Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Model Aligned Offline Reinforcement Learning

Author:
Aditya Goel

Supervisor:
Dr. Edward Johns

Second Marker:
Dr. Antoine Cully

June 20, 2022

Abstract

A significant obstacle facing widespread adoption of reinforcement learning methods is the need for trial-and-error, experience driven learning which can be infeasible when tackling real world problems. Offline reinforcement learning methods aim to decouple data collection from policy learning by learning policies from diverse, static datasets without real world interaction. The development of successful offline reinforcement learning methods could have a transformative effect in areas ranging from robotics to healthcare. Unfortunately, performance of existing offline methods have traditionally lagged their online counterparts.

In this report we identify misalignment between data sampled from the experience replay buffer and the state-action visitation of the agent as being as being a significant cause of performance degradation in offline methods. Through experimental study, we illustrate that this problem is distinct from existing challenges being addressed in the field such as errors introduced by extrapolation and argue that algorithmic innovation is needed to address this.

We systematically address this problem by introducing a novel class of offline reinforcement learning algorithms we call *Model Aligned Offline Reinforcement Learning* (MAORL). Unlike any offline reinforcement learning method that precedes it, MAORL algorithms tune the buffer sampling regime rather than applying constraints on policy improvement. By framing the misalignment problem in a quantitative way, we are able to derive an iterative, gradient-based solution to continuously tune the buffer sampling regime over the course of training.

We find that our approach successfully corrects misalignment and can reach performance levels comparable with online algorithms on a suite of synthetic GridWorld tasks.

Acknowledgements

I would like to thank my supervisor Dr. Edward Johns for the opportunity to work on such a fascinating project. It has been deeply fulfilling work and has rekindled my fascination with machine intelligence and love of research.

I would also like to thank my friends and my girlfriend, Priya, for sharing the ups and downs of the university experience with me over the last four years.

Last but certainly not least, I want to thank my family. You have inspired me to try and be better every day.

The first principle is that you must not fool yourself — and you are the easiest person to fool.
Richard Feynman

Contents

Chapter 1

Introduction

1.1 Motivation

Modern deep reinforcement learning algorithms operate in a fundamentally *online* learning paradigm where trial-and-error based data collection and exploration forms an integral part of the training process. While these methods have been extremely successful in simulated environments and games [1], the need to collect large amounts of data during training for these data intensive algorithms is a major obstacle facing adoption of reinforcement learning methods for real world problems. Trial-and-error learning can be impractical, dangerous or expensive in real world tasks and environments. In areas such as healthcare and autonomous driving, trying out a policy that fails can be extremely dangerous and costly.

It stands to reason that pairing reinforcment learning objectives with large, precollected static datasets can be an effective way to make reinforcement learning an order of magnitude more useful, since this decouples data collection from policy learning. In particular, supervised learning methods have found great success because of their ability to efficiently scale performance with data and compute, both of which are resources that are becoming ever more abundant. Prior work has established that applying existing off-policy reinforcement learning algorithms without exploration does not transfer well [2] [3], suggesting that there are open algorithmic challenges. *Offline* reinforcement learning methods describe the class of algorithms that aim to resolve these deficiencies in order to make data-driven reinforcement learning without exploration a reality.

Development of truly successful offline reinforcement learning algorithms could have a transformative effect. Fundamentally, it would enable us learn decision making engines from large, diverse datasets that are capable of considering future actions as well as present ones. Applications could include the training of robots to perform many different tasks without intermediate interaction with the real world or the creation of new kinds of recommender systems for personalisation on web-based services [4] that take potential future user actions into account when making decisions.

1.2 Objectives

This is a research-driven project. As such, the objectives of the project materialised through our own investigative research where we analysed the existing literature in a first-principles fashion and identified a gap in the traditional analysis, which we validate through experimental work.

This investigative work forms the first contribution of our project and in it we specifically isolate misalignment between the data sampling regime and the state-action visitation of the learned policy as a fundamental algorithmic bottleneck when conducting reinforcement learning without exploration.

We frame our remaining objectives in terms of the high level contributions we hope to make:

1. Present an exposition of the buffer sampling misalignment problem and provide a theoretical explanation for its existence.
2. Construct a statistic that quantifies the degree of misalignment.
3. Develop solutions to the problem and identify a new algorithmic framework that can be used to categorise algorithms that aim to resolve the misalignment problem.

Since identifying a problem to solve fell within the remit of the project's work, it was impossible to identify our objectives ahead of time. This distinguishes our project in some ways since we both identify and solve a problem within the scope of our work.

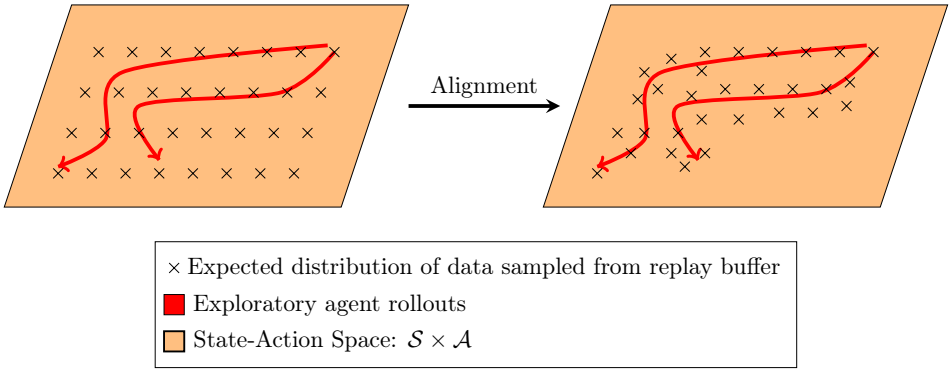


Figure 1.1: Representation of a *poorly aligned* sampling regime (left) compared to a *well aligned* regime (right). Notice that even if the training data covers the state-action visitation of the agent, it can still be poorly aligned with the subspace the agent has visited.

1.3 Contributions

We present multiple novel contributions to the offline reinforcement learning field in this project:

1. An exposition of distributional misalignment between the data sampled from the experience replay buffer and the state-action visitation of the learned policy as a challenge for offline reinforcement learning. To our knowledge, this is the first time this problem has been explictly isolated (Chapter 3).
2. A novel statistic that quantifies this distributional misalignment (Section 3.5).
3. The Model Aligned Offline Reinforcement Learning (MAORL) framework which characterises a new class of algorithms that address the misalignment problem while also mitigating *action distribution shift* (Section 4.1).
4. Buffer sampling alignment mechanisms that fit within the MAORL framework to address the misalignment problem:
   (a) A gradient-based alignment mechanism that uses techniques from convex optimisation to efficiently evolve the buffer sampling priority assignment for sampling from our offline dataset in a way that mitigates the misalignment problem (Section 4.3).
   (b) A fixed capacity, queue-based alignment mechanism inspired by the way the experience replay buffer is managed in online reinforcement learning algorithms (Section 4.2).
5. A proposal to use Spectral-normalised Gaussian Processes (SNGPs) to construct uncertainty-penalised dynamics models in reinforcement learning. We also provide one of the first reference open-source PyTorch implementations of SNGPs that can be immediately applied to regression tasks (Chapter 5).

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning provides us a framework within which to frame problems and solutions associated with objective-oriented, sequential decision making tasks where long term planning can be required [5].

Markov Decision Process

The reinforcement learning problem is formally one of optimal control in a potentially incompletely known Markov Decision Process (MDPs). We use MDPs to mathematically model the decision making process and can use them to define the optimisation problem we try and solve with reinforcement learning solution methods. In the context of reinforcement learning, we will define the underlying MDP using the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$:

- **State space ($\mathcal{S}$)**: defines the space of all observations the agent can receive from the environment.
- **Action space ($\mathcal{A}(s)$)**: defines the space of all actions the agent can take, given it is currently at state $s \in \mathcal{S}$.
- **Transition probability function ($\mathcal{T}$)**: models the environment dynamics by computing the probability that taking action $a \in \mathcal{A}$ from state $s \in \mathcal{S}$ leads to a transition to $s' \in \mathcal{S}$. This is usually implicit and not directly known.
- **Reward function ($\mathcal{R}$)**: computes the numerical reward signal received by the agent upon taking action $a \in \mathcal{A}$ in the current state $s \in \mathcal{S}$ to transition to destination state $s' \in \mathcal{S}$.

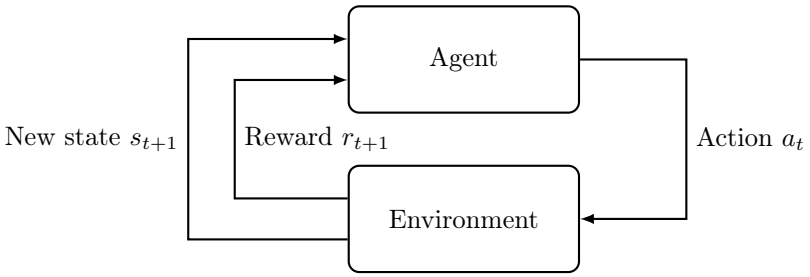


Figure 2.1: The agent-environment interaction in a Markov decision process [5]

We aim to use learning methods to learn an optimal *policy*. A policy π is defined as function $\pi(a|s)$ defining the probability distribution over actions $a \in \mathcal{A}$ that define how the agent acts optimally when in state $s \in \mathcal{S}$.

Acting optimally and rationally means we aim to learn a policy π^* that maximises the expected cumulative reward received from the environment. The cumulative reward for a finite trace of states and actions $s_0, a_0, s_1, a_1, ..., s_T, a_T, s_{T+1}$ is also defined as the return $R_T = \sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})$ where $\gamma \in [0, 1)$ is a discounting factor and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$ denotes the instantaneous reward function. Therefore, as practitioners we are able to define the agent's objective by designing the reward function to align with the overall objective.

Optimisation Problem

Using this, we denote the value of a state V^π or a state-action pair Q^π under some policy π as follows:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[R_t | S_t = s] \\ Q^\pi(s, a) &= \mathbb{E}_\pi[R_t | S_t = s, A_t = a] \end{aligned} \tag{2.1}$$

Using Equation (2.1), we can define the objective of reinforcement learning as that of learning an optimal policy π^*:

$$\begin{aligned} \forall s \in \mathcal{S} : \pi^* &= \arg\max_\pi [V^\pi(s)] \\ \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s) : \pi^* &= \arg\max_\pi [Q^\pi(s, a)] \end{aligned} \tag{2.2}$$

The *Policy Improvement Theorem* expresses that acting greedily with respect to the value function (i.e. deterministic $\pi'(a|s) = \arg\max_a Q^\pi(s, a)$) will only improve on the original policy.

The *Bellman optimality equation* can be used to compute π^* and can be extracted from Equation (2.2) and Equation (2.1) via expectation algebra. It intuitively expresses that the value of a state under an optimal policy must equal the expected return of the best action from that state.

$$\begin{aligned} V^*(s) &= \arg\max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s, a) \\ Q^*(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a] \end{aligned} \tag{2.3}$$

Solution Classes

We generally subdivide reinforcement learning algorithms as *model-based* or *model-free*.

1. **Model-based** learning is used when we can either directly access the transition dynamics of the underlying environment $\mathcal{T}$, or when we can estimate $\mathcal{T}$ well (typically using prior data and a powerful function approximator like a deep neural network). We use the term *dynamic programming* to refer to algorithms that compute optimal policies given a model of the environment as a Markov decision process. Typically such methods use bootstrapping to efficiently update value estimates based on neighbouring value estimates.
2. **Model-free** learning approaches rely on samples from the environment and so require no explicit model of the environment, and retain no explicit information about the dynamics. Algorithms founded on these principles are known as *Monte-Carlo algorithms*.

Temporal-difference learning combines the approaches of bootstrapping and sampling from the environment to reap some of the benefits of both approaches. Bootstrapping yields lower variance and better training efficiency while sampling from experience reduces bias.

Canonical one-step TD algorithms are Q Learning [6] and SARSA. Q Learning and SARSA illustrate another subdivision in reinforcement learning algorithms which is based on how we perform *exploration* during training.

1. **On-policy** methods explore and sample traces from the environment using the exact same up-to-date version of the policy they are learning π. SARSA falls under this category, since the action used to explore the next state S_{t+1}, $A_{t+1} \sim \pi(.|S_t)$, as seen in the update step Equation (2.4).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.4}$$

2. **Off-policy** methods introduce the notion of a behaviour policy π_β which is the policy used for exploration. In general this means the actions sampled for exploration are not necessarily identical to those that would be sampled using the learned policy, since in general $\pi_\beta \neq^d \pi$. Q Learning is the canonical example of an off-policy algorithm. We can observe in the Q Learning update step Equation (2.5) that we compute $\max_a Q(S_{t+1}, a)$ as part of the target value for the update computation. This implicitly means the policy being learnt π is maximising with respect to the subsequent action while the exploration policy π_β is not since it is stochastic with respect to the optimum policy. Since $\pi_\beta \neq^d \pi$, this is an off-policy method.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \arg\max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.5}$$

   Note that Q Learning can be viewed as a direct extension of Equation (2.3).

By learning Q we are implicitly learning our optimal policy π as well by taking $\pi(a|s) = \arg\max_a Q(s, a)$.

Deep Reinforcement Learning

Many recent successes of reinforcement learning [1, 7, 8] have been driven by combining reinforcement learning with powerful deep neural networks as function approximators. We can approximate either (or both of) the value function or policy function using deep artificial neural networks typically very successfully. This alleviates fundamental problems with tabular methods like the exponentially exploding memory requirements of storing Q-value estimates in high-dimensional state spaces and the lack of generalisation which slows learning. Typically making use of deep neural networks in a reinforcement learning context cannot be done naively and certain adaptations are required. This is because most deep learning techniques are developed in a supervised learning domain making certain underlying assumptions that are violated when adapted for use in reinforcement learning tasks.

Discrete Action Control

The most basic use of deep neural networks is to use them to approximate the value function. For a neural network parameterised by weights w, we aim to learn an estimator $\hat{Q}_w(s, a) \approx Q^*(s, a)$. The Deep Q-Network (DQN) [1] does this by using a neural network architecture where the state observation vector is the input and the Q value estimates for all discrete actions are output simultaneuously at the output neurons as illustrated by Figure 2.2.

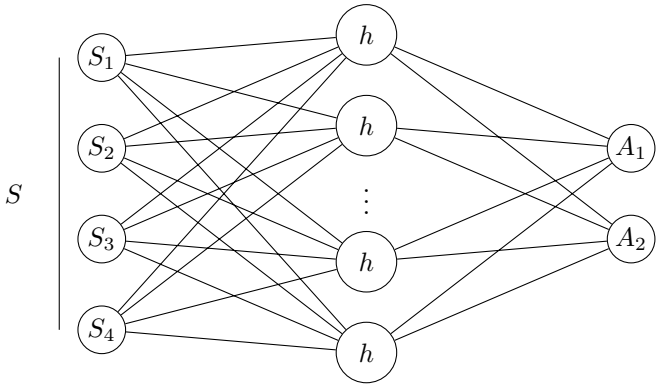


Figure 2.2: DQN

The authors also transform the Q Learning update Equation (2.5) into a the *Bellman Mean Square Error* loss function Equation (2.6) in order to be able to use gradient descent to optimise network weights w and minimise this loss. This involves regressing our prediction onto a target y that is based on the models value prediction of a future action.

$$\begin{aligned} y_j &= \begin{cases} r_j & s_{t+1} \text{ is a terminal state} \\ r_j + \gamma \hat{Q}_w(s_{t+1}, a) & \text{otherwise} \end{cases} \\ L_{BMSE}(w) &= \frac{1}{2}\,\mathbb{E}[(y - \hat{Q}_w(s_t, a))^2] \end{aligned} \tag{2.6}$$

Note that this yields the following gradient descent Equation (2.7).

$$w \leftarrow w - \alpha \nabla_w L_{BMSE}(w) \text{ where } \nabla_w L_{BMSE}(w) = \mathbb{E}[(y - \hat{Q}_w(s_t, a)) \nabla_w \hat{Q}_w(s_t, a)] \tag{2.7}$$

Training a DQN also involves an *experience replay buffer* which helps to stabilise learning and improve data utilisation. The agent conducts rollouts, experiencing (s, a, s', r) transition tuples that it stores into a fixed size replay buffer. Each iteration during training the agent then trains the deep neural network by sampling a uniform random batch from the buffer. A later innovation was to try and replicate the benefits of Double Q Learning [9] by introducing the concept of a target network $\hat{Q}_{\bar{w}}$ whose parameters $\bar{w}$ trail w by being frozen for a pre-determined number of iterations in order to stabilise learning by using the slower improving target network to calculate the TD-target in the Bellman update.

Continuous Action Control (Policy Gradient Methods)

While discrete action settings are useful to think about in the context of games and as a foundation, most real world applications such as robotics require continuous actions (e.g. motor velocities in a robot) with high degrees of precision making them impossible or inefficient to quantise into discrete chunks. For continuous control we primarily turn to **policy gradient methods**, where we aim to produce a policy network $\hat{\pi}$ that takes state observations as input and outputs actions. This is most commonly used in domains like robotics where our actions might be numerical values like joint velocities or forces.

The probability of observing state-action trace τ then the optimal policy network with weights θ^* is the one that produces the maximum expected return i.e. $\theta^* = \arg\max_\theta \mathbb{E}[\sum_{t=1}^{T} r(s_t, a_t)]_{(s_t,a_t)\sim p_\theta(\tau)}$. We can use gradient ascent to compute this maximum.

The *Policy Gradient Theorem* tells us that given our expected return $J(\theta)$ can be approximated by the empirical mean return over N traces, $\nabla_\theta J(\theta)$ can be computed as follows:

$$\begin{aligned} J(\theta) = \mathbb{E}[\sum_{t=1}^{T} r(s_t, a_t)]_{(s_t,a_t)\sim p_\theta(\tau)} \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t} r(s_{i,t}, a_{i,t}) \implies \\ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}))(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})) \end{aligned} \tag{2.8}$$

The remarkable result here is that we can remove the dependence on knowing the environment dynamics that knowing the probability of observing a trace $p_\theta(\tau)$ implies, enabling us to optimise $J(\theta)$ in a model-free manner. If we approximate the policy using a deep neural network we can use gradient ascent to maximise $J(\theta)$.

Actor-critic algorithms approximate the return calculation using a value-function approximator (the critic) to approximate $\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})$ rather than sampling from the environment exclusively. This is analogous to the bootstrapping-sampling tradeoff mentioned earlier and using an explicit critic can help reduce variance during learning.

The Deep Deterministic Policy Gradient algorithm [10] (DDPG) is a continuous control extension of DQNs to continuous action spaces which operates under the actor-critic framework. As with DQN, DDPG utilises target networks and experience replay buffers to stabilise training.

Over time, novel variance control techniques such as using clipped double Q Learning have been proposed, many of which have been incorporated into the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [11]. TD3 and associated methods represent the current state-of-the-art for general, continuous action deep reinforcement learning.

2.2 Offline Reinforcement Learning

Traditional reinforcement learning methods such as those discussed in Section 2.1 fundamentally work in an *online* fashion. This means that exploring the environment and trial and error is fundamental to the learning process. These methods are also typically very data intensive, meaning that to successfully learn good policies they require a very large number of episodes. While this is effective in games and simulated environments as evidenced by recent successes [1], this learning paradigm is untenable for most real world tasks where repeatedly trying strategies that fail can be impractical to manage, expensive, or outright dangerous such as in robotics and healthcare applications.

Offline reinforcement learning methods characterise a class of reinforcement learning methods that aim to learn policies from diverse, static datasets without exploration in the real world. By decoupling the data collection task from the policy learning task, successful offline reinforcement learning methods aim to remove the barriers associated with application of reinforcement learning methods for real world tasks and explore how to improve the data efficiency of reinforcement learning methods more generally.

2.2.1 Challenges

In principle, it should be possible to apply off-policy algorithms to our offline domain where exploration is not allowed by initialising the experience replay buffer with our precollected dataset and preventing any online data collection. However, as is discussed in [3] and [2], naive application of these methods suffer from problems termed *distributional shift*:

- **State distributional shift:** the learned policy π might encourage the agent to visit states that diverge significantly from the training data used to learn the policy. Extrapolating beyond the support of the training data in general can be dangerous and can lead to the learned policy producing erroneous actions in out-of-distribution states, so this is something that is best avoided. One way to mitigate this problem is to place a bound on how far the learned policy π can deviate from the offline buffer data distribution π_β. In [3], it is claimed that state distributional shift only affects test time performance and not training.
- **Action distributional shift:** target values for Bellman updates in Equation (2.3) can depend on value predictions for actions outside the support of the training data, leading to potentially erroneous target value predictions. Since we regress our value estimate onto the target when learning a value function for Actor-Critic algorithms, this could actually propagate errors throughout the value network. This causes an "unlearning" effect [3] where policy performance deteriorates drastically as errors propagate throughout the value function. This is also termed *bootstrapping error* and is further analysed in [2] and [12].

