Modelling Agents with Variational Autoencoders in Multi-Agent Sequential Decision Making

by

Herman Luc Lenferink

to obtain the degree of

Master of Science

at the Delft University of Technology,
to be defended publicly on Wednesday, January 25, 2022 at 15:00.

Student number:	4388356		
Faculty:	EEMCS		
Master programme:	Computer Science		
Track:	Artificial Intelligence Technology		
Thesis committee:	Dr. F. A. Oliehoek,	TU Delft,	Thesis Supervisor
	Dr. J. W. Böhmer,	TU Delft	

An electronic version of this thesis is available at
`http://repository.tudelft.nl/`.

Acknowledgements

First of all, I would like to thank Frans Oliehoek and Elena Congeduti for the opportunities they have given me throughout my studies. Starting with an individual research project that concluded my Bachelor and ending with this final thesis project which originated from a separate preceding exploratory literature survey. I really enjoyed the opportunity to combine machine learning, artificial intelligence and sequential decision making in these projects. I would also like to thank them both for their time and valuable feedback. I really appreciate it. Moreover, I would like to thank Elena for her guidance and all the meetings we have had. Even though the high-level concepts of this study are easily explained to other people, it was really nice to have the ability to discuss the topic in the finest detail. Grazie mille! Next, I would like to thank Wendelin Böhmer for willing to participate in my thesis committee, his time and his valuable feedback following the green light meeting as well as Davide Mambelli for his time and helpful feedback. Finally, I would like to thank my friends and family for their support. In particular, my parents and my girlfriend for their unconditional support and belief throughout my studies.

Abstract

The ability to model other agents can be of great value in multi-agent sequential decision making problems and has become more accessible due to the introduction of deep learning into reinforcement learning. In this study, the aim is to investigate the usefulness of modelling other agents using variational autoencoder based models in partially observable settings. Previous studies that model other agents using (variational) autoencoders have shown promising results. In these studies, a single protagonist agent learns representations of other agents to then use them as additional components of its observation space which is, as such, augmented with those representations. It is, however, not always entirely clear what is being modelled and what would be the best feature of the other agent to represent. Moreover, in these works, a comparison between the used variational autoencoder based models and a baseline classifier trained to solve the same classification task is missing. This study investigates which features can best be used for the augmentation of the observations of deep reinforcement learning agents and if these features can be represented by variational autoencoder based models. Subsequently, it compares these models with a baseline classifier that solves the same classification problem to find out which model yields the best results when used for augmenting observations. Overall, the results suggest that it is beneficial to augment the observations of deep reinforcement learning agents with features related to other agents learned in a pre-training phase. Another interesting result is that the baseline classifier achieves similar or better performance compared to the variational autoencoder based model. Further research needs to be conducted to confirm the soundness of these findings.

Contents

1. Introduction

Modern society is filled with problems that can be modelled as environments where multiple intelligent autonomous entities, called agents, have sequential interactions over time. Examples include traffic signal control [1], robotics [2] and resource balancing [3]. These problems can be hard to solve because, in addition to being large and complex, they require dealing with other agents' behaviour which potentially affects the environment dynamics. It would therefore be interesting to be able to model such behaviour. While there are numerous techniques to solve multi-agent sequential decision making problems [4, 5, 6, 7], until recently, employing sequential decision making algorithms in high dimensional real-world problems was computationally unfeasible. The introduction of deep learning into reinforcement learning (RL) [8, 9, 10] changed this and opened up new opportunities, making the ability to model the behaviour of other agents more accessible and interesting to investigate.

1.1 Multi-Agent Systems

The formal framework that defines environments with multiple agents is called a multi-agent system (MAS) [11]. An agent in this context is anything that is able to observe the environment through sensors and has the ability to act upon these observations [12]. Agents interact with the environment by taking actions based on these observations, which in turn affect the environment by changing its state. Subsequently, the agents receive new observations of the new state of the environment and are rewarded or punished for their actions. This study focuses on the common case in which the sensors only give the agent a partial and/or noisy view of the environment, i.e., the environment is partially observable. The strategy of an agent, a stochastic or deterministic decision rule that determines which action is taken, is called a policy.

1.2 A Running Example of Agent Modelling in Multi-Agent Systems

The tiger problem [13, 14] provides an example of a MAS in which some knowledge about the behaviour of the other agent can be of great value for successful decisions. In this problem, two agents stand in front of two closed doors: behind one of the doors lies a hungry tiger and behind the other door lies a treasure. The agents receive shared rewards and need to collect the treasure without communication. They first observe the environment by hearing the tiger behind one of the doors and then act either by opening a door or by listening. If both agents listen the probability of making a correct observation next turn is high. If instead a door is opened, the tiger and treasure are reset randomly and the observations are random. If the correct door is opened, then it is in the agents' interest to take the same action. Together they are able to collect a larger part of the treasure then alone. Moreover, if the wrong door is opened, it is also in the agents' interest to take the same action. In this case the agents can fend off the tiger together reducing the damage they incur. It can be concluded that in the tiger problem acting jointly is often beneficial and that it is therefore potentially valuable to model the behaviour of the other agent. Having a belief over the other agent's action allows the agent to cooperate, for instance, by opening the same door if it beliefs that the other agent is going to open a door.

1.3 Agent Modelling in the Literature

Modelling the behaviour of other agents is a broad subject and a variety of methods have been implemented for this task [15]. It is often referred to as "opponent modelling" which originates from the field of game theory where it was used in games such as poker and chess. In this study, the term opponent is avoided as it suggests that the techniques only apply in settings where agents present adversarial behaviour. One of the first examples of agent modelling is a study by Gmytrasiewicz and Doshi [16] who introduced a framework that models other agents recursively. In this work the authors take the perspective of one particular agent in a MAS. Panella and Gmytrasiewic [17] dropped the idea of modelling other agents recursively and generalized the framework by modelling the other agents with stochastic processes. More recently, advances in representation learning have been used to model other agents. A group of studies use (variational) autoencoders to learn representations of other agents [18, 19, 20]. In these studies, a single protagonist agent learns representations of other agents to then use them as additional components of its observation space which is, as such, augmented with those representations. The augmented decision making problem is then solved using deep RL. Variational autoencoders (VAEs) are of particular interest as they learn a distribution (a posterior) and thereby provide the ability to form a belief over the learned representations. Regular autoencoders simply learn a representation. Moreover, VAEs offer a way to influence the representations by means of specifying a prior over them, thereby giving more control over the representations than a regular autoencoder.

1.4 Agent Modelling with (Variational) Autoencoders

Modelling other agents with VAEs is quite challenging as it is difficult to control or know exactly which features are represented and preserved by the autoencoder. However, the following studies introduce some interesting approaches. Papoudakis and Albrecht [18] consider a partially observable setting and try to represent the policy of the other agent using a conditional variational autoencoder (CVAE). The CVAE is trained and subsequently, the samples of the learned posterior are used to augment the observation space of the agent whose perspective is taken. In a subsequent study by Papoudakis, Christianos and Albrecht [19] a different approach is taken. They also consider a partially observable setting but drop the idea of a CVAE and use an autoencoder instead. They assume that the latent variable contains information about both the policy of the other agent as well as the dynamics of the environment as perceived by the other agent. The samples of the learned posterior are used to augment the observation space of the agent whose perspective is taken. Finally, Zintgraf et al. [20] use a VAE in a fully observable setting and see the latent variables as a representation of the agent types. Instead of augmenting the observation space of the agent whose perspective is taken with samples from the learned posterior, they apply this framework in the context of Bayesian RL and augment the observation space with the parameters of the learned posterior.

1.5 Scientific Gap & Hypothesis

Although the studies show an interesting new way of modelling other agents using VAEs and how to leverage this knowledge for the sequential decision making problem, many aspects of these approaches still remain unclear. For example, in [18] it is not entirely clear what is being represented and a comparison with a baseline that uses unaugmented observations is lacking. In [19], the augmented models seem to outperform an unaugmented baseline; however, it remains unclear if using an autoencoder is beneficial over using a baseline classification network such as a long short-term memory neural network (LSTM). Moreover, in [19] as well as in [18] and in [20] it is unclear whether this intuition of the augmentation of the observation space of a RL agent is beneficial at all: suppose that the observations are augmented with a perfect representation of the desired feature, how does the RL agent then perform? Finally, it is also unclear why [18] and [20] use different components of the trained VAE to augment the observations with. Based on the results from these studies, it is hypothesized that because of the complexity of solving a general (deep) RL problem in a multiagent context, it may be beneficial to augment the observation space with features related to the other agents learned during a pre-training phase in a supervised fashion.

1.6 Aim & Research Questions

The aim of this study is to investigate the usefulness of the augmentation of the observations of a RL agent with representations of features related to other agents generated by VAE based models in partially observable settings. With this aim in mind the following questions are addressed:

1. What would be the best feature related to another agent to augment the observations with?

2. To what extent can these features be predicted by a VAE based model and how does this compare to an LSTM classifier?

3. What is the best way to leverage the VAE based model to augment the observation space in the RL problem and how does this model compare to an LSTM classifier?

This study is the first study that, in the context of agent modelling, compares the augmentation of observations with representations learned by a VAE based model and predictions made by an LSTM classifier that solves the same classification problem. Moreover, to the authors knowledge, it is the first study that, in the context of agent modelling, compares the approaches adopted by [18] and [20] to augment the observations and the first that applies the augmentation method used in [20] in a partially observable setting. Hopefully, this research will contribute to a deeper understanding of agent modelling with VAEs and the augmentation of observations using the learned models.

1.7 Document Structure

The remaining part of this study consists of 4 sections. First of all, Section 2 explains the necessary background knowledge required for this study and introduces the corresponding notation. It covers decision making frameworks, autoencoders and reinforcement learning. Next, Section 3 covers the approach and implementation. It explains in detail how the models of the other agents are created as well as how the observations are augmented. It also covers the neural network architectures and the RL algorithms used in this study as well as the software implementation. Subsequently, Section 4 covers the conducted experiments and corresponding analysis of results. Moreover, this section restates the aim and research questions, explains the experiment setup and covers the details of the environment used in the experiments. Finally, Section 5 draws the conclusions by revisiting the research questions and giving recommendations for future work.

2. Background

This section covers the three main elements of this study: decision making frameworks in Section 2.1, autoencoders in Section 2.2, and reinforcement learning in Section 2.3. The sections provide the necessary background knowledge for this study and introduce the notation and naming used throughout this study.

2.1 Decision Making Frameworks

Multi-agent sequential decision making problems can be modelled with a variety of (in some cases overlapping) frameworks, including: control theory frameworks, game theory frameworks, graph theory frameworks, decision theory frameworks and swarm intelligence frameworks [5]. This section introduces partially observable stochastic games (POSGs) [21, 22] which lie in the intersection of game theory and decision theory. Additionally, it introduces the frameworks on which POSGs are built and the notation corresponding to these frameworks.

Firstly, the frameworks on which POSGs are built will be explained, starting with the most basic framework, the Markov decision process (MDP) [12], in Section 2.1.1. Subsequently, its generalization to partially observable settings, partially observable Markov decision processes (POMDPs) [23, 21, 12], will be explained in Section 2.1.2. Next, POSGs which generalize POMDPs to the multi-agent setting as well as POSGs with shared rewards, called decentralized partially observable Markov decision processes (Dec-POMDPs) [21, 24, 25, 22], are addressed in Section 2.1.3[1]. Finally, the context where the subjective perspective of a single agent in a multi-agent problem is taken, is explained in Section 2.1.4.

2.1.1 Markov Decision Processes

An MDP is a mathematical framework that can be used to model the decision making of a single agent that interacts with a fully observable stochastically changing environment in discrete-time. An MDP is defined as a tuple $\langle S, A, \mathcal{T}, \mathcal{R}, h, b_0 \rangle$, where:

- S is a set of states s.
- A is a set of actions a.
- $\mathcal{T} : S \times A \mapsto \Delta(S)$ is the transition function, a mapping from states and actions to probability distributions over states[2].
- $\mathcal{R} : S \times A \mapsto \mathbb{R}$ is the reward function, a mapping from states and actions to real numbers r.
- h is the horizon, the number of times the agent will interact with its environment.
- $b_0 \in \Delta(S)$, is the initial state distribution at time $t = 0$.

1. The author assumes that the reader is familiar with the concept of agents, MDPs, POMDPs and Dec-POMDPs. More information about POMDPs and Dec-POMDPs can be found in [21] and an introduction of the other topics can be found in [12].
2. Δ(S) is used to denote the set of probability distributions over the set S.

In this model the agent acts according to its policy π which returns an action or a distribution over actions based on the current state. An agent can either have a deterministic policy, $\pi : S \mapsto A$, which maps states to actions or a stochastic policy, $\pi : S \mapsto \Delta(A)$, which maps states to distributions over actions. The probability of taking an action $a \in A$ given a state $s \in S$ is denoted as $\pi(a \mid s) : A \times S \mapsto [0, 1]$.

A step in an MDP is referred to as a time step $t \in 0, ..., h$. Moreover, a state, action and reward at time t are denoted by s_t, a_t and r_t respectively. At each time step $t = 0, ..., h-1$ the agent has to make a decision and takes an action according to its policy π based on the current state s_t. The chosen action a_t then influences the environment and causes a state transition, as defined by $\mathcal{T}$. Subsequently, the agent receives a reward or punishment[3], r_t, defined by $\mathcal{R}$ and the new state s_{t+1}. This is repeated until the horizon is reached, i.e. $t = h$. MDPs are designed to have the Markov property, which ensures that the next state is independent of the previous states given the current state, i.e., knowing the current state is sufficient to reason about the future. Figure 1 shows the dynamics of an MDP.

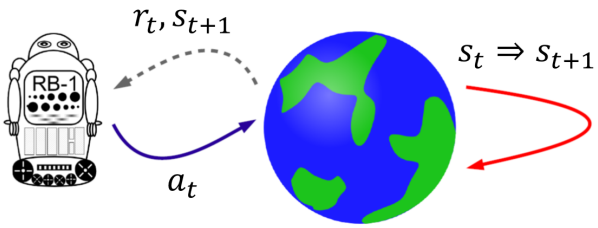


Figure 1: A schematic representation of an MDP adapted from [21]. At time t, the agent "RB-1" takes an action a_t, which initiates the state transition from the current state s_t to the next state s_{t+1}. Subsequently, the agent receives a reward r_t and the next state s_{t+1}.

The objective in a decision making problem is to find a policy π that maximizes or minimizes a measure of the long-run reward received [26]. For example, the expected sum of rewards:

$$E\left[\sum_{t=0}^{h} r_t\right],$$

or, when the horizon is infinite ($h = \infty$), the expected sum of discounted rewards:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

where a discount factor $0 \leq \gamma < 1$ is introduced to make the sum converging. The discount factor γ determines how interesting rewards in the distant future are for the agent [26]. This study only considers problems with a finite horizon.

3. A punishment in this framework is a negative reward.

2.1.2 Partially Observable Markov Decision Processes

POMDPs extend MDPs to partially observable settings. MDPs make the assumption that the sensors of the agent always observe the entire agent's environment, this is however rarely the case. In a more practical situation, the agent's sensors are noisy and/or the agent has a limited number of sensors preventing the agent from observing its entire environment. To model this partial observability, the POMDP framework extends the MDP framework with a set of observations O and an observation function $\mathcal{O} : S \times A \mapsto \Delta(O)$, a mapping from states and actions to a probability distribution over observations.

2.1.3 Partially Observable Stochastic Games

Whereas MDPs model a single agent decision making problem, stochastic games [27, 28] are designed to deal with multiple agents. A stochastic game is an extension of an MDP to a multi-agent setting. The set of actions becomes a set of joint actions, which results in a joint reward function. Introducing partial observability to this stochastic game changes the game to a POSG, i.e., an extension of a POMDP to a multi-agent setting. The agents now only have a partial view of the environment, so the observations need to be tracked and an observation function is required. A POSG is defined as a tuple $\langle D, S, A, \mathcal{T}, O, \mathcal{O}, \mathcal{R}, h, b_0 \rangle$, where:

- $D = \{1, ..., n\}$ is the set of n agents.
- S is a set of states s.
- $A = \times_i A^i$ is a set of joint actions, where A^i is the set of actions a^i available to agent i.
- $\mathcal{T} : S \times A \mapsto \Delta(S)$ is the transition function, a mapping from states and joint actions to probability distributions over states.
- $O = \times_i O^i$ is the set of joint observations, where O^i is the set of observations o^i available to agent i.
- $\mathcal{O} : S \times A \mapsto \Delta(O)$ is the observation function, a mapping from states and joint actions to probability distributions over joint observations.
- $\mathcal{R} = \langle \mathcal{R}^1, ..., \mathcal{R}^n \rangle : S \times A \mapsto \mathbb{R}^n$ is the joint reward function, a mapping from states and joint actions to a vector of real numbers. $\mathcal{R}^i$ is the individual reward function of agent i.
- h and b_0 are defined as in the MDP framework.

Multiple agents also means multiple policies, a collection of policies for all the n agents is called a joint policy $\pi = (\pi^1, ..., \pi^n)$. A policy in a POSG is defined using the notion of action-observation histories. Given an arbitrary agent $i \in D$, the action-observation history of i, $\bar{ao}^i_t$, is the sequence of actions taken by and observations received by agent i at time t:

$$\bar{ao}^i_t = (a^i_0, o^i_1, ..., a^i_{t-1}, o^i_t).$$

Moreover, $\bar{AO}_t^i$ defines the set of all possible action-observation histories at time t for i. The policy π^i of agent i is then defined as the mapping from action-observation histories to distributions over actions:

$$\pi^i : \bar{AO}^i \mapsto \Delta(A^i).$$

Finally, with a slight abuse of notation, $\pi^i(a_t^i \mid \bar{ao}_t^i) : A^i \times \bar{AO}^i \mapsto [0, 1]$ is used to denote the probability of taking the action $a_t^i \in A^i$ given $\bar{ao}_t^i \in \bar{AO}^i$ when following π^i at time t. Moreover, in this study, the rewards are assumed to be observed (agent i observes r^i) and $\bar{h}_t^i$ is used to denote the action-reward-observation history, referred to as history or interaction history:

$$\bar{h}_t = (a_0^i, r_0^i, o_1^i, ..., a_{t-1}^i, r_{t-1}^i, o_t^i).$$

Figure 2 shows the dynamics of a POSG.

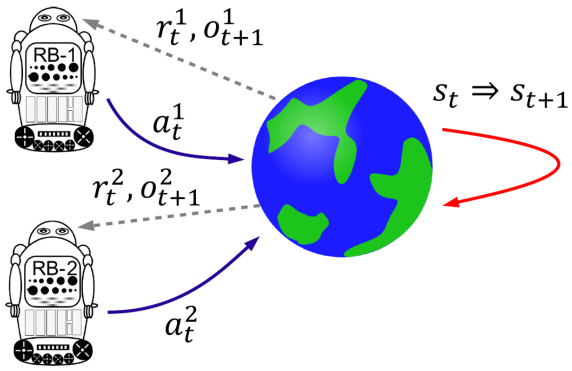


Figure 2: A schematic representation of a POSG with two agents "RB-1" and "RB-2" denoted by 1 and 2 respectively. Adapted from [21]. At time t, agent 1 and 2 both take actions, a_t^1 and a_t^2 respectively, that initiate the state transition from the current state s_t to the next state s_{t+1}. Subsequently the agents receive rewards for their action, r_t^1 and r_t^2 respectively, and observations of the new state, o_{t+1}^1 and o_{t+1}^2 respectively.

This study focuses on both POSGs with an individual reward and POSGs with a team reward called Dec-POMDPs. Dec-POMDPs are POSGs in which all reward functions are the same, so instead of a joint reward function a single reward function is defined: $\mathcal{R} : S \times A \mapsto \mathbb{R}$ [21], i.e., all agents receive the same reward.

2.1.4 Subjective Perspective

The POSG framework models MASs from a bird's eye viewpoint, an objective perspective. However, instead of focusing on solutions that find a joint policy, this study focuses on solutions that take the perspective of a particular agent, called the subjective perspective, and find a single policy for the agent whose perspective is taken in a POSG [21]. Thereby

the assumption is made that the policies of the other agents in the environment are fixed. In the field of game theory this is referred to as finding the best response [29]. Figure 3 shows a schematic representation of the subjective perspective. Note that this almost identical to interactive partially observable Markov decision processes (I-POMDPs) [16, 30, 22, 31], in particular to sub-intentional modelling introduced by Panella and Gmytrasiewicz [17]. However, to avoid confusion this study speaks of POSG from the perspective of a particular agent as this study is not modelling the other agents recursively or as stochastic processes which is the case with intentional and sub-intentional modelling of I-POMDPs respectively.

