Know what it does not know:

Improving Offline Deep Reinforcement Learning with Uncertainty Estimation

Jordi Smit

TU Delft

Know what it does not know:

Improving Offline Deep Reinforcement Learning with Uncertainty Estimation

by

Jordi Smit

August 22, 2021

To obtain the degree of **Master of Science Computer Science** with a specialization in **Data Science and Technology** at the EEMCS faculty of Delft University of Technology, to be defended publicly on Tuesday August 31, 2021.

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

Student number: 4457714
Project duration: November 17, 2020 – August 31, 2021
Thesis Commitee: Dr. F. A. Oliehoek, TU Delft, supervisor
Dr J. C. van Gemert, TU Delft thesis committee member
C. T. Ponnambalam, TU Delft, daily supervisor

TU Delft

Preface

Last year, I happened to talk to Gosia Migut about my interest in reinforcement learning. Based on this conversation, she suggested I should do my master's thesis with Dr. Frans Oliehoek. Based on this suggestion, I contacted Dr. Frans Oliehoek with topic proposals for both a literature survey and a master's thesis. During this email exchange, he brought me in contact with Canmanie Ponnambalam, who would become my daily supervisor for my literature survey and master's thesis. Interestingly, the topic I originally proposed for my master's thesis was related to sample efficient reinforcement learning. However, during my literature survey, Canmanie Ponnambalam introduced me to her interesting research topic: offline reinforcement learning. This interesting information made me realize that offline reinforcement learning was a much better approach towards sample efficient reinforcement learning. This realization eventually made me decide to select offline deep reinforcement learning as my master's thesis topic. This choice eventually led to this thesis and the accompanying paper, which are both the final products of my graduation project as part of the Interactive Intelligence group at the Delft University of Technology.

This work would not have been possible without the advice and support of several people. Therefore, I'm thankful to Dr. Frans Oliehoek for being my supervising professor and for all the critical questions he asked during our meetings. I'm also thankful to Dr. Jan van Gemert for his willingness to be on my thesis committee. I'm also thankful for the critical feedback from Dr. Matthijs Spaan on the paper that came forth of this thesis. Finally, I would like to thank Canmanie Ponnambalam. Her honest feedback and her immense interest in offline reinforcement learning were the key ingredients that made this thesis possible. Also, I am very thankful for all her personal advice about working-at-home-related problems, being stuck on research problems, and her advice with respect to taking time off to deal with the death of my grandfather. I could not have asked for a better daily supervisor.

Jordi Smit
Katwijk aan zee, August 22, 2021

Abstract

Offline reinforcement learning, or learning from a fixed data set, is an attractive alternative to online reinforcement learning. Offline reinforcement learning promises to address the cost and safety implications of taking numerous random or bad actions online, which is a crucial aspect of traditional reinforcement learning that makes it difficult to apply in real-world problems. However, when offline reinforcement learning is naïvely applied to a fixed data set, the resulting policy may exhibit poor performance in the real environment. This happens due to over-estimations of the expected return for state-action pairs not sufficiently covered in the data set. Therefore, offline reinforcement learning agents *must know what they do not know*, allowing them to avoid these over-estimated state-action pairs and their potentially erroneous outcomes. A promising way to instill offline reinforcement learning agents with this ability is the pessimism principle, which states that agents should select actions that maximize an uncertainty-based lower bound of the expected return. This pessimism principle has drastically improved the performance of offline reinforcement learning methods in the tabular and linear function approximation domain. However, in deep reinforcement learning, uncertainty estimation is highly non-trivial, and the development of effective uncertainty-based pessimistic algorithms remains an open question. That is why in this thesis, we explore various existing deep learning-based uncertainty estimation techniques with the aim to combine them with existing deep reinforcement learning methods to create an uncertainty-aware offline deep reinforcement learning algorithm. This research has resulted in two novel offline deep reinforcement learning methods built on Double Deep Q-Learning and Soft Actor-Critic. We applied these methods to various benchmarks and experiments to demonstrate their interesting and unique properties. In some situations, they even beat the current state-of-the-art results of these benchmarks.

Contents

Abbreviations

BC behaviour cloning. 36, 37, 39

BCQ Batch-Constrained Q-learning. 30, 31, 36

BEAR Bootstrapping Error Accumulation Reduction. 31

CBCQ Continuous Batch-Constrained Q-learning. 37, 39

COMBO Conservative Offline Model-Based policy Optimization. 32, 39

CQL Conservative Q-Learning. 31, 32, 39

D4RL Datasets for Deep Data-Driven Reinforcement Learning. 34, 37, 38, 42

DDQN Double Deep Q-Network. 9, 10, 14, 26, 27, 35, 36, 39, 42, 50, 51

DQN Deep Q-Network. 9, 14, 27

i.i.d. independent and identically distributed. 2, 8

IOD in-of-distribution. 2, 3, 20, 30–32

KL Kullback–Leibler. 31

MC Monte Carlo. 21

MDP Markov decision process. 5–7, 10, 11, 14, 16, 26, 28, 29, 32, 33, 37, 41

MOPO Model-based Offline Policy Optimization. 32, 33

MSBE mean squared Bellman error. 6–10, 14, 32

OOD out-of-distribution. 2, 3, 19, 20, 30, 31

PEBL PEssimistic ensemBLe. 4, 26, 28, 34–37, 39, 42, 50, 51

RL reinforcement learning. 1–18, 22–26, 28, 30, 32–39, 41–43

SAC Soft Actor Critic. 10, 14, 26, 28, 29, 37–39, 42, 51

VAE variational autoencoder. 31

List of Symbols

Algorithms related symbols

α	Entropy trade-off parameter;
$\bar{\mu}_x$	The sample mean over x is defined as $\frac{1}{n}\sum_{i=1}^{N} x_i$.
β	Prior weight in the multi-headed bootstrap with priors method.
$\mathcal{L}$	A loss function;
∇_x	The gradient with respect to parameter x;
ϕ	Parameter of the policy;
σ_x	The sample standard deviation over x is defined as $\sqrt{\frac{1}{n-1}\sum_{i=1}^{N}(x_i - \bar{\mu}_x)^2}$.
θ	Parameters of the Q-function;
θ'	Parameters of the target Q-function;
C_π	Uncertainty trade-off parameter;
H	The number of heads in an ensemble.
m	A Boolean mask.

Markov Decision Process related symbols

$\mathcal{A}$	The action space;
γ	The discount factor of the MDP $\mathcal{M}$.
$\mathcal{M}$	A MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ containing state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$, transition function $\mathcal{T}$ and a discount factor γ.
$\mathcal{M}_\mathcal{D}$	The empirical MDP based on the finite data set $\mathcal{D}$.
π	A policy is a distribution over actions conditioned on the current state.
$\mathcal{R}$	Reward function where $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$.
$\mathcal{S}$	The state space;
$\mathcal{T}$	Transition that maps state-action pairs to a distribution over next states $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$.
$Q(s, a)$	The expected discounted return for state s and action a, also called a Q-value.

Transition related symbols

$\mathrm{a_t}$	The action the agent took at the t-th time step in state s_t.
$\mathcal{D}$	A data set which consist of transitions $(s_t, a_t, r_t, d_t, s_{t+1})$.
$\mathrm{d_t}$	A terminal flag that indicates whether the episode terminated or not after taking action a_t in state s_t at the t-th time step.
$\mathrm{r_t}$	The reward an agent receives for taking action a_t in state s_t at time step t.
$\mathrm{s_t}$	The state at time step t.
y_t	The TD-target at time step t.

List of Figures

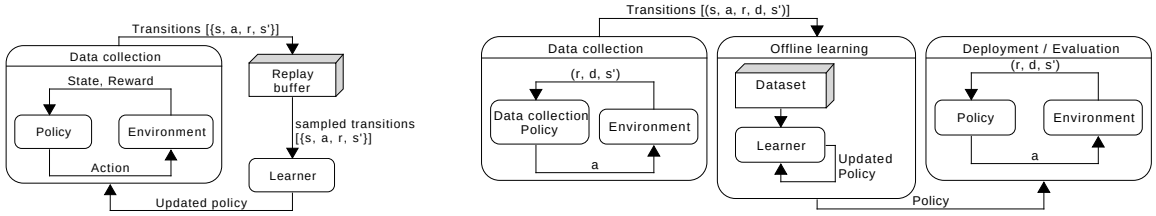
List of Tables

List of Algorithms

1

Introduction

Reinforcement learning (RL) is a machine learning paradigm for solving sequential-decision making problems. Using this framework, an agent can automatically learn near-optimal sequential-decision making policies that maximize a user-specified reward function. Typically, RL algorithms learn these policies using a trial and error-based approach. In this approach, the agent gathers more and more data in an environment while it keeps iteratively improving its policy until it has sufficient information to learn a near-optimal policy (Figure 1.1a). In recent years, this online learning-based approach combined with deep neural networks has allowed RL methods to attain excellent results in various domains such as robotics [4, 16] and games [33, 40, 49]. However, this *online* learning paradigm has proven to be one of the biggest obstacles to its widespread adoption outside simulated problems such as games and physics simulators [29]. Due to this *online* paradigm, the training of RL agents quickly becomes too costly e.g., robotics or dangerous e.g., autonomous driving or healthcare if the method cannot do the data collection part inside a simulator. In these situations, it would be preferred if RL agents could learn from large pre-collected data sets without the need for any online data collection. Even when online interactions are feasible, it could be preferable to utilize pre-collected data sets because neural networks tend to perform and generalize better on more complex tasks with larger and more diverse data sets [15, 29]. *Offline* RL promises to address these key issues of online RL. In this data-driven paradigm of RL, the agents are no longer allowed to perform online data gathering. Instead, a data set is available that contains transitions sampled from the environment that ideally provide useful information about its dynamics and the task we want to optimize. Using only this data set, the agent must learn a sequence of actions, called a policy, that maximizes the expected reward in the environment that generated the data set (Figure 1.1b).



(a) Online reinforcement learning

(b) Offline reinforcement learning

Figure 1.1: A visualization of the difference between the online and offline reinforcement learning process. In the offline process, the agent only interacts with the environment once it has been deployed. This is a significant difference from online reinforcement learning, where the agent interacts with the environment after every policy update.

In recent years, many offline deep RL algorithms have emerged [2, 11, 12, 19, 24, 25, 27, 50, 52, 53]. However, it remains challenging for offline deep RL methods to work with any type of data set, independent of their data collection strategy [10, 38]. However, recent theoretical results have shown that

agents with the ability to *know what they do not know* based on their uncertainty do not have this issue independent of the used data collection strategy [9, 38]. Using this ability, the agents can detect the limit of their knowledge of their environment, allowing them to avoid sequences of action with unforeseen and potentially erroneous outcomes. These theoretical results have resulted in algorithms with theoretical optimal regrets bounds in the tabular and linear function approximation setting [9, 23, 38]. However, at the point of writing, there exist no model-free uncertainty-aware offline deep RL algorithms.

This thesis explores how deep RL agents can be instilled with the ability to *know what they do not know*. This thesis will do this in three ways. Firstly, it explores the offline RL problem and why agents with the ability to *know what they do not know* based on their uncertainty have optimal regret bounds based on the existing offline RL literature. Secondly, it explains the difficulties of integrating the existing deep learning uncertainty estimation techniques with existing RL agents. Finally, it empirically compares how well the resulting uncertainty-based offline deep RL agents follow the theoretically predicted properties.

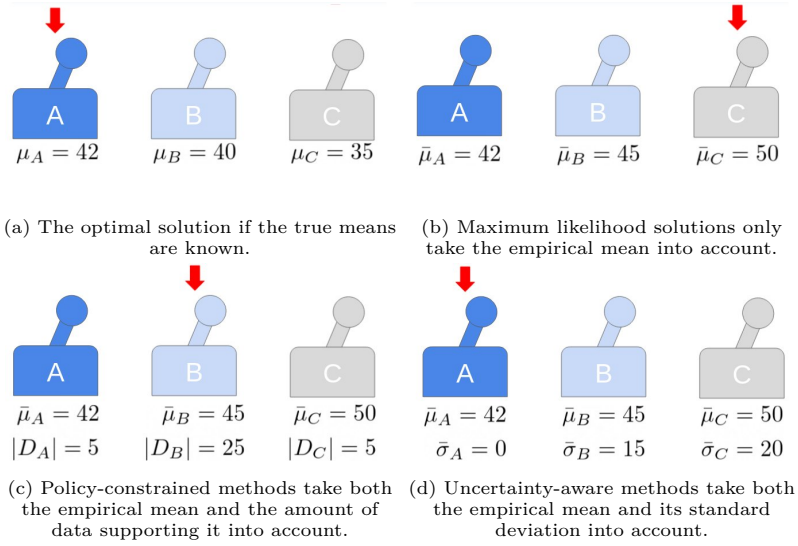
1.1. Knows what it does not know

When we speak about agents with the ability to *know what they do not know*, we mean the ability to recognize that you have insufficient information to make a specific prediction. This ability is crucial in offline RL, where agents are no longer able to gather more data. Because of this, offline RL agents must know the limits of their knowledge, or else they will learn overoptimistic and potential erroneous policies. To demonstrate why this ability is essential, let us consider the following questions.

Why should RL agents know what they do not know in the offline setting?
For an agent, the ability to *know what it does not know* is crucial if it wants to reason about counterfactual queries sometimes called “what-if" questions, which are a core part of offline RL. To answer these types of questions, an agent has to reason about what might have happened if it would have carried out a different action. This ability is necessary for offline RL since the goal is to find a better policy than the empirical policy observed in the data set.

In offline RL, agents can only learn to distinguish between rewarding and non-rewarding sequences of actions from the data available in the data set. However, by definition, if an agent wants to find a better policy than the empirical policy, the agent will have to evaluate sequences of actions that were not in the original data set, so-called out-of-distribution (OOD) actions. In this situation, the independent and identically distributed (i.i.d.) assumption of function approximators, such as neural networks, no longer holds, which means that these distinctions will most likely be uncertain and erroneous [15]. Therefore, it is essential that agents *know what they do not know*. With this ability, an agent can determine if it has sufficient information to estimate the expected reward for a specific sequence of actions. If an agent knows it has too little information for a specific part of the action sequence, it can avoid these OOD actions and their potentially erroneous expected reward evaluations. In this way, the agent can still improve upon the empirical policy by combining promising parts of trajectories that are in-of-distribution (IOD), while avoiding uncertain and potentially erroneous OOD actions.

Why must offline RL agents take the amount of information in the data set into account to know what they do not know?
Suppose we have the simple offline multi-armed bandit problem of Figure 1.2. In an offline multi-armed bandit problem, the agent aims to find the arm that maximizes the expected reward using only a fixed data set. The naive approach would be to use any existing online RL method and prevent the method from collecting any additional data. These online RL methods estimate the expected return per arm using a maximum likelihood-based approach such as the empirical mean per arm. These naive approaches do not measure if an agent has too little information to estimate the empirical mean and naively select the arm with the highest empirical mean. It does not matter for these methods if this empirical mean is highly uncertain or has little data supporting it. This property causes these naive approaches to learn erroneous policies if the data set contains sampling errors or too little information to estimate the mean correctly (Figure 1.2b).
In contrast, policy-constrained methods avoid the issues from the naive methods by taking the amount of data supporting the empirical mean into account. These methods measure the data concentrations



(a) The optimal solution if the true means are known.

(b) Maximum likelihood solutions only take the empirical mean into account.

(c) Policy-constrained methods take both the empirical mean and the amount of data supporting it into account.

(d) Uncertainty-aware methods take both the empirical mean and its standard deviation into account.

Figure 1.2: A visualization of the different approaches in offline reinforcement learning and the different information they use on a simple offline multi-armed bandit problem. In this problem, every arm has an unknown true mean μ_i, an empirical mean $\bar{\mu}_i$, a sample standard deviation $\bar{\sigma}_i$, and a sample size $|D_i|$. In offline reinforcement learning problems, such as this one, only the uncertainty-aware approach is safe against statistical issues such as sampling errors.

in the data set, allowing them to constrain the learned policy to areas with high data concentrations. This constraint makes it less likely that the method will learn an incorrect policy due to sampling issues. However, this constraint also has a major side effect: it makes the learned policy dependent on the quality of the data collection policy instead of the information in the data set. For example, the policy-constrained method will pick arm-B in Figure 1.2c because it has more data even though the data for arm-A is far more informative. This property is non-ideal for offline RL because the goal of offline RL is to find the best possible policy given the data set, which is impossible for policy-constrained methods if the data collection policy is sub-optimal [9].
Finally, there are the *uncertainty-aware methods* (Figure 1.2d). These methods take the uncertainty in the empirical mean into account by picking the action with the highest lower bound on the expected return. The general idea is that the uncertainty will be high for OOD actions and low for IOD data, allowing the agent to avoid OOD actions if it maximizes this lower bound on the expected return. The main benefit of these uncertainty-aware methods is that they only depend on the amount of information in the data set and thus are independent of the data collection strategy. This property makes the uncertainty-aware methods resilient against statistical issues such as sampling errors and ensures that these methods have information-theoretical optimal regret bounds [9, 22, 38]. Thus, an agent should *know what it does not know* in the ideal case by measuring how uncertain it is about a specific state-action pair evaluation. Although it is clear how to estimate this uncertainty in tabular and linear settings [9, 22, 38], it is still an open problem to estimate uncertainty in the deep RL setting. That is why in this thesis, we explore various existing deep learning-based uncertainty estimation techniques with the aim to combine them with existing deep RL methods to create an uncertainty-aware offline deep RL algorithm.

1.2. Research Objective

This thesis aims to explore how deep RL agents can be instilled with the ability to *know what they do not know* using uncertainty estimation techniques. We aim to achieve this goal by structuring our research around the following primary research questions:

1. Which theoretical and implementation-related properties make deep uncertainty estimation techniques suitable for uncertainty-aware offline deep RL algorithms?

2. Does our resulting model-free uncertainty-aware offline deep RL algorithm experimentally match the theoretically predicted properties of uncertainty-aware algorithms?

To answer the first primary research question, we will survey the offline RL, deep RL, and deep uncertainty estimation literature to answer the following sub-research questions:

1.1 What is the pessimism principle, and what are the theoretical requirements it imposes on the uncertainty estimation techniques?

1.2 What are the implementation requirements imposed by deep RL methods on deep uncertainty estimation techniques to ensure learning stability?

1.3 Which existing deep uncertainty estimation techniques match these requirements and can be used to create a model-free uncertainty-aware offline deep RL algorithm?

Using the answers for this first primary research question, we will design and create two model-free uncertainty-aware offline deep RL algorithms. The first algorithm will specifically be designed for discrete action spaces, while the second algorithm will be designed for continuous action spaces. Both algorithms will be based on the deep uncertainty estimation technique that was found most suitable according to primary research question 1. To answer the final primary research question, we will evaluate our algorithm by answering the following sub-research questions:

2.1 Does our uncertainty-aware offline deep RL algorithm adhere to the theoretically predicted properties, such as not overestimating the expected return and being independent of the data collection strategy?

2.2 How well does our method perform compared to prior offline deep RL methods?

1.3. Contributions

The main contribution of this thesis is the model-free uncertainty-aware offline deep RL algorithm named PEssimistic ensemBLe (PEBL) [1]. This method integrates an ensemble-based epistemic uncertainty estimation technique with deep RL, instilling the resulting agent with the ability to *know what it does not know*. This deep epistemic uncertainty technique has been chosen after an extensive survey, and the resulting PEBL algorithm has been evaluated extensively. Therefore, we split our remaining contributions into *survey* and *experimental* related contributions. Our *survey* contributions are as follows:

1. We provide the reader with an overview of the offline (deep) RL problem.
2. We provide the reader with an overview of the pessimism framework and discuss how it can find information-theoretical optimal solutions for the offline RL problem.
3. We provide the reader with an overview of the theoretical and implementation requirements deep uncertainty estimation techniques must fulfill to make them suitable for a uncertainty-aware offline deep RL algorithm.

Our *experimental* contributions are as follows:

1. We show experimentally how this PEBL algorithm compares to the theoretically predicted properties of uncertainty-aware offline RL algorithms.
2. We show experimentally how the PEBL algorithm compares against state-of-the-art methods.

1.4. Outline

The remainder of this work is organized as follows. Chapter 2 outlines important background information and concepts concerning both online and offline deep RL and the necessary background for deep uncertainty estimation. Chapter 3 discusses the fundamental challenges of offline RL and how uncertainty-aware pessimism can solve these issues. Chapter 4 discusses the difficulties of integrating existing uncertainty estimation techniques from the supervised learning field in the offline RL field and which methods are most suitable. In chapter 5, we discuss our proposed algorithm in detail. Chapter 6 informs the reader how our methods relates to prior work in the offline deep RL field. Chapter 7 will evaluate our algorithm against its theoretically predicted properties and existing offline deep RL methods. Finally, Chapter 8 provides a summary of our findings and directions for future work.

[1] An open-source implementation is available at: github.com/j0rd1smit/PEBL.

2

Background

In this chapter, we introduced the mathematical formalism and our notations for the concepts in this thesis. We do this by first introducing the mathematical formalism of online reinforcement learning (RL) and its deep learning-based algorithms. Using these definitions, we define the offline reinforcement learning problem setting, where the goal is to learn a near-optimal policy from a fixed pre-collected data set. Finally, we conclude this chapter by introducing the mathematical formalism for deep uncertainty estimation, which we will use in this thesis to design a uncertainty-aware offline deep RL algorithm.

2.1. Reinforcement learning

Reinforcement learning (RL) is a machine learning paradigm for solving sequential-decision making problems. In RL, sequential-decision making problems are typically formalized as an Markov decision process (MDP). An MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, d_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward function, $\mathcal{T}$ is the transition function, $d_0(s)$ is a distribution over the initial states S_0, and γ is the discount factor. All these components work together as follows: an agent starts in the initial state s_0, sampled from the initial state distribution d_0. At time step t, the agent receives the state s_t and then selects and executes the action a_t, which causes the environment to transition according to $\mathcal{T}(s_t|a_t, s_{t+1})$, which is a function that maps state-action pairs to a distribution over next states $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. The agent enters the next state s_{t+1}, and receives the reward r_t according to the reward function $\mathcal{R}(s_t, a_t)$, where $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Note that both the reward and transition function in this process only depends on the current state and action. This property is known as the Markov property.

In an MDP, the reward r_t describes the desirability of taking action a_t in state s_t. The eventual goal in an MDP is to maximize the total amount of obtained reward. To do this, the agent learns a policy $\pi(a_t|s_t)$, which is a distribution over actions conditioned on the current state. The performance of this policy, denoted by $J(\pi)$, is measured as the expected return the agent will obtain in the MDP over the initial state distribution d_0:

$$G_t(s_t) = \sum_{k=t+1}^{H} \gamma^{t-t-1} r_k \tag{2.1}$$

$$J(\pi) = \mathbb{E}_{s_0 \sim d_0} \left[G_0(s_0) \right] \tag{2.2}$$

In this equation, $G_t(s_t)$ is the discounted episodic return when starting in state s_t at time step t, $\gamma \in (0, 1]$ is a discount factor that ensures that the return remains finite and controls the trade-off between short and long-term rewards, and H is the number of steps before the environment is terminated, which can be ∞. When $H = \infty$, we refer to the task as a continuous infinite-horizon task. Otherwise, we refer to the task as an episodic task.

For every policy π in an MDP, there exists a value function. These value functions estimate how good it is for a policy to be in a specific state based on its expected return. Formally, the value function

of a state s under policy π, denoted by $V^\pi(s)$, is defined as the expected return when following policy π from state s:

$$V^\pi(s) = \mathbb{E}_\pi \left[G(s_t) | s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{H} \gamma^k r_{t+k+1} | s_t = s \right] \tag{2.3}$$

Similarly, the state-action value function under policy π, denoted by $Q^\pi(s_t, a_t)$, is defined as the expected return for taking action a_t in starting state s_t and following the policy π from state s_{t+1} onwards:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G(s_t) | s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{H} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right] \tag{2.4}$$

Note that by definition, the value of terminal states is always zero for both the state and state-action value function. In the remainder of this thesis, we will refer to this state-action value function as the Q-function. Using these definitions, it is possible to define the optimal policy π^* as the policy that maximizes the value function and thus solves the MDP. The optimal policy's state and state-action value functions are denoted by $V^*(s)$ and $Q^*(s, a)$. This optimal policy can be found using dynamic programming algorithms such as *value iteration* and *policy iteration* [42]. However, this is only possible if the agent has access to the reward and transition function of the MDP, which is not always possible. In these situations, we have to resort to RL algorithms that can utilize the agent-environment interface framework.

The agent-environment interface is a framework for RL problems whereby the reward and transition function of the MDP are unknown to the algorithm or agent (Figure 2.1). In this framework, the agent receives the initial state $s_0 \sim d_0$ from the environment. Then at every time step t, the agent selects an action a_t based on the observed state s_t and sends this to the environment. The environment processes this action according to its internal MDP and provides the agent with the tuple (r_t, d_t, s_{t+1}), where r_t is the obtained reward, s_{t+1} is the newly observed state, and d_t is a terminal state indicator, which is 1 if s_{t+1} is a terminal state and 0 otherwise. These interactions with the environment result in experience tuples $(s_t, a_t, r_t, d_t, s_{t+1})$, which can be used to learn the underlying MDP of the environment.

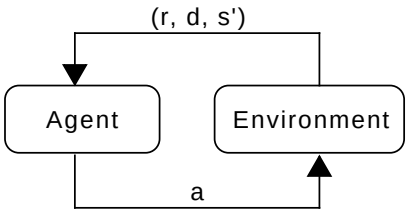


Figure 2.1: The agent-environment interface. The agent sends action a to the environment, which changes the internal state of the environment based on the underlying MDP. The environment provides the agent a reward r, a terminal state flag d, and the new state s based on this transition.

