Action planning in discrete state spaces: A comparison of reinforcement learning and active inference

Vegard Sanden

December 2021

Supervisor: Anastasios Lekkas

Norwegian University of Science and Technology

Department of Engineering Cybernetics

Abstract

In recent years machine learning, and more specifically deep learning, has been responsible for the most significant developments in artificial intelligence. Even though deep learning is useful for solving complex problems and learn from large amount of data it has several challenges. A few examples are that deep learning lacks transparency and robustness. Due to these challenges it might be of interest to look at other fields of artificial intelligence. Recently there have been released several papers about *the free energy principle* and *active inference*. The free energy principle is a way of describing how self-organizing systems remains inside a set of states that is defined for the specific system. By minimizing the free energy of the system, the behaviour can be optimized. Active inference uses the free energy principle to make a framework for solving and understanding the behaviour of self-organizing systems where decision-making in an uncertain environment is required. Thus, in this thesis active inference is explored to see how well it performs doing high-level action planning in discrete environments with different characteristics. For comparison, several reinforcement learning algorithms are implemented.

In the first part of the thesis theory about reinforcement learning and active inference is presented. Then, active inference, value iteration and Q-learning are tasked to perform action planning by calculating a policy on different Frozen Lake environments with both stochastic and deterministic transitions. Active inference is implemented in two different ways. One where the expected free energies are calculated from predefined policies and another where the expected free energies are calculated in a recursive manner, called *sophisticated active inference*. The results of these models are compared and discussed to give an overview of characteristics, advantages and disadvantages. Sophisticated active inference is also implemented on a simplified docking problem to begin to explore how it performs on a larger problem. Here, also a Deep Q-Network is implemented for comparison.

The results show that active inference can be used for action planning in a deterministic environment as both implementations perform optimally. For the simplified docking problem sophisticated active inference manages to guide an agent from every open position to the goal, which the Deep Q-Network does not achieve as training would take to much time to achieve that. Even though sophisticated active inference works well in the simplified docking problem, it has a long run time and need to know the prior and likelihood matrices beforehand. On the other hand, it uses probabilities instead of a reward function to reinforce the agent which is an advantage as it can be hard to find the correct reward function. In the stochastic environment active inference calculated with predefined policies performs a lot worse than value iteration and Q-learning. Sophisticated active inference performs almost as well as the reinforcement learning algorithms and shows that it also can be used for action planning in uncertain environments.

Preface

This project thesis was written in the autumn of 2021 at Norwegian University of Science and Technology (NTNU). The aim of this thesis is to explore active inference and how it compares to different reinforcement learning algorithms for high-level action planning.

The algorithms implemented in this thesis were implemented with Python run on the Python distribution platform Anaconda. The Python libraries NumPy and Matplotlib were used for performing linear algebra and plotting, respectively. In addition, the machine learning library PyTorch was used for construction of the deep neural network when implementing the Deep Q-Network. To create and explore the Frozen Lake-environment made by OpenAI, the OpenAI Gym library was used. Also, the pymdp package hosted on the infer-actively GitHub organization was used when implementing active inference.

I would like to thank my supervisor Anastasios Lekkas for the help throughout the semester. The guidance and discussions have been very valuable and useful in the process of writing this thesis.

Contents

List of Figures

List of Tables

1 Introduction

This thesis investigates how the free energy principle in an active inference agent can be used in planning tasks and how its performance compares to reinforcement learning. In this introduction the background and motivation for exploring this topic is presented as well as the contributions of the paper. In addition, an outline of the structure of the thesis is given.

1.1 Background and motivation

Machine learning is an area of artificial intelligence which makes it possible for computers to learn from data without being explicitly programmed to do so [43]. These algorithms adapt the architecture when presented with data inputs to reproduce the desired outcomes from the training inputs and generalizes to produce desired outcomes from new data. In such a way computers can handle large amount of data efficiently and extract information from the data [6]. In robotics, machine learning is one of the major parts of research. Several areas in robotics utilize machine learning and machine learning has become an important part of robotics in the recent years [25, 30, 39]. Some examples of applications of machine learning in areas of robotics are robot vision [32], motion control for robot navigation [42] and robotic process automation [41].

Deep learning is a class of machine learning and has been one of the most significant developments in artificial intelligence and computer science in the recent years. The recent development of deep learning has had an impact in many scientific fields and is transforming businesses and industries [36]. Also, almost all new artificial intelligence applications are driven by deep learning. Even though application of deep learning can be useful for solving complex problems and learn from large amount of data, the methods in general have some drawbacks. For example, deep learning lacks transparency which means that it is hard to know why an algorithm reached a specific output. It is more or less a black box, where data is used as input which yields an output, but it is hard to understand the behaviour of the model. Also, deep learning is not very robust which means it is vulnerable to changes in the input data [40]. For example, if the machine learning model is trained on some data and tested on another independent set it is most likely to have larger errors on the test data. When it comes to robotic applications these drawbacks are difficult to deal with.

Due to the challenges presented above, it might be of interest to look into different fields of artificial intelligence to explore the potential of promising methods to improve autonomy within robotic applications. In this thesis the free energy principle will be explored. The free energy principle was introduced by Karl Friston and there exists several papers explaining the principle and its background [8, 9,

12]. In short, the free energy principle says that an agent must minimize its free energy, which is defined as an upper bound on surprise. Here, surprise is defined as how likely a it is for an agent to observe or be in a certain state. If an agent is in a state with high probability it means that the state is not surprising for the agent. By minimizing free energy the agent will minimize surprise and in this way maintain in its non-surprising states and resist the tendency to disorder. The agent receives observations from the environment and can minimize the free energy by either optimizing its internal model of the environment or acting on the it [9].

In recent years the free energy principle has been used in artificial intelligence as active inference, which is the free energy principle applied to action selection for decision-making in uncertain environments. It has been researched how active inference can be used when the objective is to optimize behaviour and learn in a environment [11, 27]. Active inference has also been tested on small or discrete state spaces and as active inference has several similarities with reinforcement learning there have been comparisons of these two methods of artificial intelligence [13, 29, 34]. As there are disadvantages with active inference when state spaces and action spaces become large we have recently seen that active inference has been used in combination with deep neural networks in what is called deep active inference [7, 23, 38]. Deep active inference has also been used for autonomous robot navigation [3]. Another way of solving the issue of large state and action spaces is sophisticated active inference which calculates expected free energy, that is used for action selection, in a recursive manner [10].

The main goal of the thesis is to explore how active inference performs as an action planner of agents in different discrete environments. As reinforcement learning and active inference have similarities and as the goal of reinforcement learning is to discover an optimal action sequence, it was seen as natural to compare these two methods of artificial intelligence. Reinforcement learning is an area of machine learning in addition to supervised learning and unsupervised learning. In reinforcement learning an agent, for example a robot, explores the space of possible strategies by performing actions and receives feedback on the outcome of the choices it takes. By using the feedback, which is called rewards, the agent can find an ideally optimal strategy of how to behave in the space [19]. With the use of reinforcement learning, robots can autonomously find an optimal behaviour by interacting with the environment. Reinforcement learning also increase the adaptability of a robotic system, which can help dealing with complex and dynamic environments [31]. In this way robots can learn and reproduce tasks that even humans cannot perform and it can find novel ways of reaching a solution of a problem [20]. Reinforcement learning can be used to solve several problems in fields like multi-agent systems, robot motion control, simulation-based optimization and action planning to mention some areas [35].

1.2 Contributions

The main contributions of this thesis are:

- A comparison of active inference and reinforcement learning algorithms for action planning on discrete state spaces with different characteristics.
- A comparison of two different implementations of active inference.
- Discussion of properties of active inference.

This thesis compares the performance of active inference and reinforcement learning on discrete state spaces with discrete actions. Also, two different implementations of active inference are tested and compared, here called active inference and sophisticated active inference. These implementations are tested on several grid worlds with different challenges and compared to value iteration and Q-learning. Sophisticated active inference is also tested and compared with a Deep Q-network

By performing the testing on the environments, properties of active inference have been found and are presented in this thesis. Both advantages and disadvantages with active inference have been discovered and the findings are explained and discussed.

1.3 Outline

The thesis is divided into seven main chapters, where the first chapter is the introduction. In Chapter 2 background theory about reinforcement learning will be presented. Here, the Markov decision processes and the Bellman equation are explained, before we dive into the reinforcement algorithms value iteration, policy iteration and q-learning. Then artificial neural networks are presented and deep q-learning is introduced. In Chapter 3 the free energy principle is presented and how it can be used in active inference to construct policies. At the end of the chapter, deep active inference and sophisticated active inference are presented.

After the theory is presented, remarks about the implementation of value iteration, Q-learning and active inference are made in Chapter 4. The tools and the environments used to assess the performance of the algorithms are also described and explained. In Chapter 5 the main results of applying the algorithms on the environments are presented. These results are discussed and compared in Chapter 6, including an assessment of active inference as an action planner and future work. Chapter 7 concludes the thesis.

2 Reinforcement learning

This chapter presents a few general aspects with reinforcement learning and several reinforcement learning algorithms such as value iteration, policy iteration and Q-learning. In addition, theory about artificial neural networks is introduced as they are important in Deep Q-learning which also is presented.

2.1 Markov decision process

A Markov decision process (MDP) is a way of modelling a sequential decision problem which has a fully observable, stochastic environment with a transition model, which has the Markov property, and additive rewards. The MDP contains a set of states S and in each state the agent has a set of actions A it can perform. In addition, it contains a transition model $T(s'|s, a)$ which represents the probability of reaching state s' if the agent performs action a is in state s, and the reward function $R(s)$ [33]. The reward function represents the rewards that are received after making a transition and is used for learning what actions should be performed in the environment. The MDP can for example be used to model a problem in an environment where an agent is in a start state and wishes to get to the end state in a a way that maximizes the reward. The solution of this problem is called a policy π, which specifies what the agent should do in every state. When there is a complete policy the agent will always know what to do next regardless of the state, and the policy that maximizes the highest expected utility is called the optimal policy, denoted π^* [33].

2.2 Bellman equation

To maximize the reward over time the highest expected utility can be found. Let an agent start in sate s and define the state S_t as the state it reaches at time t when it performs a particular policy π. Given a policy π, the expected utility of the states in the state sequence is found by

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t * R(S_t)\right] \tag{1}$$

where $0 < \gamma < 1$ is the discounting factor [33]. This expression implies that the utility of being in a particular state is the expectation of the sum of the discounted reward from the state and forward. Hence, assuming optimal actions from the agent, the utility of a state can be defined as the immediate reward the agent obtains in that state added by the expected discounted utility of the next state. This yields the Bellman equation,

$$V(s) = R(s) + \gamma * \max_{a \in A(s)} \sum_{s'} T(s'|s,a)V(s') \tag{2}$$

where the utilities of the state are solutions of the Bellman equation [33].

2.3 Value iteration

Value iteration is a way to solve the MDP with the use of the Bellman equation. This algorithm focuses on estimating the value function for every state until it converges to a limit V^*. When the value function has converged it is used to compute the optimal policy for the environment [33]. The value iteration algorithm is given in Algorithm 1 [33]. As there are n states, there are n Bellman equations and first the utilities $V(s)$ are initialized to an arbitrary value for every n states. Then, the algorithm loops through every state and calculates (2). This is repeated until the change in the value function of every state is smaller than a given error. Thus, the utilities have reached an equilibrium and the optimal policy can be found with the use of the utilities. To be able to use this method of solving MDPs the algorithm needs the complete model of the environment available, i.e. the transition model and rewards have to be known [16].

Algorithm 1 Value iteration

Require: MDP, V(s) initialized to zero for every s
  repeat
    $V \leftarrow V', \delta \leftarrow 0$
    for each $s \in S$ **do**
      $V'(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} T(s'|s,a)V(s')$
      if $|V'(s) - V(s)| > \delta$ **then**
        $\delta \leftarrow |V'(s) - V(s)|$
      end if
    end for
  until $\delta < \epsilon(1-\gamma)/\gamma$ **return** V

2.4 Policy iteration

Policy iteration is another method for solving the MDP when then complete model is available. The policy iteration is made up by two steps; the policy evaluation and the policy improvement. The algorithm for the two steps can be seen in Algorithm 2 [33]. The policy evaluation step is used for calculating the value function for a particular policy and is the first step of the policy improvement algorithm. By looking at the policy evaluation function in Algorithm 2, it is seen that this step does almost the same as the value iteration algorithm. The only difference is that

instead of calculating the value function with the action that gives the highest utility for the next state, the algorithm now follows a policy and calculates the utility for performing the actions in the policy.

The policy improvement step checks it if it is possible to find another policy that maximizes the expected utility. After evaluating the value function with policy evaluation, the algorithm iterates through every state and calculates the utility of all possible actions for a particular state s. If it finds an action that yields a higher utility than the action in the policy, it means that there exists a policy that gives a better result than the current one. If that is the case, the action in π_i is replaced with the one that gave a higher value which leads to the new policy π_{i+1}. This is repeated for every state and when the policy is improved it is evaluated with the policy evaluation step. After the evaluation step the algorithm again checks for improvements and when the policy stops improving it has reached the optimal policy [33].