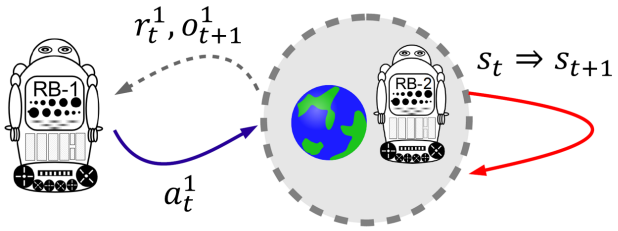


Figure 3: A schematic representation of the subjective perspective with two agents "RB-1" and "RB-2", denoted by 1 and 2 respectively, where the perspective of agent 1 is taken. Adapted from [21].

2.2 Autoencoders

This study uses a variety of autoencoders called variational autoencoders (VAEs) [32, 33] to create representations of other agents. This section covers autoencoders in general in Section 2.2.1, VAEs in Section 2.2.2, and generalized VAEs as well as conditional VAEs (CVAEs) [34, 33] in Section 2.2.3.

2.2.1 Autoencoders

An autoencoder is an artificial neural network that learns a representation of the input data while learning to reconstruct the original input [35]. An autoencoder can be subdivided into two parameterized functions in the form of neural networks, called the encoder and the decoder. In between sits a hidden layer that is a representation of the input. More formally, given input data x, the encoder $e_\theta(x)$ constructs a representation of x, called z, and subsequently the decoder $d_{\theta'}(z)$ produces an approximate reconstruction $\hat{x}$, that is:

$$d_{\theta'}(e_\theta(x)) = \hat{x}.$$

The latent representation $z = e_\theta(x)$ of an autoencoder is restricted to attempt to reduce the dimensionality of the data or to extract useful and essential features out of the data. This can be done by setting the dimension of z smaller than x, in such a case a bottleneck is formed and the autoencoder is called undercomplete. If this is the case and if the decoder is

linear and the loss function is the mean squared error, then the autoencoder learns to span the same subspace as principal component analysis (PCA) [35, 36]. It is therefore the case that autoencoders with non-linear encoders and decoders can learn a more powerful nonlinear generalization of PCA [35]. If the bottleneck is not small enough or if the autoencoder is overcomplete (the dimension of z is greater than or equal to dimension of x) other forms of regularization might be needed. The risk in these cases is that the autoencoder learns to perform the copying task without learning useful information about the distribution of the data or that the autoencoder learns nothing at all.

2.2.2 Variational Autoencoders

VAEs are a variety of autoencoders, which incorporate a prior $p(z)$ over the latent representation into the model. The encoding becomes a latent distribution $p(z \mid x)$ from which latent representations can be sampled. Moreover, VAEs model the problem in a Bayesian way, that is, the assumption is made that the data x is generated from a latent variable z, let $\mathcal{N}$ denote the gaussian distribution then:

Prior:	$z \sim \mathcal{N}(0, I)$, where I is the identity matrix
Likelihood:	$x \mid z \sim \mathcal{N}(f(z), cI)$, where f is a function and $c > 0$ is a hyperparameter.

Modelling the problem this way provides a way to control how the latent representation z represents the input data x.

VAE are called variational because they use variational inference (VI) [37, 36, 38] to infer the (intractable) posterior $p(z \mid x)$, which is, in this context, the encoding and called the latent distribution or approximate prior. VI is chosen over sampling techniques as it is an optimization problem which are in general less precise but faster and easier to scale to large datasets [39, 37]. VI solves the following optimization problem: given an intractable probability distribution p, in this case: $p(z \mid x)$, and a class of tractable distributions Q, in this case:

$\{\mathcal{N}(g(g'(x)), h(g'(x))) \mid g' \in G', g \in G, h \in H\}$ where G', G and H are families of functions,

find the q, in this case: $q(z; x)$, in Q that is most similar to p.

In summary, given input data x, the encoder computes the parameters μ and σ of the latent distribution using $g(g'(x))$ and $h(g'(x))$, which represent the neural network of the encoder. This latent distribution $z \mid x \sim \mathcal{N}(\mu, \sigma)$ is then used to sample a latent representation z which is passed to the decoder. The decoder then produces an approximate reconstruction $\hat{x}$ using $f(z)$, which represents the neural network of the decoder. The complete architecture of the VAE is depicted in Figure 4.

The goal of a VAE is to minimize the difference between the approximate posterior $q(z; x)$ and the actual posterior $p(z|x)$, which can be achieved by taking the Kullback-Leibler divergence[4] (KL divergence) D_{KL} as a loss:

$$L(g', g, h) = D_{KL}(p(z|x) \parallel q(z; x)). \tag{1}$$

4. Given an approximation q and the true distribution p, the KL divergence, or relative entropy, is the average number of extra bits needed to encode the data when q is used instead of p [40]. It is defined as follows: $D_{KL}(p(x) \parallel q(x)) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) dx = \mathbb{E}_p[\log\left(\frac{p(x)}{q(x)}\right)]$. The KL divergence is asymmetric, in general: $D_{KL}(p(x) \parallel q(x)) \neq D_{KL}(q(x) \parallel p(x))$.

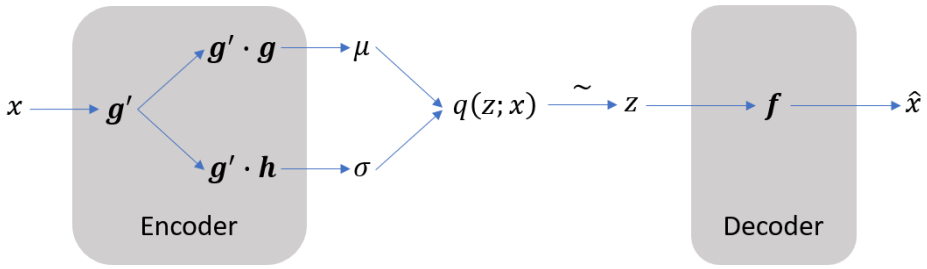


Figure 4: The architecture of a VAE. The data x is fed into the encoder which computes the parameters σ and μ of the approximate posterior. Subsequently, the approximate posterior is used to draw a representation z that is fed into the decoder that approximates x. The encoder and the decoder both represent neural networks, where g', g and h are introduced for mathematical reasons.

However, as $p(z|x)$ is assumed to be intractable, taking expectations with respect to $p(z|x)$ in $D_{KL}(p(z|x) \parallel q(z;x))$ is intractable as well [41]. Therefore, a natural alternative, the reverse KL divergence, is chosen instead:

$$L(g', g, h) = D_{KL}(q(z;x) \parallel p(z|x)). \tag{2}$$

Moreover, taking $D_{KL}(p(z|x) \parallel q(z;x))$ as a loss is called expectation propagation [36] which is a different Bayesian inference technique that is in general more computationally expensive than VI.

Nonetheless, minimizing $D_{KL}(q(z;x) \parallel p(z|x))$ is still is intractable as marginalizing out z to compute $p(x)$ is intractable which is required for the computation [37, 36]. Instead, the evidence lower bound $\mathbb{E}_{z\sim q(z;x)}[\log p(x|z)] - D_{KL}(q(z;x) \parallel p(z))$ is maximized, since:

$$g'^*, g^*, h^* = \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmin}} D_{KL}(q(z;x) \parallel p(z|x)) \tag{3}$$

$$= \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmax}} \left(\mathbb{E}_{z\sim q(z;x)}[\log p(x|z)] - D_{KL}(q(z;x) \parallel p(z))\right). \tag{4}$$

See Appendix A for the derivation of the evidence lower bound. Moreover, as f is unknown and needed to compute $p(x|z)$, the loss function becomes[5]:

$$L(g', g, h, f) = -\left(\mathbb{E}_{z\sim q(z;x)}[\log p(x|z)] - D_{KL}(q(z;x) \parallel p(z))\right). \tag{5}$$

The resulting loss function can be split into two parts, a reconstruction term, $\mathbb{E}_{z\sim q(z;x)}[\log p(x|z)]$, that makes the encoding-decoding scheme efficient and a regularization term, $D_{KL}(q(z;x) \parallel p(z))$, which regularizes the latent space. Moreover, $\mathbb{E}_{z\sim q(z;x)}[\log p(x|z)]$ is the expectation that $\hat{x} = x$ given x and $D_{KL}(q(z;x) \parallel p(z))$ is the difference between the learned distribution over the representations and the prior over the representations.

5. Note that the sign flipped because $L(g', g, h, f)$ refers to a loss function and is therefore minimized.

Finally, as the sampling from the latent distribution is not differentiable but required to back-propagate the errors, a reparameterization trick is used [32]. The reparameterization trick makes use of the fact that the family of normal distributions is closed under linear transformations. Instead of sampling z directly from the posterior $p(z \mid x)$, an auxiliary noise variable $\epsilon \sim \mathcal{N}(0, I)$ is used as an input and removes the stochasticity from the differentiation problem:

$$\begin{aligned} z &\sim \mathcal{N}(\mu, \sigma^2) \\ &\sim \mathcal{N}(\sigma * 0 + \mu, \sigma^2 * I) \\ &= \mu + \sigma\epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, I). \qquad \text{(closed under linear transformations)} \end{aligned}$$

2.2.3 Generalized & Conditional Variational Autoencoders

VAEs can be generalized to "reconstruct" or rather predict a different variable than x. VAEs use variational inference to infer the intractable posterior $p(z|x)$. In a regular VAE the approximation $q(z; x)$ is a Gaussian parameterized on the input data x, this is however not strictly necessary [33, 37]. The approximate distribution q is parameterized on x because the goal of VAEs is to reconstruct the input. If the output (target) of the decoder is changed from x to y the underlying problem changes, instead of x the variable of interest now becomes y, i.e., the likelihood and posterior change to $p(y|z)$ and $p(z|y)$ respectively. The approximation $q(z; x)$ stays the same, it still maps x to z, it is however no longer evident, if the size of the bottleneck is disregarded, that z contains enough information to "reconstruct" or rather predict y. The only thing that changes in the loss function is the likelihood, $p(y|z)$ is used instead of $p(x|z)$:

$$L(g', g, h, f) = -\left(\mathbb{E}_{z\sim q(z;x)}\left[\log p(y|z)\right] - D_{KL}(q(z;x) \parallel p(z))\right). \tag{6}$$

Because this generalization of a VAE is no longer reconstructing the input data, the term VAE may cause confusion. That is why this study refers to this generalization as a VAE based architecture.

Conditional Variational Autoencoders (CVAEs) influence the latent representation z by conditioning the VAE on an extra input b. The prior, likelihood and posterior are all conditioned on b [33, 34], i.e., the prior, likelihood and posterior change to $p(z|b)$, $p(x|z, b)$ and $p(z|x, b)$ respectively. The CVAE no longer has to rely on z alone to reconstruct x, therefore z is likely to encode other information than contained in the extra input b. Take, for example, images of differently coloured cats and dogs as an input. A representation generated by a regular VAE will most likely contain the species and the colour of the animal as features, however if this VAE is conditioned on the species, it will probably focus on the colour instead. A generalized version of a CVAE is similar to a generalized VAE. The approximation $q(z; x)$ stays the same and still maps x to z, but the actual probabilities used in the loss function are now conditioned on b:

$$L(g', g, h, f) = -\left(\mathbb{E}_{z\sim q(z;x)}\left[\log p(y|z, b)\right] - D_{KL}(q(z;x) \parallel p(z|b))\right). \tag{7}$$

Figure 5 shows the architecture of a generalized (conditional) VAE.

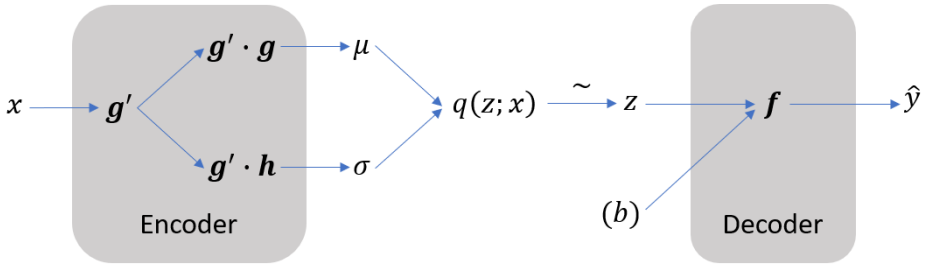


Figure 5: A generalized (conditional) VAE. The data x is fed into the encoder which computes the parameters σ and μ of the approximate posterior $q(z;x)$. Subsequently, the approximation is used to draw a representation z that is fed (together with b in case of a CVAE) into the decoder that approximates y. The encoder and the decoder both represent neural networks, where g', g and h are introduced for mathematical reasons.

2.3 Reinforcement Learning

This study uses reinforcement learning to learn the policy of the agent whose perspective is taken and whose observations are augmented with representations related to other agents. Moreover, this section defines value functions (Section 2.3.1), policy gradient methods (Section 2.3.2), and finally, actor-critic methods (Section 2.3.3), which are the type of reinforcement learning algorithms used in this study.

2.3.1 Value Functions

In reinforcement learning, the performance of a policy π is quantified by a value. Functions that derive such a value are called value functions. This section will cover 3 different value functions: the state value function or value function $V^\pi(s_t, \bar{a}o_t)$, the state-action value function or Q-function $Q^\pi(s_t, \bar{a}o_t, a_t)$ and the advantage function $A^\pi(s_t, \bar{a}o_t, a_t)$. The first of the three, the state value function, is the expected cumulative reward and defined as follows:

$$V^\pi(s_t, \bar{a}o_t) = \mathbb{E}\left[\sum_{k=0}^{h-1} R(s_{t+k}, a_{t+k}) \mid s_t, \bar{a}o_t, \pi\right],$$

which can be expanded into a recursive relation:

$$V^\pi(s_t, \bar{a}o_t) = \sum_{a_t \in A} \pi(a_t \mid \bar{a}o_t) \left(R(s_t, a_t) + \sum_{s_{t+1}, o_{t+1} \in S \times O} Pr(s_{t+1}, o_{t+1} \mid s_t, a_t, \bar{a}o_t) V^\pi(s_{t+1}, \bar{a}o_{t+1}) \right),$$

where for $t = h - 1$:

$$V^\pi(s_{h-1}, \bar{a}o_{h-1}) = \sum_{a_{h-1} \in A} \pi(a_{h-1} \mid \bar{a}o_{h-1}) R(s_{h-1}, a_{h-1}).$$

The actual value is computed using the initial state distribution:

$$V(\pi) = \sum_{s_0 \in S} b_0(s_0) V^{\pi}(s_0, \bar{ao}_0),$$

where $\bar{ao}_0$ can be ignored as it is empty. The second metric, the state-action value function, is the sum of the immediate reward and the expected future cumulative reward:

$$Q^{\pi}(s_t, \bar{ao}_t, a_t) = R(s_t, a_t) + \sum_{s_{t+1}, o_{t+1} \in S \times O} Pr(s_{t+1}, o_{t+1} \mid s_t, a_t, \bar{ao}_t) V^{\pi}(s_{t+1}, \bar{ao}_{t+1}).$$

Finally, the third metric, the advantage function, is the difference between the two and can be rewritten into the temporal difference:

$$\begin{aligned} A^{\pi}(s_t, \bar{ao}_t, a_t) &= Q^{\pi}(s_t, \bar{ao}_t, a_t) - V^{\pi}(s_t, \bar{ao}_t) \\ &= R(s_t, a_t) + \sum_{s_{t+1}, o_{t+1}} Pr(s_{t+1}, o_{t+1} \mid s_t, a_t, \bar{ao}_t) V^{\pi}(s_{t+1}, \bar{ao}_{t+1}) - V^{\pi}(s_t, \bar{ao}_t). \end{aligned}$$

The advantage function measures whether or not the action a_t is better or worse than the action resulting from following the current policy and is a popular choice as it yields almost the lowest possible variance [42].

The value functions could also be defined from the perspective of a single agent but these definitions are omitted because the functions are approximated with neural networks and the definitions are not required for the derivations in this study.

2.3.2 Policy Gradient Methods

In reinforcement learning a distinction can be made between action-value methods and policy gradient methods. Action-value methods learn a value function and then select actions based on the values generated by this function. Policy gradient methods learn a parameterized policy, $\pi(a_t \mid \bar{ao}_t\ ; \theta)$, that is able to select actions without the need of a value function. Note that in the latter category, a value function can still be used to learn the parameters of the policy. The methods used in this study are all of the second category. Policy gradient methods update the parameters of the policy, θ, by approximating the gradient of an objective function J:

$$\theta_{new} = \theta + \eta \widehat{\nabla J(\theta)},$$

where η is the learning rate that controls the size of the step. The expectation of $\widehat{\nabla J(\theta)}$, which is a stochastic estimate, approximates the gradient of the objective with respect to θ [43]. In policy gradient ascent, it is common to take a value function as the objective function.

2.3.3 Actor-Critic Methods

A subcategory of policy gradient methods are actor-critic methods. Actor-critic methods are policy gradient methods that not only learn an approximate policy, called the "actor", but also learn an approximate value function, called the "critic", in an online manner. They combine action-value methods and policy gradient methods. Moreover, the online update of the critic reduces the variance [43, 44]. The algorithms used in this study are all actor-critic methods.

This study uses two actor critic reinforcement learning algorithms, advantage actor critic (A2C) [45] and policy optimization algorithm (PPO) [46]. They are chosen because they are used in [19] and [20] respectively. The remainder of this section discusses the algorithms in detail and defines their corresponding objectives. Moreover, the action observation history $\bar{ao}_t$ is dropped from the value function notation to keep the notation clean. A2C is an actor critic method where the actor is a version of the classical REINFORCE algorithm [47] that includes an estimate of the value function as a baseline. The actor and critic parameters are denoted by θ^A and θ^C respectively. The REINFORCE algorithm is derived as follows. Let $J(\theta^A) = V(\pi; \theta^A)$, then:

$$\nabla_{\theta^A} V(\pi;\theta^A) \propto \sum_{s\in S}\mu(s)\sum_{a\in A} Q^\pi(s_t,a)\nabla_{\theta^A}\pi(a \mid s_t;\theta^A) \quad \text{(by the policy gradient theorem}^6\text{)} \tag{8}$$

$$= \sum_{s\in S}\mu(s)\sum_{a\in A}\pi(a \mid s_t;\theta^A)Q^\pi(s_t,a)\frac{\nabla_{\theta^A}\pi(a \mid s_t;\theta^A)}{\pi(a \mid s_t;\theta^A)} \quad \text{(multiply and divide by } \pi(a \mid s_t;\theta^A)\text{)} \tag{9}$$

$$= \mathbb{E}_\pi\left[Q^\pi(s_t,a_t)\frac{\nabla_{\theta^A}\pi(a_t \mid s_t;\theta^A)}{\pi(a_t \mid s_t;\theta^A)}\right] \tag{10}$$

$$= \mathbb{E}_\pi\left[Q^\pi(s_t,a_t)\nabla_{\theta^A}\log\pi(a_t \mid s_t;\theta^A)\right], \quad \text{(by the identity } \nabla\log x = \frac{\nabla x}{x}\text{)} \tag{11}$$

where $\mu(s) = \frac{\eta(s)}{\sum_{s'}\eta(s')}$ denotes the state distribution, $\eta(s)$ denotes the number of visits of s and $\mathbb{E}_\pi$ denotes $\mathbb{E}_{s_t\sim\mu(s),a_t\sim\pi(a|s_t;\,\theta^A)}$. This equation can be further reduced to the objective function of the REINFORCE algorithm using the equality: $Q^\pi(s_t,a_t) = \mathbb{E}\left[\sum_{k=0}^{h-1} R(s_t,a_t) \mid s_t,a_t\right]$. A2C improves upon REINFORCE by adding a baseline which keeps the expectation the same but reduces the variance. Derivation wise, this is achieved by generalizing the policy gradient theorem to include a baseline $b(s_t)$:

$$\nabla_{\theta^A} V(\pi;\theta^A) \propto \sum_{s\in S}\mu(s)\sum_{a\in A}(Q^\pi(s_t,a) - b(s_t))\nabla_{\theta^A}\pi(a \mid s_t;\theta^A) \tag{12}$$

$$= \mathbb{E}_\pi\left[(Q^\pi(s_t,a_t) - b(s_t))\nabla_{\theta^A}\log\pi(a_t \mid s_t;\theta^A)\right]. \quad \text{(using the save derivation as above)} \tag{13}$$

6. policy gradient theorem [43, Chapter 13]

A2C uses the state value as a base line: $b(s_t) = V^\pi(s_t)$. That is:

$$\begin{aligned}\nabla_{\theta^A} V(\pi;\theta^A) \propto &\quad \mathbb{E}_\pi\left[(Q^\pi(s_t,a_t) - V^\pi(s_t))\nabla_{\theta^A}\log\pi(a_t \mid s_t;\,\theta^A)\right] &(14)\\ = &\quad \mathbb{E}_\pi\left[A^\pi(s_t,a_t)\nabla_{\theta^A}\log\pi(a_t \mid s_t;\,\theta^A)\right]. \qquad (A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t)) &(15)\end{aligned}$$

Next, the advantage function $A^\pi(s_t,a_t)$ is estimated using a truncated version of generalized advantage estimation (GAE) [42]: $\hat{A}^\pi(s_t,a_t;\theta^C) = \sum_{l=0}^{m}(\gamma\lambda)^l\delta_{t+l}$, where $\delta_t = r_t + \gamma\hat{V}^\pi(s_{t+1};\theta^C) - \hat{V}^\pi(s_t;\theta^C)$ is called the temporal difference error, γ is the discount factor (1 in this study), λ is a GAE hyperparameter and m is the (mini)batch size. The critic $\hat{V}^\pi(s_t;\theta^C)$ is used as an approximation for the value function. Hence:

$$\begin{aligned}\mathbb{E}_\pi\left[A^\pi(s_t,a_t)\nabla_{\theta^A}\log\pi(a_t \mid s_t;\,\theta^A)\right] \approx &\quad \mathbb{E}_\pi\left[\hat{A}^\pi(s_t,a_t;\theta^C)\nabla_{\theta^A}\log\pi(a_t \mid s_t;\,\theta^A)\right] &(16)\\ = &\quad \nabla_{\theta^A}\mathbb{E}_\pi\left[\log\pi(a_t \mid s_t;\,\theta^A)\hat{A}^\pi(s_t,a_t;\theta^C)\right]. &(17)\end{aligned}$$

Resulting in the following objective function:

$$J(\theta^A)^{Actor} = \quad \mathbb{E}_\pi\left[\log\pi(a_t \mid s_t;\,\theta^A)\hat{A}^\pi(s_t,a_t;\theta^C)\right], \tag{18}$$

which is the actor objective in the A2C algorithm. An entropy regularization term is added to this objective to improve exploration by discouraging early convergence to sub-optimal deterministic policies [45]:

$$\mathbb{E}_\pi\left[\log\pi(a_t \mid s_t;\,\theta^A)\hat{A}^\pi(s_t,a_t;\theta^C) + \beta_2 * H(\pi(a_t \mid s_t;\,\theta^A))\right], \tag{19}$$

where H denotes the entropy[7] and β_2 denotes a hyperparameter coefficient that is used to tune the regularization term. Moreover, the critic is trained by maximizing the following objective[8]:

$$J(\theta^C)^{Critic} = \quad -\mathbb{E}_\pi\left[(\hat{V}^\pi(s_t;\theta^C) - V^\pi(s_t))^2\right], \tag{20}$$

where $\hat{V}^\pi(s_t;\theta^C)$ is the value function that is learned and $V^\pi(s_t)$ is the target value function calculated using the received rewards. If the neural network architecture shares parameters between the policy and value function, $\theta^A = \theta^C$, then the objectives are combined in one objective function:

$$J(\theta)^{A2C} = \quad \mathbb{E}_\pi\left[J(\theta)^{Actor} + \beta_1 * J(\theta)^{Critic} + \beta_2 * H(\pi(a_t \mid s_t;\,\theta))\right], \tag{21}$$

where both β_1 and β_2 are hyperparameters.