One way to utilize the experiences obtained from the agent-environment loop is to learn a value function. This approach is called model-free RL since the value function indirectly models the underlying MDP. To learn this value function, we need an estimate of the return. Most commonly, this estimate is obtained using the recursive property of the value function know as the Bellman equation:

$$\hat{Q}^\pi(s_t, a_t) = r_t + \gamma \hat{Q}^\pi \left(s_{t+1}, \pi(s_{t+1}) \right) \tag{2.5}$$

In this equation, $\hat{Q}$ is the current estimate of the state-action value function, and the value estimate for the next state is obtained by bootstrapping the current value function. Using this property, it is possible to learn the value function of a policy by minimizing the mean squared Bellman error (MSBE). This error is defined as the difference between the current predictions and the estimates of the returns

in the next state:

$$\mathcal{L} = \frac{1}{N} \sum_{\mathcal{D}} ||(y_t - \hat{Q}(s_t, a_t))||^2 \tag{2.6}$$

$$y_t = r_t + d_t \cdot \gamma \cdot \hat{Q}^{\pi} \left(s_{t+1}, \pi(s_{t+1})\right) \tag{2.7}$$

In these equations, y_t is the temporal difference target in the MSBE, N is the number of experiences in the data set $\mathcal{D}$, and $(s_t, a_t, r_t, d_t, s_{t+1})$ is a transition in the data set where $d_t = 0$ if s_{t+1} is a terminal state, $d_t = 1$ in all other cases. This procedure can either be used to learn an estimator for the value function of the policy that generated the data or to learn an estimator for another policy. When the estimator learns the value function of the policy that generated the data, it is called an on-policy estimator, and in the other case, it is called an off-policy estimator. This thesis focuses only on off-policy value estimators because they theoretically work even with experiences collected by different policies [42], which is an essential property for offline RL.

An alternative approach to utilize the experiences obtained from the agent-environment loop is to directly learn the reward and transition function of the underlying MDP. This approach is called *model-based* RL. There are many possible approaches to model-based RL. However, the main differences between these approaches lay in how they utilize their approximated versions of the reward and transition function. For example, a common way to utilize these approximated functions is to combine them with a planning algorithm such as value iteration to plan a trajectory that maximizes the expected return. Alternatively, it is also possible to use the approximated versions of the reward and transition function to generate additional syntactic experiences. These additional experiences can then be used to learn a policy using a model-free RL algorithm. These are only two possible approaches to model-based RL. For additional model-based approaches, we refer the reader to chapter 8 of [42] and to the survey [36].

Both these model-free and model-based approaches make it possible to learn a (near) optimal policy even if the reward function and transition function of the MDP is unknown to the agent. However, these approaches do not address the run time complexity issues when the state-action space becomes too large. The next section will discuss how function approximation approaches using deep learning can address this issue.

2.2. Deep reinforcement learning

The theory for RL was mostly developed in the tabular setting, where every state-action pair can be stored explicitly in a table. However, when the number of states becomes too large, as in continuous state spaces, it is no longer possible to explicitly store all these values in a table. Worst still, even if we could store all these values, most of these states will never be explicitly encountered by the agent. Therefore, we require a function that approximates the value function and policy with a manageable amount of parameters and can generalize to some extent across the state-space.
In the machine learning field, many different function approximators fulfill these requirements. However, in recent years, it has become popular to approximate the value function and policy with neural networks, often referred to as deep RL. These neural networks are non-linear function approximators that can represent any smooth function when given enough parameters [15, 18]. Using these neural networks, it is possible to approximate the Q-function and optimal policy of an MDP even in continuous state spaces. In this thesis, we refer to this approximated Q-function as $Q(s, a; \theta)$, while we refer to the approximated policy as $\pi(a|s; \phi)$, where θ and ϕ are the parameter vectors of the respected neural networks. Although the neural network-based function approximation approach sounds straightforward, there are some particularities that make deep RL difficult. Therefore, section 2.2.1 will discuss the deep RL components that address these issues, focusing on the components relevant to offline RL. Once we have established these common deep RL components, section 2.2.2 will discuss the common deep RL algorithms we build upon in this thesis.

2.2.1. Common deep RL components

In recent years, many new deep RL algorithms have been proposed. All these algorithms have to address the challenges that arise from combining RL and neural networks. To do this, most deep RL algorithms

use the same components that solve the most common issues in deep RL. This section will discuss the most relevant components for this thesis, including the experience replay, target network, importance sampling, and Lagrangian dual gradient descent.

Experience replay
The gradient-based optimization techniques used to train neural networks typically require independent and identically distributed (i.i.d.) estimates of the gradients. However, in deep RL, this assumption does not always hold. For example, all the data in a batch might come from a single episode. In this situation, the experiences in the batch will result in highly correlated gradient estimates. To prevent these issues, deep RL methods typically use an experience replay buffer. This buffer stores the highly correlated experiences and makes it possible to randomly sample batches from different episodes. This random sampling breaks the correlation and allows the algorithm to satisfy the i.i.d. estimates of the gradients assumption [32, 33].

Target network
In deep RL, the Q-function is typically learned by minimizing the MSBE. This error is defined as the difference between the network predictions and the estimates of the returns in the next state (Equation 2.6). The temporal difference targets in the process are obtained by bootstrapping the approximated the Q-function in the next state, making these target highly correlated with the predictions that are being optimized. This correlation is problematic for neural networks because parameters updates in a neural network have a global effect. For example, a poor update in one area of the state space can affect the prediction and temporal difference targets in other areas. Combined with the correlation between the predictions and the targets, this effect can create an unstable feedback loop. The effects of this feedback loop can be mitigated if the temporal difference targets are obtained using a target network [32, 33]. This target network is an older version of the current Q-function network with parameters θ'. The parameters of this target network can either be copied every c steps from the current Q-function or be updated every step using a slow-moving average. Either way, this approach reduces the correlation and allows the Q-function network to learn from a more stable target, resulting in a more stable learning process.

Importance sampling
Some RL algorithms assume that the current policy has generated the experiences they are learning from. These algorithms are called on-policy algorithms, while algorithms that do not make this assumption are called off-policy algorithms. Sometimes, it is desirable to turn an on-policy algorithm into an off-policy algorithm, e.g., when an on-policy algorithm has desirable properties, but no on-policy data is available. In RL, this can be done using importance sampling [42]. Importance sampling is a technique that estimates the expected value under one distribution based on samples from another distribution. Using this technique, it is possible to re-weight experiences such that it appears like it was generated by the current policy, making it possible for on-policy algorithms to learn from off-policy data. This ability is beneficial for deep RL because most continuous action-space algorithms are on-policy algorithms. Using importance sampling, it is possible to train these continuous action-space algorithms using off-policy data. An additional benefit of importance sampling is that it reduces the bias in the return estimate introduced by the bootstrapping procedure [37]. However, a major disadvantage of importance sampling is that the importance weights will approach zero when the difference between the policies becomes too large, making it impossible to learn from the sample. If this happens for a lot of samples, we will waste a lot of computation. Therefore, importance sampling should be used with care.

Lagrangian dual gradient descent
Deep RL algorithms often have many hyper-parameters that impact the training process in complex and sometimes unforeseen ways. Tuning these hyper-parameters can be very difficult because their ideal value often varies during the training process. However, some deep RL algorithms remove the need to manually tune these hyper-parameters using a technique called Lagrangian dual gradient descent [16, 27]. Lagrangian dual gradient descent is a method that optimizes an objective under a constraint:

$$\min_x f(x) \text{ s.t. } C(x) = 0 \tag{2.8}$$

In this equation, $f(x)$ is the original optimization objective of the RL algorithm, e.g., maximizing the expected return, and $C(x) = 0$ is a constraint that expresses the desirable effect of the hyper-parameter, e.g., a minimum amount of entropy in the policy. The Lagrangian dual gradient descent method transforms this constrained objective into a Lagrangian dual function which can be optimized iteratively. Formally this problem is defined as follows:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda C(x) \tag{2.9}$$

$$g(\lambda) = \mathcal{L}(x^*(\lambda), \lambda), \text{ where } x^* = \underset{x}{\operatorname{argmin}}\, \mathcal{L}(x, \lambda) \tag{2.10}$$

In these equations, the Lagrangian multiplier λ is the scalar value of the hyper-parameter, $x^*(\lambda)$ is a function that transforms λ into an input for $f(x)$ that satisfies the constraint $C(x) = 0$, and $g(\lambda)$ is the lower bound dual optimization function of the original objective in Equation 2.8. Assuming the objective function $f(x)$ is convex, then the λ that maximizes $g(\lambda)$ will ensure that $x = x^*(\lambda)$ solves the constraint optimization objective in Equation 2.8 [5]. A pleasant property of this process is that it works iteratively. So if you keep taking small gradient-based optimization steps towards maximizing $g(\lambda)$ and minimizing $\mathcal{L}(x, \lambda)$, you will eventually optimize both the original objective and the hyper-parameter.

2.2.2. Deep RL algorithms

In this section, we will discuss the most relevant deep RL algorithms for this thesis. These algorithms address the common problems in deep RL using one or more of the components and techniques discussed in the previous section.

Deep Q-learning

The Deep Q-Network (DQN) algorithm is a discrete action-space algorithm, which aims to approximate the optimal Q-function $Q^*(s, a)$ using a neural network [32, 33]. Using this optimal Q-function it will select actions according to the optimal:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a; \theta) \tag{2.11}$$

To approximate this optimal Q-function, the DQN algorithm minimizes the MSBE using the temporal difference target:

$$y_t = r_t + (1 - d_t) \cdot \gamma \underset{a}{\operatorname{argmax}} \{Q(s_{t+1}, a; \theta')\} \tag{2.12}$$

In this equation, r_t is the reward at time step t, and d_t is a flag that indicates whether the episodes terminated or not at the t-th time step. Note that the Q-values in the next state are parameterized by θ' instead of θ, which indicates that these values are calculated using a target network. A target network is a version of the network with parameters θ' that lags L gradient updates behind the current version of the network. In the DQN algorithm, the target network is synced with the current network only every c steps.

Double Deep Q-learning

The Double Deep Q-Network (DDQN) algorithm is an extension upon DQN, which aims to reduce over-estimations caused by the max operator in the temporal difference target of the DQN algorithm (Equation 2.12). These over-estimations can cause instability issues, which limits the final performance of the algorithm [47]. The DDQN algorithm prevents these issues by selecting the action that maximizes the current network instead of the target network. The target network is still used to estimate the Q-value of this state-action pair. This means that the MSBE temporal difference target changes into:

$$y_t = r_t + (1 - d_t) \cdot \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}} \{Q(s_{t+1}, a; \theta); \theta'\}) \tag{2.13}$$

In this equation, θ is the parameters vector of the current neural network, θ is the parameters vector of the target neural network, r_t is the reward at time step t, and d_t is a flag indicating whether the episodes terminated or not at the t-th time step.

Soft-actor critic
The Soft Actor Critic (SAC) algorithm is a deep actor-critic algorithm that is very popular for continuous action spaces [16]. This algorithm learns both a policy and value function, which we refer to as $\pi(a|s;\phi)$ and $Q(s,a;\theta)$, whereby θ and ϕ are their respective parameter vectors. The SAC algorithm can learn this value function and policy from off-policy data using the importance sampling technique from the previous section. The general idea behind the SAC algorithm is that an agent should spreads its bets, by acting as randomly as possible, if multiple actions have the same expected return. This algorithms implements this idea in both the value learning and policy learning process. In the value learning process it implements the spreading its bets idea by adding an entropy bonus to the value function, which changes the MSBE temporal difference target from Equation 2.6 to:

$$y_t = r_t + (1 - d_t) \cdot \gamma \min_{j=1,2} \{Q(s_{t+1}, \tilde{a}_{t+1}; \theta'_j)\} - \alpha \log \pi(\tilde{a}_{t+1}|s_{t+1};\phi) \tag{2.14}$$

while, the SAC algorithm implements this idea in the policy learning process by changing the policy optimization objective to:

$$\operatorname*{argmin}_{\phi} \alpha \log \pi(\tilde{a}_t, s_t;\phi) - \min_{i=1,2} \{Q(s_t, \tilde{a}_t; \theta_i)\}, \text{ where } \tilde{a}_t \sim \pi(a|s_t;\phi) \tag{2.15}$$

In these equations, r_t is the reward at time step t, and d_t is a flag indicating whether the episodes terminated or not at the t-th time step, $\tilde{a}$ is sampled from the learned policy $\pi_\phi(.|s)$, θ'_i is the parameter vector of the i-th target network, and α is the entropy trade-off hyper-parameter, which controls the amount of entropy in the policy. This entropy trade-off parameter is rather difficult to tune manually because the ideal amount of entropy in the policy varies during the training process. Therefore, it is common to automatically tune this hyper-parameter using the Lagrangian dual gradient descent technique from the previous section [16]. Similar to the DDQN algorithm [47], the SAC algorithm tries to minimize over-estimations caused by the function approximation. It does this by learning two Q-value functions, parameterized by θ_1 and θ_2. It uses the lowerest Q-value estimation for its policy losses and its MSBE temporal difference target to prevent over-estimations.

2.3. Offline reinforcement learning

The previous section showed how deep neural network and RL algorithms can be combined to create deep RL algorithms that work in continuous state and action spaces. An interesting property of the deep neural networks used in these approaches is that they perform better when the amount of data in their training's data set increases [15]. Using the traditional online data gathering approach from the previous sections, these large data sets must be gathered from scratch every time a deep RL algorithm is trained. One can easily imagine that this approach quickly becomes very expensive when the amount of required data keeps increasing. Therefore, offline reinforcement learning aims to address this issue by reformulating the reinforcement learning problem into a data-driven problem.

Offline RL is the problem of choosing how to act using only a fixed amount of data from the environment. In offline RL, the goal is still to maximize the expected return (Equation 2.2). However, the key difference from *online* RL is that the agent can no longer collect new information from the environment (Figure 2.2). The agent is only allowed to interact with the environment at test time to evaluate its *final* performance. Thus, the agent must learn a policy that maximizes the expected return in the real environment consisting of transitions, denoted by $J_{\mathcal{M}}(\pi)$, using only a static data set, denoted by $\mathcal{D} = \{(s^i_t, a^i_t, s^i_{t+1}, r^i_t)\}$. From this data set, the offline RL algorithm must learn to represent the transition and reward function of the MDP that generated the data set either explicitly or implicitly in the model-free case.

The name offline RL makes it seem like it is best understood as a variant of RL. However, the core issues of RL, such as the exploration vs. exploitation trade-off, are not present in offline RL. Therefore, it is better to view offline RL as dynamic programming from a data set. In the dynamic programming setting, we are guaranteed to find the optimal policy if we have access to the reward and transition function. However, in offline RL, we do not have access to the reward and transition function. Instead, we only have access to a data set that contains information about the reward and transition function. This information is not necessarily complete. For example, it is possible that the data set is empty or

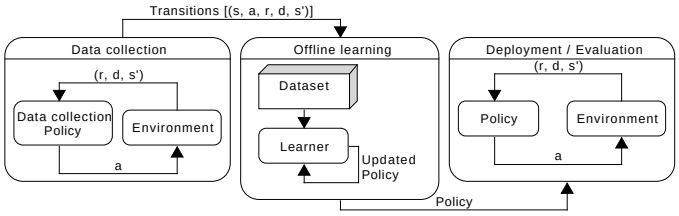


Figure 2.2: A visualization of the learning process in offline reinforcement learning.

only contains information about a specific part of the state-action space. Therefore, we cannot guarantee we can recover the true reward and transition function. As a result, we cannot guarantee we can find the optimal policy and optimal value function. Instead, the goal in offline RL should be to find a data optimal policy, denoted by $\pi^*_{\mathcal{D}}$, which achieves the highest possible expected return in the real environment while requiring only the data available in the data set.

Ideally, offline RL algorithms should be able to find this data optimal policy for any data set that obeys the underlying properties of the MDP. However, specific data set properties can make some offline RL data sets harder than others in practice. Therefore, we often classify offline RL data sets into the following non-mutually exclusive data set types:

Expert and random data sets
Expert and random data sets are the two extremes on the data collection strategy spectrum. Expert data sets have been collected by an expert, near-optimal policy. These expert data sets typically have a very biased sample of the state-action space due to the absence of non-rewarding transitions [19]. This property makes expert data sets very difficult for offline RL methods that assume a uniform sample distribution, such as online deep RL methods [9, 38]. In contrast, random data sets have been collected using a policy that samples actions at random. These random data sets typically have a much more uniformly distributed sample of the state-action space due to the presence of non-rewarding transitions. This property makes random data sets very difficult for offline RL methods, such as behavior cloning and some policy-constrained methods, that assume any sort of near optimality in the data collection strategy [9, 10]. Although expert and random data sets are the two extremes on the expert vs. non-expert data set composition spectrum, the exact point on this spectrum is typically unknown for an offline RL problem. Therefore, in an ideal situation, an offline RL algorithm should be able to learn from any data set on this spectrum.

Undirected data sets
Undirected data sets are data sets where the demonstrator's behavior does **not** align with the goal to maximize the expected discounted return. A typical example of an undirected data set is a data set that has been collected by an agent with the goal to explore the entire environment, while the offline RL agent has the goal to find the fastest route to a specific state in the environment. This misalignment makes these data sets very difficult to solve for offline RL methods such as behavior cloning and some policy-constrained methods that assume that the empirical behavior of the demonstrator aligns with its own reward function [9, 10]. However, in theory, offline RL methods should be able to learn from undirected data sets because these data sets are still consistent with an MDP. The only difference is that the data collection strategy maximized a different reward function. Therefore, in an ideal situation, an offline RL algorithm should be able to learn from any undirected data set.

Mixture data sets
Mixture data sets are data sets where two or more different data collection policies have collected the transitions in the data set. The difficulty of mixture data sets is that the empirical policy is no longer approximately equal to the data collection policy. This property is troublesome for methods that make this specific assumption, such as behavior cloning and some policy-constrained methods [9, 29]. This is unfortunate because real-world data set will most likely be collected by observing multiple agents (e.g.,

humans or robots) in parallel, making the ability to work with mixture data sets essential for practical offline RL.

2.4. Uncertainty estimation

This thesis aims to create an offline RL algorithm that is aware of the uncertainty in its learned value function. However, before we can create this algorithm, let us first discuss the different types of uncertainties in the deep learning field. In deep learning, uncertainty occurs when the test and training data are mismatched or when the labels overlap due to noise in the data. Based on these characteristics, we can decompose a model's predictive uncertainty σ_p into two components: [1]:

$$\sigma_p = \sigma_a + \sigma_e \tag{2.16}$$

The first component in this equation is σ_a, which is called *aleatoric* uncertainty. This type of uncertainty arises from the stochasticity that is naturally present in the observations. Its key property is that it cannot be reduced by adding more data. A typical example of aleatoric uncertainty is a random or noisy reward function; adding more data from this function will *not* remove the noise in the observations. Besides aleatoric uncertainty, this type of uncertainty is sometimes also called irreducible or data uncertainty. However, for the remainder of this thesis, we will refer to it as aleatoric uncertainty.

The second type of uncertainty is called *epistemic* uncertainty. This type of uncertainty is an inherent property of the model because it describes what the model does not know due to limitations in the observed data. Due to this property, epistemic uncertainty tends to be higher in areas of low data density. Formally it has been shown that epistemic uncertainty is approximately inversely proportional to the density $p(x)$ of the training data [7]:

$$\sigma_e(x) \propto p^{-1}(x) \tag{2.17}$$

Due to this property, it is possible to reduce epistemic uncertainty by adding more data. Therefore, is epistemic uncertainty is sometimes also referred to as knowledge uncertainty. However, for the remainder of this thesis, we will refer to it as epistemic uncertainty.

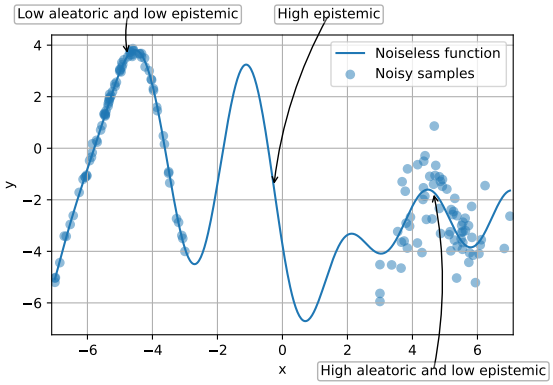


Figure 2.3: Visualization of the main differences between aleatoric and epistemic uncertainty. Aleatoric uncertainty is high in areas with very noisy data and low in areas with low data noise. In contrast, epistemic uncertainty is high in areas with little or no data and low in areas with large amounts of data.

s

3

Motivation for pessimism in offline RL

The previous chapter formalized the definitions of both *offline* reinforcement learning (RL) and *online off-policy* RL. Although these two approaches seem similar on the surface, the key difference is that *offline* RL algorithms have no control over their data collection strategy, whereas off-policy methods do. This subtle difference has immense implications because statistical issues introduced by the finiteness of the data set can no longer be ignored in *offline* RL. In this chapter, we will discuss the main difficulties that arise due to these statistical issues. After identifying these issues, we conclude this chapter by introducing the pessimism principle and explaining how it can prevent these statistical issues.

3.1. Difficulties in offline RL

Offline RL can best be viewed as dynamic programming from a data set. In the dynamic programming setting, we are guaranteed to find the optimal policy if we have access to the reward and transition function. However, in offline RL, we do not have access to the reward and transition function. Instead, the agent only has access to a fixed-sized data set containing information about the reward and transition function. This information is not necessarily complete. For example, it is possible that the data set is empty or only contains information about a specific part of the state-action space. This missing information is not a problem in *online* RL since the agent can sample the environment for additional data to fill in these gaps. However, this approach is not possible in *offline* RL since no interaction with the environment is allowed. Therefore, we cannot guarantee we can recover the true reward and transition function in offline RL using only the provided data set. As a result, we **cannot** guarantee we can find the optimal policy and optimal value function. Instead, the goal in offline RL should be to find a data optimal policy, which achieves the highest possible expected return in the real environment while requiring only the information available in the data set. Formally, this objective can be formalized as:

$$\text{minimize } J_{\mathcal{M}}(\pi^*_{\mathcal{M}}) - J_{\mathcal{M}}(\pi^*_{\mathcal{D}}) \tag{3.1}$$

$$\pi^*_{\mathcal{M}} = \operatorname*{argmax}_{\pi_{\mathcal{M}}} J_{\mathcal{M}}(\pi_{\mathcal{M}}) \tag{3.2}$$

$$\pi^*_{\mathcal{D}} = \operatorname*{argmax}_{\pi_D} J_{\mathcal{D}}(\pi_D) \tag{3.3}$$

In these equations, $\pi^*_{\mathcal{M}}$ is the optimal policy in the real environment, $\pi^*_{\mathcal{D}}$ is the data optimal policy, $J_{\mathcal{M}}(\pi)$ is the expected return in the real environment, and $J_{\mathcal{D}}(\pi)$ is a proxy objective that can be evaluated using only the information in data set $\mathcal{D}$. These equations show that the main difficulty of offline RL is: how do we choose a good proxy objective such that the regret in Equation 3.1 is minimized?

Before we can answer this question, we must first look at the relationship between the real objective and the proxy objective and their effect on the regret in Equation 3.1. Buckman et al. [9] showed that this relationship obeys the following regret bound:

$$J_{\mathcal{M}}(\pi^*_{\mathcal{M}}) - J_{\mathcal{M}}(\pi^*_{\mathcal{D}}) \leq \overbrace{\inf_{\pi_{\mathcal{D}}} \left[J_{\mathcal{M}}(\pi^*_{\mathcal{M}}) - J_{\mathcal{D}}(\pi_D)\right]}^{\text{(A)}} + \overbrace{\sup_{\pi_{\mathcal{D}}} \left[J_{\mathcal{D}}(\pi_D) - J_{\mathcal{M}}(\pi_D)\right]}^{(B)} \tag{3.4}$$

In this equation, the term labeled (A) reflects the accuracy of the proxy optimization objective on a near-optimal policy. This term will be small whenever there is at least one reasonable policy that is not underestimated by the proxy objective [9]. In contrast, the term (B) corresponds to the largest overestimation error of any policy. Since term (B) contains a supremum, it will be small only when the proxy objective does not overestimate any policy. Even a single overestimation can cause significant regret [9]. Therefore, it is essential in offline RL to select a proxy objective that has the following two properties. Firstly, the proxy objective can only use the information available in the data set. Secondly, the proxy objective should underestimate at least one near-optimal policy as little as possible while also overestimating all other policies as little as possible. So based on this observation, the key question is how do we find such a proxy objective function for offline RL?

In offline RL, we typically use an off-policy value function as our proxy objective because it predicts the expected return for the current policy, while it can also learn from any pre-collected transitions [29, 42]. However, recent theoretical results have shown that naively applying this off-policy learning approach can result in a value function that over-estimates the true return [9, 38], which is undesirable given Equation 3.4. This overestimation effect mainly happens when the state-action space is not sufficiently sampled to recover the underlying reward and transition functions. For example, a data set collected using an expert demonstrator will contain only information about the expert demonstrator's policy, while it contains no information about other policies. Therefore, the areas of the state-action space which were not covered by the data set will be regarded as low-information areas. In these low-information areas, the learned value function can be arbitrarily bad and be potentially overestimated due to missing information and sampling errors. In *online* RL, this is **not** a problem because if the agent overestimates these areas, it will eventually sample them and collect additional information that will help it to learn the correct reward and transition function in these low-information areas. However, in *offline* RL, interactions with the environment are not allowed, meaning that the value function will remain arbitrarily bad and potentially overestimated in these low information areas. This is problematic for offline RL algorithms because they aim to learn a policy that maximizes this approximated value function. Formally this policy optimization objective is defined as:

$$\pi_{\mathcal{D}} \leftarrow \underset{\pi_{\mathcal{D}}}{\operatorname{argmax}} \hat{Q}^{\pi_{\mathcal{D}}}(s,a) \, \forall s \in \mathcal{D} \tag{3.5}$$

The problem with this objective is that it aims to find a policy that maximizes the Q-function over all possible actions. This is problematic because the Q-function can be arbitrarily bad for actions not or poorly covered in the data set. These arbitrarily bad Q-function predictions in low information regions combined with the maximization objective result in an agent that will learn a policy that will be biased towards actions with overestimated Q-values. By itself, this problem would only be minor since the regret will be bounded by the proxy optimization bound in Equation 3.4. However, in off-policy algorithms, this policy is also used to obtain the temporal difference target, which will be used to improve the value function by minimizing the mean squared Bellman error (MSBE) iteratively. Formally this MSBE optimization objective is defined as:

$$\theta \leftarrow \underset{\theta}{\operatorname{argmin}} \frac{1}{|D|} ||\hat{Q}^{\pi_D}(s_t, a_t; \theta) - y_t||^2 \tag{3.6}$$

$$y_t = r_t + d_t \cdot \gamma \cdot \hat{Q}^{\pi_D}(s_{t+1}, \pi_D(s_{t+1}); \theta) \text{ where } (s_t, a_t, r_t, d_t, s_{t+1}) \in \mathcal{D}$$

In this equation, the red $\pi_D(s_{t+1})$ is the action obtain from the learned policy that is biased towards actions with overestimated Q-value in the low information regions, and y_t is the likely overestimated temporal difference target that has been obtained using this policy. This MSBE optimization objective, combined with the policy optimization objective, means that the initially small overestimations of the value function will be prioritized during the value learning process in the low-information regions. Due to this effect, the value function overestimation will compound over time, which can result in divergent behavior. This effect is undesirable since we just showed in Equation 3.4 that overestimation is a highly undesirable property that results in large regret bounds. Empirically these results have also been observed many times when a naive off-policy algorithm (e.g., Deep Q-Network (DQN), Double Deep Q-Network (DDQN), or Soft Actor Critic (SAC)) gets provided with a fixed data set that does not contain enough information to recover the dynamics of the real Markov decision process (MDP) and

is not allowed to gather any additional data [10, 12, 19, 25, 29]. Therefore, it is essential in offline RL that the policy and value function optimization objectives are constructed so that they both introduce as little overestimation as possible in the low-information regions while still underestimating the true value as little as possible in the high-information regions. In the next section, we will discuss how this goal can be obtained using the pessimism principle.