Algorithm 2 Policy iteration

Require: MDP, $V(s)$ initialized to zero for every $s.\pi(s)$ initialized random for every s.
  function POLICY EVALUATION
    repeat
      $V \leftarrow V^{\pi_i}, \delta \leftarrow 0$
      for each $s \in S$ **do**
        $V^{\pi_i}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s'|s, \pi_i(s))V(s')$
        if $|V^{\pi_i}(s) - V(s)| > \delta$ **then**
          $\delta \leftarrow |V^{\pi_i}(s) - V(s)|$
        end if
      end for
    until $\delta < \epsilon(1-\gamma)\gamma$ **return** V
  end function
  function POLICY IMPROVEMENT
    repeat
      $V \leftarrow$ `POLICY EVALUATION`(π, V, MDP)
      unchanged policy $\leftarrow$ true
      for each $s \in S$ **do**
        if $\max_{a \in A(s)} \sum_{s'} T(s'|s,a)V(s') > \sum_{s'} T(s'|s, \pi_i(s))V(s')$ **then**
          $\pi_{i+1}(s) \leftarrow \text{argmax}_{a \in A(s)} \sum_{s'} T(s'|s,a)V(s')$
          unchanged policy $\leftarrow$ false
        end if
      end for
    until unchanged policy **return** π_i
  end function

2.5 Q-learning

In the last two sections we have looked at ways to solve the MDPs with dynamic programming. But it is not always the case that the model of the environment is available, i.e. that the rewards and transition probability is unknown to the agent. This means that the agent itself must explore the environment on its own and learn the effect of the actions it can take. In this case, the previous algorithms becomes insufficient as they are model-based. To solve this problem model-free reinforcement learning algorithms are used. These types of reinforcement learning algorithms do not require any prior information about the model and interacts with the environment to gather information to learn the optimal policy [16].

Q-learning is a reinforcement learning-algorithm which learns a Q-function giving us the expected utility of taking a specific action in a given state. Q-learning is a temporal difference method which means that when it takes an action, it receives the reward and makes estimates of the value of the state. In this way the algorithm can solve the prediction problem that has occurred, which means that on every step the agent learn something about the environment that it can use to estimate the value function. In Q-learning $Q(s, a)$ is used to denote the value of performing the action a in state s. The algorithm as seen in Algorithm 3 starts by initializing $Q(s, a)$ to zero for each s and a. Next, a state s is initialized as the starting state and one of the possible actions is performed. This sends the agent into a new state s' where it receives a reward. With this reward the Q-value is updated for that particular action. That is done by using the temporal difference method TD(0), where the update rule is given by

$$Q(s, a) = Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{3}$$

where α is the steps-size parameter which determines how fast the estimate is updated to changes in Q [37]. An episode of the algorithm is finished when the agent reaches the final state. To get a good estimation of the Q-value this is repeated for a number of different episodes. When the Q-values have converged we can use the values for each action in the states to find an optimal policy. The action with the highest value in each state is chosen for the policy [33]. As the algorithm is exploration insensitive the optimal policy will be found regardless of how the environment is explored as long as it can visit the state-action pairs enough times [16].

2.5.1 Exploration and exploitation

The agent now has to explore and take samples of the MDP to learn about the environment and the way the agent does this is important. If this is not done correctly, the agent may not learn the optimal policy at all. An agent that is a

Algorithm 3 Q-learning

Require: Initialize $Q(s, a) \forall s \in S$ and $a \in A, \gamma, \alpha$
  for each episode **do**
    Initialize starting state s
    repeat
      Choose an action a from state s using an exploration strategy
      Take action a, observe the next state s' and the reward r
      $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a \in A} Q(s', a) - Q(s, a))$
      $s \leftarrow s'$
    until s is a terminal state
  end for

greedy agent when learning an environment will often find a policy that is not optimal [33].

Assume there is a grid where we want the agent to get from a starting state to a goal state where the agent receives a positive reward. In the beginning, the agent will perform random actions as it has no knowledge about the environment. But when it eventually reaches the goal it knows a path that will take it to the goal state and the rewards. From now on the agent will only use this path for all the other episodes. If the environment is large enough, this path will most likely not be optimal and the agent will not be able to find the optimal path later. So instead of just exploiting its current knowledge to maximize the rewards, the agent should also explore the environment further to improve the model such that it can receive greater rewards in the future. In this way the agent can find a path that is better and is more likely to find the optimal path. Thus, the agent has to find the balance between exploitation such that it maximizes its reward and exploration to maximize the future rewards, even though this can lead to negative outcomes short term [37].

A way to balance exploration and exploitation is to use an ϵ-greedy strategy. This strategy can be implemented in several ways [37]. For example, let an epsilon have the value between zero and one. If a random number is smaller than the epsilon, the agent will choose a random action. Otherwise, it will choose the action which yields the largest Q-value. An example is to start with an epsilon equal to one and as the algorithm iterate through the episodes, the epsilon will get smaller and smaller as the agent learns more about the environment. In other words, the agent explores the environment in the beginning when it has little information and exploits its knowledge when it has more information such that it finds the optimal path.

2.6 Artificial neural networks

An artificial neural network is a way of learning structures in data based on the idea that activity in the brain consists of electrochemical activity transported by neu-

rons which are a network of brain cells. A neuron works in such a way that it sends information further when it receives a combination of inputs that exceeds a threshold. Then it will "fire" and it will send electric signals to the neighbouring neurons [33]. A simplified mathematical model of a single neuron is called a perceptron. A perceptron can be modelled by the expression

$$y = \begin{cases} 1 & \text{if } \boldsymbol{w}\boldsymbol{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $\boldsymbol{w}$ is a vector of weights, $\boldsymbol{x}$ is the input vector and b is the bias [17].

A complete artificial neural network is made by several perceptrons connected to each other. The network consists of several nodes which connect to other nodes by links. These links have a corresponding weight which will affect how strong the connection is between two nodes. The link between two nodes propagates the output from the first node to the input of the other node. Nodes are structured in layers where nodes from one layer send information to the next layer. There exists different types of neural networks, for example feed-forward networks, convolutional neural networks and recurrent neural networks. An example of a feed-forward network can be seen in Figure 1 where all nodes are connected to those in the next layer. First, the input layer receives the input information. The layer then sends the information to the next layer, the hidden layer, through links and each node in the receiving layer will compute the weighted sum of its input. After performing the computations, the nodes will use an activation function to find the output [1]. If the network consist of several hidden layers it is called a deep neural network. Here, the input is propagated through all the layers until it reaches the output layer. By using a deep neural network, more and more complex features of the input is represented and neural networks can model very complex functions [33].

For the network to yield the desired outputs, the network needs to be trained. This is done by adjusting the weights of the network and can for example be done by gradient decent. Training of a network depends on what kind of feedback the network has available. If the network has example inputs and the corresponding outputs available it is called supervised learning, while if the network only has some inputs but do not know what the corresponding outputs are it is called unsupervised learning. Assume that the network has some inputs and the corresponding outputs, where we call the outputs the target values $\boldsymbol{y}$. The task of training the network is to find the weights such that the network outputs the same as the target from a given input. The network calculates the output, which we can call $\boldsymbol{h}(\boldsymbol{x})$, from an input $\boldsymbol{x}$ in a forward phase and uses a loss function that finds the error between the output and target in a backward phase. This error is backpropagated from the output layer through the hidden layer to adjust the weights such that they fit the training data better [1]. An example of an algorithm that can be used to train a neural network is gradient decent. Here a loss function $J(\boldsymbol{w})$ is backpropagated through the hidden

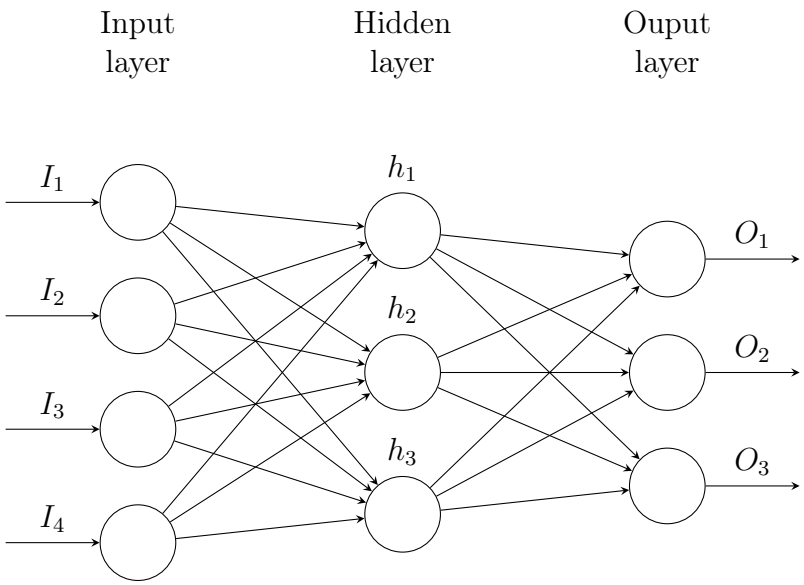


Figure 1: A feed-forward neural network.

layers and this can be exemplified through the expression

$$\boldsymbol{w} = \boldsymbol{w} - \alpha \nabla J(\boldsymbol{w}) \tag{5}$$

where α is the learning rate [1].

2.7 Deep Q-learning

To perform Q-learning in large environments with many states would require huge tables to store the Q-values. Also, it would take a lot of time to explore all the possible actions in every state. To deal with these issues the Q-learning algorithm can be combined with a deep neural network, called a Deep Q-Network.

The Deep Q-learning model uses deep neural networks to implement the Q-table. The input to the network is just the state, which is the m most recent frames of the environment stacked and the output is approximations of the Q-values corresponding to the possible actions in this state. This is different from Q-learning where the input is a state-action pair and the output is the Q-value. The network of Q-values is initialized with random weights θ. The Deep Q-Network learns by executing actions according to a ϵ-greedy policy in state s_t. An action a_t is selected and executed which results in a reward r_t and the observation of the next frame x_{t+1}. Then the state s_t, the action taken a_t and the next frame x_{t+1} is used as a representation of the next state and stored in s_{t+1}. As there is a need for several frames to get a complete picture of a state, the states are preprocessed with a function ϕ which yields the next input $\phi_{t+1}(s_{t+1})$ to the Q-function [24]. In Deep Q-learning a technique called experience replay can be used where the experience at each time step is stored in

a data set D which is called the replay memory. This means that the transition $(\phi_t, a_t, r_t, \phi_{t+1})$ is stored in D and when the Q-learning updates are performed a random set of transitions from the data set D is sampled, $(\phi_j, a_j, r_j, \phi_{j+1})$, and applied to the target function

$$y_j = \begin{cases} r_j & \text{if episode terminates as step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases} \tag{6}$$

The reason for using experience replay is to break the correlations between samples. If the network learned from the samples as they happen, there will be strong correlations between the data. Another reason for using experience replay is to make sure that we train the network on the whole environment. If the learning was performed with consecutive samples, the samples would for example have been dominated by samples on the right-hand side of the environment if the first maximizing action was to move to the right [24].

In the equation for the target function (6), $\hat{Q}$ is used and not the network Q. The network $\hat{Q}$ is called the target network and is only used when calculating the targets, y_j. Every C updates, the weights in the main network Q are cloned to the target network $\hat{Q}$. Otherwise the weights of the target network, here represented by θ^-, are constant. The reason for using a target network is to keep the weights constant when the target value is found. If they where not constant an update in $Q(s_t, a_t)$ would also update the target y_j which could lead to oscillations or divergence in the policy. By using the target network there is a delay between when Q is updated and when this update affects y_j. This makes oscillations or divergence in the policy less likely [24].

After the target y_j is calculated, a gradient decent step is performed on the loss function

$$L(\theta_j) = (y_j - Q(\phi_j, a_j; \theta))^2 \tag{7}$$

with respect to θ, which gives the update function of the weights

$$\theta = \theta + \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i) \tag{8}$$

The update of the weights is performed for M number of episodes until the network Q is ready to be used [24].

2.7.1 Deep Deterministic Policy Gradient

Deep Q-learning networks can only handle discrete and low-dimensional action spaces. To solve these issues we can use a method called deep deterministic policy gradient which can be thought of as Deep Q-learning in continuous action spaces.

This method is able to handle a continuous action set by using a technique that uses both Q-learning and policy gradients. The deep deterministic policy gradient consists of two neural networks, the actor and critic. The actor is also called a policy network that takes the state as input and outputs the optimal action. The optimal action is found by finding the action that approximately maximizes Q corresponding to the state. The action is executed and stored in a replay buffer with the corresponding reward, the new state and current state. Then, the critic samples a random batch from the replay buffer and calculates the Q-value and the target value with the help of a target network. The weights to the critic are updated by minimizing the loss with a use of a loss function similar to the one in (7). The actor is then updated by using policy gradient [21].

3 Active inference

In this chapter the basic theory behind active inference and how an agent can find a policy that ensures low variational free energy is explained. The free energy principle is also presented as a theoretical basis for active inference.

3.1 The free-energy principle

An important characteristic of biological systems is that they maintain their states in a changing environment. The free-energy principle is a way of describing how any self-organizing system remains inside the set of restricted states within its environment. Through minimization of the free energy the systems can keep themselves within the set of states that is defined for their phenotype. As a system has fa limited amount of physiological and sensory states it can be in, there is a high probability that a system will be in these states and a low probability that it is in the remaining states. The states with high probability are not surprising for a system, while the states with low probability are surprising. Free energy is defined as an upper bound on surprise, and by minimizing free energy the system will also minimize surprise and maintain its states. Thus, the free-energy principle can explain why and how biological systems act, percept and learn to maintain their states in a changing environment [9].

This principle says that agents must avoid surprise in order to remain in their own states. One problem is that the agent does not know if what it is sensing is surprising or not. To solve this issue the free energy is introduced. Free energy is an information theory measure that limits the surprise given a generative model. To be able to be surprised agents need to be able to receive sensations. These sensations are represented by the sensory input s received by the agent and these are generated by the external environment which is represented by $\vartheta \supset x, \theta, \gamma$ where x are the external hidden states. The factors θ and γ control the amplitude of the noise in the external states w and the noise in the sensations z respectively [9]. The hidden states are the states of the environment relevant to the agent. For an agent trying to find its way between two locations a hidden state can for example be the location of the agent. That these states are hidden means that the agent cannot observe them directly, but they generate sensory inputs to the agent such that it can observe them indirectly. We assume that the hidden states evolve according to the equation

$$\dot{x} = f(x, a, \vartheta) + w \tag{9}$$

and that the sensory sampling can be described by the probabilistic mapping

$$s = g(x, \vartheta) + z \tag{10}$$

as described by Karl Friston in [9].

