selection procedure through MDL, namely *universal code*. This is presented in the following.

Let $\bar{P}$ and P be two distributions on $\mathcal{X}^t$. Suppose we try to encode a sequence $x^t \in \mathcal{X}^t$ based on $\bar{P}$, and compare the number of bits we need with how well we would have done if we had used P. The difference may be called *redundancy* as defined below.

$$\text{RED}(\bar{P}, P, x^t) := -\log \bar{P}(x^t) - [-\log P(x^t)] \tag{4.6}$$

We may consider the worst-case redundancy of $\bar{P}$ over P, over all possible sequences of length t:

$$\text{RED}_{\max}(\bar{P}, P) := \max_{x^t \in \mathcal{X}^t} \text{RED}(\bar{P}, P, x^t) \tag{4.7}$$

we call distribution $\bar{P}$ is universal relative to set of models $\mathcal{M}$ if for all $P \in \mathcal{M}$, $\text{RED}_{\max}(\bar{P}, P)$ only grows slowly with t. Or, loosely speaking, let $\mathcal{L}$ is the corresponding set of codes of $\mathcal{M}$, define $\hat{L}(x^t) := \min_{L \in \mathcal{L}} L(x^t)$ (i.e. the best code in $\mathcal{L}$ to describe x^t), **universal code** $\bar{L}(x^t)$ is the code that, no matter what x^t arrives, $\bar{L}(x^t)$ is not much larger than $\hat{L}(x^t)$.

According to the MDL principle, suppose we observe some data X and the goal is to select a model from a list $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, ...\}$ that is a good explanation (not overfit-parsimonious) for data X. The general recipe for doing such a task via MDL is as follows (Grünwald, 2007):

1. Associate a universal code, which represented by universal probabilistic model $\bar{P}(\cdot|j)$ for each $\mathcal{M}_j$
2. Design a code on the model indices j, with some codelength function $\dot{L}(j)$

3. Define a code with length $\bar{L}(X) := \dot{L}(j) - \log \bar{P}(X|j)$
4. Choose the model $\mathcal{M}_j$, where j is the minimizer index of codelength in step 3 above

While we can perform the procedure above on both latent variable distributions (s_t for Phase 1, z_t for Phase 2), in this thesis we only induce the covariance structure over the latent variable distributions on Phase 1 since it is this phase that would typically be more complex than Phase 2. Intuitively, this is because Phase 1 models the overall *environment state*, while Phase 2 just models the other agents' *mental state*.

4.2.1 Prequential plug-in code

We use the *prequential plug-in code* as the universal code for our MDL model selection. Providing a glimpse background on this universal code, prequential plug-in code on some data sequence x^T is the code that corresponds to the following factorized probability model.

$$\bar{P}_{\text{plug-in}}(x_{1:T}) := \prod_{t=1}^{T} P_{\mathbf{MLE}(x_{1:t-1})}(x_t) \tag{4.8}$$

where $\mathbf{MLE}(x_{1:t-1})$ denotes the maximum likelihood estimator of probability model P given data sequence x^{t-1}. Meanwhile, to be able to use prequential plug-in code, there is one condition that has to be satisfied. That is, the probability model considered must constitute a *probabilistic source*.

Definition 4.2.2 (Probabilistic source)**.** *Let $\mathcal{X}$ be a sample space. A probabilistic source with outcomes in $\mathcal{X}$ is a function $P : \cup_{t\geq 1}\mathcal{X}^t \cup \mathcal{X}^0 \to [0, \infty)$ such that for all $t \geq 0$, all $x^t \in \mathcal{X}^t$ we have:*

1. $\sum_{z\in\mathcal{X}} P(x^t, z) = P(x^t)$ *(compatibility condition)*
2. $P(x^0) = 1$

Proposition 4.2.1. *The generative model in Phase 1 given in Section 3.2.1 constitutes a probabilistic source.*

Proof. Suppose that we denote the domain of observation o_t and environment state s_t as $\mathcal{O}$ and $\mathcal{S}$, respectively. Note that these domains are identical throughout all time steps. To prove the proposition, it suffices to show that the generative model satisfies the *compatibility condition* (Grünwald, 2007), i.e. for all $T \geq 1$, all $(o_{1:T-1}, s_{1:T-1}) \in \mathcal{O}_{1:T-1} \times \mathcal{S}_{1:T-1}$,

$$\int_{\mathcal{O}} \int_{\mathcal{S}} p(o_{1:T}, s_{1:T}|c_{1:T}) ds_T do_T = p(o_{1:T-1}, s_{1:T-1}|c_{1:T-1}) \tag{4.9}$$

Note that, by the factorization of the generative model given in (3.1), we can write

$$\begin{aligned}
\int_{\mathcal{O}}\int_{\mathcal{S}} p(o_{1:T}, s_{1:T}|c_{1:T})ds_T do_T &= \int_{\mathcal{O}}\int_{\mathcal{S}} \prod_{t=1}^{T} p_\theta(o_t|s_t)p_\gamma(s_t|s_{t-1}, c_t)ds_T do_T \\
&= \int_{\mathcal{O}}\int_{\mathcal{S}} \left(\prod_{t=1}^{T-1} p_\theta(o_t|s_t)p_\gamma(s_t|s_{t-1}, c_t)\right) \\
&\qquad \cdot p_\theta(o_T|s_T)p_\gamma(s_T|s_{T-1}, c_T)ds_T do_T \\
&= \prod_{t=1}^{T-1} p_\theta(o_t|s_t)p_\gamma(s_t|s_{t-1}, c_t) \\
&\qquad \cdot \int_{\mathcal{O}}\int_{\mathcal{S}} p_\theta(o_T|s_T)p_\gamma(s_T|s_{T-1}, c_T)ds_T do_T \\
&= \prod_{t=1}^{T-1} p_\theta(o_t|s_t)p_\gamma(s_t|s_{t-1}, c_t) \\
&= p(o_{1:T-1}, s_{1:T-1}|c_{1:T-1}) \qquad (4.10)
\end{aligned}$$

Note that $p_\theta(o_T|s_T)p_\gamma(s_T|s_{T-1}, c_T)$ in the third equality is just the joint probability of the modelled variables at time T, hence it integrates to 1. This implies the fourth equality. □

Using the probabilistic model in (4.8), we arrive at the codelength function (denoted by $\bar{L}$) for the prequantial plug-in code as follows.

$$\begin{aligned}
\bar{L}_{\textbf{plug-in}}(x_{1:T}) &= -\log \bar{P}_{\textbf{plug-in}}(x_{1:T}) \\
&= \sum_{t=1}^{T} -\log P_{\textbf{MLE}(x_{1:t-1})}(x_t) \qquad (4.11)
\end{aligned}$$

Before we discuss the implementation details of using the code, it might be rewarding to understand intuitively why such a formulation of universal model in (4.8) corresponds to a codelength function that suits the goal we set, i.e. to select the best-parsimonious model. The answer is, notice each term in the sum in (4.8) represents the model that only be trained based on the first $t-1$ data entries to predict the $t-$th sequence entry. It means that the $t-$th data entry is a completely new for the trained model's point of view. Therefore, we can interpret the model construction as iteratively performing *quasi leave-one-out cross-validation* at each time step (we add the term *"quasi"* since we do not perform any permutation on the taken-out data). This resembles the principle of parsimony in the first place, i.e. we are encouraged to select the simpler model, that is, the one that fits the data already good although only be trained with less training data. Observe that more complex models with comparable fitting performance will result in a larger codelength in this code construction, since they need to be trained with more samples to achieve that same level of fitting performance.

4.2.2 Implementation details

To proceed with the implementation, we have to define the expression of the prequential plug-in code (the one that analogue to (4.11)) in the context of our model. Since in Phase 1 the data we model is the joint sequence of observations and environment states, conditioned on all actions i.e. $(o_{1:T}, s_{1:T}|c_{1:T})$, the model that becomes

the prequential plug-in model is really the generative model given in (3.1). The expression we are looking for is thus

$$\begin{aligned}\bar{L}_{\textbf{plug-in}}(o_{1:T}, s_{1:T}|c_{1:T}) &= -\log \bar{P}_{\textbf{plug-in}}(o_{1:T}, s_{1:T}|c_{1:T}) \\ &= -\log p(o_{1:T}, s_{1:T}|c_{1:T}) \\ &= -\log \left(\prod_{t=1}^{T} p_{\theta_{t-1}}(o_t|s_t) p_{\gamma_{t-1}}(s_t|s_{t-1}, c_t) \right) \\ &= \sum_{t=1}^{T} -\log \left(p_{\theta_{t-1}}(o_t|s_t) p_{\gamma_{t-1}}(s_t|s_{t-1}, c_t) \right) \end{aligned} \tag{4.12}$$

where we have introduced a slightly new different notation on the model parameters (or equally the neural networks parameters). θ_{t-1} denotes the emission net parameters yielded from training the model with only using data sequence up to time index $t-1$, i.e. $o_{1:t-1}$ and $c_{1:t-1}$. This is analogous for γ_{t-1}. Thus, notice that for each term in (4.12), the predicted data o_t and s_t are really new (unseen before) for the models.

