Reinforcement Learning: An Overview[1]

Kevin P. Murphy

December 9, 2024

Contents

Chapter 1

Introduction

1.1 Sequential decision making

Reinforcement learning or **RL** is a class of methods for solving various kinds of sequential decision making tasks. In such tasks, we want to design an **agent** that interacts with an external **environment**. The agent maintains an internal state s_t, which it passes to its **policy** π to choose an action $a_t = \pi(s_t)$. The environment responds by sending back an observation o_{t+1}, which the agent uses to update its internal state using the state-update function $s_{t+1} = U(s_t, a_t, o_{t+1})$. See Figure 1.1 for an illustration.

1.1.1 Problem definition

The goal of the agent is to choose a policy π so as to maximize the sum of expected rewards:

$$V_\pi(s_0) = \mathbb{E}_{p(a_0, s_1, a_1, \ldots, a_T, s_T | s_0, \pi)} \left[\sum_{t=0}^{T} R(s_t, a_t) | s_0 \right] \tag{1.1}$$

where s_0 is the agent's initial state, $R(s_t, a_t)$ is the **reward function** that the agent uses to measure the value of performing an action in a given state, $V_\pi(s_0)$ is the **value function** for policy π evaluated at s_0, and the expectation is wrt

$$p(a_0, s_1, a_1, \ldots, a_T, s_T | s_0, \pi) = \pi(a_0 | s_0) p_{\text{env}}(o_1 | a_0) \delta(s_1 = U(s_0, a_0, o_1)) \tag{1.2}$$

$$\times \pi(a_1 | s_1) p_{\text{env}}(o_2 | a_1, o_1) \delta(s_2 = U(s_1, a_1, o_2)) \tag{1.3}$$

$$\times \pi(a_2 | s_2) p_{\text{env}}(o_3 | a_{1:2}, o_{1:2}) \delta(s_3 = U(s_2, a_2, o_3)) \ldots \tag{1.4}$$

where p_{env} is the environment's distribution over observations (which is usually unknown). We define the optimal policy as

$$\pi^* = \arg \max_\pi \mathbb{E}_{p_0(s_0)} [V_\pi(s_0)] \tag{1.5}$$

Note that picking a policy to maximize the sum of expected rewards is an instance of the **maximum expected utility** principle. There are various ways to design or learn an optimal policy, depending on the assumptions we make about the environment, and the form of the agent. We will discuss some of these options below.

1.1.2 Universal model

A generic representation for sequential decision making problems (which is an extended version of the "universal modeling framework" proposed in [Pow22]) is shown in Figure 1.2. Here we have assumed the