Because the external environment cannot be directly observed, the agent has to make a model of the environment. For example if the agent is a human, it makes a model of the environment in its own brain. This is done by using Bayesian inference [8], where the brain tries to infer the probability of the hidden states with the help of the sensory input. The model can be represented by the probability density $p(\vartheta|s)$ and is calculated with the use of Bayes' theorem

$$p(\vartheta|s) = \frac{p(s|\vartheta) * p(\vartheta)}{p(s)} \tag{11}$$

In this equation, $p(s|\vartheta)$ is the probability distribution of the observation s given the causes of the observations ϑ, $p(\vartheta)$ is the prior belief about the causes of the observation and $p(s)$ is the prior belief about the observation. The denominator $p(s)$ is represented as

$$p(s) = \int p(s|\vartheta) * p(\vartheta) d\vartheta \tag{12}$$

which implies that we need to average over all the possible states that could have caused the sensory input. This is problematic as there are many possible causes for a sensory input. It is even hard to know them all and will take a lot of time to compute. The solution to this is to find a density that can approximate $p(\vartheta|s)$ [2]. This can be done by finding a recognition density $q(\vartheta|\mu)$ and use this as an approximation of $p(\vartheta|s)$. The recognition density is a function of the causes of the sensory input, i.e. the hidden states ϑ and the internal states in the brain μ. It is also called the approximating conditional density and is the internal model of the external world that represents probabilities of what caused a certain sensation [9].

The free energy principle is about minimizing surprise so we wish to minimize the prediction error of the internal model in relation to the external environment. As we have an internal model $q(\vartheta|\mu)$ of the world the brain can predict what is going to happen. To minimize the prediction error we wish to get the recognition density as close to the posterior density $p(\vartheta|s)$ as possible. To find the difference between the internal model $q(\vartheta|\mu)$ and $p(\vartheta|s)$ the Kullback-Leibler divergence is used [22]. The Kullback-Leibler divergence measures the divergence between two probability densities and is given by

$$D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta|s)} d\vartheta \tag{13}$$

As it is problematic finding the posterior distribution $p(\vartheta|s)$, it can be rewritten as

$$p(\vartheta|s) = \frac{p(\vartheta, s)}{p(s)} \tag{14}$$

By inserting this into the formula for Kullback-Leibler divergence and using the fact that $\ln(ab) = \ln(a) + \ln(b)$, the Kullback-Leibler divergence between the recognition density and the posterior density is given by

$$D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta + \ln p(s) \int q(\vartheta|\mu) d\vartheta \tag{15}$$

The integral in the rightmost term equals to 1 [22]. The variational free energy is given by

$$F(s, \mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta = -\int q(\vartheta|\mu) \ln \frac{p(\vartheta, s)}{q(\vartheta|\mu)} d\vartheta \tag{16}$$

as defined in [12]. Here it is seen that the variational free energy is a function of the recognition density and the generative model $p(\vartheta, s)$. The generative model is a probability density in the brain that generates samples from the sensors and the corresponding causes. In other words, it encodes the internal probabilistic model of the environment [9]. By shuffling the expressions in (15) we get that the free energy is represented by

$$F(s, \mu) = D_{KL}[q(\vartheta|\mu)||p(\vartheta|s)] - \ln p(s) \tag{17}$$

The term $-\ln p(s)$ is the negative logarithm model evidence and is called the surprise [9]. Because the surprise is represented as the negative natural logarithm of the probability distribution of s, a high probability of a sensory input gives a low surprise and vice versa. As the Kullback-Leibler divergence is always larger than zero we get that

$$F(s, \mu) \geq -\ln p(s) \tag{18}$$

which means that the variational free energy is an upper bound on the surprise. Hence, if variational free energy is minimized, the surprise will also be minimized [27].

3.2 Minimizing free energy

To minimize the free energy, agents have two things they can change, the sensory input and the recognition density. They can either change the recognition density by changing the internal states or act on the environment such that the sensor inputs changes.

3.2.1 Minimizing with perception

When the variational free energy is minimized by changing the recognition density we wish to optimize the internal states such that the recognition density $q(\vartheta|\mu)$ gets as close as possible to the posterior $p(\vartheta|s)$. This will reduce the divergence between the two densities which means that the brain obtains a more accurate model of the environment. By doing this the perception is improved, which makes the agent better at inferring the causes of the sensory input. To find the internal states that minimize the variational free energy we have to solve the equation

$$\mu = \underset{\mu}{\operatorname{argmin}} F(s, \mu) \tag{19}$$

This way of minimizing the variational free energy corresponds to Bayesian inference [13].

3.2.2 Minimizing with action

The other way of minimizing the variational free energy is to take an action that changes the observations such that the sensory inputs corresponds to our representation of the environment. To see this, the variational free energy is rewritten

$$F(s, \mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta, s)} d\vartheta = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(s|\vartheta)p(\vartheta)} d\vartheta \tag{20}$$

with the use of the definition of conditional probability. By using that $\ln(\frac{a}{b}) = \ln(a) - \ln(b)$ we get that

$$F(\tilde{s}, \mu) = \int q(\vartheta|\mu) \ln \frac{q(\vartheta|\mu)}{p(\vartheta)} d\vartheta - \int q(\vartheta) \ln p(s|\vartheta) d\vartheta \tag{21}$$

Eventually, this can be represented as

$$F(s, \mu) = D_{KL}[q(\vartheta|\mu)||p(\vartheta)] - \langle \ln p(s|\vartheta) \rangle_q \tag{22}$$

Here, the variational free energy is represented by the complexity minus the accuracy, where the complexity is the divergence between the recognition density and the prior, and $\langle \cdot \rangle_q$ is the expected value of $\ln p(s|\vartheta)$ under q [9]. The accuracy is given as the surprise of sensations that are expected under the recognition density. To minimize the variational free energy, the agent will select actions such that it receives the sensory inputs that it expects in the recognition density. By doing this, the agent reduces the surprise from the sensory input which in turn maximizes the accuracy. In essence, this means that minimizing variational free energy will maximize the

accuracy of the predictions made by the agent. The action that minimizes the variational free energy is given by

$$a = \underset{a}{\operatorname{argmin}} F(s, \mu) \tag{23}$$

as seen in [13].

3.3 Active inference

By combining the two methods of minimizing free energy, a policy that ensures low variational free energy of an agent can be constructed. While perception tries to obtain better predictions, the action tries to satisfy these predictions such that the level of surprise is low. This is called active inference [13].

Active inference gives a set of rules for finding and understanding how autonomous agents behave in situations where they have to make a decision under uncertainty. By using the free energy principle to describe the properties of the agent and its corresponding environment and minimizing expected free energy over time, Bayes-optimal behaviour can be attained [34].

Active inference uses minimization of the variational free energy to maximize the generative model evidence. The agent minimizes with perception to get a representative internal model of the environment which gives the agent the possibility of perceiving the environment. In this way it can make inferences about the world. But, we would also like the agent to take actions, which corresponds to the active part in active inference. In addition to minimizing the variational free energy, an active inference agent will also minimize the expected free energy, denoted G. The expected free energy is dependent on what we think the observations are going to look like in the future, and can help the agent to figure out which series of action it should perform to reach a certain state [34].

The agent can now plan the actions it should take in the future and in this way make sure that the preferred outcomes are realised. This ensures that the surprise is avoided on average [4]. The expected free energy is a function of policies, which is defined in the same way as in reinforcement learning. The agent evaluates the expected free energy by using policies to find plausible outcomes which are yet to be observed. Before we find the expression for the expected free energy, we rewrite the expression for the variational free energy in equation (16) such that it depends on time τ and policy π:

$$F(\tau, \pi) = \sum_{\vartheta_\tau^\pi} q(\vartheta_\tau|\pi) q(\vartheta_{\tau-1}|\pi) \ln \frac{q(\vartheta_\tau|\pi)}{p(\vartheta_\tau, s_\tau|\vartheta_{\tau-1}, \pi)} \tag{24}$$

The expected free energy G can be derived by using this expression of the variational free energy [34]. By adding the beliefs about future outcomes $q(s_\tau|\vartheta_\tau, \pi)$ and

condition the generative model $p(\vartheta_\tau, s_\tau|\vartheta_{\tau-1}, \pi)$ on the desired states C instead of a policy, gives the expression

$$G(\tau, \pi) = \sum_{\vartheta_\tau, s_\tau} q(s_\tau|\vartheta_\tau, \pi) q(\vartheta_\tau|\pi) q(\vartheta_{\tau-1}|\pi) \ln \frac{q(\vartheta_\tau|\pi)}{p(\vartheta_\tau, s_\tau|\vartheta_{\tau-1}, C)} \tag{25}$$

which also can be found in [34] with different symbols. The desired states C are the preferred observations, i.e. the goal states the agent wants to be in.

The expected free energy can be rewritten and represented in many different ways. A compact way of expressing the quantity is by expressing it as expected cost and expected ambiguity:

$$G(\tau, \pi) = D_{KL}[q(s_\tau|\pi)||p(s_\tau)] + E_{\tilde{q}}[H(p(s_\tau|\vartheta_\tau))] \tag{26}$$

Here $\tilde{q}$ is given by $\tilde{q} = q(s_\tau, \vartheta_\tau)$. The expected cost, which is represented as the Kullback-Leibler divergence, is the difference between the predicted outcomes $q(s_\tau|\pi)$, and the preferred outcomes as $C = \ln p(s_\tau)$. The expected ambiguity is the expected entropy of the likelihood under the predicted states [11].

Now that we have a way of computing the free energy of beliefs in the future with the help of policies, we wish to infer the optimal policy such that agent can behave optimally. To be able to find the best actions the approximate posterior of policies $q(\pi)$ has to be found. The approximate posterior of policies is given by the sum of the negative expected free energy along the trajectory created by a particular policy [4]. This gives us the expression

$$q(\pi) = \sigma(-G(\pi)) \tag{27}$$

where σ is the softmax-function and $G(\pi) = \sum_\tau G(\tau, \pi)$. With the help of the approximate posterior of policies $q(\pi)$ the agent can select the most plausible action by finding the most likely action under every policy [4]. This provides the expression

$$u_\tau = \underset{u}{\operatorname{argmax}}(\sum_{\pi} q(\vartheta|\pi) q(\pi)) \tag{28}$$

In this way the action the agent should do for every time step can be obtained. To summarize, the agent evaluates the expected free energy for every possible policy, which gives a way of quantifying how good a policy is relating to prior preferences. Then, the expected free energies are uses this to find an approximate posterior of the policies $q(\pi)$ [4]. Furthermore, this approximate posterior is used to select the best action at every time step. After performing the action, the agent receives an observation and uses this to infer the posterior beliefs about the hidden state $q(\vartheta|\mu)$. This is performed by minimizing the variational free energy such that the recognition density approximates the true posterior as good as possible.

3.4 Active inference on larger spaces

As described in the previous section, active inference calculates the expected free energy of every possible policy. When the size of the state space nS and the size of the action space nA get large, the number and length of the policies becomes large. The length of the policies depends on the size of the state space, while the number of policies needed to cover all the possible policies can be calculated with the expression nA^{nS}. When these sizes get large, the time used by the algorithm to calculate the expected free energy for every policy will be very long. To reduce the run time and to make active inference handle large states and policies, it is possible to use active inference in combination with neural networks or to find the expected free energy in a recursive way.

3.4.1 Deep active inference

In deep active inference, deep neural networks are used to approximate the densities. The posterior distribution can be represented as

$$q(\vartheta) = \prod_{t=1}^{T} q(\vartheta_t|\vartheta_{t-1}, a_{t-1}, s_t) \tag{29}$$

where a is actions. The single time step posterior distribution $q(\vartheta_t|\vartheta_{t-1}, a_{t-1}, s_t)$ is approximated by a network $q_\phi(\vartheta_t|\vartheta_{t-1}, a_{t-1}, s_t)$. Also, the likelihood model $p(s_t|\vartheta_t)$ is approximated by the network $p_\xi(s_t|\vartheta_t)$ and the prior $p(\vartheta_t|\vartheta_{t-1}, a_{t-1})$ is approximated with $p_\theta(\vartheta_t|\vartheta_{t-1}, a_{t-1})$ as a network [3].

The networks work together in a similar way as a variational autoencoder (VAE). In the context of a neural network, a VAE is consisting of an encoder, a decoder and a loss function. Both the encoder and decoder is a neural network. The encoder takes the input, and outputs a hidden representation. The decoder receives this hidden representation as an input and outputs the parameters of the probability distribution of the data. In this case $q_\phi(\vartheta_t|\vartheta_{t-1}, a_{t-1}, s_t)$ is the encoder which encodes the high-dimensional input and outputs the parameters of a multivariate Gaussian as a hidden representation. Then the decoder $p_\xi(s_t|\vartheta_t)$ gets the hidden representation as an input and outputs the reconstructed input [15].

These networks are trained by using a representation of the free energy as the objective function in equation (30),

$$\forall t \min_{\phi,\theta,\xi} -\log p_\xi(s_t|\vartheta_t) + D_{KL}(q_\phi(\vartheta_t|\vartheta_{t-1}, a_{t-1}, s_t)||p_\theta(\vartheta_t|\vartheta_{t-1}, a_{t-1})) \tag{30}$$

After training, the neural networks can be used to infer what action the agent should take next. By using $p_\xi(s_t|\vartheta_t)$ and $p_\theta(\vartheta_t|\vartheta_{t-1}, a_{t-1})$ trajectories for policies can be constructed, and the expected free energy can be calculated [3]. Then, the expected free energies are used in the process of choosing the best action, just as for active inference without deep neural networks.