3.2. The pessimism principle

The previous section showed that missing information in the data set and poorly selected optimization objectives result in highly overestimated value functions in offline reinforcement learning (RL). These overestimated value functions are problematic because they result in unrealistic policies with poor regret bounds that seem rewarding based on the overestimated value function but are not in the real environment. To avoid this issue, algorithms can follow the pessimism principle, which states that: *"we should choose the policy which acts optimally in the worst possible world"* [9]. In offline RL, this principle translates to learning a policy that maximizes the worst-case return by being pessimistic towards the expected return based on the observed data. This pessimism principle can be integrated into off-policy algorithms by learning a value function that is pessimistic towards low-information regions. Formally, this means that we change the Bellman equation (Equation 2.5) into the following pessimistic Bellman equation:

$$\hat{Q}^{\pi_D}(s_t, a_t) = r_t + \gamma \left(\hat{Q}^{\pi_D}(s_{t+1}, a_{t+1}) - u_{\mathcal{D}}^{\pi_D}(s_{t+1}, a_{t+1}) \right) \text{ where } a_{t+1} \sim \pi_D(s_{t+1}) \tag{3.7}$$

This equation is exactly the same as the original Bellman equation except for the addition of the pessimistic penalty, denoted by $u_{\mathcal{D}}^{\pi_D}(s_{t+1}, a_{t+1})$. This pessimistic penalty allows us to control the amount of overestimation and underestimation in the learned value function. Different choices for this pessimistic penalty result in different overestimation and underestimation trade-offs, which directly affect the regret bound formulated in Equation 3.4. However, generally speaking, we can divide the choices for this pessimistic penalty into three choices, which result in the naive, policy-constrained, and uncertainty-aware algorithmic families [9]. In the following sections, we will discuss the differences between these pessimistic penalties as well as the unique properties of each algorithmic family.

The naive algorithmic family

The naive algorithmic family does not make a distinction between low and high information regions in the state-action spaces. They assume that all data points are equally informative. Formally, this means that these algorithms choose a pessimistic penalty that is independent of the state-action pair:

$$u_{\mathcal{D}}^{\pi_D}(s_t, a_t) = c \tag{3.8}$$

where typically $c = 0$. Due to this constant pessimistic penalty, the algorithms in this family are unable to distinguish between low and high information regions, which means that their learned value function is equivalent to the maximum likelihood estimate of the value function [9]. Therefore, this naive algorithmic family will always relatively overestimate the expected return in the low-information regions as long as c is constant, which means that no constant value for c will improve their regret bound in Equation 3.4. The only way to remove this relative overestimation is by ensuring that the data set contains a uniform sample of the state-action space. This requirement is undesirable since offline RL algorithms should ideally be able to learn from any data set. However, even if this data set requirement was considered acceptable, the naive algorithms still need huge and diverse data sets before they can recover the optimal policy due to the combinatorial size of the state-action space [2, 9]. This effect can be observed in the replicated result of Buckman et al. [9] in Figure 3.1, which shows that the naive algorithmic family performs badly on small and narrowly sampled data sets, while it starts to perform increasingly better as the number of sample and their diversity increases.

The policy-constrained algorithmic family

The policy-constrained algorithmic family distinguishes between low and high information regions based on the divergences from the empirical policy in the data sets. The main assumption here is that data points which are more likely under the empirical policy are more informative since they will occur more

often in the data set [9]. This penalty can be formalized as:

$$u_{\mathcal{D}}^{\pi_D}(s_t, a_t) = \frac{\delta(\pi_D, \pi_E)}{V_{max}^2} \tag{3.9}$$

In this equation, V_{max} is the maximum expected discounted return of the Markov decision process (MDP) that generated the data set, π_E is the empirical policy in the data set, and $\delta(\pi_D, \pi_E)$ is the total variation distance between the empirical policy and the policy learned using the data set. Due to this choice of constraint, these algorithms prefer policies that stay near the empirical policy. Therefore, these algorithms can be best understood as imitation learning algorithms, which permit minor deviations from the empirical policy. However, constraining the policy in such a way has one major disadvantage the learned policy will be unable to converge towards the optimal policy if the data set has not been gathered according to a near-optimal policy, even in the limit of infinite data [9] (Figure 3.1). This effect occurs because these algorithms conflate data with information. The empirical policy used in this approach is estimated based on the data concentrations in the data set. However, these data concentrations alone are not enough to determine if you know enough about the underlying MDP to deviate from the empirical policy. Despite this fact, this approach is quite common in the offline deep RL field [11, 19, 25, 27, 50] because estimating the divergence between the empirical policy and the learned policy is relatively easy in the deep learning setting.

The uncertainty-aware algorithms algorithmic family
The uncertainty-aware pessimistic algorithmic family selects their pessimistic penalty based on an agent's epistemic uncertainty, which measures an agent's knowledge about the true MDP. This uncertainty-aware pessimistic penalty can be formalized as an upper-bound on the difference between the real value function, denoted by $Q_{\mathcal{M}}(s_t, a_t)$, and the empirical maximum likelihood-based value function, denoted by $Q_{\mathcal{D}}(s_t, a_t)$ [9]:

$$|Q_{\mathcal{M}}(s_t, a_t) - Q_{\mathcal{D}}(s_t, a_t)| \leq c \cdot u_{\mathcal{D}}^{\pi_D}(s_t, a_t) \tag{3.10}$$

In this equation, c is a scaling parameter, and $u_{\mathcal{D}}^{\pi_D}(s_t, a_t)$ the uncertainty-based penalty. This penalty has the property that it is high in areas where the value function is uncertain, while the penalty approaches zero when the model is more certain about its value function prediction. Therefore, the pessimistic value function will be low in low information regions and close to the real value function in high information regions. With this property, the uncertainty-aware pessimistic offline RL algorithm will learn a policy that avoids low information regions and maximizes its expected discounted reward in the high information regions. This property allows uncertainty-aware algorithms to maximize the expected discounted reward in regions with sufficient information to recover the true (local) transition and reward function of the MDP while forcing it to follow the empirical policy in regions with insufficient information [9]. Due to this property, these algorithms achieve the theoretically best possible results even when the data set is only informative of a limited area of the state-action space, independent of the used data collection strategy [9, 22, 38].

In the tabular setting, it is possible to derive this uncertainty-aware penalty exactly using Hoeffding's inequality [9]. It is also possible to derive this penalty in the linear function approximation setting based on the uncertainty of a maximum likelihood linear regression-based function approximator [22]. However, there is currently no theoretical correct approach to derive this uncertainty-aware penalty exactly in the deep learning setting.

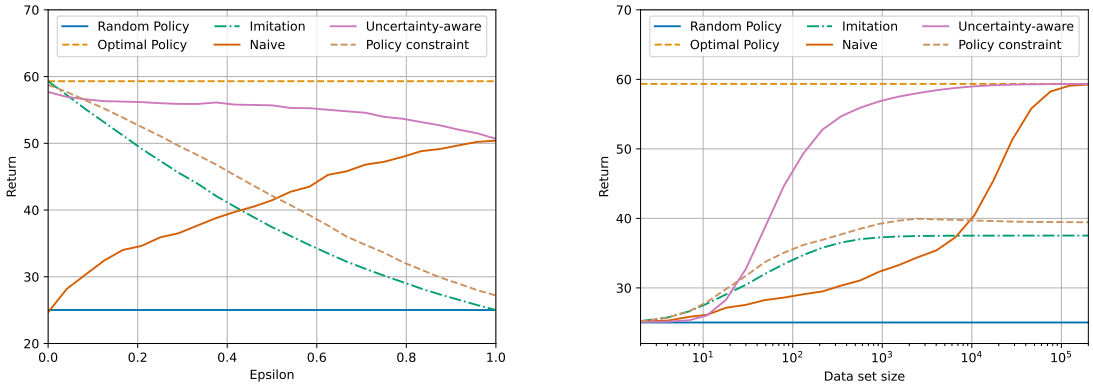
3.3. The need for uncertainty-based algorithms

Even though the policy-constrained and the uncertainty-aware pessimistic families seem different on the surface, theoretical analysis shows that they are, in fact, closely related. Theoretical results show that policy-constrained methods are uncertainty-aware algorithms that use a trivial value uncertainty function based on the maximum expected discounted return in the MDP [9]. Therefore, Equation 3.10 always holds if we substitute the policy constrain penalty from Equation 3.9:

$$|Q_{\mathcal{M}}(s_t, a_t) - Q_{\mathcal{D}}(s_t, a_t)| \leq c \cdot \frac{\delta(\pi_D, \pi_E)}{V_{max}^2} \tag{3.11}$$

However, uncertainty-aware algorithms that use an epistemic-based penalty are strictly better than their policy-constrained counterparts because, for any policy-constrained, we can always find a tighter epistemic uncertainty-aware penalty [9]. This insight has led to multiple theoretical works that prove the policies learned by uncertainty-aware algorithms are optimal and learn the theoretically best possible policy given any data sets. In contrast, the policies learned using policy-constrained methods do not have these properties [9, 22, 38]. Uncertainty-aware offline RL algorithms already exist for the tabular and linear function approximation settings. However, currently, there exists no uncertainty-aware offline deep RL method. The main reason for this is that no theoretical proof shows equation 3.10 holds for specific epistemic uncertainty estimation methods.

Even though there is no theoretical proof that uncertainty-aware offline RL algorithms are possible in the deep learning setting, their predicted properties are still desirable targets to aim for. Therefore the focus of this thesis is to experimentally explore which deep epistemic uncertainty estimation methods allow us to create offline deep RL methods with properties that match the theoretical properties of uncertainty-aware algorithms as close as possible.



(a) Performance on the expert vs. random data sets spectrum. Data consist of 2000 transitions and is samples using an epsilon greedy strategy.

(b) Performance for different data set sizes where the data is collected using an epsilon greedy strategy with $\epsilon = 0.5$.

Figure 3.1: Replicated experimental results of Buckman et al. (2020). These figures show the properties of different families of offline reinforcement learning methods in a tabular grid world. Both figures show that uncertainty-aware algorithms perform significantly better than all other families independent of the data set composition, while naive and policy-constrained methods depend heavily on the data set composition.

4

Uncertainty estimation techniques

In chapter 3, we established that there exists **no** mathematical proof that any deep uncertainty estimation technique matches the theoretical requirement of uncertainty-aware algorithms. However, it is hypothesized that in practice, it should be possible to approximate this requirement using deep epistemic uncertainty estimation techniques [9]. In this chapter, we explore various deep uncertainty estimation techniques and their epistemic uncertainty estimation capabilities. Furthermore, we also discuss the main differences between the supervised learning and reinforcement learning problem and how these differences impose implementation-related requirements on deep uncertainty estimation techniques. Finally, we concluded this chapter by selecting the most suitable uncertainty estimation technique for an uncertainty-aware offline reinforcement learning (RL) algorithm.

4.1. Methods

During this research, we considered multiple uncertainty estimation techniques. These methods were the Monte Carlo dropout method [14], the multi-headed bootstrap ensemble with random priors method, and the orthonormal certificates method [43].

4.1.1. Monte Carlo dropout

Monte Carlo dropout is a Bayesian inference technique used in deep learning to estimate the uncertainty in the model's prediction [14]. As the name suggests, Monte Carlo dropout is based on the dropout regularization technique [41]. Dropout works as a regularization technique by randomly masking or "dropping out" a certain percentage of the output units of the network's layers during each forward pass in the training process. This change makes the training process noisy, forcing nodes within a layer to probabilistically take on more or fewer inputs from the previous layer, which regularizes the network [41]. Normally, dropout is disabled after the training phase, and inference happens as if the dropout layers are not present. However, in Monte Carlo dropout, the dropout layers are left enabled at inference time because it allows us to estimate the uncertainty in the prediction. Monte Carlo dropout measures this uncertainty by making n predictions, each with its own dropout mask M_i, and measuring the standard deviation $\overline{\sigma}$ over these n predictions:

$$\overline{\mu}(x;\theta) = \frac{1}{n}\sum_{i=1}^{n} \hat{f}(x;\theta,M_i) \tag{4.1}$$

$$\overline{\sigma}(x;\theta) = \sqrt{\frac{\sum_{i=1}^{n}\left(\hat{f}(x;\theta,M_i) - \overline{\mu}(x;\theta)\right)^2}{n-1}} \tag{4.2}$$

The key idea here is that each subset of neurons that have not been dropped out defines a new network. Therefore, we can view the training process as training 2^m in different models simultaneously, where m is the number of neurons in the network [41]. In a situation where the model is uncertain, many of these 2^m different models will predict something different. At the same time, these models will mostly make similar predictions in situations where the model is certain. Thus, allowing us to measure the uncertainty in the prediction.

4.1.2. Multi-headed bootstrap ensemble with random priors

The multi-headed bootstrap with priors method is another effective uncertainty estimation technique [35, 39]. This approach is based on the ensemble-based uncertainty estimation method [28], but improves upon this uncertainty estimation technique in two ways. Firstly, the method creates a different prior function for each of the H ensemble members. These prior functions are randomly initialized, but frozen networks, meaning their output will be input-dependent, but their weights will not change during the training process. The final prediction of the network adds these prior functions to the prediction of their corresponding trainable ensemble members to create the final prediction of the network:

$$\hat{f}_i(x) = \hat{f}_l(x;\theta_i) + \beta \cdot \hat{f}_{p_i}(x) \tag{4.3}$$

In this equation, $\hat{f}_i(x)$ is the i-th ensemble prediction, $\hat{f}_l(x;\theta_i)$ is i-th learnable ensemble function, $f_{p_i}(x)$ is the i-th prior function, and β is a hyper-parameter that scales the importance of the prior function. The main idea behind these prior functions is that each ensemble member learns to ignore their prior in high data concentration areas while learning a different function from each other in the low data concentration areas, resulting in better epistemic uncertainty estimation capabilities for out-of-distribution (OOD) data [35]. A visualization of the resulting architecture can be found in Figure 4.1.

The second improvement upon the original ensemble-based uncertainty estimation method [28] is the addition of the bootstrap. This bootstrapping procedure ensures that each ensemble member is trained on a slightly different subset of the data. This procedure increases the diversity between the ensemble members near the edges of data concentrations , especially if it is combined with the previously added priors [35]. These two improvements significantly improve the epistemic uncertainty estimation capabilities of the multi-headed bootstrap with priors method compared to the original ensemble method [28, 35, 39].

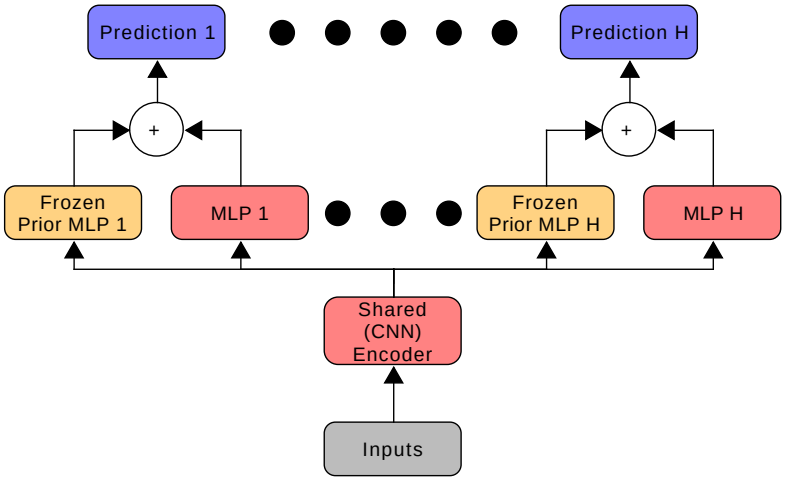


Figure 4.1: The multi-headed bootstrap ensemble with random priors network architecture. The architecture is similar to traditional ensembles but adds the output of a frozen prior network to each head's output to create a state depended prior. In this figure, the red boxes are the trainable parts of the architecture, and the yellow parts are the frozen untrainable parts.

The multi-headed bootstrap with priors method is still an ensemble based uncertainty estimation technique. Therefor, the methods estimates the uncertainty in its predictions as the standard deviation between the prediction of its ensemble members:

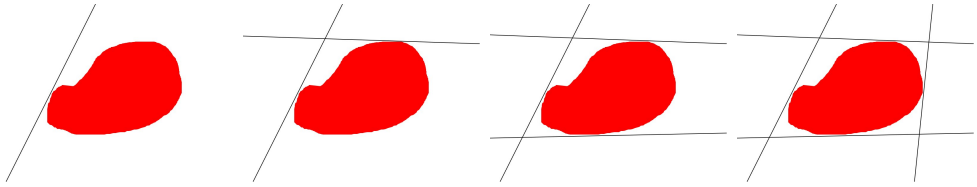
$$\overline{\mu}(x) = \frac{1}{H}\sum_{i=1}^{H} \hat{f}_i(x) \tag{4.4}$$

$$\overline{\sigma}(x) = \sqrt{\frac{\sum_{i=1}^{H}\left(\hat{f}_i(x) - \overline{\mu}(x)\right)^2}{H-1}} \tag{4.5}$$

In this equation, H is the number of ensemble members, and $\hat{f}_i(x)$ is the prediction of the ith ensemble member and its prior.

4.1.3. Orthonormal certificates

The orthonormal certificates method [43] is another epistemic uncertainty estimation technique. This approach's main advantage is that it requires only a single model and a single forward pass through the network to estimate the epistemic uncertainty of a prediction [43]. The method works by learning a collection of diverse non-constant functions called orthonormal certificates that map the feature of all in-of-distribution (IOD) data to zero and all the features of OOD data to non-zero values. Intuitively, the method can be thought of as constructing n binary classifiers, where each classifier adds a decision boundary between IOD data and a subset of the OOD data. By increasing the number of binary classifiers, we eventually end up with a decision boundary around the IOD data, allowing us to estimate the epistemic uncertainty as shown in Figure 4.2.



(a) 1 OCs (b) 2 OCs (c) 3 OCs (d) 4 OCs

Figure 4.2: The intuition behind Orthonormal Certificates (OCs) for epistemic uncertainty estimation. Each orthonormal certificate can be thought of as a binary classifier c_i that adds a decision boundary where samples for the in-distribution class (red) are mapped to zero and samples from the out-distribution class are mapped to one. By adding more and more decision boundaries, we can eventually construct an entire decision boundary around the in-distribution.

We can make this procedure more mathematically formal by considering a deep learning model as $\hat{y}(x) = f(\psi(x))$, where ψ is a deep feature extractor that extracts high-level features from the data, e.g., a convolutional neural network, and f is a shallow model that groups the high-level features into classes, e.g., a linear output layer. Next, we define the certificates $C = (C1, ..., C_k)$ where each C_i is a simple linear layer that maps the high-level features to a single output. These certificates are then trained to map the high-level features of the data set to zero by minimizing the mean squared error loss with an orthonormality constraint to ensure that every C_i learns a different non-constant function [43].

$$\underset{C}{\operatorname{argmin}} \frac{1}{n} \sum_{i=0}^{n} ||C^T \psi(x_i)||^2 + \lambda \cdot ||C^T C - I_k||, \text{ where } C \in R^{h \times k} \tag{4.6}$$

Using these orthonormal certificates, we can estimate epistemic uncertainty as the mean squared error:

$$u(x) = ||C^T \psi(x)||^2 \tag{4.7}$$

4.2. Epistemic uncertainty estimation experiment

In this section, we will test the epistemic uncertainty estimation capabilities of the previously introduced methods. As a reminder, epistemic uncertainty specifies the model's uncertainty in its prediction due to inadequate knowledge and training data. Formally this means that the epistemic uncertainty is approximately inversely proportional to the density $p(x)$ of the training data [7]:

$$\sigma_e(x) \propto p^{-1}(x) \tag{4.8}$$

To verify that these methods measure this epistemic uncertainty and not the aleatoric uncertainty caused by the noise in the data, we will use a data set consisting of two clusters (Figure 4.3). The left cluster has no aleatoric uncertainty in its data, while the right cluster has significantly more aleatoric uncertainty in its data. Ideally, we want our epistemic uncertainty estimation to have the following three key properties. Firstly, the epistemic uncertainty estimation must be low in areas with high data concentrations, such as inside the two clusters. Secondly, the method should measure only epistemic uncertainty and no aleatoric uncertainty. Therefore, the amount of epistemic uncertainty measured should be approximately equal in both clusters, independent of the fact that the right cluster has

significantly more noise in its data. Finally, the epistemic uncertainty estimation should be significantly higher in areas with low or no data concentrations, such as between the two clusters and near the plot's edges.

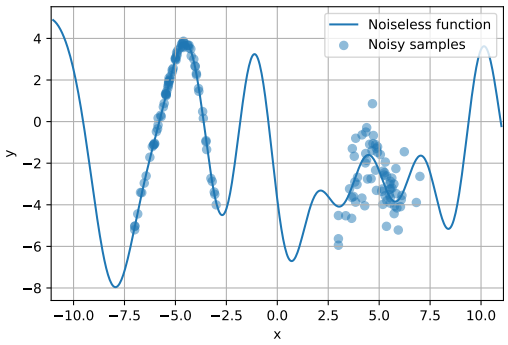
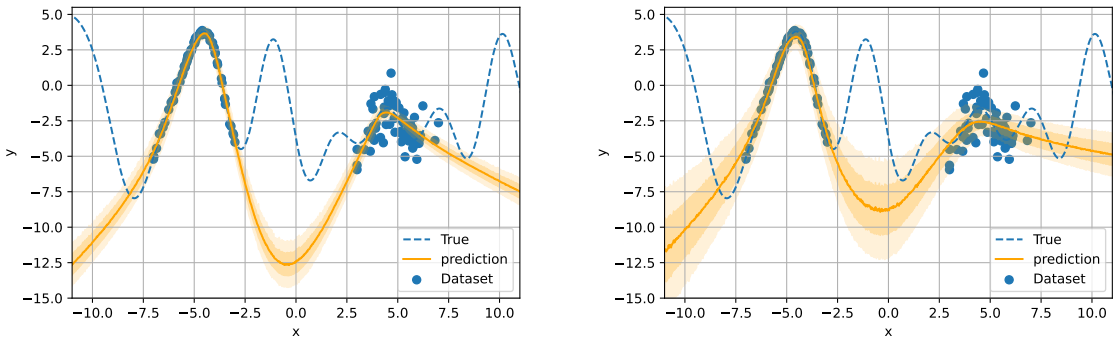


Figure 4.3: A visualization of the data set used in the epistemic uncertainty estimation capabilities experiment.

MC dropout results The experimental results for the Monte Carlo (MC) dropout method [14] are shown in Figure 4.4. Based on this figure, the method has three noteworthy properties. Firstly, the method is able to detect the gap between the two clusters. However, it is only able to do this with reasonably high dropout rates. Secondly, the uncertainty does not reduce towards zero in areas with high data concentrations. Finally, uncertainty estimation in the right cluster is relatively higher than the uncertainty estimation for the left cluster. This result indicates that the method measures both epistemic and aleatoric uncertainty. This is an undesirable property since we want a method that only measures epistemic uncertainty.



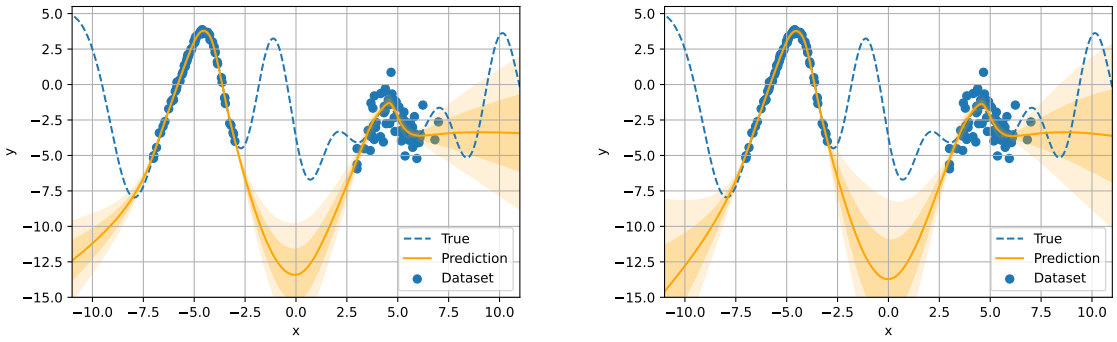
(a) Dropout rate 0.01

(b) Dropout rate 0.1

Figure 4.4: A visualization of the uncertainty estimated by Monte Carlo dropout. The dropout rate has a significant effect on the uncertainty estimation capability of Monte Carlo dropout.

Ensemble with random priors results The results for the multi-headed bootstrap with priors method [35, 39] are shown in Figure 4.5. This figure has three key things to note. Firstly, the uncertainty is almost zero in areas with high data concentrations even if the data is noisy, while it is significantly higher in areas with no data. This observation indicates that the method mainly focuses on epistemic uncertainty. Secondly, the bootstrapping procedure causes the uncertainty near the edges to depend on the amount of noise in the data. This is desirable because the noiseless edge data on the left can probably be trusted more than the noisy edge data on the right. Note, this property is only desirable near the edges of the data set because if it happens inside data clusters, it means that the method measures aleatoric uncertainty. Finally, the figure also shows the effect of the prior importance weight hyper-parameter β. If this parameter β increases, the learned function has to overcome a stronger prior,

which results in more diverse functions and thus better epistemic uncertainty estimation capabilities. However, this diversity comes at the expense of the expressiveness of $\hat{f}_i(x)$ because there are fewer functions for which $\hat{f}_l(x;\theta_i) + \beta \cdot \hat{f}_{p_i}(x)$ is as close as possible to the target variable y.