With respect to encoding the model indices, recall that we have D models to be considered, where D is the dimension of the latent variables. Let us represent the model with no covariance structure with index $j = 1$, and the model with low rank inducing covariance matrix $r = 2, ..., D$ with $j = 2, ..., D$, respectively. Therefore, our indices set is $J = \{1, 2, ..., D\}$. In this thesis, we use the code that corresponds to uniform distribution $w(\cdot)$ over the set J. Thus, the corresponding codelength is

$$\dot{L}(j) = -\log w(j), \qquad \forall j \in J \tag{4.13}$$

Note that the codelengths are identical for all j, therefore it does not play any role in our model selection task. Because of this, we omit this part of code hereafter.

4.2.3 Discussion on using other universal codes

In MDL, we prefer codes that have strong universality property. Regarding this desiderata, we can distinguish the universality property of codes into two categories, based on what kind of small redundancy the codes achieve: individual universality or stochastic universality (Grünwald, 2007). For the first, we require that for each $P \in \mathcal{M}$ under consideration, the codes achieve small redundancy for all sequences. Whilst, for the latter, we *only* require the codes to have small redundancy in *P-expectation*, where $P \in \mathcal{M}$ is our assumed underlying distribution of the sequences. It is known that stochastic universality is a weaker notion of universality compared to individual universality, i.e. every code that achieves individual universality also achieves stochastic universality, but not vice-versa.

Meanwhile, despite being naturally fit with the model characteristics (predictive and sequential), it is known that the prequential plug-in code we use in this thesis only achieves stochastic universality. We therefore might want to use the other universal codes that typically known to achieve individual universality, such as normalized maximum likelihood (NML) or Bayesian code. Nonetheless, we demonstrate below that using them in this work is not an easy task to accomplish.

NML code

For every sequence $x^T \in \mathcal{X}^T$, its NML code is the code that corresponds to the following distribution.

$$\bar{P}_{\mathbf{NML}}(x^T) = \frac{P_{\mathbf{MLE}(x^T)}(x^T)}{\sum_{y^T \in \mathcal{X}^T} P_{\mathbf{MLE}(y^T)}(y^T)} \tag{4.14}$$

Recalling our setting, the data is the joint sequence of observations and environment states, conditioned on actions, i.e. $(o_{1:T}, s_{1:T}|c_{1:T})$. Therefore, the distribution that analogue to (4.14) is

$$\bar{P}_{\mathbf{NML}}(o_{1:T}, s_{1:T}|c_{1:T}) = \frac{P_{\mathbf{MLE}}(o_{1:T}, s_{1:T}|c_{1:T})}{\sum_{o'_{1:T} \in \mathcal{O}^T, s'_{1:T} \in \mathcal{S}^T} P_{\mathbf{MLE}}(o'_{1:T}, s'_{1:T}|c_{1:T})} \tag{4.15}$$

While the domain of observation $\mathcal{O}$ can be simply finite depending on the specific problem setting, one fundamental assumption in our models is that $\mathcal{S}$ is real valued. This implies the denominator in (4.15) incorporates multiple integration over the whole real line, which results in an infinite value. Thus, we conclude that the NML code is undefined, since the denominator in (4.15) is infinite.

Bayesian code

In this type of universal code, for sequence $x^T \in \mathcal{X}^T$ and prior $w(\cdot)$ over the model parameter space Θ, the distribution (universal model) that corresponds to the Bayesian code is

$$\bar{P}_{\mathbf{Bayes}}(x^T) = \int_{\Theta} p_\theta(x^T) w(\theta) d\theta \tag{4.16}$$

Thus in our setting it becomes

$$\begin{aligned} \bar{P}_{\mathbf{Bayes}}(o_{1:T}, s_{1:T}|c_{1:T}) &= \int_{\Theta,\Gamma} p_{\theta,\gamma}(o_{1:T}, s_{1:T}|c_{1:T}) w(\theta, \gamma) d\theta, \gamma \\ &= \int_{\Theta} \int_{\Gamma} p_\theta(o_{1:T}|s_{1:T}) p_\gamma(s_{1:T}|c_{1:T}) w_1(\theta) w_2(\gamma) d\gamma d\theta \end{aligned} \tag{4.17}$$

Notice in our models, θ and γ are the parameters (weights and biases) of neural networks, specifically the emission and transition net, respectively. This is due to the fact that all distributions involved in our models are parameterized by neural networks. The bad news is, this implies a large number of parameters. To get an idea about this, the transition model (Phase 1, without covariance) alone has 473 parameters, not to mention the emission net's. Therefore, it is immediate to see that the Bayesian code is hardly tractable to compute in our case, because of so many terms of integrations in (4.17).

Chapter 5

Models Implementation

In this chapter, we perform an empirical study by implementing the models on a simple problem. We start by introducing the problem in details. After that, we apply our models on the problem to model observations, as well as the other agent's policy, following the guidelines presented in Section 3.3. Moreover, especially for modelling the observations, we also induce covariance structures and perform model selection over the resulting models via MDL principle (see Chapter 4).

5.1 Problem exposition

We consider a simple "2D Sailing" problem. In this problem, there is a boat sailing in the ocean through time, whose position is determined by the boat speed and heading direction. This problem constitutes a MAS since the boat speed and heading direction are each implied from actions taken by two distinct agents.

5.1.1 Deterministic system dynamics

The environment in this MAS is a boat which currently in stationary (idle) situation in a certain ocean. To take into account the time dimension, we impose the discrete time framework on this problem. Therefore, we can say the idle situation of the boat as the initial state, i.e. the boat state at time step $t = 0$. In the next time step $t = 1$, the boat will start sailing, exploring the area.

Regarding the system dynamics, we assume that there are two things which determine the boat position. They are speed and heading direction, both are subject to change through time. We can think of this problem as a MAS in a collaborative setting where for each time step, the first agent's (the controllable agent) task is to set the boat speed while the second agent sets the boat direction. Next, both agents might want to monitor some information that relevant to describe the latest situation of the boat (environment) in a timely basis. We assume the information consists of the current speed (v_t), heading direction (α_t), and spatial coordinates (x_t, y_t) of the boat. From now on, we call these four time-dependent variable together as the true environment state and denote it as $\tilde{s}_t = (v_t, \alpha_t, x_t, y_t)$.

To have a simple problem at hand, we assume that there are only two options or actions available for changing the boat speed, either increasing or decreasing the previous speed by one speed unit. Similarly, we also assume that there are also only two options for changing the heading direction, either be $+\pi/4$ or $-\pi/4$ relative to the previous heading direction. To this end, suppose for a given t we denote the

action to change the speed as u_t, while a_t is the notation for action changing the direction. Using this notation, we have $u_t \in \{\cdot + 1, \cdot - 1\}$, $a_t \in \{\cdot + \frac{\pi}{4}, \cdot - \frac{\pi}{4}\}$.

Based on these two actions sequences, i.e. $u_{1:T} = [u_1, ..., u_T]$ and $a_{1:T} = [a_1, ..., a_T]$, we can derive all sequence values of variables that establish the environment state by the following set of recursive relations:

$$
\begin{aligned}
v_t &= v_{t-1} + u_t \\
\alpha_t &= \alpha_{t-1} + a_t \\
x_t &= x_{t-1} + v_t \cos \alpha_t \\
y_t &= y_{t-1} + v_t \sin \alpha_t
\end{aligned}
$$

5.1.2 System dynamics with noise

After having the deterministic system dynamics as in the previous section, we want to slightly increase the problem complexity by also considering some noises that could be in place in the real world setting. One reasonable type of noise could come from the fact that, the realization of executed actions can be slightly different from the exact magnitude intended, because of some disturbances. For example, we could have a change of $+\frac{\pi}{4} + \frac{\pi}{36}$ in direction, instead of the one $+\frac{\pi}{4}$ we executed, due to severe wind. To accommodate this value discrepancy, we introduce new notations Δv_t and $\Delta \alpha_t$ to denote the true changes occurred as compared to the intended actions executed: u_t and a_t, respectively. There is another type of noise we consider, namely slip and drift. This second type of noise causes a discrepancy between the true coordinate (x_t, y_t) where the boat actually ends up with and their derived (analytic) values based on the system dynamics.