Another actor critic method is PPO. In contrast to standard policy gradient methods, PPO enables multiple epochs of minibatch updates. PPO has its origins in trust region

7. $H(p(x)) = -\int_x p(x)\log p(x)dx$
8. Note the minus sign in front of the expectation.

policy optimization (TPRO) [48] which is based on the idea of trust regions methods [49, Chapter 4] instead of line search methods such as gradient ascent. Instead of generating a search direction, trust region methods define a region which they trust and maximize over, i.e., they choose the direction and length of the step simultaneously. In TPRO this method is used to ensure that the new policy is not too far away from the old policy, thereby reducing the variance in the policy. Let $\hat{A}_t(\theta^C) = \hat{A}^\pi(s_t, a_t; \theta^C)$. TPRO solves the following optimization problem:

$$\underset{\theta^A}{\text{maximize}} \quad \mathbb{E}_\pi \left[\frac{\pi(a_t \mid s_t;\, \theta^A)}{\pi(a_t \mid s_t;\, \theta^A_{old})} \hat{A}_t(\theta^C) \right] \tag{22}$$

$$\text{subject to} \quad \mathbb{E}_\pi[D_{KL}(\pi(\cdot \mid s_t;\, \theta^A_{old}) \parallel \pi(\cdot \mid s_t;\, \theta^A))] \leq \delta, \tag{23}$$

where θ^A_{old} is the vector of policy parameters of the old policy and δ is the size of the trust region. The difference between the new policy and the old policy is controlled by the KL divergence between the old and the new policy, $D_{KL}(\pi(\cdot \mid s_t;\, \theta^A_{old}) \parallel \pi(\cdot \mid s_t;\, \theta^A))$[9], which quantifies the difference between the two distributions. Let $r_t(\theta^A) = \frac{\pi(a_t|s_t;\, \theta^A)}{\pi(a_t|s_t;\, \theta^A_{old})}$. Schulman et al. [46] rewrite this constrained optimization problem into an unconstrained optimization problem by means of a penalty which is controlled by a coefficient β:

$$\underset{\theta^A}{\text{maximize}} \quad \mathbb{E}_\pi \left[r_t(\theta^A)\hat{A}_t(\theta^C) - \beta * D_{KL}(\pi(\cdot \mid s_t;\, \theta^A_{old}) \parallel \pi(\cdot \mid s_t;\, \theta^A)) \right]. \tag{24}$$

The downside of this unconstrained version is that it is hard to choose a single value of β that has a good performance in general. They overcome this problem by clipping the objective instead of using a constraint or a penalty:

$$J(\theta^A)^{Clip} = \quad \mathbb{E}_\pi \left[\min \left(r_t(\theta^A)\hat{A}_t(\theta^C), \text{clip}(r_t(\theta^A), 1-\epsilon, 1+\epsilon)\hat{A}_t(\theta^C) \right) \right], \tag{25}$$

where ϵ is a hyperparameter and the clip function[10] ensures that $r_t(\theta^A) \in [1-\epsilon, 1+\epsilon]$. ϵ limits the incentive for moving $\frac{\pi(a_t|s_t;\, \theta^A)}{\pi(a_t|s_t;\, \theta^A_{old})}$ outside the interval $[1-\epsilon, 1+\epsilon]$, thereby, in a way, limits the trust region avoiding excessively large policy updates. PPO uses $J(\theta^A)^{Clip}$ as its actor objective. The critic objective and the entropy regularisation term are the same as in A2C. If the neural network architecture shares parameters between the policy and value function, $\theta^A = \theta^C$, then the objectives are combined in one objective function:

$$J(\theta)^{PPO} = \quad \mathbb{E}_\pi \left[J(\theta)^{Clip} + \beta_1 * J(\theta)^{Critic} + \beta_2 * H(\pi(a_t \mid s_t;\, \theta)) \right], \tag{26}$$

where both β_1 and β_2 are hyperparameters.

9. $D_{KL}(p(x) \parallel q(x)) = \int_x p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$
10. $\text{clip}(x, lowerbound, upperbound) = \max(\min(x, upperbound), lowerbound)$

3. Approach & Implementation

The approach taken in this study differs from the approach that is taken in most of the related work. Instead of learning agent models alongside solving the RL problem using delayed information, this study separates the modelling from the RL problem. It is a two-stage process, first the agent model is learned in a pre-training step and subsequently the RL problem is solved:

Stage 1: Learn the Model

1. Collect observations, actions and rewards by acting randomly
2. Train the VAE
3. Evaluate the learned posterior

Stage 2: Solve the RL problem using deep RL (every step)

1. Predict the posterior using the encoder of the trained VAE
2. Augment the observation using the approximate posterior
3. Train the RL agent using the augmented observation

This setup provides a better view on the potential of modelling agents with VAE based structures as the VAE can be trained on large set of data which can be controlled. Moreover, instead of updating the VAE every step in the environment, the VAE can now be trained on larger batches which significantly speeds up the process.

This section explains the two stages in detail, Section 2 explains how agents are modelled including the details of the neural network architectures, Section 3.2 explains how the RL problem is solved using the learned models and, finally, Section 3.3 gives a high-level overview of the corresponding software implementations.

3.1 Learning Models of Other Agents

In this study, agent models are defined as functions m that take as an input the observed interaction history $\bar{h}^i$ of the agent whose perspective is taken, agent i, and return a prediction/representation of a feature of interest ψ related to the agent that is modelled, agent j, i.e., $m(\bar{h}^i) = \hat{\psi}$. Examples of features of interest are the action of the modelled agent, its policy[11] or its goal. Figure 6 shows a general agent model.

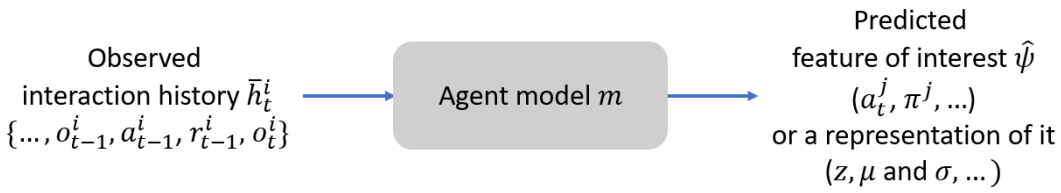


Figure 6: General agent model adapted from [15].

11. Section 4 gives a detailed explanation of how the policy of the modelled agent is used as a feature of interest in this study.

The agent models in this study are created using the encoder of a VAE based neural network architecture and predict a representation of a feature of interest. In specific, at time t, agent i tries to predict a representation z of a particular feature ψ of agent j using its observed interaction history $\bar{h}_t^i$. The VAE based neural network architecture that is used to train the encoder is very similar to a regular VAE, however, instead of reconstructing the input $x = \bar{h}^i$, a target y is predicted, see Figure 7. Section 2.2.3 gives a detailed explanation of this generalization and the consequences for the loss function. This section will not go into the details of the loss function. Moreover, the exact loss function used in the experiments is specified explicitly in Section 4.4. The target y is one-hot encoded or normalized and is specified explicitly for each experiment in Section 4. The VAE based neural network architecture used in this study is "unconditional" and learns a representation of y. So, in order to learn a representation of the feature of interest, the target is set equal to the feature of interest, i.e., $y = \psi$. For example, if the feature of interest is the action of agent j, $\psi = a_t^j$, then the target y is a one-hot encoding of a_t^j. This approach is used in [19] (in case of conditional VAEs, as used in [18], the feature of interest is no longer equal to the target, i.e., $y \neq \psi$). The agent models created using the encoder of a VAE based neural network architecture are compared to an agent model that directly tries to predict the feature of interest. That is, at time t, agent i tries to predict a particular feature ψ of agent j using its observed interaction history $\bar{h}_t^i$.

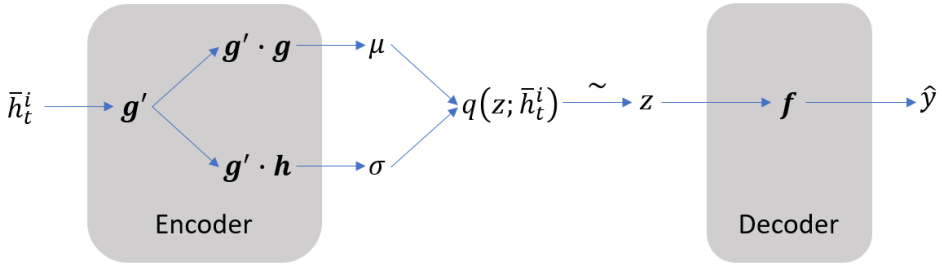


Figure 7: The VAE based architecture used to model other agents. The architecture is a generalized VAE as explained in Section 2.2.3. The architecture takes as an input the history $\bar{h}^i$ of agent i whose perspective is taken and the target y is set equal to a feature of interest related to the modelled agent j.

The encoder of the VAE based neural network architecture is depicted in Figure 8. The input of the encoder, $\bar{h}_t^i$, is a collection of vectors to which a concatenation of a one-hot encoded action, a scaled reward and a one-hot encoded observation is added each time step. This input is fed into a long short-term memory neural network (LSTM) which has an intermediate dimension of 16 and uses the rectified linear unit activation function[12] (ReLu). The output is subsequently fed into two single layer feed forward neural networks (FNNs) which output the parameters μ and σ of the approximate posterior $q(z; \bar{h}_t^i)$. The dimension of these outputs equals the "latent dimension" which is the dimension of z (a sample of $q(z; \bar{h}_t^i)$). In this study the dimension of z is set to 2, $|z| = 2$, unless otherwise mentioned.

12. $f(x) = max(0, x)$

This makes it possible to visualize the latent space by sampling from the approximate posterior and subsequently plot the sampled representations in a 2d plot where an axis represents one of the two dimensions.

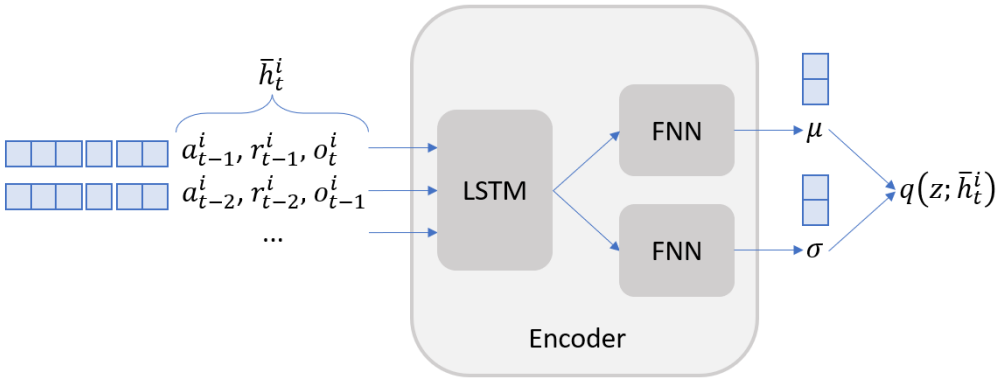


Figure 8: The encoder used to model other agents. The encoder takes as an input the history $\bar{h}_t^i$ of agent i whose perspective is taken. This history, which is one-hot encoded and normalized depending on the feature, is fed into an LSTM which subsequently feds the output into two FNNs that output the parameters μ and σ of the approximate posterior $q(z; \bar{h}_t^i)$. The blue boxes indicate the sizes of the vectors to give an idea of what is going in and out, here $|A| = 3$, $r \in \mathbb{R}$, $|O| = 2$ and $|z| = 2$.

The decoder of the VAE based neural network architecture is depicted in Figure 9. The input of the decoder is a latent representation z sampled from the approximate posterior $q(z; \bar{h}_t^i)$. This input is fed into a FNN which has an intermediate dimension of 16 and uses the ReLu activation function. The output of this FNN is fed into a single FNN layer which uses a softmax[13] or sigmoid[14] activation for one-hot encoded targets and normalized targets respectively. If y consists of multiple features, e.g, $y = \{a_t^j, r_t^j\}$, then multiple single FNN layers are used with action functions corresponding to the desired outputs. In this case the reconstruction loss is computed per feature and averaged.

Finally, Figure 10 shows the architecture that is used to learn the agent model that directly predicts the feature of interest. This architecture is apart from the last layers equal to the encoder. Instead of feeding the output of the LSTM into two FNNs to output the parameters of the approximate posterior, the output of this LSTM is fed into a last layer that is identical to the last layer of the decoder which uses a softmax or sigmoid activation function depending on the target y.

13. $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{|x|} e^{x_j}}$, where x_i and x_j are used to denote a specific entry of x
14. $f(x) = \frac{1}{1+e^{-x}}$

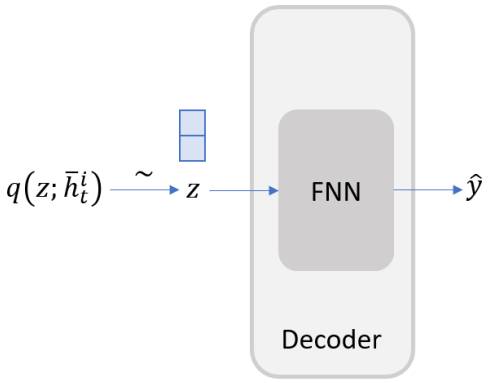


Figure 9: The decoder used to model other agents. A sample z from the approximated posterior $q(z;\bar{h}_t^i)$ is fed into a FNN that predicts the target y.

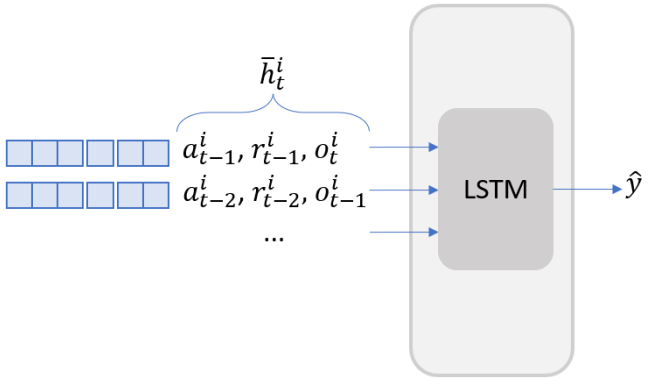


Figure 10: The LSTM classifier used to directly predict a feature of interest.

3.2 Using Agent Models in Reinforcement Learning

The learned agent models are used to make a prediction of a feature of interest which in turn is used to augment the observation that is used to solve the RL problem. Figure 11 depicts this process in a schematic way. Let agent i be the agent whose perspective is taken and agent j be an arbitrary other agent that both have taken actions a_t^i and a_t^j respectively. The augmentation then works as follows. First, agent i expands its interaction history with the newly available information. So $\bar{h}_{t+1}^i = \bar{h}_t^i \cup \{o_{t+1}^i, r_t^i, a_t^i\}$. This history is then fed into the learned model of agent j to get a prediction of a feature of interest $\hat{\psi}$. Subsequently, this prediction is used to augment the observation o_{t+1}^i that is received upon taking action a_t^i. The augmented observation $o_{t+1}'^i = \langle o_{t+1}^i, \hat{\psi} \rangle$ is the result of merging the vector representations of o_{t+1}^i and $\hat{\psi}$. Finally, the augmented observation is used, together with the reward r_t^i corresponding to the action a_t^i that is taken, to solve the RL problem.

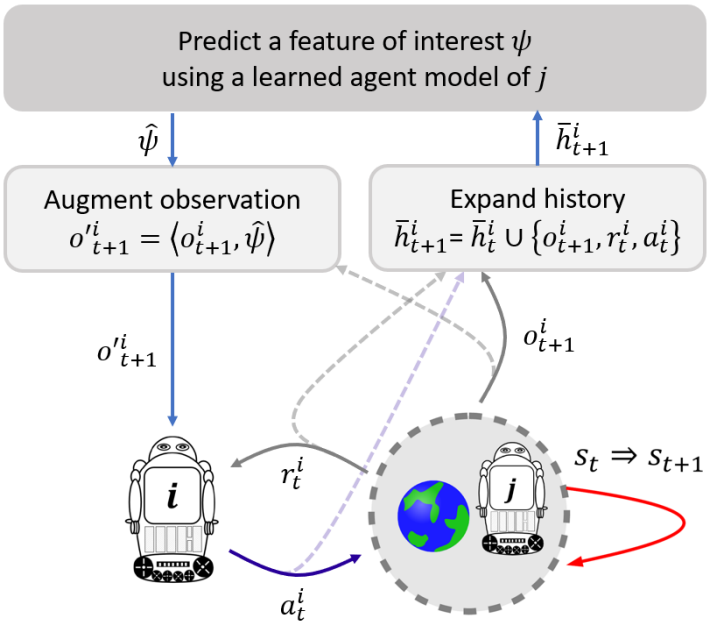


Figure 11: A schematic representation of the augmentation of observations with predictions from an agent model. After taking actions, the history of the agent whose perspective is taken $\bar{h}^i_t$ is expanded with the newly available information: a^i_t, r^i_t and o^i_{t+1}. Subsequently, the expanded history $\bar{h}^i_{t+1}$ is fed into the agent model which predicts some feature of interest ψ related to the other agent j. Finally, the new observation o^i_{t+1} is augmented with the prediction $o'^i_{t+1} = \langle o^i_{t+1}, \hat{\psi} \rangle$ and passed to the agent together with the reward r^i_t corresponding to the action a^i_t that is taken.

This study uses three different agent models to augment observations with. Two of the agent models are created using the encoder of a trained VAE based model, the other agent model is an LSTM classifier. The details of these agent models are explained the previous section, Section 2. The remainder of this section defines the augmented observation spaces used in this study and explains which agent model is used to create them. Let Z be the set of all possible latent representations z sampled from the approximate posterior, let M and Σ be the sets of all possible parameters μ and σ the approximate posterior respectively and let $\hat{Y}$ be the set of all possible predictions of the feature of interest (assuming that the feature of interest is set as a target, i.e., $y = \psi$). The observation space of agent i can then be augmented in the following ways:

1. $O'^i = O^i \times Z$,
2. $O'^i = O^i \times M \times \Sigma$ and
3. $O'^i = O^i \times \hat{Y}$,

which are referred to as augmenting with samples from the approximate posterior, augmenting with parameters from the approximate posterior and augmenting with predictions

respectively. The first two augmented observation spaces are created using the encoder of a trained VAE based model and the third is created using an LSTM classifier. The third augmented observation space could also be created using the predictions made by the decoder of the VAE based model, however the encoding and subsequent decoding would most likely not improve the predictions. Figure 12 shows the three different ways to augment observations using the learned VAE based model/LSTM classifier.