9
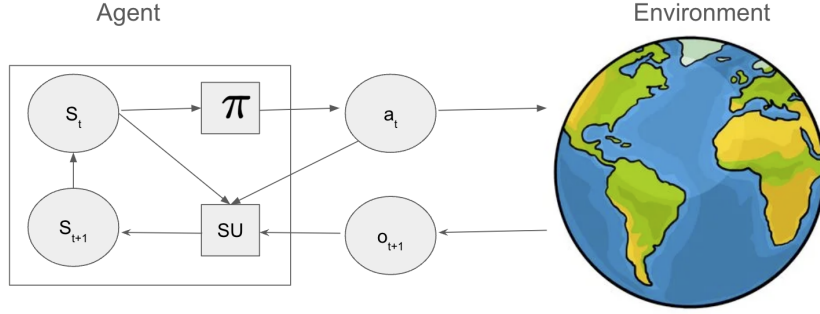
Figure 1.1: A small agent interacting with a big external world.
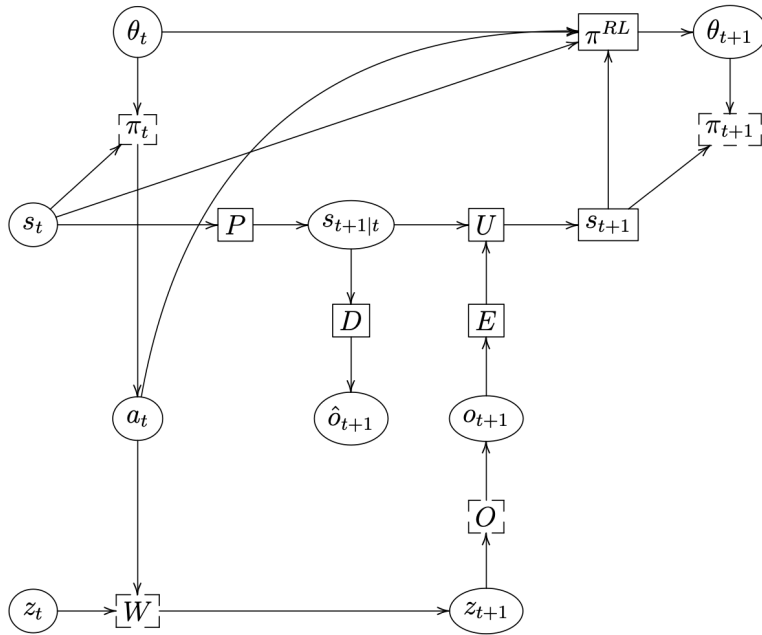
Figure 1.2: Diagram illustrating the interaction of the agent and environment. The agent has internal state s_t, and chooses action a_t based on its policy π_t. It then predicts its next internal states, $s_{t+1|t}$, via the predict function P, and optionally predicts the resulting observation, $\hat{o}_{t+1}$, via the observation decoder D. The environment has (hidden) internal state z_t, which gets updated by the world model W to give the new state $z_{t+1} = W(z_t, a_t)$ in response to the agent's action. The environment also emits an observation o_{t+1} via the observation model O. This gets encoded to e_{t+1} by the agent's observation encoder E, which the agent uses to update its internal state using $s_{t+1} = U(s_t, a_t, e_{t+1})$. The policy is parameterized by θ_t, and these parameters may be updated (at a slower time scale) by the RL policy π^{RL}. Square nodes are functions, circles are variables (either random or deterministic). Dashed square nodes are stochastic functions that take an extra source of randomness (not shown).

environment can be modeled by a controlled **Markov process**[1] with hidden state z_t, which gets updated at each step in response to the agent's action a_t. To allow for non-deterministic dynamics, we write this as $z_{t+1} = W(z_t, a_t, \epsilon_t^z)$, where W is the environment's state transition function (which is usually not known to the agent) and ϵ_t^z is random system noise.[2], The agent does not see the world state z_t, but instead sees a potentially noisy and/or partial observation $o_{t+1} = O(z_{t+1}, \epsilon_{t+1}^o)$ at each step, where ϵ_{t+1}^o is random observation noise. For example, when navigating a maze, the agent may only see what is in front of it, rather than seeing everything in the world all at once; furthermore, even the current view may be corrupted by sensor noise. Any given image, such as one containing a door, could correspond to many different locations in the world (this is called **perceptual aliasing**), each of which may require a different action. Thus the agent needs use these observations to incrementally update its own internal **belief state** about the world, using the state update function $s_{t+1} = SU(s_t, a_t, o_{t+1})$; this represents the agent's beliefs about the underlying world state z_t, as well as the unknown world model W itself (or some proxy thereof). In the simplest setting, the internal s_t can just store all the past observations, $\boldsymbol{h}_t = (\boldsymbol{o}_{1:t}, \boldsymbol{a}_{1:t-1})$, but such non-parametric models can take a lot of time and space to work with, so we will usually consider parametric approximations. The agent can then pass its state to its policy to pick actions, using $a_{t+1} = \pi_t(s_{t+1})$.