Let us denote the aforementioned noises respectively as $\epsilon_v, \epsilon_\alpha, \epsilon_x$, and ϵ_y. We then have these following recursive relations of the stochastic system dynamics:

$$
\begin{aligned}
\Delta v_t &= u_t + \epsilon_v \\
\Delta \alpha_t &= a_t + \epsilon_\alpha \\
v_t &= v_{t-1} + \Delta v_t \\
\alpha_t &= \alpha_{t-1} + \Delta \alpha_t \\
x_t &= x_{t-1} + v_t \cos \alpha_t + \epsilon_x \\
y_t &= y_{t-1} + v_t \sin \alpha_t + \epsilon_y
\end{aligned}
$$

5.1.3 Partial observability

To view the problem as a suitable use case for Phase 1 of the models, we assume that both agents have the problem of *partial observability*, i.e. they only have the access to observe the boat current coordinate (x_t, y_t) which they read from the boat's radar output. In other words, the two remaining information from the true environment state $\tilde{s}_t$, namely actual speed v_t and actual heading direction α_t are unknown to them.

5.1.4 Illustration of trajectory

In order to have a better understanding of the problem, in this section we look at two samples of the boat's trajectory, each for both setting of dynamics (deterministic and stochastic). To proceed further, we need to define several more characteristics. First is the policies of both agents. We assume that the first agent's policy is just randomly selected from a specified Bernoulli distribution, particularly we assume for all t, $u_t \sim$ i.i.d. Bernoulli$(p(\cdot + 1) = 0.5)$. While for the second agent, we assume that his policy depends on two things, namely his compatriot's action u_t and some information (the observable parts) from the previous time step environment state $\tilde{s}_{t-1}$. Specifically, we set the policy as given in Table 5.1.

u_t	Condition	a_t
$+1$	$\lvert x_{t-1}\rvert \geq \lvert y_{t-1}\rvert$	$+\pi/4$
	$\lvert x_{t-1}\rvert < \lvert y_{t-1}\rvert$	$-\pi/4$
-1	$\lvert x_{t-1}\rvert \geq \lvert y_{t-1}\rvert$	$+\pi/4$
	$\lvert x_{t-1}\rvert < \lvert y_{t-1}\rvert$	$+\pi/4$

TABLE 5.1: The second agent's policy (heading direction)

Another thing to be specified is the initial state of the boat, we assume that the boat starts right at the center of coordinate, heading east, with zero initial speed, i.e.

$$\tilde{s}_0 = (v_0, \alpha_0, x_0, y_0) = (0, 0, 0, 0),$$

Lastly, we assume that each trajectory has an identical length of 20 time steps, i.e. $t = 1, ..., 20$. With these characteristics, the sample of trajectories for both deterministic and stochastic setting can be found in Figure 5.1. Note that, an assumption on the noises characteristics is needed to create a sample trajectory for the latter setting. To this end, we assume all noises are Gaussian with zero mean and a specified variance as given below.

$$\begin{aligned}
\epsilon_v &\sim \mathcal{N}(0, 0.1^2) \\
\epsilon_\alpha &\sim \mathcal{N}\left(0, \left(\frac{\pi}{60}\right)^2\right) \\
\epsilon_x &\sim \mathcal{N}(0, 0.25^2) \\
\epsilon_y &\sim \mathcal{N}(0, 0.25^2)
\end{aligned}$$

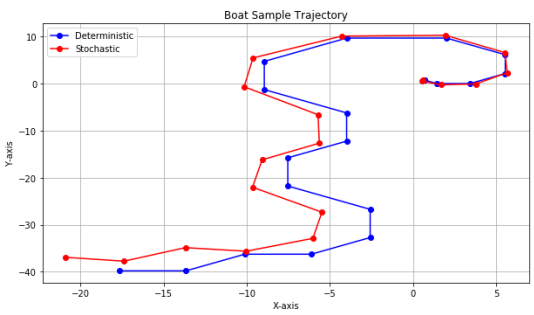

FIGURE 5.1: Boat sample trajectory

5.2 Modelling observations

We generate 1500 trajectories of the described **system with noise** to become our training data, while we construct another 200 trajectories to become test data. In this section, we utilize Phase 1 of the proposed models to fit the sequence of observations conditioned on both actions, just as presented in Section 3.3.1. In the following sections, we first discuss the implementation details of the model for each assumption on the latent variable (environment state) distributions, i.e. with or without covariance structure. Afterwards, we discuss the results from the model training, as well as model selection implementation amongst the constructed models. In this work, we set the dimension of latent variable s_t as four dimensional. Note that we could go with a different number, since we position ourselves not knowing the ground truth, i.e. the true environment state characteristics at all.

5.2.1 Implementation details

First of all, we always model the observations to be Gaussian with no covariance structure in any setting, since theay are coordinate pairs (real valued) in this problem. The first model is the one that assumes Gaussian with no covariance structure on the latent variables. In addition, we set the initial value as an isotropic Gaussian. The generative model in this model setting is thus the following. The graphical representation of this generative model can be seen more clearly in Figure 5.2.

$$s_0 \sim \mathcal{N}(s_0; 0, 0.01 \cdot I) \tag{5.1}$$

$$s_t \sim p_\gamma^{(1)}(s_t|s_{t-1}, c_t) = \mathcal{N}(s_t; {}_\gamma\mu_t, \text{diag}({}_\gamma\sigma_t^2)) \tag{5.2}$$

$$o_t \sim p_\theta^{(1)}(o_t|s_t) = \mathcal{N}(o_t; {}_\theta\mu_t, \text{diag}({}_\theta\sigma_t^2)) \tag{5.3}$$

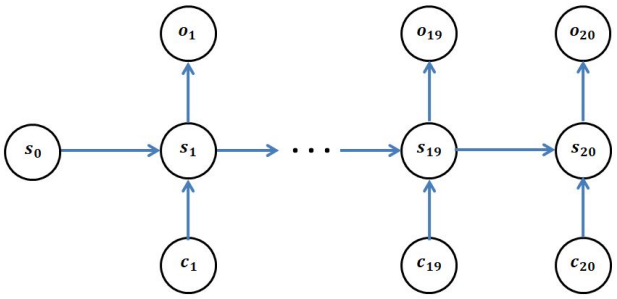

FIGURE 5.2: Generative model of Phase 1 for the problem

We parameterize each distribution $p_\theta^{(1)}, p_\gamma^{(1)}$, and $q_\phi^{(1)}$ by neural networks, such that θ, γ and ϕ are the parameters of neural networks $p_\theta^{(1)}, p_\gamma^{(1)}$, and $q_\phi^{(1)}$ respectively. For detailed specifications of the neural networks, see Table 5.2, while a concrete architectural form of one of them can be found in Figure 5.3. When training the model, we found that the loss (negative ELBO) decreases extremely slow when the emission net emits both μ_θ and $\log \sigma_\theta^2$ (even when epoch 5000-th is reached, the loss is still considerably high). Hence, we make the variance of observation fixed with the value of 1 (isotropic Gaussian).