3.4.2 Sophisticated active inference

Another way of handling the large policy spaces is sophisticated active inference. This sophisticated form of active inference does the calculations of the expected free energies in a recursive way, which implements a deep search tree over the outcomes and actions in the future. The term 'sophistication' comes from economics and an agent that is sophisticated has beliefs about its own beliefs or another agent beliefs [10]. In the context of active inference this means that one implements a deep tree search over actions and outcomes in the future, where this search is done over sequences of belief states. The agent is now considering beliefs about what it would believe if it did a certain action instead of beliefs of what would actually happen [10].

As we now have a recursive formulation of the expected free energy, the expected free energies have to be calculated in a finite temporal horizon. The recursive formulation is given in (31) for $\tau = T - 1$ and in (32) for $\tau = 1, ..., T - 2$ [29].

$$G(a_\tau|\vartheta_\tau) = G(a_{T-1}|\vartheta_{T-1}) = D_{KL}(q(\vartheta_\tau|a_{T-1}, \vartheta_{T-1}||C(\vartheta_T)) \tag{31}$$

$$G(a_\tau|\vartheta_t) = D_{KL}(q(\vartheta_{\tau+1}|a_\tau, \vartheta_\tau)||C(\vartheta_{\tau+1})) + E_q(G(\text{nextstep})) \tag{32}$$

The expectation of $G(\text{nextstep})$ is given by

$$E_q(G(\text{nextstep})) = E_{q(a_{\tau+1},\vartheta_{\tau+1}|s_\tau,a_\tau)}(G(a_{\tau+1}|s_{\tau+1})) \tag{33}$$

as seen in [29]. When the expected free energies are calculated, a distribution for action selection is found by the expression

$$q(a_\tau|\vartheta) = \sigma(-G(U|\vartheta)) \tag{34}$$

where U is the set of actions. This approximate posterior of actions given the hidden states gives each action a probability of how plausible that action is. At every time step an action is sampled from the distribution. In this way the expected free energy of a potential action a_τ is a combination of the expected free energy at the current time step plus the average expected free energy of all future actions [29]. Instead of calculating the expected free energy for every possible policy, the expected free energy is calculated recursively such that the calculation of the expected free energy of an action in a specific state can use that it already has calculated the expected free energy for future actions.

4 Implementation

In this chapter remarks about the implementation of active inference, sophisticated active inference, value iteration, Q-learning and Deep Q-networks are made. The environments used for testing the algorithms are also introduced. To see if the implementations of active inference can perform action planning we wish to see how they work on different simulated environments.

4.1 Tools

The programming language Python was used for implementation of the algorithms and evaluation of performance. In addition, the numerical library NumPy was used for scientific computing and the OpenAI Gym library was used in order to get access to the Frozen Lake environment [26]. Also, Matplotlib was used to plot results.

For the implementation of neural networks in Deep Q-learning PyTorch was used. PyTorch is a deep learning research platform for Python that provides tensor computation and deep neural networks. With the use of Pytorch, neural network architecture can easily be implemented. Features like network models, optimizers and automatic differentiation to compute gradients are implemented and ready to be used. Pytorch also allows bidirectional exchange of data with other external Python libraries, which makes it possible to use PyTorch in combination with libraries like NumPy [28].

4.2 Frozen Lake

Firstly, the algorithms are simulated with the help of the Frozen Lake environment using the OpenAIGym library, which is a toolkit for doing experiments on reinforcement learning algorithms made by OpenAI. The Frozen Lake environment is a discretized grid world of $4x4$ states where some of states are walkable and some are holes where the agent will fall in to the water. To move in the environment the agent can choose between four actions: Up, Right, Down and Left. The task of the agent is to get from a start state to the goal state without falling into the water. If the agent reaches the goal state it is rewarded with a value of 1 and 0 otherwise. But, if the agent walks into a hole the episode ends [26]. A representation of the environment can be found in Table 1. Active inference uses preferred outcomes to find out where to move instead of rewards. Thus, the density of the preferred outcomes is manually set before the simulation where the goal state was set to a probability of 1 and the other states to 0.

The Frozen Lake environment has the property that it can either be slippery or not slippery. If it is slippery the environment is stochastic and the agent will not neces-

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Table 1: Frozen lake environment. S is starting state, F is frozen lake, H is hole and G is goal state.

sarily move in the direction as intended. It will move in the preferred direction with a probability of 0.33, move to the left of the preferred direction with a probability of 0.33 or move to the right with the same probability. If it is not slippery the agent will move in the desired direction with a probability of 1. In this thesis a slippery Frozen Lake environment is referred to as the stochastic case, while a not slippery environment is referred to as the deterministic case.

Active inference, sophisticated active inference, value iteration and Q-learning are used to calculate a policy in the Frozen Lake environment. The performance is tested by using the policies created by the algorithms to see how many times an agent reach the goal state.

4.2.1 Custom environment

It is also possible to make a custom grid in the Frozen Lake environment. When defining a custom map, it is possible to define the start position, goal position and the positions of the holes. To test the algorithms on a larger environment, a custom map of $6x6$ is constructed and tested in the stochastic case. The map can be seen in Table 2.

S	F	F	F	H	F
F	F	F	F	F	F
F	F	H	F	F	F
F	F	F	F	F	F
H	F	F	F	F	H
F	F	H	F	F	G

Table 2: Custom Frozen lake environment. S is starting state, F is frozen lake, H is hole and G is goal state.

The reason for implementing this specific environment is to see how the active inference algorithms compares to value iteration and Q-learning on a larger environment. It is also used to see how the resulting policy of active inference depends on the length of the policies used to compute the expected free energy. There are also placed out holes to make the action planning more difficult, but it is possible for the agent to

avoid them even in the stochastic case. It is thought that it is interesting to see if the algorithms manage to find policies that avoid every hole.

Active inference, sophisticated active inference, value iteration and Q-learning are also in this environment used to calculate a policy and tested in the same way as for the Frozen Lake-environment.

4.2.2 Value iteration

In the implementation of the value iteration algorithm a discount factor of 0.99 is used as the rewards of the future states are important. When the change in the value function is smaller than ϵ the iteration is stopped, and this error quantity ϵ is set to 0.01.

4.2.3 Q-learning

In the implementation of the Q-learning algorithm a discount factor of 0.99 and learning rate of 0.1 is used. The number of episodes is set to 10000. For the action selection in the algorithm an epsilon-greedy approach is used and the epsilon is given by $e^{(-0.001*i)}$ where i is the number of episodes.

4.2.4 Active inference

The implementation of active inference is done with the help of the pymdp-environment hosted on the infer-actively GitHub organization, which is a package for simulating active inference agents in MDP environments [14]. Active inference is in this thesis used to find a policy by iterating through every state and extracting the state with the lowest accumulated expected energy by using the approximate posterior. The expected free energies are calculated with the use of policies and the length of the policy has to be defined before execution. In the Frozen Lake environment the expected free energies are calculated with a policy length of 6, while for the custom environment policies of different length are tested.

It is possible for the active inference agent to learn the transition matrix B and the likelihood matrix A by exploring the environment, but for simplicity they where defined before training. As the probabilities of how the agent moves are known, the transition matrix is calculated beforehand. Also, as we are in a grid world it is assumed that the agent has full transparency over its own location. This means that the likelihood matrix is set to the identity matrix.

4.2.5 Sophisticated active inference

The active inference algorithm was first calculated with a customization of the pymdp-environment as explained in the section above. With stochastic transitions the active inference model computed a worse policy than the reinforcement learning algorithms. As the results in [29] for a fully observable stochastic environment with known transition dynamics show that active inference performs better or equal than Q-learning, these results seemed suspicious. Even though the environments are not equal, these results give a reason to expect that also in this Frozen Lake-environment the active inference agent should give similar performance. As it was not found any bugs in the code by debugging, sophisticated active inference is also implemented on the Frozen Lake environments to see if it reaches a better policy.

When calculating the expected free energy for $\tau = 1, ..., T-2$ the expression for posterior estimates $q(\vartheta_{\tau+1}|a_\tau, \vartheta_\tau)$ is used in the Kullback-Leibler divergence. As the environment is a discrete grid world we assume that the states are observable for the agent. Thus, the agent can access the states through the transition dynamics, which means that the posterior estimates can be directly found through the transition matrix B, such that $q(\vartheta_{\tau+1}|a_\tau, \vartheta_\tau) = B$ [29].

After the calculation of the approximate posterior of the actions given the hidden state $q(a_\tau|\vartheta)$, the agent uses the approximate posterior to select the action that is assigned the highest probability. This is done for every state and in this way the policy gets constructed. In addition, sophisticated active inference calculates expected free energy in a recursive manner over a time horizon. Different time horizons are tested to see how the performance of the policy changes with the length of the time horizon.

4.3 Simplified docking problem

After testing active inference and reinforcement learning algorithms on the Frozen Lake environments, it was suggested that it would be interesting to see if active inference is able to solve the docking problem presented in [5]. The docking problem deals with the case of guiding a maritime vessel from outside of a port to a docking space while avoiding traffic and obstacles. In [5] reinforcement learning methods such as Dyna Q-learning and Deep Q-Networks were applied to a marine vessel as a high-level support system such that the autonomy of the vessel was increased. In the thesis it was shown that an agent using Deep Q-Networks is able to learn the environment while guiding the vessel and avoid the traffic.

As a beginning of implementing sophisticated active inference and Deep Q-learning in the docking problem, a simplified version of the simulation environment in [5] is made. The original simulation environment can be seen in Figure 2.

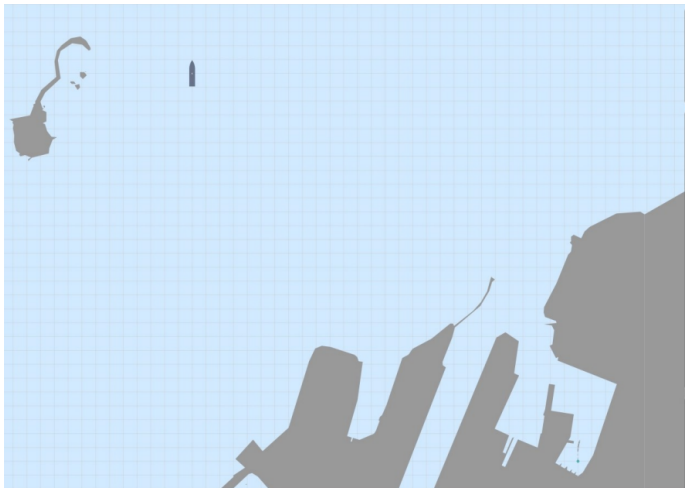


Figure 2: The simulation environment for the docking problem, from [5].

A simplified version of this environment is made by discretizing it into a grid of 30x30 cells. Only the part of the environment to the right of the boat seen to the upper left was used in the discretization. Then a 900x1 array is constructed with ones and zeros, where the ones represented an obstacle, while the zeros represented an open state. The array is constructed in a way such that every 30 states represented a row of the discretized grid world. The array is seen in Figure 3 and is an attempt of getting a simple representation of the environment in Figure 2. We define the port area as the area where the obstacles are present. The port entrance is defined as the area to the right where the agent has the possibility of moving inside the port area and towards the goal in the lower right corner. Here, one of the states is represented by the number 2. This is the goal state, which we want the agent to reach. The ones are highlighted to easier distinguish them from the zeros.

The discretized environment is made with the help of the Gym environment as done in [18]. To keep track of which state the agent is located in, a state vector of size 900x1 is used. In addition to having a state vector, the environment also keeps track of where the agent is in matrix form. By transforming the state vector to a 30x30 matrix and defining the x-direction as the horizontal axis and y-direction as the vertical axis, the agent can also keep track of where it is in the environment in x- and y-coordinates. When the agent arrives at a state, the corresponding position in the state vector is set to one while the rest is set to zero. The agent has the action set as seen in Figure 4 in addition to be able to stand still, which gives the agent nine possible actions to perform. When an action is made in a certain state it takes a step to the next state where it again chooses the next action. To guide the agent in the environment Deep Q-learning and sophisticated active inference are implemented.

The problem considered here is only a simplified version of the docking problem. There are several ways of defining the action space that are more realistic to a

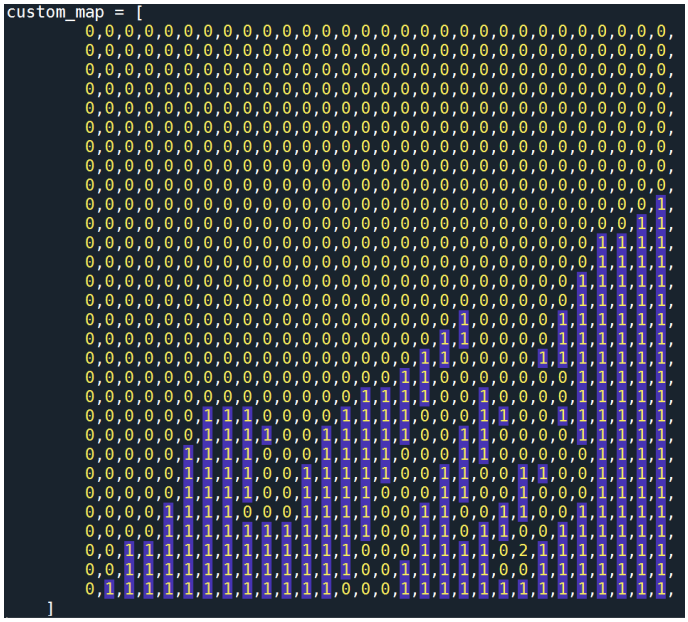


Figure 3: The discrete simplified docking environment.