We can further elaborate the behavior of the agent by breaking the state-update function into two parts. First the agent predicts its own next state, $s_{t+1|t} = P(s_t, a_t)$, using a **prediction function** P, and then it updates this prediction given the observation using **update function** U, to give $s_{t+1} = U(s_{t+1|t}, o_{t+1})$. Thus the SU function is defined as the composition of the predict and update functions: $s_{t+1} = SU(s_t, a_t, o_{t+1}) = U(P(s_t, a_t), o_{t+1})$. If the observations are high dimensional (e.g., images), the agent may choose to encode its observations into a low-dimensional embedding e_{t+1} using an encoder, $e_{t+1} = E(o_{t+1})$; this can encourage the agent to focus on the relevant parts of the sensory signal. (The state update then becomes $s_{t+1} = U(s_{t+1|t}, e_{t+1})$.) Optionally the agent can also learn to invert this encoder by training a decoder to predict the next observation using $\hat{o}_{t+1} = D(s_{t+1|t})$; this can be a useful training signal, as we will discuss in Chapter 4. Finally, the agent needs to learn the action policy π_t. We parameterize this by $\boldsymbol{\theta}_t$, so $\pi_t(s_t) = \pi(s_t; \boldsymbol{\theta}_t)$. These parameters themselves may need to be learned; we use the notation π^{RL} to denote the RL policy which specifies how to update the policy parameters at each step. See Figure 1.2 for an illustration.

We see that, in general, there are three interacting stochastic processes we need to deal with: the environment's states z_t (which are usually affected by the agents actions); the agent's internal states s_t (which reflect its beliefs about the environment based on the observed data); and the the agent's policy parameters $\boldsymbol{\theta}_t$ (which are updated based on the information stored in the belief state). The reason there are so many RL algorithms is that this framework is very general. In the rest of this manuscript we will study special cases, where we make different assumptions about the environment's state z_t and dynamics, the agent's state s_t and dynamics, the form of the action policy $\pi(s_t | \boldsymbol{\theta}_t)$, and the form of the policy learning method $\boldsymbol{\theta}_{t+1} = \pi^{RL}(\boldsymbol{\theta}_t, s_t, a_t, o_{t+1})$.

1.1.3 Episodic vs continuing tasks

If the agent can potentially interact with the environment forever, we call it a **continuing task**. Alternatively, the agent is in an **episodic task**, if its interaction terminates once the system enters a **terminal state** or **absorbing state**, which is a state which transitions to itself with 0 reward. After entering a terminal state, we may start a new **epsiode** from a new initial world state $z_0 \sim p_0$. (The agent will typically also reinitialize its own internal state s_0.) The episode length is in general random. For example, the amount of time a robot takes to reach its goal may be quite variable, depending on the decisions it makes, and the randomness in the environment. Finally, if the trajectory length T in an episodic task is fixed and known, it is called a **finite horizon problem**.

We define the **return** for a state at time t to be the sum of expected rewards obtained going forwards,

---

[1]The Markovian assumption is without loss of generality, since we can always condition on the entire past sequence of states by suitably expanding the Markovian state space.

[2]Representing a stochastic function as a deterministic function with some noisy inputs is known as a functional causal model, or structural equation model. This is standard practice in the control theory and causality communities.

where each reward is multiplied by a **discount factor** $\gamma \in [0, 1]$:

$$G_t \triangleq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} \tag{1.6}$$

$$= \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{j=t}^{T-1} \gamma^{j-t} r_j \tag{1.7}$$

where $r_t = R(s_t, a_t)$ is the reward, and G_t is the **reward-to-go**. For episodic tasks that terminate at time T, we define $G_t = 0$ for $t \geq T$. Clearly, the return satisfies the following recursive relationship:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1} \tag{1.8}$$