Name	Notation	Inputs	Outputs	Architecture
Transition net	$p_\gamma^{(1)}$	s_{t-1}, c_t	$\mu_\gamma, \log \sigma_\gamma^2$ defining s_t	MLP with two hidden layers
Emission net	$p_\theta^{(1)}$	s_t	$\mu_\theta, \log \sigma_\theta^2$ defining o_t	MLP with two hidden layers
Inference net	$q_\phi^{(1)}$	s_{t-1}, c_t, o_t	$\mu_\phi, \log \sigma_\phi^2$ defining posterior s_t	MLP with two hidden layers

TABLE 5.2: Model components specification (no covariance)

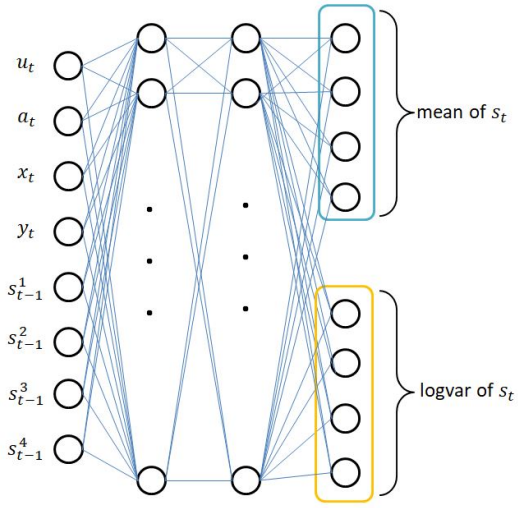

FIGURE 5.3: Concrete architectural form of $q_\phi^{(1)}$ (no covariance)

In addition to the details given in Table 5.2, we set each layer of all multi-layered perceptrons (MLP) to have 15 neurons, while tanh function acts as the non-linearity for both the inference and transition nets, and ReLU function for the emission net. Regarding neural networks training configuration, ADAM (Kingma and Ba, 2014) with annealed learning rate $\{0.01, 0.001\}$ is used as the SGD optimizer, and the number of epochs is set 250.

When the second setting is concerned, namely models with covariance structures, there are just a few changes on the setting that needed in place. To conform with the covariance structures assumption, we evoke an additional block on the output layer of both the inference and transition nets, whose task is to emit the entries of the matrix F (recall Section 4.1.2). Note that the number of output neurons added depends on the rank $r \in \{2, 3, 4\}$ (recall that the dimension of latent variable here is assumed 4). The rest configurations are exactly the same as what are set for the first model setting.

5.2.2 Model training and selection results

As dictated in Section 3.2.1, training is carried by maximizing the ELBO. The ELBO improvement for each model over the training epochs is given in Figure 5.4. We see that at the end of training, model with rank matrix $F = 4$ yields the highest ELBO

compared to the others, while the model without covariance yields the lowest value. This is confirmed by Table 5.3 that displays the average ELBO of each model over 5 repetitions (different pieces of training). Reading the table, we might be curious since the values are higher on test data. We argue that this is due to the considerably smaller size of the test data (200) compared to the training data (1500). Hence, the training data tends to contain some *uncommon* trajectories which lead to smaller likelihood. This argument is later confirmed. When we generate a new set of 1500 entries data (same size as train data) and compute the ELBO upon this new test data, the values are relatively the same as the ELBO on the train data.

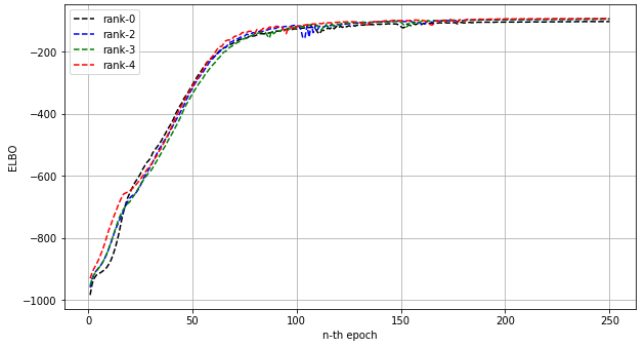

FIGURE 5.4: Phase 1 ELBO improvement over training between models

Model	Train data	Test data
rank-0	−102.36	−98.80
rank-2	−95.48	−91.77
rank-3	−95.45	−91.58
rank-4	**−93.64**	**−90.41**

TABLE 5.3: Phase 1 models ELBO results

After all training processes are complete, we have three well tuned neural networks ($p_\gamma^{(1)}$, $p_\theta^{(1)}$, and $q_\phi^{(1)}$) for each model. To recover the sequence of observation $o_{1:T}$, we apply the models on the test data by following the instruction given in Section 3.3.1. A sample of predicted trajectory (obtained by taking the mean of observations from their inferred distributions) from all trained models is depicted in Figure 5.5. Please note that the predictions are made sequential, i.e. every t-th of solid colored-point is the prediction of the corresponding model based on $t-1$-th solid black-point (the real observation).

With respect to model selection, the codelength of each model is given in Table 5.4. We breakdown the total codelength into its two sources, namely modelling the latent variables and modelling observations. To prevent a misguided conclusion, each of table entry is taken from an average of over 10 repetitions. According to the table, we conclude that **the model with rank matrix $F = 2$ is really our best-parsimonious model**, since it outperforms the rest of models in term of total codelength yielded. Notice the model with no covariance results in the largest codelength, both in modelling latent variables and observations. This, along with the ELBO values in Table 5.3 evidence our motivation to refine the model through inducing covariance in the first place.

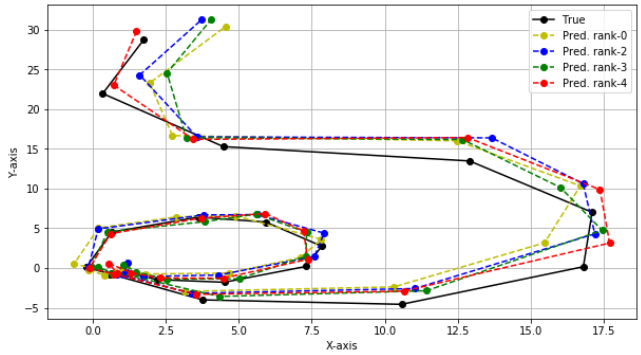

FIGURE 5.5: Sample of predicted trajectory

Model	Latent	Observations	Total
rank-0	132.86	125.06	256.92
rank-2	**97.13**	**113.97**	**211.10**
rank-3	102.38	121.02	223.40
rank-4	121.95	122.56	244.51

TABLE 5.4: Codelength results

5.3 Modelling the other agent's policy

In this section, we use our proposed models to perform the prediction on the other agent's anticipated action. As far as the problem described in this chapter is concerned, this means we are going to predict the action of the agent that tasked with the boat heading direction. With respect to the specific type of implementation used, we carry out the standard implementation version (see Section 3.3.2). To this end, we first have to train our Phase 2 model. The implementation details of it is given in the following paragraphs.

One of the ingredients of Phase 2 is the sequence of *system summary*, i.e. the environment state sequence $s_{1:T}$ from the previous Phase 1. We gather this sequence specifically from the model with rank matrix $F = 2$ in Phase 1, as the model is our best-parsimonious model according to the MDL principle. Regarding the model assumption, we set the latent variables (here is called *mental state* of the other agent) as two dimensional Gaussian with diagonal structured covariance matrix. Next, since there is only another agent with a binary action to be predicted, we model the emission model $p_\theta^{(2)}$ as Bernoulli distribution. Our generative model in this model setting is given below. Note that the first prediction is the one with index $t = 2$, it is due to the fact that s_1 (which becomes one input of the transition model $p_\gamma^{(2)}$ to yield the first z_1) is computed after seeing both first actions $c_1 = [u_1, a_1]$ (recall Figure 5.2). See Figure 5.6 for the graphical representation of the generative model.

$$z_0 \sim \mathcal{N}(z_0; 0, 0.01 \cdot I) \tag{5.4}$$

$$z_t \sim p_\gamma^{(2)}(z_t | z_{t-1}, s_t) = \mathcal{N}(z_t; {}_\gamma\mu_t, \text{diag}({}_\gamma\sigma_t^2)) \tag{5.5}$$

$$a_{t+1} \sim p_\theta^{(2)}(a_{t+1} | z_t) = \text{Bernoulli}(a_{t+1}; p(\cdot + \pi/4 = {}_\theta\eta_t)) \tag{5.6}$$

With respect to neural networks details, all distributions are parameterized by

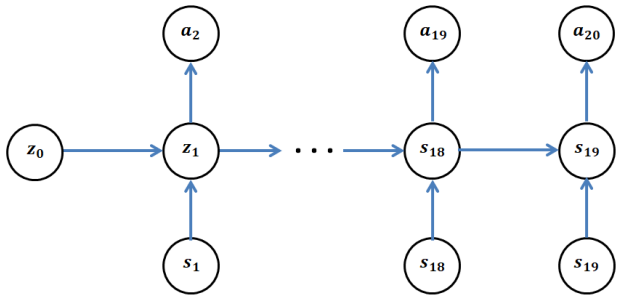

FIGURE 5.6: Generative model of Phase 2 for the problem

MLP with 2 hidden layers, each with 10 neurons. The non-linearities are set identical as Phase 1. ADAM optimizer with learning rate 0.001 is used, running for 300 epochs. Also to mention here, the output neuron of the emission net represents the logit value, instead of the Bernoulli parameter.