Action distributional shift is often exacerbated by the subtle maximisation operation involved in a Q Learning update step. Since out-of-distribution predictions typically have higher variance, in expectation we can expect target values to be biased *towards* out-of-distribution target estimates. Such small errors are remedied in *online* reinforcement learning by optimistically exploring overestimated regions and receiving corrective reward signals from the environment that force the agent to reduce its value estimate for the action. This bias can be interpreted as being optimistic in the face of uncertainty and is a useful

characteristic in online algorithms for encouraging exploration of fringe regions. Offline reinforcement learning algorithms often aim to reverse this by instead being conservative in the face of uncertainty since remedial exploration is not possible.

2.2.2 Current Approaches

Offline reinforcement learning methods in the literature primarily propose algorithmic improvements to mitigate the problem of *action distributional shift* by using various techniques to constrain the bootstrapping mechanism and prevent it from sampling out-of-distribution action value estimates. In constructing a taxonomy of the approaches in the field, we can first subdivide methods into *model-free* and *model-based* approaches. We will first explore *model-free* methods.

2.2.3 Model-free offline reinforcement learning

These can generally be subdivided into *policy constraint*, *pessimistic value functions*, and *uncertainty estimation* approaches, each of which try and tackle the *action distributional shift* problem in their own way. These methods are described as *model-free* since they do not require access to a learned MDP for training.

Policy Constraint Methods

Policy constraint methods aim to ensure that the actions we might want to use in computing the *target* value for a Bellman update are close to the behaviour distribution induced by the static dataset $\pi_\beta(a'|s')$.

When operating within an actor-critic framework, this constraint is applied during the policy improvement step in make sure policy improvements do not diverge significantly from the behaviour policy π_β as shown in Equation (2.9). The intention behind this is that this will constrain the actions used to construct target value estimates when conducting value network improvements to be close to the support of the training data, diminishing training problems from bootstrapping from out-of-distribution predictions.

$$\begin{aligned} \hat{Q}^\pi_{k+1} &\leftarrow \arg\min_Q \mathbb{E}_{(s,a,s')\sim D}[(Q(s,a) - (r(s,a) + \gamma\, \mathbb{E}_{a'\sim\pi_k(a'|s')}[\hat{Q}^\pi_k(s',a')]] \\ \pi_{k+1} &\leftarrow \arg\max_\pi \mathbb{E}_{s\sim D}[\mathbb{E}_{a\sim\pi(a|s)}[\hat{Q}^\pi_{k+1}(s,a)] \quad s.t. \quad D(\pi,\pi_\beta) < \epsilon \end{aligned} \tag{2.9}$$

In general, these approaches can suffer from two main drawbacks:

- Constraining policy improvement with respect to π_β can be too conservative and can prevent significant improvement of the learned policy over and beyond our modelling of the dataset.
- Many of these approaches require us to model the static dataset to obtain a more convenient form for π_β which adds complexity and can lead to a loss of precision.

Examples of this approach include BEAR [12], BCQ [2], and ABM [13]. These algorithms differ in many ways but an important consideration is the choice of the divergence metric D, which is often an f-divergence of some kind. These are discussed in Table 2.1.

Policy Penalty methods have a very similar approach except that they incorporate the constraint into the Q-values themselves by regularising our target value estimate using the divergence of the policy iterate with respect to the behavioural policy. This means that the policy also learns to avoid deviating from π_β on future time steps as well since penalties for doing so are incorporated into our value estimate. This has been discussed in the formulation of BRAC [14]. This yields a modified actor-critic algorithm

	BCQ [2]	BEAR [12]	ABM [13]
D policy divergence measure	The VAE modelling the data π_β aims to maximise the state-conditioned marginal likelihood.	Maximum Mean Discrepancy (MMD).	Kullback–Leibler (KL) divergence with the current advantage weighted behaviour model.
π_β model estimate	Actions are sampled from VAE used to model the behaviour policy distribution.	Actor policy network is constrained towards behaviour policy support during training by using dual gradient descent to solve constrained optimisation problem.	Advantage-weighted behaviour model computed by maximising the advantage-weighted log likelihood, using gradient ascent like policy gradient methods.

Table 2.1: Comparison of BCQ, BEAR and ABM approaches.

with critic and actor updates as follows.

$$\hat{Q}^{\pi}_{k+1} \leftarrow \arg\min_{Q} \mathbb{E}_{(s,a,s')\sim D}[(Q(s,a) - (r(s,a) + \gamma\, \mathbb{E}_{a'\sim\pi_k(a'|s')}[\hat{Q}^{\pi}_{k}(s',a')] - \alpha\gamma D(\pi_k(.|s'), \pi_\beta(.|s')))^2]$$
$$\pi_{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{s\sim D}[\mathbb{E}_{a\sim\pi(a|s)}[\hat{Q}^{\pi}_{k+1}(s,a)] - \alpha D(\pi(.|s), \pi_\beta(.|s))] \tag{2.10}$$

Pessimistic Value Functions

Pessimistic Value Functions present an alternative to imposing policy constraints in an actor-critic framework and are implemented by directly regularising the value function or Q-function in a way that avoids overestimation for out-of-distribution actions by adding a *conservative penalty* term, yielding the objective Equation (2.11) where different methods are distinguished by their choice of $\mathcal{C}$.

$$\bar{\mathcal{E}}(B,\phi) = \alpha\mathcal{C}(B,\phi) + \mathcal{E}(B,\phi) \tag{2.11}$$

One of the benefits of this approach is that we don't need to explicitly model the behaviour policy. It also is formulated in a manner that makes it applicable for both value-based methods as well as actor-critic methods unlike *policy constraint* methods which require a policy representation to constrain. The state-of-the-art approach here is CQL [15] which has the theoretical guarantee that conservative value function lower bounds the true value function in expectation (i.e. $\mathbb{E}_{\pi(a|s)}[\hat{Q}^{\pi}(s,a)] \leq V^{\pi}(s)$). The authors formulate the final optimisation of the value function in a way that includes the standard *Bellman Mean Squared Error*, and a term that aims to minimise in-distribution value estimates further while introducing a counter-balancing Q-value maximisation term that aims to maximise value estimates for actions sampled from π_β, and a regularisation term constraining the policy μ being used to effectively act out of distribution. Using a regulariser $\mathcal{R}(\mu)$, Equation (2.12) defines the CQL optimisation objective. A typical choice of $\mathcal{R}(\mu)$ might be a KL-divergence with a prior distribution.

$$CQL(\mathcal{R}) : \min_{Q}\max_{\mu} \alpha(\mathbb{E}_{s\sim D, a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim D, a\sim\hat{\pi_\beta}}[Q(s,a)]) + \frac{1}{2}\,\mathbb{E}_{BMSE} + \mathcal{R}(\mu) \tag{2.12}$$

Uncertainty Estimation Methods

Uncertainty estimation based techniques aim to estimate the *epistemic* uncertainty of the Q-function and use this to detect out-of-distribution actions. Formally, these techniques might aim to learn an uncertainty set or a distribution over possible Q-functions from the dataset D and use this to determine how uncertain any given value prediction is during training. Once we can estimate the uncertainty associated with a value prediction, we can compute a conservative estimate of the Q-function that corresponds to the *lower confidence bound* value estimate by adjusting naive value estimates downwards based on how uncertain the estimate is. Since out-of-distribution actions will have very high epistemic uncertainty (since they were never explicitly sampled and so there is no ground truth data associated with them), this will mean

that out-of-distribution actions are discounted when during policy improvement which should deter out-of-distribution actions being sampled.

New work such as EDAC [16] and UWAC [17] utilise ensemble networks and Monte-Carlo dropout respectively to compute uncertainty estimates - these uncertainty estimation techniques build on [18] which shows that dropout training in deep neural networks can be cast as approximating Bayesian inference in deep Gaussian processes, and [19] which shows that ensembles of deep neural networks can very successfully be used to approximate predictive uncertainty on regression and classification tasks. A key takeaway from UWAC is that the implementation scales the size of the loss during training of the value function approximator inverse proportionately to the uncertainty. This is an effective technique at reducing the weighting afforded to out-of-distribution Bellman Errors during training.

2.2.4 Model-based offline reinforcement learning

Model-based offline reinforcement learning methods can be distinguished from model-free methods by their use of learned predictive dynamics models. Predictive models can be learned from the large transition data making up the offline dataset through standard supervised learning to construct predictive models that predict the reward and next state given a current state and action $T(s_{t+1}, r_{t+1}|s_t, a_t)$.

Two model-based methods that provide solutions to mitigate *action distributional shift* in similar ways are MOPO [20] and MOReL [21]. Both of the methods propose adapting the learned MDP to penalise out-of-distribution actions in order to induce behaviour that is conservative against exploring outside the support of the offline data. This is important since the predictive model is only likely to be correct within the support of the offline data used to train it, and without guarding against out-of-distribution exploration the agent could potentially exploit erroneous reward predictions from the model far from where it was trained. Assuming we have a measure of uncertainty $u(s, a)$ for any given state-action tuple, the approaches adapt the MDP as follows:

1. MOPO [20] modifies the reward function $\tilde{r}(s, a) = r(s, a) - \lambda u(s, a)$ to be conservative in the face of uncertainty.

2. MOReL [21] only changes the reward function to a penalty value k if $u(s, a)$ is greater than a threshold, at which point the agent also reaches a terminal HALT state.

2.3 Uncertainty Estimation in Deep Neural Networks

While deep neural networks have been immensely successful in recent times and have been deployed in very many real world applications for predictive tasks, basic neural networks still cannot quantify their uncertainty or quantify what they *know what they don't know* effectively. This is a limiting factor for deployment of vanilla deep neural networks in safety critical applications. In applications like medical diagnosis, if the neural network knows when it is uncertain about its prediction then this might prompt human intervention in the process. We break down *predictive* uncertainty into three components [22]:

- *Aleatoric* uncertainty is irreducible through statistical means and comes from inherent noise in the data implicit in the way we observe and collect the data.

- *Epistemic* uncertainty is the caused by knowledge gaps in the model itself. Causes might be the model architecture or the way in which provided data is utilised.

- *Distributional* uncertainty is related to the uncertainty caused by a change in the input-data distribution, or the difference between the training and test data distribution (an example difference is covariate shift).

2.3.1 Gaussian Process Regression

Gaussian process regression bears many similarities to Bayesian linear regression. A *Gaussian process* [23] is described as a collection of random variable, any finite subset of which have a joint Gaussian distribution. As such we can characterise a Gaussian process by a mean function $m(x)$ and a covariance function $k(x, x')$, which are defined with respect to a true process $f(x)$. The covariance function specifies the covariance between pairs of random variables.

Typically we have $m(x) = \mathbf{0}$. A standard choice of covariance function is the *squared exponential kernel* as defined by Equation (2.13). Conceptually this is similar to having a Gaussian prior in Bayesian linear regression. k implies a distribution over candidate functions over any set of inputs if we sample an infinite stream of output values from $f \sim \mathcal{N}(0, K(X, X))$ given input points X. Note that l in Equation (2.13) is known as the length scale parameter, which controls roughly the distance in input space we would need to travel before the functions sampled change dramatically. Often we simply set l to 1.

$$cov(f(x_p), f(x_q)) = k(x_p, x_q) = \exp(\frac{1}{2l}|x_p - x_q|^2) \tag{2.13}$$

Being able to sample random functions from a prior is useful as we can apply a Bayesian treatment to the problem now to compute a predictive distribution which enables us to model uncertainty in predictions made using a mean sample function on different inputs of data at test time.

Predictive Uncertainty

Computing the predictive distribution amounts to attempting to compute the probability density function $p(y^*|\mathcal{Y}, \mathcal{X}, x^*)$ where x^* is our test input and where we have trained on data examples $\{x_n, y_n\}_{n=1}^N$ with $x_n \in \mathcal{X}$ and $y_n \in \mathcal{Y}$, the training dataset.

We are trying to model some true function $f(x)$ where we have access to noisy observations $y_n = f(x_n) + \epsilon_n$, $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ where ϵ is independent Gaussian noise representing irreducible aleatoric uncertainty. We use a covariance function K to compute the predictive distribution Equation (2.14) as follows through careful application of the product rule of probability on the joint probability distribution $p(y^*, \mathcal{Y}|\mathcal{X}, x^*)$.

$$\begin{aligned} y^*|\mathcal{Y}, \mathcal{X}, x^* &\sim \mathcal{N}(\bar{y^*}, \Sigma^*) \quad \textit{where} \\ \bar{y^*} &= K(x^*, \mathcal{X})[K(\mathcal{X}, \mathcal{X}) + \sigma_n^2 I]^{-1} y \\ \Sigma^* &= K(x^*, x^*) - K(x^*, \mathcal{X})[K(\mathcal{X}, \mathcal{X}) + \sigma_n^2 I]^{-1} K(\mathcal{X}, x^*) \end{aligned} \tag{2.14}$$

While this closed form solution is convenient to analyse, practical implementations of Gaussian Process Regression typically use low rank approximations due to the high computational cost of inverting $K(\mathcal{X}, \mathcal{X}) + \sigma_n^2 I$ especially when dealing with large amounts of data.

2.3.2 Deep Ensembles

[19] presents a simple, parallelisable approach to constructing epistemic uncertainty estimate. The authors propose using an ensemble of deep neural networks rather than one single one. If we initialise each network with slightly different parameters, then each deep neural network might learn slightly different function approximations by finding different local optima during training.

At inference time we can treat each of the outputs as samples from a random variable. Using this perspective, we can estimate a sample mean and spread from the predictions. On in-sample inputs close to the training data where we are *interpolating* data close to training examples there will be more mutual agreement between the constituent networks due to the presence of similar ground truth examples. This

leads to lower predictive uncertainty and variance in outputs. When *extrapolating* far from the support of the training data, it is more likely that the networks will disagree more significantly about the prediction. This makes taking the spread of predictions is a useful approximation to the true underlying epistemic uncertainty of the ensemble network.

While the random initialisation of the different networks in the ensemble can often be sufficient to ensure good diversity in predictions, we can boost diversity by using *bagging*, where ensemble members are trained on different subsets of the training set to decorrelate their predictions and boost diversity.

2.3.3 Spectral-normalised Gaussian Processes

Spectral-normalised Gaussian Processes present a deep learning architecture that can produce high quality uncertainty estimates using only a single Deep Neural Network. The authors identify that *distance awareness* is important for good uncertainty estimation through a rigorous minimax analysis [24], where the distance between the input test example and previously seen training examples can be used to return a uniform distribution over output labels if the input is out-of-domain but place more probability mass on prediction when more certain if the data to predict is close to the training data used to train the predictive model.

Since Gaussian Processes with suitable kernels such as the *squared exponential kernel* Equation (2.13) maintain distance awareness property, the authors propose adding a Gaussian Process layer (GP layer) to the end of a standard Residual Neural Network architecture so that we can combine this distance awareness with the deep feature extraction and powerful classification abilities of deep neural networks. To ensure this is computationally scalable, this is approximated using a Laplace approximation to the random feature expansion of the Gaussian Process.

However, simply adding a Gaussian Process layer on top of the last hidden layer of a neural network might not immediately work since distances using the hidden representations in a DNN might not represent a meaningful distance in the input data space which is really what we desire. The authors propose applying *spectral normalisation* to weights in each residual hidden layer after each gradient-based update step as a way to preserve distance awareness.

Preserving a correspondence between the semantic distance of points in the hidden representation of an input point $h(x)$ and the point x itself amounts to satisfying the *bi-Lipschitz* condition Equation (2.15) where $0 < L_1 < 1 < L_2$.

$$L_1 * \|x - x'\| \leq \|h(x) - h(x')\| \leq L_2 * \|x_1 - x_2\| \tag{2.15}$$

The authors of [24] realised that in modern deep learning architectures composed of residual blocks $h(x) = h_{L-1} \circ ... \circ h_1(x)$, where $h_l(x) = x + g_l(x)$ and $g_l(x) = \sigma(W_l x + b_l)$, bounding the residual maps less than 1 will ensure that h is distance preserving by the definition of Equation (2.15). This is because if for some $0 < \alpha \leq 1$ if $\|g_l(x) - g_l(x')\| \leq \alpha\|x - x'\|$ then Equation (2.15) is satisfied with $L_1 = (1-\alpha)^{L-1}$ and $L_2 = (1+\alpha)^{L-1}$, making h distance-preserving. A proof of this result can be found in [24].

The way in which the authors propose we satisfy this is to ensure the weight matrices in the residual blocks have spectral norm (largest singular value) less than 1. At every training step, we first estimate the spectral norm $\hat{\lambda} \approx \|W_l\|_2$ using the power iteration method before then normalising the matrix with the step $W_l = c * W_l / \hat{\lambda}$ if $c < \hat{\lambda}$, where c is a hyperparamter that can be used to adjust the spectral norm bound on W_l.

This enables classification with accurate epistemic uncertainty estimates based on distance from the data manifold, as can be seen in Figure 2.3). We can produce the following uncertainty detection on a naive 2-class dataset extracted from the *Two Moons sklearn* dataset.

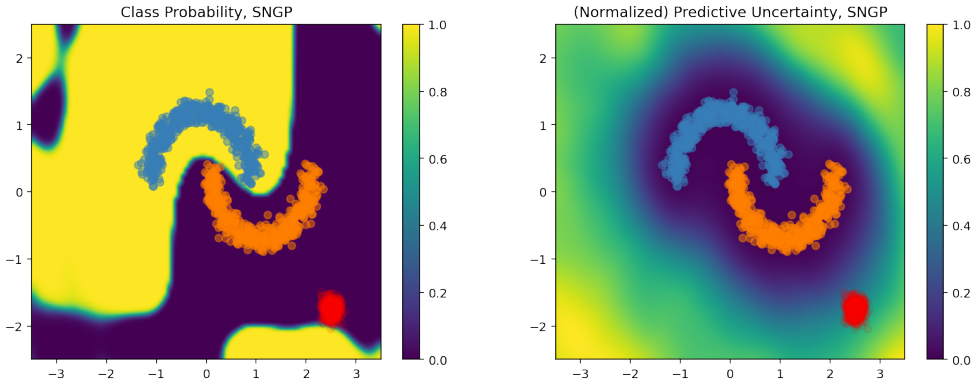


Figure 2.3: Epistemic uncertainty estimate on two-class dataset. The data in red represents out-of-distribution data which the network accurately estimates high uncertainty for. My open-sourced PyTorch SNGP implementation, this figure, and further study into SNGPs can be found at [25]

2.4 Convex optimisation

Convex optimisation problems are optimisation problems concerned with minimising convex functions with feasible solutions defined by convex sets.

A function f is convex if satisfies the following condition:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y) \tag{2.16}$$

Twice differentiable functions can be shown to be convex if their second derivative (or Hessian matrix of second partial derivatives) is positive (or positive definite). This can be a convenient way to illustrate a function is convex.

A set $\mathcal{C}$ is convex if $\forall x, y \in \mathcal{C}$ and $\forall \theta \in [0,1]$ we have:

$$\theta x + (1-\theta)y \in \mathcal{C} \tag{2.17}$$

We can therefore summarise a convex optimisation problem as $\min_x f(x) \quad s.t. \quad x \in \mathcal{C}$. One very useful property is that every local minimum point in a convex optimisation problem is also a global one. This, alongside, the amenability of these problems to gradient-based methods, has led to numerous techniques to efficiently solve the various subclasses of convex optimisation problems.

2.4.1 Standard form

If our objective f is convex, then a convex optimisation problem is deemed to be in standard form if written in the following way:

$$\begin{aligned} \min_x \quad & f(x) \\ s.t. \quad & g_i(x) \leq 0 \quad \forall i \\ & h_j(x) = 0 \quad \forall j \end{aligned} \tag{2.18}$$

where inequality constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}$ are convex and equality constraint functions $h_j : \mathbb{R}^N \to \mathbb{R}$ are affine transformations of the optimisation variable x.

2.4.2 Projected Gradient Descent

Convex optimisation problems are especially amenable to gradient-based solutions due to the fact that local minima are also globally minimum. Unconstrained convex optimisation can be solved with standard gradient descent. For constrained problems, we cannot guarantee that each intermediate iterate remains feasible, and so we cannot guarantee the final solution is a feasible one either. We can augment the standard gradient descent update with a projection operation that projects an intermediate iterate $\bar{x}_{t+1}$ below onto the convex set $\mathcal{C}$ defining the problem constraints.

$$\begin{aligned} \bar{x}_{t+1} &= x_t - \alpha \nabla L(x_t) \\ x_{t+1} &= \text{proj}_{\mathcal{C}}(\bar{x}_{t+1}) \end{aligned} \tag{2.19}$$

This can be an efficient way to find a solution to the problem where a direct solution is impossible or computationally expensive. Determing a good projection is highly dependent on the constraint set $\mathcal{C}$.

Projection onto a simplex

One especially interesting convex set is the *simplex* ($\Delta \subset \mathbb{R}^N$), defined as follows:

$$\Delta = \{x \in \mathbb{R}^N | 1^T x = c \;\; \& \;\; x^{(n)} \geq 0 \quad \forall n \in 1 \dots N\} \tag{2.20}$$

Note that when $c = 1$, Δ is known as the probability simplex since it encompasses the set of vectors that satisfy the constraints to parameterise a categorical probability distribution. For this reason, it is of special interest to find good projections onto Δ since in many problems we want to optimise an objective with respect to a probability vector.