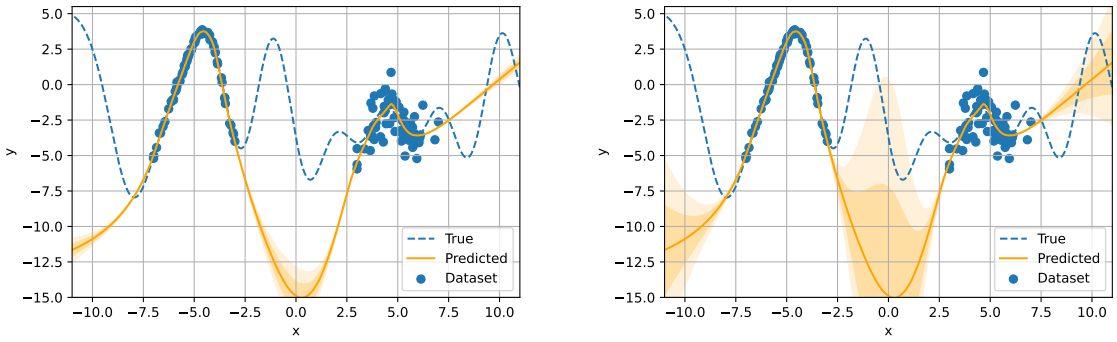


(a) Prior importance weight $\beta = 3$

(b) Prior importance weight $\beta = 10$

Figure 4.5: A visualization of the uncertainty estimated by the multi-headed bootstrap with priors method. The prior weight scale β rate has a significant effect on the uncertainty estimation capability of the method.

Orthonormal certificates results The results for the orthonormal certificates method [35, 39] are shown in Figure 4.5. In this figure, there are two key things to note. Firstly, the uncertainty is almost zero in areas with high data concentrations even if the data is noisy, while it is significantly higher in areas with no data. This observation indicates that the method mainly focuses on epistemic uncertainty estimation. Secondly, a difficult property of the uncertainty measurement is that it has a different scale than the predicted unit. This property makes it impossible to calculate a meaningful lower bound, such as an n-standard deviation lower bound. However, this property does not mean that the method does not work, as shown in Figure 4.5. It only means that we need to manually tune the scale, which can be difficult in higher dimensions.



(a) Uncertainty multiplied by 10

(b) Uncertainty multiplied by 75

Figure 4.6: A visualization of the uncertainty estimated by the orthonormal certificates method. An undesirable property of this method is that the uncertainty measurement is on an unrelated scale, meaning that we have to manually scale the measurement to make it visible.

4.3. Implementation requirements

Deep uncertainty estimation is a widely studied topic in the supervised learning domain, which has resulted in a wide range of uncertainty estimation techniques. However, integrating existing methods into deep RL algorithms is not straightforward because deep RL algorithms have specific properties that make them fundamentally different from their supervised learning counterparts. For example, one

of the most important differences between the two approaches is that offline deep RL methods need their uncertainty estimate during the learning process. In contrast, supervised learning methods only need it at inference time. Uncertainty-aware offline deep RL methods have this property because they need an estimate of the current uncertainty to calculate their pessimistic temporal difference target in the bootstrapping procedure that is used to learn the value function (Equation 3.7). This fundamental difference imposes certain computational requirements on the uncertainty estimation technique and shows the need for online learning-based uncertainty estimation.

Another important difference between deep reinforcement and supervised learning is that there tends to be a large amount of variance in the learning signals for deep RL, and this variance can cause major issues for learning stability and speed [48]. For example, in deep RL, the only ground truth learning signal comes from the rewards and state transitions. All other information is obtained by bootstrapping the agent's own value function. Due to this bootstrapping procedure, the temporal difference target will keep changing over time during the learning process, introducing large amounts of variance in the learning signal and correlated learning targets. Furthermore, the natural stochasticity of the environment and the sampling strategy can also add additional variance in the learning signal. Most of the recent improvements in deep RL have been focused on reducing this variance in the learning signal [11, 16, 33, 47]. Therefore, the used uncertainty estimation technique must introduce a minimal amount of additional variance into the learning signal to keep the fragile balance in deep RL algorithms stable.

Based on these key differences between offline deep RL-based uncertainty estimation and supervised learning-based uncertainty estimation, we identified the following key implementation related requirements:

1. **End-to-end epistemic uncertainty:** When an empirical Bellman update is applied to a neural network, the change in value can impact any state due to the generalization capabilities of the network [9, 35]. Therefore, the epistemic uncertainty must be estimated in an end-to-end manner such that the estimate is aware of the internal generalization of the model.

2. **Dynamic uncertainty estimation:** The epistemic uncertainty-based penalty must depend on the current policy and value target estimate, which keep changing during the learning process [9]. Therefore, the uncertainty estimation method must be able to capture these changes, allowing it to measure the current knowledge uncertainty with respect to both the current policy and value function in an online learning-based manner.

3. **Minimal computational cost:** The epistemic uncertainty-based penalty must be calculated for every value function loss calculation. Therefore, calculating the epistemic uncertainty should have minimal computational costs to keep the learning speed of the algorithm manageable.

4. **Same scale uncertainty measurement unit:** The epistemic uncertainty-based penalty will be used to calculate a lower-bound estimate of the temporal difference target by subtracting the penalty from the value estimate. Therefore, it is desirable if the uncertainty estimation is in the same units as the value estimate.

5. **Works with moving learning targets:** The temporal difference targets used to learn the value function keep changing over type, due to the bootstrapping procedure used to obtain these targets. Therefore, it is essential that the uncertainty estimation technique works with these moving targets and does not assume any stationary in the temporal difference targets [48].

6. **Minimal impact on the learning stability:** Deep RL algorithms tend to have a high amount of variance in their learning signals. These algorithms can become unstable and will fail to learn when their learning signal variance increases even further [48]. Therefore, the used uncertainty estimation technique must introduce a minimal amount of additional variance into the learning signal.

In Table 4.1, we created an overview of how each uncertainty estimation technique measures our implementation requirements. This table shows that the Monte Carlo dropout uncertainty estimation technique [14] meets all but one of these implementation requirements. The main advantage of this technique is that it estimates the uncertainty as the standard deviation between N Monte Carlo based

Requirement	Monte Carlo dropout [14]	Multi-headed bootstrap ensemble with random priors [35]	Orthonormal certificates [43]
End-to-end epistemic uncertainty	✓	✓	✓
Dynamic uncertainty estimation	✓	✓	■
Minimal computational cost	✓	■	✓
Same scale uncertainty measurement unit	✓	✓	✗
Works with moving learning targets	✓	✓	■
Minimal impact on the learning stability	✗	✓	✓

Table 4.1: An overview of how well each uncertainty estimation technique matches the implementation requirements. In this table, the ✓ symbol means that the method matches the requirement, the ✗ symbol means that the method does **not** matches the requirement, and the ■ symbol means that it is debatable if the method matches the requirement.

forward passes through the network, which means that the method fulfills requirements 1, 2, and 4. In addition, these Monte Carlo forward passes through the network can be performed in parallel on a GPU, which means that the method also fulfills requirement 3. However, the main disadvantage of the Monte Carlo dropout method is that the dropout used to obtain the Monte Carlo samples increases the amount of variance in the learning signal. This additional variance in the learning signal can prevent the deep RL method from learning and might even lead to divergent behavior [48]. The only way to reduce the amount of additional variance is by reducing the dropout rate. However, in the previous section, we showed that reducing the dropout rate also reduces the uncertainty estimation capabilities of this method. Therefore, it is challenging to fulfill requirement 6 while also ensuring that the uncertainty estimation capabilities of this method remain strong enough for our purposes.

In contrast, the Multi-headed bootstrap ensemble with random priors method [48] meets all requirements, except for requirement 2, which will only be met in certain situations. The main advantage of this technique is that it estimates the uncertainty as the standard deviation between the predictions of the different ensemble members. Due to this property, its uncertainty estimate is aware of the internal generalization of the model, changes in tandem with the current value function during the learning process, and is on the same scale as the prediction from the value function. Therefore, the method meets requirements 1, 2, and 4. Furthermore, the only change in the training process is that we need to train multiple models instead of one. Thus, this uncertainty estimation method will not change the properties of a deep RL method when they are integrated together, which means that the method meets requirements 5 and 6. Besides all these benefits, the method has one big disadvantage: the number of weights and compute of this method increases linearly with the number of heads used. However, it is possible to share a feature extractor such as a convolutional neural network. In theory, this shared encoder can reduce the diversity in the ensemble, but it has been observed empirically that this effect is minimal, making this a valid computational trade-off [34, 35, 39]. In practice, the parallel heads and their priors contain only one or two fully connected layers making the memory and performance requirements manageable. Therefore, this method only meets requirements 3 when this trade-off is sufficient. However, the method will still require more computational resources compared to the other single model uncertainty methods.

Finally, Table 4.1 shows that the orthonormal certificates method [43] meets some of the requirements, except for requirements 2, 4, and 5. The main advantage of the orthonormal certificates method is that it requires only a single model and a single forward pass through the network to estimate the epistemic uncertainty. Due to this property, this method is the most efficient method with respect to requirement 3. Another advantage of this method is that it uses the internal representations of the value function's network in its uncertainty estimate, which means that it is aware of the internal generalization of the model and thus meets requirement 1. However, a major disadvantage of the method is that it is not well suited for online uncertainty estimation due to original design assumptions. In the original design of the method, it was assumed that the orthonormal certificates would be learned when the prediction model was done training [43]. During initial experiments, it appears that this assumption causes the method to struggle with the moving targets and moving features in the offline deep RL setting since it assumes some level of stationarity in these features. Therefore, it is unclear to which extent the method meets requirements 2 and 5. The final disadvantage of this method is that the uncertainty measure is not proportional to the predicted values, which means that the method cannot fulfill requirement 4.

4.4. Conclusion

This chapter explored different uncertainty estimation techniques for their suitability in an uncertainty-aware offline deep RL algorithm. To this end, we explored their epistemic uncertainty estimation capabilities and evaluated them against multiple implementation requirements. Based on these results, we found the multi-headed bootstrap ensemble with random priors method [35] the most suitable epistemic uncertainty estimation for an uncertainty-aware offline deep RL algorithm. This method has several advantages that make it suitable for offline deep RL. For example, the prior functions and the bootstrapped data sets ensure that the method mainly focuses on epistemic uncertainty, covering the theoretical requirements. Furthermore, the method also meets all implementation requirements, making integrating this uncertainty estimation technique with existing deep RL methods significantly easier. Therefore, we will use this uncertainty estimation technique in the next chapter to design a model-free uncertainty-aware offline deep RL algorithm.

5

Methods

In the previous chapter, we evaluated multiple deep uncertainty estimation techniques. Based on these results, we found the multi-headed bootstrap ensemble with random priors method [35] the most suitable method to be used in an uncertainty-aware offline deep reinforcement learning (RL) algorithm. In this chapter, we propose two new offline deep RL algorithms, named PEssimistic ensemBLe (PEBL) that use the multi-headed bootstrap with priors architecture [35] to approximate a pessimistic version of the Q-function. The first algorithm is a pessimistic version of the Double Deep Q-Network (DDQN) algorithm [47], which is aimed at problems with discrete action spaces. The second algorithm is a pessimistic version of the Soft Actor Critic (SAC) algorithm [16], which works in both continuous and discrete action-space cases. Although SAC can also be applied to discrete action spaces, DDQN is often preferred in this setting due to its lower memory and computation costs. Therefore, we will only derive the continuous action space version of PEBL SAC.

5.1. DDQN version

Our pessimistic version of DDQN, which we call PEBL DDQN, acts and learns similarly to the original DDQN algorithm. It still prevents over-estimations cause by the max operator in the temporal difference target by selecting the action that maximizes the current network instead of the target network while using the target network to estimate the Q-value. However, the key difference between the original algorithm and our pessimistic version is that in the pessimistic algorithm, we use a pessimistic Q-function which we define as the one standard deviation lower bound over the prediction of each of the ensemble members:

$$Q_p(s, a; \theta) = \overline{\mu}_Q(s_t, a; \theta) - \overline{\sigma}_Q(s_t, a; \theta) \tag{5.1}$$

where $\overline{\mu}_Q(s_t, a; \theta)$ is the sample mean over the different heads, and $\overline{\sigma}_Q(s_t, a; \theta)$ is the sample standard deviation over the different heads, which we calculate as follows:

$$\overline{\mu}_Q(s_t, a; \theta) = \frac{1}{h} \sum_{i=0}^{h-1} Q_i(s_t, a; \theta) \tag{5.2}$$

$$\overline{\sigma}_Q(s_t, a; \theta) = \sqrt{\frac{\sum_{i=0}^{h-1} (Q_i(s_t, a; \theta) - \overline{\mu}_Q(s_t, a; \theta))^2}{h - 1}} \tag{5.3}$$

In these equations, Q_i is the Q-value prediction of the ith ensemble head. This change has some impacts on the other learning formulas. For example, this change also changes the TD-target of DDQN in Equation 2.13 to:

$$y_t = r_t + (1 - d_t) * \gamma \left(\overline{\mu}_Q(s_{t+1}, a^*_{t+1}; \theta') - \overline{\sigma}_Q(s_{t+1}, a^*_{t+1}; \theta') \right) \tag{5.4}$$

$$a^*_{t+1} = \underset{a}{\operatorname{argmax}} \{ \overline{\mu}_Q(s_{t+1}, a; \theta) - \overline{\sigma}_Q(s_{t+1}, a; \theta) \}$$

In this equation, $(s_t, a_t, r_t, d_t, s_{t+1})$ is a state transition sampled from the data set $\mathcal{D}$ where d_t is a Boolean flag that indicates where s_{t+1} is a terminal state, γ is the discount factor of the Markov

decision process (MDP), θ' are the parameters of the target network, and θ is the parameter of the online network. One thing to note here is that the TD-target in this equation differs from the original definition from the bootstrapped Deep Q-Network (DQN) [34, 35] algorithm, which trained each head on its own target head. We differ in aspect because we need to subtract an uncertainty penalty; if the penalty was subtracted from each head directly, an unstable feedback loop is created, resulting in potentially very large negative Q-values. We this knowledge in mind, we can derive the loss function per head by adding the bootstrapping masks as described in the efficient implementation of bootstrapped DQN [39], giving the following definition for the TD-loss per head i:

$$\mathcal{L}_{Q_i} = m_{t,i} \cdot \mathcal{L}\left(Q_i(s_t, a_t; \theta), y_t\right) \tag{5.5}$$

In this equation, $\mathcal{L}$ is the Huber loss [20], and $m_{t,i}$ is a Boolean mask that has been sampled for each training sample t and head i from a Bernoulli distribution with $p = 0.8$ [39]. Note that the m remains constant throughout the entire training process. The final change we make to the algorithm is how its select actions. Our algorithm changes this to:

$$\pi_t = \underset{a}{\text{argmax}}\{\overline{\mu}_Q(s_t, a; \theta) - \overline{\sigma}_Q(s_t, a; \theta)\} \tag{5.6}$$

To get a better understanding of how these changes impact the algorithm, we created an overview in Algorithm 1 where differences from the DDQN algorithm [47] are in red. This pseudo-code summarizes the proposed training process, which is very similar to the original DDQN algorithm, except for the addition of the H heads and the bootstrapping masks m. Furthermore, it is important to note that we do **not** propagate gradients through y_t just like the original algorithm.

Algorithm 1: Pseudo code for PEBL DDQN, differences from DDQN [47] are in red.

Input: Data set $\mathcal{D}$, discount γ, number of gradient step N, target network sync rate K, bootstrap probability p, number of heads H

For every data point in $\mathcal{D}$ add $m \in \mathbb{R}^H$, where $m_i \sim Ber(p)$;

Initialize Q-network θ and target Q-network θ' both with the random bootstrap architecture and H heads per action;

for $i = 0$ *to* N **do**

  Sample $(s_t, a_t, d_t, s_{t+1}, m) \sim \mathcal{D}$;

  Calculate the TD-target without gradients using:
  $y_t \leftarrow r_t + (1 - d_t) * \gamma \left(\overline{\mu}_Q(s_{t+1}, a^*_{t+1}; \theta') - \cdot\overline{\sigma}_Q(s_{t+1}, a^*_{t+1}; \theta')\right)$, where
  $a^*_{t+1} \leftarrow \text{argmax}_a\{\overline{\mu}_Q(s_{t+1}, a; \theta) - \cdot\overline{\sigma}_Q(s_{t+1}, a; \theta)\}$;

  Update the Q-function with parameters θ by one ADAM [26] step using:
  $\nabla_\theta \mathcal{L}_{Q_\theta} \leftarrow \frac{1}{H} \sum_{h=0}^{H-1} m_h \cdot \text{Huberloss}\left(Q(s_t, a_t; \theta_h), y_t\right)$;

  Update the target network every K updates using: $\theta' \leftarrow \theta$;

end

$\pi = \text{argmax}_a\{\overline{\mu}_Q(s_t, a; \theta) - \overline{\sigma}_Q(s_t, a; \theta)\}$;

return π

5.2. SAC version

Our pessimistic version of SAC, which we call PEBL SAC, acts and learns similarly to the original SAC algorithm. This pessimistic version of the algorithm still aims to learn a policy that acts as randomly as possible if multiple actions have the same expected return. It also still aims to reduce the over-estimations caused by the Bellman equation by estimating the expected return by taking the lowest Q-value of two independent predictions. However, the key difference between the original algorithm and our pessimistic version is that in the pessimistic algorithm, we estimate the expected return using a pessimistic Q-function which we define as the one standard deviation lower bound over the prediction of each of the ensemble members per independent Q-function:

$$Q_p(s, a; \theta) = \overline{\mu}_Q(s_t, a; \theta) - \overline{\sigma}_Q(s_t, a; \theta) \tag{5.7}$$

where $\overline{\mu}_Q(s_t, a; \theta)$ is the sample mean over the different heads, and $\overline{\sigma}_Q(s_t, a; \theta)$ is the sample standard deviation over the different heads, which we calculate as follows:

$$\overline{\mu}_Q(s, a; \theta_1, \theta_2) = \frac{1}{h} \sum_{i=0}^{h-1} \min_{j=1,2} Q(s, a; \theta_{j,i}) \tag{5.8}$$

$$\overline{\sigma}_Q(s, a; \theta_1, \theta_2) = \sqrt{\frac{1}{h-1} \sum_{i=0}^{h-1} (\overline{\mu}_Q(s, a; \theta_1, \theta_2) - \min_{j=1,2} Q(s, a; \theta_{j,i}))^2} \tag{5.9}$$

In the equations, we use the minimum of the two Q-values as our prediction, just like the original SAC method. The original SAC method took this minimum because prior work has shown that this prevents over-estimations [16, 47]. In our case this minimization might not be necessary due to our pessimistic approach. However, we kept this minimization in the algorithm to ensure that our method stays as close to the original SAC algorithm as possible. Therefore, $Q_{j,i}$ is the Q-value prediction of the ith ensemble head in the jth network. This change has some impacts on the other learning formulas. For example, this change also changes the TD-target in Equation 2.14 to:

$$y_t = r_t + (1 - d_t) \cdot \gamma \cdot \left(\overline{\mu}_Q(s, \tilde{a}_{t+1}; \theta'_1, \theta'_2) - \overline{\sigma}_Q(s_{t+1}, \tilde{a}_{t+1}; \theta'_1, \theta'_2) - \alpha \log \pi(\tilde{a}_{t+1} | s_{t+1}; \phi) \right) \tag{5.10}$$

$$\tilde{a}_{t+1} \sim \pi(\cdot | s_{t+1}; \phi)$$

In this equation, $(s_t, a_t, r_t, d_t, s_{t+1})$ is a state transition sampled from the data set $\mathcal{D}$ where d_t is a Boolean flag that indicates where s_{t+1} is the terminal state, γ is the discount factor of the MDP, $\tilde{a}_{t+1}$ is sampled stochastically from the learned policy, and α is the entropy trade-off parameter. Using the TD-targets from Equation 5.10, we can calculate the loss per head j for each of the Q-network $i \in \{1, 2\}$ with mask m_j:

$$\mathcal{L}_{Q_{i,j}} = m_j \cdot \mathcal{L}\left(Q_j(s_t, a; \theta_{i,j}), y_t\right) \tag{5.11}$$

In this equation, $\mathcal{L}$ is the Huber loss [20], just like the original SAC algorithm.

The original SAC algorithm calculates the policy loss per data point using the formula:

$$\mathcal{L}_\pi = \alpha \log \pi(\tilde{a}_t, s_t; \phi) - \min_{i=1,2} \{Q(s_t, \tilde{a}_t; \theta_i)\}, \text{ where } \tilde{a}_t \sim \pi(a | s_t; \phi) \tag{5.12}$$

where $\tilde{a}_t$ is sampled using the reparameterization trick, which is differentiable with respect to ϕ. Our pessimistic version of this algorithm changes this loss function to:

$$\mathcal{L}_\pi = \alpha \log \pi(\tilde{a}_t, s_t; \phi) - \left(\overline{\mu}_Q(s, a; \theta_1, \theta_2) - C_\pi \cdot \overline{\sigma}_Q(s, a; \theta_1, \theta_2) \right) \tag{5.13}$$

In this equation, we also introduce the uncertainty weight trade-off parameter C_π. This parameter controls the trade-off between minimizing uncertainty and maximizing the Q-values in the learned policy. This trade-off parameter is needed because the policy loss in SAC is similar to a white box adversarial attack on the Q-function due to the reparameterization trick [3]. This adversarial formulation of the policy loss is not a problem in deep online RL because it forces the agent to learn about the flaws in its Q-function, which helps with exploration. However, in offline RL, this adversarial formulation is a problem because the agent can no longer collect counterexamples in the environment. Therefore,

selecting the right parameter for C_π is crucial. Empirically, we found that it is difficult to find the right value for C_π because it depends on many factors, such as the size of the data set, the state-action space coverage, the difficulty of modeling the true MDP, etc. However, we also discovered it is possible to learn the right value for this parameter using dual gradient descent, a technique that is increasingly common in deep reinforcement learning [16, 27]. We apply dual gradient descent using the technique described in section 2.2.1 and transform C_π into a Lagrangian multiplier by adding the constraint that the average uncertainty of the actions selected by the learned pessimistic policy π should be equal to the average uncertainty observed in the actions for the data set:

$$\frac{1}{n}\sum_{s_t,a_t}^{\mathcal{D}} \overline{\sigma}_Q(s_t, a_t; \theta) = \frac{1}{n}\sum_{s_t}^{\mathcal{D}} \overline{\sigma}_Q(s_t, a_p; \theta), a_p \in \pi_p \tag{5.14}$$

This constraint is possible because we are only interested in avoiding epistemic uncertainty, which is high in areas where the network cannot model the function well. Thus, we allow the selection of out-of-distribution actions as long as we avoid areas with above-average epistemic uncertainty. The constraint in Equation 5.14 captures this property, which results in a higher value of C_π if the π_p chooses actions with above-average epistemic uncertainty. It results in a lower value for C_π if π_p tends to choose actions with below-average epistemic uncertainty.

With all the derivation from SAC complete, we can summarize our training process using Algorithm 2. This training process is very similar to the original SAC algorithm, except for the addition of the H heads, the bootstrapping masks m, and the automatic tuning of the trade-off parameter C_π. Note that we do **not** propagate gradient through y_t just like the original algorithm.

Algorithm 2: Pseudo code for PEBL SAC, differences from SAC [16] are in red.

Input: Data set $\mathcal{D}$, discount γ, number of gradient step N, entropy target α_{target}, bootstrap probability p, number of heads H

For every data point in $\mathcal{D}$ add $m \in \mathbb{R}^H$, where $m_i \sim Ber(p)$;

Initialize Q-network one and two and their target networks with θ_1, θ_2, θ_1' and θ_2' all with the random bootstrap architecture and H heads;

Initialize trade-off parameters α and C_π ;

for *0 to N* **do**

  Sample $(s_t, a_t, d_t, s_{t+1}, m) \sim \mathcal{D}$;

  Calculate TD-target without gradients using:
  $y_t = r_t + (1 - d_t) \cdot \gamma \cdot \left(\overline{\mu}_Q(s, \tilde{a}_{t+1}; \theta_1, \theta_2) - C_y \cdot \overline{\sigma}_Q(s_{t+1}, \tilde{a}_{t+1}; \theta_1, \theta_2) - \alpha \log \pi(\tilde{a}_{t+1}|s_{t+1}; \phi)\right)$,
  $\tilde{a}_{t+1} \sim \pi(\cdot|s_{t+1}; \phi)$;

  Update the ith Q-function with parameters θ_i by one ADAM [26] step using:
  $\nabla_{\theta_i}\mathcal{L}_{Q_{\theta_i}} \leftarrow \frac{1}{H}\sum_{h=0}^{H-1} m_h \cdot \text{Huberloss}\left(Q(s_t, a_t; \theta_{i,h}), y_t\right)$ for $i = 1, 2$

  Update the policy π_ϕ by one ADAM step using:
  $\nabla_\phi \mathcal{L}_\pi = \alpha \log \pi_\phi(\tilde{a}_t, s_t) - (\overline{\mu}_Q(s, \tilde{a}_t; \theta_1, \theta_2) - C_\pi \cdot \overline{\sigma}_Q(s, \tilde{a}_t; \theta_1, \theta_2))$, where $\tilde{a}_t \sim \pi(\cdot|s_t; \phi)$ is sampled using the reparameterization trick making it differential w.r.t. ϕ;

  Update the entropy trade-off parameter α by one ADAM step using:
  $\nabla_\alpha \mathcal{L}_\alpha \leftarrow \log(\alpha)\left(\log \pi_\phi(\tilde{a}_t, s_t) - \alpha_{target}\right)$;

  Update the uncertainty trade-off parameter C_π by one ADAM step using:
  $\nabla_{C_\pi}\mathcal{L}_{C_\pi} \leftarrow -C_\pi\left(\overline{\sigma}_Q(s, \tilde{a}_t; \theta_1, \theta_2) - \overline{\sigma}_Q(s, a_t; \theta_1, \theta_2)\right)$;

  Update target network using Polyak averaging: $\theta_i' \leftarrow \rho\theta_i' + (1 - \rho)\theta_i$ for $i = 1, 2$;

end

return π_ϕ

6

Related offline deep RL methods

The previous chapters showed that missing information in the data set and poorly selected optimization objectives result in highly overestimated value functions in offline reinforcement learning (RL). These overestimated value functions are problematic because they result in unrealistic policies with poor regret bounds that seem rewarding based on the overestimated value function but are not in the real environment. Only recently, it has become clear that the pessimism principle is the information-theoretical optimal approach for the offline RL problem [9, 23, 38]. Although many prior works did not have access to this proof, they were already aware of the overestimation and unrealistic policies problem and tried to combat these issues in different ways. This chapter will discuss these prior methods and how they attempted to combat these issues, and how it differs from our approach. To this end, we have divided these prior methods into three categories. The first category of methods is the explicit policy-constrained methods. The methods in this category apply explicit constraints on the learned policy to prevent overestimation issues in offline RL. The second category is the pessimistic value function category. The methods in this category appear very close to our desired goal of a model-free uncertainty-aware offline RL algorithm. Still, they are, in reality, also policy-constrained methods with implicit policy constraints because they still rely on the data collection strategy [9]. Finally, we conclude the chapter by discussing model-based uncertainty-aware algorithms. The methods in this category are the closest to our goal of creating a model-free uncertainty-aware offline RL algorithm. However, as the name suggests, they are model-based approaches instead of model-free approaches.