Reading the problem exposition, observe that it is implicitly assumed that the second agent (the one who sets the heading direction) executes his action (immediately) after the first agent (our controllable agent). Therefore, it makes sense to condition the distribution of latent variable z_t on the first agent's action u_{t+1} when predicting a_{t+1}, i.e. it does not hurt the online setting. Nevertheless, we also perform the original implementation as dictated by the generative model, i.e. without conditioning on u_{t+1}. The ELBO values (average over 5 repetitions) for the two implementations are given in Table 5.5. Also, the ELBO improvement over the training epochs for both implementations is given in Figure 5.7. Notice that, in contrast with the modified implementation (with u_{t+1}), the model seems to have a bottleneck to further increase the ELBO after the value reached around -12 in the original implementation. Based on the results, we conclude that our proposed models indeed can be used to predict the other agent's action reasonably well.

Implementation	Training data	Test data
without u_{t+1}	-12.42	-12.30
with u_{t+1}	-3.64	-3.54

TABLE 5.5: Phase 2 models ELBO results

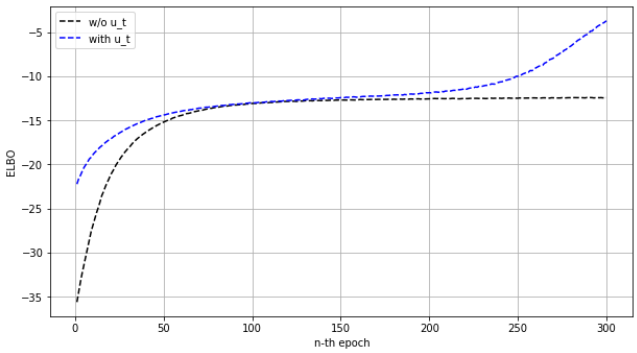

FIGURE 5.7: Phase 2 ELBO improvement over training between implementations

Chapter 6

Application on Multi-Agent Maximum Entropy Reinforcement Learning

Maximum entropy reinforcement learning is a relatively new framework within reinforcement learning, which also consider a maximization of an entropy term–along with the standard reward value term–on its learning objective (Ziebart et al., 2008). This term addition results in a more principled exploration strategy. Moreover, by also maximizing the entropy, the resulting policies try to learn all of the ways of performing the task. This is in contrast with merely learning the best way to perform the task, as in the standard reinforcement learning objective (Haarnoja et al., 2017). In this chapter, we present one extension of maximum entropy reinforcement learning in multi-agent settings, when an opponent policy model is assumed available. The motivation of doing this is to demonstrate how the opponent policy models we propose in this thesis can explicitly affect the agent's learned policy in a multi-agent maximum entropy reinforcement learning task. To this end, the section is first preceded by a discussion on the single-agent case of maximum entropy reinforcement learning.

6.1 Single-agent maximum entropy reinforcement learning

In the standard reinforcement learning setting, there is an agent who interacts with an environment $\mathcal{E}$ over a number of discrete time steps. At each time step t, the agent experiences a state s_t and selects an action u_t from some set of possible actions $\mathcal{U}$ according to its policy π, where π is a mapping from states s_t to actions u_t. In return, the agent receives the next state s_{t+1} and receives a scalar reward r_{t+1}. The process continues until the agent reaches a terminal state after which the process restarts. The recurrent diagram of this process is given in Figure 6.1. Reinforcement learning is formalized as a Markov Decision Process (MDP) which is defined below (Sutton and Barto, 1998).

Definition 6.1.1 (Markov Decision Process (MDP))**.** *A MDP is a tuple* $(\mathcal{S}, \mathcal{U}, T, R, \gamma)$*, where* $\mathcal{S}$ *is the set of states,* $\mathcal{U}$ *is the set of actions,* T *is a transition function* $\mathcal{S} \times \mathcal{U} \times \mathcal{S} \rightarrow [0, 1]$*,* R *is a reward function* $\mathcal{S} \times \mathcal{U} \rightarrow \mathbb{R}$*, and* $\gamma \in [0, 1)$ *is a discount factor. The transition function defines a probability distribution over next states as a function of the current state and the agent's action. The reward function defines the reward received when selecting an action from the given state.*

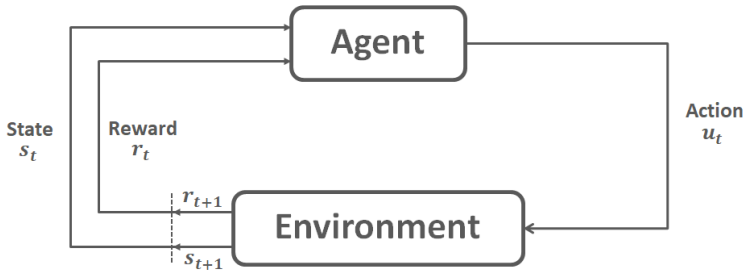

FIGURE 6.1: Agent-environment interaction in reinforcement learning

The agent experiences a sequence of state-action-reward tuples and each such trajectory yields a stochastic return, i.e. $\sum_{t\geq 0}\gamma^t r_t$. This return is subject to uncertainty in the policy, environment transition, and reward distribution. In light of this, it is useful to introduce a notion of the value of a state, since it can be used to represent how good is a state for the agent to be in. This is called state-value functions $V^\pi(s)$. The value function estimates the expected return for trajectories starting in state s, i.e. it is defined as the total discounted expected reward as follows.

$$V^\pi(s) = \sum_{t\geq 0}\gamma^t \mathbb{E}_\theta[r_{t+1}|s_0 = s], \qquad \forall s \in \mathcal{S} \tag{6.1}$$

where θ is the (unknown) reward distribution. Still in a close relation, we can make the action explicit by defining state-action value functions $Q^\pi(s,u)$ as the expected return for selecting action u in state s and following policy π. This function is useful to represent how good it is for the agent to pick action u while being in state s.

$$Q^\pi(s,u) = \sum_{t\geq 0}\gamma^t \mathbb{E}_\theta[r_{t+1}|s_0 = s, u_0 = u] \qquad \forall s \in \mathcal{S}, \forall u \in \mathcal{U} \tag{6.2}$$

$$= \mathbb{E}_\theta[r|s,u] + \gamma \mathbb{E}_{p_s}[V^\pi(s')|s,u] \tag{6.3}$$

where p_s is the state transition distribution over s' induced by the transition function T, and s' is the successor state that follows s. Note that, we can interpret (6.2) as the total discounted expected reward that follows from choosing action u in state s, and then following the policy π.

If we know both distributions p_s and θ, then we can use standard methods from dynamic programming (e.g. Value Iteration (Bertsekas, 1995)) to find the optimal state-action value function

$$Q^*(s,u) = \max_\pi Q^\pi(s,u) \tag{6.4}$$

where π is a policy, $\pi : \mathcal{S} \to \mathcal{U}$, mapping states to actions. Relatedly, solving an MDP (or equally reinforcement learning), consists of finding the policy that maximizes the optimal state-action value function in (6.4), that is

$$\pi^* = \arg\max_\pi Q^\pi(s,u) \tag{6.5}$$

Unfortunately, we often do not know the distribution θ. In such a case, the agent obtains samples from $p_s(s_{t+1}|s_t,u_t)$ and $\theta(r_{t+1}|s_t,u_t)$ through its interaction with the

environment. In this setting, Q^* is therefore subject to be estimated. We can do so by using a method called Q-learning (Watkins and Dayan, 1992). It starts with an arbitrary Q, and in step t upon observing s_t, u_t, r_{t+1}, and s_{t+1}, performs the update

$$Q_{k+1}(s_t, u_t) \leftarrow (1 - \alpha_t) Q_k(s_t, u_t) + \alpha_t \left(r_{t+1} + \gamma \max_u Q_k(s_{t+1}, u) \right) \tag{6.6}$$

where $\alpha_t \in [0, 1]$ is a learning rate and k is a learning step. If the learning rate satisfies certain conditions and the interaction itself uses an exploration policy that experiences each state-action pair infinitely many times, then Q-learning in (6.6) will converge to Q^*.