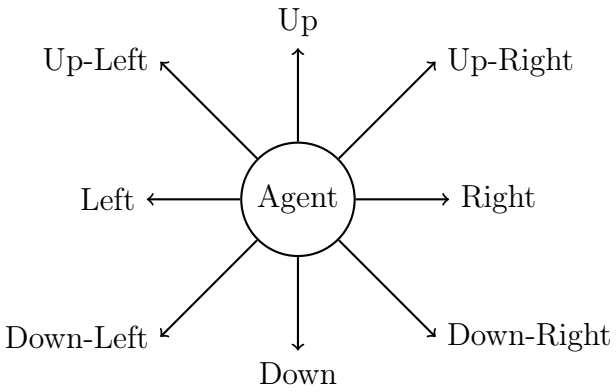


Figure 4: Action set of the agent.

marine vessel, but here it is made simple. Also, as a marine vessel has to go into the docking in a certain pose, the agent should also keep track of its pose and make sure that it enters the docking in the correct manner. This was also omitted to simplify things.

4.3.1 Deep Q-learning

As the number of states has increased drastically from the Frozen Lake environment, a Deep Q-Network is used to represent the Q-values. The network is implemented with linear layers and ReLU as the activation function. The inputs to the network are the state vector that represents the state the agent is located in, while the output is the Q-values of the possible actions corresponding to the state. The network was

tested with different structures. The numbers of hidden layers ranged between one and three, and these were tested with different number of nodes per layer. One combination that gives satisfactory performance is two hidden layers with 100 nodes each and it was decided that this structure would be used for this task.

The Deep Q-Network is also implemented with experience replay and a target network as explained in Section 2.7. The memory size of the replay memory is set to 1000 and a batch size of 200 is used to train the network. Regarding the target network, it is initialized to the same weights as the main network and after that updated every 50 iterations.

Firstly, the Deep Q-Network agent is tested on a simple environment to check if it is able to get from a starting point to a goal with occasional obstacles like in the Frozen Lake environments. The training strategy is that the agent moves around until it finds the goal or hits an obstacle. Then it receives a reward of 2 for hitting the goal and a reward of -1 for hitting an obstacle. The training of the agent is implemented with an ϵ-greedy strategy to make sure that it explores the environment even though it has found the goal. This approach was tested for environments of size 10x10, 20x20 and 30x30, and the agent finds the goal efficiently and trains the network accordingly in every case. After training the agent, it is put out in different position to check if it manage to reach the goal, and it would always reach the goal.

Then, the Deep Q-Network agent is tested on the environment in Figure 3. Now, the agent has troubles finding the goal consistently. The goal is located in the bottom right corner, in column 22, row 27 by considering the environment in Figure 3 as a matrix. As the goal is quite hard to find the agent uses very long time to find it. It would probably find it with unlimited time, but as it did not reach the goal within 500 episodes at a consistent rate it was deemed that the training strategy had to be changed. A solution to this problem is to give the agent information about if it got closer to the goal or not. After taking an action, the absolute value of the distance between the x-coordinate of the new state and x-coordinate of the goal state is calculated. The same is done for the y-coordinate of the new state and the y-coordinate of the goal state. Furthermore, the distances are added together and this process is repeated for the old state. If the absolute value of distance between the new state and goal state is smaller than the absolute value of distance between the old state and goal state, the agent is deemed closer to the goal. If the agent moves towards the goal it receives a reward of 0.1. The agent also receives a reward of -0.1 for moving further away from the goal such that the agent cannot move away from the goal and then towards the goal to accumulate positive reward. This made the agent find the port area consistently, but it still has problems finding the goal. A possible reason for this is that the docking area is quite complex and it is hard for the agent to find the goal.

To deal with this problem it is suggested in [5] that the training should be split in two scenarios. In the first scenario, the agent is supposed to move from the starting position in the upper left corner to the port entrance. The goal in this scenario is

located in the position to the right of uppermost one in the middle of the Figure 3, which is in column 20, row 15. Then, the second scenario consists of taking the agent from the port entrance, where the first scenario ended, to the goal position.

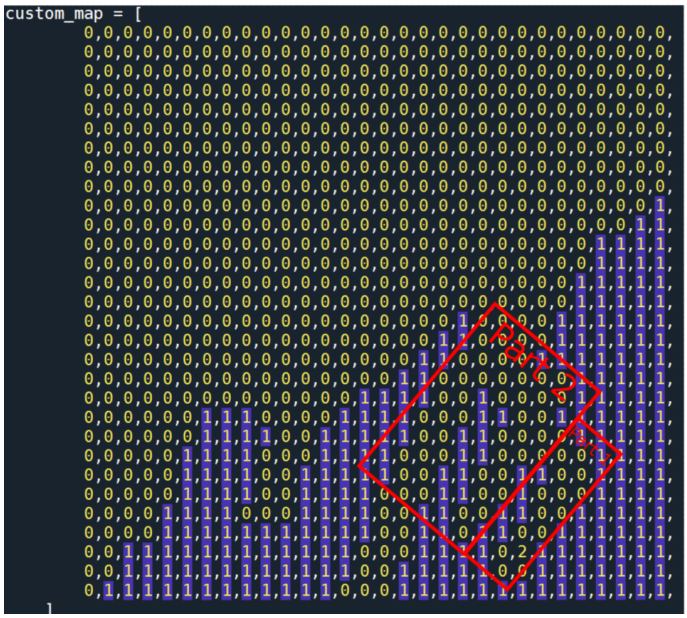


Figure 5: The discrete simplified environment with a visualization of the docking split into two parts.

After implementing these training scenarios, the agent still has trouble finding the goal in the second scenario. The agent does not manage to get into the rightmost part of the docking area, but it keeps crashing into the walls splitting the area. In [5], it was recognized that non-convexity was a problem, and it seems that this a problem in the simplified environment as well. Thus, the docking area are split into two parts as convex as possible and the training areas can be seen in Figure 5. The agent is first trained in the area that consists of the goal. When it has hit the goal 30 times, the agent is moved to a starting position in the second area and it tries to find the goal from here. As the same neural network is trained in both areas and the agent is able to find the goal in the first area, the agent is also able to get to the goal when starting in the second area.

4.3.2 Sophisticated active inference

In the implementation of the active inference agent on the Frozen Lake-environments every possible policy is used to find the expected free energy. This is possible for small state and action spaces, but as we now want to find the goal state in a 30x30 discrete environment, the number of policies becomes enormous. Thus, only sophisticated active inference is implemented in this environment and not the other implementation of active inference which was implemented on the Frozen Lake-

environment. Most of the remarks made about the implementation and behaviour of sophisticated active inference in Section 4.2.5 also apply in this setting.

However, a difference is that instead of finding the policy, the agent is run on an environment for T time steps and at every step it samples an action from $q(a_\tau|\vartheta)$. In this implementation this is done by choosing from the set of actions which are assigned the probabilities from the approximate posterior. This means that an action $a_1$in state ϑ is selected with the probability $q(a_{1,\tau}|\vartheta)$. When an action is sampled, the agent performs this action and updates the state. This is performed until the agent reaches the goal or an obstacle.

5 Results

In this chapter we will look at the results obtained with the different algorithms. First, the results of active inference, sophisticated active inference, Q-learning and value iteration from the default Frozen Lake environment are described before the results from the custom Frozen Lake environment are presented. Lastly, the results of Deep Q-network and sophisticated active inference in the simplified docking problem are presented.

5.1 Frozen Lake

First, the algorithms are tested when it is not slippery. The value iteration, active inference and sophisticated active inference algorithms yielded the same policy which can be seen in Table 3.

Down	Right	Down	Left
Down	H	Down	H
Right	Down	Down	H
H	Right	Right	G

Table 3: Policy of the Frozen Lake environment when it is not slippery, calculated with value iteration, active inference and sophisticated active inference.

The Q-learning algorithm produced the policy as seen in Table 4.

Right	Right	Down	Left
Down	H	Down	H
Right	Right	Down	H
H	Right	Right	G

Table 4: Policy of the Frozen Lake environment when it is not slippery, calculated with Q-learning.

This policy is the same as the one in Table 3, except from two states. As the policies represents two different ways to the goal with the same amount of steps from the starting state, the two policies are equally good.

These policies will make the agent reach the goal state every time as we have deterministic transitions. The agent will always take a step on the frozen parts of the lake and never walk into a hole. In other words, the agent will always reach the goal state as long as it starts on a frozen part of the lake.

Second, the algorithms were run on the Frozen Lake environment when it is slippery, i.e. the transitions are stochastic. Now, the value iteration and Q-learning algorithm calculated the policy given in Table 5.

Left	Up	Up	Up
Left	H	Left	H
Up	Down	Left	H
H	Right	Down	G

Table 5: Policy of the Frozen Lake environment when it is slippery, calculated with value iteration and Q-learning.

To see how well this policy performs on the environment, we checked how many times the agent succeed to reach the goal out of 100 episodes using the policy. As the transitions are stochastic this will yield a slightly different result every time, so we do this 1000 times and find the average. With the use of the policy in Table 5, the agent reaches the goal in 74.1 of the 100 episodes on average. As there is a possibility of getting into a hole from the state in row 2 column 3, the agent will not always reach the goal.

In this environment, the active inference algorithm calculated a policy which is different from the policy in Table 5 in the first and third state in the first row. The policy can be seen in Table 6.

Down	Up	Left	Up
Left	H	Left	H
Up	Down	Left	H
H	Right	Down	G

Table 6: Policy of the Frozen Lake environment when it is slippery, calculated with active inference.

We can check how the policy calculated by the active inference performs by doing the same performance measure as done for the policy calculated by the reinforcement learning algorithms. In this case the the agent reach the goal state in 51.1 of the 100 episodes on average.

As the result in the stochastic case is not satisfactory, sophisticated active inference is also used to calculate the policy. As sophisticated inference uses a time horizon and calculates the expected free energies and the approximate posterior of the actions $q(a_\tau|\vartheta)$ for every time step, the approximate posterior will represent the actions with different probabilities at different time steps. Because the approximate posterior is calculated in a recursive way and ends with calculating it at time step 0, the approximate posterior at this time step is also used when extracting the policy. The resulting policy can be seen in Table 7.

The only difference from the policy calculated with active inference is the first state in the upper left corner, but this makes the agent hit the goal in 73.0 of 100 episodes on average. This is not as good as the policy calculated with value iteration and Q-learning, but is a lot better than the policy calculated in Table 6.

Left	Up	Left	Up
Left	H	Left	H
Up	Down	Left	H
H	Right	Down	G

Table 7: Policy of the Frozen Lake environment when it is slippery, calculated with sophisticated active inference.

When testing sophisticated active inference different time horizons are used and the performance as a function of how long the time horizon is can be seen in Figure 6

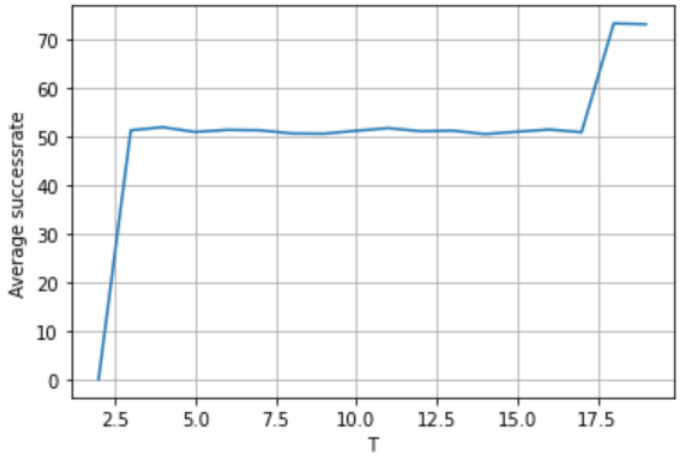


Figure 6: Performance of sophisticated active inference on the Frozen Lake as a function of time horizon.

Here it is seen that with a time horizon between 3 and 15 generates a policy that is equal to the one generated by active inference. When the time horizon gets longer than 16, the policy in the start state switches from "Down" to "Left" and the algorithm reaches the policy seen in Table 7.

5.2 Custom Frozen Lake

When applying value iteration on the custom Frozen Lake-environment with stochastic transitions we get the resulting policy seen in Table 8.

This policy makes the agent reach the goal state in 83.7 of the 100 episodes on average. The policy calculated by Q-learning is given in Table 9 and guides the agent to the goal in 83.01 of 100 episodes on average.

These policies are constructed in a way such that the agent avoids the holes, but the Frozen Lake environment is built such that the episode stops when the agent has performed 100 steps. If the agent was allowed to run forever on the environment, the agent would reach the goal state every time with both policies.

Down	Right	Right	Left	H	Right
Down	Right	Up	Right	Down	Down
Down	Left	H	Right	Down	Down
Up	Down	Down	Right	Down	Up
H	Right	Up	Right	Left	H
Down	Left	H	Right	Down	G

Table 8: Policy calculated with value iteration in custom a Frozen lake environment

Right	Right	Right	Left	H	Right
Down	Down	Up	Right	Down	Down
Down	Left	H	Right	Down	Down
Up	Down	Down	Right	Down	Up
H	Right	Up	Right	Left	H
Down	Left	H	Right	Down	G

Table 9: Policy calculated with Q-learning in custom a Frozen lake environment

Next, active inference is tested on the custom environment. To explore how the length of the policies used to calculate the expected free energy affects the policy of the environment, different lengths are tested. As the least amount of steps from the starting state to the goal state is 10, it is expected that the length of the policy has to be at least 10. First a length of 6 is used which leads to the policy seen in Table 10.