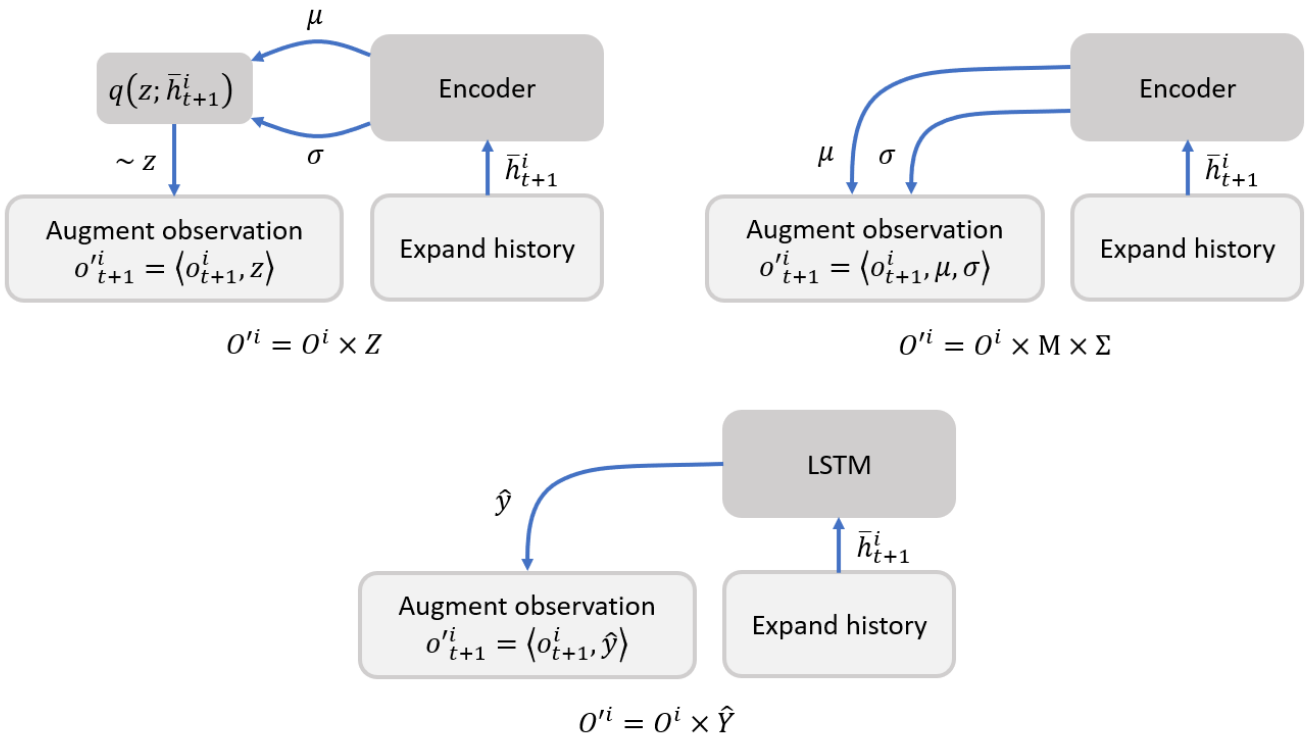


Figure 12: The augmentation of observations with samples from the approximate posterior, parameters from the approximate posterior and predictions. The top left figure shows an agent model that outputs sampled representations of the feature of interest, $o'^i_{t+1} = \langle o^i_{t+1}, z \rangle$. Next, the top right figure shows an agent model that outputs the parameters μ and σ of a distribution over representations of the feature of interest, $o'^i_{t+1} = \langle o^i_{t+1}, \mu, \sigma \rangle$. Finally, the bottom figure shows an agent model that outputs an approximation of the feature of interest, $o'^i_{t+1} = \langle o^i_{t+1}, \hat{y} \rangle$. Here it is assumed that the feature of interest is set as a target, i.e., $y = \psi$.

3.3 Software Implementation

The VAEs used in this study are implemented using Tensorflow [50, 51] and Keras [52]. TensorFlow is an open-source machine learning library for Python and Keras is the high-level API of TensorFlow. The VAE and its components are programmed in an object-oriented way by subclassing Keras Model and keras Layer. The programmed VAE class takes as arguments an encoder layer, a decoder layer, a sampler layer, the reconstruction loss, the regularization loss and the input shapes, which makes the VAE fully customizable.

The reinforcement learning part of this study is implemented using RLlib [53, 54], which is an open-source Python library for reinforcement learning supporting multi-agent setups, vector wise evaluation of environments and a range of implemented algorithms. The augmentation of the observations is achieved by making a wrapper for the RLlib multi-agent environment. This wrapper is initialized with the trained VAEs/neural networks that are used to generate the representations. The wrapper keeps track of the action observation histories of all the agents and augments the observations of the agent whose perspective is taken after taking a step in the wrapped environment. The policies of the other agents are instances of a special policy class that extends a RLlib policy and applies the policies of the other agents batch wise. The policy of the agent whose perspective is taken is trained using a test bench class that stores checkpoints, is able to use a learning rate schedule and stores performance plots as well as other metrics.

4. Experiments & Analysis

This section covers the experiments that are conducted to answer the research questions (Sections 4.5, 4.6 and 4.7) as well as the environment (Section 4.1), policy sets (Section 4.2), settings (Section 4.4) and baselines (Section 4.3) used for the experiments.

The work in this thesis is based on the assumption that given the complexity of solving (deep) RL problems in multi-agent contexts, it is beneficial to exploit models of other agents that are learned off-line during a pre-training phase. The aim of this study is to investigate the usefulness of the augmentation of the observation space with samples and distributions related to other agents generated by VAE based architectures in partially observable settings. In this study, it is assumed that the agent that is being modelled, agent j, acts according to a policy that is randomly selected from a set of policies, Π^j. Moreover, it is assumed that this happens at the start of an episode and that the policy is fixed for the entire episode. All the experiments are conducted with two agents. One of the agents, agent i, is the agent which perspective is taken and one of the agents, agent j, is the agent that is being modelled by agent i.

For the agent modelling, two features are considered: the next action that the modelled agent is going to take, a_t^j, and the index[15] of the policy that the modelled agent is currently using, $index(\pi^j)$, so which policy π^j in the set of policies Π^j agent j is using that episode. Moreover, the experiments are conducted with two different sets of policies Π^j, which are defined in Section 4.2. Here the research questions targeted in this work are summarized:

1. What would be the best feature related to another agent to augment the observations with?

   This question is answered by comparing the episode reward means of a naive RL agent that uses unaugmented observations with RL agents that use observations augmented with a_t^j and $index(\pi^j)$ respectively.

2. To what extent can these features be predicted by a VAE based model and how does this compare to an LSTM classifier?

   To answer this question, the VAE based model is trained with a_t^j and/or $index(\pi^j)$ as target(s) depending on the results from question 1. Subsequently, the losses and corresponding learning curves are evaluated as well as the resulting representations by plotting the latent space. These results are then compared to the losses of LSTM classifiers that are trained on the same targets.

3. What is the best way to leverage the VAE based model to augment the observation space in the RL problem and how does this model compare to an LSTM classifier?

   This final question is answered by comparing the episode reward means of VAE based models and LSTM classifiers learned in question 2 with configurations and settings chosen according to the results obtained in question 1 and 2.

15. Assuming that the set of policies Π^j is an indexed family with index set $\{0, 1, ..., |\Pi^j| - 1\}$, i.e., each policy π^j in Π^j is associated with a unique number.

4.1 Environment

The environment used in this study is called the tiger problem. It is an adapted multi-agent version of the original tiger problem by Kaelbling et al [13], transformed into a Dec-POMDP by Nair et al [14]. In this problem, two agents, $D = \{1, 2\}$, face two doors. Behind one of the doors lies a hungry tiger and behind the other door lies a treasure. This also defines the state of the environment: $S = \{TigerLeft, TigerRight\}$. The exact location of the tiger is unknown, but the agents can get an idea of the location through their observations. The agents observe the environment by either hearing the tiger behind the left door or behind the right door, that is: $O^i = \{HearLeft, HearRight\}$ for all $i \in D$. The initial observations are randomly chosen. Based on these observations, the agents need to individually take an action and decide which door to open or to listen instead. Therefore: $A^i = \{OpenLeft, OpenRight, Listen\}$ for all $i \in D$.

The state of the environment changes according to the actions taken: when both agents listen, the state does not change, i.e., it transitions to the same state, however the state is reset when one of the agents or both agents open a door, see Table 1.

State	Joint Action	P($S = TigerLeft$)	P($S = TigerRight$)
$TigerLeft$	<*Listen*, *Listen*>	1.0	0.0
$TigerRight$	<*Listen*, *Listen*>	0.0	1.0
*	<*, *>	0.5	0.5

Table 1: The transition function, $\mathcal{T} : S \times A \mapsto \Delta(S)$, of the tiger problem.

After the state transitions, the agents receive observations about the new state as defined in Table 2. Here the purpose of the *Listen* action becomes clear as opening a door leaves the agent with a random impression of the state. However, if both agents listen, they observe the correct state with a high probability.

State	Joint Action	P(HL, HL)	P(HL, HR)	P(HR, HL)	P(HR, HR)	P($O^i = O^j$)
$TigerLeft$	<*Listen*, *Listen*>	0.8	0.05	0.05	0.1	0.9
$TigerRight$	<*Listen*, *Listen*>	0.1	0.05	0.05	0.8	0.9
*	<*, *>	0.4	0.1	0.1	0.4	0.8

Table 2: The observation function, $\mathcal{O} : S \times A \mapsto \Delta(O)$, of the tiger problem. Where P(A, B) denotes P($O^i = A$, $O^j = B$), e.g., P(HL, HR) denotes P($O^i = HL$, $O^j = HR$).

Finally, the agent receives a reward based on the state and taken action. As the tiger problem is defined to be a Dec-POMDP, the agents receive a shared reward. Moreover, the reward function, see Table 3, is shaped in such a way that it is beneficial to cooperate. For example, if both agents open a wrong door, that is, the door with the tiger behind it, they get punished less then when they open two different doors.

Action/State	$TigerLeft$	$TigerRight$
$<OpenRight, OpenRight>$	+20	-50
$<OpenLeft, OpenLeft>$	-50	+20
$<OpenRight, OpenLeft>$	-100	-100
$<OpenLeft, OpenRight>$	-100	-100
$<Listen, Listen>$	-2	-2
$<Listen, OpenRight>$	+9	-101
$<OpenRight, Listen>$	+9	-101
$<Listen, OpenLeft>$	-101	+9
$<OpenLeft, Listen>$	-101	+9

Table 3: The reward function, $\mathcal{R} : S \times A \mapsto \mathbb{R}$, of the tiger problem (transposed).

4.2 Policy sets

The policy sets Π^j used for the modelled agent j consist of a combination of two of the following policies:

- *act accordingly*

  The agent that follows this policy acts greedy: it never listens and acts **according** to the received observation, i.e., when the agent observes $HearLeft$ it opens the right door ($OpenRight$).

- *act contrarily*

  The agent that follows this policy acts greedy: it never listens and acts **contrary** to the received observation, i.e., when the agent observes $HearLeft$ it opens the left door ($OpenLeft$).

- *listen and act accordingly*

  The agent that follows this policy listens once and then acts according to the received observation, i.e., it ignores the first observation, takes the listen action and acts according to the second observation received upon listening.

In specific, the experiments are conducted with the following sets:

- $\Pi^j_A = \{act\ accordingly,\ act\ contrarily\}$
- $\Pi^j_B = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$

4.3 Baselines

The RL agents' performance is compared to three baselines:

- always listen

  An agent that always listens, i.e., the agent never opens a door.

- use the same policy

  An agent that uses the same policy as the other agent, i.e., it applies the policy of the other agent on its own observations.

- take the exact same action

  An agent that takes exactly the same action as the other agent.

4.4 Settings

This section covers the settings that are the same for all experiments, it first covers the RL settings and subsequently the VAE settings.

The RL problem experiments are performed for 25000 or 50000 time steps/transitions depending on the policy set. The number of samples equals the number of time steps. The batch size is 1000 (25 or 50 updates with 1000 samples) and the samples are retrieved from 10 environments in parallel. The horizon of the environment is set to 100, so an episode consists of 100 time steps. Moreover, appendix B gives an overview of the number of runs (number of times the experiment is performed) used to produce the figures in Section 4.5 and Section 4.7. For example, if the RL experiment is performed for 50000 time steps (50 batches) and the number of runs is 30 then the average episode reward mean as shown in the plots is calculated by averaging the episode reward means of the first batch of every run, so by averaging 30 episode reward means. This is repeated for the second batch of every run, the third batch of every run, etc. 50 times in total.

The RL agents have been tested with action-observation-reward histories $\bar{h}_t^i = (a_0^i, r_0^i, o_1^i, ..., a_{t-1}^i, r_{t-1}^i, o_t^i)$, action-observation histories $\bar{ao}_t^i = (a_0^i, o_1^i, ..., a_{t-1}^i, o_t^i)$ and observation histories $(o_1^i, ..., o_t^i)$ as input. The agents that used the observation history showed the best and most stable results; therefore, the observation history was chosen to perform the experiments, both for the agents that use unaugmented observations and for the agents that use augmented observations. Moreover, both RL algorithms, A2C and PPO, where tested and the results showed that A2C converged slower and did not always converge to the same reward mean. PPO on the other hand showed fast convergence and stable results; therefore, PPO was chosen to perform the experiments. Moreover, PPO is used with separate parameters for the actor θ_A and the critic θ_C to further stabilize the learning. Both the actor and the critic network consist of an LSTM with one hidden layer of size 128 and a FFN with one hidden layer of size 128. Equation 28 and Equation 27, show the used critic and actor objective respectively. Section 2.3.3 provides a more in-depth analysis of PPO and its corresponding loss functions. PPO is used with a minibatch size of 100, the other hyperparameters can be found in Appendix C. The actor objective function is defined as

follows:

$$J(\theta^A) = \mathbb{E}_\pi \Big[\min \Big(r_t(\theta^A)\hat{A}_t(\theta^C), \text{clip}(r_t(\theta^A), 1-\epsilon, 1+\epsilon)\hat{A}_t(\theta^C) \Big) + \beta_2 * H(\pi^i(a_t \mid \bar{ao}_t^i; \theta^A)) \Big], \tag{27}$$

where $r_t(\theta^A) = \frac{\pi^i(a_t|\bar{ao}_t^i;\, \theta^A)}{\pi^i(a_t|\bar{ao}_t^i;\, \theta^A_{old})}$ and $\hat{A}_t(\theta^C) = \hat{A}^{\pi^i}(s_t, \bar{ao}_t^i, a_t; \theta^C)$. Furthermore, θ^A_{old} is the vector of policy parameters of the old policy and ϵ is a hyperparameter. The clip function[16] ensures that $r_t(\theta^A) \in [1-\epsilon, 1+\epsilon]$. Moreover, H denotes the entropy[17] and β_2 is a hyperparameter used to tune the regularization term. $\mathbb{E}_\pi$ refers to the expectation under π and is defined in detail in Section 2.3.3. Finally, the critic objective function, which is also being maximized[18], is defined as follows:

$$J(\theta^C) = -\mathbb{E}_\pi \Big[(\hat{V}^{\pi^i}(s_t, \bar{ao}_t^i; \theta^C) - V^{\pi^i}(s_t, \bar{ao}_t^i))^2 \Big], \tag{28}$$

where $\hat{V}^{\pi^i}(s_t, \bar{ao}_t^i; \theta^C)$ is the value function that is learned and $V^{\pi^i}(s_t, \bar{ao}_t^i)$ is the target value function calculated using the received rewards.

For the VAE based models 300000 samples are collected by acting randomly, i.e., i acts randomly and j follows a policy that is randomly sampled from Π^j at the beginning of the episode. These samples are split into a train, validation and test set according to the following distribution: [0.6, 0.2, 0.2]. The validation set is used during training to estimate the eventual performance of the model during training. It is also used as an indication for overfitting and for the tuning of hyperparameters. The test set is used to do the final evaluation. The VAE based models are trained for 15 epochs. Equation 29 shows the loss function used to train the VAE based models, which is explained in more detail in Section 2.2.1.

$$L(g', g, h, f) = -\left(\mathbb{E}_{z\sim q(z;x)} \left[\log p(x|z) \right] - D_{KL}(q(z;x) \parallel p(z)) \right). \tag{29}$$

where D_{KL} denotes the KL divergence[19][20], $p(y|z)$ is the likelihood, $q(z; \bar{h}_t^i))$ is the approximate posterior, $p(z)$ is the prior and g', g, h and f refer to the functions/parameters of the encoder and decoder. The second part of the loss, $D_{KL}(q(z; \bar{h}_t^i)) \parallel p(z))$, is the regularization term which regularizes the latent space. The regularization term can be simplified to:

$$D_{KL}(q(z; \bar{h}_t^i)) \parallel p(z)) = -\frac{1}{2}[1 + \log(\sigma^2) - \sigma^2 - \mu^2], \tag{30}$$

as proven by [32]. The first part of the loss, $\mathbb{E}_{z\sim q(z;\bar{h}_t^i)} \left[\log p(y|z) \right]$, is the reconstruction term that makes the encoding-decoding scheme efficient. The computation of this reconstruction term starts by taking one sample z' from $q(z; \bar{h}_t^i)$ which paves the way for the following approximation:

$$\mathbb{E}_{z\sim q(z;\bar{h}_t^i)} \left[\log p(y|z) \right] \approx \log p(y|z'). \tag{31}$$

16. $\text{clip}(x, lowerbound, upperbound) = \max(\min(x, upperbound), lowerbound)$
17. $H(p(x)) = -\int_x p(x) \log p(x) dx$
18. Note the minus sign in front of the expectation.
19. $D_{KL}(p(x) \parallel q(x)) = \int_x p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$
20. Section 2.2.1 explains why the KL divergence is used in this order.

Next, the deterministic decoder f can be used to make a prediction: $f(z') = \hat{y}$. This means that $p(y|z')$ can be seen as $p(y|\hat{y})$. In this study, the target y is one hot encoded, so $p(y \mid \hat{y})$ can be approximated by a multinomial distribution. The target y is a categorical random variable: $y : \Omega \mapsto \{1, ..., k\}$, where k is the number of categories, e.g. if $y = a_t^j$ (one-hot encoded) then $k = 3$. Moreover, as $P(y = j) = 0$ or $P(y = j) = 1$ for all j in $\{1, ..., k\}$, $P(y = j)$ can be seen as "counts" or "occurrences" which in this case is either 0 or 1 and indicates the category. The prediction $\hat{y}$ is a vector of probabilities, that is, $P(\hat{y} = j) \in [0, 1]$ for all $j \in \{1, ..., k\}$. Therefore:

$$\log p(y \mid \hat{y}) = \log \left(\frac{1!}{\prod_{j=1}^{k} P(y = j)!} \prod_{j=1}^{k} P(\hat{y} = j)^{P(y=j)} \right) \tag{32}$$

$$= \log \prod_{j=1}^{k} P(\hat{y} = j)^{P(y=j)} \qquad \text{(as } P(y = j) \in \{0, 1\} \text{ and } 0! = 1\text{)} \tag{33}$$

$$= \sum_{j=1}^{k} P(y = j) \log P(\hat{y} = j) \tag{34}$$

$$= -H(p(y), p(\hat{y})) \tag{35}$$

where $H(p(y), p(\hat{y}))$ denotes the categorical cross-entropy between $p(y)$ and $p(\hat{y})$. Therefore, the reconstruction loss can be computed by sampling z once, decoding z into $\hat{y}$ and computing the categorical cross-entropy between $p(y)$ and $p(\hat{y})$.