6.1. Explicit policy-constrained methods

Many prior methods claimed that out-of-distribution (OOD) actions and distributional shifts caused the commonly observed overestimations in the value function. To combat this issue, many methods proposed various constraints that should prevent the learned policy from selecting these OOD actions. In hindsight, this approach is similar to the policy-constrained algorithmic family we discussed in chapter 3. The most obvious example of a deep policy-constrained method is Batch-Constrained Q-learning (BCQ) [11, 12]. This algorithm aims to avoid overestimated OOD Q-function evaluations by detecting OOD actions based on the likelihood of state-action pairs appearing in the empirical policy. Using this technique, the BCQ algorithm can ignore these OOD actions while using policy optimization techniques such as Q-learning [12] or actor-critic [11] on the remaining in-of-distribution (IOD) state-action pairs. Formally, this means that the learned constrained value function, denoted by $Q_c(s, a; \theta)$, and the constrained policy, denoted by $\pi_c(s)$, are defined as:

$$Q_c(s, a; \theta) = \begin{cases} Q(s, a; \theta), & \text{if } \frac{\pi_\beta(a|s;\phi_\beta)}{\max_{\hat{a}} \pi_\beta(\hat{a}|s;\phi_\beta)} > \tau \\ -\infty, & \text{otherwise} \end{cases} \tag{6.1}$$

$$\pi_c(s) = \underset{a}{\operatorname{argmax}}\, Q_c(s, a; \theta) \tag{6.2}$$

In these equations, $\pi_\beta(a|s; \phi_\beta)$ is the learned empirical policy, and $\tau \in [0, 1)$ is the threshold hyper-parameter that specifies how much the learned policy is allowed to deviate from the empirical policy.

The main item of interest in this equation is the policy constraint, which is formulated in the first case of Equation 6.1. This constrain normalizes the action distribution based on the relative likelihood of selecting an action given the empirical policy. If this relative likelihood is higher than the threshold hyperparameter, we consider the state-action pair as an IOD sample. It is more difficult to calculate this constraint in continuous action spaces due to the max operator in the denominator. Therefore, the denominator is typically approximated using the maximum of multiple samples from a variational autoencoder (VAE) [11]. The main difference between the BCQ algorithm and our algorithm is that the performance of the BCQ algorithm depends on the quality of the data collection strategy due to its dependence on the empirical policy, while our method only depends on the amount of information in the data set.

Another example of an explicit policy-constrained method in the literature is the Bootstrapping Error Accumulation Reduction (BEAR) algorithm [25]. This algorithm is interesting because it also aims to reduce the overestimation in the TD-targets using a small ensemble, just like our algorithm. The BEAR algorithm aims to reduce this overestimation in two ways. Firstly, this method constrains the policy used to estimate the TD-target to ensure that the action selected for the next state would be relatively close to the empirical policy of the data set. It does this by sampling n actions from the learn policy within a predefined maximum mean discrepancy of the empirical policy [25]. Secondly, it estimates the TD-target as the weighted average between a small ensemble of Q-functions. This weight ensemble of the TD-target is supposed to reduce the overestimations caused by the max operator in the Bellman equation [25]. Due to this ensemble of Q-functions, the method seems similar to our algorithm. However, the BEAR algorithm differs from our algorithm due to its usage of the maximum mean discrepancy constrain. Due to this constraint, the BEAR algorithm depends on the empirical policy, making it a policy-constrained method. In contrast, our method uses the ensemble to estimate the uncertainty in the TD-target, which removes this dependency on the empirical policy.

Besides these two examples, there are many more examples of policy-constrained methods, each with a slightly different policy divergent measure that will be used to detect OOD state-action pairs. For example, other common divergent measures are the maximum mean discrepancy, Kullback–Leibler (KL) divergence, Wasserstein Distance, and VAE based metrics [19, 25, 50]. On paper, all these methods seem different. However, in the end, they are all part of the policy-constrained algorithmic family, which means that their performance always depends on the quality of the used data collection strategy [9]. Our method differs in this aspect since it does not depend on such a constraint. Instead, our method aims to avoid actions whose epistemic uncertainty is relatively higher. Due to this property, our method only depends on the quality of the epistemic uncertainty estimate and the amount of information in the data set. Therefore, our method should perform significantly higher than these policy-constrained methods if the data set has been collected with a non-optimal data collection strategy. In the next chapter, we will verify this by comparing the performance of our algorithm against the performance of the BCQ method when the level of optimality in the data collection strategy varies.

6.2. Pessimistic value function methods

Another interesting approach in the literature to avoid value function overestimation is the Conservative Q-Learning (CQL) approach [27]. This approach aims to avoid these overestimations by adding a regularization objective to the value function optimization objective that minimizes all possible Q-values while simultaneously maximizing the Q-values in the data set. The idea behind this algorithm is that the additional regularization objective will help us to learn a Q-function who on average will lower bounds the true Q-function [27]. Formally, this optimization objective can be formalized as:

$$\min_Q \overbrace{\alpha\, \mathbb{E}_{s\sim\mathcal{D}}\left[\log \sum_a \exp(Q(s_t,a;\theta)) - \mathbb{E}_{a\sim\pi_\beta(a|s_t)}\left[Q(s_t,a;\theta)\right]\right]}^{\text{(A)}} + \underbrace{\frac{1}{2}\,\mathbb{E}_{s_t,a_t,r_t,s_{t+1}\sim\mathcal{D}}\left[Q(s_t,a_t;\theta) - y_t(r_t,s_{t+1};\theta')\right]}_{\text{(B)}} \tag{6.3}$$

In this equation, term (A) is the regularization objective, and term (B) is the normal mean squared Bellman error (MSBE) optimization objective commonly used in deep RL to learn the value function. Furthermore, α is a trade-off hyperparameter that controls the trade-off between optimizing the regularization objective and the MSBE objective, π_β is the empirical policy in the data set, and y_t is the TD-target obtained using a target value network. While it appears that this algorithm aims to avoid low-information regions similar to those in the uncertainty-aware pessimistic algorithmic family, this is not the case because it uses data concentrations as a proxy for information. Due to this property, the algorithm still depends on the empirical policy and data collection policy, meaning it still belongs to the policy-constrained pessimistic algorithmic family [9]. Interestingly, even though this algorithm officially belongs to the policy-constrained pessimistic algorithmic family, the constraint it uses is extremely flexible, which has allowed this method to achieve state-of-the-art results in almost all offline deep RL benchmarks [27].

At the time of writing, the only method that performs slightly better on these benchmarks is the Conservative Offline Model-Based policy Optimization (COMBO) algorithm [53]. This algorithm is a model-based extension of the CQL algorithm, which uses its learned reward and transition functions to generate additional training samples. These additional synthetic training samples help improve the final performance of the algorithm because the regularization term in the original CQL algorithm tends to reduce the Q-values for all state-action pairs that are not in the data set, even if these state-action pairs should be considered IOD. These synthetic training samples solve this problem because they fill these gaps, ensuring that these IOD areas will not be unnecessarily regularized [53]. Although these synthetic training samples solve an important problem in the conservative Q-learning approach, it does not remove the dependence on the data collection strategy. In contrast, our method does not have this dependency, which means that our method should theoretically be able to deviate and improve even further from the empirical policy than these conservative Q-learning-based algorithms. Therefore, we will use the CQL and COMBO algorithms in the next chapter to see how well our algorithm compares to their state-of-the-art results.

6.3. Model-based uncertainty-aware methods

The two previous sections discussed different model-free approaches to offline deep RL and how they aimed to avoid overestimations in the value function. Model-based RL is an alternative set of approaches that aim to solve the offline RL problem by approximating the reward and transition function of the underlying Markov decision process (MDP) of the data set. The main advantage of the model-based approach is that the models learned using this approach receive more supervision, making it more similar to a supervised learning problem [24, 52]. Interestingly, offline model-based RL suffers from the overestimation problem in the same way as model-free RL does if the data set does not contain enough information to recover the true reward and transition functions. This problem is also present in model-based RL because it aims to maximize the expected return of the approximated reward function. Therefore, the same missing information and poorly selected proxy optimization objective problems that caused overestimations in model-free RL are also present in model-based RL [9].

The Model-based Offline Policy Optimization (MOPO) [52] algorithm is a model-based algorithm that aims to prevent the overestimation problem using an approach that is similar to ours. The MOPO algorithm prevents these overestimations in three steps. Firstly, it learns an ensemble of rewards and transition functions, where each model is trained independently via maximum likelihood. In the second step, the MOPO algorithm uses these ensembles to create a pessimistic version of the underlying MDP that penalizes the reward in areas where the model is uncertain. This uncertainty is measured as the maximum difference between different predictions within an ensemble of models. Formally, this means that the pessimistic reward function is defined as:

$$\hat{R}_p(s,a) = \frac{1}{N}\sum_{i=1}^{N} R(s,a;\psi_i) - \lambda \max_{i,j} \left\{ \|\hat{R}(s,a;\psi_i) - \hat{R}(s,a;\psi_j)\|^2 + \|\hat{T}(s,a;\psi_i) - \hat{T}(s,a;\psi_j)\|^2 \right\} \quad (6.4)$$

In this equation, $\hat{R}_p$ is the pessimistic reward function, $\hat{R}(s,a;\psi_i)$ is the i-th likelihood estimate of the reward function, $\hat{R}(s,a;\psi_i)$ is the i-th likelihood estimate of the transition function, N is the number of ensemble members, and λ is a hyperparameter that controls the strength of the pessimistic penalty.

In the final step, the MOPO algorithm uses any existing model-free algorithm to sample and explore the newly created MDP to find a policy that maximizes the expected return in this pessimistic version of the underlying MDP. Besides this algorithm, there are other pessimistic offline deep model-based RL algorithms [24, 52]. However, since they are completely identical except for their slightly different reward function, we decided not to discuss them in more detail.

These model-based methods approach the overestimation problem using an approach that is similar to our method. Both the model-based approaches and our method penalize their predictions using an ensemble-based uncertainty estimate. However, there is one important difference between the model-based methods and ours. The model-based algorithms measure their uncertainty based on the differences within their ensemble of reward and transition functions. In contrast, our method measures its uncertainty based on the differences within its ensemble of Q-value functions. This difference is important because both methods eventually aim to optimize a Q-function. Therefore, the model-based algorithms do not learn a pessimistic policy using an end-to-end based technique, while our method does. This is potentially an issue because when an empirical Bellman update is performed on a function approximated Q-network, the value of a particular state is impacted by generalizations from other states, which potentially confounds the pessimistic penalties [9]. This issue could potentially be solved by ensuring that the reward function and transition function, value function, and policy have the same internal representation and generalization by sharing an encoder network [17]. However, as far as we are aware, no method does this in pessimistic model-based RL. Therefore, our method has the advantage since its uncertainty measure is aware of the internal generalizations of the function approximator that is being optimized.

6.4. Conclusion

In the previous sections, we discussed multiple prior approaches to the offline deep RL problem. Based on these results, we conclude there exists a gap in the offline deep RL literature for an end-to-end model-free uncertainty-aware pessimistic algorithm, which we aim to address using the methods proposed in this thesis. In the next chapter, we will compare our proposed methods experimentally against these previously discussed offline deep RL methods.

7

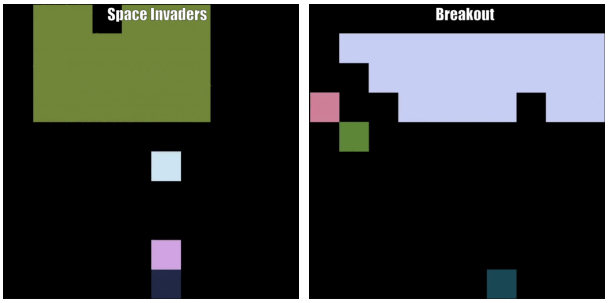
Evaluation experiments

This chapter discusses the different experiments we performed to test our PEssimistic ensemBLe (PEBL) methods in three different environments. The first environment is the MinAtar environment suite [51], which contains several miniaturized versions of the Atari 2600 games. We use the flexibility and computational efficiency of this MinAtar suit to experimentally validate that our PEBL methods work as expected and to validate that our PEBL method follows the theoretically predicted properties of uncertainty-aware algorithms on a wide range of data set compositions. The second environment we use is the Maze-2D environment from the D4RL benchmark [10]. We use this environment to experimentally show that our PEBL method has the unique ability to solve undirected data set problems. Undirected data set problems are offline reinforcement learning (RL) problems whereby the demonstrator's behavior does **not** align with the goal to maximize the expected discounted return. Finally, we used the MuJoCo gym task from the D4RL benchmark to experimentally compare our method against current state-of-the-art offline RL techniques [8, 10, 44].

7.1. MinAtar

In this section, we experimentally validate that our PEssimistic ensemBLe (PEBL) methods work as expected and we validate that our PEBL method follows the theoretically predicted properties of uncertainty-aware algorithms on a wide range of data set compositions. The result of these experiments will allow us to answer research question 2.1. We perform these experiments using the MinAtar environment. This environment is a testbed for reinforcement learning agents which implements a miniaturized version of several Atari 2600 games such as Space Invaders and Breakout [51]. MinAtar is comparable to the arcade learning environment [6]. However, MinAtar simplifies the games and their observations by reducing the game to an 10×10 grid and the observations to an $n \times 10 \times 10$ boolean grid. Each of these n channels indicates a game-specific object's location, such as the ball, the paddle, and the bricks in the game breakout (Figure 7.1). This simplified version of the game still captures the general mechanics and difficulty of the behavior task while simplifying the representational complexity of the game. With the representation learning problem simplified, it is possible to perform significantly more experiments, even for a relatively low computational budget such as ours.

Currently, there exist no pre-collected offline reinforcement learning (RL) data set for the MinAtar suite. However, in the offline RL literature, it is common practice to create these data sets yourself [2, 9, 11, 12, 29]. In this process, we use an online RL method to find a near-optimal policy in the environment. At this stage, the agent is still allowed to interact with the environment. This near-optimal policy can then be used to collect different types of data sets by varying the data collection strategy. Using this policy, we generate the *ϵ-greedy* data. This data set has been collected using an ϵ-greedy data collection strategy, which means that for every transition, the data collection agent takes either a greedy action with probability $1 - \epsilon$ or it takes a random action with probability ϵ. When the value of ϵ increases, the data collection strategy will collect less optimal data, thus allowing us to control the optimality of the data sets. Increasing the value of ϵ also reduces the bias for high rewarding trajectories because the randomly sampled action will cause the data collection strategy to sample less



(a) Space Invaders (b) Break out

Figure 7.1: A visualization of some of the Atari 2600 games in the MinAtar suite. The color of a block indicates a game-specific object's channel at a specific location on the 10×10 grid. Note that the agent does not observe these colors, but instead, it observes an $n \times 10 \times 10$ boolean grid, where n is the number of different types of objects.

rewarding trajectories. By collecting multiple data set with varying values for ϵ, it is possible to validate the properties of the offline RL method when the optimality of the data collection strategy changes. Using these data sets, we can experimentally validate that our uncertainty-aware algorithm has the theoretically predicted properties of uncertainty-aware algorithms and that these properties hold even when the optimality of the data collection changes with ϵ. The remainder of this section will discuss the experiments we performed using the ϵ-*greedy* data set to validate these properties.

7.1.1. Overestimation properties

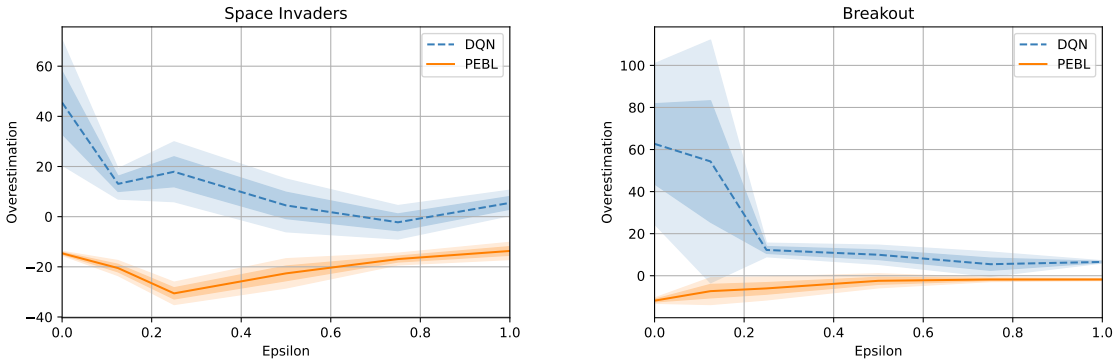
The first property we will validate is that our uncertainty-aware algorithm adheres to the pessimism principle. This principle states that the agent should learn a policy that acts optimally in the worst possible world. Uncertainty-aware algorithms adhere to this principle by learning a pessimistic value function, which does not overestimate the expected discounted return in the real environment [9]. Using the ϵ-*greedy* data set, we can validate that an agent learns a value function that does not overestimate the expected discounted return in three steps. First, we train the agent on an ϵ-*greedy* data set to obtain its policy and value function. Secondly, we evaluate this policy in the real environment for 250 episodes per learned policy. However, before starting every evaluation episode, we first let the value function estimate the expected discounted return given the initial state. Finally, we can assess how much the value function overestimates by comparing the predicted expected discounted return and the real return obtained by the policy.

Baseline In this experiment, we evaluate the overestimation capabilities of our PEBL Double Deep Q-Network (DDQN) method. In this experiment, we use the DDQN algorithm as a baseline because this is a member of the naive offline RL algorithmic family [47]. These naive offline RL algorithmic family members are an ideal baseline because these methods have no value function overestimation technique build in [9].

Results The outcome of the experiment is shown in Figure 7.2. The results show that the value function learned by our PEBL DDQN method does not overestimate the expected discounted return for any ϵ value in both the Space Invaders and Breakout environment. In contrast, the baseline DDQN method overestimates the expected discounted return for every ϵ value. These results experimentally show that our PEBL DDQN method adheres to the pessimism principle, even when the optimality of the data collection changes.

7.1.2. Dependence on the optimality of the data collection strategy

The second property we will validate is that our uncertainty-aware algorithm learns a policy that is less dependent on the empirical policy in the data set than the policy-constrained methods [9]. We will experimentally validate this property by replicating the results presented by Buckman et al. in their paper (2020). Their original experiment only compared the behavior cloning, policy-constrained, and naive algorithmic families, excluding the deep uncertainty-aware algorithmic family, leaving it as an open research question. Their original experiments used the same ϵ-*greedy* data collection strategy and



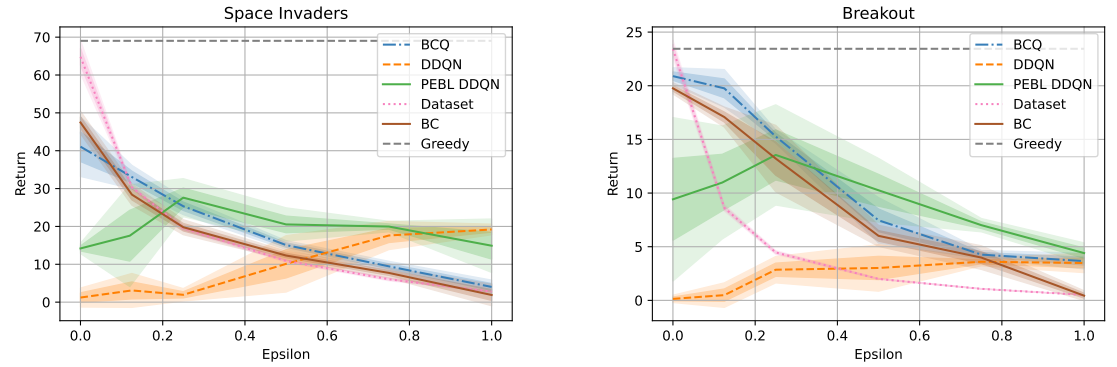
(a) Space Invaders (b) Breakout

Figure 7.2: The results for the over-estimation property experiment. Our PEBL DDQN method does not over-estimate the expected discounted return for any ϵ in both the Space Invaders and Breakout environment. In contrast, the naive DDQN method overestimates for every ϵ. The measurements are obtained by taking the average overestimation of the final policy over 250 episodes averaged over 3 independent training runs each with their own random seeds.

the same MinAtar environments [9]. In our version of the experiment, we train every method for $75,000$ gradient updates on a data set of $50,000$ samples. We chose this relatively small data set because it amplifies the effect of missing information, highlighting the advantage of the uncertainty-based method.

Baseline In this experiment, we compare the performance of PEBL DDQN against the performance of the three other offline RL algorithmic families. Our version of the experiment will use the behaviour cloning (BC) algorithm to represent the imitation learning family [21]. The policy-constrained algorithmic family is represented by the Batch-Constrained Q-learning (BCQ) algorithm [11, 12]. Finally, the naive algorithmic family is represented by the DDQN algorithm [47].

Results The outcomes of the experiment are shown in Figure 7.3. The results show that our PEBL DDQN method often performs best for a wider range of ϵ-greedy policy data sets than the other algorithmic families. This result is mostly in line with the theoretical results we aimed to replicate [9]. However, interestingly our method does not follow the theoretically predicted behavior for uncertainty-aware pessimistic algorithms in the expert data regime when ϵ approaches zero. We expected the method to perform as well or similarly to the behavior cloning method in this area, but it falls notably short. This suggests that our epistemic uncertainty estimation technique is not expressive enough for highly biased data in this expert data regime.



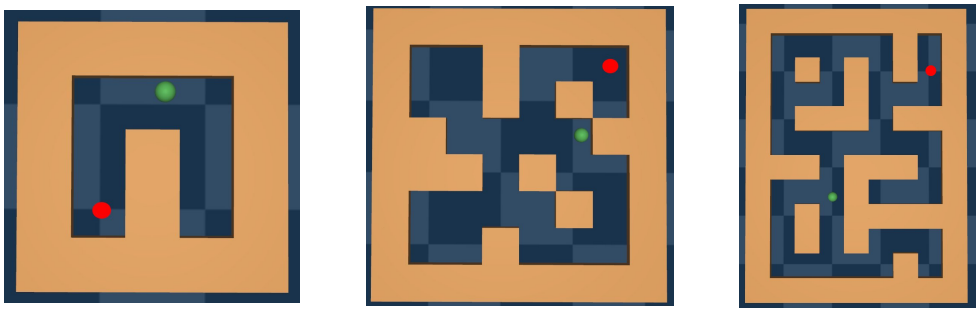
(a) Space invaders (b) Breakout

Figure 7.3: The performance of different representatives of each offline algorithmic family compared to our PEBL DDQN algorithm on the ϵ-greedy data set. The measurements are obtained by taking the average performance of the final policy over 250 episodes averaged over 3 independent training runs each with their own random seeds.

7.2. D4RL: maze-2D

This experiment aims to show the unique ability of our uncertainty-aware algorithm to learn from undirected data sets. Undirected data set problems are offline reinforcement learning (RL) problems whereby the demonstrator's behavior does **not** align with the goal to maximize the expected return. We experimentally show that our method has this ability by applying our PEssimistic ensemBLe (PEBL) Soft Actor Critic (SAC) to the maze-2D task of the D4RL benchmark [10].

In the maze-2D task, agents must learn a continuous action policy that can move a ball from any point in a maze to a pre-specified point (Figure 7.4). The data set used in this task consist of random routes through the maze. Thus, the agent must stitch together different trajectories to learn the shortest path to the goal location from every possible starting point. This property makes the maze-2D task an undirected data set problem because the demonstrator's behavior does not align with the goal to maximize the expected discounted return. Another interesting property of the data set is that the observations contain only the agent's current coordinates and velocity. The agent is thus unable to see the entire maze and has to learn the layout from the data set. These properties have made the larger maze-2D tasks very difficult for policy-constrained methods because they cannot deviate enough from the behavior policy even though the data set covers every possible location in the maze [10].



(a) U-maze (b) Medium maze (c) Large maze

Figure 7.4: A visualization of the different mazes in the maze-2D task of the D4RL benchmark. The agent is the green ball, and the red dot is the endpoint. Note that the agent does not observe these images, but instead, it observes a state-vector, which contains the coordinates and velocities of the ball. Thus, the agent has to learn the layout of these mazes based only on their data sets.

Baselines We compare our method to the results provided in the D4RL benchmark paper for Soft Actor Critic (SAC), behaviour cloning (BC), and Continuous Batch-Constrained Q-learning (CBCQ) [10, 11].

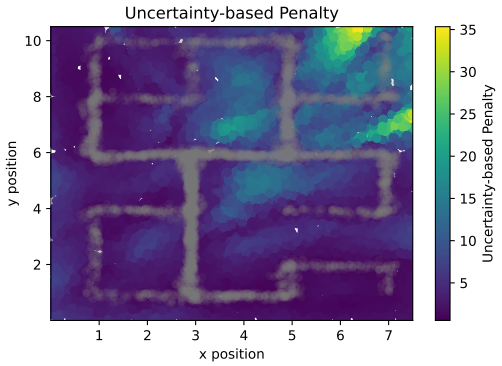
Results The performance of the algorithms on the Maze-2D experiment is displayed in Table 7.1. PEBL SAC can deviate further from the observed behavior policy than the other baselines. This allows it to solve all mazes, including the large maze, which has not yet been solved by any existing policy-based pessimistic method [10]. Figure 7.5 visualizes the value of the uncertainty-based penalty and the penalized Q-values. We see that the result of our pessimistic penalty is that the agent assigns low Q-values to areas which it is uncertain about, such as positions occupied by walls. These results empirically show that our method only needs sufficient information about the Markov decision process (MDP) to find the optimal policy. The fact that the empirical behavior in this data set is not optimal and even counterproductive does not matter for our method. This is a very desirable property that is only present in uncertainty-aware algorithms [9], which make our method preferred for these kinds of data sets .