Down	Right	Up	Left	H	Right
Down	Down	Up	Down	Down	Down
Left	Left	H	Right	Down	Down
Up	Right	Down	Right	Left	Up
H	Right	Up	Right	Left	H
Down	Left	H	Right	Down	G

Table 10: Policy calculated by active inference in a custom Frozen lake environment using a policy length of 6.

We can see that this policy also makes sure that the agent cannot step in a hole. As the agent is stopped after performing 100 steps, the agent will not hit the goal state every time, but on average it hits the goal state in 75.6 of 100 episodes. This is lower than the policy of the reinforcement learning algorithms managed, which means that that the policy found by value iteration and Q-learning is better at directing the agent towards the goal.

Next, active inference is then tested with policies of length 10. The resulting policy is seen in Table 11. The resulting policy is almost the same as the one that is calculated with policy length 6. Also this policy ensures that it is impossible for the

Down	Right	Up	Left	H	Right
Down	Up	Up	Down	Down	Down
Left	Left	H	Right	Down	Down
Up	Right	Down	Down	Left	Up
H	Right	Up	Right	Left	H
Down	Left	H	Right	Down	G

Table 11: Policy calculated with active inference in a custom Frozen lake environment using a policy length of 10.

agent to end in a hole. However, the policy makes the agent reach the goal state in 71.3 of 100 episodes on average. This is a worse performance than for the policy calculated with policy lengths of 6. As this yielded a worse response, it could be interesting to find the resulting policies when other policy lengths are used. Because of this, the active inference model is also tested with policy lengths of 4 and 8 on the custom environment.

With a policy length of 4, the resulting policy makes the agent prone to fall into the upper hole and the hole to the left. This policy makes the agent reach the goal state in 64.1 of the 100 episodes on average. With a policy length of 8, the active inference algorithm yields the same results as when it use a policy length of 10. As it takes very long time to find the policy with the use of a policy length of 10, it would take very long time to use longer policies. The number of policies considered when using a policy length of 10 is 1048576 and if we increase the length of the policy by 1, the algorithm has to consider 4194304 policies. Thus, it is decided to not to test this implementation of active inference with policy lengths of larger than 10 to calculate the expected free energy.

Then, sophisticated active inference is tested on the custom environment. The policy is seen in Table 12. This policy cause the agent to hit the goal in 81.2 of 100 episodes on average. This is better than the policy calculated by the active inference in Table 10 and a bit lower than the performance of the policies calculated with value iteration and Q-learning seen in Table 8 and Table 9, respectively.

Down	Right	Right	Left	H	Right
Down	Down	Up	Right	Down	Down
Down	Left	H	Right	Down	Down
Up	Down	Down	Down	Down	Up
H	Right	Up	Right	Left	H
Down	Left	H	Right	Down	G

Table 12: Policy calculated with sophisticated active inference in a custom Frozen lake environment.

Here, also the performance of sophisticated active inference is tested with different

time horizons. The results are presented in Figure 7.

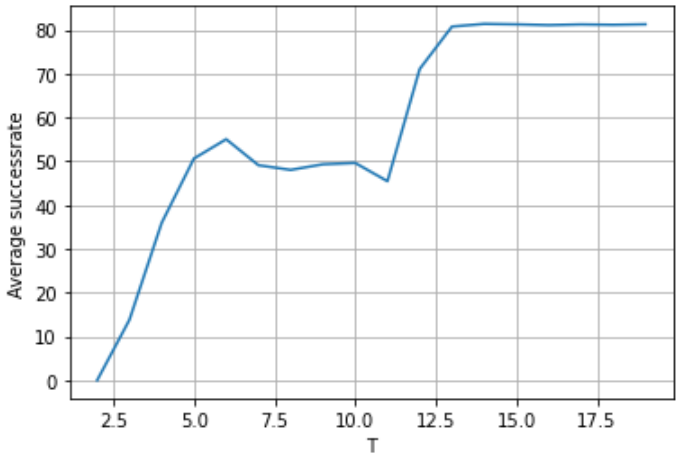


Figure 7: Performance of sophisticated active inference on the custom Frozen Lake as a function of time horizon

Here the sophisticated active inference agent reaches its best performance for a time horizon of 13 or longer. For time horizons smaller than this the performance increases until the time horizon is of length 6, where it decreases. Then, the performance increases again when the length of the time horizon is larger than 11.

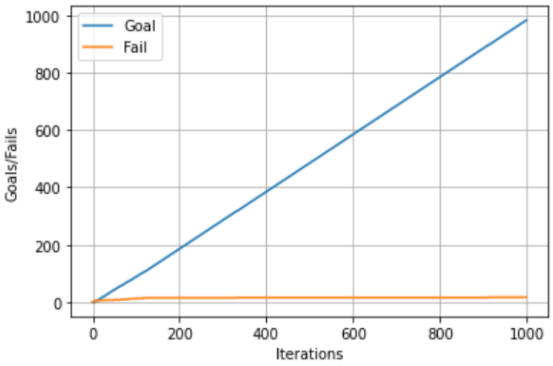
5.3 Simplified docking problem

5.3.1 Deep Q-Network

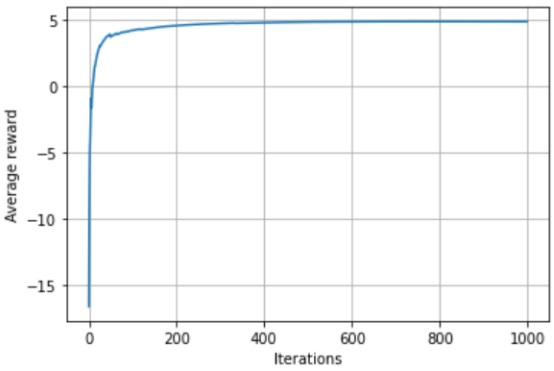
The first scenario of moving to the port entrance is solved easily by the agent. The algorithm is run over 1000 episodes, but almost immediately the agent finds the goal and keeps hitting it with consistency as seen in Figure 8a. The reason for not hitting the goal every time after finding it is due to exploration.

The average reward curve is seen in Figure 8b. In the beginning the negative reward is large because the agent is exploring the environment and does not exactly where the goal is. Then it finds the goal and the reward increases until it stabilizes when it consistently hits the goal with close to optimal amount of steps.

After training the network it was tested to get the agent from the start state (0,0) to the temporary goal state (20,15). An example of the resulting actions taken can be seen in Table 13. Here, the actions taken by the agent when it is in particular time steps and states are seen. By looking at the table we see that the Deep Q-Network manages to guide the agent between the two states. In time step 20 when the agent is in state (19,14) it take the action "Down-Right" which means that it ends in state (20,15). As the agent uses 20 steps which is the least amount of steps it can take,



(a) Results from scenario 1. Counts how many times the agent hits the goal and fails by hitting obstacles.



(b) Results from scenario 1. The average reward the agent receives.

Figure 8

it moves to the goal optimally.

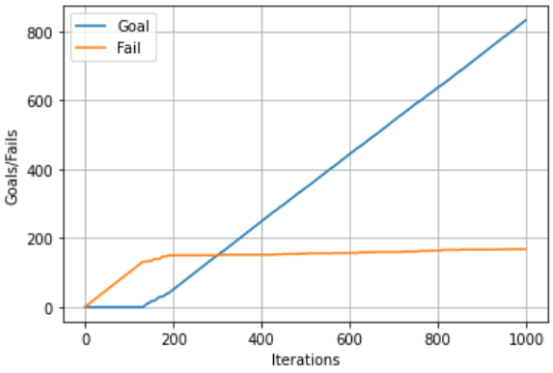
Time step, state	Action	Time step, state	Action
1, (0,0)	Down-Right	11, (10,10)	Down-Right
2, (1,1)	Down-Right	12, (11,11)	Down-Right
3, (2,2)	Down-Right	13, (12,12)	Down-Right
4, (3,3)	Down-Right	14, (13,13)	Down-Right
5, (4,4)	Down-Right	15, (14,14)	Down-Right
6, (5,5)	Down-Right	16, (15,15)	Down-Right
7, (6,6)	Down-Right	17, (16,16)	Right
8, (7,7)	Down-Right	18, (17,16)	Up-Right
9, (8,8)	Down-Right	19, (18,15)	Up-Right
10, (9,9)	Down-Right	20, (19,14)	Down-Right

Table 13: Policy of how the agent moves from the start state to the goal state in scenario 1.

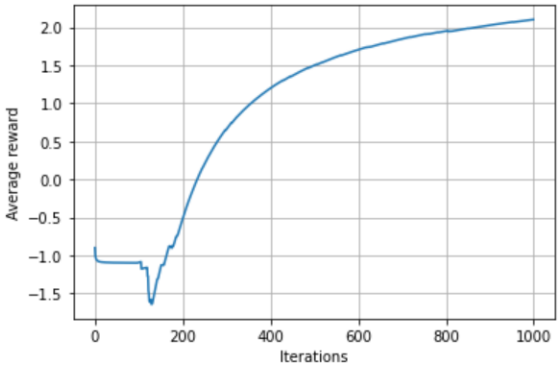
For the second scenario, the agent has some trouble finding the goal in the beginning even though it started in the first area. From Figure 9a it is seen that it struggles finding the goal for about the first 150 episodes. Then it finds the goal and starts hitting it consistently. After hitting the goal 30 times, the start position of the agent is moved to the second area. We see that the fail count increases again until around 210 episodes where the agent manages to get into the first area again. As it knows what to do here it starts hitting the goal consistently again.

The average reward of the agent can be seen in Figure 9b. This figure shows that the agent in the beginning accumulate negative rewards until it finds the goal and the average rewards increases. Here the agent learns what to do to improve the accumulated reward. Then the average reward drops again because the agent is

moved to the second area and starts accumulate negative rewards as it does not know where the goal is anymore. However, the drop in average reward does not last long as the agent learns quite fast how to reach the goal from this area as well. From here on out, the average reward continues to increase.



(a) Results from scenario 2. Counts how many times the agent hits the goal and fails by hitting obstacles.



(b) Results from scenario 2. The average reward the agent receives through training.

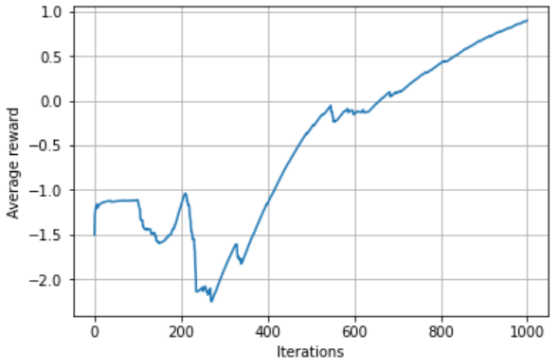
Figure 9

How the agent moves within the docking area is seen in Table 14. Here it is seen that the agent uses 12 steps by starting where scenario 1 stopped in (20,15) and move to the goal in (22,27). As this is the minimal amount of steps the agent has to take between these two states, it moves optimally.

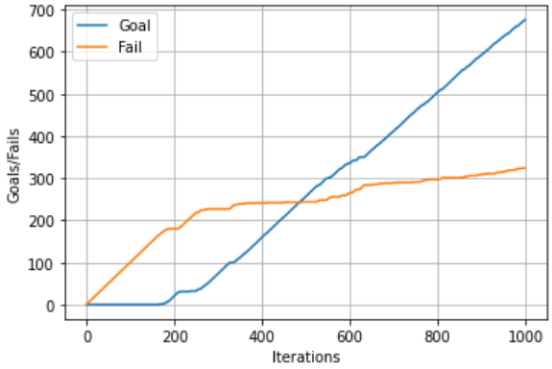
Time step, state	**Action**	**Time step, state**	**Action**
1, (20,15)	Down-Right	7, (24,21)	Down-Right
2, (21,16)	Down-Right	8, (25,22)	Down-Left
3, (22,17)	Down-Left	9, (24,23)	Down-Left
4, (21,18)	Down-Right	10, (23,24)	Down
5, (22,19)	Down-Right	11, (23,25)	Down-Left
6, (23,20)	Down-Right	12, (22,26)	Down

Table 14: Policy of how the agent moves from the start state to the goal state in scenario 2.

It is also possible to put the two scenarios together and train the Deep Q-Network on the whole environment. This gives the results seen in Figure 10a and Figure 10b. It is seen that the agent in the first part of scenario 2 has problems of hitting the goal in the first 180 iterations. It is seen that the average reward is quite stable until around 100 iterations where it starts to decrease. The reason for the decrease is that the agent tries to move around more to find the goal. When the agent has hit the target 30 times, at around 200 iterations, the start position is moved out the second area, but now it finds the goal again quite quickly. After hitting the goal



(a) Results from the complete environment. Counts how many times the agent hits the goal and fails by hitting obstacles.



(b) Results from the complete environment. The average reward the agent receives through training.

Figure 10

100 times, the start position is moved to the upper left corner of the environment to train it on the complete environment. Here, the agent almost immediately finds the goal again and the average reward increases after a small dip after a bit over 300 episodes. It is also seen that after 600 episodes the fail count increases and the average reward is flattened. This is probably due to exploration and that the agent has entered a new part of the environment.

To do a comparison with sophisticated active inference, the agent is set out in every open state and it is seen if it can move to the goal state. The agent reaches the goal from 200 of the 650 open states. From Figure 11 it is seen that it uses 32 steps from the starting state in the upper left corner, which is the least amount of steps it can take from that position. That means that it finds a optimal path from the starting state to the goal state which it is trained for. This can also be seen by combining the steps made by the agent in the two training scenarios, seen in Table 13 and Table 14.