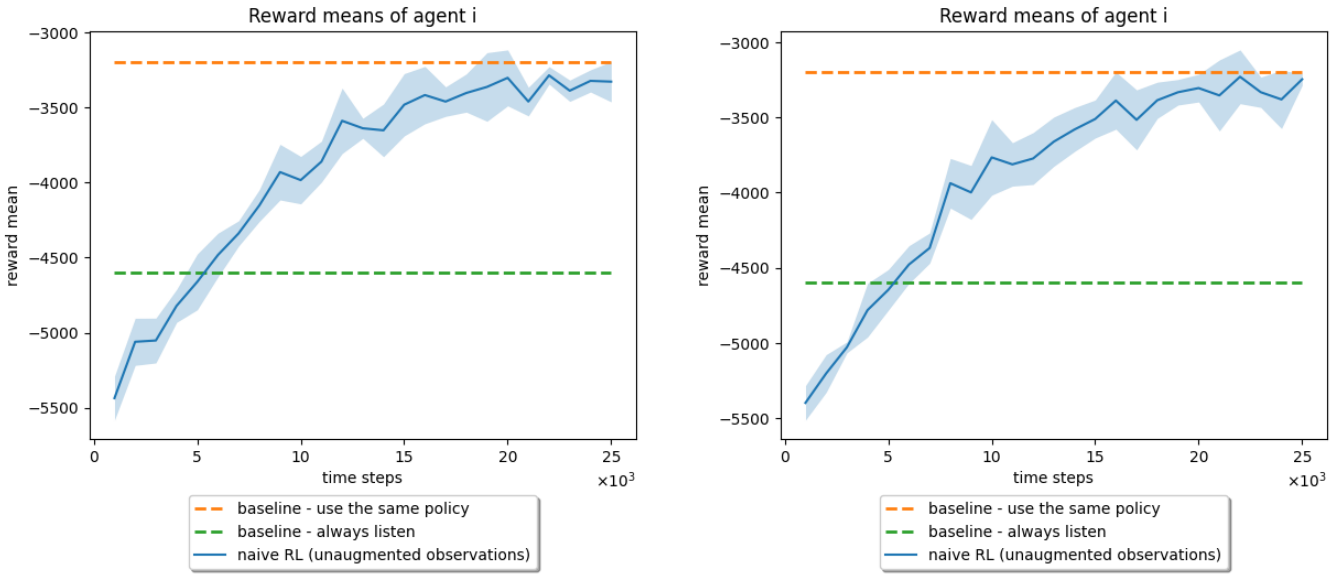
4.5 Augmenting Observations with Features

This section is devoted to answering the first research question:

- What would be the best feature related to another agent to augment the observations with?

The first question is posed to get an idea of what would be the ideal feature to augment observations with. To answer this question, the episode reward means of a naive RL agent that uses unaugmented observations is compared with: (1) a RL agent that uses observations augmented with the index of the policy that the modelled agent is currently using, $index(\pi^j)$ and subsequently with (2) a RL agent that uses observations augmented with the next action that the modelled agent is going to take, a_t^j. Both features are one-hot encoded. First this comparison is made for the greedy policy set $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$, then for the other set $\Pi_B^j = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$.

Before looking at the results for Π_A^j, the results for when Π^j only contains one of the policies, so $\Pi^j = \{act\ accordingly\}$ and $\Pi^j = \{act\ contrarily\}$, are analysed to get an idea of the potential of the RL agent. Figure 13 shows these results. A naive RL agent that uses unaugmented observations settles in both cases at the same reward mean as a baseline that uses the same policy as the other agent. Note that the figures are almost identical. This is a consequence of the greediness of the policies, both policies never listen so in both cases all observations are random. Although both policies settle at the same reward mean, they do act differently based on the same observations. The consequences of this will become clear when both policies are put in one set, Π_A^j.



(a) $\Pi^j = \{act\ accordingly\}$ (b) $\Pi^j = \{act\ contrarily\}$

Figure 13: The average episode reward means of agent i, the agent whose perspective is taken, when set of policies Π^j used for the modelled agent j only contains one of the policies of Π_A^j. The x-axis shows the "epochs" (25 epochs * 100 steps * 10 environments in parallel = 25000 steps). And the y-axis shows the episode reward mean. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL" refers to a RL agent that uses unaugmented observations.

Figure 14 shows the results for Π_A^j. From the figure it becomes clear that when the set contains two different policies, the naive RL is unable to reach the same reward mean as in the individual case and settles at the same reward as a baseline that always takes the listen action. However, if the RL agent uses observations augmented with the index of the policy that the modelled agent is currently using, $index(\pi^j)$, then the agent is able to outperform the always listen baseline. In this case, the reward mean approaches the use the same policy baseline which is the same baseline that is approached in the individual cases. Note that although this is the case, the learning takes longer as the RL agent not only needs to learn the best response but it also needs to learn which policy maps to which index.

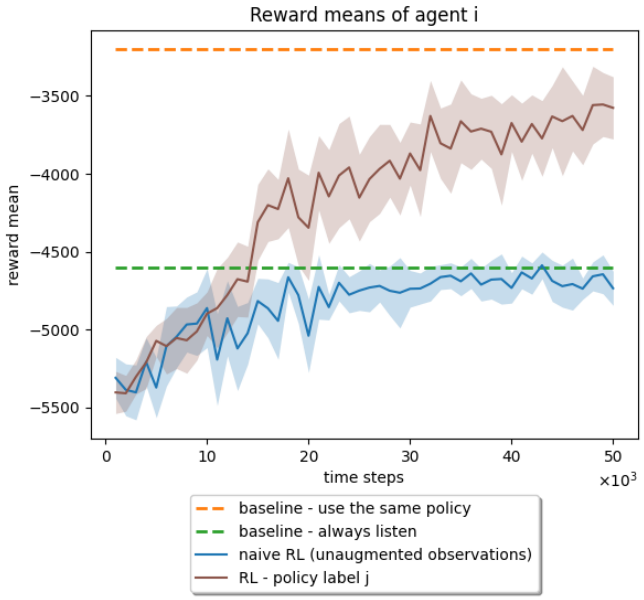


Figure 14: The average episode reward means of agent i, the agent whose perspective is taken, when the modelled agent, agent j, uses policy set $\Pi_A^j = \{act\ accordingly, act\ contrarily\}$. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL" and "RL - policy label j" refer to RL agents that use unaugmented observations and observations augmented with the index of the policy that the modelled agent is currently using, $index(\pi^j)$, respectively.

Next, Figure 15 builds on Figure 13 that displays the results for when Π^j only contains one of the policies, by adding a RL agent that uses observations augmented with the next action that the modelled agent is going to take, a_t^j. The figure for $\Pi^j = \{act\ contrarily\}$ is omitted as the results are almost identical. Here it becomes clear that that knowing the action of the other agent is beneficial for the RL agent. The episode reward mean settles at the same reward mean as a baseline that copies the actions of the other agent and the RL agent learns to take the exact same actions.

Finally, Figure 16 adds the results of a RL agent that uses observations augmented with the next action that the modelled agent is going to take, a_t^j, to Figure 14 which shows the results for Π_A^j. Here it becomes clear that knowing the action of the other agent gives a higher advantage than knowing the index of the policy. The episode reward mean settles at the same reward mean as a baseline that copies the actions of the other agent.

Figure 15: The average episode reward means of agent i, the agent whose perspective is taken, when the set of policies Π^j used for the modelled agent j only contains one policy: $\Pi^j = \{act\ accordingly\}$. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL" and "RL - action j" refer to RL agents that use unaugmented observations and observations augmented with the action of agent j, a_t^j, respectively.

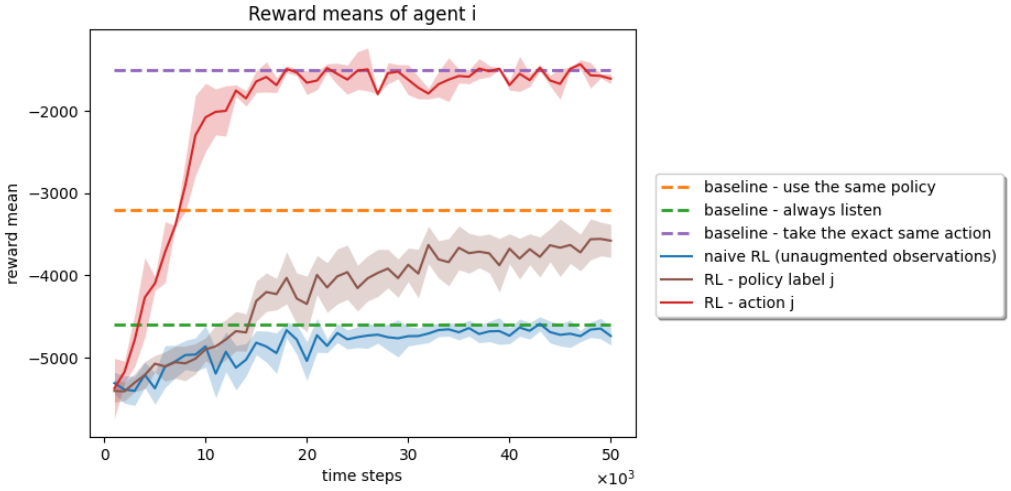
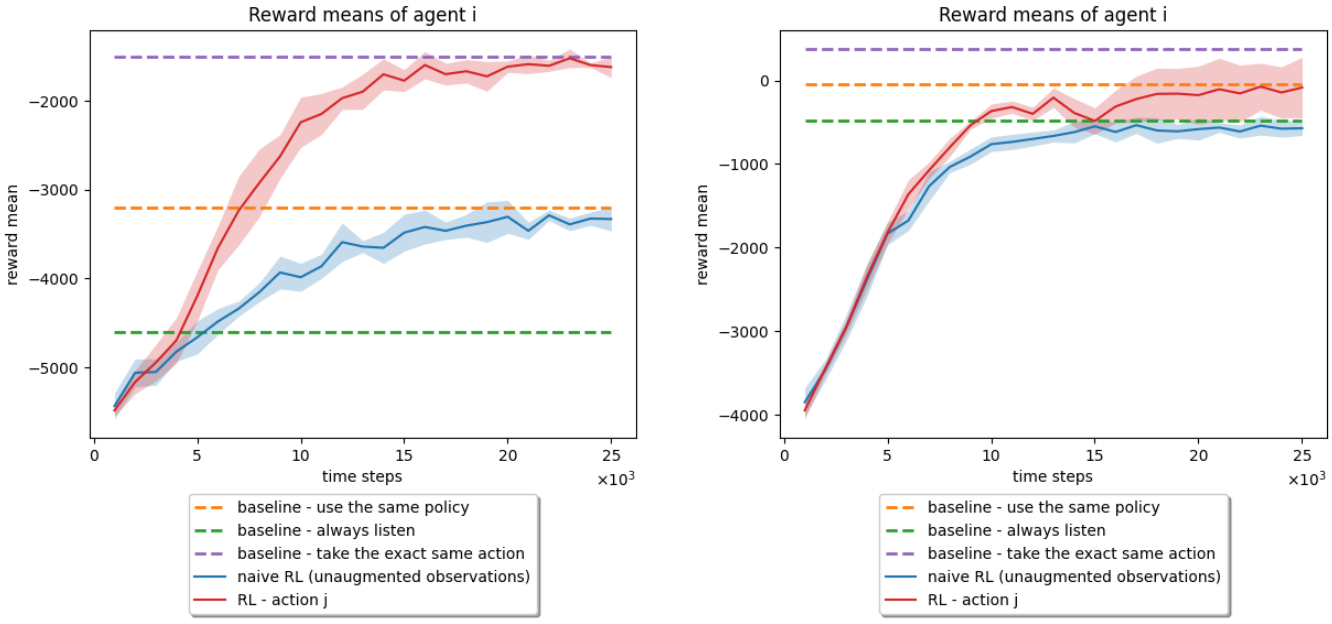


Figure 16: The average episode reward means of agent i for policy set $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$. The shaded aeras depict 95% confidence intervals (1.96 * the standard error). "naive RL", "RL - policy label j" and "RL - action j" refer to RL agents that use unaugmented observations, observations augmented with the index of the policy of agent j, $index(\pi^j)$, and observations augmented with the action of agent j, a_t^j, respectively.

Next, the results for policy set $\Pi^j_B = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$ are covered. Starting again by looking at the individual policies, but this time directly comparing a naive RL agent that uses unaugmented observations with an agent that uses observations augmented with the next action that agent j is going to take, a^j_t. Figure 17b shows these results for $\Pi^j = \{listen\ and\ act\ accordingly\}$ and Figure 17a, which is identical to Figure 15, shows the results for $\Pi^j = \{act\ accordingly\}$ and is shown for comparative purposes. From Figure 17b it becomes clear that for $\Pi^j = \{listen\ and\ act\ accordingly\}$, the naive RL agent is no longer able to get to the same level as a baseline that uses the same policy. Moreover, the RL agent that uses observations augmented with the action that agent j is going to take, a^j_t, is not always settling at the same level of a baseline that takes the exact same action, as is the case with $\Pi^j = \{act\ accordingly\}$. It sometimes gets stuck in the local minimum of always listen.



(a) $\Pi^j = \{act\ accordingly\}$

(b) $\Pi^j = \{listen\ and\ act\ accordingly\}$

Figure 17: The average episode reward means of agent i, the agent whose perspective is taken, when the set of policies Π^j used for the modelled agent j only contains one of the policies of Π^j_B. Sub-figure a is identical to Figure 15 and is plotted for comparative purposes. Sub-figure b shows the results for $\Pi^j = \{listen\ and\ act\ accordingly\}$. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL" and "RL - action j" refer to RL agents that use unaugmented observations and observations augmented with the action of agent j, a^j_t, respectively.

Finally, Figure 18 shows the results for the set with both policies, Π_B^j. Because of the noise present in the retrieved data, not only the average episode reward mean is plotted but also the moving average of the average episode reward mean. The moving average is a low-pass filter which smoothens the data and shows the longer term trend [55, Chapter 15]. It is obtained by repeatedly taking the average of a forward shifting subset. The size of this subset is referred to as window size. Figure 18 shows the moving average with window size 5 (middle subplot) and window size 10 (right subplot) alongside the original data (left subplot). Here it becomes clear that the RL agent that uses observations augmented with the action of agent j, a_t^j, settles at a baseline that always takes the exact same action as agent j. Moreover, it becomes apparent that the RL agent that uses observations augmented with the index of the policy that the modelled agent is currently using, $index(\pi^j)$, is hardly outperforming the naive RL agent that uses unaugmented observations.

Overall, it can be concluded that augmenting observations with the action that agent j is going to take, a_t^j, shows most potential. However, this is presumably harder to learn than the index of the policy that the modelled agent is currently using, $index(\pi^j)$, which showed some potential in case of Π_A^j.

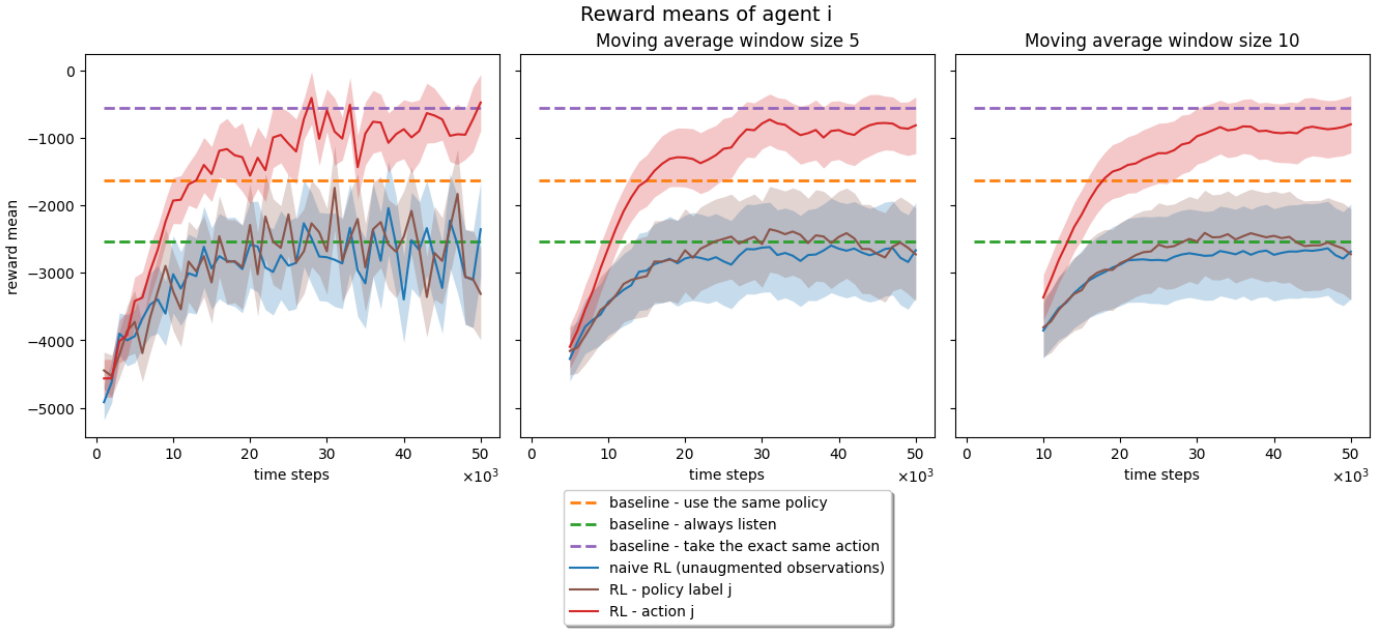


Figure 18: The average episode reward means of agent i for policy set $\Pi_B^j = \{act\ accordingly,\ \ listen\ and\ act\ accordingly\}$. The middle and rightmost plot show the moving average of reward means for windows size 5 and 10 respectively. The shaded aeras depict 95% confidence intervals (1.96 * the standard error). "naive RL", "RL - policy label j" and "RL - action j" refer to RL agents that use unaugmented observations, observations augmented with the index of the policy of agent j, $index(\pi^j)$, and observations augmented with the action of agent j, a_t^j, respectively.

4.6 Predicting Features

This section is devoted to answering the second research question:

- To what extent can these features be predicted by a VAE based model and how does this compare to an LSTM classifier?

The second question is posed to see if the features from the first question can be predicted and represented by a VAE based model as described in Section 2. The VAE based model is trained with various settings and the question is answered by looking at the resulting losses as well as the resulting latent space. Moreover, the resulting reconstruction losses are compared with the losses of corresponding LSTM classifiers that are trained to solve the same classification task. This section is split into three subsections, each one presenting the experiments for a different target variable y:

- Section 4.6.1 covers $y = index(\pi^j)$,
- Section 4.6.2 covers $y = a_t^j$ and
- Section 4.6.3 covers $y = \{a_t^j, o_t^j\}$.

The first two, $y = index(\pi^j)$ and $y = a_t^j$ are chosen based on the results from the first question. Even though augmenting observations with a_t^j outperformed augmenting observations with $index(\pi^j)$, a_t^j is presumably harder to learn which makes $index(\pi^j)$ still worth investigating. This is only investigated for $\Pi^j = \Pi_A^j$. In case of $\Pi^j = \Pi_B^j$ a RL agent that uses observations augmented with $index(\pi^j)$ showed very little to no improvement over a naive RL agent that uses unaugmented observations. The last one, $y = \{a_t^j, o_t^j\}$, is chosen as is it used in [19] and as the higher dimensional target might be better suited to show the difference between using learned representations and using the predictions of an LSTM classifier in the third question. Each subsection first addresses the greedy policy set, $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$, and finishes with the policy set, $\Pi_B^j = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$.

The experiments conducted in this section all use a latent dimension of 2, $|z| = 2$, unless otherwise mentioned. Setting the latent dimension to 2 makes it possible to depict the latent space by plotting the 2-dimensional samples z from the approximate posterior $q(z; \bar{h}^i)$ in a plane. The latent space plots in this section are constructed by sampling 10 times per data point for 20000 data points, so 200000 samples in total per plot. Moreover, the experiments all make use of a monotonic annealing schedule [56, 57] for the loss hyperparameter β which controls the regularization term in the loss: $D_{KL}(q(z; \bar{h}^i) \parallel p(z))$ where $p(z) = \mathcal{N}(0, I)$. The schedule starts with $\beta = 0$ and gradually increases when the reconstruction loss is below a certain threshold[21]. In this case 0.3. This is done to avoid what is called "kl vanishing" [57] which refers to a regularization loss that drops to zero and causes the model to get stuck in a local minimum. Moreover, a regularization loss of zero would mean that the learned posterior exactly follows a standard normal distribution which makes it impossible to represent anything. Therefore, the goal is not to have the regularization loss as low as

21. Every epoch end, the following callback is called: **if** reconstruction loss < 0.3 **then** $\beta \leftarrow \min(1.0, \beta + 0.1 + 2^{step}/100)$ **and** $step \leftarrow step + 1$, where $step$ is initialized with 0 and $\min(x, y)$ returns the item with the lowest value.

possible but to have a regularized reconstruction, so a regularization loss that is as low as possible given a reasonably low reconstruction loss. The top row of Figure 19 shows what would happen over time in the case of "kl vanishing", the bottom row shows what happens when an annealing schedule is applied: first the model learns to reconstruct and subsequently the model learns a regularized reconstruction.