7.3. D4RL: MuJoCo gym task

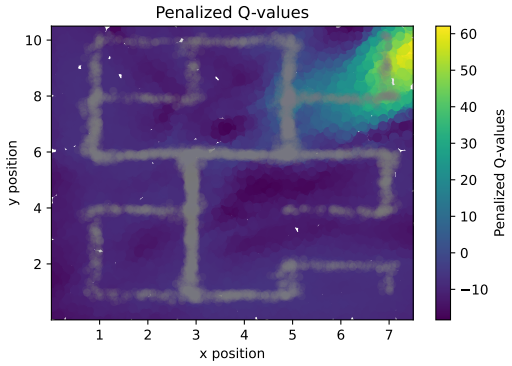
In this experiment, we aim to answer research question 2.2: *How well does our method perform compared to prior offline deep reinforcement learning (RL) methods?* We do this by experimentally comparing

Dataset type	SAC	BC	CBCQ	PESAC
U	88.2	3.8	12.8	**151.1**
Medium	26.1	30.3	8.3	**146.6**
Large	-1.9	5.0	6.2	**129.5**

Table 7.1: Results for D4RL's Maze-2D benchmark. Each score is the average normalized reward over 100 runs at the last iteration of training.



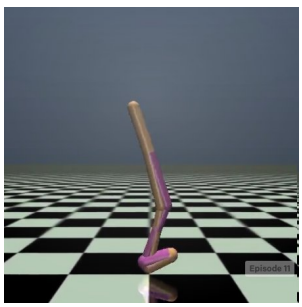
(a) The uncertainty-based penalty $C_\pi \cdot \overline{\sigma}_Q(s, a; \phi_1, \phi_2)$ where C_π was learned and converged to $C_\pi = 12.53$.



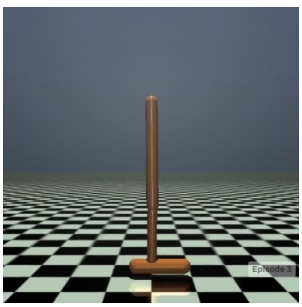
(b) The the penalized Q-values $\overline{\mu}_Q(s, a; \phi_1, \phi_2) - C_\pi \cdot \overline{\sigma}_Q(s, a; \phi_1, \phi_2)$.

Figure 7.5: A visualization of the uncertainty-based penalty and the penalized Q-values for the large Maze-2D environment. The gray areas in both figures indicate the number of data points available in the data set per location. Note that the agent only observes (x, y, v_x, v_y), and still, the agent can learn where the walls are located using its own uncertainty.

our method against current state-of-the-art offline RL techniques, using the MuJoCo gym task from the D4RL benchmark [10]. All the data sets in the benchmark task are available online, and they have been extensively benchmark by prior work, allowing us to compare our method against a wide range of prior methods.
In the MuJoCo gym task, the agent has to learn a continuous action policy that controls the joints of various robots in the MuJoCo physics simulator [44]. The goal of these tasks is to learn a policy that moves the robot as fast as possible to the right edge of the simulator. Depending on the type of robot, they will either learn to walk (Figure 7.6a), jump (Figure 7.6b), or run like a cheetah (Figure 7.6c).



(a) Walker2d



(b) Hopper

(c) Half Cheetah

Figure 7.6: A visualization of the different robots in the MuJoCo gym task of the D4RL benchmark. Note that the agent does not observe these images, but instead, it observes a state-vector, which contains the coordinates and velocities of each joint.

This benchmark has four different data sets available for each of the tasks, each consisting of one million transitions, that test the offline RL agent's abilities. The *expert* data set was collected by first training a Soft Actor Critic (SAC) agent to near-optimal performance, which was then used to collect the data set. This *expert* data set aims to evaluate how well an offline RL agent can handle the narrow distribution of expert data sets. The *medium* data set was also collected by first training a SAC agent, but this training process was stopped early when the agent reached medium performance. This partially trained policy

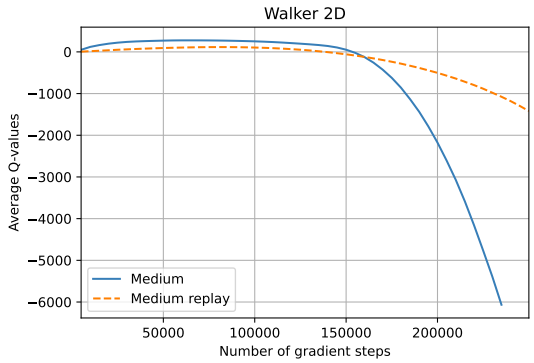
was then used to collect the *medium* data set. This *medium* data set aims to evaluate how much an offline RL agent can improve upon the partially trained policy. The *medium-replay* data set consists of the last one million samples from the replay buffer used to train the medium agent. This *medium-replay* data set aims to evaluate an offline RL agent's ability to handle non-stationary data sets. Finally, the *random* data set was collected by randomly re-initializing the policies for every roll-out. This *random* data set aims to evaluate how much an offline RL agent can learn from random data. All these data sets are available online, and they have been extensively benchmark, allowing us to compare our method against a wide range of prior work.

Baselines Similar to the experiment in section 7.1.2, we compare our methods to representatives of each algorithmic family. The methods are Soft Actor Critic (SAC) [16] and Continuous Batch-Constrained Q-learning (CBCQ) [13] for the naïve and the policy-based pessimistic algorithmic family. We also compare our method to Conservative Q-Learning (CQL) [27] and Conservative Offline Model-Based policy Optimization (COMBO) [53]. Both methods are considered state-of-the-art offline RL methods, and both methods are members of the policy-based pessimistic algorithmic family [9]. Compared to other policy-based pessimistic algorithms, these methods have a significantly looser constraint, allowing for larger improvements upon the data collection policies.

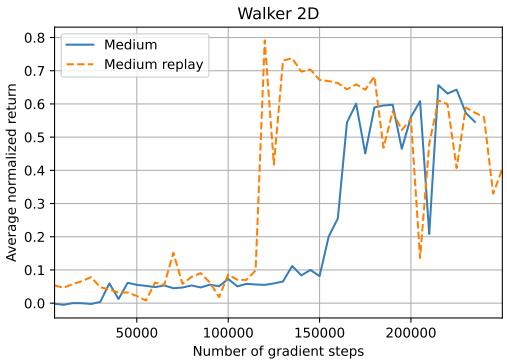
Results The results are shown in Table 7.2, where the performance of SAC, BC, and CBCQ is given as reported in [13], and for CQL and COMBO are the number reported in [27] and [53], respectively. Similar to the PEBL DDQN algorithm, we cannot match the performance of behavior cloning in the expert data regime. In contrast, our method performs remarkably well in the *half cheetah* environment for data sets generated using non-expert policies, marked *Random*, *Medium*, and *Medium-replay*. It even achieves state-of-the-art results for the *Medium* and *Medium-replay* data sets. This is quite interesting since the current state-of-the-art method is a model-based method while PEBL SAC is a model-free method. In the walker2d and hopper environments, the performance of our method is nowhere near state-of-the-art. In these environments, our method only achieves results comparable to policy-constrained methods such as CBCQ. This result is especially interesting for the *Walker2D-Medium*, and *Walker2D-Medium-Replay* data set since the value function learned by our method diverges here to large negative values. With this diverging value function, we would expect the method not to learn anything, but surprisingly the method only starts to perform reasonably well when the divergent behavior starts (Figure 7.7). We suspect this divergent behavior happens because the method tends to have relatively high uncertainty for the walker2D data set. This high uncertainty results in large penalties, which in turn result in larger negative value targets. Thus, creating the negative feedback loop, we observe in the learning curve. Interestingly, these large uncertainty values still indicate the method's epistemic uncertainty and thus allow the method to keep on performing reasonably well.

Dataset type	Environment	SAC	BC	CBCQ	CQL	COMBO	PEBL SAC (ours)
Random	halfcheetah	30.5	2.1	2.2	35.4	**38.8**	35.7
Random	hopper	11.3	1.6	10.6	10.8	**17.9**	9.5
Random	walker2d	4.1	**9.8**	4.9	7.0	7.0	3.6
Medium	halfcheetah	-4.3	36.1	40.7	44.4	55.1	**61.5**
Medium	hopper	0.8	29.0	54.5	86.6	**94.9**	53.2
Medium	walker2d	0.9	6.6	53.1	74.5	**75.5**	55.9
Medium-Replay	halfcheetah	1.9	38.4	38.2	46.2	55.1	**57.5**
Medium-Replay	hopper	3.5	11.8	33.1	48.6	**73.1**	37.8
Medium-Replay	walker2d	1.9	11.3	15.0	32.6	**56.0**	26.3
Expert	halfcheetah	-1.9	**107.0**	-	104.0	-	3.6
Expert	hopper	0.7	109.0	-	**109.9**	-	2.5
Expert	walker2d	-0.3	**125.7**	-	121.6	-	3.9

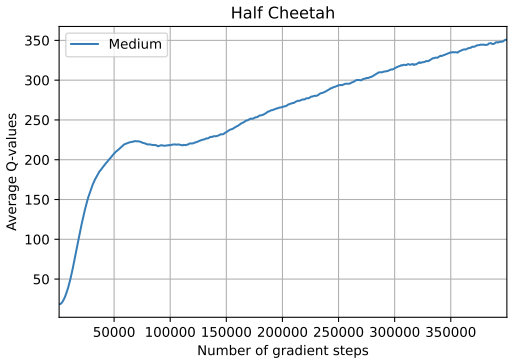
Table 7.2: Results for the D4RL benchmark. Each score is the average normalized reward over 100 runs at the last iteration of training as described in the D4RL benchmark.



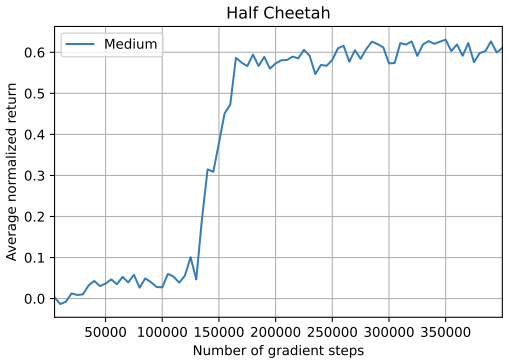
(a) The Q-value learning curve of PEBL SAC for the walker2D data sets.



(b) The performance in the real environment learning curve of PEBL SAC for the walker2D data sets.



(c) The Q-value learning curve of PEBL SAC for the half cheetah data set.



(d) The performance in the real environment learning curve of PEBL SAC for the half cheetah data set.

Figure 7.7: Surprisingly, our PEBL SAC method shows divergent behavior for the *walker2D-medium* and *walker2D-medium-replay* data sets. In contrast, our method shows stable behavior for the other data sets. Therefore, we included the learning curves for the *half-cheetah-medium* data sets as a reference point.

8

Conclusion and future research

This thesis has investigated how offline deep reinforcement learning (RL) agents can be instilled with the ability to *know what they do not know*. In this chapter, we will summarize the main conclusion and contributions of our work, and address the limitations and possible future directions of our proposed methods.

8.1. Conclusion

Before we can create an uncertainty-aware offline deep RL algorithm, we need to know what the pessimism principle is and which theoretical requirements it imposes on the uncertainty estimation techniques. To this end, Chapter 3 explained that the pessimism principle is a theoretical framework for offline RL that prevents overestimations in the value function of offline RL agents caused by missing information in the data set and poorly selected proxy optimization objectives. Uncertainty-aware algorithms, the algorithmic family we are interested in, implements this principle by penalizing the agent's value function using an uncertainty-aware penalty. This algorithmic family has the property that it will learn an information-theoretical optimal policy independent of the data collection strategy used to gather the data set. The only requirement for this optimality is that the penalty should be an upper bound on the difference between the real value function and the empirical maximum likelihood-based value function [9, 38, 46]. Currently, there exists no mathematical proof that any deep uncertainty estimation technique has this property. However, it is hypothesized that in practice, it should be possible to approximate this requirement if the estimated uncertainty has two properties [9]. Firstly, the estimated uncertainty should be high in areas with insufficient information to model the true Markov decision process (MDP). Secondly, the estimated uncertainty should be low in areas with sufficient information to model the true MDP. Only deep epistemic uncertainty estimation techniques come close to this property. Therefore, the main theoretical requirement imposed by the pessimism principle is that the used deep uncertainty estimation technique should have excellent epistemic uncertainty measurement capabilities, which will help to prevent the proxy objective from overestimating the true objective.

Besides this theoretical requirement, there are also implementation-related requirements that determine if a deep uncertainty estimation technique is suitable for offline deep RL. These implementation-related requirements originate from the fundamental differences between reinforcement learning and supervised learning problems. Chapter 4 investigated these implementation-related requirements and came to the conclusion that there are six major implementation-related requirements. Firstly, when an empirical Bellman update is applied to a neural network, the change in value can impact any state due to the generalization capabilities of the network [9, 35]. Therefore, the epistemic uncertainty must be estimated in an end-to-end manner such that the estimate is aware of the internal generalization of the model. Secondly, the epistemic uncertainty-based penalty must depend on the current policy and value target estimate, which keep changing during the learning process. Therefore, the uncertainty estimation method must be able to capture these changes, allowing it to measure the current knowledge uncertainty with respect to both the current policy and value function in an online learning-based manner. Thirdly, the epistemic uncertainty-based penalty must be calculated for every value function loss calculation.

Therefore, calculating the epistemic uncertainty should have minimal computational costs to keep the learning speed of the algorithm manageable. Fourthly, the epistemic uncertainty-based penalty will be used to calculate a lower-bound estimate of the temporal difference target by subtracting the penalty from the value estimate. Therefore, it is desirable if the uncertainty estimation is in the same units as the value estimate. Fifthly, the temporal difference targets used to learn the value function keep changing over type, due to the bootstrapping procedure used to obtain these targets. Therefore, it is essential that the uncertainty estimation technique works with these moving targets and does not assume any stationary in the temporal difference targets. Finally, deep RL algorithms tend to have a high amount of variance in their learning signals. These algorithms can become unstable and will fail to learn when their learning signal variance increases even further [48]. Therefore, the used uncertainty estimation technique must introduce a minimal amount of additional variance into the learning signal. Thus, if a deep uncertainty estimation technique adheres to these implementation requirements and the theoretical requirement, it should be suitable for an uncertainty-aware offline RL algorithm.

Based on these theoretical and implementation-related requirements, we found the multi-headed bootstrap ensemble with random priors the most suitable uncertainty estimation technique for our purposes [35]. In Chapter 5, we showed how this uncertainty estimation technique was used to create our model-free uncertainty-aware offline deep RL algorithms. The first algorithm, named PEBL DDQN, is specifically designed for discrete action spaces, while the second algorithm, named PEBL SAC, is designed for continuous action spaces.

To validate that our PEBL methods adhere to the pessimism principle and mimic the properties of uncertainty-aware algorithms, we tested our methods in multiple environments and on multiple data sets with a wide variety of data collection strategies in Chapter 7. In every situation, the learned value function learned by our method always underestimates the real expected discounted return, showing that the method adheres to the pessimism principle. Even with this underestimating value function, the methods performed well on a wide range of data set distributions. The methods matched the theoretically predicted performance when the data collection strategy was between uniform random and semi-expert. However, the methods did not match the theoretically predicted performance when the data was collected using an expert data collection strategy. This is interesting because, based on the theoretical predictions for uncertainty-aware algorithms, we expected our algorithm to perform as well on these types of data sets as behavior cloning-based methods. Although our methods do not perform as well as predicted for these expert data sets, it still performs significantly better on these expert data sets than naive algorithms that do not adhere to the pessimism principle. However, our methods are still outperformed on these expert data set by policy constrained-based methods and behavior cloning-based methods. These results thus show that our deep epistemic uncertainty estimation technique can approximate the theoretical requirements for the uncertainty-aware algorithm, especially for data sets collected using either a uniform random or a semi-expert data collection strategy. However, our method's approximation of the theoretical requirement is not precise enough to obtain the expected performance for the relatively biased and sparse data distribution of expert data sets.

With these uncertainty-aware properties validated, the only research question left is: *how well does our method perform compared to prior offline deep RL methods*? To answer this research question, we empirically tested our method against the D4RL MuJoCo gym benchmark [10]. This benchmark is widely used in offline deep RL [19, 27, 53] and thus allows us to compare our method against a wide range of prior work. The results from this experiment allowed us to draw two conclusions. Firstly, our method significantly underperforms for data sets in the expert data regime compared to prior work. This result supports our previous conclusion that our uncertainty measure is not strong enough for the expert data regime. Therefore, we can conclude that our method in its current state is not yet suitable for these expert data sets. Secondly, our method shows great potential for data sets generated by non-optimal data collection strategies. For these types of data sets, our method even obtained state-of-the-art performance for one of the three environments. In this specific environment, our method even performed better than a model-based method, even though our method is a model-free method. However, our method also became unstable for the same type of data set in another environment. Interestingly, our method still performed reasonably well despite this instability. Therefore, we can conclude that our method can be quite suitable for these non-expert data set. However, these instability issues

must be removed first before our method can be safely picked over prior work.

8.2. Limitations and future work

Our method has some interesting and desirable properties for offline RL. However, it also has certain limitations that limit its practical usage. For example, our methods do not match the theoretically predicted performance of uncertainty-aware reinforcement learning algorithms for expert data sets. This property is problematic because the theoretical appeal of uncertainty-aware algorithms is that they should work on any data set independent of the used data collection strategy. Based on our results, we suspect that our method performs poorly on these data sets because our method's approximation of the theoretical requirement of the pessimism principle is not precise enough. Therefore, we suspect that future work can significantly improve upon our results if it can identify a deep uncertainty estimation technique with stronger epistemic uncertainty estimation capabilities.

Another limitation of our method is its instability in the walker2D data sets. Based on our experiments, we suspect this instability is caused by the relatively high epistemic uncertainty that is present in this data set. This increased epistemic uncertainty results in larger uncertainty penalties, which in turn result in larger negative learning targets. This creates a negative feedback loop because the uncertainty measured by our epistemic uncertainty estimation technique is dependent on the predictions' magnitudes. Therefore, we suspect that future work can remove this instability if it can identify an uncertainty estimation technique independent of the predictions' magnitudes. A promising research direction for this problem is feature-based epistemic uncertainty estimation. This family of methods estimates the epistemic uncertainty based on the internal representations of the method instead of the predicted values. Although these methods exist in the supervised learning domain [31, 43, 45], we were unable to identify such a method that works in the deep RL domain. However, we hope that future work will identify such a method that also works in the deep RL domain.

The final limitation of our method is its computational resource requirements. Our epistemic uncertainty estimation technique uses an ensemble of methods to measure epistemic uncertainty. This property significantly increases the number of weights in the network, increasing the GPU memory requirements and learning time. These computational resources are currently manageable for the relatively small network architectures used in deep RL [16, 33, 47]. However, this approach will not scale to larger networks. This limitation has also been identified in the supervised-based uncertainty estimation domain, which has resulted in the single model uncertainty estimation family [31, 43, 45]. However, we have been unable to identify a method in this family that also works in the deep RL domain so far. We hope that future work will identify such a method that also works in the deep RL domain, allowing them to remove this computational resource bottleneck.

Bibliography

[1] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Abbas Khosravi, U Rajendra Acharya, Vladimir Makarenkov, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *arXiv preprint:2011.06225*, 2020.

[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[3] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6:14410–14430, 2018.

[4] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint:1910.07113*, 2019.

[5] Brian Beavis and Ian Dobbs. *Optimisation and stability theory for economic analysis*. Cambridge university press, 1990.

[6] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[7] Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. cite arxiv:1606.01540.

[9] Jacob Buckman, Carles Gelada, and Marc G Bellemare. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint:2009.06799*, 2020.

[10] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020.

[11] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[12] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint:1910.01708*, 2019.

[13] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[14] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[16] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint:1812.05905*, 2018.

[17] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint:1912.01603*, 2019.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4 (2):251–257, 1991.

[19] Qiang He Hou et al. Popo: Pessimistic offline policy optimization. *arXiv preprint:2012.13682*, 2020.

[20] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[21] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[22] Nan Jiang and Jiawei Huang. Minimax value interval for off-policy evaluation and policy optimization. *Advances in Neural Information Processing Systems*, 33, 2020.

[23] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? *arXiv preprint:2012.15085*, 2020.

[24] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21810–21823. Curran Associates, Inc., 2020.

[25] Seungwon Kim. Stabilizing off-policy q-learning via bootstrapping error reduction, 2020. Submitted to NeurIPS 2019 Reproducibility Challenge.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint:1412.6980*, 2014.

[27] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint:2006.04779*, 2020.

[28] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[29] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint:2005.01643*, 2020.

[30] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page. *arXiv preprint:1712.09381*, 2017.

[31] Jeremiah Zhe Liu, Zi Lin, Shreyas Padhy, Dustin Tran, Tania Bedrax-Weiss, and Balaji Lakshminarayanan. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *arXiv preprint:2006.10108*, 2020.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint:1312.5602*, 2013.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[34] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[35] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *arXiv preprint:1806.03335*, 2018.

[36] Aske Plaat, Walter Kosters, and Mike Preuss. Model-based deep reinforcement learning for high-dimensional problems, a survey. *arXiv preprint:2008.05598*, 2020.

[37] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[38] Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. Bridging offline reinforcement learning and imitation learning: A tale of pessimism. *arXiv preprint:2103.12021*, 2021.

[39] Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. Uncertainty-based out-of-distribution classification in deep reinforcement learning. *arXiv preprint:2001.00496*, 2019.

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[42] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] Natasa Tagasovska and David Lopez-Paz. Single-model uncertainties for deep learning. *arXiv preprint:1811.00908*, 2018.

[44] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[45] Joost Van Amersfoort, Lewis Smith, Yee Whye Teh, and Yarin Gal. Uncertainty estimation using a single deep deterministic neural network. In *International Conference on Machine Learning*, pages 9690–9700. PMLR, 2020.

[46] Joost van Amersfoort, Lewis Smith, Andrew Jesson, Oscar Key, and Yarin Gal. Improving deterministic uncertainty estimation in deep learning for classification and regression. *arXiv preprint:2102.11409*, 2021.

[47] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[48] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint:1812.02648*, 2018.

[49] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[50] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint:1911.11361*, 2019.

[51] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint:1903.03176*, 2019.

[52] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14129–14142. Curran Associates, Inc., 2020.

[53] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *arXiv preprint:2102.08363*, 2021.

Appendices

A

Experimental details

A.1. MinAtar experiments

Hyperparameter	Setting	Used in algorithms
Q-network: channels	16	BC, DDQN, PEBL DDQN, BCQ
Q-network: filters	3×3	BC, DDQN, PEBL DDQN, BCQ
Q-network: strides	1	BC, DDQN, PEBL DDQN, BCQ
Q-network: hidden units	128, 128	BC, DDQN, PEBL DDQN, BCQ
Q-network: number of heads	15	PEBL DDQN
Q-network: activation function	ELU	BC, DDQN, PEBL DDQN, BCQ
Q-network: prior weight β	10	PEBL DDQN
Q-network: ensemble implementation	conv1d	PEBL DDQN
Target network update period	every 500 steps	DDQN, PEBL DDQN, BCQ
Target network procedure	hard copy	DDQN, PEBL DDQN, BCQ
Imitation threshold	0.3	BCQ
Imitation regularizer weight	0.01	
Optimizer	Adam	BC, DDQN, PEBL DDQN, BCQ
Learning rate	0.00025	BC, DDQN, PEBL DDQN, BCQ
Maximum gradient norm	10	BC, DDQN, PEBL DDQN, BCQ
Number of gradient updates	75,000	BC, DDQN, PEBL DDQN, BCQ
Data set size	50,000 transitions	BC, DDQN, PEBL DDQN, BCQ
Batch size	256	BC, DDQN, PEBL DDQN, BCQ
Discount factor	0.99	DDQN, PEBL DDQN, BCQ
Number of evaluation episodes	250	BC, DDQN, PEBL DDQN, BCQ
Seeds	0, 1, 2	BC, DDQN, PEBL DDQN, BCQ
Hardware: GPU	Geforce RTX 2070	BC, DDQN, PEBL DDQN, BCQ

Table A.1: The hyperparameters used in the MinAtar experiments. The BCQ specific hyperparameters come from the author's implementation on GitHub [12]. The DDQN related hyperparameters are inspired by the parameters used in the RLLIB library [30]. Finally, the architecture of the Q-network is the recommended architecture for the MinAtar environment [51].

A.2. D4RL experiments

Hyperparameter	Setting
Policy network: hidden units	256, 256
Q-network: activation function	ELU
Q-network: hidden units	256, 256
Q-network: number of heads	15
Q-network: activation function	ELU
Q-network: prior weight β	10
Q-network: ensemble implementation	conv1d
Target network update period	every 500 steps
Target network procedure	polyak
Polyak rate τ	0.001
Auto entropy trade-off parameter tuning	Enabled
Auto uncertainty trade-off parameter tuning	Enabled
Optimizer	Adam
Learning rate	0.00025
Maximum gradient norm	10
Number of gradient updates	500,000
Batch size	256
Discount factor	0.99
Number of evaluation episodes	250
Seeds	0
Hardware: GPU	Geforce RTX 2070

Table A.2: The hyperparameters used in the experiments for both the maze2D and MuJoCo gym tasks in the D4RL benchmark. The hyperparameters are inspired by the parameters used in the SAC implementation of the RLLIB library [30]

B

Hyper-parameter tuning experiments

B.1. Assessing the effect of the prior weight parameter

This experiment assesses the effect of prior weight parameter β on the performance of the PEssimistic ensemBLe (PEBL) Double Deep Q-Network (DDQN) method. The original paper suggested that this hyperparameter should either be set to 3 or 10 [35]. Thus, we trained our method with both values on the ϵ-*greedy* data set and evaluated the final performance to assess the effect of this parameter.

Results The experiment's outcomes are shown in Figure B.1. These results show that the parameter has little effect on the average performance of the method. However, the parameter significantly affects the variance in the performance between random seeds when the value is relatively low. Therefore, we decided to use $\beta = 10$ in our experiments.