5.3.2 Sophisticated active inference

The sophisticated active inference agent has no issues solving the task. Unlike the Deep Q-Network it did not need to split the environment into several parts to get to the goal. After calculating the expected free energies and the approximate posterior of the actions given the state $q(a_\tau|\vartheta)$, it is first tested if the active inference agent is able to find the goal from a start position in the upper left corner, in state $(0, 0)$. The calculations are done with a time horizon of 45, and it makes the agent hit the goal by using 32 steps which is the least amount of steps it can take from that position. The steps and actions taken by the agent are seen in Table 15. Here, the goal state is in column 22 and row 27, represented as (22,27), and this shows that the agent has taken an optimal route to the goal.

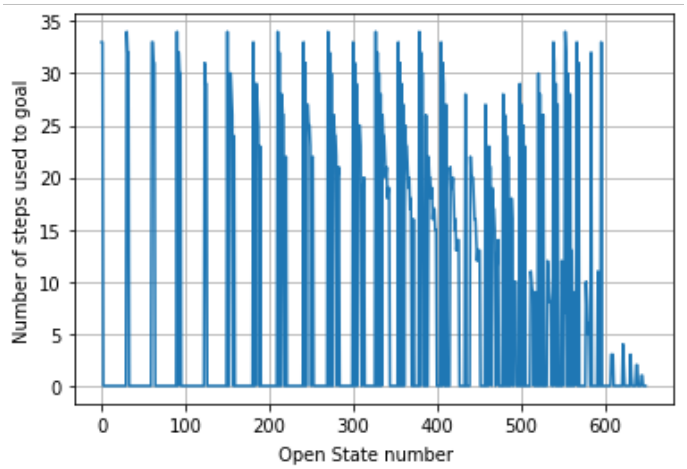


Figure 11: The number of steps the agent uses from a given state to the goal when using a Deep Q-Network.

After seeing that the active inference agent hit the goal successfully from the upper left corner, different start positions are used to see from where the agent is able to hit the goal. Every open position is used as a start position. With a time horizon of 45, the agent manages to reach the goal 650 times from the 650 different start positions. The reason for using a time horizon of 45 is that it is the smallest time horizon that gave the best results.

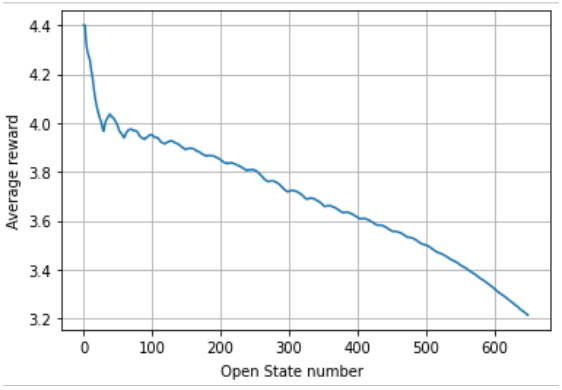
The average reward the agent accumulates by starting from the different start positions is shown in Figure 12a. The rewards are highest in the beginning because the agent is first tested from positions from the first row, then the second and so on. Therefore, when the agent gets closer and closer to the goal it is not able to accumulate as many rewards, as there are fewer steps to the goal than in the beginning. We also see that the average reward fluctuates. The reason for this is that the agent will use different amount of steps from different starting positions in the same row. As the goal state is located towards the right of the environment, a starting position far to the left will require more steps to the goal than starting to the right in the environment.

The number of steps the agent used from the different states is seen in Figure 12b. We see that the agent never uses more than 36 steps, which is the minimum amount of steps an agent need to take from the starting position furthest away from the goal. The number of steps fluctuates as the state numbers of a 30x30 grid is represented by a 900x1 vector. The different start positions are first taken from the left to the right of a row before start positions of the next row are tested. We see that the amplitude of the fluctuations get larger and larger as the state number gets closer to the goal state. Towards the end the fluctuations get larger. As there are more obstacles when the starting point gets closer to the goal, the agent in some cases needs to take many steps to reach the goal. For example by starting in the left corner it needs to go around the land area and into the port entrance. This takes

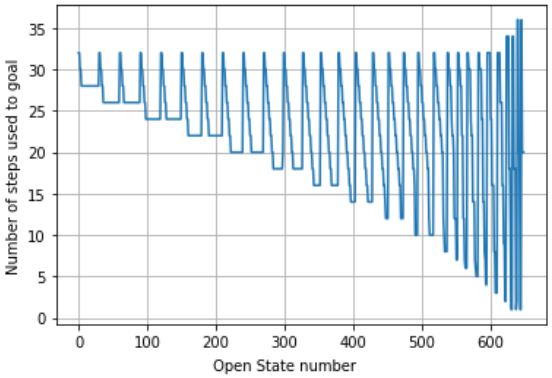
Time step, state	Action	Time step, state	Action
1, (0,0)	Down-Right	17, (16,12)	Down-Right
2, (1,1)	Down-Right	18, (17,13)	Right
3, (2,2)	Down-Right	19, (18,13)	Down-Right
4, (3,3)	Down-Right	20, (19,14)	Down-Right
5, (4,4)	Right	21, (20,15)	Down-Right
6, (5,4)	Right	22, (21,16)	Down
7, (6,4)	Right	23, (21,17)	Down
8, (7,4)	Right	24, (21,18)	Down
9, (8,4)	Down-Right	25, (21,19)	Down-Right
10, (9,5)	Down-Right	26, (22,20)	Down-Right
11, (10,6)	Down-Right	27, (23,21)	Down-Right
12, (11,7)	Down-Right	28, (24,22)	Down
13, (12,8)	Down-Right	29, (24,23)	Down
14, (13,9)	Down-Right	30, (24,24)	Down-Left
15, (14,10)	Down-Right	31, (23,25)	Down-Left
16, (15,11)	Down-Right	32, (22,26)	Down

Table 15: Policy of how the agent moves from the start state to the goal state with sophisticated active inference.

many more steps than for when the agent starts inside of the docking area. As some of these starting points are located in the same row, there are big fluctuations. However, these fluctuations in such a pattern tells that the agent uses the minimal amount of steps from the start positions, which show optimal performance.



(a) The average reward the agent receives when moving from a given open state to the goal.



(b) The number of steps the agent used from a given open state to the goal.

Figure 12

6 Discussion

In this chapter the results obtained from the different tests are discussed. Active inference and the reinforcement learning algorithms used on the various environments are compared, where advantages and disadvantages with the methods are discussed. Also, further work is proposed.

6.1 Discussion and comparison of results and algorithms

From the results we see that both the active inference agent and sophisticated active inference agent manage to perform as good as the reinforcement learning algorithms tested in a deterministic environment. The results in the deterministic Frozen Lake environment show that an active inference agent manages to do action planning in an optimal manner in a small grid with obstacles. The algorithms are optimal in the sense that they use the least amount of steps possible from every position to the goal state. As it is evident from these results that both active inference algorithms perform optimally on a deterministic space, it was decided that there is no need to explore the deterministic case in the custom Frozen Lake environment.

The simplified docking problem is also a implemented as a deterministic problem. Here, both the Deep Q-Network and sophisticated active inference manage to guide the agent from the starting state to the goal state optimally. In Table 13, Table 14 and Table 15 it is seen how the agent moves from the start state to the goal states. As the task is divided in two scenarios for the Deep Q-Network, Table 13 and Table 14 have to be combined to see how the agent moves trough the whole environment. However, it is seen that they combined use 32 steps which is the same amount for the sophisticated active inference agent. Even though the Deep Q-Network agent and sophisticated active inference agent use the same amount of steps, they do not take the same path. This shows that there are several optimal paths. Also, it is interesting to see that sophisticated active inference takes care of setting up the agent to move around the obstacles and into the port entrance quite early by taking the action "Right" from time step 5 to 8. The Deep Q-Network agent does this much later and starts moving around the obstacles and into the port entrance at time step 17 when it is in the same row as the goal state for scenario 1.

Actually, sophisticated active inference manage to guide the agent from every possible open state to the goal and it does this in an optimal manner. This shows that sophisticated active inference works well in action planning in an deterministic system and manages to guide an agent around whatever obstacle in the environment. By comparing Figure 12b and Figure 11 we see that the Deep Q-network agent is not able to do the same. The reason for this is that while an agent explores the environment under training of Deep Q-network, it does not hit every state enough times or might not hit every state at all. Sophisticated active inference on the other

hand iterates through every possible state and action combination. Hence, we know that the algorithm has calculated what the agent should do no matter the state which gives it an advantage if the agent was to start from a different position every time. On the other hand it makes the training slower than Deep Q-network. Where Deep Q-network uses around 10 minutes when training on the whole environment all together, sophisticated active inference uses around an hour. So for a docking problem where the agent starts from the same position every time, sophisticated active inference uses a lot of time and resources on calculating the expected free energies for states that does not get visited.

Another point to note is that Deep Q-network has to divide the environment in several parts to get an efficient training and to be able to hit the goal consistently without taking too much time. Here sophisticated active inference has an advantage as it can find the approximate posteriors of the actions of the whole environment as one. It is not affected by the complex docking area, which makes the training of the agent easier for the user. Also, by computing the approximate posteriors of the action for every state, the algorithm guarantees that it is has calculated what it needs for every state before the agent is applied on the environment.

While both active inference and sophisticated active inference perform as well as or better than the reinforcement learning algorithms in an environment with deterministic transitions, that is not the case for stochastic transitions. It is clear from the results that active inference does not perform as well as value iteration and Q-learning. Where the performance of active inference is significantly worse than the reinforcement learning algorithms, sophisticated active inference calculates a policy with better performance. The only difference between the policy of active inference and sophisticated active inference in the Frozen Lake environment is the policy of the starting state. Sophisticated active inference makes the agent choose the action "Left" and active inference agent takes the action "Down". As the transitions are stochastic the agent risks of moving to the left or the right with probability of 0.33 in both directions in addition to the intended direction. By taking the action "Left" instead of "Down" we make sure that the agent does not risk moving to the right towards the third state in the second row where it runs a risks of moving into the hole. The agent can still find a path to end up in that hole, but then it must take the path below the hole and up to the third state in row two. As the agent has two different routes to the goal with the policy calculated with the active inference algorithm, it performs worse than the policy calculated with sophisticated active inference.

It can also be seen why the two ways of calculating active inference choose a different action in the first state by looking at the approximate posterior of the actions given the state $q(a|\vartheta = 0)$, where 0 is the first state.

Active inference:

$$q(a|\vartheta = 0) = [0.14\ 0.36\ 0.36\ 0.14] \tag{35}$$

Sophisticated active inference:

$$q(a|\vartheta = 0) = [0.869\ 0.066\ 0.066\ 0.000] \tag{36}$$

Here "Left" corresponds to the first probability, "Down" corresponds to the second probability, "Right" corresponds to the third and "Up" corresponds to the third. It is seen that active inference gives a probability of 0.36 to the actions "Down" and "Right" and only 0.14 to "Left" which is the best action to take in this state. Sophisticated active inference assigns the highest probability to the "Left" action. Why active inference does not calculate the same policy in the start state as sophisticated active inference is hard to say. There is always the possibility of bugs in the code or some minor errors in the calculations of the expected free energies or approximate posterior, but there was not found any. On the other hand, the two ways of calculating active inference outputs the same policy for all the other states and the active inference implementation gives an optimal policy in the deterministic case. Even though active inference has the advantage of seeing why a particular action is chosen by being able to see the approximate posterior of an action and how it is calculated with the expected free energies, it is not necessarily easy. Because the number of policies used for calculation of the free energies is large, it is hard to see how the expected free energy of a specific policy contributes to the approximate posterior of the actions.

As we have seen, sophisticated active inference produces a policy that gives a bit worse performance than the reinforcement learning algorithms. The only difference in policy is the third state in the first row. Here instead of keeping the agent away from the hole to the left in the second row, with the "Up" action, sophisticated active inference instead assigns the action "Left" with the highest probability. This gives the agent a possibility of moving down again to the third state in the second row and risks moving into the hole. Thus, the sophisticated active inference agent does not reach quite the same performance as the reinforcement learning agents. We look at the approximate posterior of the actions in the state, which is called state 2:

$$q(a|\vartheta = 2) = [0.907\ 0.001\ 0.092\ 0.000] \tag{37}$$

Here it is seen that a high probability is assigned to "Left" and a probability of around zero is assigned to the action "Up". What is common for the three actions assigned a probability of around zero is that they all have a probability of moving right of 0.33. By taking the "Left" action the agent does not risk moving to the right. A possible explanation is that the expected free energy is high if the agent moves to the right as it then has two walls and a hole around itself. Also, the agent has to move around the hole to move to the goal. However, taking the "Left" action does not affect the performance very much and the sophisticated active inference agent still performs well.

With the custom Frozen Lake environment it is also seen that the reinforcement learning algorithms value iteration and Q-learning perform better than active infer-

ence and sophisticated active inference. Again, it is seen that sophisticated active inference performs better than active inference, but the difference is not that big now. In this case all algorithms calculate policies that keeps the agents away from the holes, but there is a difference in how efficient they are at guiding the agent towards the goal. By comparing the policy calculated by value iteration and Q-learning and sophisticated active inference, it is seen that the only differences are the second state in the second row and the fourth state in the fourth row. Even though there are some differences in the results of the sophisticated active inference and the reinforcement learning algorithms, the resulting policy still performs quite well and is suitable for action planning in an environment where the transitions are stochastic.