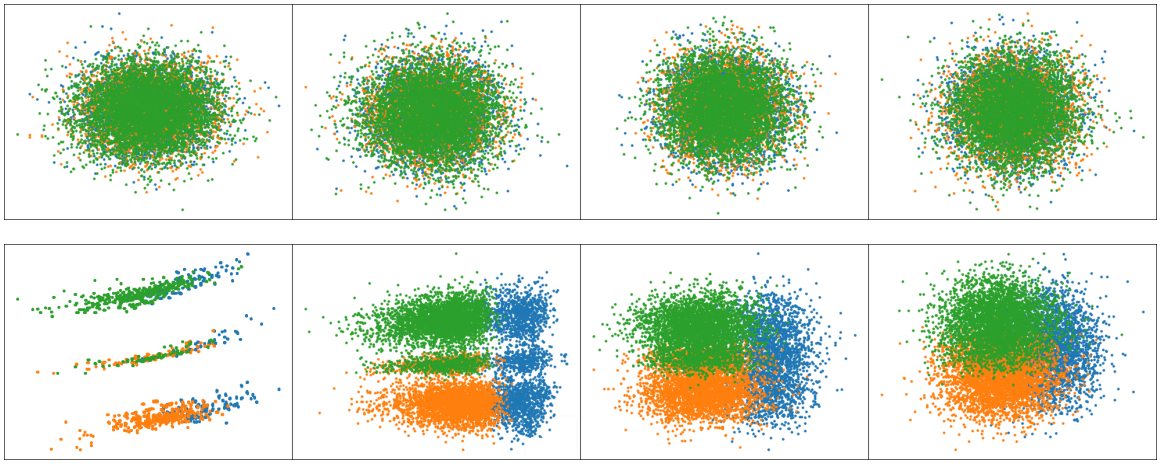


Figure 19: The latent space evaluation over training time (left to right) in general. The top row shows the phenomenon of "kl vanishing" and the bottom row shows the result of using an annealing schedule in the same setting. The samples in the plots are labelled with different colours, blue, orange and green, which denote different outcomes of a feature, in this case the feature is the action of j and the colours denote $OpenLeft$, $OpenRight$ and $Listen$.

4.6.1 Predicting the Policy Index

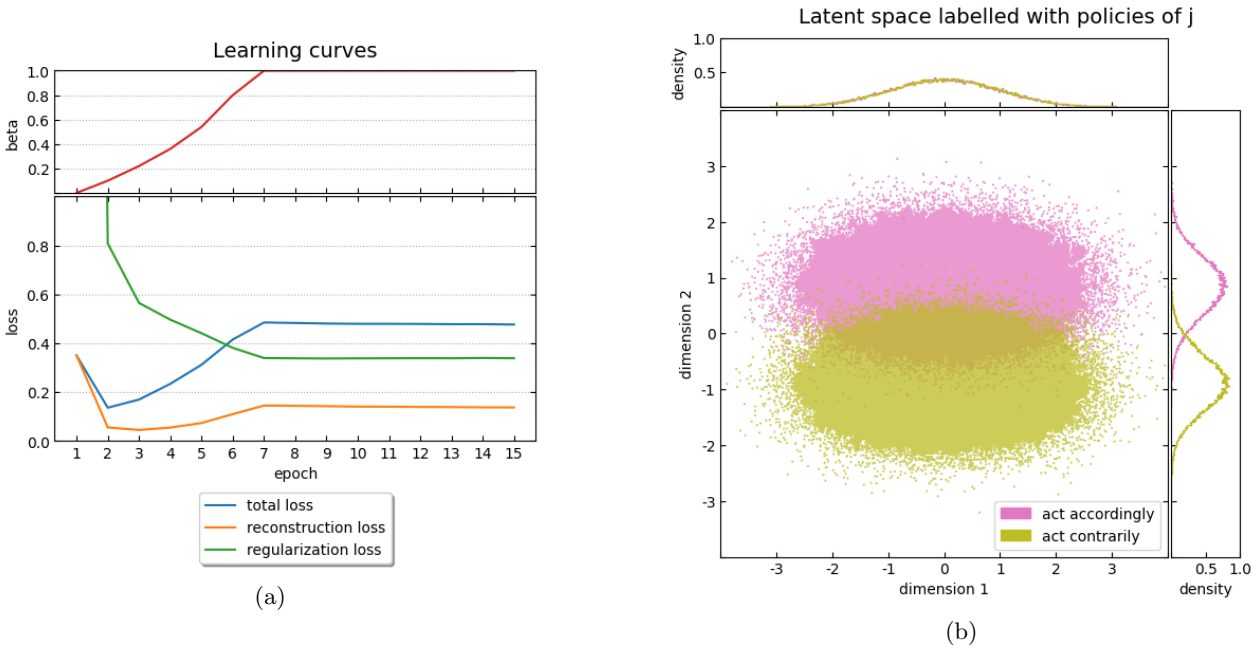
The first setting that is addressed is the greedy policy set $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$ in combination with the index of the policy that the modelled agent is using as a target, $y = index(\pi^j)$. From the learning curves, depicted in Figure 20a, it becomes clear that the resulting regularization loss, 0.339, is higher than the resulting reconstruction loss, 0.138. an LSTM classifier with the same settings settles at a (reconstruction) loss of 0.015. See Table 4 for an overview. The latent space plot, see Figure 20b, shows that dimension 2 captures $index(\pi^j)$ and the density plot of dimension 2 shows that there is little overlap between the distributions. Moreover, the latent space plot also shows that samples labelled with the same policy are grouped together and that there is still some overlap present, which are features of a well regularized latent space.

		Π_A^j
VAE based model	Regularization Loss	0.339
	Reconstruction Loss	0.138
LSTM classifier	(Reconstruction) Loss	0.015

Table 4: The final losses for $y = index(\pi^j)$.

Overall, it can be concluded that the policy that the modelled agent is using as a target, $y = index(\pi^j)$, can be represented quite well using a VAE based model. Besides that, it can be concluded that an LSTM classifier outperforms the VAE based model in terms of predicting or "reconstructing" the target $y = index(\pi^j)$.

Setting: $\Pi^j = \Pi^j_A$ & $y = index(\pi^j)$



(a) (b)

Figure 20: The learning curves & β (a) and latent space (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_A$ and target $y = index(\pi^j)$.

4.6.2 Predicting Actions

Next, the target is changed to $y = a^j_t$ and the policy set stays the same: Π^j_A. From the learning curves depicted in Figure 21a it becomes clear that the resulting reconstruction loss, 0.312, and the regularization loss, 0.317 are almost equal. an LSTM classifier with the same settings settles at a (reconstruction) loss of 0.155. See Table 5 for an overview. Additionally, the latent dimension is set to $|z| = 4$ and $|z| = 8$ to see what the effects would be on the reconstruction loss. Figure 22a and Figure 22b show the learning curves for $|z| = 4$ and $|z| = 8$ respectively. The plots show that increasing the latent dimension simultaneously lowers the reconstruction loss and regularization loss. It is however questionable how useful the learned representations would be for the RL problem as larger representations are harder to interpret. From Figure 21b, which shows the resulting sampled latent space, it becomes clear that dimension 1 captures the action that the modelled agent is going to take: a^j_t. The figure also shows that the samples labelled with the same action are grouped together and that the labelled groups overlap quite a bit.

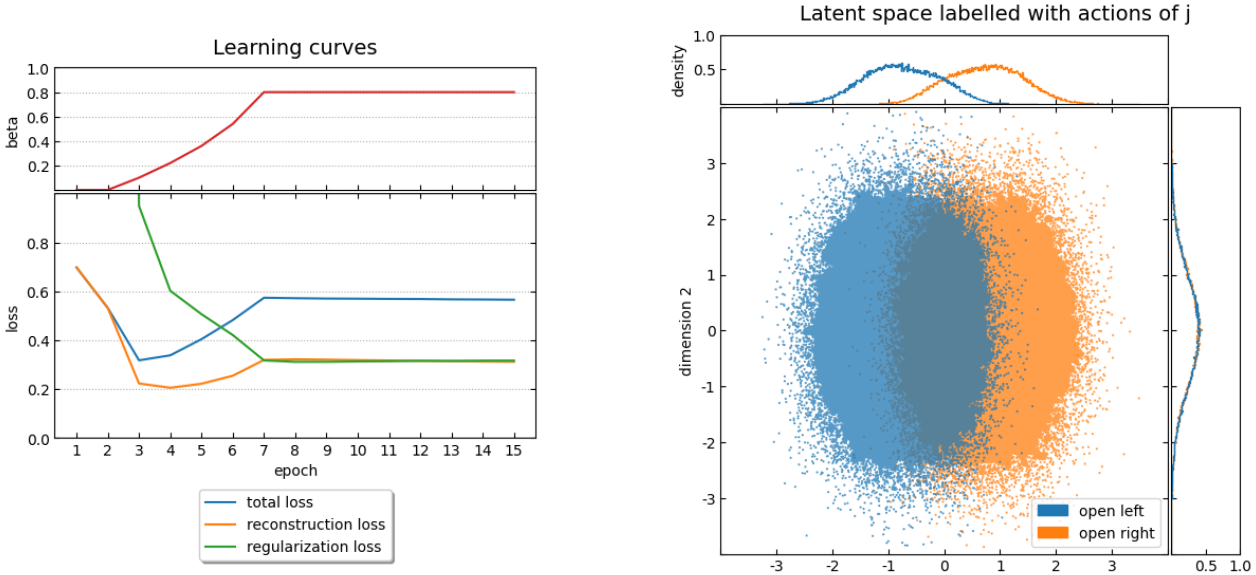
The results for the other plot for policy set $\Pi^j_B = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$ are shown in Figure 23. The reconstruction loss settles at 0.378 and the regularization loss settles at 0.459. an LSTM classifier with the same settings settles at a (reconstruction) loss of 0.111. Finally, the sampled latent space, see Figure 23b, shows that dimension 1 captures whether agent j is going to listen or not and dimension 2 captures which door is opened if a door is opened. Moreover, the scatter plot shows that the samples labelled with the same action are grouped together and that all labelled groups show at least some overlap.

		Π^j_A	Π^j_B
VAE based model	Regularization Loss	0.317	0.459
	Reconstruction Loss	0.312	0.378
LSTM classifier	(Reconstruction) Loss	0.155	0.111

Table 5: The final losses for $y = a^j_t$.

Overall, it can be concluded that the action that the modelled agent is going to take, $y = a^j$, can be represented quite well using a VAE based model. When comparing the results of the two policy sets, the learning curves would suggest that this is easier in case of Π^j_A; however, the latent space plots give no real reason for that. Additionally, it can be concluded that for both policy sets an LSTM classifier outperforms the VAE based model in terms of predicting or "reconstructing" the target $y = a^j_t$.

Setting: $\Pi^j = \Pi^j_A$ & $y = a^j_t$

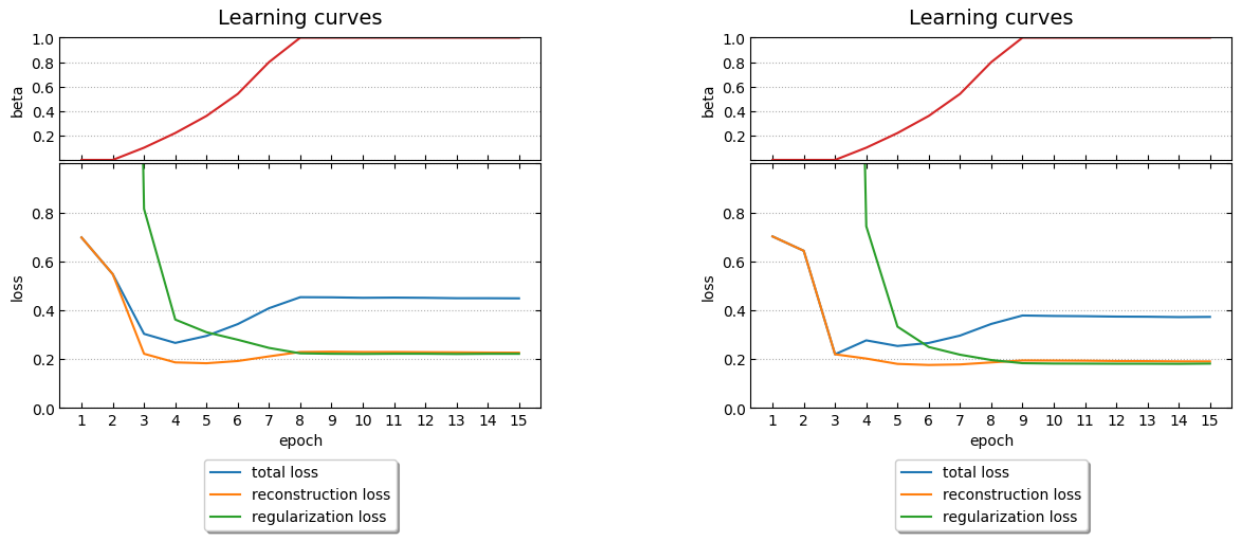


(a) $|z| = 2$

(b)

Figure 21: The learning curves & β (a) and latent space (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_A$ and target $y = a^j_t$.

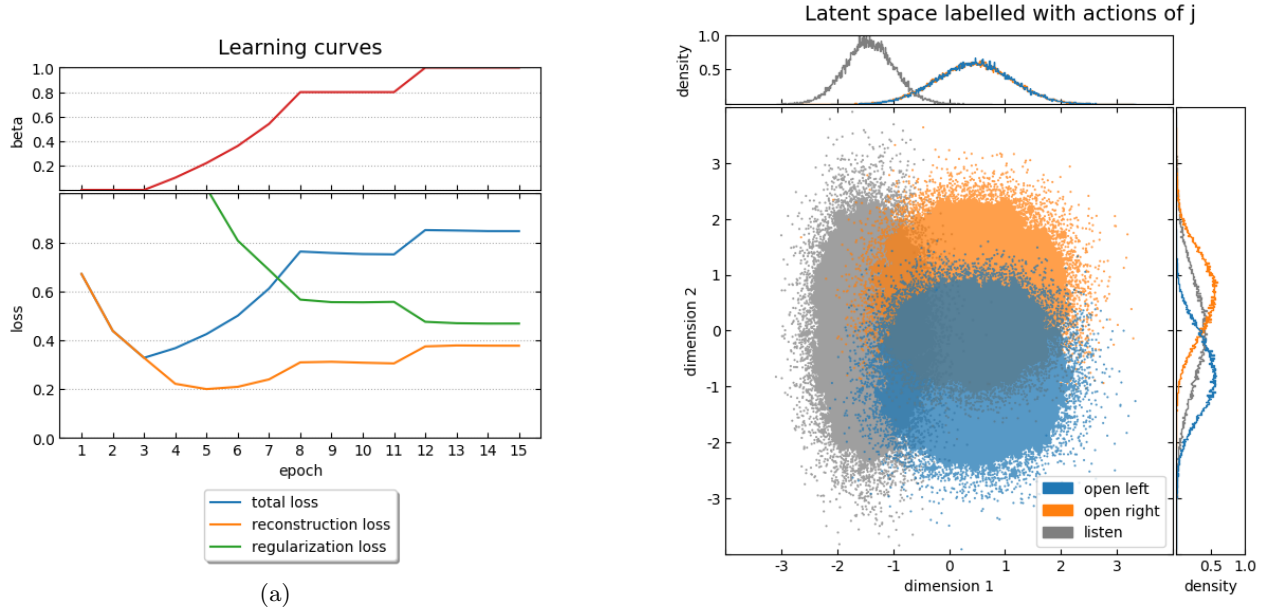
Setting: $\Pi^j = \Pi^j_A$ & $y = a^j_t$



(a) $|z| = 4$ (b) $|z| = 8$

Figure 22: The learning curves & β corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_A$ and target $y = a^j_t$ for $|z| = 4$ (a) and $|z| = 8$ (b).

Setting: $\Pi^j = \Pi^j_B$ & $y = a^j_t$



(a) (b)

Figure 23: The learning curves & β (a) and latent space (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_B$ and target $y = a^j_t$.

4.6.3 Predicting Actions and Observations

The last target that is addressed is a target containing both the action that the modelled agent is going to take and its current observation: $y = \{a_t^j, o_t^j\}$. Firstly, the results of the greedy policy set, $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$, are covered and secondly, the results of the policy set, $\Pi_B^j = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$.

From the learning curves plot, see Figure 24a, it becomes clear that the regularization loss, which settles at 0.595, is higher than in previous experiments. This is the result of the annealing schedule used for β which stopped increasing β because the reconstruction loss did not drop back below the 0.3 threshold. See the β plot in Figure 24a. The resulting reconstruction loss is 0.315 (0.383 for a_t^j and 0.247 for o_t^j), compared to 0.150 (0.160, 0.140) for an LSTM classifier with the same settings. See Table 6 for an overview. From the latent space plots it becomes clear that the latent variable captured the policy of the modelled agent $index(\pi^j)$ in dimension 1, see density plot in Figure 25b, and the current observation of the modelled agent o_t^j in dimension 2, see density plot in Figure 25a. This means that the actions are not captured in a dimension but predicted using the combination of the captured features, $index(\pi^j)$ and o_t^j. The latent space plot labelled with the actions, a_t^j, shows that the combination of the two features can be used to distinguish between the two actions, see the scatter plot in Figure 24b.

Next, policy set $\Pi^j = \Pi_B^j$ is addressed. From the learning curves, plotted in Figure 26a, it becomes clear that the regularization loss, 0.442, and reconstruction loss, 0.407 (0.486 for a_t^j and 0.327 for o_t^j), settle at almost the same level. Moreover, the LSTM classifier settles at a loss of 0.123 (0.113, 0.133). The latent space plot labelled with observations, see Figure 27a, shows that the dimension 2 captured the current observation of the modelled agent: o_t^j. Moreover, instead of capturing the policy of the modelled agent $index(\pi^j)$ or the action that the modelled agent is going to take a_t^j, dimension 1 captured whether the modelled agent is going to listen or not, see the density plot in Figure 26b.

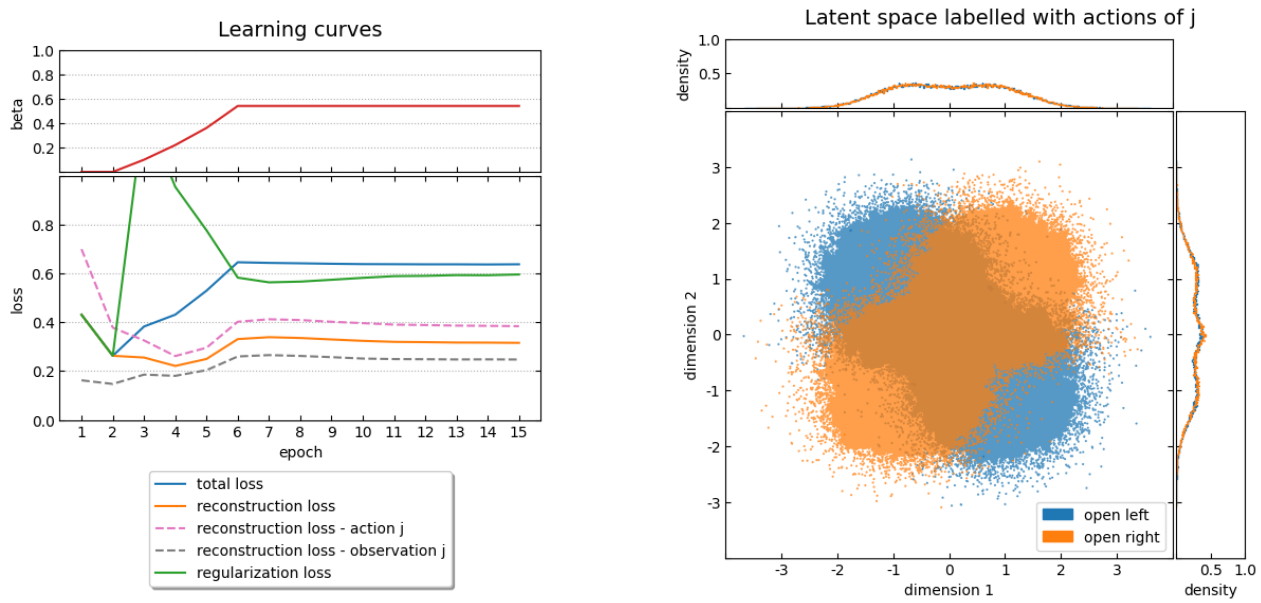
Another observation can be made by looking at the individual reconstruction losses, which show that in general is more difficult to predict the action that the modelled agent is going to take a_t^j than its current observation o_t^j. Finally, the LSTM classifiers show that there is little difference in reconstruction difficulty between the two policy sets, 0.150 for Π_A^j and 0.123 for Π_B^j. Moreover, the latent space plots also show that samples with the same label are grouped together and that there is still some overlap present, which are both features of a well regularized latent space.

		Π_A^j	Π_B^j
VAE based model	Regularization Loss	0.595	0.442
	Reconstruction Loss	0.315 (0.383, 0.247)	0.407 (0.486, 0.327)
LSTM classifier	(Reconstruction) Loss	0.150 (0.160, 0.140)	0.123 (0.113, 0.133)

Table 6: The final losses for $y = \{a_t^j, o_t^j\}$. The numbers between parentheses are the values for a_t^j and o_t^j individually.