Finding a good Euclidean projection $x \in \Delta$ of some vector $\bar{x} \in \mathbb{R}^N$ can be defined through the following optimisation problem:

$$\min_x \quad \frac{1}{2}||x - \bar{x}||^2 \quad \text{s.t.} \quad \forall n \in \{1, .., N\} : x^{(n)} \geq 0 \quad 1^T x = c \tag{2.21}$$

A sorting based approach to solve this was proposed in [26], yielding the following algorithm:

Algorithm 1 Procedure for projecting $\bar{x} \in \mathbb{R}^N$ onto Δ

procedure ProjectOntoSimplex($\bar{x}$)
  $y \leftarrow \bar{x}$ sorted descending
  $K \leftarrow \max_{1 \leq k \leq N} \{k | (\sum_{r=1}^{k} \frac{u_r - a}{k} < u_k\}$
  $\tau = \frac{\sum_{k=1}^{K} u_k - a}{K}$
  For $n = 1, \dots, N$, set $x_n = \max\{y_n - \tau, 0\}$
  return x
end procedure

Follow on approaches [27] have introduced new performance improvements and have managed to bring the observed computational complexity of the projection down from $\mathcal{O}(N \log N)$. Interested readers are encouraged to read [27] for a summary of existing approaches.

2.5 Ethical Discussion

This project aims to develop novel approaches to improve the state of the art in offline reinforcement learning without a specific practical application in mind. While we ourselves do not have any specific practical aims in mind, we must consider the potential for misuse of work produced in this project.

One of the promises of offline reinforcement learning is the ability to develop decision making engines from data. The novel algorithms proposed may be leveraged by actors with malicious intent to develop automated agents that could do genuine harm in the real world.

With the focus of offline reinforcement learning on producing a data-driven reinforcement learning paradigm, this directly benefits holders of large amounts of data such as government organisations and big technology companies. In many situations these methods might be leveraged to automate decision making for objectives that do not align well with with the good of wider society due to profit incentives or political goals that can be obtained by application of successful offline reinforcement learning techniques. Using offline reinforcement learning algorithms, bad actors with vast data resources can be more efficient.

Another consideration that comes with moving to a data-driven paradigm for reinforcement learning is the way in which we use and store data. All datasets we collect for our experiments are constructed using benchmarks in simulation and no personal data is involved in any way at any point in the process. As robots and other applications of offline reinforcement learning begin to move into the homes of consumers and interact with real people, designers of robots that leverage offline reinforcement learning methods to learn how to do tasks will have to be mindful of the potential use of personal data and the dataset inherited bias that can influence the policies learned. This will become more relevant as practical applications emerge.

Looking at the bigger picture, machine learning training generally requires a significant amount of computation power, and therefore can consume a lot of electricity. We must be mindful when running large machine learning experiments on compute clusters that this incurs a significant power demand which may be contributing to climate change if facilitated through non-renewables.

Chapter 3

The Sampling Alignment Problem in Deep Offline Reinforcement Learning

In this chapter, we present analysis that isolates the misalignment between the buffer sampling distribution and the policy state-action visitation distribution as a cause of performance degradation in offline reinforcement learning. We argue that this is distinct from *action distributional shift*, a related issue present when training reinforcement learning algorithms without exploration that has been investigated in detail in the literature [3] [2]. To our knowledge, this is the first time misalignment has been explicitly isolated as a distinct issue and we expect it can open up new avenues for algorithmic development.

We started our investigation by investigating an idealised but as yet unconsidered attempt at doing offline reinforcement learning where we *know what we don't know* and are able to manually apply negative rewards ahead of time to actions outside the support of our training data. While this is an unrealistic set up, it helps us disentangle the causes of performance degradation in off-policy offline reinforcement learning methods since manually penalising out-of-distribution actions with negative rewards will deter the agent from learning policies that take it away from the data it was trained on. By making these actions terminal, we can also make bootstrapping from out-of-distribution actions impossible.

Under the hypothesis that *action distributional shift* is indeed the only bottleneck present when training a reinforcement learning without exploration, this manual reward labelling would be sufficient to ensure performance on par with an online baseline agent since no actions are now outside the support of the training data. One perspective to take is that we have a perfect uncertainty estimator for detecting out-of-distribution actions in a *uncertainty estimation* based offline reinforcement learning algorithm.

In this chapter, we discuss the following:

- We find experimentally that despite ablating *action distributional shift* as a problem, naively applying a DQN algorithm in an offline fashion still lags online learning in terms of performance.
- We isolate divergence between the buffer sampling distribution and the on-policy state-action visitation distribution (visualised in Figure 1.1) as a fundamental algorithmic bottleneck that explains this performance gap. Since much of the algorithmic development that has occurred in the offline reinforcement learning community has been motivated primarily to mitigate boostrapping error caused by action distributional shift [2] [12] [13] [14] [15], this insight opens up new avenues for algorithmic development in the offline reinforcement learning field.
- We propose a metric to approximately quantify this divergence and illustrate how online agents are able to control this divergence implicitly, contributing to their superior performance.

We also hypothesise that the better alignment between sampling distribution and policy state-action visitation in model-based algorithms explains the question posed by the authors of the model-based offline

reinforcement learning algorithm MOPO [20] of why model-based approaches are currently better than their model-free counterparts.

3.1 Experimental Setup

To investigate whether naive application of reinforcement learning without exploration is hindered solely by action distributional shift, in our experimental set up we need to make it impossible to bootstrap off out-of-distribution value estimates. One way to do so is to collect every conceivable *(state, action, next state, reward)* transition tuple that can be experienced into our offline dataset. Transitions that take the agent *out-of-distribution* are explictly encoded as terminal and penalised with negative rewards. In this way we are able to guarantee that the offline agent is able access all transitions an online agent could possibly experience through exploration and the only remaining difference between online and offline training regimes is now the order and distribution of the transitions sampled for learning.

GridWorld

For the environment we forked the gym-minigrid repository to construct a suite of 21×21 2D GridWorld tasks. We illustrate *GridWorld-v0* in Figure 3.1. Each state is encoded by a three dimensional state vector which includes the x and y position as well as the orientation $\{\text{North}, \text{South}, \text{East}, \text{West}\}$. The discrete action set is defined $\mathcal{A} = \{\text{move forward}, \text{rotate left } 90^o, \text{rotate right } 90^o\}$, meaning that moving to an adjacent state can require multiple actions to first orient the agent before moving forward. We have a sparse reward function, with Lava states ($\mathcal{L}$) and Goal states ($\mathcal{G}$) terminating and receiving negative and positive rewards respectively. Further details about the dynamics of our GridWorld implementations can be found in Appendix C.

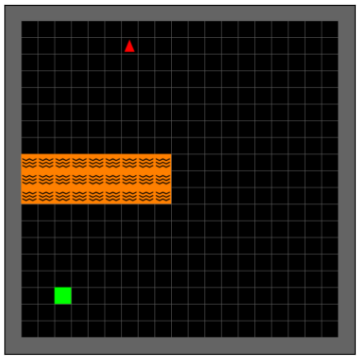


Figure 3.1: 21×21 GridWorld-v0 used for experimentation. The green state denotes the goal state. The orange lava states are used to model out-of-distribution regions. We can dynamically vary the location of these lava regions to auto-generate different test environments.

To validate our reward set up and to construct a baseline, we train an online Double DQN agent with a linearly decaying ϵ-greedy exploration strategy. We use 2-step returns and apply a small step cost to encourage quickly reaching the goal. Hyperparameter details can be found in Appendix A.

Static experience replay buffer construction

We collect an offline dataset by collecting each possible transition that the agent could have theoretically taken in the environment. This was the primary motivation for constructing a discrete action environment for testing rather than a continuous control task. In this way, we can make sure that the data available to

our offline agent is identical to the online agent and bootstrapping error cannot be a cause of performance degradation. In line with major model-free offline reinforcement learning algorithms published to date [12] [2] [14] [15] [13], we sample uniform randomly from our static experience replay buffer.

Results

Surprisingly, we find performance of the offline DQN agent to be significantly worse than the online agent baseline. We illustrate the mean rollout returns as training progresses of the optimum policy learned by the agent at the end of each episode on the GridWorld-v0 environment. The offline agent performs so poorly despite all other hyperparameters being the same and having access to all possible transitions that can be experienced by the online agent in the static replay buffer.

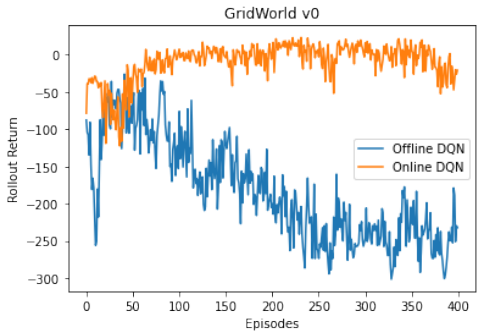


Figure 3.2: Mean Undiscounted Return Each Rollout over 5 Random Seeds

By inspecting individual training logs and plotting the distance from the goal at the end of each episode, we notice that like the online agent the offline agent does in fact learn good policies at certain points during training that reach the goal. However, unlike the online agent, it seems to consistently *unlearn* these policies shortly thereafter after learning more.

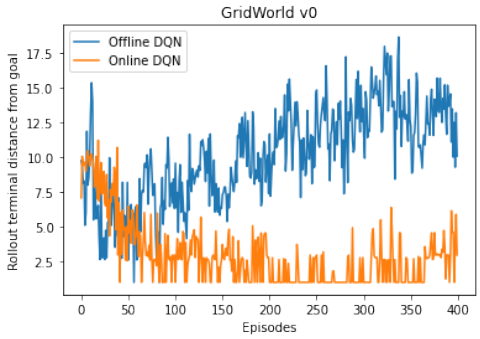
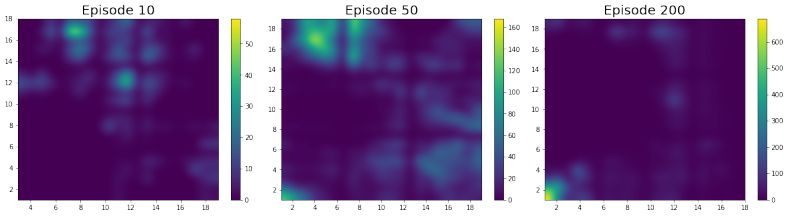
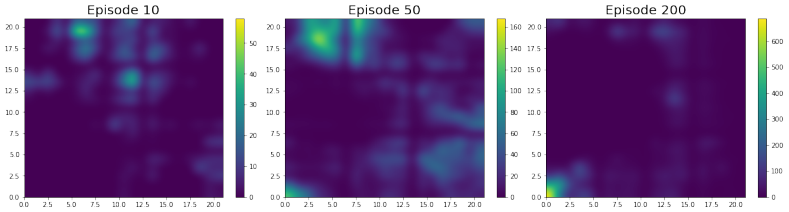


Figure 3.3: Mean Rollout Terminal State Distance from Goal State over 5 Random Seeds

The only difference between the agents can be clearly identified as the order and distribution of transitions being sampled as a subset of the pool of available transitions. One of the benefits of working in a 2D environment is we can easily visualise this sampling distribution to understand how it evolves as training progresses by calculating the relative frequency of transitions starting at each each state in the online replay buffer. We can visually see in Figure 3.4 that the buffer sampling distribution of the online agent moves in tandem with the last window of transitions it has experienced and used to populate the buffer. This is expected since in the DQN algorithm we use explored transitions to populate a fixed capacity experience replay buffer which we later sample uniform randomly from.



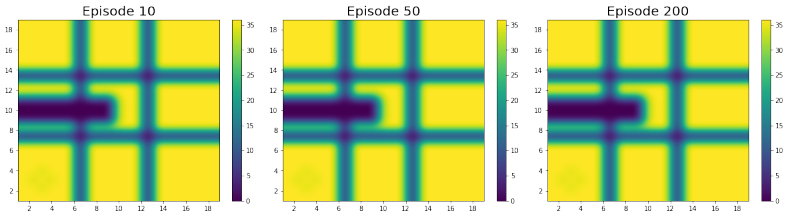
(a)



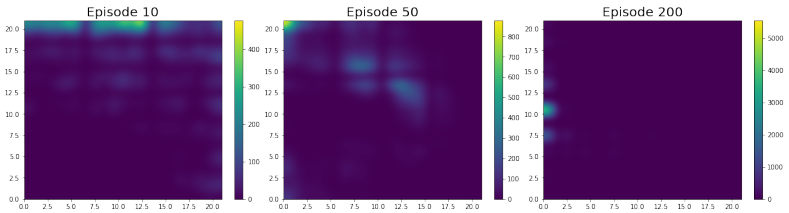
(b)

Figure 3.4: Figure 3.4a illustrates how the relative frequency of transitions beginning at each state evolves in the online buffer as training progresses. Figure 3.4b illustrates how the states explored by the agent evolve. The online agent is able to exploit a path to the goal and the buffer contents are aligned with the state-action visitation of the agent.

For the offline agent, as expected, this sampling distribution does not change as the replay buffer is fixed. All of the stored transitions are equally likely to be sampled, which in our case means any visitable transition has an equal likelihood of being sampled. As the policy is optimised, the divergence between the states and actions visited by the agent and those sampled grows. Eventually, good policies that are moving in the region of the goal state are *unlearned* and the agent does not recover.



(a)



(b)

Figure 3.5: Figure 3.5a illustrates how the priorities assigned to transitions beginning at each state changes in an naive offline set up where our data collection is uniform over the non-lava region. Figure 3.5b illustrates how the states explored by the agent evolve. The difference between these stays large.

This suggests that having very poor alignment between the on-policy state-action visitation given access to the true MDP and the buffer sampling regime can have a detrimental influence on the training of offline reinforcement learning algorithms. We make the argument that this is distinct from the *action distributional shift/bootstrapping error* class of problems discussed heavily in the literature due to the fact that in no situation does the offline agent actually sample an out-of-distribution transition when computing its Q Learning target in our experimental set up. To our knowledge, the *misalignment problem* has never been explictly identified in the literature.

Explaining why this derails training is the focus of much of this chapter.

3.2 Explaining the *unlearning effect*

We have just experimentally seen how even in the absence of out-of-distribution bootstrapping, offline reinforcement learning can fail. Our experiments suggest that the data sampling distribution is a core reason why. We now attempt to provide an explanation for this problem. We begin by constructing an explanatory example to illustrate how off-policy learning with function approximators can lead to *unlearning* of good policies during training. After this, we make the argument that offline reinforcement learning is really a limiting case of off-policy learning and consider how off-policy online methods control off-policyness of the data they learn from.

3.2.1 A didactic example

As discussed in [5], naively learning from *off-policy* data collected using a policy π_β that is significantly different our current policy is significantly more challenging compared to using *on-policy* data from rollouts of π, in the presence on function approximation. We construct a simple, instructional example to demonstrate the problem in Figure 3.6 of *unlearning* which can happen when learning from off-policy data. We attempt to learn a value function V_θ over states via a TD error based gradient update.

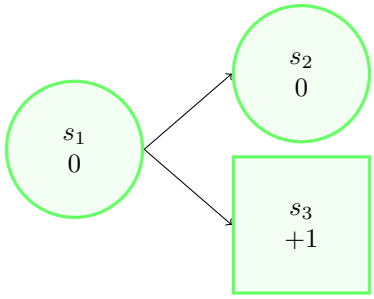


Figure 3.6: Environment set up. We can assume that our value predictions for each state are interlinked because we use a deep neural network with global parameters θ to approximate V_θ. We can also assume that the states form a part of a larger MDP. s_3 represents our terminal goal state in this MDP.

Figure 3.6 illustrates a small part of a larger MDP for which we seek to learn a value function using a function approximator. s_3 is our terminal goal state, and s_1 is a state that can be reached directly before it. We can assume that due to global parameterisation of the function approximation, the value prediction of all the states are linked such that if the value prediction for any one state changes by x, this results in a value prediction change of $0.1x$ for the other states. We assume the reward is solely tied to the state, and that we have a TD update in the following form:

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s)) \tag{3.1}$$

We will assume that $\gamma = 1$ and $\alpha = 0.1$, meaning that sampling the step $(s_t \to s_{t+1})$ yields the update $V(s_t) = 0.9V(s_t) + 0.1(r_{t+1} + V(s_{t+1}))$.

Since we are using a randomly initialised function approximator to represent our value function, we can assume there is some noise initially in our value function estimates as shown in Figure 3.7.

$V_\theta(s_1)$	$V_\theta(s_2)$	$V_\theta(s_3)$
0.01	−0.04	0.03

Figure 3.7: Iteration 0 (Random V_θ initialisation)

If the agent samples the step $(s_1 \rightarrow s_3)$, we end up updating $V_\theta(s_1)$ as per Equation (3.1) with an increase of 0.102. This means the other states increase by 0.0102.

$V_\theta(s_1)$	$V_\theta(s_2)$	$V_\theta(s_3)$
0.112	-0.0298	0.0402

Figure 3.8: Iteration 1 (Samples $(s_1 \rightarrow s_3)$)

The policy that maximises return accoding to the agent is one through s_1 and terminating at s_3, which is desirable. If our sampling strategy was to sample based on the on-policy visitation distribution, we would again soon sample $(s_1 \rightarrow s_3)$. This time, this would yield an update of 0.09282, influencing the other state value predictions by 0.009282. We can see that this is good. We are propagating reward information from s_3 to a predecessor state s_1. This will help construct a good optimal policy that maximises return by reaching s_3.

$V_\theta(s_1)$	$V_\theta(s_2)$	$V_\theta(s_3)$
0.20482	-0.019918	0.049482

Figure 3.9: Iteration 2A (On-policy update sampling $(s_1 \rightarrow s_3)$)

However, if we are learning from off-policy data, we might instead sample $(s_1 \rightarrow s_2)$. In this situation, we would see an update of -0.01418 towards $V(s_1)$ and -0.001418 for the other states.

$V_\theta(s_1)$	$V_\theta(s_2)$	$V_\theta(s_3)$
0.19064	-0.021336	0.048064

Figure 3.10: Iteration 2B (Off-policy update sampling $(s_1 \rightarrow s_2)$)

We can see here that $V(s_1)$ decreases as expected, but so too does $V(s_2)$. It is worth considering what might happen if we now repeatedly sampled $(s_1 \rightarrow s_2)$ without ever visiting the states after s_2 in the MDP to obtain a concrete reward signal. We can see that each time we decrease $V(s_1)$ to be closer to $V(s_2)$, $V(s_2)$ will also decrease! Since we do not process the transitions after s_2 in the wider MDP in a systematic way, our predictions of the values in this region will decrease over time. $V(s_2)$ becomes a constantly decreasing moving target for $V(s_1)$. In this way $V(s_1)$ will eventually decrease to a point where we forget the good policy to s_3, our target goal state.

While this may seem a little bit contrived in our 3 state set up, it is worth recalling that when learning from sparse reward data, a very small proportion of transitions have strong reward signal. While we may not sample the same transition again and again, we will sample many nearby transitions that similarly have 0 reward. Sampling in an unstructured manner has the potential to drown out these reward signals. Carefully ordering which transitions we do sample by aligning them with the states the agent would visit would correct for this. The way to correct for the ever decreasing $V(s_2)$ is to instead sample and process transitions in a more ordered fashion more closely linked to the states and actions we would visit. For instance, it is in fact okay to sample $(s_1 \rightarrow s_2)$ but then we should also be sampling transitions that lead out from s_2 so that our $V(s_2)$ is based on some downstream reward signal. This helps correct for both over and under predictions and it means that ultimately, value predictions are based on reward signals propagated through their adjacent states and not simply on vacuous bootstrapping.

3.3 Offline Reinforcement Learning as a special case of off-policy learning

In *off-policy* learning we seek to learn a value or policy function for a target policy π given access to a experience collected using a different behaviour policy π_β, with $\pi_\beta \neq^d \pi$. Since we sample transitions based on π_β, it controls which transitions are used to optimise π. We will denote the divergence between these as $D(\pi, \pi_\beta)$. When $D(\pi, \pi_\beta)$ is large, then the distribution of the data used to perform updates does

not match the current on-policy distribution and we can consider $D(\pi, \pi_\beta)$ to denote how "off-policy" our data sampling distribution is.

In the context of offline reinforcement learning, we learn our policy from a static offline dataset. The data sampling distribution from our replay buffer defines our behaviour policy π_β in this situation. In typical model-free offline reinforcement learning algorithms, we sample uniform randomly from the precollected replay buffer, which implies that π_β does not evolve over the course of training. In some sense, this makes offline algorithms as off-policy as it gets.

When we have a fixed π_β, attempting to improve π beyond π_β will increase the divergence between the current policy and the data collection policy, $D(\pi, \pi_\beta)$. In the presence of function approximation, this can lead to *unlearning* during training. We propose that the lack of precise consideration of the buffer sampling mechanism is a drawback for existing model-free offline reinforcement learning methods, even those that successfully counteract the problem of *action distributional shift*.