Furthermore, we define the value function to be the expected reward-to-go:

$$V_\pi(s_t) = \mathbb{E}\left[G_t | \pi\right] \tag{1.9}$$

The discount factor γ plays two roles. First, it ensures the return is finite even if $T = \infty$ (i.e., infinite horizon), provided we use $\gamma < 1$ and the rewards r_t are bounded. Second, it puts more weight on short-term rewards, which generally has the effect of encouraging the agent to achieve its goals more quickly. (For example, if $\gamma = 0.99$, then an agent that reaches a terminal reward of 1.0 in 15 steps will receive an expected discounted reward of $0.99^{15} = 0.86$, whereas if it takes 17 steps it will only get $0.99^{17} = 0.84$.) However, if γ is too small, the agent will become too greedy. In the extreme case where $\gamma = 0$, the agent is completely **myopic**, and only tries to maximize its immediate reward. In general, the discount factor reflects the assumption that there is a probability of $1 - \gamma$ that the interaction will end at the next step. For finite horizon problems, where T is known, we can set $\gamma = 1$, since we know the life time of the agent a priori.[3]

1.1.4   Regret

So far we have been discussing maximizing the reward. However, the upper bound on this is usually unknown, so it can be hard to know how well a given agent is doing. An alternative approach is to work in terms of the **regret**, which is defined as the difference between the expected reward under the agent's policy and the oracle policy π_*, which knows the true MDP. Specifically, let π_t be the agent's policy at time t. Then the **per-step regret** at t is defined as

$$l_t \triangleq \mathbb{E}_{s_{1:t}} \left[R(s_t, \pi_*(s_t)) - \mathbb{E}_{\pi(a_t|s_t)} \left[R(s_t, a_t) \right] \right] \tag{1.10}$$

Here the expectation is with respect to randomness in choosing actions using the policy π, as well as earlier states, actions and rewards, as well as other potential sources of randomness.

If we only care about the final performance of the agent, as in most optimization problems, it is enough to look at the **simple regret** at the last step, namely l_T. Optimizing simple regret results in a problem known as **pure exploration** [BMS11], where the agent needs to interact with the environment to learn the underlying MDP; at the end, it can then solve for the resulting policy using planning methods (see Section 2.2). However, in RL, it is more common to focus on the **cumulative regret**, also called the **total regret** or just the **regret**, which is defined as

$$L_T \triangleq \mathbb{E} \left[\sum_{t=1}^{T} l_t \right] \tag{1.11}$$

Thus the agent will accumulate reward (and regret) while it learns a model and policy. This is called **earning while learning**, and requires performing exploratory actions, to learn the model (and hence optimize long-term reward), while also performing actions that maximize the reward at each step. This requires solving the exploration-exploitation tradeoff, as we discussed in Section 1.4.

---

[3]We may also use $\gamma = 1$ for continuing tasks, targeting the (undiscounted) average reward criterion [Put94].

1.1.5 Further reading

In later chapters, we will describe methods for learning the best policy to maximize $V_\pi(s_0) = \mathbb{E}[G_0|s_0, \pi])$. More details on RL can be found in textbooks such as [Sze10; SB18; Aga+22a; Pla22; ID19; RJ22; Li23; MMT24], and reviews such as [Aru+17; FL+18; Li18; Wen18a]. For details on how RL relates to **control theory**, see e.g., [Son98; Rec19; Ber19; Mey22], and for connections to operations research, see [Pow22].

1.2 Canonical examples

In this section, we describe different forms of model for the environment and the agent that have been studied in the literature.