The expression in (6.5) denotes the best action to make for every state $s \in \mathcal{S}$ in the system. By rewriting the definition of $Q^\pi(s, u)$ and expressing the reward r_{t+1} as a function that depends on state s_t and action u_t, we have an equal expression for the optimal policy as follows.

$$\pi^* = \arg\max_\pi \sum_t \gamma \mathbb{E}_{s_t \sim p_s, u_t \sim \pi}[r(s_t, u_t)] \tag{6.7}$$

Note that the optimal policy resulting from (6.7) is deterministic. More accurately, a deterministic policy exists that is a maximizer, and commonly optimization procedures will select deterministic solutions. Meanwhile, in some cases, we might want to slightly modify the above objective and encourage instead a stochastic policy. This is particularly true when we emphasize more on agent exploration in multimodal problems (on the reward distribution). In such cases, we can modify our objective by augmenting the reward with an entropy term, such that the optimal policy aims to maximize its entropy at each visited state as follows (Haarnoja et al., 2017).

$$\pi^* = \arg\max_\pi \sum_t \gamma \mathbb{E}_{s_t \sim p_s, u_t \sim \pi}[r(s_t, u_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))] \tag{6.8}$$

If we use the above objective, instead of in (6.7), we work in the realm of (single-agent) *maximum entropy reinforcement learning*. It is a form of regularization, which adds the entropy bonus to differentiate between policies.

6.2 Multi-agent maximum entropy reinforcement learning

The generalization of single-agent reinforcement learning to a multi-agent setting is straightforward. It is given by the generalization of MDP, namely stochastic game (Busoniu, Babuška, and De Schutter, 2010).

Definition 6.2.1 (Stochastic Game). *A stochastic game is a tuple* $(n, \mathcal{S}, \mathcal{U}_{1\ldots n}, T, R_{1\ldots n}, \gamma)$, *where* n *is the number of players,* $\mathcal{S}$ *is the set of states,* $\mathcal{U}_i$ *is the set of actions available to player* i *(and* $\mathcal{U}$ *is the joint action space* $\mathcal{U}_1 \times \ldots \times \mathcal{U}_n$*),* T *is the transition function* $\mathcal{S} \times \mathcal{U} \times \mathcal{S} \to [0, 1]$, R_i *is the reward function for the* i*-th agent* $\mathcal{S} \times \mathcal{U} \to \mathbb{R}$

Notice that the above definition looks very similar to the MDP framework except there are multiple agents selecting actions and the next state and rewards depend on the joint action of those agents. With respect to the objective function to be optimized, one available approach is to take the point of view of the single agent which is our controllable agent. In this approach, the goal is to maximize the agent's own

reward. The objective function in the standard reinforcement learning setting is similar as in (6.7).

In the following, we provide one extension of maximum entropy reinforcement learning in multi-agent settings, when an opponent policy model is assumed available. We start with a new objective function for the learning task. Afterwards, we derive the optimal policy for the controllable agent, as well as the learning update to learn the optimal state-action value function that analogue to (6.6). In the construction, we explicitly demonstrate the usability of our opponent (other agents) policy model to affect the optimal policy of the controllable agent. For now on, we will always say the agent and the opponent to refer to the controllable agent and the other agents, respectively, for convenience.

6.2.1 New objective function

For the problem setting, we assume that the opponent's policy is stationary, i.e. it does not depend on time step. Let us distinguish the time periods into two: the first period which represents the *training* period of the agent to construct its opponent policy model (i.e. Phase 1 and Phase 2 models, see Chapter 3), and the second period which represents the *test* period. Our time framework in this chapter is the second period. That is, we consider an agent which wants to utilize its opponent policy model in a certain multi-agent reinforcement learning setting.

We consider the maximum entropy framework of reinforcement learning. To make use of the opponent policy model, we must (in some way) incorporate the model into the learning objective function. In addition, here we also equip a prior policy for the agent. The application of a prior on a policy is known to have several benefits for the agent, especially on exploration (Fox, Pakman, and Tishby, 2016). Let us denote the agent's policy (to be learned), the agent's prior policy, and the opponent's policy model as π, π_0, and ρ, respectively. To this end, we define the objective function by augmenting the reward term with the following terms:

$$\text{Entropy}(\rho)\,(\text{Entropy}(\pi) - \text{KL}(\pi||\pi_0)) \tag{6.9}$$

The motivation of the above terms is as follows. The $\text{Entropy}(\pi)$ term indicates that we work on maximum entropy setting, while the Kullback-Leibler (KL) term defines the relation between the agent's learned policy π and the prior policy π_0. We can think of this term as a regularizer of the learned policy to not deviate far away from the prior policy. Lastly, the $\text{Entropy}(\rho)$ term acts as a scaling coefficient. Notice that, when $\text{Entropy}(\rho)$ is small (i.e. we are relatively certain about the prediction of the upcoming opponent's action), a large KL term value will not penalize the objective value so much, meaning that it is okay for the policy to deviate significantly from the prior policy. In other words, the agent is encouraged to exploit the information it has to learn a brand new policy (the one that differs from the prior). On the contrary, when $\text{Entropy}(\rho)$ is large, the KL term is strongly encouraged to be small, i.e. the learned policy must be similar to the prior policy.

Before we give the formal expression of the proposed new objective function, we first have to settle on a notational matter. Recall when developing the models in Section 3.2, particularly in the first paragraph, we state that any action with time index t is based on perceiving observation with time index $t-1$, i.e. the action

mapping is $s_{t-1} \rightarrow u_t$. Meanwhile, in the standard reinforcement learning notation, the action mapping is always $s_t \rightarrow u_t$. In other words, what we refer to s_{t-1} in our DSSMs is actually s_t in reinforcement learning settings. Therefore, the notation in our DSSMs is not standard and we mention here that **especially in this chapter, we adjust the time index of observations and environment states in our developed DSSMs by adding each index by one**. Thus, our objective function is

$$\sum_t \gamma^t \mathbb{E}_{s_t \sim p_s, u_t \sim \pi(s_t), a_t \sim \tilde{\rho}(s_t)} \left[r(s_t, [u_t, a_t]) + \mathcal{H}(\rho(s_t)) \left(-\log \pi(s_t) - \log \frac{\pi(s_t)}{\pi_0} \right) \right] \tag{6.10}$$

where we have denoted the opponent's action at time t as a_t, the joint action as $[u_t, a_t]$. We also use $\pi(s_t), \tilde{\rho}(s_t)$, and $\rho(s_t)$ as the shorthand of $\pi(u_t|s_t), \tilde{\rho}(a_t|s_t)$, and $\rho(\hat{a}_t|s_t)$, respectively. Notice the notation $\tilde{\rho}(\cdot)$ denotes the true opponent policy (unknown), while the non-tilde version of the expression denotes our opponent policy model.

6.2.2 Optimal policy

Let us define the state-value function that analogue to (6.1) as follows.

$$V^\pi(s) = \sum_t \gamma^t \mathbb{E}_{p_s, \pi(s_t), \tilde{\rho}(s_t)} \left[r(s_t, [u_t, a_t]) - \mathcal{H}(\rho(s_t)) \left(\log \pi(s_t) + \log \frac{\pi(s_t)}{\pi_0} \right) \middle| s_0 = s \right] \tag{6.11}$$

Next, we introduce the shorthand

$$g^\pi(s_t, u_t) = \log \pi(s_t) + \log \frac{\pi(s_t)}{\pi_0} \tag{6.12}$$

Subsequently, we define the state-action value function that analogue to (6.2) as

$$\begin{aligned} Q^\pi(s, [u, a]) &= \mathbb{E}(r|s, [u, a]) + \gamma \mathbb{E}[V^\pi(s')|s, [u, a]] && (6.13) \\ &= \sum_t \gamma^t \mathbb{E}_{p_s, \pi(s_t), \tilde{\rho}(s_t)} [r(s_t, [u_t, a_t]) \\ &\qquad - \gamma \mathcal{H}(\rho(s_{t+1}))(g^\pi(s_{t+1}, u_{t+1})|s_0 = s, u_0 = u, a_0 = a)] && (6.14) \end{aligned}$$

Notice that $g^\pi(s_t, u_t)$ is not included in (6.14) since u is already known (given).