3.4 How online off-policy Reinforcement Learning algorithms control $D(\pi, \pi_\beta)$

In successful off-policy online reinforcement learning algorithms such as DQN [1] and DDPG [10], we typically see both π_β and π evolve as training goes on. The former evolves typically due to mechanisms such as fixed sized experience replay buffers that evict old transitions after a fixed number of steps and because fundamentally, in both of these algorithms, the data collection policy π_β is actually tied to π in some sense, meaning as we improve our policy we keep π_β.

1. In the discrete action DQN algorithm, the exploration policy is epsilon-greedy with respect to π. The parameter ϵ controls strictly to what degree the two are tied, and typically we see ϵ decayed as the learning process proceeds. This mechanism gradually reduces the difference between π and the exploration policy. In DDPG, the authors opt to add Ornstein-Uhlenbeck noise to the current non-stochastic policy π to control exploration.

   Clearly policy improvements will implicitly influence the data-collection policies in both algorithms since these are noise-augmented versions of the current policy. We note that model-free offline reinforcement learning algorithms generally lack this kind of mechanism.

2. In both of these algorithms, the use of the experience replay buffer introduces a certain amount of off-policyness. This is because the buffer stores transitions collected using older versions of the policy. As analysed in [28], the number of policy optimisation steps present between the oldest transition and the newest transition in the experience replay buffer can be a good proxy for off-policyness of the data used for learning. The finite capacity of the replay buffer therefore acts as a control on the degree of the divergence of the data sampling distribution π_β from the current policy π. It is no surprise therefore that studies into the experience replay buffer [28] have found that having a very large replay buffer can degrade performance with the DQN algorithm.

These mechanisms typically implicitly mean that the divergence between π_β and π does not grow unbounded in an online set up with exploration. One can make the argument that successful, off-policy deep reinforcement learning algorithms such as DQN and DDPG are not *truly off-policy*, since the exploration policy is directly coupled to the most recent policy at each iteration of training.

Model-free offline reinforcement learning methods lack this mechanism since they cannot explore the environment to the populate a buffer of transitions to learn from. We propose that this distinction partly answers the question posed by the authors of MOPO [20] of why model-based methods seem more successful in offline reinforcement learning. Model-based offline reinforcement learning algorithms such as MOPO [20] use an uncertainty-aware dynamics model learned from the transition information in the offline dataset to optimise a policy using any standard reinforcement learning algorithm by learning from model rollouts, benefiting from the same mechanisms that control $D(\pi, \pi_\beta)$ as online algorithms.

3.5 Quantifying policy-buffer state action visitation divergence

We attempt to quantify the degree of divergence between the policy π and the data collecting process π_β, $D(\pi, \pi_\beta)$ in order to begin precisely thinking about the problem of quantifying sampling mismatch and to motivate solutions to this problem.

It is first worth analysing what it means for the data sampled from the static buffer to be distributionally *different* from the on-policy distribution. Alongside the lack of consensus in the reinforcement learning community as how to quantify *off-policyness*, we identify the following challenges.

- Our policy is represented by a complex neural network and our static dataset is collected using an unknown process, so we do not have access to concise, parametric descriptions of our data buffer distribution π_β. We therefore would like a non-parametric, sample based statistic so we can avoid the shortcomings of trying to model π_β.
- The Markovian nature of samples from the on-policy distribution π and possibly from the dataset distribution π_β can be challenging since this violates independence assumptions at the heart of many non-parametric probability density comparison techniques.
- Transitions in the data buffer may be generated in a completely unknown way, meaning we can make very few assumptions if any on the nature of π_β if we want a general statistic.

We note that we can compare the distributions induced by the data buffer π_β and the policy π by comparing samples from states and actions they visit in the real world, rather than trying to compare an underlying parametric form. Unfortunately there is no consensus on how to do this and so we attempt to develop our own approximate statistic.

3.5.1 Defining a sample-based statistic

Maximum Mean Discrepancy is a non-parametric, kernel-based two sample test [29]. The authors of this test showed that provided access to independent and identically distributed samples $X, X', X_1, .., X_N$ from probability distribution p and $Y, Y', Y_1, .., Y_M$ from probability distribution q, we can quantify a measure of statistical distance as follows:

$$MMD^2[\mathcal{H}, p, q] = \mathbb{E}[k(X, X') - 2k(X, Y) + k(Y, Y')] \tag{3.2}$$

The statistic is based on the idea of representing the distance between distributions by using the distances between mean embeddings of features.

Using this, the authors also proposed a kernel-based, two sample test to see if distributions p and q are different. This has the advantage that it is sample-based, which is a desirable quality for the statistic we are looking for since do not have a concise, parametric description of the replay buffer.

$$\hat{MMD}^2[\mathcal{H}, p, q] = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} k(x_i, x_j) + \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} k(y_i, y_j) - \frac{2}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M} k(x_i, y_j) \tag{3.3}$$

We propose adapting this to our problem in order to quantify off-policyness (and to later develop an objective function to optimise priorities attached to each transition in the static data buffer). Acknowledging that we are violating the independence assumption in the two-sample MMD test to do so, we propose an *weighted MMD* metric for quantifying how distributionally different transitions in the buffer are to those in policy rollouts.

$$MMD^2_{weighted}[\pi, \pi_\beta] = \mathbb{E}_{X\sim p_\pi X'\sim p_\pi Y\sim pY'\sim p}[k(X, X') - 2k(X, Y) + k(Y, Y')] \tag{3.4}$$

We denote transitions from the policy rollouts as $\{x_i\}_{i=1}^N$ and transitions from the data buffer as $\{y_i\}_{i=1}^M$. Note that each transition is encoded as a row vector containing all the non-reward information required to distinguish a transition (the state, the action and the next state is sufficient). Concatenating these row vectors together, we arrive at X to represent the stacked rollout transitions and Y to represent the stacked buffer transitions. We have illustrated this explicitly in (3.5) .

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} [s_1 a_1 s'_1] \\ [s_2 a_2 s'_2] \\ \vdots \\ [s_N a_N s'_N] \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} [s_1 a_1 s'_1] \\ [s_2 a_2 s'_2] \\ \vdots \\ [s_M a_M s'_M] \end{bmatrix} \tag{3.5}$$

We also specify the buffer priorities as $p \in \mathbb{R}^M$, which ultimately controls the likelihood each transition is sampled from the offline dataset. Formally, p explicitly models the event probabilities when modelling π_β using a categorical distribution with support that is the transitions in the offline buffer. The probability of sampling transition y_m in the offline buffer has priority $p^{(m)}$, the mth component of p.

We denote p_π as the probability vector describing the probability of reaching each subsequent rollout transition using policy π, defined analogously to Equation (3.6). The nth element $p_\pi^{(n)}$ is the probability of reaching the nth transition in the rollout. Note that if transition x_{t+k} comes after x_t in a rollout, then necessarily $p_\pi^{(t)} \geq p_\pi^{(t+k)}$.

This follows from the fact that we can compute the probability of reaching a state and action in a transition by computing all the conditional probabilities required to reach it. Given a policy π, environmental transition dynamics $\mathcal{T}$, and the probability of the initial state s_0 being $v_\pi(s_0)$, we can define the probability of observing the following finite trace of state-actions $\{s_0, a_0, s_1, a_1, ..., s_{N-1}, a_{N-1}, s_N\}$ as:

$$\begin{aligned} p_\pi(\{s_0, a_0, s_1, a_1, ..., s_{N-1}, a_{N-1}, s_N\}) &= v_\pi(s_0)\pi(a_0|s_0)p_\mathcal{T}(s_1|s_0, a_0)\dots\pi(a_{N-1}|s_{N-1})p_\mathcal{T}(s_N|s_{N-1}, a_{N-1}) \\ &= v_\pi(s_0)[\prod_{n=0}^{N} \pi(a_n|s_n)p_\mathcal{T}(s_{t+1}|s_t, s_t)], \end{aligned} \tag{3.6}$$

The empirical form of our divergence measure is as follows:

$$M\hat{M}D^2_{weighted}[\pi, \pi_\beta] = \sum_{i=1}^{N}\sum_{j=1}^{N} p_\pi^{(i)} p_\pi^{(j)} k(x_i, x_j) + \sum_{i=1}^{M}\sum_{j=1}^{M} p^{(i)} p^{(j)} k(y_i, y_j) - \sum_{i=1}^{N}\sum_{j=1}^{M} 2p_\pi^{(i)} p^{(j)} k(x_i, y_j) \tag{3.7}$$

We can vectorise this into the following form.

$$MMD^2_{weighted}[\pi, \pi_\beta] = L(p) = p^T K(Y,Y)p + p_\pi^T K(X,X)p_\pi - 2p_\pi^T K(X,Y)p \tag{3.8}$$

where $K(A, B)$ is a matrix containing all the pairwise kernel distances between elements in $A = \{a_n\}_{n=1}^N$ and $B = \{b_m\}_{m=1}^M$ such that $K(A,B)_{ij} = k(a_i, b_j)$.

One thing to immediately notice in Equation (3.8) is that this statistic is quadratic in terms of the buffer priorities p, with Hessian $2K(Y,Y)$. With a standard kernel choice such as a Radial Basis Function (RBF) kernel, $K(Y,Y)$ is well known to be positive semi-definite. It immediately follows that $L(p)$ in *Equation* (3.8) is convex in p and has a unique global minimum point. This hints at the amenability of this statistic to be used for gradient based optimisation of our buffer priorities, which is something we explore in greater detail later in Chapter 4.

Building intuition

For now, we can use the presence of a global minimum to build intuition and sanity check our statistic. We attempt to find the global optimum p^* that minimises this metric directly and explore if the optimal form form minimising this measure of divergence makes intuitive sense. We find this by solving for p when $\nabla_p L(p) = 0$. Note that in practice if using this statistic as an objective function to minimise with respect to p, we would have to adapt this into a constrained optimisation problem since p must describe a categorical probability distribution. We can ignore this for our intuition building exercise.

$$\nabla_p L(p) = 2K(Y,Y)p - 2K(X,Y)^T p_\pi \tag{3.9}$$

$$\begin{aligned} \nabla_p L(p) = 2K(Y,Y)p^* - 2K(X,Y)^T p_\pi = 0 \\ \implies p^* = K(Y,Y)^{-1} K(X,Y)^T p_\pi \end{aligned} \tag{3.10}$$

The solution $K(Y,Y)^{-1}K(X,Y)^T p_\pi$ can be intuitively interpreted, giving us confidence in the statistic. We gain intuition for what this means by building up the solution term by term.

1. $K(X,Y)$ computes the kernel similarity between transitions in the dataset $\{y_i\}_{i=1}^M$ and those from the policy rollout $\{x_i\}_{i=1}^N$. In effect, this quantifies the pairwise similarity between each data point in the offline buffer and in the rollout, and different choices of kernel should yield slightly different levels of leniency towards differences in transitions.

   As a concrete example, using a dot product kernel we have $k(x_i, y_j) = x_i \cdot y_j = |x_i||y_j| \cos\theta$, where θ is the angle between x_i and y_j, the ith rollout transition and the jth transition in the replay buffer. Using this kernel we encode transitions to be similar if their vectorised representations point in a similar direction.

2. Next, $K(X,Y)^T p_\pi$ has dimensionality $\mathbb{R}^M$ and it quantifies the expected total kernel similarity between the rollout using policy π, for each data point $y \in Y$ in the offline buffer, weighted by the probability of reaching each transition in the rollout. The ith component $[K(X,Y)^T p_\pi]^{(i)} = \sum_{n=1}^N k(x_n, y_i) p_\pi^{(n)} = \mathbb{E}_{x\sim\pi}[k(x, y_i)]$.

3. Finally, multiplying by $K(Y,Y)^{-1}$ to obtain $K(Y,Y)^{-1}K(X,Y)^T p_\pi$ helps us to regularise against regions where we have a high density of offline data. Simply using $K(X,Y)^T p_\pi$ can be problematic if we have duplicate or similar transitions in our offline buffer, since we would effectively be independently assigning the same priority to these transitions. The total probability mass at a duplicate transition would therefore be more than it should be to distributionally match the state-action visitation of π. The term $K(Y,Y)^{-1}$ regularises against this.

   Interestingly, $K(Y,Y)$ is an identity matrix in the situation where all of our transitions in the replay buffer are dissimilar to each other according to the kernel in use, and in this situation the priority for a sample in the offline buffer is exactly its expected similarity to all the transitions in the rollout.

This analysis of the p^* that minimises our metric makes intuitive sense since we can summarise it as being the priority assignment that encourages the sampling process of distinct transitions in the buffer to be similar to the distribution of transitions in the rollout.

3.5.2 Experiment

We plot the weighted maximum mean discrepancy of the transitions that are in the replay buffer and the state-action visitation of the argmax policy $\pi(s) = \arg\max_{a\in\mathcal{A}} Q(s,a)$ at each iteration. We do this for both the online and naive offline DQN agents in our GridWorld environments.

We see that the online agent generally exhibits less discrepancy between the on-policy state-action visitation than the offline agent, and this discrepancy declines slightly as training proceeds due to the use

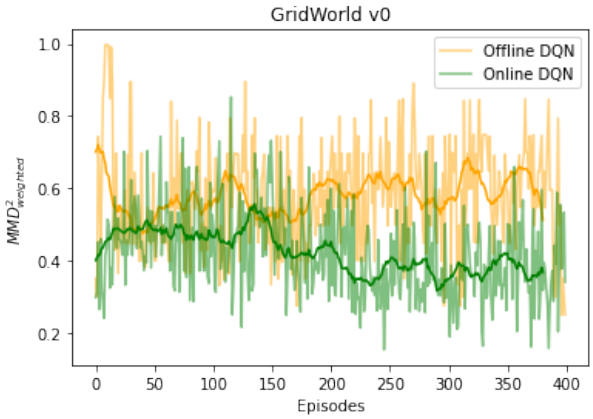


Figure 3.11: $MMD^2_{weighted}$ between the argmax policy rollout and the contents of the data buffer

of a decaying ϵ for our epsilon-greedy exploration and data collection policy. This is in line with what we observed in Figure 3.4, giving us some experimental confidence in our new statistic.

We will make use of this statistic to quantify off-policyness and to motivate optimisation-based solutions to the sample distribution problem.

Chapter 4

Model Aligned Offline Reinforcement Learning (MAORL) (Known Unknowns)

In the previous chapter we analysed and experimentally isolated the problem of misalignment between the on-policy state-action distribution π and the offline buffer sampling strategy π_β. We observed that using uniform random sampling, as is the norm in state-of-the-art offline reinforcement learning methods, can be problematic. In this chapter, we present buffer alignment mechanisms to reduce this misalignment and in doing so, propose a novel offline reinforcement learning algorithm that corrects for this problem.

As part of this effort, we introduce a new general framework for reward-penalty offline reinforcement learning called Model Aligned Offline Reinforcement Learning (MAORL).

4.1 General Framework

We would like to minimise the divergence between the state-action visitation of the policy and the data we sample from the offline buffer. As discussed earlier when defining our state-action visitation divergence metric Equation (3.8), we can (and often must) resort to using samples from policy rollouts and from the offline buffer to assess how large this divergence is if we want to avoid constructing parametric models. While collecting samples from the replay buffer is easy to do, in offline reinforcement learning, by definition we do not have access to the true MDP required to collect model rollout samples.

In order to obtain sample rollouts of the agent, we propose first learning a uncertainty penalised, reward-free MDP $\tilde{\mathcal{M}}$. We can perform rollouts on this approximate MDP instead. $\tilde{\mathcal{M}}$ is completely parameterised by a learned dynamics model (T), out-of-distribution (OOD) detector (U), and reward penalty (k). This is very similar to the pessimistic MDP (P-MDP) proposed in model-based offline reinforcement learning algorithm MOReL [21] and uncertainty penalised MDPs in MOPO [20].

We briefly consider each component:

- **Learned dynamics model** ($T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$): this maps states and actions to the next state, and can be learned through supervised learning on (s, a, s') tuples in the offline dataset $\mathcal{D} = \{(s_m, a_m, r_m, s'_m)\}_{m=1}^{M}$. We use this to conduct rollouts using our policy in order to understand its state-action visitation distribution in order to better align the sampling regime.
- **Out-of-distribution detector** ($U : \mathcal{S} \times \mathcal{A} \to \{\textbf{TRUE}, \textbf{FALSE}\}$): this determines whether a state-action tuple lies outside the support of the training data. In practice, this is not distinct from our dynamics model since we can use epistemic uncertainty estimation techniques to determine if

the dynamics model is uncertain of its prediction (Figure 5.5). In $\tilde{\mathcal{M}}$, taking an action such that $U(s, a)$ is TRUE leads the agent to a terminal HALT state.

- **Reward penalty** ($k \in \mathbb{R}$): when an agent attempts to transition into an OOD region during a rollout, a negative reward is applied to this transition and the transition is incorporated into the sampling process π_β for learning. This is the only time we might learn from a transition not explictly present in the offline dataset under the MAORL framework, and this discourages the agent from learning policies and exploring outside the support of the training data by associating it with negative outcomes, preventing *action distributional shift*.

We will explore the specifics of learning an uncertainty-penalised MDP later in Chapter 5 so that we can initially isolate the alignment task. Provided we have our uncertainty-penalised, reward-free MDP $\tilde{\mathcal{M}}$, we can conduct rollouts with the current policy π. Samples from these rollouts can be used to approximately understand the state-action visitation distribution of the policy, which we can then use to improve our buffer sampling process π_β to better align with this distribution. Finding efficient and intelligent buffer alignment mechanisms is the focus of this chapter.

Since the misalignment problem has not been identified before, algorithms (such as those presented in this project) that address this by using a dynamics model in this fashion define a new class of offline reinforcement learning algorithms distinct from previous definitions of *model-based* and *model-free* offline reinforcement learning. We call these algorithms model *aligned* offline reinforcement learning algorithms (MAORL).

We outline a general framework for MAORL algorithms in Algorithm 2.

Algorithm 2 General Framework for Model Aligned Offline Reinforcement Learning (MAORL)

Require: Dataset $\mathcal{D} = \{s_m, a_m, r_m, s'_m, a_m\}_{m=1}^{M}$
Initialise policy π_θ, initial sampling mechanism π_β
Learn reward-free, uncertainty-penalised MDP $\tilde{\mathcal{M}} = \{T, U, k\}$ where T is a learned transition dynamics model, U is an OOD detector, and k is a reward penalty (Chapter 5)
for episode $\in \{1, .., M\}$ **do**
    $s_{\text{cur}} \leftarrow s_{\text{init}}$
    while episode not done **do**
        Sample action using the current policy $a \sim \pi(s_{\text{cur}})$
        if $U(s, a)$ **then**
            Incorporate transition (s, a, k, HALT) into sampling mechanism π_β
        else
            Execute this action in dynamics model using current policy. $s_{\text{cur}} \leftarrow T(s_{\text{cur}}, a)$
        end if
        Update π_β to increase state-action visitation alignment (Section 4.2, Section 4.3)
        Sample minibatch of transitions $(s, a, r, s') \sim \mathcal{D}$ using current sampling mechanism π_β
        Optimise policy π_θ using any reinforcement learning algorithm
    end while
end for

Distinction from model-based offline reinforcement learning

Note that unlike model-based offline reinforcement learning techniques [20] [21], we specifically do not use transitions from the model rollouts to directly train the policy using a reinforcement learning algorithm. We do not even learn a reward model. After learning an uncertainty aware dynamics model, model-based techniques make no further use of the offline data. Instead, our approach solely uses the dynamics model for reducing the sampling misalignment problem and learns from the ground truth transitions present in the offline buffer. For this reason, we refer to this approach as model-*aligned* offline reinforcement learning (MAORL) rather than model-*based*. We illustrate the difference in Figure 4.1.

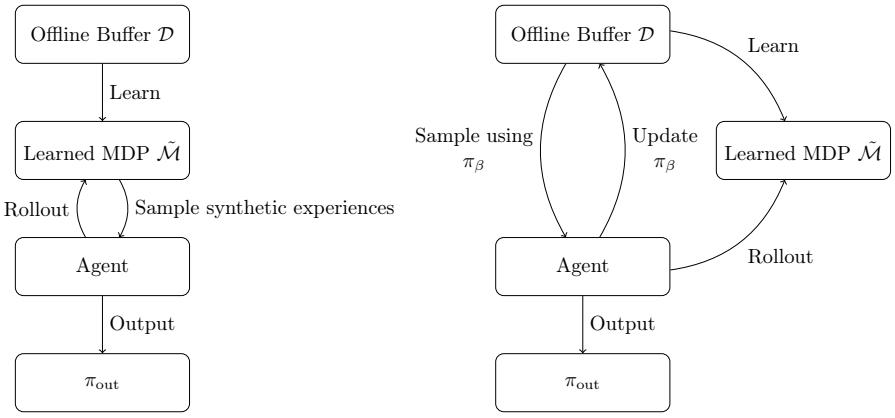


Figure 4.1: The differences between model based offline reinforcement learning (left) and model aligned offline reinforcement learning (right). In model based offline reinforcement learning the learned MDP $\tilde{\mathcal{M}}$ is used directly for policy learning, while in model aligned offline reinforcement learning it is solely used for aligning the sampling process π_β.