1.2.1 Partially observed MDPs

The model shown in Figure 1.2 is called a **partially observable Markov decision process** or **POMDP** (pronounced "pom-dee-pee") [KLC98]. Typically the environment's dynamics model is represented by a stochastic transition function, rather than a deterministic function with noise as an input. We can derive this transition function as follows:

$$p(z_{t+1}|z_t, a_t) = \mathbb{E}_{\epsilon_t^z}\left[\mathbb{I}\left(z_{t+1} = W(z_t, a_t, \epsilon_t^z)\right)\right] \tag{1.12}$$

Similarly the stochastic observation function is given by

$$p(o_{t+1}|z_{t+1}) = \mathbb{E}_{\epsilon_{t+1}^o}\left[\mathbb{I}\left(o_{t+1} = O(z_{t+1}, \epsilon_{t+1}^o)\right)\right] \tag{1.13}$$

Note that we can combine these two distributions to derive the joint world model $p_{WO}(z_{t+1}, o_{t+1}|z_t, a_t)$. Also, we can use these distributions to derive the environment's non-Markovian observation distribution, $p_{\text{env}}(o_{t+1}|o_{1:t}, a_{1:t})$, used in Equation (1.4), as follows:

$$p_{\text{env}}(o_{t+1}|o_{1:t}, a_{1:t}) = \sum_{z_{t+1}} p(o_{t+1}|z_{t+1})p(z_{t+1}|a_{1:t}) \tag{1.14}$$

$$p(z_{t+1}|a_{1:t}) = \sum_{z_1}\cdots\sum_{z_t} p(z_1|a_1)p(z_2|z_1, a_1)\dots p(z_{t+1}|z_t, a_t) \tag{1.15}$$

If the world model (both $p(o|z)$ and $p(z'|z, a)$) is known, then we can — in principle — solve for the optimal policy. The method requires that the agent's internal state correspond to the **belief state** $s_t = \boldsymbol{b}_t = p(z_t|\boldsymbol{h}_t)$, where $\boldsymbol{h}_t = (o_{1:t}, a_{1:t-1})$ is the observation history. The belief state can be updated recursively using Bayes rule. See Section 1.2.5 for details. The belief state forms a sufficient statistic for the optimal policy. Unfortunately, computing the belief state and the resulting optimal policy is wildly intractable [PT87; KLC98]. We discuss some approximate methods in Section 1.3.4.

1.2.2 Markov decision process (MDPs)

A **Markov decision process** [Put94] is a special case of a POMDP in which the environment states are observed, so $z_t = o_t = s_t$.[4] We usually define an MDP in terms of the state transition matrix induced by the world model:

$$p_S(s_{t+1}|s_t, a_t) = \mathbb{E}_{\epsilon_t^s}\left[\mathbb{I}\left(s_{t+1} = W(s_t, a_t, \epsilon_t^s)\right)\right] \tag{1.16}$$

---

[4]The field of control theory uses slightly different terminology and notation. In particular, the environment is called the **plant**, and the agent is called the **controller**. States are denoted by $\boldsymbol{x}_t \in \mathcal{X} \subseteq \mathbb{R}^D$, actions are denoted by $\boldsymbol{u}_t \in \mathcal{U} \subseteq \mathbb{R}^K$, and rewards are replaced by costs $c_t \in \mathbb{R}$.
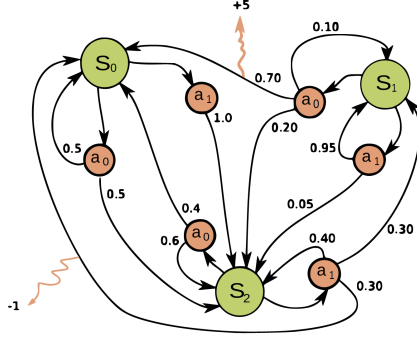
Figure 1.3: Illustration of an MDP as a finite state machine (FSM). The MDP has three discrete states (green cirlces), two discrete actions (orange circles), and two non-zero rewards (orange arrows). The numbers on the black edges represent state transition probabilities, e.g., $p(s' = s_0|a = a_0, s' = s_0) = 0.7$; most state transitions are impossible (probability 0), so the graph is sparse. The numbers on the yellow wiggly edges represent expected rewards, e.g., $R(s = s_1, a = a_0, s' = s_0) = +5$; state transitions with zero reward are not annotated. From https://en.wikipedia.org/wiki/Markov_decision_process. Used with kind permission of Wikipedia author waldoalvarez.