In this work, we assume that the joint policy between the agent and the opponent is conditionally independent given s, i.e. $\pi^{\text{joint}}([u, a]|s) = \pi(u|s) \cdot \tilde{\rho}(a|s)$. Using this assumption, from (6.11) and (6.14) we have

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{[u,a]}[Q^\pi(s, [u, a])] \\ &= \sum_{[u,a]} \pi^{\text{joint}}([u, a]|s)[-\mathcal{H}(\rho) g^\pi(s, u) + Q^\pi(s, [u, a])] \\ &= \sum_{[u,a]} \pi(u|s) \cdot \tilde{\rho}(a|s)[-\mathcal{H}(\rho) g^\pi(s, u) + Q^\pi(s, [u, a])] \end{aligned} \tag{6.15}$$

Proposition 6.2.1. *Given a fixed state-action value function $Q^\pi(s,[u,a])$, the optimal policy for the agent according to (6.15) is*

$$\pi^*(u|s) = \frac{\sqrt{\pi_0} e^{\frac{Q^\pi(s,[u,a])}{2\mathcal{H}(\rho)}}}{\sum_{[\bar{u},\bar{a}]} \sqrt{\pi_0} e^{\frac{Q^\pi(s,[\bar{u},\bar{a}])}{2\mathcal{H}(\rho)}}} \tag{6.16}$$

Proof. Note that the full form of (6.15) with a fixed $Q^\pi(s,[u,a])$ is

$$\begin{aligned} V^\pi(s) &= \sum_{[u,a]} \pi(u|s)\cdot\tilde{\rho}(a|s)[-\mathcal{H}(\rho)(\log\pi(u|s) + \log\frac{\pi(u|s)}{\pi_0}) + Q^\pi(s,[u,a])] \\ &= \sum_{[u,a]} \pi(u|s)\cdot\tilde{\rho}(a|s)[-\mathcal{H}(\rho)(2\log\pi(u|s) - \log\pi_0) + Q^\pi(s,[u,a])] \end{aligned} \tag{6.17}$$

The first order derivation of each term in (6.17) with respect to $\pi(u|s)$ is

$$\begin{aligned} \frac{dV^\pi(s)}{d\pi(u|s)} &= \tilde{\rho}(a|s)(-\mathcal{H}(\rho)(2\log\pi(u|s) - \log\pi_0) + Q^\pi(s,[u,a])) - \pi(u|s)\tilde{\rho}(a|s)\frac{2\mathcal{H}(\rho)}{\pi(u|s)} \\ &= \tilde{\rho}(a|s)(-\mathcal{H}(\rho)(2\log\pi(u|s) - \log\pi_0) + Q^\pi(s,[u,a])) - 2\tilde{\rho}(a|s)\mathcal{H}(\rho) \\ &= \tilde{\rho}(a|s)\mathcal{H}(\rho)(-2\log\pi(u|s) + \log\pi_0 + \frac{Q^\pi(s,[u,a])}{\mathcal{H}(\rho)} - 2) \end{aligned} \tag{6.18}$$

By making the above terms equal to zero, we have

$$\log\pi^*(u|s) = \frac{1}{2}\log\pi_0 + \frac{Q^\pi(s,[u,a])}{2\mathcal{H}(\rho)} - 1$$

or equivalently

$$\begin{aligned} \pi^*(u|s) &= \frac{\sqrt{\pi_0} e^{\frac{Q^\pi(s,[u,a])}{2\mathcal{H}(\rho)}} e^{-1}}{\sum_{[\bar{u},\bar{a}]} \sqrt{\pi_0} e^{\frac{Q^\pi(s,[\bar{u},\bar{a}])}{2\mathcal{H}(\rho)}} e^{-1}} \\ &= \frac{\sqrt{\pi_0} e^{\frac{Q^\pi(s,[u,a])}{2\mathcal{H}(\rho)}}}{\sum_{[\bar{u},\bar{a}]} \sqrt{\pi_0} e^{\frac{Q^\pi(s,[\bar{u},\bar{a}])}{2\mathcal{H}(\rho)}}} \end{aligned} \tag{6.19}$$

Observe that the second-order derivation of (6.17) with respect to $\pi(u|s)$ is always negative. Thus we conclude that the policy in (6.19) is indeed the optimal (maximizer) policy.

□

6.2.3 Q-learning update rule

In the previous section, we have derived the optimal policy for the proposed multi-agent reinforcement learning task. As we see in (6.19), the entropy of our opponent policy model explicitly affects the agent's learned policy. In this section, we continue to derive the learning rule for the optimal state-action value function $Q^*(s,[u,a])$. To this end, especially in this section, we made two simplifications on notation to save space. First, we denote the entropy of our opponent model only by $\mathcal{H}$. Secondly, the

concatenation of $[u, a]$ is replaced by simply with c.

We begin by substituting the optimal policy π^* in (6.19) to $V^\pi(s)$ in (6.15). Doing so, we have the value of $V^\pi(s)$

$$
\begin{aligned}
&= \frac{1}{\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} \sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \tilde{\rho}(s) \left[-\mathcal{H} \cdot \left(2 \log \frac{\sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}}{\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} - \log \pi_0 \right) + Q^\pi(s,c) \right] \\
&= \frac{1}{\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} \sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \tilde{\rho}(s) \left[-\mathcal{H} \cdot \left(\log \frac{\pi_0 e^{\frac{Q^\pi(s,c)}{\mathcal{H}}}}{\pi_0 \left(\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \right)^2} \right) + Q^\pi(s,c) \right] \\
&= \frac{1}{\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} \sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \tilde{\rho}(s) \left[-\mathcal{H} \cdot \left(\frac{Q^\pi(s,c)}{\mathcal{H}} - 2 \log \left(\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \right) \right) + Q^\pi(s,c) \right] \\
&= \frac{1}{\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} \sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \tilde{\rho}(s) \left[-Q^\pi(s,c) + 2\mathcal{H} \log \left(\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \right) + Q^\pi(s,c) \right]
\end{aligned}
\tag{6.20}
$$

To continue the simplification, here we assume that the prior policy π_0 is a uniform distribution. Moreover, we replace the true opponent's policy $\tilde{\rho}(s)$ by our opponent policy model $\rho(s)$. Thus we have

$$
V^\pi(s) \approx \frac{2\mathcal{H}}{\sum_c e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}}} \sum_c e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \rho(s) \log \left(\sum_c \sqrt{\pi_0} e^{\frac{Q^\pi(s,c)}{2\mathcal{H}}} \right) \tag{6.21}
$$

Finally, we plug (6.21) into (6.13). The optimal state-action value function $Q^*(s,c)$ is thus a fixed point of the following equation. Note that in the following we write s' and c' to denote the next state and the next joint action, respectively.

$$
Q^*(s,c) \approx \mathbb{E}(r|s,c) + \gamma \mathbb{E} \left[\frac{2\mathcal{H}}{\sum_{c'} e^{\frac{Q^*(s',c')}{2\mathcal{H}}}} \sum_{c'} e^{\frac{Q^*(s',c')}{2\mathcal{H}}} \rho(s) \log \left(\sum_{c'} \sqrt{\pi_0} e^{\frac{Q^*(s',c')}{2\mathcal{H}}} \right) \right] \tag{6.22}
$$

According to (6.22) in the above, we propose the Q-learning update rule to learn Q^* as follows.

$$
\begin{aligned}
Q_{k+1}(s_t, c_t) \leftarrow\ & (1 - \alpha_t) Q_k(s_t, c_t) \\
& + \alpha_t \left(r_{t+1} + \frac{2\mathcal{H}\gamma}{\sum_{c'} e^{\frac{Q_k(s',c')}{2\mathcal{H}}}} \sum_{c'} e^{\frac{Q_k(s',c')}{2\mathcal{H}}} \rho(s) \log \left(\sum_{c'} \sqrt{\pi_0} e^{\frac{Q_k(s',c')}{2\mathcal{H}}} \right) \right)
\end{aligned}
\tag{6.23}
$$

where k is the learning step, α_t is the learning rate, and r_{t+1} is the experienced reward at time $t+1$, i.e. $r(s_t, c_t)$.

Remark.

Note that the above Q-learning update is not yet guaranteed to converge to the optimal $Q^*(s_t, c_t)$. To be so, one first needs to prove that the operator on Q derived

from the recursive equation in (6.22) constitutes a contraction in a certain norm (Fox, Pakman, and Tishby, 2016).