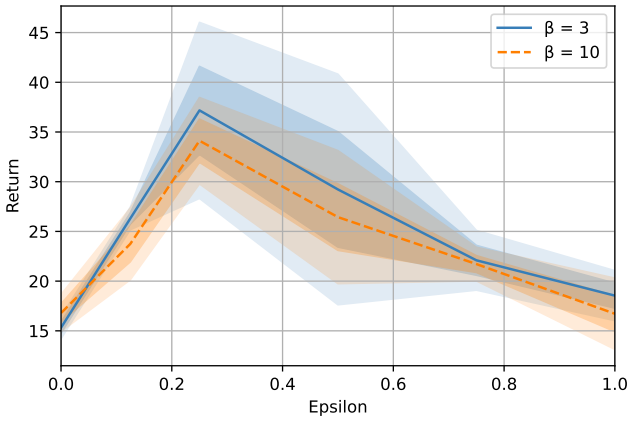


Figure B.1: The performance of the PEBL DDQN method depending on the prior weight hyperparameter. The measurements are obtained by taking the average return of the final policy over 250 episodes averaged over 3 random seeds. There is no statistical significant difference between $\beta = 3$ and $\beta = 10$. However, the method appears to have a lower variance over random seeds with $\beta = 10$.

B.2. Assessing the effect of the number of ensemble heads

This experiment assesses the effect of the number of ensemble heads H on the performance of the PEBL DDQN method. This experiment tests the values $H = 10$ and $H = 15$. We do this by training our PEBL DDQN method using both values on the ϵ-*greedy* data set. The resulting policies obtained by both values are evaluated to assess the effect of this parameter.

Results The experiment's outcomes are shown in Figure B.2. Interestingly, there is no significant difference between $H = 10$ and $H = 15$ for $\epsilon < 0.8$. The only difference lies in the variances over the different seeds. Based on these results and the required additional computation resources for $H = 15$, we decided to go for $H = 10$.

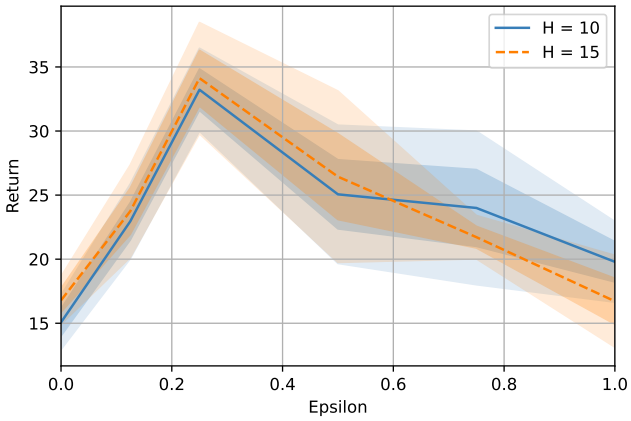


Figure B.2: The performance of the PEBL DDQN method depending on the number of ensemble heads hyperparameter. The measurements are obtained by taking the average return of the final policy over 250 episodes averaged over 3 random seeds. There is no statistical significant difference between $H = 10$ and $H = 15$. However, the method appears to have a lower variance over random seeds with $H = 10$.

B.3. Assessing the effect of automatic hyper-parameter tuning on PEBL DDQN

This experiment assesses if the PEBL DDQN method also benefits from the automatic tuning procedure for hyper-parameter C_π. This automatic tuning procedure was originally designed for the PEBL Soft Actor Critic (SAC) method, but it could also benefit the DDQN version. We test this hypothesis by comparing the results of the PEBL DDQN method with automatic tuning procedure enabled and disabled on the ϵ-*greedy* data set. In the disabled trial, we set $C_\pi = 1$. In the enabled trial, we set $C_y = 1$ and let the automatic tuning procedure find a value for C_π.

Results The outcomes of the experiment are shown in Figure B.3. The results show that with the automatic tuning procedure enabled, the method learns a policy that performs slightly worse. Interestingly, it also appears that the automatic tuning procedure reduces the variance in the results. However, we cannot confirm this result with certainty because most of the results are within one standard deviation of each other for these three independent trials. Therefore, we will not use the automatic tuning procedure in our PEBL DDQN method. Note that these results do **not** mean that the automatic tuning procedure does not work. It only means that the procedure appears not to be beneficial for the PEBL DDQN method. In the experiments for the PEBL SAC method, we show that this procedure is essential.

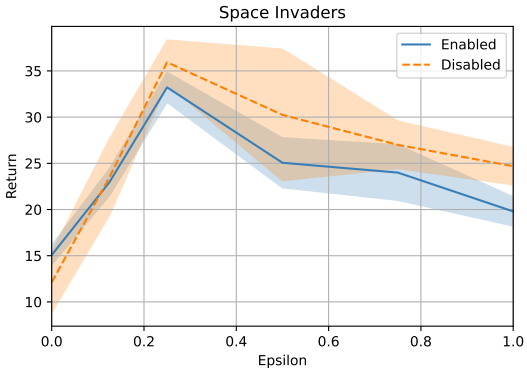


Figure B.3: The performance of the PEBL DDQN method when the automatic tuning of C_π is both enabled and disabled. The measurements are obtained by taking the average return of the final policy over 250 episodes averaged over 3 random seeds. The method appears to perform slightly better when the tuning procedure is disabled, but the difference is not statistically significant.

C

Published paper

PEBL: Pessimistic Ensembles for Offline Deep Reinforcement Learning

Jordi Smit, Canmanie T. Ponnambalam, Matthijs T. J. Spaan, Frans A. Oliehoek
Delft University of Technology
j.smit-6@student.tudelft.nl, {c.t.ponnambalam, m.t.j.spaan, f.a.oliehoek}@tudelft.nl

Abstract

Offline reinforcement learning (RL), or learning from a fixed data set, is an attractive alternative to online RL. Offline RL promises to address the cost and safety implications of taking numerous random or bad actions online, a crucial aspect of traditional RL that makes it difficult to apply in real-world problems. However, when RL is naïvely applied to a fixed data set, the resulting policy may exhibit poor performance in the real environment. This happens due to over-estimation of the value of state-action pairs not sufficiently covered by the data set. A promising way to avoid this is by applying pessimism and acting according to a lower bound estimate on the value. In deep reinforcement learning, however, uncertainty estimation is highly non-trivial and development of effective uncertainty-based pessimistic algorithms remains an open question. This paper introduces two novel offline deep RL methods built on Double Deep Q-Learning and Soft Actor-Critic. We show how a multi-headed bootstrap approach to uncertainty estimation is used to calculate an effective pessimistic value penalty. Our approach is applied to benchmark offline deep RL domains, where we demonstrate that our methods can often beat the current state-of-the-art.

Introduction

Offline (batch) reinforcement learning (RL), addresses some of the key problems that make general reinforcement learning unsuitable for real-world applications such as robotics (Cabi et al. 2020), healthcare (Wang et al. 2018), recommender systems (Strehl et al. 2010), and chatbots (Pietquin et al. 2011). In offline RL, a particular core issue of standard RL is absent: the exploration-exploitation trade-off. Instead, a data set is available that contains transitions sampled from the environment that ideally provide useful information about its dynamics and the task we want to optimize. This suggests that we can find a policy that optimizes the task, given only the data set, and avoid taking additional exploratory actions online whose consequences are unknown. In safety-critical applications or anywhere ample offline data is available, offline RL is an attractive approach to automation. However, naïvely applying policy optimization to a fixed data set has been repeatedly shown in practice to produce a policy that performs very poorly on the true task (Levine et al. 2020; Fujimoto et al. 2019; Fu et al. 2020; Kim 2020; Hou et al. 2020). The fundamental issue arises in the likely case that the data has insufficient information to allow the transition and reward model to be adequately estimated, either explicitly or implicitly, in the model-free case. This issue arises from the strong bias introduced by the policies that generated the data set, rendering the data unrepresentative of the true MDP.

A theoretical analysis of the offline RL problem suggests that policies learned on fixed data will become over-optimistic, meaning they assign values to actions that are higher than the true value in areas with insufficient information (Buckman, Gelada, and Bellemare 2020). If the offline data is collected by selecting actions at random, it is more likely that the coverage of the problem space is sufficient to recover the transition and reward functions. In this case, over-estimation is less likely, but significantly more data is needed to uncover optimal behavior. In the other extreme, where data is collected according to expert demonstrations, the data is highly biased and will contain transitions from only a subset of the overall state-action space. The value attributed to parts of the state-action space that were not represented enough in the data set is likely to not only be incorrect but over-estimated. This effect occurs because over-estimations have a larger impact on the performance of the offline RL agent in the true environment (Buckman, Gelada, and Bellemare 2020). One way to bypass this issue is to constrain the learned policy to lie near the policy exhibited in the data. This has been shown to be effective (Levine et al. 2020; Fujimoto et al. 2019; Kim 2020; Fujimoto, Meger, and Precup 2019), but such an approach is limited in its ability to improve on the policy emergent in the data set. The ideal approach should be robust to many data collection strategies, ranging from random actions to expert demonstrations.

An alternative to penalizing actions that are under-represented in the data set is to penalize actions whose estimated value is highly uncertain. The penalty allows policies to deviate from the empirical policy in high information regions while closely imitating the empirical policy in low information regions. This principle, referred to as uncertainty-based pessimism, is a promising approach to producing offline RL algorithms that are robust to various data collection policies (Jin, Yang, and Wang 2020; Rashidinejad et al. 2021). The effectiveness of such an ap-

proach has been demonstrated in tabular problems, where a principled approach can be taken to uncertainty estimation (Buckman, Gelada, and Bellemare 2020). We aim to extend these findings to deep RL, where the continuous nature of the state-action space and use of function approximation means that count-based methods can no longer be used for uncertainty estimation. We propose a multi-headed bootstrap approach with randomized priors to estimating epistemic uncertainty in deep learning settings (Osband, Aslanides, and Cassirer 2018). With the inclusion of randomized priors, every ensemble member learns a different function in regions with no training data, resulting in better epistemic uncertainty estimation. With this, we derive a pessimistically-penalized offline deep reinforcement learning approach, which we call PEssimistic ensemBLe, or PEBL (pronounced "pebble"). This paper introduces two versions of PEBL, one built on Double Deep Q-Learning (Van Hasselt, Guez, and Silver 2016) and another variant based on Soft Actor-Critic (Haarnoja et al. 2018). Our implementation of these techniques is available in an open-source repository at: github.com/j0rd1smit/PEBL.

This paper first highlights related literature and describes how our work builds upon recent advances in offline deep RL. We then discuss uncertainty estimation in deep RL and the motivations for the approach used in our method. This is followed by the description of our two PEBL variants for offline deep RL, tackling both discrete and continuous action spaces. In experiments, we demonstrate that our uncertainty-based pessimistic methods achieve state-of-the-art performance, often improving on existing benchmarks. Finally, we conclude and suggest directions for future work in this area.

Background

In this section, we formalize established concepts that provide a foundation for our work.

Reinforcement Learning

In reinforcement learning, the environment or task is modeled as a Markov decision process (MDP). Formally, an MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$ containing state space $\mathcal{S}$, action space $\mathcal{A}$, reward function $\mathcal{R}$, transition function $\mathcal{T}$, and a discount factor γ. An agent acting in an MDP at time step t receives a state s_t and then executes an action a_t which causes the environment to transition according to $\mathcal{T}(s_t, a_t, s_{t+1})$, a function that maps state-action pairs to a distribution over next states $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. The agent enters the next state s_{t+1}, and receives a reward r_t according to reward function $\mathcal{R}(s_t, a_t, s_{t+1})$, where $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$. The solution to an MDP is a policy π that maps states to actions. The value of a policy $V^{\pi}_{s_0=s_i}$ is the sum of expected discounted rewards when following policy π from initial state s_i. An optimal policy π^* maximizes this value for every state. The action-conditioned value is referred to as $Q(s, a)$, which returns the value for a particular action in a given state.

In reinforcement learning, the transition and reward dynamics of the MDP are unknown. An agent must therefore interact with the environment in order to learn to optimize rewards. In Q-learning, a common model-free approach for tabular problems, this is done by continuously applying the Bellman update on experience sampled from the MDP (s, a, r, s') (collected by the agent) until Q converges with learning rate α (Watkins and Dayan 1992):

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a')) + (1 - \alpha)Q(s, a)$$

Deep Reinforcement Learning

In continuous state-space environments, the Q-function can no longer be represented in a table, and function approximation is required. In deep reinforcement learning, a multi-layered neural network is used to approximate Q. The most basic method applying this is Deep Q-Learning (DQN) (Mnih et al. 2013, 2015), where the TD-target y_t is calculated by the neural network (with parameters θ) is:

$$y_t = r_t + \gamma \max_{a'} Q(s', a' : \theta)$$

Double Deep Q-Learning (DDQN) Naïve applications of DQN has been known to exhibit instability due to over-estimation, thus DDQN (which originated in tabular settings) was posed as a solution to this problem (Van Hasselt, Guez, and Silver 2016). In DDQN, two Q-functions are maintained, one of which is used for action selection (parameterized by θ) and the other for evaluation (parameterized by θ'). The original DDQN algorithm calculates the target using the formula:

$$y_t = r_t + (1 - d_t) \cdot \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\{Q(s_{t+1}, a; \theta)\}; \theta')$$

In this equation, r_t is the reward, and d_t is a flag indicating whether the episode terminated or not at the t-th time step.

Soft Actor-Critic (SAC) Both DQN and DDQN are suitable for discrete-action problems. Soft Actor-Critic is an algorithm that can also be applied to problems with continuous action spaces. It builds on the popular Actor-Critic approach, which splits learning the policy from the value function. In this approach, the actor learns an action distribution that maximizes the expected reward based on the feedback from the critic, while the critic learns to approximates the value function (Sutton and Barto 2018). Actor-Critic approaches are typically used in on-policy settings. However, using importance sampling, it is also possible to use this approach in an off-policy setting (Haarnoja et al. 2018). In SAC, the goal is to optimize for a trade-off of expected return and entropy of the policy. The TD-target y_t is calculated using the formula:

$$y_t = r_t + (1 - d_t) \cdot \gamma \min_{j=1,2} \{Q(s_{t+1}, \tilde{a}_{t+1}; \theta'_j)\} - \alpha \log \pi_\phi(\tilde{a}_{t+1} | s_{t+1})$$

In this equation, $\tilde{a}_{t+1}$ is sampled from the learned policy $\pi_\phi(.|s_{t+1})$, and α is the entropy trade-off parameter. Similar to DDQN, SAC tries to minimize instability issues due to over-estimations caused by the function approximation. It does this by learning two Q-value functions, parameterized by θ_1 and θ_2. It uses the lowerest Q-value estimation for its policy losses, and it is TD-target to prevent over-estimations.

Uncertainty-Aware Offline Reinforcement Learning

Buckman, Gelada, and Bellemare describe two types of offline reinforcement learning algorithms that apply pessimism: proximal (which we call policy-constrained) pessimistic algorithms and uncertainty-aware pessimistic algorithms (2020). In their experiments, the performance of four algorithmic families was demonstrated: naïve (simply applies standard reinforcement learning on the given data set), behavior cloning (copies the empirical policy followed in the data set), policy-constrained pessimistic (penalizes actions not well-represented in the data set), and uncertainty-aware pessimistic (penalizes actions with high uncertainty in the data set). A behavior cloning-based policy can only perform as well as the data collection policy. The policy-constrained pessimistic family can improve upon the data collection policy, but only slightly, as these algorithms are constrained to stay close to it. The naïve algorithmic family performs well if the data set contains sufficient exploratory data, such as in a huge randomly generated data set. However, its performance quickly deteriorates when the data set is more biased (such as that generated by expert demonstrations). Theoretically, and then in tabular experiments, it was shown that the uncertainty-aware pessimistic approach was the most robust to different types of data sets.

Related work

Uncertainty-aware approaches to reinforcement learning have been used in the traditional online setting, notably in the robust MDP framework which aims to build policies that are robust to modeling errors (Zhou et al. 1996; Givan, Leach, and Dean 2000; Nilim and El Ghaoui 2005; Wiesemann, Kuhn, and Rustem 2013). Recently, the theoretical advantages of uncertainty-aware algorithms in the offline setting have been demonstrated in tabular experiments (Buckman, Gelada, and Bellemare 2020; Rashidinejad et al. 2021) and linear function approximation experiments (Jin, Yang, and Wang 2020). We are concerned with the application of uncertainty-aware pessimism in offline deep reinforcement learning, where quantifying uncertainty is particularly difficult (Buckman, Gelada, and Bellemare 2020; Abdar et al. 2020). Recently, deep learning approaches for offline reinforcement learning have received considerable attention. Several of these advances can be viewed as members of the policy-constrained pessimistic family (Kim 2020; Kumar et al. 2020; Hou et al. 2020), where the primary differences between approaches are the choice of constraint (Buckman, Gelada, and Bellemare 2020). For example, both Batch Constrained Q-Learning (Fujimoto et al. 2019) and Pessimistic Offline Policy Optimization (POPO) (Hou et al. 2020) can be viewed as pessimistic algorithms because they are pessimistic with respect to Q-values for state-action pairs not covered by the behavior policy. However, because this pessimism is based on the observed behavior policy, they belong to the policy-constrained pessimistic family, whereas we are interested in the uncertainty-based pessimistic family and its theoretical advantages over the alternative.

Like our method, many other offline deep RL algorithms also use an ensemble of Q-functions to prevent overestimating the Bellman backup. For example, the methods BEAR (Kim 2020), POPO (Hou et al. 2020), and BRAC (Wu, Tucker, and Nachum 2019) use ensembles to do this by picking either the Q-function with the lowest value or they pick a weighted average of the current highest and lowest Q-value in the ensemble. This technique reduces the over-estimation propagation caused by the target network and the max operator in the Bellman equation. This works well in online RL (Haarnoja et al. 2018; Fujimoto, Hoof, and Meger 2018), but in offline RL, insufficient information and insufficient coverage in the data set also causes over-estimations which are not addressed by this technique (Rashidinejad et al. 2021; Jin, Yang, and Wang 2020). Methods like BEAR, POPO, and BRAC have to resort to policy-constrained techniques such as the maximum mean discrepancy, KL divergence, and Wasserstein Distance to counteract this secondary source of over-estimations (Kim 2020; Hou et al. 2020; Wu, Tucker, and Nachum 2019). In contrast, our method addresses the over-estimations caused by insufficient information by penalizing the Q-values based on the epistemic uncertainty.

Another interesting approach is Conservative Q-learning (Kumar et al. 2020) and its model-based version Conservative Offline Model-Based Policy Optimization (COMBO) (Yu et al. 2021). These methods regularize the Q-values by simultaneously minimizing all the Q-values and maximizing the Q-values in the data set, finding a lower bound on the Q-value. While it appears that this algorithm aims to avoid low-information regions similar to the algorithms in the uncertainty-aware pessimistic family, this is not the case because it uses data concentration as a proxy for information. In reality, it belongs to the policy-constrained pessimistic algorithmic family because it aims to stay close to the empirical policy, especially in very noisy or small data sets. This constraint is much looser than the previously mentioned policy constraints in practice, allowing for larger improvements. In contrast, our uncertainty-aware pessimistic algorithm is not constrained by any empirical policy, which theoretically allows for even larger improvements upon the empirical policy in a wider range of data sets (Rashidinejad et al. 2021; Jin, Yang, and Wang 2020).

Currently, Model-based Offline Policy Optimization (MOPO) (Yu et al. 2020) and Model-Based Offline Reinforcement Learning (MOReL) (Kidambi et al. 2020) are the most similar to our model-free algorithm. These model-based methods use an ensemble of models to estimate the uncertainty in the MDP. This uncertainty estimate is used to construct a pessimistic version of the MDP by subtracting the state uncertainty from the modeled rewards. The resulting pessimistic MDP is then used as input to a model-free method to learn a pessimistic policy. The major disadvantage of these methods is that they do not learn a pessimistic policy using an end-to-end based technique, as our method does. This is potentially an issue because when a Bellman update is performed on a function approximated Q-network, the value of a particular state is impacted by generalizations from other states, which potentially confounds the pessimistic penalties (Buckman, Gelada, and Bellemare

2020). We conclude that there is a gap in the offline deep RL literature for an end-to-end model-free uncertainty-aware pessimistic algorithm, which we aim to address using our method PEBL.

Uncertainty-Based Pessimism in Deep Reinforcement Learning

The uncertainty in reinforcement learning can be decomposed into two types of uncertainty, aleatoric and epistemic (Abdar et al. 2020; Kendall and Gal 2017). *Aleatoric* uncertainty arises from the stochasticity that is naturally present in the observations. Its key property is that it cannot be reduced by adding more data. A typical example of aleatoric uncertainty is a random or noisy reward function; adding more data from this function will *not* remove the noise in the observations. *Epistemic* uncertainty describes what the model does not know due to limitations in the observed data. This type of uncertainty has the key property that it *can* be reduced by adding more data. In this work, we are mainly concerned with epistemic uncertainty.

We seek to apply uncertainty-based pessimism to address two key challenges emergent in offline deep reinforcement learning. The first issue arises when a naïve algorithm, such as an online version of DDQN or SAC, is applied to an offline data set. In this scenario, a wide range of literature has observed that the Q-values can increase continuously until the algorithm diverges (Levine et al. 2020; Fujimoto et al. 2019; Kim 2020). This effect is most apparent in data sets with low state-action space coverage, such as small data sets and expertly generated data sets. The issue is caused by overoptimistic value estimations of the next state. These overestimations can be small initially, but they compound over time due to the Bellman equation, resulting in divergent behavior. One way to apply pessimism to counter this compounding effect is by penalizing the estimated Q-value in the Bellman backup of the target network by a factor of the uncertainty (estimated by the standard deviation) (Levine et al. 2020):

$$y_t = \overline{\mu}_Q(s_{t+1}, a; \theta') - \overline{\sigma}_Q(s_{t+1}, a; \theta') \tag{1}$$

In this equation, $\overline{\mu}_Q$ is the mean, $\overline{\sigma}_Q$ is the standard deviation of the Q-value approximations, and θ' is the weights of the target network.

Another issue that must be addressed is the trade-off between minimizing uncertainty and maximizing the Q-values in the learned policy. Uncertainty-aware offline RL algorithms do this by learning a policy that maximizes the uncertainty lower bound of the Q-values:

$$\pi_p = \underset{a}{\operatorname{argmax}}\{\overline{\mu}_Q(s_t, a; \theta) - \overline{\sigma}_Q(s_t, a; \theta)\} \tag{2}$$

In the remainder of this section, we will discuss how this estimate $\overline{\sigma}_Q$ can be obtained.

Estimating Uncertainty in Deep Reinforcement Learning

We seek a penalty that represents the epistemic uncertainty, not the aleatoric uncertainty, which means it should be inversely proportional to the density in the data set as well as our confidence. The deep learning literature proposes multiple ways to estimate the uncertainty in the predictions of a neural network (Osband et al. 2016; Osband, Aslanides, and Cassirer 2018; Gal and Ghahramani 2016; Blundell et al. 2015; Kendall and Gal 2017; Liu et al. 2020). One of the best-known uncertainty estimation techniques methods is Monte Carlo dropout (Gal and Ghahramani 2016). This method measures the uncertainty as the sample standard deviation over N Monte Carlo samples of slightly differing network configurations. Dropout has been shown to successfully capture uncertainty in several applications (Wang et al. 2019; Nair et al. 2020; Do et al. 2020). However, it mainly focuses on aleatoric uncertainty because its Bayesian posterior does not concentrate on the observed data and it cannot propagate its uncertainty through the Bellman fixed point (Osband, Aslanides, and Cassirer 2018). These properties make Monte Carlo dropout an unsuitable candidate for our problem.

Another well-known uncertainty estimation technique is the ensemble (Pearce, Leibfried, and Brintrup 2020; Osband, Aslanides, and Cassirer 2018; Lakshminarayanan, Pritzel, and Blundell 2017). An ensemble measures the uncertainty as the sample standard deviation between the prediction of its members. Ensembles are very good at estimating epistemic uncertainty as long as every member learns a different function in low data density areas. Another advantage of the ensemble is that its uncertainty measure is function-dependent and can propagate uncertainties through the Bellman fixed-point by definition (Osband, Aslanides, and Cassirer 2018).

Our method uses the multi-headed bootstrap ensemble with random priors to ensure that the ensemble members are as diverse as possible (Osband, Aslanides, and Cassirer 2018). This approach is similar to the traditional deep learning ensemble, but it improves upon it in two ways. First, it adds a different parallel prior to the prediction of each head (Figure 1). This prior is a randomly initialized but frozen network, meaning its weights will not change during the training process. Due to this strategy, each head has an input-dependent prior, while the mapping remains constant throughout the training process. Thus, in high data concentration areas, each head learns to ignore its prior and approaches a similar function, while in the low data concentration areas, each head is biased by its prior and there are more significant disagreements in the learned function between heads. This results in better epistemic uncertainty estimation for out-of-distribution data. The second improvement is the addition of the bootstrap. By training each ensemble member on a slightly different subset of the data, we ensure that it learns a different function where data is sparse, which results in improved epistemic uncertainty estimation at the edges of dense data concentration.

There is one notable disadvantage to our proposed uncertainty estimation technique: the number of weights and compute of this method increases linearly with the number of heads used. However, it is possible to share a feature extractor such as a convolutional neural network (CNN). In theory, this shared encoder can reduce the diversity in the ensemble, but it has been observed empirically that this negative ef-

fect is minimal, making this a valid computational trade-off (Osband et al. 2016; Osband, Aslanides, and Cassirer 2018; Sedlmeier et al. 2019). In practice, the parallel heads and their priors contain only one or two fully connected layers making the memory and performance requirements manageable.

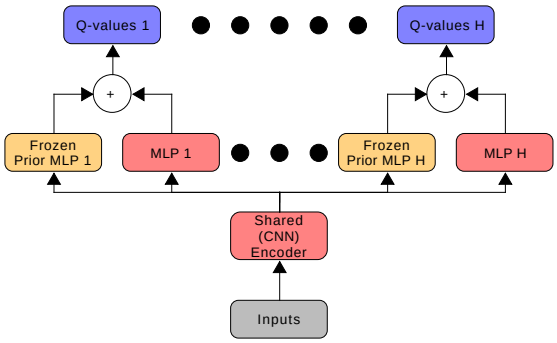


Figure 1: The multi-headed bootstrap ensemble with random priors network architecture. The architecture is similar to traditional ensembles but adds the output of a frozen prior network to each head's output to create a state depended prior.