We propose two strategies to better distributionally align the data sampling distribution with the policy state-action visitation distribution. In Section 4.2 we describe an ad-hoc method that is inspired by the first-in-first-out queuing mechanism used in typical online reinforcement learning algorithms where the agent is able to explore its environment. In Section 4.3 we define a method that makes use of mathematical optimisation techniques to learn π_β directly as a categorical probability distribution by using the weighted maximum mean discrepency metric defined earlier in Equation (4.1).

4.2 Nearest State-Action (NSA) Queue Alignment

In this approach, we take inspiration from the fixed size, first-in-first-out queue data structures used as buffers in traditional online reinforcement learning. We argued earlier that when combined with exploration, the fixed capacity places an inherent limit on the degree of off-policyness of the transitions being sampled.

Each step in our learned MDP $\tilde{M}$, we generate a *synthetic* transition using the dynamics model. We then scan through the offline dataset to find any transitions that are close within a specified level of tolerance in terms of states, actions, and next states, and insert these *real* transitions from the dataset into a separately maintained buffer queue. The search operation is referred to as `find`. By only inserting close transitions we ensure the queue consists of transitions much closer distributionally to the current policy state-action visitation in our learned MDP $\tilde{\mathcal{M}}$. We can then sample from this queue uniform randomly. Search tolerance is a useful parameter that can be scaled depending on our confidence in the dynamics model.

Algorithm 3 Procedure for NSA replay buffer alignment to update sampling process π_β

▷ $\mathcal{T}$ is the synthetic transition, $\mathcal{Q}$ is the sample queue, $\mathcal{D}$ is the offline dataset
procedure UpdateSamplingProcessNSA($\mathcal{D}$, $\mathcal{Q}$, $\mathcal{T}$, $\epsilon_{\text{tolerance}}$)
    `find` transitions in $\mathcal{D}$ that match $\mathcal{T}$ within tolerance $\epsilon_{\text{tolerance}}$
    Insert the unique transitions into fixed capacity queue $\mathcal{Q}$
    $\pi_\beta \leftarrow$ uniform random sampling without replacement from $\mathcal{Q}$
end procedure

4.2.1 Incorporating OOD transitions

As part of the MAORL framework defined in Algorithm 2, if we encounter an OOD transition, we must include it into the sampling process with a negative reward penalty to discourage learning a policy that

takes the agent outside the training data support. This can be done quite naturally using NSA replay buffer alignment since we sample from a distinct queue to the offline data buffer. We simply insert the synthetic transition that would have required taking an OOD action, with the reward labelled as k, the predetermined reward penalty, and the transition specified as terminal, into this distinct queue ($\mathcal{Q}$ in Algorithm 3). In this way transitions that take the agent outside the support of the training data are included into the sampling mechanism and the agent can learn to avoid these actions.

4.2.2 Relation to Online Reinforcement Learning

It is worth noting that in the limiting case where we have an uncertainty-penalised MDP with no dynamics prediction error and perfect OOD transition detection, we can obtain identical behaviour to an online agent with low tolerance in our `find` operation. With no dynamics error, then there is an exact match for any transition experienced in the learned MDP in the replay buffer. With all else equal, since there is no dynamics error, using NSA Queue alignment will then process transitions the same order as an online agent.

We validate whether this is indeed true by running NSA Queue Alignment with access to the true *GridWorld-v0* environment, but without allowing the agent to explictly insert transitions experienced and instead forcing the agent to use the `find` operation instead. Despite being simply being an adjustment to the buffer sampling mechanism, we find that NSA Queue Alignment vastly outperforms our naive offline baseline, despite having access to the exact same dataset. All learning is performed using the offline dataset in line with the general MAORL framework. We have identical hyperparameters for all agents tested.

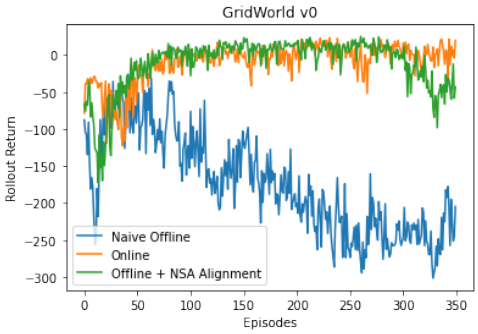
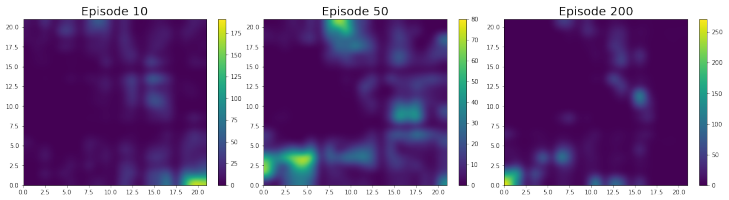


Figure 4.2: Comparing the rollout returns using the true environment during training using NSA Alignment. Ablating NSA Alignment leads to a collapse in the returns being observed.

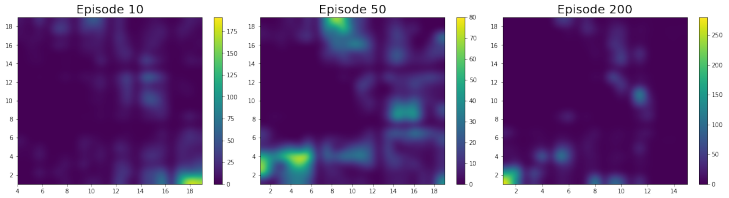
We observe a very similar learning return curve to the online agent. The fact that this agent does not exhibit the previously mentioned *unlearning effect* even despite having access to the same data as the naive offline agent illustrates that exactly correcting for buffer sampling misalignment plugs the gap required to reach performance comparable with online reinforcement learning.

In Figure 4.3, we see virtually the same buffer priority and state visitation distribution as learning progresses, demonstrating the effectiveness of the alignment mechanism in a visual fashion.

This validates our earlier analysis that the buffer sampling distribution misalignment is an essential limiting factor for offline reinforcement learning.



(a)



(b)

Figure 4.3: Figure 4.3a illustrates how the transitions stored in the sample experience replay buffer evolve during training using NSA. Figure 4.3b illustrates how the state visitation of the agent changes.

4.2.3 Implementation Details

When implementing this alignment mechanism in a practical setting, we noted the following implementation details that help improve performance.

- **n-step returns**: in sparse reward problems, using n-step returns with $n > 1$ can be a very useful strategy for speeding up training. Rather than training the action-value estimate $Q(s_t, a_t)$ on the basis of the single-step temporal difference error we use an n-step target with intermediate actions generated in our case by the same policy being used for exploration. The target is $\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_a Q(s_{t+n}, a)$. We support this even if the static dataset consists of single-step traces by finding close aligning transitions in the offline dataset for each of the n consecutive transitions being used to compute the target. This can dramatically improve performance on sparse reward problems where we only have an idea of what the final goal looks like.
- **Bucketing and parallelism in find**: `find`, if applied naively, can be a bottleneck in this approach since we need to query our offline dataset each iteration to determine which transitions to insert into the queue used for policy learning. Fortunately, we can apply two tricks to speed this up:
  1. Due to the fact that we only need the nearest transitions under $\epsilon_{\text{tolerance}}$, we can bin our dataset based on the start state of the transition or another appropriate sharding mechanism in order to speed up `find`. We can then localise our scans in bins of interest. Defining a total ordering on transitions can also enable us to store these bins in sorted order so the final scan can be done via binary search rather than a linear scan.
  2. We can take advantage of the offline dataset's immutability by maintaining a small thread pool for dispatching localised `find` operations at each bin of interest in parallel. The resources provisioned for the thread pool can be scaled depending on the scale of the learning task and the quantity of data.

4.2.4 Drawbacks

While the NSA replay buffer alignment approach Section 4.2 illustrates how we might solve the buffer alignment problem and improves performance over our baseline, it has a scalability bottleneck in the `find` operation that requires intelligent software engineering and ad-hoc performance optimisations to resolve. Furthermore, the hyperparameter $\epsilon_{\text{tolerance}}$ has a major influence on learning, and determining a good value is not intuitive since a good value depends on a complex combination of the offline data density and the accuracy of the predictive model used. We would like to move beyond this procedural approach and aim to derive an efficient, mathematically sound method to achieve buffer alignment.

4.3 Gradient-based Alignment

We would like to mathematically frame the misalignment problem to try and derive a principled, efficient solution. In this approach, each transition in the offline buffer will have an associated priority which reflects its likelihood of being sampled, much like in a Prioritised Experience Replay Buffer [30]. We aim to use optimisation-based techniques to efficiently learn good priority assignments to tackle the buffer sampling misalignment problem by continuously adapting the distribution of priorities alloted to each transition in the offline buffer. By doing this, we hope to move beyond the drawbacks facing the highly procedural NSA replay buffer alignment mechanism.

4.3.1 Optimisation Problem

To approach this, we can make use of the sample-based weighted maximum mean discrepancy statistic we developed in Equation (4.1) to quantify the divergence between the state-action visitation of the policy π and the sampling process π_β. In this statistic, recall that we encode $\pi_\beta \sim \text{Categorical}(p)$, where $p \in \mathbb{R}^M$ encodes the event probabilities of sampling each of the M transitions in the replay buffer. Recall that we can define the statistic in vectorised form as follows:

$$MMD^2_{weighted}[\pi, \pi_\beta] = L(p) = p^T K(Y,Y)p + p_\pi^T K(X,X)p_\pi - 2p_\pi^T K(X,Y)p \tag{4.1}$$

where K is a matrix containing all the pairwise kernel distances between row elements in $X = \{x_n\}_{n=1}^N$ and $Y = \{y_m\}_{m=1}^M$ such that $K(X,Y)_{ij} = k(x_i, y_j)$. Recall that x_n is a vector representation of the nth transition observed during the agent's rollout using the dynamics model, while y_m is an equivalent vector representation of the mth transition in the offline dataset. For a reminder see Equation (3.5).

We can write this statistic as a function in terms of p which we will call $L(p)$, yielding an objective function framed in terms of the buffer sampling priorities. We propose minimising misalignment between the buffer sampling regime π_β and the state-action visitation of the agent by optimising the priorities p in order to minimise this objective. We would expect that the transitions sampled from the replay buffer would be closer distributionally to those that would be visited by the agent for priority arrangements that minimise $L(p)$. This approach can be viewed as a continuous way to align the buffer sampling process with the policy state-action visitation distribution.

It is worth noting that we cannot minimise this loss naively with respect to p in an unconstrained fashion. In order to encode a valid categorical distribution, p must respect two constraints:

- *Non-negativity*: we must ensure that $\forall m \in \{1, .., M\} : p^{(m)} \geq 0$ so we have a valid weighted random sampling mechanism.
- *Sum to a positive constant*: we must ensure that $\sum_{m=1}^M p^{(m)} = C$ for some $C \geq 1$. We will typically set C to be the current size of the buffer. We have unnormalised probabilities because for large datasets, enforcing a sum to 1 could lead to undesirable floating point imprecision when representing very small priorities.

Our optimisation problem can therefore be defined as follows.

$$\begin{aligned} \min_p \quad & L(p) = p^T K(Y,Y)p + p_\pi^T K(X,X)p_\pi - 2p_\pi^T K(X,Y)p \\ \text{s.t.} \quad & \forall m \in \{1, .., M\} : p^{(m)} \geq 0 \\ & 1^T p = C \end{aligned} \tag{4.2}$$

Convexity

One thing to immediately notice in that the objective $L(p)$ is quadratic in terms of the buffer priorities p and has Hessian $2K(Y,Y)$. We also note that with an appropriate choice of kernel such as using a Radial Basis Function (RBF) kernel, $K(Y,Y)$ is well known to be positive semi-definite. Being a quadratic in terms of p and because $K(Y,Y)$ is positive semi-definite, it immediately follows that $L(p)$ in *Equation* (4.1) is convex in p and has a unique global minimum point. This convexity provides us with a strong motivation to consider gradient-based optimisation approaches to find a priority assignment p that minimises $L(p)$. We note that the feasible set of solutions define a simplex $\Delta \subset \mathbb{R}^M$, where $\Delta = \{p \in \mathbb{R}^M | \forall m : p^{(m)} \geq 0 \wedge 1^T p = C\}$. This means that the constraints define a convex set and the overall optimisation problem defined in Equation (4.2) is a convex optimisation problem. This suggests that we can derive a priority assignment p that both minimises the divergence with the state-action visitation of the agent while also being feasible as sampling priorities by using convex optimisation techniques.

We outline the general procedure for aligning our sampling process π_β in Algorithm 4.

Algorithm 4 Procedure for updating sampling process π_β using gradient-based alignment

▷ $\mathcal{T}$ is a queue of the N previously visited transitions in the learned MDP $\tilde{M}$
▷ $\mathcal{D}$ is the offline dataset
procedure UPDATESAMPLINGDISTRIBUTION($\mathcal{D}$, p, $\mathcal{T}$)
    Construct p_π using $\mathcal{T}$
    Solve constrained convex optimisation task set out in Equation (4.2) to find $p^* \in \mathbb{R}^M$
    $\pi_\beta \leftarrow$ sample transitions from the offline buffer $\mathcal{D} = \{y_m\}_{m=1}^M$ under Categorical($p^*, \{y_m\}_{m=1}^M$)
end procedure

4.3.2 Projected Gradient descent to obtain feasible buffer priority alignment

A direct solution to even the unconstrained priority assignment problem has extremely high computational complexity, as discussed later in Equation (4.6). Instead, we derive an iterative, gradient-based update rule to find a good priority assignment more efficiently. Due to the convexity of the objective and of our constraint set, we can be satisfied that any local optima found are actually global optima, making our problem especially amenable to gradient-based approaches.

A naive gradient-descent update rule is not guaranteed to produce feasible intermediate values of p under the categorical distribution constraints set out earlier. We instead perform a *projected gradient descent*, where each iterative step we perform a projection step to bring the updated $\bar{p}$ onto the simplex Δ, where $\Delta = \{p \in \mathbb{R}^M | \forall m : p^{(m)} \geq 0 \wedge 1^T p = C\}$ in order to ensure the final iterate encodes a valid sampling process that can be applied to our problem.

Recall that $\nabla_p L(p)$ has the following form:

$$\nabla_p L(p) = 2K(Y,Y)p - 2K(X,Y)^T p_\pi \tag{4.3}$$

We can now write down the following gradient update rule for updating p_t to p_{t+1}:

$$\begin{aligned} \bar{p} &= p_t + \alpha[K(X,Y)^T p_\pi - K(Y,Y)p_t] \\ p_{t+1} &= \text{proj}_\Delta(\bar{p}) \end{aligned} \tag{4.4}$$

This update has a very intuitive interpretation on closer investigation, making it very appealing.

- $+\alpha K(X,Y)^T p_\pi$ is a term that encourages priorities for those transitions in the data buffer (in Y) to be similar in terms of the chosen kernel k to the recent transitions visited by the policy encoded

in X. This is because $K(X,Y)_{ij}$ will be larger when $X^{(i)}$ and $Y^{(j)}$ are similar. This has very intuitive appeal and directly relates to ideas in the previous queue-based approach Algorithm 3 for promoting alignment. It does this weighted by the state-action visitation distribution under the policy which ensures the sampling priorities reflect the conditional dependencies in the traces they are modelling (since transitions are not independent).

- $-\alpha K(Y,Y)p_t$ is a term that regularises against changing priorities too aggressively in dense data regions, weighted by the previous priority for these transitions. $K(Y,Y)$ is strongest for highly similar transitions and this term down weighs the priority change afforded based on the density of transitions, weighted by the kernel similarity (which is also the same kernel used to promote priorities in the other term). In effect, this term prevents "double counting" transitions.

Projection Choice

Projecting our intermediate vector $\bar{p} \in \mathbb{R}^M$ onto a simplex is a well-studied task in convex optimisation, with numerous proposed algorithms. Formally, the task of finding the Euclidean projection of vector $\bar{p}$ onto our simplex Δ can be specified as follows:

$$\min_{p} \quad \frac{1}{2}||p-\bar{p}||^2 \quad \text{s.t.} \quad \forall m \in \{1,..,M\} : p^{(m)} \geq 0 \quad 1^T p = C \tag{4.5}$$

We use a fast, heap-based algorithm as originally proposed in [31] and further discussed in [27] to find this projection.

This projection operation has computational complexity $\mathcal{O}(M + K \log M)$, where M is the size of the offline dataset and K is the number of non-zero elements in the solution. While M is certainly fixed, the size of K is contingent on two things:

1. The degree to which the agent explores the state-action space during training. If an agent explores more narrowly, it will encounter synthetic transitions that are close to fewer distinct transitions in the offline dataset. As a consequence, an aligned buffer sampling mechanism will yield fewer distinct transitions from the offline dataset, implying we have fewer non-zero elements in our solution priorities.
2. The number of synthetic transitions we are fitting our buffer sampling mechanism to. With fewer transitions, it is likely we will assign non-zero priorities to fewer distinct transitions in the buffer.

We find that K is very maintainable in practice since we align our buffer priorities with a fixed capacity queue of synthetic transitions which leads to limited non-zero transitions.

4.3.3 Incorporating OOD transitions

As part of the MAORL framework (Algorithm 2), if we encounter an OOD transition we must include it into the sampling process with a negative reward penalty to discourage learning a policy that takes the agent outside the training data support. Initially, this may seem impossible to achieve since we encode our sampling process as a categorical distribution over our offline samples, with unnormalised event probabilities represented by a fixed size vector $p \in \mathbb{R}^M$. We provide a solution in Algorithm 5.

Note that $p^{(M+1)}$ encodes the probability of sampling the new OOD transition after insertion. We expect a future optimisation sweep will increase the priority associated with this new transition when necessary, bringing it into the learning process.

Algorithm 5 Procedure for incorporating a new OOD transition (s, a, s') into the gradient-based alignment mechanism

Require: Offline dataset $\mathcal{D} = \{s_m, a_m, r_m, s'_m\}_{m=1}^{M} = \{y_m\}_{m=1}^{M}$

1: **procedure** IncorporateTransition($x = (s, a, s')$)
2: **if** $\max\{k(x, y_m | y_m) \in \mathcal{D}_{\text{OOD}}\} < 1 - \gamma$ (see Section 4.3.3) **then**
3: Insert new transition (s, a, k, s') with reward penalty k into data store
4: Augment priority vector $p \in \mathbb{R}^M$ with a new zeroed entry, such that $p \in \mathbb{R}^{M+1}$ and $p^{(M+1)} = 0$
5: **end if**
6: **end procedure**

Optimisation

After experimentation, we realised that quite often the agent would reach a very similar OOD transition already present in our sampling process from an earlier rollout. We noticed that instead of inserting essentially duplicate transitions into our replay buffer, we could omit all insertions after the first. This is because the next optimisation sweep will promote the sampling priority associated with the similar transition already present in the dataset since that will bring the buffer sampling regime into closer alignment with the state-action visitation of the agent.

We can compute a check by computing the kernel similarity of the proposed transition with the existing OOD transitions in the data store. We use the same kernel as is used in the optimisation procedure. Noting that the kernel outputs values in range $(0, 1]$, if any of these is close to 1, this implies a similar transition already exists in the data buffer and will be promoted in the next optimisation step. We therefore forgo the insertion operation. This explains the if statement on Line 2 of Algorithm 5.

We find that in practice, this optimisation prevents the majority of OOD transition insertions.

4.3.4 Recovering hard alignment in a limiting case

Unconstrained optimum

The fact that $L(p)$ is quadratic and has positive semi-definite Hessian $2K(Y, Y)$ means that a local optimum p^* that minimises L will also be a global minimum point. This means we can derive the optimal priority arrangement for the unconstrained problem (without requiring p to encode a categorical distribution) directly by solving for p when $\nabla_p L(p) = 0$.

Recall from Equation (3.10) that the closed form optimum priority assignment takes the following form:

$$p^* = K(Y, Y)^{-1} K(X, Y)^T p_\pi \tag{4.6}$$

Computing the optimum priority arrangement p^* in this way has very high computational complexity. Noting that Y consists of all M transitions in the data buffer, $K(Y, Y) \in \mathbb{R}^{M \times M}$, which implies the inversion operation $K(Y, Y)^{-1}$ will have computational complexity $\mathcal{O}(M^3)$. This has unacceptable performance implications when dealing with larger and larger datasets and is a scalability bottleneck. Furthermore, naively computing inverses can be numerically unstable, providing additional reasons to stay away from a direct solution. $\mathcal{O}(MN)$ cost can be attributed to the matrix multiplication $K(X, Y)^T p_\pi$, assuming a naive matrix multiplication algorithm is used. The final matrix multiplication has cost $\mathcal{O}(M^2)$. The overall computational complexity of the direct method is therefore $\mathcal{O}(M^3 + MN)$, with the inversion operation dominating since $M \gg N$ typically when working with large datasets.

Relation to online reinforcement learning experience replay sampling in a limiting case

Even though it is computationally not an option to directly compute $K(Y, Y)^{-1} K(X, Y)^T p_\pi$, it is still valuable to briefly explore what it means in a limiting case where it can be computed efficiently. If we

have access to a perfect dynamics model and use a kernel k_δ where $k_\delta(x, y) = 1$ if $x = y$, but 0 otherwise, then the matrix holding the pairwise kernel relationships of the transitions in the data buffer $K(Y, Y)$ is an identity matrix (assuming we have unique data points). In this situation, the closed form optimum has form $K(X, Y)^T p_\pi$.