The active inference performs better in the custom Frozen Lake environment, but again not as good as the other algorithms. Now, also different lengths of policies are tested. As an optimal path from the start state to the goal state requires 10 steps it was expected that policy lengths of 10 would give the best result. However, from the results it is seen that a policy length of 6 gives the best results. In this environment after an action is taken the agent can be located three states. When the expected free energy of the next action in the policy is to be evaluated we can look at it as the action is taken from three different states where the probability of being in that state is 0.33. Hence, the probabilities of which state the agent is located in propagates out in the environment as more actions are taken and the probabilities is dependent on the actions taken which gives a different KL-divergence each time. This explains why a policy length of 6 can perform good results where the agent in reality needs more than 6 steps to reach the goal state. In a deterministic case that would not be the case for the states that needs more than 6 steps to reach the goal state as the KL-divergence would be 0 for every policy. However, it does not explain why active inference with the use policy lengths of 6 outperforms policy lengths of 10 in a stochastic environment. It is hard to find out why this is the case and an explanation was not found. But, it is clear that the assumption that a policy length of 10 is needed does not apply in the stochastic environment. Overall, from the results in this thesis it seems that the way active inference has been implemented in this thesis with predefined policies is not suited for action planning in an environment with stochastic transitions.

By comparing the run times of the algorithms in the custom Frozen Lake environment, it is seen that value iteration and sophisticated active inference takes the least amount of time. They only use a couple of seconds before the policy is calculated. The slowest one is the active inference. That is because it calculates the expected free energies of every possible policy of a given length. The run time is dependent on how long the policies used for calculation of policies are. If the policy length is of 6 the computation of the policy takes around one minute, but if the policy length is of 10 the computation takes about 6 hours. As the run time of sophisticated active inference is around an hour for the simplified docking problem, it is seen that the run time of the sophisticated active inference algorithm also is dependent on the

size of the state space and action space.

The sophisticated active inference algorithm is also dependent on the length of the time horizon which is seen in Figure 6 and Figure 7. From these figures it is seen that it is important to use a time horizon that is long enough to reach the best possible performance. It is also seen that even though the Frozen Lake environment is smaller than the custom one, the time horizon needs to be longer to reach best policy. A possible explanation is because the Frozen Lake environment is harder to navigate in as the agent cannot avoid the holes completely, it needs a longer time horizon to find the correct policy. It is also interesting to see in Figure 7 that the policy is worse when using a time horizon of 11 than a time horizon of 6. The reason is that in the beginning of the time horizon the probabilities in the approximate posterior are often quite close to each other. Thus, a small change in the expected free energy might change which action is chosen in the policy. This can give fluctuations in the policy. However, when the time horizon is longer, the calculated expected free energy has contributions from expected energies from more time steps and the algorithm can be more certain about the approximate posteriors of the actions. Therefore, the algorithm gives a better result for time horizons larger than 12.

Instead of using rewards as in reinforcement learning active inference and sophisticated active inference algorithms use prior beliefs about the states the agent wants to be in. So, instead of assigning values to favourable states, probabilities are instead assigned to these state. The prior beliefs are in essence rewards modelled as probabilities. But, using probabilities instead of a reward function has an advantage. To make a reward function that encourages the agent to find an optimal policy is hard, and it took a long time before the DQN gave satisfactory performance as it took time to find a good reward function. By using prior beliefs about the goal state, the problem of finding the reward function is avoided which can save a lot of time and testing.

Where Q-learning and Deep Q-learning does not need to know the transition probabilities between the states and the reward function to learn the value of an action, active inference and sophisticated active inference need to know the likelihood $p(s_\tau|\vartheta_\tau)$ and prior $p(\vartheta_\tau|\vartheta_{\tau-1}, a_{\tau-1})$ probabilities. These are given as the matrices A and B in this thesis. The matrix B contains the transition probabilities, which is comparable to the transition probabilities of reinforcement learning. The matrices have to be known before performing active inference either by being defined or by being learned before training as in [4]. By learning the likelihood and the prior makes the algorithms more flexible as the user of the algorithms does not need to know the environment. If the environment is quite complex and hard to define it might be better to just make the agent explore the environment and estimate those matrices. On the other hand, this takes a lot of time. By looking at the docking problem, it takes a lot of time to learn the matrices as the agent has to visit every state several times. In an environment like this, it is hard to make sure that the

agent manages to do so, just as seen in the Deep Q-Network case. However, for a deterministic problem where the agent is going to move from the same spot to the same docking position every time, it might be feasible to only learn the matrices for the states around the optimal path. In a stochastic problem this would probably not be a good solution.

6.2 Future work

We have seen from the results and the discussion that active inference and sophisticated active inference perform good for action planning in a deterministic environment. However, the environments these algorithms are tested on are quite simple. There are only a few possible actions and the environments have been directly observable, i.e. the agent knows exactly which state it is located in. For further work, the action space should be expanded to be continuous and the environment should be partially observable such that the agent has some uncertainties about which state it is in. To make the action space continuous, active inference must be calculated in a different way than it has been done in this thesis. When the action spaces gets large the calculations of the algorithms presented here will take too much time. Thus, further work may be to implement the active inference agent in combination with deep neural networks as seen in papers like [15] or [38] to give some examples. Also, as the two different implementations of active inference in this thesis use a lot of time to calculate the approximate posteriors for large environments, the use of neural networks can reduce the computation time.

In addition, the docking problem is extremely simplified. The area should be expanded and divided into a finer grid and the action space should be made more realistic of a marine vessel. Also, in reality the boat has to enter the docking position with the correct pose. Because of this, the agent should keep track of the pose and make sure that it enters the docking space in the correct manner. It was tried to do this in this task, but the efforts were unsuccessful. Furthermore, vessel dynamics and disturbances should also be added to make the simulations more realistic. In overall, the model of the marine vessel and its environment should be improved and made more realistic. It is seen in this thesis that sophisticated active inference performs well on a stochastic environment and is able to handle uncertainties. Thus, it would be interesting to see how it handles a more complex environment.

7 Conclusion

In this thesis we have explored how two different implementations of active inference, here called active inference and sophisticated active inference, performs as action planners in different discrete environments. In addition, characteristics of active inference, how the performance of active inference changes with policy length and the performance of sophisticated active inference as a function of the time horizon were investigated.

Active inference and sophisticated inference were tested on two different Frozen Lake environments with both deterministic and stochastic transitions. To get a reference point of how well the algorithms performed in this environment the reinforcement learning algorithms value iteration and Q-learning were also implemented. First, the algorithms were tested on the environments with deterministic transitions. Here, the active inference algorithms achieved as good performance as the reinforcement learning algorithms. However, when tested with stochastic transitions the policies calculated with the active inference algorithms performed worse. Where sophisticated active inference performed almost as good as the reinforcement learning algorithms, active inference performed significantly worse.

In addition, sophisticated active inference was also implemented on a simplified docking problem where an environment imitating a docking problem was constructed. For comparison, a Deep Q-Network was also implemented on this environment. Sophisticated active inference performed good on this environment and achieved optimal action planning such that an agent could move from every open state to the goal state in an optimal manner. As sophisticated active inference achieved optimal action planning from every state, it performed better than the Deep Q-Network which achieved optimal action planning from only a subset of the open states.

Overall, both active inference and sophisticated works well for action planning on small deterministic environments. As active inference use predefined policies for calculation of expected free energies, it will quickly use a lot of computational time when the state space or action space gets large. Thus, it is restricted to the small environments. Sophisticated active inference on the other hand calculates the expected free energies in a recursive manner and can be used on larger spaces as the simplified docking environment. However, this still takes a lot of time and other ways to implement active inference should be explored. In addition, sophisticated active inference works well for action planning on stochastic environments which shows that it may have the potential of guiding a marine vessel in a docking area.

Considering the way active inference is implemented compared to reinforcement learning it has both advantages and disadvantages. The way active inference uses prior probabilities instead of a reward function can be useful as it is hard to find good reward functions. However, the calculation of active inference and sophisticated inference in general takes longer time than the reinforcement learning algorithms.

The algorithms also need to know the likelihood and prior matrices, in contrast to Q-learning, which makes the calculations use more time and can make the calculations somewhat inaccurate.

References

[1] C. C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] R. Bogacz. "A tutorial on the free-energy framework for modelling perception and learning". In: *Journal of mathematical psychology* 76 (2017), pp. 198–211.

[3] O. Çatal, S. Wauthier, T. Verbelen, C. De Boom, and B. Dhoedt. "Deep active inference for autonomous robot navigation". In: *arXiv preprint arXiv:2003.03220* (2020).

[4] L. Da Costa, T. Parr, N. Sajid, S. Veselic, V. Neacsu, and K. Friston. "Active inference on discrete state-spaces: a synthesis". In: *Journal of Mathematical Psychology* 99 (2020), p. 102447.

[5] E. Eilertsen. "High-level Action Planning for Marine Vessels Using Reinforcement Learning". MA thesis. NTNU, 2019.

[6] I. El Naqa and M. J. Murphy. "What is machine learning?" In: *machine learning in radiation oncology*. Springer, 2015, pp. 3–11.

[7] Z. Fountas, N. Sajid, P. A. Mediano, and K. Friston. "Deep active inference agents using Monte-Carlo methods". In: *arXiv preprint arXiv:2006.04176* (2020).

[8] K. Friston. "The free-energy principle: a rough guide to the brain?" In: *Trends in cognitive sciences* 13.7 (2009), pp. 293–301.

[9] K. Friston. "The free-energy principle: a unified brain theory?" In: *Nature reviews neuroscience* 11.2 (2010), pp. 127–138.

[10] K. Friston, L. Da Costa, D. Hafner, C. Hesp, and T. Parr. "Sophisticated inference". In: *Neural Computation* 33.3 (2021), pp. 713–763.

[11] K. Friston, T. FitzGerald, F. Rigoli, P. Schwartenbeck, G. Pezzulo, et al. "Active inference and learning". In: *Neuroscience & Biobehavioral Reviews* 68 (2016), pp. 862–879.

[12] K. Friston, J. Kilner, and L. Harrison. "A free energy principle for the brain". In: *Journal of physiology-Paris* 100.1-3 (2006), pp. 70–87.

[13] K. J. Friston, J. Daunizeau, and S. J. Kiebel. "Reinforcement learning or active inference?" In: *PloS one* 4.7 (2009), e6421.

[14] C. Heins, A. Tschantz, B. Millidge, B. Klein, A. Niranjan, and D. Demekas. *infer-actively/pymdp*. 2021. URL: `https://github.com/infer-actively/pymdp`.

[15] O. van der Himst and P. Lanillos. "Deep active inference for partially observable MDPs". In: *International Workshop on Active Inference*. Springer. 2020, pp. 61–71.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[17] L. N. Kanal. "Perceptron". In: *Encyclopedia of Computer Science*. 2003, pp. 1383–1385.

[18] A. King. *Create custom gym environments from scratch - A stock market example*. Apr. 2019. URL: https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e.

[19] J. Kober, J. A. Bagnell, and J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[20] P. Kormushev, S. Calinon, and D. G. Caldwell. "Reinforcement learning in robotics: Applications and real-world challenges". In: *Robotics* 2.3 (2013), pp. 122–148.

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

[22] A. J. Maren. "Derivation of the variational Bayes equations". In: *arXiv preprint arXiv:1906.08804* (2019).

[23] B. Millidge. "Deep active inference as variational policy gradients". In: *Journal of Mathematical Psychology* 96 (2020), p. 102348.

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

[25] A. Mosavi and A. Varkonyi. "Learning in robotics". In: *International Journal of Computer Applications* 157.1 (2017), pp. 8–11.

[26] OpenAI. *FrozenLake-v0*. URL: https://gym.openai.com/envs/FrozenLake-v0/.

[27] T. Parr and K. J. Friston. "Generalised free energy and active inference". In: *Biological cybernetics* 113.5 (2019), pp. 495–513.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[29] A. Paul, N. Sajid, M. Gopalkrishnan, and A. Razi. "Active Inference for Stochastic Control". In: *arXiv preprint arXiv:2108.12245* (2021).

[30] H. A. Pierson and M. S. Gashler. "Deep learning in robotics: a review of recent research". In: *Advanced Robotics* 31.16 (2017), pp. 821–835.

[31] A. S. Polydoros and L. Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[32] J. Ruiz-del-Solar, P. Loncomilla, and N. Soto. "A survey on deep learning methods for robot vision". In: *arXiv preprint arXiv:1803.10862* (2018).

[33] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall, 2010.

[34] N. Sajid, P. J. Ball, T. Parr, and K. J. Friston. "Active inference: demystified and compared". In: *Neural Computation* 33.3 (2021), pp. 674–712.

[35] I. H. Sarker. "Machine learning: Algorithms, real-world applications and research directions". In: *SN Computer Science* 2.3 (2021), pp. 1–21.

[36] A. Shrestha and A. Mahmood. "Review of deep learning algorithms and architectures". In: *IEEE Access* 7 (2019), pp. 53040–53065.

[37] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. 2, in progress. The MIT Press, 2015. URL: `https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf`.

[38] K. Ueltzhöffer. "Deep active inference". In: *Biological cybernetics* 112.6 (2018), pp. 547–573.

[39] W. Wang and K. Siau. "Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda". In: *Journal of Database Management (JDM)* 30.1 (2019), pp. 61–79.

[40] R. Wason. "Deep learning: Evolution and expansion". In: *Cognitive Systems Research* 52 (2018), pp. 701–708.

[41] A. Wróblewska, T. Stanisławek, B. Prus-Zajączkowski, and Ł. Garncarek. "Robotic process automation of unstructured data with machine learning". In: *Annals of Computer Science and Information Systems* 16 (2018).

[42] X. Xiao, B. Liu, G. Warnell, and P. Stone. "Motion control for mobile robot navigation using machine learning: a survey". In: *arXiv preprint arXiv:2011.13112* (2020).

[43] X.-D. Zhang. "Machine learning". In: *A Matrix Algebra Approach to Artificial Intelligence*. Springer, 2020, pp. 223–440.