Chapter 7

Conclusion & Future Work

In this thesis, we proposed a new formulation of deep state-space models (DSSMs) in multi-agent systems. The formulation constructed models that can be used to represent the environment dynamics (Phase 1), and ultimately to predict the other agents' future actions (Phase 1 together with Phase 2). Specifically, in Phase 1, the sequence of observations were fitted via self-introduced latent environment states. Whilst in Phase 2, other agents' mental states were modelled, conditioned on the environment states gathered from Phase 1. The models were built upon the framework of latent variable models, armed with variational inference principle in regard to approximating the true (but intractable) posterior distributions.

In addition, we explored extensions to make the model richer and thus more flexible, by inducing covariance structures on the latent variable distributions. This left us with several models available, corresponding to the rank of the covariance inducing matrix. We then performed model selection on these models, in the spirit of choosing the best-parsimonious model, via the minimum description length (MDL) principle. For the result in a given simple problem, it turned out that the model with low rank (rank $= 2$) inducing covariance matrix is really the best-parsimonious model we were looking for, marked by the shortest average codelength over the others. To elaborate more, the model with no covariance structure performed the worst, while the performance decreased gradually for the model with higher ranks (rank 3 and 4). From this empirical implementation, we found that the models perform reasonably well to infer both the environment dynamics and the opponent's policy.

Finally, we also demonstrated one use case of the proposed models in a multi-agent reinforcement learning task. We worked on maximum entropy reinforcement learning framework and first defined a new learning objective function. It was shown that, for such a task, the agent's optimal policy does depend explicitly on the *opponent policy model*, i.e. the proposed models, in a certain way. Subsequently, a type of Q-learning update rule was proposed, based on a derived optimal state-action value function of the learning task.

To conclude, the proposed models are an effort to tackle the fundamental challenge in multi agent systems (MAS) field. By knowing the other agents' policy using our models, the agent will better design its own policy through choosing actions which are best responses to those opponents' anticipated moves. This in turns could lead to a better agents coordination. Nonetheless, the models are also beneficial in a competitive setting, where they can help the agent to outplay the opponents through a superior strategy. Overall, let us mention several good features of the proposed models, which make the models highly flexible. First, as already hinted, the models can be implemented in both collaborative and competitive setting of MAS. Second,

they can be used (in fact naturally) in partially observed environment settings, due to the inference net of Phase 1 of the models. Third, the models can be implemented in both paradigms of reinforcement learning, i.e. model-based and model-free learning. The models can act as the model of the environment to later be implemented in a model-based learning. This is particularly true when one only uses Phase 1 of the models. On the other hand, one can also work in a model-free setting using the models, for example, as demonstrated in Chapter 6. Lastly, thanks to its generative nature, the models can also be used to *imagine* any behavior of the other agents (not necessarily suggested by their–previously seen–original behavior) and later evaluate the effect it yielded to the system.

Nevertheless, there are several possible directions to follow-up the work done in this thesis. First, note that our proposed models possess one limitation in term of other agents' policy assumption, i.e. it is assumed that the policy of the opponents (the other agents) are stationary (recall Chapter 3). It is then of the importance to relax this assumption by dealing with learning opponents. That is, opponents which constantly change their policy along the trajectory. Second, to refine the models, we induce covariance structures across the dimensionality of latent variables (recall Chapter 4). Meanwhile, note that covariance structures do not capture multimodality of distributions. In light of this, it might be interesting to make the models even more flexible by also anticipating multimodality. This can be pursued next, by using for example normalizing flow methods (Rezende and Mohamed, 2015). Lastly, our study on multi-agent reinforcement learning use case of the models emphasizes more on the theoretical side (recall Chapter 6). It might be rewarding to perform an experimental study on this respect. This study would be beneficial due to two reasons. Firstly, we can empirically evaluate the goodness of the proposed multi-agent reinforcement learning setting, particularly with respect to the new objective function. Secondly, we can also evaluate how good is our opponent policy model in such a task. Overall, through this thesis, we have initiated the ground-work towards well-rounded inference about latent variable representations on opponent policy model. We believe that this could lead to fruitful developments in the area.

Bibliography

Albrecht, S. and P. Stone (2018). "Autonomous agents modelling other agents: A comprehensive survey and open problems". In: *Journal Artificial Intelligence* 258, 66-95.

Bertsekas, D. (1995). *Dynamic programming and optimal control*. Athena Scientific.

Blei, D.M., A. Kucukelbir, and J.D. McAuliffe (2017). "Variational Inference: A Review for Statisticians". In: *Journal of the American Statistical Association* 112 (518), 859-877.

Bloembergen, D. et al. (2015). "Evolutionary dynamics ofmulti-agent learning: A Survey". In: *Journal of Artificial Intelligence Research* 53, 659-697.

Brown, G. (1951). "Iterative solution of games by fictitious play". In: *Proceedings of The Conference of Activity Analysis of Production and Allocation*.

Busoniu, L., R. Babuška, and B. De Schutter (2010). "Multi-agent reinforcement learning: An overview". In: *Innovations in multi-agent systems and applications*.

Chalkiadakis, G. and C. Boutilier (2003). "Coordination in multiagent reinforcement learning: a Bayesian approach". In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*.

Claus, C. and C. Boutilier (1998). "The dynamics of reinforcement learning in cooperative multiagent systems". In: *Proceedings of AAAI/IAAI*.

Dorri, A., S. S. Kanhere, and R. Jurda (2018). "Multi-Agent Systems: A Survey". In: *IEEE Access*.

Fox, R., A. Pakman, and N. Tishby (2016). "Taming the noise in reinforcement learning via soft updates". In: *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*.

Fraccaro, Marco. (2018). "Deep Latent Variable Models for Sequential Data". In: *PhD Thesis, Technical University of Denmark*.

Furmston, T. and D. Barber (2010). "Variational methods for reinforcement learning". In: *Artificial Intelligence and Statistics*.

Grünwald, Peter. (2007). *The Minimum Description Length Principle*. MIT Press.

Haarnoja, T. et al. (2017). "Reinforcement learning with deep energy-based policies". In: *International Conference on Machine Learning* 1352–1361.

Hernandez-Leal, P. et al. (2017). "A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity". In: *arXiv preprint* arXiv:1707.09183.

Kingma, D.P. and J. Ba (2014). "A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Kingma, D.P. and M. Welling (2013). "Auto-Encoding Variational Bayes". In: *Arxiv preprint* arXiv:1312.6114v10.

Krupnik, O., I. Mordatch, and A. Tamar (2019). "Multi Agent Reinforcement Learning with Multi-Step Generative Models". In: *arXiv preprint* arXiv:1901.10251.

Levine, S. and V. Koltun (2013). "Variational policy search via trajectory optimization". In: *Advances in Neural Information Processing Systems*.

Levine, Sergey (2018). "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: *arXiv preprint* arXiv:1805.00909.

Miller, AC., NJ. Foti, and RP. Adams (2017). "Variational boosting: Iteratively refining posterior approximations". In: *International Conference on Machine Learning*.

Neumann, G. (2011). "Variational Inference for Policy Search in changing Situations". In: *International Conference on Machine Learning*.

Pu, Y. et al. (2016). "Variational autoencoder for deep learning of images, labels and captions". In: *Advances in Neural Information Processing Systems*.

Rawlik, K., M. Toussaint, and S. Vijayakumar (2012). "On stochastic optimal control and reinforcement learning by approximate inference". In: *International Joint Conference on Artificial Intelligence*.

Rezende, D.J. and S Mohamed (2015). "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning*.

Rezende, D.J., S. Mohamed, and D. Wierstra (2014). "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *International Conference on Machine Learning*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by backpropagating errors". In: *Nature*.

Sen, S. and G. Weiss (2001). *Learning in Multiagent Systems*. MIT Press.

Sun, C. et al. (2019). "Stochastic Prediction of Multi-Agent Interactions from Partial Observations". In: *International Conference on Learning Representation*.

Sutton, R. and A. Barto (1998). *Reinforcement Learning*. MIT Press.

Tian, Z. et al. (2019). "A Regularized Opponent Model with Maximum Entropy Objective". In: *arXiv preprint* arXiv:1905.08087.

Watkins, C. and P. Dayan (1992). "Q-learning". In: *Machine Learning*.

Wen, Y. et al. (2019). "Probabilistic Recursive Reasonig for Multi-Agent Reinforcement Learning". In: *International Conference on Learning Representation*.

Woolridge, M.J. (2002). *An Introduction to Multiagent Systems*. John Wiley Sons.

Ziebart, B.D. et al. (2008). "Maximum entropy inverse reinforcement learning". In: *AAAI Conference on Artificial Intelligence*.