In a traditional experience replay buffer with fixed capacity N, all transitions have an equal chance of being sampled. To replicate this we set $p_\pi \in \mathbb{R}^N$ to be 1. This means the optimal priority at any given point reduces to:

$$p^{*^{(m)}} = \sum_{n=1}^{N} k_\delta(x_n, y_m) = \text{frequency } y_m \text{ is present in } X \tag{4.7}$$

As a result, the priority assigned to any data point $y^{(m)}$ in the replay buffer is the frequency it has occurred in the last N steps of the agent's rollouts. This is exactly the same as in an online set up when we sample uniform randomly from an experience replay buffer containing the last N transitions experienced by the agent in its exploration rollouts. What this result illustrates that in the limiting case where we have access to perfect model dynamics and use an equality kernel, then we recover identical sampling behaviour to an online agent.

We note that p^* is feasible under the constraints set out earlier Equation (4.2) (non-negativity and summation to C) because $K(Y, Y)$ is an identity matrix in this analysis:

- $K(X, Y)$ and p_π only include non-negative entries, meaning p^* is non-negative since $p^* = K(X, Y)^T p_\pi$. If our dataset has coverage of the start points for rollouts then necessarily $K(X, Y)$ contains at least one positive entry.
- We can project non-negative p^* to to ensure summation to C by rescaling our priorities without any change to our sampling distribution:

$$p^{(i)} = C \frac{p^{*^{(i)}}}{\sum_{m=1}^{M} p^{*^{(m)}}}$$

4.3.5 Convergence, Learning Rate and Kernel Choice

In order to discuss convergence characteristics of our gradient-based optimisation procedure, we will briefly omit the projection operation from our analysis in this subsection. We find experimentally that the size of this projection is usually quite small, meaning that we can gain some insight into convergence characteristics of the overall optimisation procedure just looking at the unprojected update.

We can see that the unprojected update rule is in fact a non-homogeneous first order matrix difference equation of form $x_{t+1} = Ax_t + b$. We can rearrange terms to obtain the following:

$$p_{t+1} = \underbrace{(I - \alpha K(Y, Y))}_{A} p_t + \underbrace{\alpha K(X, Y)^T p_\pi}_{b} \tag{4.8}$$

The steady state point occurs when $p_{t+1} = p_t = p_{\text{steady}}$ as $t \to \infty$. We can validate that the steady point reached by iterating this relation to convergence is in fact the optimum point p^* from earlier. Using the fact that $p_{t+1} = p_t \wedge p_t = p_{\text{steady}} \implies p_{\text{steady}} = (I - A)^{-1} b$:

$$\begin{aligned} p_{\text{steady}} &= (I - I + \alpha K(Y, Y))^{-1} \alpha K(X, Y)^T p_\pi \\ &= \alpha^{-1} K(Y, Y)^{-1} \alpha K(X, Y)^T p_\pi \\ &= K(Y, Y)^{-1} K(X, Y)^T p_\pi \\ &= p^* \end{aligned} \tag{4.9}$$

We can establish that stability of this first-order matrix difference equation (i.e. that p_t converges asymptotically to p^*, the steady state optimum). We use p^* to rewrite the equation in homogeneous form.

$$p_t - p^* = A(p_{t-1} - p^*) = (I - \alpha K(Y,Y))(p_{t-1} - p^*) \tag{4.10}$$

Unfolding the relation, we get the following equation for $p_t - p^*$:

$$p_t - p^* = A^t(p_0 - p^*) = (I - \alpha K(Y,Y))^t(p_0 - p^*) \tag{4.11}$$

From this we can see that in order to have p_t converge to p^* as $t \to \infty$, we need $(I - \alpha K(Y,Y))^t \to 0$ as $t \to \infty$. Given $I - \alpha K(Y,Y)$ is diagonalisable by construction using a symmetric kernel, we can rewrite the above using its eigendecomposition $A = P\text{diag}(\lambda_A^{(1)}, \ldots, \lambda_A^{(M)})P^T$:

$$p_t - p^* = A^t(p_0 - p^*) = P\text{diag}(\lambda_A^{(1)^t}, \ldots, \lambda_A^{(M)^t})P^T(p_0 - p^*) \tag{4.12}$$

To ensure stable convergence we therefore this we must ensure that all of the eigenvalues of $I - \alpha K(Y,Y)$ have absolute value less than 1 so that when repeatedly multiplied, they decay towards 0; we can use this restriction to guide our choice of learning rate α. Noting that $A = I - \alpha K(Y,Y)$, by linearity we have the fact that the eigenvalues of A λ_A are related to the eigenvalues of $K(Y,Y)$, $\lambda_{K(Y,Y)}$, as follows:

$$\lambda_A = 1 - \alpha\lambda_{K(Y,Y)} \tag{4.13}$$

To guarantee asymptotic convergence we require every eigenvalue of $I - \alpha K(Y,Y)$ to have absolute value less than 1. Taking an arbitrary eigenvalue of A, λ_A:

$$\begin{aligned}
&|\lambda_A| < 1 \\
&\quad \iff |1 - \alpha\lambda_{K(Y,Y)}| < 1 \\
&\quad \iff 1 - \alpha\lambda_{K(Y,Y)} < 1 \ \ \& \ \ 1 - \alpha\lambda_{K(Y,Y)} > -1 \\
&\quad \iff 0 < \alpha\lambda_{K(Y,Y)} \ \ \& \ \ \frac{2}{\lambda_{K(Y,Y)}} > \alpha \\
&\quad \iff 0 < \alpha < \frac{2}{\lambda_{K(Y,Y)}} \quad (\lambda_{K(Y,Y)} > 0 \text{ by positive definiteness of } K(Y,Y))
\end{aligned} \tag{4.14}$$

Given this constraint on fixed learning rate α actually applies for all eigenvalues of $K(Y,Y)$, we can safely write the following bound to guarantee stable convergence with a fixed learning rate:

$$0 < \alpha < \frac{2}{\lambda^{\max}_{K(Y,Y)}} \tag{4.15}$$

This gives us bounds on what fixed learning rate α we can set in order to guarantee convergence, and illustrates how our choice of kernel and the structure of the data being used can influence learning which can be useful for practitioners. The intuition gained here suggests that our choice of learning rate depends on the conditioning of $K(Y,Y)$, which in practical terms can be influenced by the spread of the data and the choice of kernel. A useful practical preprocessing step for speeding up learning might be to *deduplicate* entries in our offline buffer. In our practical implementation, we use Adam [32] instead of a fixed learning rate but use the intuition gained here to guide our choice of kernel and learning rate.

Iteration starting point (p_0)

We can build some quick intuition on what good starting points p_0 are. This is especially valuable since we will need to do this iterative update every time we want to realign priorities and so tricks for accelerating learning are valuable. Analysing the closed-form optimal priority for the unconstrained problem $p^* = K(Y,Y)^{-1}K(X,Y)^T p_\pi$, we notice a couple of things: $K(Y,Y)$ is unchanged between priority update runs unless an OOD transition has been encountered and our approximation to p_π is unchanged too. The change in optimal priority can be quantified as follows in this situation:

$$\delta p^* = p_2^* - p_1^* = K(Y,Y)^{-1}[K(X_2,Y) - K(X_1,Y)]^T p_\pi \tag{4.16}$$

Note that X_1 and X_2 encode the synthetic transitions seen in subsequent steps of a rollout in batch form. As a consequence, many rows of X_2 are the same as X_1, only shifted down a few rows. With X_2 typically being not vastly dissimilar from X_1, our intuition tells us that the new optimum point will be in the vicinity of the old one, meaning δp^* is not particularly large. Therefore we set the initial priority assignment p_0 to be the optimum point found in the previous alignment optimisation. While the analysis of this is admittedly imprecise, we observe that this works well experimentally as illustrated in Figure 6.8.

4.3.6 Amortising the cost of maintaining kernel matrices

Naively computing the kernel matrices each time we need to align our buffer sampling mechanism can quickly become expensive due to their size. We include the following optimisations in our practical implementation to amortise the cost across multiple priority optimisations:

- Caching $K(Y,Y)$: we observe that the offline dataset is only augmented when a dissimilar OOD transition is encountered, meaning Y rarely changes. We can lazily only recompute $K(Y,Y)$ when this occurs rather than constructing it from scratch every time we want to align buffer priorities.
- Piecewise computation of $K(X,Y)$: computing $K(X,Y)$ from scratch each transition has a computational cost on order of $\mathcal{O}(MN)$ where M is the number of transitions in the offline dataset and N is the number of transitions in the rollout, since we need to compute $k(x,y)\ \forall (x,y) \in X \times Y$. We can instead compute each row of $K(X,Y)$ piecewise as each new rollout transition x arrives, and stitch these together to recover the overall matrix $K(X,Y)$. Computing $K(x,Y)$ for a singular transition x has cost $\mathcal{O}(M)$. Stitching together N rows $K(x,Y)$ in place (without additional memory allocation) has computational cost $\mathcal{O}(N)$. This means that each step, computing $K(X,Y)$ incurs a cost of $\mathcal{O}(M+N)$ rather than $\mathcal{O}(MN)$ as it would by naively computing $K(X,Y)$ from scratch each update. Carefully prealloating memory for this matrix can yield significant speed improvements.

4.3.7 Further Implementation Details

- In sparse reward problems, we can ensure scarce but important high reward transitions are oversampled by encoding our vector p_π to increase the priority assigned to high reward transitions. We can do this by setting $p_\pi^{(i)}$ to be larger if the ith transition $x^{(i)}$ has a high reward. This encourages higher priorities to be assigned to similar transitions in the offline buffer. Selectively boosting priorities has been previously explored in the context of prioritised experience replay buffers [30].
- Initially having a higher learning rate when the number of transitions seen in rollouts is small can be useful for expediting training. We find a linearly decaying learning rate schedule, when combined with Adam, works well across all tasks assessed:

  $$\text{learning rate } (\alpha) = \max\{0.001, 0.1(1 - \frac{N}{C})\}$$

  where N is the number of transitions collected from synthetic rollouts and C is the target for the sum of the priorities.

Chapter 5

Learning and Using Uncertainty Penalised Markov Decision Processes (Unknown Unknowns)

Previously in our construction of the MAORL framework Section 4.1, we mention the construction of a reward-free, uncertainty-penalised MDP that penalises out-of-distribution transitions. In this chapter, we will define what this is based on work done in [20] and [21] and discuss a practical approach to learning this. We will propose the use of Spectral-normalised Gaussian Processes (SNGPs) [24] to construct an uncertainty-aware dynamics model

5.1 Uncertainty-penalised Markov Decision Process

We first define how we will construct our uncertainty-penalised MDP and how this is integrated within the MAORL framework as set out in Algorithm 2. Our definition is very similar to the proposal in MOReL [21] of a *pessimistic* MDP (P-MDP), in that reward-penalties are applied to transitions outside the support of the training data through an uncertainty-based unknown state-action detector. However, it distinguishes itself from P-MDPs since we are not learning a reward model since we are only using the predictive model to *align* our sampling process with the current state-action visitation of the agent rather than to *learn* from synthetic transitions as in model-based offline reinforcement learning methods.

An uncertainty-penalised, reward-free MDP $\bar{\mathcal{M}}$ can be parameterised as $\{T, U\}$, where $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a predictive dynamics model of the environment mapping states and actions to the next state reached. This is something that needs to be learned in an offline reinforcement learning context since we do not have access to the true MDP. U is an out-of-distribution state-action detector, and is needed to be able to determine which actions take the agent outside the support of the training data so they can be penalised.

$$U(s,a) = \begin{cases} \text{TRUE if } (s,a) \text{ is OOD} \\ \text{FALSE otherwise} \end{cases} \tag{5.1}$$

We illustrate how the state-action space $\mathcal{S} \times \mathcal{A}$ is partitioned by the OOD detector U in Figure 5.1.

When conducting a rollout during training using the dynamics model T, if we encounter a transition (s, a, s') such that $U(s, a) = \text{TRUE}$, we then incorporate this transition into our sampling process for the future, augmented with a reward penalty. This reward penalty is the only instance that we will use a reward not directly accessed from the offline dataset and is done in order to discourage the agent from learning a policy that takes it away from the support of the training data. We also make OOD transitions terminal due to potential degradation of the dynamics model in high uncertainty regions.

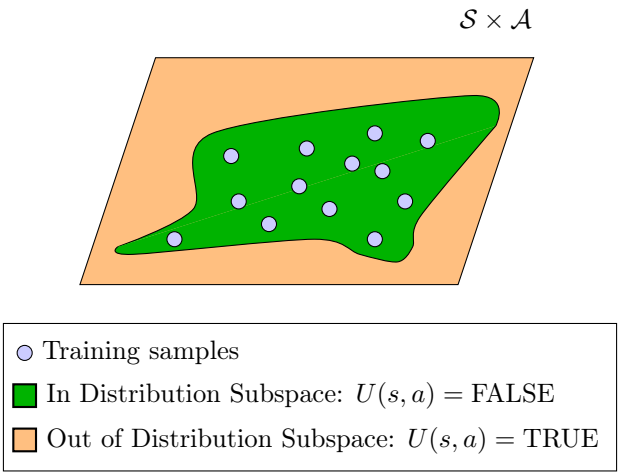


Figure 5.1: Representation of our uncertainty penalised learned environment

We can write down an augmented transition function T_u induced by this as follows:

$$T_u(s'|s,a) = \begin{cases} \delta(s' = \text{HALT}) \text{ if } U(s,a) = \text{TRUE} \\ T(s'|s,a) \text{ otherwise} \end{cases} \tag{5.2}$$

A practical challenge is determining which transitions are outside the support of the training data. In related work such as MOPO [20] and MOReL [21], the primary way this is tackled is by using ensemble networks to estimate the epistemic uncertainty of the predictive dynamics/reward model.

5.2 Using Spectral-normalised Gaussian Processes (SNGPs) to learn an uncertainty-penalised MDP

We argue that SNGPs can be used to represent an uncertainty-penalised MDP, $\mathcal{M} = \{T, U\}$, on their own:

- An SNGP is fundamentally a residual neural network, which means that it can model complex, non-linear functions when trained using supervised learning. We have access to a static offline dataset $\mathcal{D} = \{s_m, a_m, r_m, s'_m, a_m\}_{m=1}^{M}$, meaning we can learn a predictive dynamics model T that maps state-action tuples (s, a) to δ, the change in state required such that the next state $s' = s + \delta$.
- A trained SNGP is able to quantify the amount of epistemic uncertainty associated with a new prediction σ^2, since the typical dense output layer is replaced with a linear approximation to a Gaussian Process with RBF kernel. We can apply a hard threshold on the epistemic uncertainty associated with the prediction of a state-action tuple to determine whether it is outside the support of the training data, since the epistemic uncertainty associated with an input is related to the distance of the input from the data trained on [24].

Given $\text{SNGP}(s, a) = (\delta, \sigma^2)$ for an arbitrary state-action tuple $(s, a) \in \mathcal{S} \times \mathcal{A}$:

$$\begin{aligned} T(s,a) &= s + \delta \\ U(s,a) &= \begin{cases} \text{TRUE if } \sigma^2 > \epsilon \\ \text{FALSE otherwise} \end{cases} \end{aligned} \tag{5.3}$$

We must select ϵ as a uncertainty threshold beforehand.

In MOReL, uncertainty quantification in the P-MDP is done using boostrap ensembles. In this setup learn multiple dynamics models $\{f_{\theta_1}, f_{\theta_2}, \dots\}$ with different weight initialisations, and a state-action tuple is determined OOD if $\max_{i,j} ||f_{\theta_i} - f_{\theta_j}||_2 >$ some threshold. While this is a reasonable technique, we attempt to introduce SNGPs to perform a similar task for two reasons:

1. SNGPs have lower memory footprint and lower computation costs associated with both inference and training. This is because ensemble-based approaches require the training of multiple models and inference requires multiple forward passes. While these considerations appear very practically minded, it is worth bearing these things in mind to make the application of offline reinforcement learning methods useful in practice especially for deployment in real-time applications.

2. SNGPs have more conceptual grounding in how the epistemic uncertainty is calculated, with derivation based on replacing the final dense layer of a neural network with an approximation to a Gaussian Process with a Radial Basis Function kernel and introducing operations that preserve a notion of distance in the latent space of the neural network that is bounded with respect to distance in the input (state-action) space. This theoretical grounding gives us very good insight into both the strengths and limitations of using SNGPs to construct our predictive model.

5.2.1 Developing and open sourcing a PyTorch SNGP library

Our first practical task was to develop a reference PyTorch SNGP library, something that was distinctly lacking reflecting the novelty of the approach. Our implementation[0] has been open-sourced in order to enable greater activity using SNGPs for Machine Learning practitioners that use PyTorch.

We provide model implementations, trainers and utilities with minimal dependencies to enable easy integration into new projects. Our trainers abstract away the details of managing the SNGP precision matrix in the training process and provide checkpointing alongside save/load features.

Accommodating regression tasks

The SNGP was developed as a method to estimate predictive uncertainty for deep *classifiers* [24]. As such, both the original paper and the official Tensorflow[1] implementation only make provisions for classification tasks. We found that there was little standing in the way of enabling SNGPs to work with regression tasks and so we include the option to do so not present in the official implementation.

We replace the use of cross-entropy loss to minimise the negative log likelihood $-\log p(\mathcal{D}|\beta)$ with mean square error when computing a maximum a posteriori estimate of output layer weights β given training data $\mathcal{D}$, which is required during SNGP training. This adjustment enables us to learn uncertainty-aware environment dynamics, which is fundamentally a regression problem. While being an admittedly small addition, as far as we are aware our library is the first to support this out of the box and we have been able to verify that SNGPs perform expectedly well on regression tasks.

Results using synthetic data

We verify our implementation using synthetic data. All of these results are available publically[2].

We first attempt to classify the synthetic *Two Moons* scikit-learn dataset[3]. As can be seen in Figure 5.2, we can correctly classify the two classes while recovering good epistemic uncertainty predictions by having high uncertainty away from the training data support.

[0] https://github.com/adityagoel4512/Spectral-normalized-Neural-Gaussian-Process-PyTorch
[1] https://github.com/google/uncertainty-baselines/blob/c4b52ea74cd83a1d99d775ed4ff597852c3c41c4/baselines/cifar/sngp.py
[2] https://github.com/adityagoel4512/Spectral-normalized-Neural-Gaussian-Process-PyTorch/sngp.ipynb
[3] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html

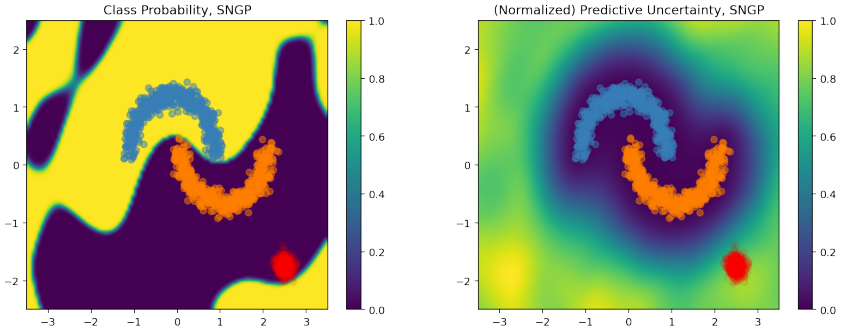


Figure 5.2: Two Moons classification using SNGP

We then consider regression tasks. The simplest task considered was a one-dimensional input Figure 5.3 task.

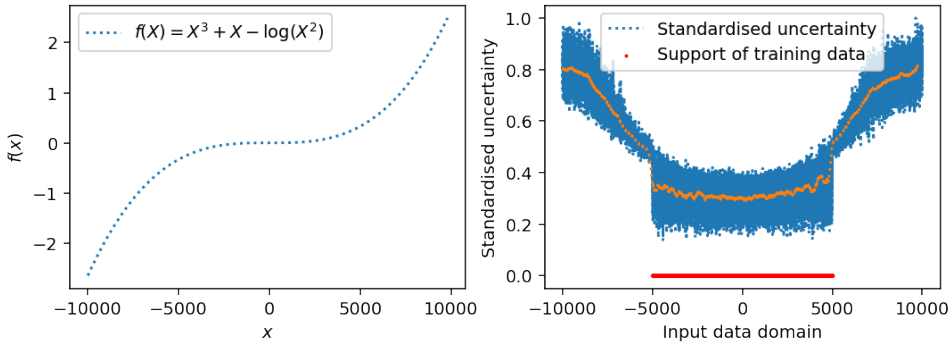


Figure 5.3: One dimensional input regression task

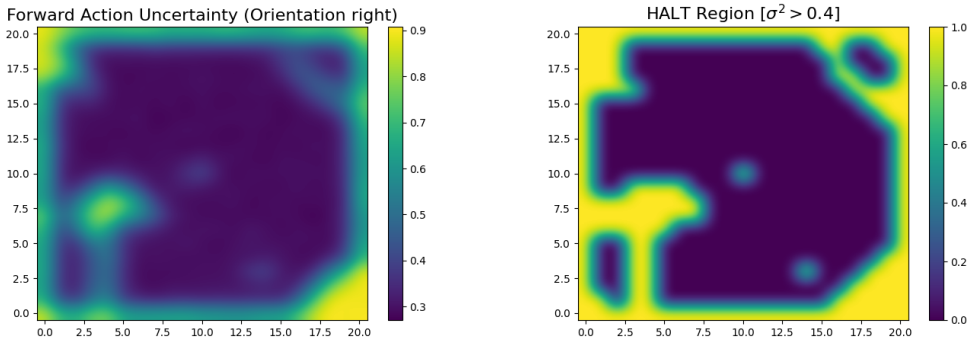
We can see in Figure 5.3 that the model's assessment of uncertainty grows consistently as we supply inputs farther away from the training data support in red. This is encouraging since we want to use uncertainty from our learned transition dynamics model in order to construct our OOD detector U.