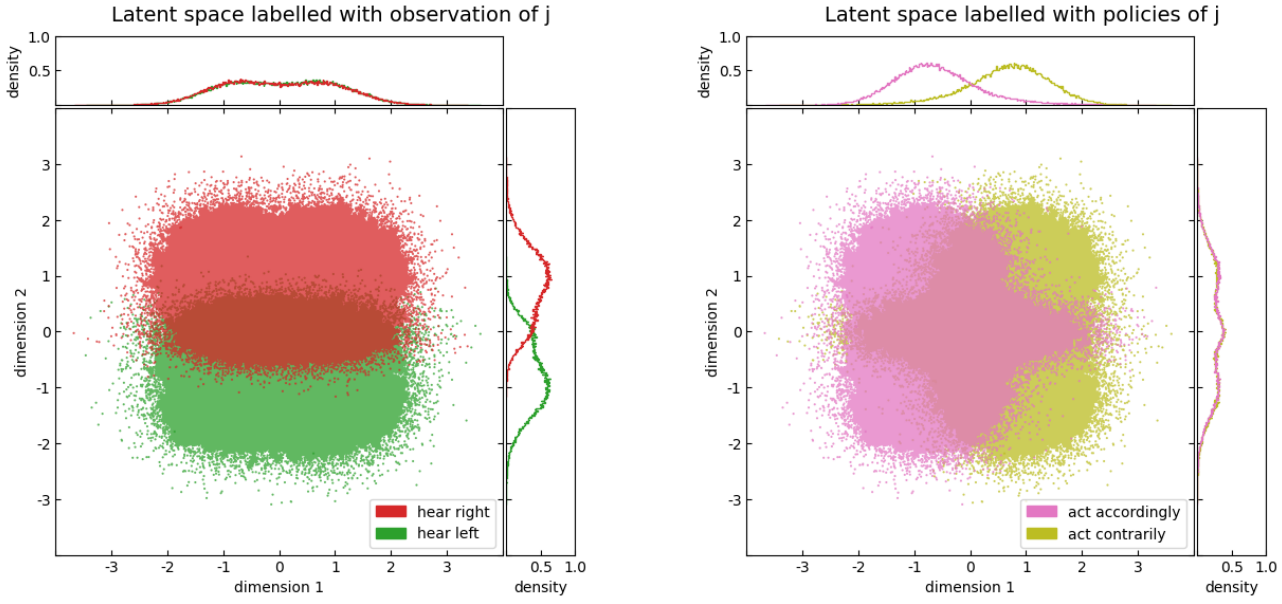
Overall, it can be concluded that both the action that the modelled agent is going to take and its current observation, $\{a_t^j, o_t^j\}$, can be represented quite well using a VAE based model. Moreover, it can be concluded that for both policy sets an LSTM classifier outperforms the VAE based model in terms of predicting or "reconstructing" the target $y = \{a_t^j, o_t^j\}$.

Setting: $\Pi^j = \Pi^j_A$ & $y = \{a^j_t, o^j_t\}$



(a) (b)

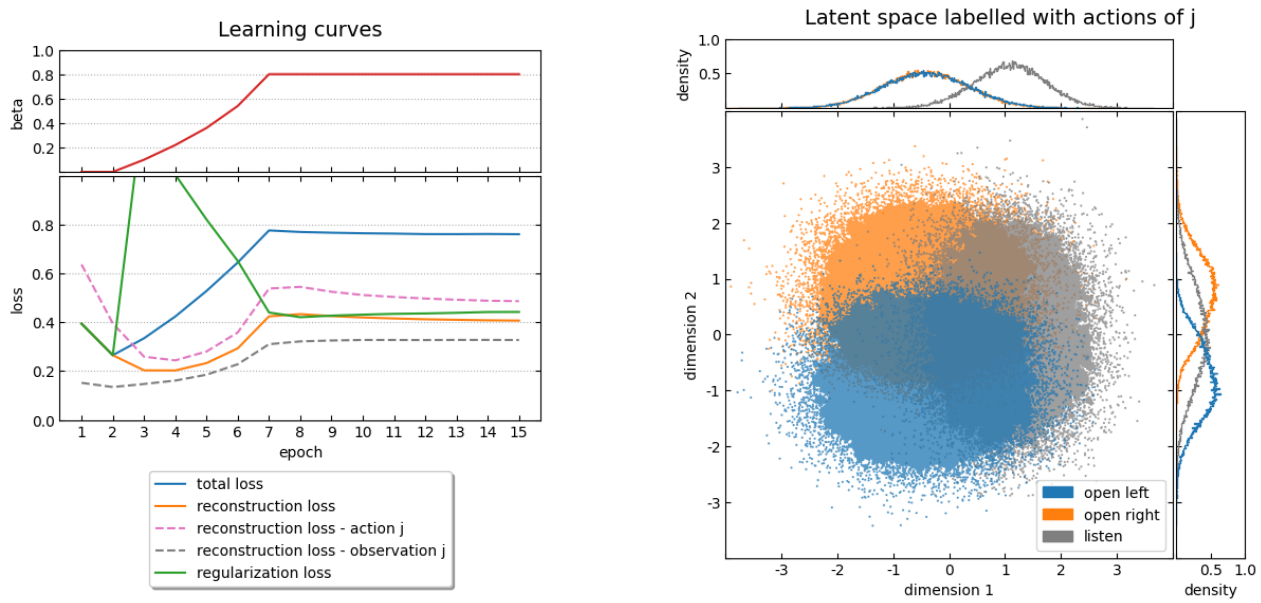
Figure 24: The learning curves & β (a) and sampled latent space labelled with a^j_t (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_A$ and target $y = \{a^j_t, o^j_t\}$.



(a) (b)

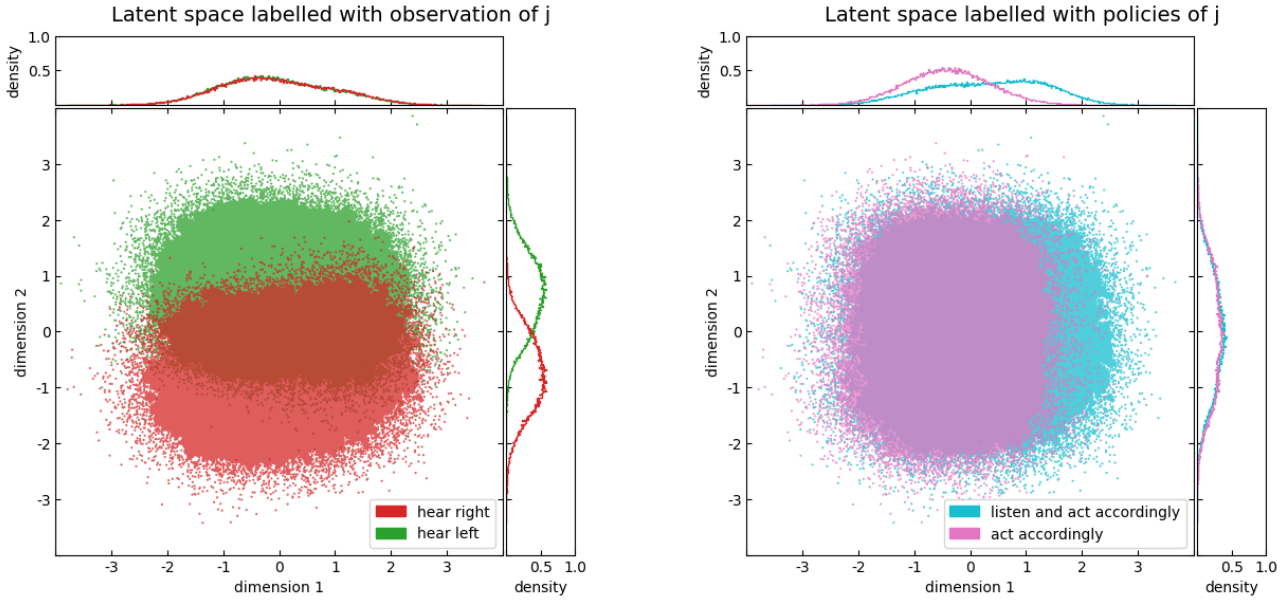
Figure 25: The sampled latent space labelled with o^j_t (a) and labelled with $index(\pi^j)$ (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_A$ and target $y = \{a^j_t, o^j_t\}$.

Setting: $\Pi^j = \Pi^j_B$ & $y = \{a^j_t, o^j_t\}$



(a) (b)

Figure 26: The learning curves & β (a) and sampled latent space labelled with a^j_t (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_B$ and target $y = \{a^j_t, o^j_t\}$.



(a) (b)

Figure 27: The sampled latent space labelled with o^j_t (a) and labelled with $index(\pi^j)$ (b) corresponding to a trained VAE based model using policy set $\Pi^j = \Pi^j_B$ and target $y = \{a^j_t, o^j_t\}$.

4.7 Augmenting Observations with Representations of Features

This last section is devoted to answering the final research question:

- What is the best way to leverage the VAE based model to augment the observation space in the RL problem and how does this model compare to an LSTM classifier?

The question is posed to see which way of augmenting observations works best, given a trained VAE based model and a trained LSTM classifier, both described in detail in Section 2. Three ways of augmenting observations are covered:

1. augmenting the observation with a sample, z, from the approximate posterior, $q(z; \bar{h}_t^i)$, of the trained VAE based model,

2. augmenting the observation with the parameters, μ and σ, from the approximate posterior, $q(z; \bar{h}_t^i)$, of the trained VAE based model and

3. augmenting the observation with a prediction, $\hat{y}$, of the trained LSTM classifier.

Section 3.2 covers these methods in detail. The question is answered by comparing the episode reward means of RL agents that each use a different way of augmenting their observations. Because of the noise present in the retrieved data, not only the average episode reward mean is plotted but also the moving average[22] of the average episode reward mean with window size 5 and window size 10.

This section is split into three subsections that each cover the trained VAE based model and the trained LSTM classifier trained with a different target y:

- Section 4.7.1 covers $y = index(\pi^j)$,

- Section 4.7.2 covers $y = a_t^j$ and

- Section 4.7.3 covers $y = \{a_t^j, o_t^j\}$.

The first two targets, $y = index(\pi^j)$ and $y = a_t^j$ compare RL agents that use the three ways of augmenting observations using the trained models (with z, μ & σ and $\hat{y}$) with RL agents that augment their observations with $index(\pi^j)$ and a_t^j or that use unaugmented observations. The latter are already covered in detail in Section 4.5 and serve as references. The last target, $y = \{a_t^j, o_t^j\}$, only compares the three ways of augmenting observations (with z, μ & σ and $\hat{y}$) to see if a larger target highlights the difference between using a VAE based model and an LSTM classifier as explained in Section 4.6. Similar to Section 4.6, the first subsection only covers the greedy policy set $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$ and the second and third subsection first address the greedy policy set, Π_A^j, and finish with the policy set, $\Pi_B^j = \{act\ accordingly,\ listen\ and\ act\ accordingly\}$.

22. The moving average is a low-pass filter which smoothens the data and shows the longer term trend [55, Chapter 15].

4.7.1 Augmenting Observations with Representations of the Policy Index

Figure 28 shows the results for the greedy policy set $\Pi_A^j = \{act\ accordingly,\ act\ contrarily\}$ in combination with the index of the policy that the modelled agent is using as a target, $y = index(\pi^j)$. Although the plot contains quite a bit of noise and the results are not settled after 50 epochs, it appears that each of the augmentation methods outperform or start to outperform the unaugmented baseline within the plotted epochs. However, none of the augmentation methods reaches the upper bound RL agent that uses observations augmented with the index of the policy that agent j is currently using, $index(\pi^j)$ within the plotted epochs. Looking at the differences between the three augmentation methods, it becomes clear that within the plotted epochs, the LSTM classifier is outperforming the VAE based model and that the VAE based model can best be used to sample from. It is likely that a sample from the approximate posterior contains just as much information as the parameters because the policy of agent j is fixed for the episode and contains no stochasticity. It is therefore plausible that the two will converge but that it takes longer to interpret the parameters because they together are twice as big as a sample, e.g., if $|z| = 2$, then $|\mu| = 2$ and $|\sigma| = 2$.

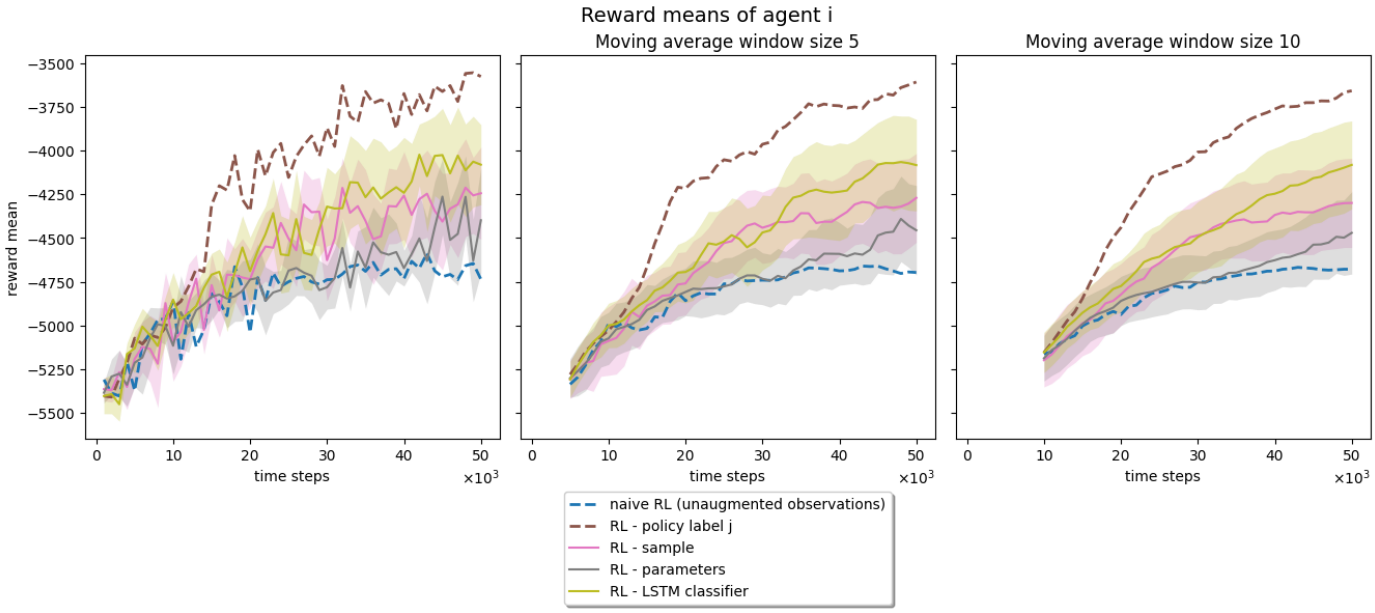


Figure 28: The average episode reward means of agent i for policy set $\Pi^j = \Pi_A^j$ when both the VAE based model and the LSTM classifier are trained using $y = index(\pi^j)$. The middle and rightmost plot show the moving average of reward means for windows size 5 and 10 respectively. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL" and "RL - policy label j" refer to RL agents that use unaugmented observations and observations augmented with the index of the policy of agent j, $index(\pi^j)$, respectively. "RL - sample", "RL - parameters" and "RL - LSTM classifier" refer to RL agents that use observations augmented with z, μ & σ and $\hat{y}$ respectively.

It can be concluded that, although the results are not settled yet, the best results can be achieved by augmenting the observations with predictions made using a trained LSTM classifier. Furthermore, in this case, because of the lack of stochasticity, the best way to utilize the learned posterior is to sample from the approximate posterior.

4.7.2 Augmenting Observations with Representations of Actions

Next, the target $y = a_t^j$ is covered. Figure 29 shows the results for both policy sets, the top row shows the results for the greedy policy set Π_A^j and the bottom row shows the results for Π_B^j. The results for policy set Π_A^j show that all three methods outperform both the naive RL agent that uses unaugmented observations and a RL agent that uses observations augmented with the policy label of the modelled agent.

The filtered results for policy set Π_B^j, depicted in the bottom row in the right columns, show little difference between a RL agent that augments its observation with the parameters of the approximate posterior learned by the VAE based model and a RL agent that augments its observation with a prediction of the learned LSTM classifier. Moreover, both outperform the naive RL agent that uses unaugmented observations and a RL agent that uses observations augmented with the policy label of the modelled agent.

Based on these results it can be confirmed that augmenting observations with a representation or a prediction of the next action that the modelled agent is going to take (a_t^j) is more beneficial than the use of observations augmented with the policy index that the modelled agent is currently using ($index(\pi^j)$). Moreover, in both cases, Π_A^j and Π_B^j, the RL agent that uses the parameters of the approximate posterior learned by the VAE based model outperforms the RL agent that samples from this approximate posterior. However, the results also suggest that the RL agent that augments its observation with a prediction of the learned LSTM classifier is hardly underperforming and settles at the same level as the RL agent that uses the parameters of the approximate posterior learned by the VAE based model. None of the agents reaches the upper bound RL agent that uses observations augmented with the next action that the modelled agent is going to take.

4.7.3 Augmenting Observations with Representations of Actions and Observations

This last section covers the target $y = \{a_t^j, o_t^j\}$. It compares the three ways of augmenting observations (with z, μ & σ and $\hat{y}$) to see if a larger target highlights the difference between using a VAE based model and an LSTM classifier. Figure 30 shows the results for target $y = \{a_t^j, o_t^j\}$, the top row shows the results for the greedy policy set Π_A^j and the bottom row shows the results for Π_B^j. For policy set Π_A^j, it can be concluded that using a VAE based model, either by augmenting the observation with a sample or the parameters from the approximate posterior, is less beneficial than using the predictions of an LSTM classifier to augment observations with. For the other policy set Π_B^j, the bottom row, the difference is less clear. Here it can be concluded that there is little to no difference and that an LSTM classifier can probably be used just as well as a VAE based model.

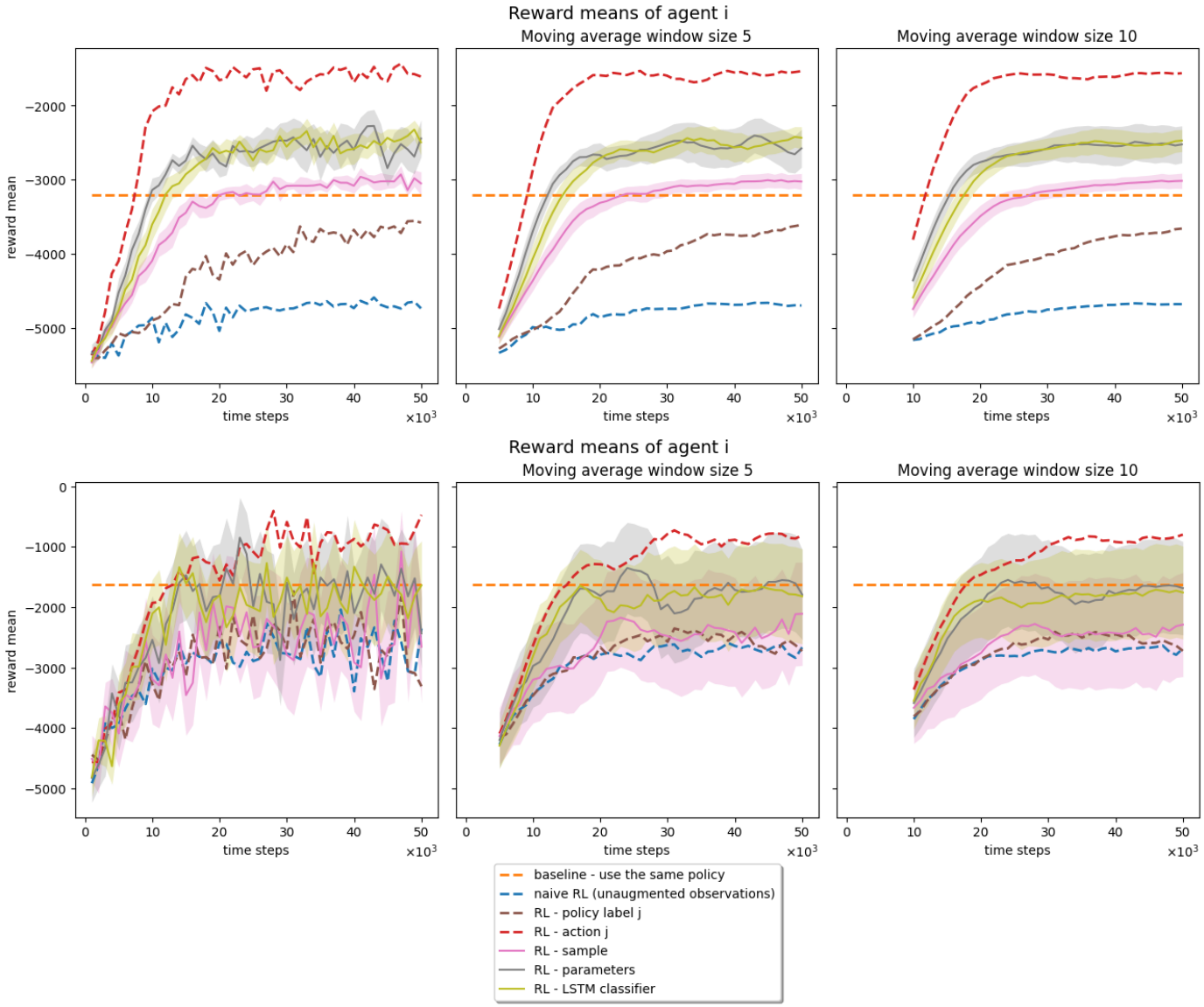


Figure 29: The average episode reward means of agent i. The top row shows the results for $\Pi^j = \Pi^j_A$ and the bottom row shows the results for $\Pi^j = \Pi^j_B$. Both the VAE based model and the LSTM classifier are trained using $y = a^j_t$. The middle and rightmost plots show the moving average of reward means for windows size 5 and 10 respectively. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "naive RL", "RL - policy label j" and "RL - action j" refer to RL agents that use unaugmented observations, observations augmented with the index of the policy of agent j, $index(\pi^j)$, and observations augmented with the action of agent j, a^j_t, respectively. "RL - sample", "RL - parameters" and "RL - LSTM classifier" refer to RL agents that use observations augmented with z, μ & σ and $\hat{y}$ respectively.