Pessimistic Ensembles for Offline Deep Reinforcement Learning

We propose two new PEssimistic ensemBLe (PEBL) algorithms that use a multi-headed bootstrap with priors architecture to approximate the Q-function. The first algorithm is a pessimistic version of DDQN (Van Hasselt, Guez, and Silver 2016) aimed at problems with discrete action spaces. The use of two Q-networks helps avoid over-estimations in the function approximation, offering more stable performance. The second algorithm is a pessimistic version of SAC (Haarnoja et al. 2018) and works in both continuous and discrete action-space cases. Although SAC can also be applied to discrete action spaces, DDQN is often preferred in this setting due to its lower memory and computation costs.

Pessimistic Ensemble DDQN

Our pessimistic version of DDQN, which we call PEBL DDQN changes the TD-target of DDQN to:

$$y_t = r_t + (1 - d_t) * \gamma \overline{\mu}_Q(s_t, a^*; \theta') - \overline{\sigma}_Q(s_t, a^*; \theta') \quad (3)$$

$$a^* = \underset{a}{\text{argmax}}\{\overline{\mu}_Q(s_t, a; \theta)\overline{\sigma}_Q(s_t, a^*; \theta)\}$$

In this equation, $\overline{\mu}_Q(s_t, a; \theta)$ and $\overline{\sigma}_Q(s_t, a; \theta)$ are the sample mean and sample standard deviation over the different heads:

$$\overline{\mu}_Q(s_t, a; \theta) = \frac{1}{h}\sum_{i=0}^{h-1} Q_i(s_t, a; \theta_i) \quad (4)$$

$$\overline{\sigma}_Q(s_t, a; \theta) = \sqrt{\frac{\sum_{i=0}^{h-1}(Q_i(s_t, a; \theta_i) - \overline{\mu}_Q(s_t, a; \theta_i))^2}{h-1}} \quad (5)$$

In this equation, Q_i is the Q-value prediction of the i-th ensemble head. This definition is different from the original definition from the bootstrapped DQN (Osband et al. 2016; Osband, Aslanides, and Cassirer 2018), which trained each head on its own target head. This is because we need to subtract an uncertainty penalty; if the penalty was subtracted from each head directly, an unstable feedback loop is created, resulting in potentially very large negative Q-values. Finally, we add the bootstrapping masks as described in the efficient implementation of bootstrapped DQN (Sedlmeier et al. 2019), giving the following definition for the TD-error for head i:

$$TD_i = m_{i,j} \cdot (Q_i(s_t, a; \theta_i) - y_t)$$

In this equation, $m_{i,j}$ is a boolean mask that has been sampled for each training sample j from a Bernoulli distribution with $p = 0.8$ as suggested by (Pearce, Leibfried, and Brintrup 2020). Note that the $m_{i,j}$ remains constant through the entire training process. The final policy selects the action with the highest pessimistic q-value.

$$\pi_t = \underset{a}{\text{argmax}}\{\overline{\mu}_Q(s_t, a; \theta) - C_\pi \cdot \overline{\sigma}_Q(s_t, a; \theta)\}$$

Pessimistic Ensemble SAC

Our pessimistic version of Soft Actor-Critic, called PEBL SAC, changes the target to:

$$y_t = r_t + (1 - d_t) \cdot \gamma \cdot (\overline{\mu}_Q(s, a; \theta_1, \theta_2) - \overline{\sigma}_Q(s, a; \theta_1, \theta_2)) - \alpha \log \pi_\phi(\tilde{a}_{t+1}|s_{t+1}) \quad (6)$$

where $\tilde{a}_{t+1}$ is sampled from the learned policy $\pi_\phi(.|s_{t+1})$. We define $\overline{\mu}_Q(s, a; \theta_1, \theta_2)$ and $\overline{\sigma}_Q(s, a; \theta_1, \theta_2)$ as:

$$\overline{\mu}_Q(s, a; \theta_1, \theta_2) = \frac{1}{h}\sum_{i=0}^{h-1} \min_{j=1,2} Q(s, a; \theta_{j,i}) \quad (7)$$

$$\overline{\sigma}_Q(s, a; \theta_1, \theta_2) = \sqrt{\frac{1}{h-1}\sum_{i=0}^{h-1}(\overline{\mu}_Q(s, a; \theta_1, \theta_2) - \min_{j=1,2} Q(s, a; \theta_{j,i}))^2} \quad (8)$$

Using these targets, we can calculate the TD-errors of the PEBL SAC algorithm in the same way we calculated the TD-errors for the PEBL DDQN algorithm with head i and mask m_i:

$$TD_{i,j} = m_i \cdot (Q_i(s_t, a; \theta_i) - y_t)$$

The original SAC algorithm calculates the policy loss using the formula:

$$\mathcal{L} = \frac{1}{|B|}\sum_{s \in B}(\alpha \log \pi_\phi(\tilde{a}_t, s_t) - \min_{i=1,2}\{Q(s_t, \tilde{a}_t; \theta_i)\})$$

Our pessimistic version of this algorithm changes the loss policy function to:

$$\mathcal{L} = \frac{1}{|B|} \sum_{s \in B} (\alpha \log \pi_\phi(\tilde{a}_t, s_t) - (\overline{\mu}_Q(s, a; \theta_1, \theta_2) - C_\pi \cdot \overline{\sigma}_Q(s, a; \theta_1, \theta_2)) \quad (9)$$

In this equation, we also introduce the uncertainty weight trade-off parameter C_π. This parameter controls the trade-off between minimizing uncertainty and maximizing the Q-values in the learned policy. This trade-off parameter is needed because the policy loss in SAC is similar to a white box adversarial attack on the Q-function due to the re-parameterization trick (Akhtar and Mian 2018). This adversarial formulation of the policy loss is not a problem in deep online RL because it forces the agent to learn about the flaws in its Q-function, which helps with exploration. However, in offline RL, this adversarial formulation is a problem because the agent can no longer collect counterexamples in the environment. Therefore, selecting the right parameter for C_π is crucial. Empirically, we found that it can be difficult to find the right value for C_π because it depends on many factors such as the size of the data set, the state-action space coverage, or the difficulty of modeling the true MDP. However, we can tune this parameter online using dual gradient descent, a technique that is increasingly common in deep reinforcement learning (Haarnoja et al. 2018; Kumar et al. 2020). We apply dual gradient descent and transform C_π into a Lagrangian multiplier by adding the constraint that the average uncertainty of the actions selected by the learned pessimistic policy π should be equal to the average uncertainty observed in the actions for the data set:

$$\frac{1}{n} \sum_{s_t, a_t}^{\mathcal{D}} \overline{\sigma}_Q(s_t, a_t; \theta) = \frac{1}{n} \sum_{s_t}^{\mathcal{D}} \overline{\sigma}_Q(s_t, a_p; \theta), a_p \in \pi_p \quad (10)$$

This constraint is possible because we are only interested in avoiding epistemic uncertainty, which is high in areas where the network cannot model the function well. Thus, we allow the selection of out-of-distribution actions as long as we avoid areas with above-average epistemic uncertainty. The constraint in Equation 10 captures this property, which results in a higher value of C_π if π_p chooses actions with above-average epistemic uncertainty. It results in a lower value for C_π if π_p chooses actions with below-average epistemic uncertainty.

Experimental Results

In this section, we discuss the results of several empirical evaluations of our PEBL methods. The main purpose of our experiments is to demonstrate that uncertainty-based pessimistic algorithms can be applied to offline deep RL problems and achieve high performance and that they can do well for a wide range of provided data distributions. In all experiments, the trade-off parameter C_π in the SAC version of our algorithm is learned using Lagrangian dual gradient descent (Equation 10). For an open-source implementation, please refer to the code in the accompanying GitHub repository: github.com/j0rd1smit/PEBL.

MinAtar

We first aimed to reproduce the results presented by Buckman, Gelada, and Bellemare in their paper (2020). In continuous experiments, they compared only three algorithmic families, excluding the uncertainty-aware pessimistic and leaving it as an open research question. We use their same experimental set-up and additionally assess the performance of our uncertainty-aware pessimistic algorithm PEBL.

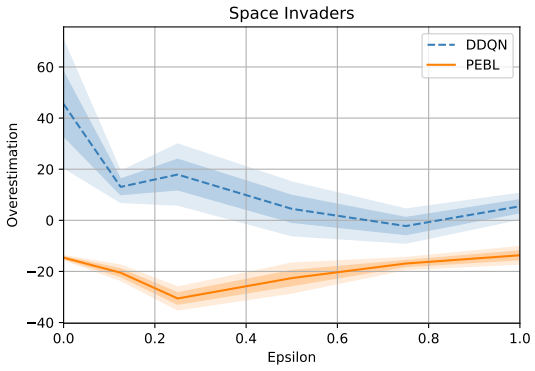
The two environments used in this experiment are from MinAtar, which contains several miniaturized versions of Atari 2600 games (Young and Tian 2019). As is common in offline deep RL experiments, we first train an online agent to obtain an optimal policy for each of the environments (Buckman, Gelada, and Bellemare 2020; Fujimoto et al. 2019; Fu et al. 2020; Agarwal, Schuurmans, and Norouzi 2020). We then use this policy to generate several data sets with different expert/random ratios by modifying epsilon in an epsilon-greedy strategy. The generated data sets used in this experiment contain 50000 transitions. We chose this slightly smaller data set size because it highlights the advantages of the uncertainty-based method. Using this experimental setup, we evaluate our method in two ways. Firstly, we validate that our method does not overestimate the expected discounted return. We do this by comparing the return our method obtains in the real environment with its prediction for the discounted return at the start of the episode. Secondly, we evaluate how well our method adheres to the theoretically predicted property of uncertainty-aware algorithms by measuring the final performance of the learned policy for varying values of epsilon.

Baselines We compare PEBL to methods representing the aforementioned algorithmic families. The first is the behavior clone, referred to as BC. The policy-based pessimistic family is represented by Batch-Constrained Q-Learning (BCQ) (Fujimoto, Meger, and Precup 2019). The third category is the naïve algorithmic family, represented by DDQN (Van Hasselt, Guez, and Silver 2016).

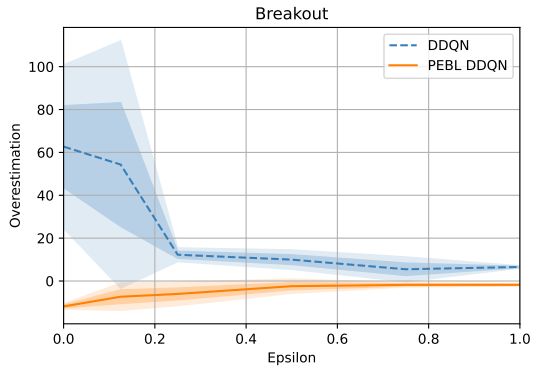
Results The outcomes of our MinAtar experiments are pictured in Figures 2 and 3. The results of Figure 2 show that the value function learned by our PEBL DDQN method does not overestimate the expected discounted return for any epsilon value in both the Space Invaders and Breakout environment. In contrast, the baseline DDQN method overestimates the expected discounted return for every epsilon value, and this overestimation gets worse when the amount of information in the data decreases.

In Figure 3, we see that PEBL DDQN performs best for a wider range of epsilon-greedy policy data sets than the other algorithmic families, in line with the theoretical results we aimed to replicate (Buckman, Gelada, and Bellemare 2020). However, our method does not follow the theoretically predicted behavior for uncertainty-aware pessimistic algorithms in the expert data regime. We expected to perform as well as or similarly to the behavior cloning method but fall notably short. This suggests that our epistemic uncertainty estimation techniques are not expressive enough for highly biased data. This result is interesting because Figure

2 shows that our method does not overestimate the expected discounted return in these highly biased expert data sets.



(a) Space invaders



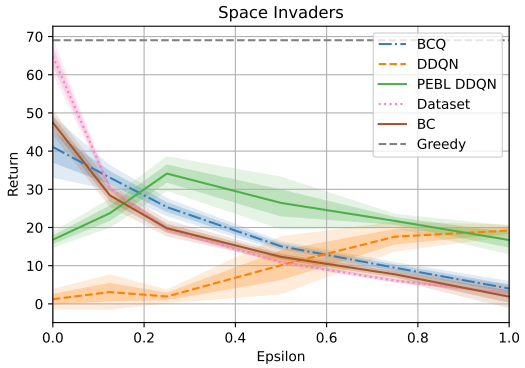
(b) Breakout

Figure 2: The over-estimations in our MinAtar experiment. We define the overestimation as the obtained return minus the predicted Q-value in the initial state. Our PEBL DDQN method does not over-estimate the expected discounted return for any ϵ in both the Space Invaders and Breakout environment. In contrast, the naive DDQN method overestimates for every ϵ.

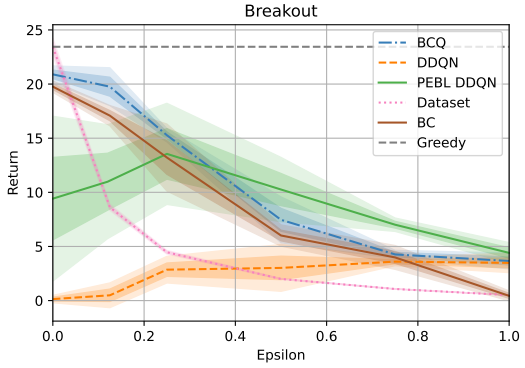
Maze-2D: Uncertainty Visualization

We applied PEBL SAC to the Maze-2D environment (Figure 4a) from the D4RL benchmark (Fu et al. 2020). A motivation for using this 2-dimensional domain is that we can visualize the influence of pessimism on what our algorithm learns. In this task, the agent is shown a data set with unrelated paths through the maze containing only x and y positions and velocities. The agent has to stitch these transitions together to find a path to a goal location in the maze. The challenge of this environment is that the agent never observes the optimal path and has to learn the maze's layout through the data set.

Baselines We compare our method to the results provided in the benchmark paper for SAC, Behavior Cloning (BC), and Continuous Batch-Constrained Q-Learning (CBCQ) (Fujimoto, Hoof, and Meger 2018; Fu et al. 2020).



(a) Space invaders



(b) Breakout

Figure 3: The performance of different representatives of each offline RL algorithmic families compared to our algorithm. Measurements are obtained by taking the average performance of the final policy over 100 episodes averaged over 3 random seeds. Each random seed has its own data set containing 50000 transitions sampled with an ϵ-greedy policy, where the greedy policy is an expert DDQN agent.

Results The performance of the algorithms in the Maze-2D experiment are displayed in Table 1. PEBL SAC is able to deviate further from the observed behavior policy than the other baselines. This allows it to solve all mazes, including the large maze, which has not yet been solved by any existing policy-based pessimistic method (Fu et al. 2020). Figure 4b visualizes the value of the uncertainty-based penalty and the penalized Q-values. We see that the result of our pessimistic penalty is that the agent assigns low Q-values to areas which it is uncertain about, such as positions occupied by walls.

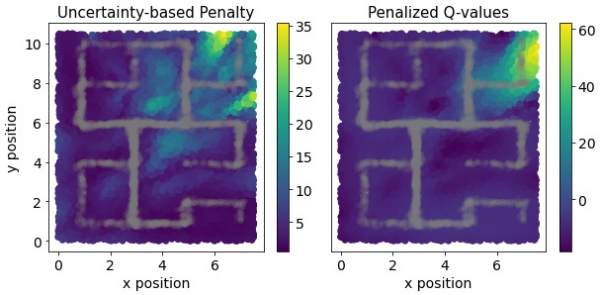
D4RL: MuJoCo gym

In this group of experiments, we compare PEBL SAC to some of the more difficult continuous-action tasks in the D4RL benchmark.

Maze type	SAC	BC	CBCQ	PEBL SAC
U	88.2	3.8	12.8	**151.1**
Medium	26.1	30.3	8.3	**146.6**
Large	-1.9	5.0	6.2	**129.5**

Table 1: Results for D4RL's Maze-2D benchmark. Each score is the average normalized reward over 100 runs at the last iteration of training.

(a) Maze-2D environment



(b) The uncertainty-based penalty and the penalized Q-values per location on the map

Figure 4: A visualization of the learned Q-function and its uncertainty for the large Maze-2D environment (Fu et al. 2020). Note that the agent only observes (x, y, v_x, v_y). Figure b) shows the uncertainty-based penalty $C_\pi \cdot \overline{\sigma}_Q(s, a; \phi_1, \phi_2)$ where C_π was learned and converged to $C_\pi = 12.53$ and the penalized Q-values, where the grey areas are data points provided in the data set.

Baselines Similarly to the first experiment, we compare our methods to representatives of each algorithmic family. The methods are SAC (Haarnoja et al. 2018) and Continuous Batch-Constrained Q-Learning (CBCQ) (Fujimoto, Meger, and Precup 2019) for the naïve and the policy-based pessimistic algorithmic family, respectively. We also compare our method to Conservative Q-learning (CQL) (Kumar et al. 2020) and Conservative Offline Model-Based Policy Optimization (COMBO) (Yu et al. 2021). Both methods are considered state-of-the-art offline RL methods and members of the policy-based pessimistic algorithmic family (Buckman, Gelada, and Bellemare 2020). Compared to other policy-based pessimistic algorithms, these methods have a significantly looser constraint, allowing for larger improvements upon the data collection policies.

Results The results are shown in Table 2, where the performance of SAC, BC, and CBCQ is given as reported in (Fujimoto, Meger, and Precup 2019), and for CQL and COMBO are the number reported in (Kumar et al. 2020) and (Yu et al. 2021), respectively. On the data sets generated using non-expert data policies, marked *Random* and *Medium*, PEBL SAC performs on par or exceeds the best prior methods. Interestingly, it even out-performs a model-based method, while PEBL SAC is a model-free method itself. Similar to the PEBL DDQN algorithm, we are unable to match the performance of behavior cloning in the expert data regime.

Data set type	SAC	BC	CBCQ	CQL	COMBO	PEBL SAC
Random	30.5	2.1	2.2	35.4	**38.8**	35.7
Medium-Replay	1.9	38.4	38.2	44.4	54.2	**57.5**
Medium	-4.3	36.1	40.7	46.2	55.1	**61.5**
Expert	-1.9	**107.0**	-	104.0	-	3.6

Table 2: Results for the D4RL benchmark. Each score is the average normalized reward over 100 runs at the last iteration of training as described in the D4RL benchmark.

Conclusion and Future Work

We introduced the multi-headed bootstrap with randomized priors approach to measuring epistemic uncertainty. Using this ensemble-based method, we penalized the value function to produce two new pessimistic offline RL algorithms, called PEBL DDQN and PEBL SAC. These algorithms are a step towards robust uncertainty-aware pessimistic offline RL algorithms in the deep learning setting. We have shown that our methods are able to perform well on a wide range of data set distributions compared to algorithms from the naïve and policy-based pessimistic families. However, in experiments, we also showed that our methods do not perform well on expert data sets, even though the theory predicts they would. We expect that our epistemic uncertainty estimation techniques are not expressive enough for highly biased data. A more suitable uncertainty measure may address this issue. Thus we hope to continue to investigate appropriate measures of uncertainty for these problems. Some promising methods for future work are Bayesian neural networks (Blundell et al. 2015) and deep Gaussian processes (Liu et al. 2020; van Amersfoort et al. 2021). Both methods are more accurate in their estimation of epistemic uncertainty but are more difficult to optimize in practice. We would also like to examine whether the epistemic uncertainty estimation of our method can be improved using self-supervised representation learning (Stooke et al. 2020; Laskin, Srinivas, and Abbeel 2020; Laskin et al. 2020). This technique has been shown to improve existing offline RL methods in various ways (Sinha and Garg 2021). It may prove beneficial for our method as the improved representations might make out-of-distribution detection easier. In general, we hope that our work inspires further investigation into the development of uncertainty-aware pessimistic algorithms that adhere to the theoretical support of their abilities.

Acknowledgments and Disclosure of Funding

This work received support as part of the research programme Physical Sciences TOP-2 project number 612.001.602, which is financed by the Dutch Research Council (NWO).

References

Abdar, M.; Pourpanah, F.; Hussain, S.; Rezazadegan, D.; Liu, L.; Ghavamzadeh, M.; Fieguth, P.; Khosravi, A.; Acharya, U. R.; Makarenkov, V.; et al. 2020. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *arXiv preprint:2011.06225* .

Agarwal, R.; Schuurmans, D.; and Norouzi, M. 2020. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 104–114. PMLR.

Akhtar, N.; and Mian, A. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access* 6: 14410–14430.

Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; and Wierstra, D. 2015. Weight uncertainty in neural network. In *International Conference on Machine Learning*, 1613–1622. PMLR.

Buckman, J.; Gelada, C.; and Bellemare, M. G. 2020. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint:2009.06799* .

Cabi, S.; Gómez, S.; Novikov, A.; Konyushova, K.; Reed, S.; Jeong, R.; Zołna, K.; Aytar, Y.; Budden, D.; Vecerik, M.; Sushkov, O.; Barker, D.; Scholz, J.; Denil, M.; Freitas, N.; and Wang, Z. 2020. Scaling data-driven robotics with reward sketching and batch reinforcement learning. In *Robotics: Science and Systems*. doi:10.15607/RSS.2020.XVI.076.

Do, H. P.; Guo, Y.; Yoon, A. J.; and Nayak, K. S. 2020. Accuracy, uncertainty, and adaptability of automatic myocardial ASL segmentation using deep CNN. *Magnetic resonance in medicine* 83(5): 1863–1874.

Fu, J.; Kumar, A.; Nachum, O.; Tucker, G.; and Levine, S. 2020. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. *CoRR* abs/2004.07219.

Fujimoto, S.; Conti, E.; Ghavamzadeh, M.; and Pineau, J. 2019. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint:1910.01708* .

Fujimoto, S.; Hoof, H.; and Meger, D. 2018. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 1587–1596. PMLR.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, 2052–2062. PMLR.

Gal, Y.; and Ghahramani, Z. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, 1050–1059. PMLR.

Givan, R.; Leach, S.; and Dean, T. 2000. Bounded-parameter Markov decision processes. *Artificial Intelligence* 122(1-2): 71–109.

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

Hou, Q. H.; et al. 2020. POPO: Pessimistic Offline Policy Optimization. *arXiv preprint:2012.13682* .

Jin, Y.; Yang, Z.; and Wang, Z. 2020. Is Pessimism Provably Efficient for Offline RL? *arXiv preprint:2012.15085* .

Kendall, A.; and Gal, Y. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Kidambi, R.; Rajeswaran, A.; Netrapalli, P.; and Joachims, T. 2020. MOReL: Model-Based Offline Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 21810–21823. Curran Associates, Inc.

Kim, S. 2020. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. Submitted to NeurIPS 2019 Reproducibility Challenge.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *arXiv preprint:2006.04779* .

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Laskin, M.; Lee, K.; Stooke, A.; Pinto, L.; Abbeel, P.; and Srinivas, A. 2020. Reinforcement Learning with Augmented Data. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 19884–19895. Curran Associates, Inc.

Laskin, M.; Srinivas, A.; and Abbeel, P. 2020. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 5639–5650. PMLR.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint:2005.01643* .

Liu, J. Z.; Lin, Z.; Padhy, S.; Tran, D.; Bedrax-Weiss, T.; and Lakshminarayanan, B. 2020. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. *arXiv preprint:2006.10108* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint:1312.5602* .

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533.

Nair, T.; Precup, D.; Arnold, D. L.; and Arbel, T. 2020. Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation. *Medical image analysis* 59: 101557.

Nilim, A.; and El Ghaoui, L. 2005. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research* 53(5): 780–798.

Osband, I.; Aslanides, J.; and Cassirer, A. 2018. Randomized prior functions for deep reinforcement learning. *arXiv preprint:1806.03335* .

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep Exploration via Bootstrapped DQN. In Lee, D.; Sugiyama, M.; Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Pearce, T.; Leibfried, F.; and Brintrup, A. 2020. Uncertainty in neural networks: Approximately bayesian ensembling. In *International conference on artificial intelligence and statistics*, 234–244. PMLR.

Pietquin, O.; Geist, M.; Chandramohan, S.; and Frezza-Buet, H. 2011. Sample-efficient batch reinforcement learning for dialogue management optimization. *ACM Transactions on Speech and Language Processing (TSLP)* 7(3): 1–21.

Rashidinejad, P.; Zhu, B.; Ma, C.; Jiao, J.; and Russell, S. 2021. Bridging Offline Reinforcement Learning and Imitation Learning: A Tale of Pessimism. *arXiv preprint:2103.12021* .

Sedlmeier, A.; Gabor, T.; Phan, T.; Belzner, L.; and Linnhoff-Popien, C. 2019. Uncertainty-based out-of-distribution classification in deep reinforcement learning. *arXiv preprint:2001.00496* .

Sinha, S.; and Garg, A. 2021. S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning. *arXiv preprint:2103.06326* .

Stooke, A.; Lee, K.; Abbeel, P.; and Laskin, M. 2020. Decoupling representation learning from reinforcement learning. *arXiv preprint:2009.08319* .

Strehl, A.; Langford, J.; Li, L.; and Kakade, S. M. 2010. Learning from Logged Implicit Exploration Data. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

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

van Amersfoort, J.; Smith, L.; Jesson, A.; Key, O.; and Gal, Y. 2021. Improving Deterministic Uncertainty Estimation in Deep Learning for Classification and Regression. *arXiv preprint:2102.11409* .

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Wang, G.; Li, W.; Aertsen, M.; Deprest, J.; Ourselin, S.; and Vercauteren, T. 2019. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing* 338: 34–45.

Wang, L.; Zhang, W.; He, X.; and Zha, H. 2018. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2447–2456.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4): 279–292.

Wiesemann, W.; Kuhn, D.; and Rustem, B. 2013. Robust Markov decision processes. *Mathematics of Operations Research* 38(1): 153–183.

Wu, Y.; Tucker, G.; and Nachum, O. 2019. Behavior regularized offline reinforcement learning. *arXiv preprint:1911.11361* .

Young, K.; and Tian, T. 2019. MinAtar: An Atari-Inspired Testbed for Thorough and Reproducible Reinforcement Learning Experiments. *arXiv preprint:1903.03176* .

Yu, T.; Kumar, A.; Rafailov, R.; Rajeswaran, A.; Levine, S.; and Finn, C. 2021. Combo: Conservative offline model-based policy optimization. *arXiv preprint:2102.08363* .

Yu, T.; Thomas, G.; Yu, L.; Ermon, S.; Zou, J. Y.; Levine, S.; Finn, C.; and Ma, T. 2020. MOPO: Model-based Offline Policy Optimization. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 14129–14142. Curran Associates, Inc.

Zhou, K.; Doyle, J. C.; Glover, K.; et al. 1996. *Robust and optimal control*, volume 40. Prentice hall New Jersey.