5.2.2 Learning uncertainty-aware dynamics models

Having developed a SNGP library, we use this library to model the dynamics of each of the GridWorld environments constructed for testing. Note that we do not include transitions that end at a Lava state in constructing the dataset used for supervised learning. In this way, Lava states simulate our OOD regions and the model is forced to actively discover them by associating them with high uncertainty. We can assess this by comparing the predictive uncertainty and HALT states to the spread of Lava states in the original environment. We note that this is an especially difficult task for ensemble approaches since there is a sharp drop to zero in the density of training data moving into the Lava region.

We visualise the OOD HALT regions and the predictive uncertainty for both GridWorld environments in Figure 5.4 and Figure 5.5. The HALT regions are determined by thresholding the predicted epistemic uncertainty at appropriate values for each environment.

In Figure 5.6 we depict the predicted displacement for executing a forward moving action when in all different possible orientations $\{\text{North}, \text{South}, \text{East}, \text{West}\}$ at each state in GridWorld-v0 to depict what the dynamics predictions look like in the learned uncertainty-penalised MDP.



(a) Predictive uncertainty from SNGP

(b) Terminal HALT regions

Figure 5.4: GridWorld-v1: predictive uncertainty and distribution of HALT states from thresholding predictive uncertainty σ^2 at 0.4

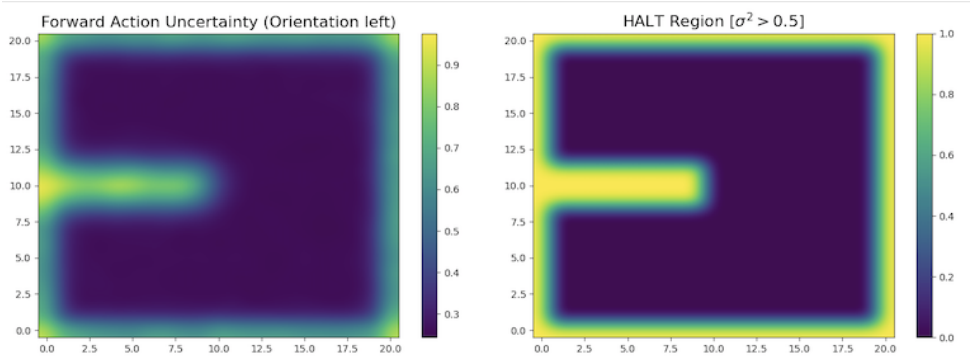
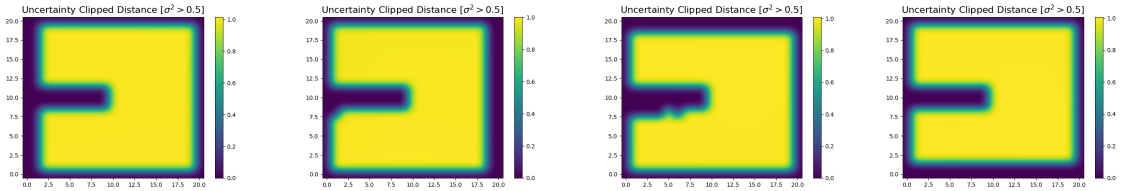


Figure 5.5: GridWorld-v0: predictive uncertainty and distribution of HALT states from thresholding predictive uncertainty σ^2 at 0.5



(a) Orientation left, moving forwards

(b) Orientation right, moving forwards

(c) Orientation up, moving forwards

(d) Orientation down, moving forwards

Figure 5.6: GridWorld-v0: change in position (δ) according to learned uncertainty-penalised MDP

Chapter 6

Evaluation

In this section we evaluate our iterative, gradient-based alignment mechanism.

6.1 GridWorld Results

We constructed multiple sparse-reward GridWorld environments with varying challenges to assess the success of our algorithm. Model architecture details required to replicate results found here can be found in Appendix A and further detailed description of the GridWorld environments can be found in Appendix C.

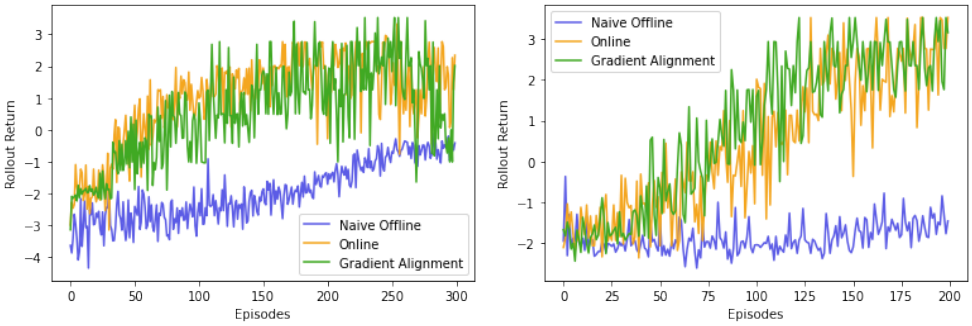
6.1.1 Methodology

Our baseline is an idealised offline DQN agent with access to all reward penalised OOD transitions in its replay buffer. This baseline is much stronger than a completely naive application of the DQN algorithm without exploration since the agent should not suffer from *action distributional shift* as every action the agent can bootstrap from is covered in the offline dataset. The Lava regions of the GridWorlds encode the OOD regions effectively. Transitions into Lava are terminal and receive a negative reward penalty.

We construct our OOD state-action detector using a Spectral-normalised Gaussian Process (SNGP) trained to predict the dynamics of the environments. The training data provided here *excludes* transitions that terminate at a lava state. This means our alignment approach does not have access to OOD transitions unlike our baseline, imitating the realistic set up in practice where we don't explictly *know what we don't know* when deploying an offline reinforcement learning algorithm using a static dataset.

6.1.2 Training Return Log

The immediate thing to notice above is how far behind our offline baseline agent is in terms of the return it receives over the course of training. Bearing in mind that the gradient-based alignment agent must discover OOD transitions as defined by the SNGP, this is clearly a massive improvement. We see returns almost on par with an online agent with the same hyperparameters which has been included for comparison, despite never exploring the true MDP.



(a) GridWorld-v0 (b) GridWorld-v1

Figure 6.1: Cumulative undiscounted rollout returns averaged over 7 distinct seeds

6.1.3 Final Policy Performance

In line with the approach used in the literature, we determine the final policy performance to be the cumulative undiscounted return achieved by the argmax policy at the end of training. We take a mean over 7 training runs.

Task	Naive Offline RL	Gradient Alignment	Online
GridWorld-v0	-2.949	-0.428	-0.174
GridWorld-v1	0.0	0.42	0.39

We can see a clear gap between naively applying reinforcement learning without exploration (even with terminal OOD transitions included in the static buffer) and when using our proposed gradient-based alignment mechanism. This suggests that, just as we assessed in Chapter 3, mitigating action distributional shift is necessary but not sufficient for addressing the performance gap with online methods.

There is a smaller gap towards our online DQN agent's performance. The performance gap here can be partly explained by slackness in the optimisation procedure. Using a gradient-based, iterative approach to buffer priority assignment we will naturally not necessarily arrive at the precise optimum priority each iteration in order to speed up training. One possible avenue for future research is to investigate new optimisation strategies and objective functions for gradient alignment. A second explanation could be predictive inaccuracy in the dynamics model.

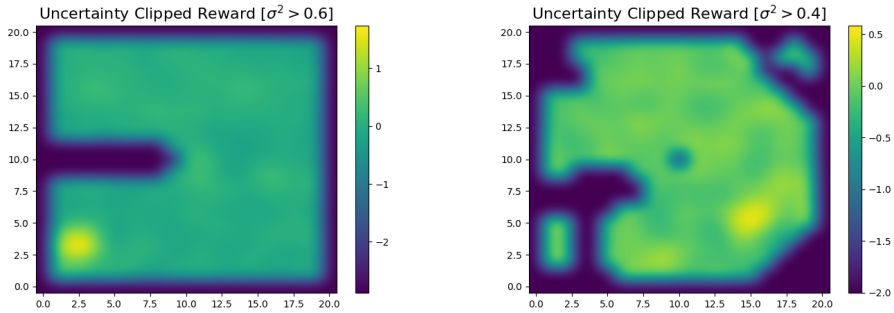
6.1.4 Comparison to model-based offline reinforcement learning methods

In Figure 4.1, we depict the fundamental differences between model-*based* and model-*aligned* approaches to offline reinforcement learning, the biggest of which is that within the MAORL framework we do not learn a reward model and solely use the transition dynamics model to obtain samples in order to improve alignment of the buffer sampling mechanism with the state-action visitation of the agent.

One of the core challenges with model-*based* methods in general is the need for errors in the predictive reward model to be sufficiently small that they are not exploited by the learning agent. We wanted to assess whether model-*aligned* approaches would suffer in a similar way, especially in a sparse-reward regime where only the goal states are specified through positive rewards.

We construct a P-MDP as in MOReL [21] for our model-based offline agent. The MOReL framework leaves open the choice of uncertainty-based OOD detector - in order to ensure a fair comparison with our model-*aligned* agent, we use an SNGP's predictive uncertainty for this purpose. High uncertainty transitions have negative reward penalty and are terminating as specified by MOReL. We then run the same DQN algorithm as our online agent using this learned P-MDP.

We find that despite having an accurate reward model and dynamics model, the agent ends up



(a) GridWorld-v0 ($\sigma^2 > 0.6$) (b) GridWorld-v1 ($\sigma^2 > 0.4$)

Figure 6.2: Predictive reward models with reward clipping when predictive uncertainty is high

exploiting local optima induced by small positive predictive errors. We also find that the model-based offline agent displays a significantly greater variance in the location it reaches. In Figure 6.3, we illustrate that the agent not only finds a local optimum point that is far from the goal consistently, but does so in different ways as illustrated by the high variance in final distance over multiple training runs. Considering both the model-aligned (MAORL) and model-based (MBORL) agents have access to identical dynamics models in this experiment and use the same hyperparameters, it is clear that the use of learned reward models especially in sparse reward domains can be problematic.

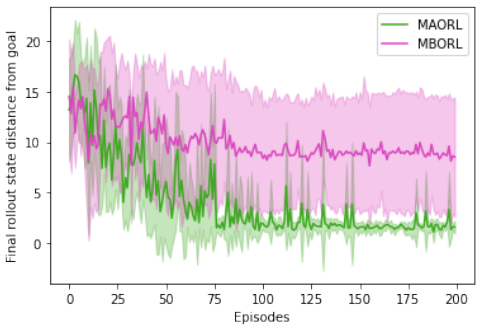
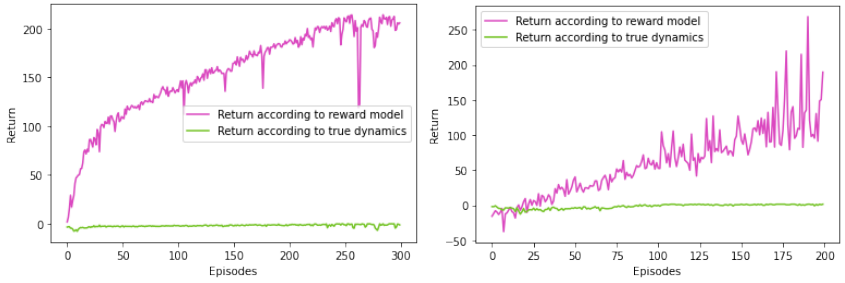


Figure 6.3: Distance of the final rollout position of the agent from the goal state over the course of training for MAORL and MBORL agents (7 training runs using GridWorld-v1)

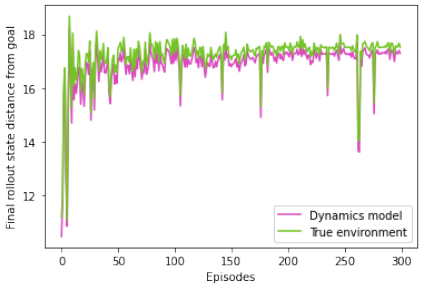
We believe that this is because in sparse reward regimes, any small errors in the predictive reward model can be heavily exploited in the absence of a strong overriding reward signal. With the ground truth reward being 0 for most transitions, small errors positive errors can be exploited if the agent does quickly find the stronger positive reward signal at the goal state while exploring.

This is reflected in the fact that according to the reward model, the agent accrues a very positive reward while in fact executing the trajectory using the true MDP yields very little (Figure 6.4a and Figure 6.4b). In Figure 6.4c and Figure 6.4d we can see that the problem is not to do with the dynamics model since the distance of the final rollout state to the goal according to the true MDP position to the goal state is very similar to that predicted by the dynamics model.

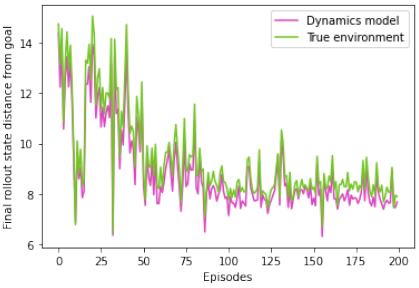


(a) GridWorld-v0 return

(b) GridWorld-v1 return



(c) GridWorld-v0 rollout distance



(d) GridWorld-v1 rollout distance

Figure 6.4: Undiscounted return and mean distance from goal state at end of rollout as assessed by the dynamics model and by the true MDP (averaged over 7 training runs)

Explaining the resilience of MAORL

The issue facing model-*based* methods is that they are effectively replacing clean, ground truth data with a predictive model that is guaranteed to have small amounts of error. In sparse reward tasks where we might only get a strong positive reward at the goal state, this is a challenge since the agent might exploit a local optima created by small prediction error before encountering the desired goal reward signal.

In MAORL, we instead learn from the exact ground truth data and simply do not learn a reward model. In this way, we can expect less error/noise in the reward signals used for training our agent. Using the same dynamics model and environment, MAORL yields much better results.

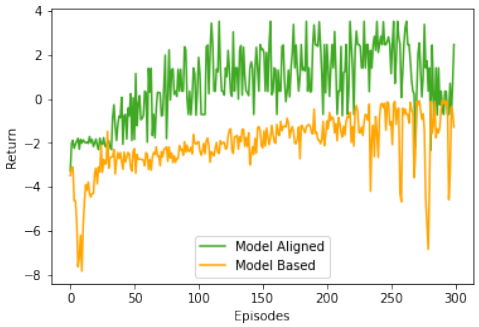
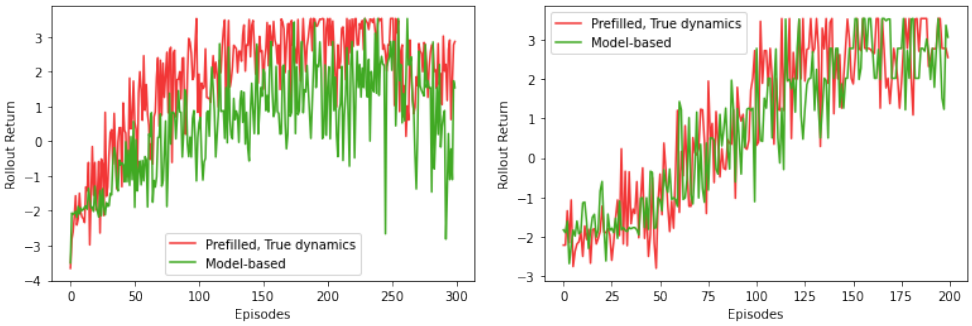


Figure 6.5: Comparing MAORL using gradient-based alignment to using a P-MDP from the MOReL approach to model-based offline reinforcement learning (GridWorld-v0).

6.1.5 Prefilling OOD transitions

In reinforcement learning with exploration, one benefit is that the agent can receive negative reinforcement from bad transitions explored as well as positive reinforcement, since this helps refine the search space of the policy away from poor return strategies. Similarly, in the MAORL framework, the learning agent also needs to explore in the learned uncertainty-penalised MDP in order to discover OOD actions to avoid.

We investigate whether having a-priori knowledge about knowledge gaps in the dataset can be used to improve performance or accelerate training when paired with a model-alignment mechanism. We aim to do this by prefilling our replay buffer with OOD transitions ahead of time rather than requiring the agent to experience them dynamically through exploration. We know from our earlier experiments with our offline baseline that this type of reward shaping is not sufficient always to reach parity with an online agent, and so here we will really be assessing how our buffer alignment mechanism incorporates *useful* OOD transitions into the sampling process during training.



(a) GridWorld-v0 (b) GridWorld-v1

Figure 6.6: Cumulative undiscounted rollout returns averaged over 7 distinct seeds

As seen in Figure 6.6, we find that the agent generally learns faster with advance access to OOD transitions even on our relatively confined GridWorld environments. The return is also more consistent across runs than when OOD transitions must be discovered, leading to a higher mean return.

Practical takeaway

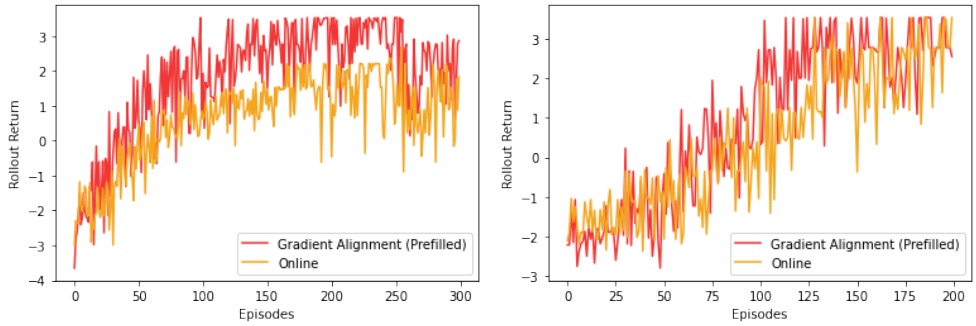
This informs us as practitioners that in practical settings where we may have some partial a-priori knowledge about the data sources used to collect our diverse offline datasets, including this knowledge by proactively populating our dataset ahead of time with examples of transitions we know are not included with reward penalties can accelerate learning within the MAORL framework. One area to explore in future work is how to automatically generate these OOD examples ahead of time. One practical, unsophisticated strategy might be to sample uniform randomly across a subset of the input domain to our OOD detector U and insert transitions where $U(s, a) = \text{TRUE}$. We expect adversarial learning approaches such as GANs to be an exciting avenue for future exploration in the area of OOD example generation.

Could it improve online learning?

We note that the previous analysis about the need to explore to find negative reinforcement applies to online algorithms as well. We investigated how using model-alignment in an offline framework with access to the true MDP compares to an online agent in our GridWorld environments. Our model-aligned approach effectively has the entire visitable state-action space collected into a buffer and uses gradient-based alignment rather than online exploration to tune what gets sampled.

We find that for our GridWorld set ups, we arrive at a good strategy slightly faster than when using online learning despite all hyperparameters being exactly the same. Especially noticeably on GridWorld-v0, the mean return is distinctly higher across runs. This reflects the fact that the agent more consistently reaches the goal state which contains the sparse positive reward signal while there is higher variance across runs using the online agent. GridWorld-v0 is an especially challenging task due to the sparsity of positive reward signals and the Lava barrier.

We hypothesise that the reason for the slightly faster convergence at a goal reaching strategy is because gradient-based buffer alignment uses a smooth kernel (in these experiments we use an RBF kernel), it smoothly incoporates transitions that may not have been visited as yet but which are similar to those that have been visited into the training process. This can present small improvements in training speed



(a) Cumulative returns for rollouts for GridWorld-v0

(b) Cumulative returns for rollouts for GridWorld-v1

Figure 6.7: Cumulative undiscounted rollout returns averaged over 7 distinct seeds

in sparse reward environments since it means we might sample a distinct and useful reward signal earlier than we otherwise might if we had to exactly visit the associated state. We leave it to future work to investigate this in greater detail.

6.1.6 Moving initial point

One possible drawback of our proposal is the additional time required to align our buffer priorities compared to using uniform random sampling as in our offline baseline. We perform a projected gradient descent every time we want to sample from the offline buffer.