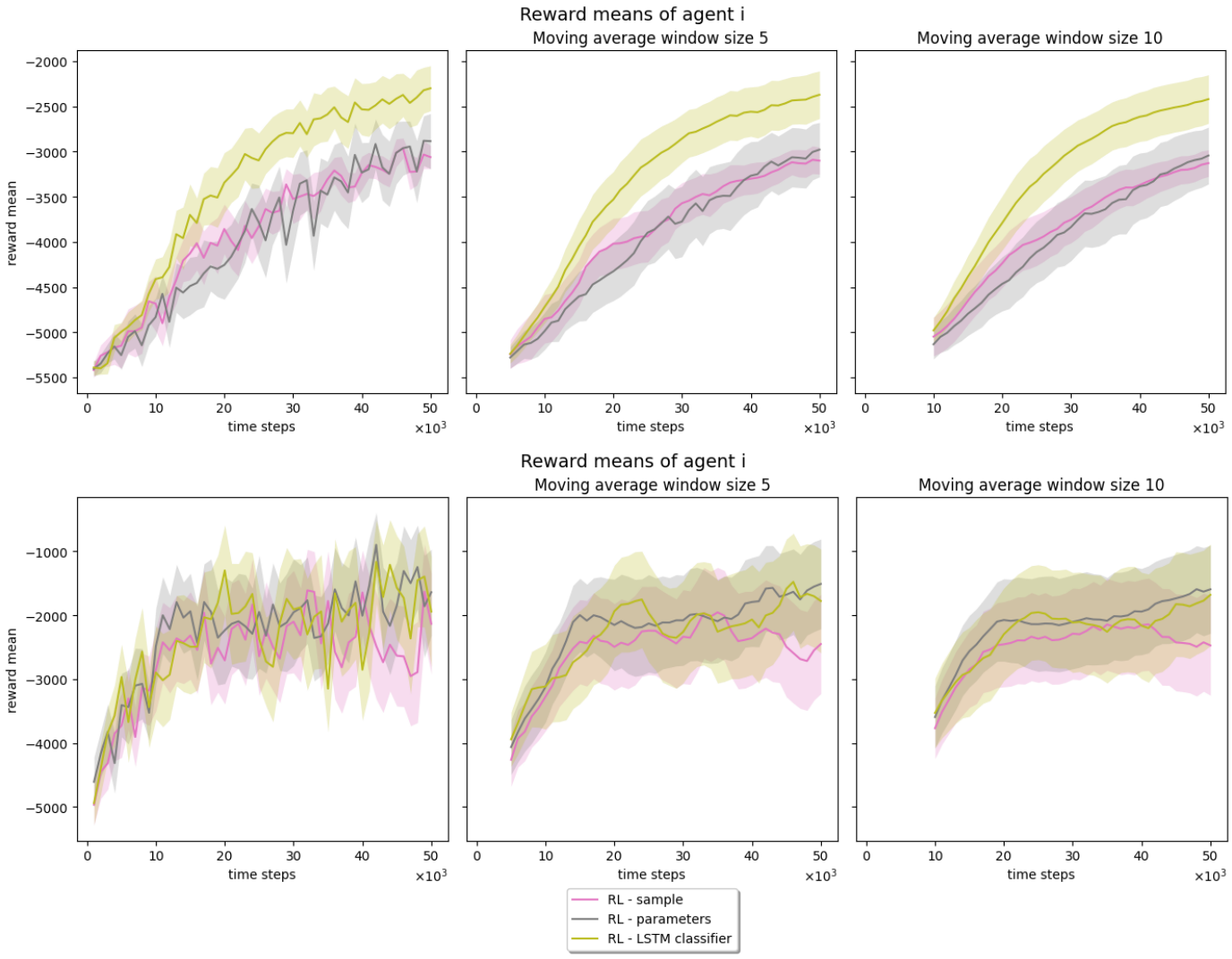


Figure 30: The average episode reward means of agent i. The top row shows the results for $\Pi^j = \Pi^j_A$ and the bottom row shows the results for $\Pi^j = \Pi^j_B$. Both the VAE based model and the LSTM classifier are trained using $y = \{a^j_t, o^j_t\}$. The middle and rightmost plots show the moving average of reward means for windows size 5 and 10 respectively. The shaded areas depict 95% confidence intervals (1.96 * the standard error). "RL - sample", "RL - parameters" and "RL - LSTM classifier" refer to RL agents that use observations augmented with z, μ & σ and $\hat{y}$ respectively.

5. Conclusion

This section consists of two parts: Section 5.1 which revisits the three research questions, tests the hypothesis and draws an overarching conclusion and Section 5.2 which covers the recommended future work.

5.1 Research Questions Revisited

The aim of this study was to investigate the usefulness of the augmentation of the observations of a RL agent with representations of features related to other agents generated by VAE based architectures in partially observable settings. This was investigated by: (1) finding the best features to augment observations with, (2) testing if these features could be predicted by a VAE based model, (3) finding the best way to augment the observations using the VAE based model and (4) comparing the results with an LSTM classifier. This was formulated in three research questions which are answered in order:

1. What would be the best feature related to another agent to augment the observations with?

   > The results show that augmenting the observations is beneficial. Among the features considered, it was most beneficial to augment the current observation with the action of the modelled agent; however, augmenting the observation with the index of the policy of the modelled agent also proved to be beneficial in one of the two policy sets that were considered for the modelled agent.

2. To what extent can these features be predicted by a VAE based model and how does this compare to an LSTM classifier?

   > Even though in the previous question the action of the modelled agent was identified as the most beneficial feature to augment observations with, the index of the policy is tested as well. The reason for this is that the action of the modelled agent is presumably harder to learn. The experiments showed that both can be predicted quite well by a VAE based model. However, in all the experiments the LSTM classifier shows equally good or better performance in terms of loss error of the classification task. The same holds for the action that the modelled agent is going to take in combination with its current observation.

3. What is the best way to leverage the VAE based model to augment the observation space in the RL problem and how does this model compare to an LSTM classifier?

   > It became clear that overall, it is most beneficial to augment the observations with the actions of the modelled agent and that this is better than using unaugmented observations. Moreover, the experiments seem to suggest that the best way to employ the VAE based model depends on the stochasticity of the feature. They suggest that if there is stochasticity present in the feature it is best to augment the observation with the parameters of the

approximate posterior otherwise it is best to augment with a sample. The comparisons with the LSTM classifier showed that the latter reaches the same or better performance in all scenarios.

Overall, the results indicate that it can be beneficial to augment the observations of deep RL agents with features related to other agents that are learned in a pre-training phase. It also became clear that an LSTM classifier reaches the same performance or better performance than a VAE based model employed to solve the same prediction task. This has not been shown in previous studies because they lacked a comparison between a VAE based model and an LSTM classifier trained to solve the same classification task. That is, a consistent comparison between the two classifiers is missing. Instead, they compared a VAE based model that tries to predict actions and observations with an LSTM classifier that tries to predict the index of the policy of the modelled agent.

Finally, some limitations of this study should be considered. This study is first of all limited in the number of environments that is used for the experiments as it only uses a single environment with a shared reward. Moreover, this tiger problem environment uses a slightly altered observation function that increases the probability that the two agents observe the same. This was done because the approach taken in this study and in previous studies works well when the decision making problems of the different interacting agents are strongly coupled. Previous studies used grid world like environments where the agents are able to observe the actions of the other agents when they are close enough. Besides the limited number of environments, the study also uses a limited number of policies for the other agent. Combinations of other environments with different policies might yield different results.

5.2 Recommendations for Future Work

Further research needs to be conducted to confirm that these findings hold in other environments. It would, for example, be interesting to see how well this approach scales to settings with more than two agents. Other settings that would be interesting to investigate are environments with individual rewards or with opponents that show adversarial behaviour. Regarding the agent modelling, it would be interesting to see if the VAE based architecture is better suited for environments that contain high dimensional features such as image observations. In such settings, the VAE based architecture might be able to learn more compact representations of the features than an LSTM classifier. Moreover, in environments that contain high dimensional action spaces, e.g., environments with a continuous action space, the VAE based architecture might come in handy as an LSTM classifier would predict high dimensional discrete distributions and an LSTM regressor would only provide a point estimate. Therefore, the VAE based architecture might be an interesting alternative for LSTMs to learn a compact representation and/or to learn a distribution in situations with high dimensional features.

Appendices

A VAE Evidence Lower Bound Derivation

This appendix contains the derivation of the evidence lower bound of the VAE loss. First note that:

$$\begin{aligned} L(g', g, h) = D_{KL}(q(z;x) \parallel p(z|x)) &= \int_z q(z;x) \log\left(\frac{q(z;x)}{p(z|x)}\right) dz \\ &= -\int_z q(z;x) \log\left(\frac{p(z|x)}{q(z;x)}\right) dz. \end{aligned}$$

The ELBO is then derived as follows, starting with rewriting the KL divergence in terms:

$$\begin{aligned} -D_{KL}(q(z;x) \parallel p(z|x)) &= \int_z q(z;x) \log\left(\frac{p(z|x)}{q(z;x)}\right) dz \\ &= \int_z q(z;x) \log\left(\frac{p(x|z)p(z)}{q(z;x)p(x)}\right) dz && \text{(apply Bayes' rule)} \\ &= \int_z q(z;x) \left(\log\left(\frac{p(x|z)p(z)}{q(z;x)}\right) - \log p(x)\right) dz && \text{(unpack logarithm)} \\ &= \int_z q(z;x) \log\left(\frac{p(x|z)p(z)}{q(z;x)}\right) dz - \int_z q(z;x) \log p(x) dz \\ &= \int_z q(z;x) \log\left(\frac{p(x|z)p(z)}{q(z;x)}\right) dz - \log p(x) \\ &= \int_z q(z;x) \left(\log\left(\frac{p(z)}{q(z;x)}\right) + p(x|z)\right) dz - \log p(x) && \text{(unpack logarithm)} \\ &= \int_z q(z;x) \log\left(\frac{p(z)}{q(z;x)}\right) dz + \int_z q(z;x) \log p(x|z) dz - \log p(x) \\ &= -D_{KL}(q(z;x) \parallel p(z)) + \int_z q(z;x) \log p(x|z) dz - \log p(x) \\ &= -D_{KL}(q(z;x) \parallel p(z)) + \mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - \log p(x) && (36) \end{aligned}$$

Then using the inequality $\log x \leq x - 1$, the following derivation can be made:

$$\begin{aligned} -D_{KL}(q(z;x) \parallel p(z|x)) &= \int_z q(z;x) \log\left(\frac{p(z|x)}{q(z;x)}\right) dz \\ &\leq \int_z q(z;x) \left(\frac{p(z|x)}{q(z;x)} - 1\right) dz \\ &= \int_z q(z;x) \left(\frac{p(z|x)}{q(z;x)}\right) dz - \int_z q(z;x) \\ &= \int_z p(z|x) dz - \int_z q(z;x) \\ &= 1 - 1 \\ &= 0, \end{aligned}$$

therefore:

$$D_{KL}(q(z;x) \parallel p(z|x)) \geq 0. \tag{37}$$

Combining Equation 36 and Equation 37 then gives:

$$\begin{aligned} D_{KL}(q(z;x) \parallel p(z|x)) &\geq 0 \\ 0 &\geq -D_{KL}(q(z;x) \parallel p(z|x)) \\ 0 &\geq -D_{KL}(q(z;x) \parallel p(z)) + \mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - \log p(x) \\ \log p(x) &\geq \mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - D_{KL}(q(z;x) \parallel p(z)). \end{aligned} \tag{38}$$

The left hand side of Equation 38 is called the evidence and the right hand side of Equation 38 is the called evidence lower bound (ELBO), which is split in two parts, the reconstruction loss $\mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right]$ and the regularization loss $D_{KL}(q(z;x) \parallel p(z))$. Moreover, $\mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right]$ is the expectation that $\hat{x} = x$ given x and $D_{KL}(q(z;x) \parallel p(z))$ is the difference between the learned distribution over the representations and the prior over the representations. Finally, using the ELBO, the following derivation can be made:

$$\begin{aligned} g'^*, g^*, h^* &= L(g', g, h) \\ &= \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmin}} D_{KL}(q(z;x) \parallel p(z|x)) \\ &= \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmin}} -\left(-D_{KL}(q(z;x) \parallel p(z)) + \mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - \log p(x)\right) \quad \text{(apply Equation 36)} \\ &= \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmin}} -\left(\mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - D_{KL}(q(z;x) \parallel p(z))\right) + \log p(x) \\ &= \underset{(g',g,h)\in G'\times G\times H}{\operatorname{argmax}} \left(\mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - D_{KL}(q(z;x) \parallel p(z))\right). \end{aligned}$$

Moreover, as f is unknown and needed to compute $p(x|z)$,the VAE loss becomes[23]:

$$L(g', g, h, f) = -\left(\mathbb{E}_{z\sim q(z;x)}\left[\log p(x|z)\right] - D_{KL}(q(z;x) \parallel p(z))\right).$$

23. Note that the sign flipped because $L(g', g, h, f)$ is minimized.

B Number of Runs Collected per Figure

The table shows for each figure and each agent configuration for agent i how many times the experiment is performed.

figure\agent	naive RL	RL - policy label j	RL - action	RL - sample	RL - parameters	RL - LSTM classifier
13a	6	-	-	-	-	-
13b	6	-	-	-	-	-
14	12	12	-	-	-	-
15	6	-	6	-	-	-
16	12	12	12	-	-	-
17a	6	-	6	-	-	-
17b	6	-	6	-	-	-
18	30	30	30	-	-	-
28	12	12	-	12	12	12
29 (top)	12	12	12	12	12	12
29 (bottom)	30	30	30	12	12	12
30 (top)	-	-	-	20	20	20
30 (bottom)	-	-	-	20	20	20

C Reinforcement Learning Hyperparameters

Parameter	Value(s) Used				Value(s) Tested
	LIAM [19]	MeLIBA [20]	Empirical Study [58]	Used in Tiger Problem	
Algorithm	A2C	PPO	-	PPO	A2C, PPO
Layers	[128, 128]	[128, 128]	2 layers[24]	[128, 128]	[128, 128]
Activation	ReLu	Tanh	Adam	Tanh	Tanh
Learning Rate η	0.0001	0.00005	0.0003	0.0003	0.001, 0.0001, 0.00001, 0.0003
Value Loss Coefficient β_1	1	0.5	-[25]	-[26]	0.5, 1
Entropy Coefficient β_2	0.01	0.2	0.0	0.01	0.01
Discount Factor γ	0.99	1	0.99	1	1
GAE Parameter λ[27]	0.95	0.9	0.9	0.9	0.9
PPO Clipping ϵ	-	0.1	0.25	0.25	0.25
Gradient Norm Clipping	0.5	0.5	0.5	0.5	0.5

24. The empirical study does not recommend a specific size.
25. The empirical study recommends to use a separate neural network to learn the critic which is trained using the value loss. Hence, in this case the value loss coefficient is redundant.
26. After testing it was decided to use PPO with separate parameters for the actor θ_A and the critic θ_C to further stabilize the learning (see Section 4.4) making the value loss coefficient redundant.
27. A parameter used for generalized advantage estimation (GAE) which is used to estimate $\hat{A}_t$[46].

References

[1] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[2] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun, "Game theoretic control for robot teams," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 1163–1169, IEEE, 2005.

[3] X. Li, J. Zhang, J. Bian, Y. Tong, and T.-Y. Liu, "A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network," *arXiv preprint arXiv:1903.00714*, 2019.

[4] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.

[5] Y. Rizk, M. Awad, and E. W. Tunstel, "Decision making in multiagent systems: A survey," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 514–529, 2018.

[6] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[7] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, 2021.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[10] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, pp. 1–49, 2021.

[11] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[12] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2010.

[13] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[14] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella, "Taming decentralized pomdps: Towards efficient policy computation for multiagent settings," in *IJCAI*, vol. 3, pp. 705–711, Citeseer, 2003.

[15] S. V. Albrecht and P. Stone, "Autonomous agents modelling other agents: A comprehensive survey and open problems," *Artificial Intelligence*, vol. 258, pp. 66–95, 2018.

[16] P. J. Gmytrasiewicz and P. Doshi, "A framework for sequential planning in multi-agent settings," *Journal of Artificial Intelligence Research*, vol. 24, pp. 49–79, 2005.

[17] A. Panella and P. Gmytrasiewicz, "Interactive pomdps with finite-state models of other agents," *Autonomous Agents and Multi-Agent Systems*, vol. 31, no. 4, pp. 861–904, 2017.

[18] G. Papoudakis and S. V. Albrecht, "Variational autoencoders for opponent modeling in multi-agent systems," *arXiv preprint arXiv:2001.10829*, 2020.

[19] G. Papoudakis, F. Christianos, and S. Albrecht, "Agent modelling under partial observability for deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[20] L. Zintgraf, S. Devlin, K. Ciosek, S. Whiteson, and K. Hofmann, "Deep interactive bayesian reinforcement learning via meta-learning," *arXiv preprint arXiv:2101.03864*, 2021.

[21] F. A. Oliehoek and C. Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016.

[22] S. Seuken and S. Zilberstein, "Formal models and algorithms for decentralized decision making under uncertainty," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, pp. 190–250, 2008.

[23] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement Learning*, pp. 387–414, Springer, 2012.

[24] C. Amato, G. Chowdhary, A. Geramifard, N. K. Üre, and M. J. Kochenderfer, "Decentralized control of partially observable markov decision processes," in *52nd IEEE Conference on Decision and Control*, pp. 2398–2405, IEEE, 2013.

[25] F. A. Oliehoek, M. T. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *Journal of Artificial Intelligence Research*, vol. 32, pp. 289–353, 2008.

[26] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[27] E. A. Hansen, D. S. Bernstein, and S. Zilberstein, "Dynamic programming for partially observable stochastic games," in *AAAI*, vol. 4, pp. 709–715, 2004.

[28] J. Hu, M. P. Wellman, *et al.*, "Multiagent reinforcement learning: theoretical framework and an algorithm.," in *ICML*, vol. 98, pp. 242–250, Citeseer, 1998.

[29] R. Gibbons *et al.*, *A primer in game theory*. Harvester Wheatsheaf New York, 1992.

[30] B. Ng, C. Meyers, K. Boakye, and J. Nitao, "Towards applying interactive pomdps to real-world adversary modeling," in *Twenty-Second IAAI Conference*, 2010.

[31] P. J. Doshi, "Decision making in complex multiagent contexts: A tale of two frameworks," *AI Magazine*, vol. 33, no. 4, pp. 82–82, 2012.

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

[33] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[34] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[36] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[37] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[38] S. Rogers and M. Girolami, *A first course in machine learning*. Chapman and Hall/CRC, 2016.

[39] C. Zhang, J. Bütepage, H. Kjellström, and S. Mandt, "Advances in variational inference," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 2008–2026, 2018.

[40] M. Thomas and A. T. Joy, *Elements of information theory*. Wiley-Interscience, 2006.

[41] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[42] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[43] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[44] E. Benhamou, "Variance reduction in actor critic methods (acm)," *arXiv preprint arXiv:1907.09765*, 2019.

[45] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, PMLR, 2016.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[47] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

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

[49] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 2006.

[50] M. Abadi, Barham, *et al.*, "Tensorflow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[51] Tensorflow, "Tensorflow." `https://www.tensorflow.org/versions/r2.8/api_docs/`, v2.8.0.

[52] F. Chollet *et al.*, "Keras." `https://keras.io/`, 2015.

[53] Ray, "Rllib." `https://docs.ray.io/en/releases-1.12.1/rllib/`, v1.12.1.

[54] Ray, "Rllib." `https://github.com/ray-project/ray/releases/tag/ray-1.12.1`, v1.12.1.

[55] S. W. Smith *et al.*, "The scientist and engineer's guide to digital signal processing," 1997.

[56] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015.

[57] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin, "Cyclical annealing schedule: A simple approach to mitigating kl vanishing," *arXiv preprint arXiv:1903.10145*, 2019.

[58] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," *arXiv preprint arXiv:2006.05990*, 2020.