We assess whether the moving initial point p_0 we introduced as an optimisation in Section 4.3.5 actually improves training speed by reducing the time taken to converge. We proposed that a good initial point p_0 for the iterative optimisation of buffer priorities is the optimum point found in the previous optimisation procedure. As can be seen in Figure 6.8, after the first priority optimisation has completed, the time taken to conduct priority optimisations significantly decreases when we start the next optimisation procedure at the previously reached optimum rather than using a fixed initial point (that represents a uniform distribution over the training samples).

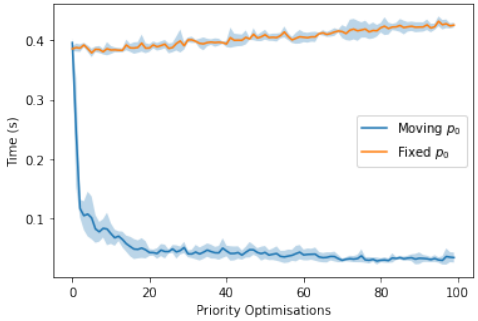


Figure 6.8: Time taken for buffer priority optimisation when using moving initial point p_0 (Section 4.3.5) compared to constant p_0 initialised to encode uniform random distribution with equal priorities at all elements. Results taken over 5 training runs on a machine with Intel(R) Xeon(R) W-2145 CPU at 3.70GHz and NVIDIA GeForce RTX 2080 GPU on a task using GridWorld-v1.

This also emprically validates our earlier assessment that each subsequent assignment optimum point is indeed close to its predecessor in the space of priority assignments.

6.1.7 Qualitatively assessing alignment

We find that qualitatively, this approach succssefully aligns buffer priorities with the state-action visitation of the agent. We can visualise this for our 2D GridWorlds through snapshots in the training process. One such snapshot is illustrated below:

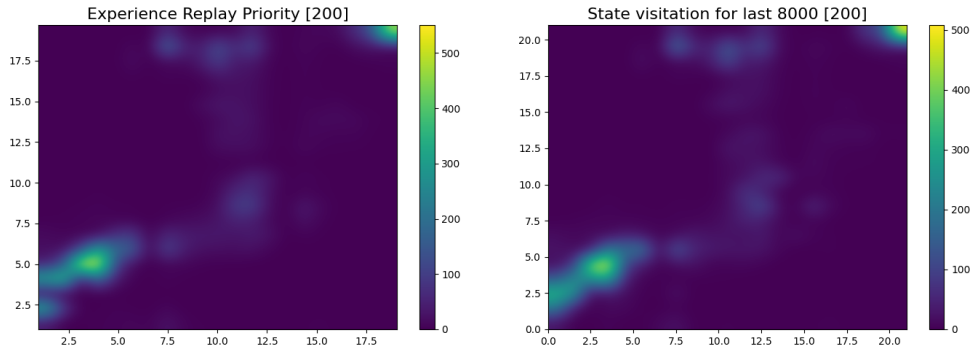


Figure 6.9: Buffer alignment visualisation

Chapter 7

Conclusion

In Chapter 3, we identified the *buffer sampling distributional alignment problem* as a core limitation of doing reinforcement learning without exploration. This is a novel insight and one of our ambitions for the project is that this can help motivate algorithmic innovation in the offline reinforcement learning space to address remaining performance gaps with online learning algorithms.

As part of our analysis we propose using an adapted Maximum Mean Discrepancy to approximately quantify the degree of misalignment (Section 3.5) in a non-parametric, sample-based fashion. The absence of quantitative techniques in the literature for this means that notions of *off-policyness* are often analysed using didactic examples rather than statistical tools. Our proposed statistic works well experimentally but we believe better solutions are waiting to be discovered.

in Chapter 4, we begin the effort to find better offline reinforcement learning algorithms that tackle the misalignment problem by proposing the MAORL framework. Unlike any offline reinforcement learning method that precedes it, MAORL algorithms tune the buffer sampling regime rather than applying explicit constraints on the policy. This puts it in a brand new class of algorithms that consider experience replay buffer sampling management as explictly part of the optimisation problem being solved by the algorithm. We briefly introduce an ad-hoc buffer alignment mechanism in Section 4.2.

Using our quantitative metric defined earlier, we are able to derive a mathematically principled approach to buffer alignment that uses techniques from convex optimisation to iteratively arrive at a sampling regime that is closely aligned to the agent's visitation continuously during training (Section 4.3).

Evaluation of our proposed algorithm suggests that this approach largely mitigates the misalignment problem, vastly outperforming our naive offline reinforcement learning baseline with access to all out-of-distribution transitions on a GridWorld task set. Performance closes in on that of an online agent.

Finally, in Chapter 5 we provide a concrete approach towards developing uncertainty-penalised MDPs by using Spectral-normalised Gaussian Processes (SNGP), which we argue have some better properties than ensemble-based techniques traditionally used in existing model-based offline reinforcement learning implementations. We want to increase awareness of other, promising uncertainty estimation techniques coming out of the deep learning community and encourage greater experimentation. As part of this effort, we produce a reference SNGP PyTorch library that has been open-sourced.

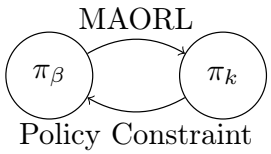
7.1 Conceptual relationship to the state-of-the-art

The MAORL framework distinguishes itself from any existing offline method by aiming to evolve the buffer sampling distribution rather than control policy optimisation as is common in the literature. The second way it is unique is in the way it uses a learned dynamics model without needing a reward model, unlike existing model-based methods.

7.1.1 Buffer tuning

Existing model-free offline reinforcement learning algorithms do not usually explicitly evolve the mechanism used for sampling (π_β). Model-free offline methods traditionally sample uniform randomly from the offline dataset, and primarily make algorithmic adjustments to prevent errors from bootstrapping from out-of-distribution actions from derailing training.

Policy constraint methods in particular aim to constrain policy improvement steps to minimise some measure of the divergence between the current policy π_k and a behavioural model of the training data each policy improvement iteration in order to prevent the policy from improving in an unconstrained manner that might lead it to take actions outside the support of π_β.



Policy constraint methods have two key drawbacks not present in MAORL. Firstly, many of these methods require us to model π_β [2] [14] [13] using tools such as VAEs, which can add complexity and inaccuracy compared to using the replay buffer as is in a non-parametric fashion. Secondly, constraining π to π_β can be too aggressive in preventing improvement over and beyond the policies used to collect the offline data as discussed in [13]. While MAORL actually aims to align with the sampled state-action visitation and not the exact policy π of the agent (these can be different in off-policy methods), in general we can argue that MAORL algorithms approximately go the opposite way to policy constraint methods, optimising the sampling process to be closer to the policy rather than vice versa.

The advantage-weighted behavioural model (ABM) [13] is a interesting proposal to alleviate some of the issues from having an excessively conservative policy improvement constraint. The authors propose learning a parametric model of the offline data by maximising an advantage-weighted log likelihood. In effect, ABM filters the behavioural policy π_β on policy improvement for transitions that have high expected advantage with respect to the current policy and have therefore potential to improve our policy, rather than simply trying to model construct a maximum likelihood model of the data buffer. At the time of writing, ABM is probably the nearest conceptual neighbour to MAORL in the literature however it too requires us to construct parametric models of the experience replay buffer.

7.1.2 What the dynamics model is used for

Model-based offline reinforcement learning methods such as MOPO [20] and MOReL [21] learn both dynamics and reward models from the offline dataset. As illustrated earlier in Figure 4.1, MAORL methods are unique in that they do not learn a reward model nor do they explictly sample synthetic transitions from model rollouts directly for learning, instead only using it to tune buffer sampling.

7.2 Evaluation Limitation

One of the limitations of our evaluation is the lack of experimentation on more challenging continuous control tasks. We have no reason to believe our approach will not scale to meet the challenge and our choice to evaluate our approach using our synthetic, discrete action GridWorld environments (Appendix C) was quite intentional. We wanted a usable test-bench that could continuously challenge our ideas as we iterated on how the final algorithm should look, and operating on a two dimensional GridWorld environment allows us to develop visualisations and graphical representations of concepts that are not possible when working with higher dimensional states. Furthermore, this environment enables us to perform idealised experiments such as including OOD transitions explicitly into the replay buffer to evaluate unique aspects of our proposals - this is not easy to do on more complex tasks. Nonetheless, it

is still essential to ultimately assess performance on continuous control problems since most real-world tasks require continuous actions and we intend to address this in future work.

7.3 Future work and continuation plans

We intend to continue work on this project with the intention of publication.

7.3.1 Continuous control tasks and real world robotics

One omission from our project is the lack of experimental data on continuous control tasks. Developing benchmark results on the D4RL benchmark suite [33] is an important step to take to experimentally to validate the scalability of our proposed approach to more complex tasks. Taking this further still, future work should include deployment of MAORL algorithms using real robots. We believe that gathering data on how our approach works in practical applications can help drive improvements to our proposals.

7.3.2 Optimisation strategies for priority assignment

As part of our project we explored numerous optimisation strategies (Appendix B) to efficiently produce solutions to the optimisation task set out for good priority assignment in Equation (4.2). While our approach has proven to be highly scalable and computationally very reasonable, we believe that there is potential for further research into new optimisation strategies that may be able to find efficiency gains over the approach we have proposed using projected gradient descent (Equation (4.4)).

7.3.3 Quantification of off-policyness

The weighted maximum mean discrepency metric we propose in Section 3.5 has good qualities and fills a vacuum in the literature for how to quantify *off-policyness*. There is an absence of quantitative techniques in the literature that quantify how *different* two policies are and off-policyness is often analysed using didactic examples rather than statistical tools. Our proposed statistic works well experimentally for our purposes in this project but we believe better solutions are waiting to be discovered from first principles that give more precise treatment to the Markovian structure of a rollout. We believe well-founded quantitative techniques to evaluate this would advance the state of affairs in reinforcement learning generally by helping us to build more precise conceptual models for how published algorithms truly work.

7.3.4 Can we do alignment in a model-free fashion?

The MAORL framework requires the use of a dynamics model and uncertainty-based out-of-distribution detector. The former enables us to obtain sample rollout traces that we can use to align our buffer priorities to and the latter helps us avoid action distributional shift. Finding ways to estimate the potential state-action visitation of the agent in a manner that is not sample-oriented would help us move past the need for a dynamics model without having to develop parametric models of our replay buffer as is common in related methods such as ABM [13].

7.3.5 Explictly codifying the experience replay buffer

Our proposed algorithm is quite unique in that it explictly manages its experience replay buffer using a strategy codified mathematically. This is not just quite unique in the offline reinforcement learning community but also in the deep reinforcement learning community at large. Most algorithmic innovations

focus on learning representations, variance control, and other problems associated primarily with the policy improvement problem. The experience replay buffer is generally included in a fairly ad-hoc fashion using a fixed size queue and it is difficult to build precise conceptual models for how this vital component of the learning process actually influences learning. One area to explore is to build on experimention in Section 6.1.5 to drive research that aims to more tightly integrate the experience replay buffer mechanism to the rest of the algorithm by framing the desired sampling process mathematically.

Attention mechanisms [34] have had tremendous success in supervised learning tasks such as machine translation by enabling the learned model to focus on important features of the input. Exploring attention-like mechanisms for self-management of the experience replay buffer sampling based objectives might be an interesting space to explore more broadly. This fits in with the doctrine that to build truly performant learning systems requires us to involve all parts of our algorithm in the learning process together.

Appendix A

Experiment Details

A.1 DQN Hyperparameters

Our offline and online Double DQN agents shared the exact same parameters for the same task where the online hyperparameter applies, so the following applies for both online and offline experiments. We make make minor adaptations for each assessed environment.

Table A.1: Double DQN Hyperparameters across online and offline tasks

Attribute	GridWorld-v0	GridWorld-v1
Hidden Layers	1	
Hidden Layer Size	32	
Batch Size	32	
Buffer Size	10000	8000
Training Episodes	300-400	300
Gradient Clipping	True at 1	
ϵ-decay schedule	Linear 0.7 to 0.05	Linear 0.6 to 0.05
γ	0.98	
Optimisation	Adam with learning rate 0.001	
Activation	ReLU	
Target Net Update Rate	Hard update every 1000 iterations	
n-step returns	2	

A.2 Dynamics Model (SNGP) Hyperparameters

When learning dynamics models, we used the same model architecture for both environments. We trained our dynamics models using our open-sourced SNGP PyTorch library.

Table A.2: SNGP Hyperparameters for uncertainty-aware dynamics learning

Attribute	GridWorld-v0	GridWorld-v1
Backbone	Resnet	
Spectral Normalisation	0.9	
Training Epochs	300	
Ridge Penalty	10^{-6}	
Units in linear RFF Approximation	1024	
Precision Matrix Computation	End of training	
Hidden Layers	1	
Activation	ReLU	

Appendix B

Attempted Optimisation Strategies

We tried numerous optimisation strategies beyond what is documented in the main body of the report to find solutions to the buffer priority assignment problem (Equation (4.2)). We list these with brief comments to motivate future work exploring new optimisation strategies.

Solve Lagrangian via Dual Gradient Descent

We can define the Lagrangian of the original problem $\mathcal{L}$, with Lagrange multipliers $\lambda \in \mathbb{R}^M$, $\beta \in \mathbb{R}$ and $\gamma \in \mathbb{R}$ with $\lambda \geq \mathbf{0}$, $\beta \geq 0$ and $\gamma \geq 0$.

$$\mathcal{L}(\{p, \lambda, \beta, \gamma\}) = L(p) - \lambda^T p + \beta(\sum_{m=1}^{M} p_m - C) + \gamma(C - \sum_{m=1}^{M} p_m) \tag{B.1}$$

We can now define the new objective as follows:

$$\max_{\{\lambda,\beta,\gamma\}} \min_{p} \mathcal{L}(\{p, \lambda, \beta, \gamma\}) \tag{B.2}$$

We have two modes in our implementation. In one implementation, we experimented with manually tuning the Lagrange variables to high positive values for the GridWorld environment being experimented in. This is not recommended, as good settings are highly dependant on the specific task and environment. In the other mode, each iteration of gradient descent with respect to p, we perform gradient ascent of the same Lagrangian loss with respect to the Lagrange multipliers in order to ensure their non-negativity in order to satisfy the constraints.

Achieving convergence of the dual gradient descent approach proved very difficult and the non-negativity constraints in particular are difficult to manage. This approach is probably not scalable for large datasets.

Interpret p as log priorities

This was a speculative approach at maintaining our buffer priorities p as log priorities instead to produce a feasible sampling regime. This approach is fundamentally unsound however since the effective projection onto the feasible region does not preserve relative priorities of each transition due to the exponentiation operation. The redefined objective looks as follows:

$$L(p) = \sum_{i=1}^{N}\sum_{j=1}^{N} p_\pi^{(i)} p_\pi^{(j)} k(x_i, x_j) + \sum_{i=1}^{M}\sum_{j=1}^{M} S(p_i)S(p_j)k(y_i, y_j) - \sum_{i=1}^{N}\sum_{j=1}^{M} 2p_\pi^{(i)} S(p_j)k(x_i, y_j) \tag{B.3}$$

where

$$S(p_t) = \frac{C \exp p^{(t)}}{\sum_{k=1}^{M} \exp p^{(k)}}$$

Appendix C

GridWorld Environment Specifications

C.1 Environment

(a) GridWorld-v0 (b) GridWorld-v1

Figure C.1: GridWorld environments. Both GridWorlds share an identical reward function and similar dynamics. Agents can interact with the environments using standard OpenAI Gym API calls.

C.2 State space ($\mathcal{S}$) and reward function

The different variants of states ($s \in \mathcal{S}$) are summarised in Figure C.2.

State Variant	Colour	Reward on arrival (small)	Reward on arrival (large)	Halting
Lava	Orange	-1.764	-21.168	Yes
Goal	Green	3.528	21.168	Yes
Floor	Black	0	-0.12	No
Wall	Grey	0	-0.12	No

Figure C.2: State space $\mathcal{S}$

We have two reward modes we term *small* and *large*. *Small* reward mode provides complete sparsity, and no reward shaping is done. Goal and Lava states are terminal. Lava states can be used to model the reward penalties we would like to apply to OOD states-action tuples. We usually apply a light penalty of -0.005 each step to encourage the agent to reach the goal faster in experiments where step penalty is specified. This is the method used to evaluate our methods in Chapter 6.

In *Large* reward mode, rewards are larger and every state has an associate reward. This mode is used in the exposition of the buffer alignment problem.

We encode an agent's current state/position by specifying its (x, y) coordinates and orientation (North, West, South or East). As such, state is encoded in a vector with 3 elements.

C.3 Action space ($\mathcal{A}$)

At any given state that can be legally reached, the agent can either *move forward*, *rotate 90 degrees left* or *rotate 90 degrees right*. Since Lava and Goal states are absorbing states, we can never execute an action from them. Furthermore, moving forwards into a wall state keeps the agent in its original position. Note that moving to an adjacent square can require anywhere between 1 and 3 actions. Combined with the sparsity in the reward signal, this means that these environments provide significantly more challenge for reinforcement learning agents than typical GridWorld test environments.

Appendix D

Offline Data Collection

We could configure offline data collection as needed for different tasks with different levels of state-action space coverage.

```
def __transitions_for_offline_data(self, extra_data=False, include_lava_actions=False, exclude_lava_neighbours=False,
                                   n_step=1, cut_step_cost=False, GAMMA=OFFLINE_GAMMA):

    self.pause_statistics()

    def neighbour_state_lava(col, row):
        for n_col, n_row in [(col - 1, row), (col + 1, row), (col, row - 1), (col, row + 1)]:
            cell = self.grid.get(n_col, n_row)
            if cell is not None and cell.type == 'lava':
                return True
        return False

    def replicate(g, n, res):
        if n == 0:
            yield [n for n in res]
        else:
            for elem in g:
                res.append(elem)
                yield from replicate(g, n - 1, res)
                res.pop()

    for col, row, cell in self.offline_cells():
        if exclude_lava_neighbours and neighbour_state_lava(col, row):
            continue
        for dir in range(len(self.action_space)):
            for action_sequence in replicate(self.actions, n_step,
                                             []):
                self._gen_grid(self.width, self.height, (col, row), dir)
                state_vector = torch.from_numpy(
                    np.concatenate(self.construct_state_vector(self.agent_pos, self.agent_dir),
                                   axis=None)).float().unsqueeze(0)
                action_vector = torch.tensor([[action_sequence[0]]], requires_grad=False)
                cumulative_reward = 0
                for i, action in enumerate(action_sequence):
                    fwd_cell = self.grid.get(*self.front_pos)
                    if include_lava_actions or (
                            self.actions.forward != action or fwd_cell is None or fwd_cell.type != 'lava'):
                        _, reward, done, _ = self.step(action)
                        if cut_step_cost:
                            cumulative_reward += (reward - STEP_COST) * (GAMMA ** i)
                        else:
                            cumulative_reward += reward * (GAMMA ** i)
                        if done or i == (len(action_sequence) - 1):
                            next_state_trace = torch.from_numpy(
```

```
                                    np.concatenate(self.construct_state_vector(self.
    agent_pos, self.agent_dir),
                                                   axis=None)).float().unsqueeze(0)
                                reward = torch.tensor([[cumulative_reward]],
    requires_grad=False)
                                if i == (len(action_sequence)-1) and n_step > 1:
                                    done = False
                                transition = Transition(state=state_vector.to(self.device
    ), action=action_vector.to(self.device), reward=reward.to(self.device),
                                                        next_state=next_state_trace.to(
    self.device), done=torch.tensor([[done]]).to(self.device))
                                yield transition
                                break
```

Listing D.1: Transition collection generator

Bibliography

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

[2] 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.

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

[4] Pavlos Athanasios Apostolopoulos, Zehui Wang, Hanson Wang, Chad Zhou, Kittipat Virochsiri, Norm Zhou, and Igor L Markov. Personalization for web-based services using offline reinforcement learning. *arXiv preprint arXiv:2102.05612*, 2021.

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

[6] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[7] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[8] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[9] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[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] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[13] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

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

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

[16] Gaon An, Seungyong Moon, Jang-Hyun Kim, and Hyun Oh Song. Uncertainty-based offline reinforcement learning with diversified q-ensemble. *Advances in Neural Information Processing Systems*, 34, 2021.

[17] Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.

[18] 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.

[19] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

[20] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

[21] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

[22] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

[23] Christopher K Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[24] 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 arXiv:2006.10108*, 2020.

[25] Aditya Goel. Spectral-normalized-neural-gaussian-process-pytorch. [Online]. Available from: https://github.com/adityagoel4512/Spectral-normalized-Neural-Gaussian-Process-PyTorch/blob/master/sngp.ipynb, 2022.

[26] Michael Held, Philip Wolfe, and Harlan P Crowder. Validation of subgradient optimization. *Mathematical programming*, 6(1):62–88, 1974.

[27] Laurent Condat. Fast projection onto the simplex and the $$\pmb {l}_\mathbf {1}$$ball. *Mathematical Programming*, 158(1):575–585, 2016. doi: 10.1007/s10107-015-0946-6. URL https://doi.org/10.1007/s10107-015-0946-6.

[28] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020.

[29] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. URL http://jmlr.org/papers/v13/gretton12a.html.

[30] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[31] Ewout Van Den Berg and Michael P Friedlander. Probing the pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2009.

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

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

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.