In lieu of an observation model, we assume the environment (as opposed to the agent) sends out a reward signal, sampled from $p_R(r_t|s_t, a_t, s_{t+1})$. The expected reward is then given by

$$R(s_t, a_t, s_{t+1}) = \sum_r r \; p_R(r|s_t, a_t, s_{t+1}) \tag{1.17}$$

$$R(s_t, a_t) = \sum_{s_{t+1}} p_S(s_{t+1}|s_t, a_t)R(s_t, a_t, s_{t+1}) \tag{1.18}$$

Given a stochastic policy $\pi(a_t|s_t)$, the agent can interact with the environment over many steps. Each step is called a **transition**, and consists of the tuple (s_t, a_t, r_t, s_{t+1}), where $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim p_S(s_t, a_t)$, and $r_t \sim p_R(s_t, a_t, s_{t+1})$. Hence, under policy π, the probability of generating a **trajectory** length T, $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots, s_T)$, can be written explicitly as

$$p(\boldsymbol{\tau}) = p_0(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t)p_S(s_{t+1}|s_t, a_t)p_R(r_t|s_t, a_t, s_{t+1}) \tag{1.19}$$

In general, the state and action sets of an MDP can be discrete or continuous. When both sets are finite, we can represent these functions as lookup tables; this is known as a **tabular representation**. In this case, we can represent the MDP as a **finite state machine**, which is a graph where nodes correspond to states, and edges correspond to actions and the resulting rewards and next states. Figure 1.3 gives a simple example of an MDP with 3 states and 2 actions.

If we know the world model p_S and p_R, and if the state and action space is tabular, then we can solve for the optimal policy using dynamic programming techniques, as we discuss in Section 2.2. However, typically the world model is unknown, and the states and actions may need complex nonlinear models to represent their transitions. In such cases, we will have to use RL methods to learn a good policy.

1.2.3 Contextual MDPs

A **Contextual MDP** [HDCM15] is an MDP where the dynamics and rewards of the environment depend on a hidden static parameter referred to as the context. (This is different to a contextual bandit, discussed in Section 1.2.4, where the context is observed at each step.) A simple example of a contextual MDP is a video game, where each level of the game is **procedurally generated**, that is, it is randomly generated each time the agent starts a new episode. Thus the agent must solve a sequence of related MDPs, which are

drawn from a common distribution. This requires the agent to **generalize** across multiple MDPs, rather than overfitting to a specific environment [Cob+19; Kir+21; Tom+22]. (This form of generalization is different from generalization within an MDP, which requires generalizing across states, rather than across environments; both are important.)

A contextual MDP is a special kind of POMDP where the hidden variable corresponds to the unknown parameters of the model. In [Gho+21], they call this an **epistemic POMDP**, which is closely related to the concept of belief state MDP which we discuss in Section 1.2.5.

1.2.4 Contextual bandits

A **contextual bandit** is a special case of a POMDP where the world state transition function is independent of the action of the agent and the previous state, i.e., $p(z_t|z_{t-1}, a_t) = p(z_t)$. In this case, we call the world states "contexts"; these are observable by the agent, i.e., $o_t = z_t$. Since the world state distribution is independent of the agents actions, the agent has no effect on the external environment. However, its actions do affect the rewards that it receives. Thus the agent's internal belief state — about the underlying reward function $R(o, a)$ — does change over time, as the agent learns a model of the world (see Section 1.2.5).

A special case of a contextual bandit is a regular bandit, in which there is no context, or equivalently, s_t is some fixed constant that never changes. When there are a finite number of possible actions, $\mathcal{A} = \{a_1, \ldots, a_K\}$, this is called a **multi-armed bandit**.[5] In this case the reward model has the form $R(a) = f(\boldsymbol{w}_a)$, where $\boldsymbol{w}_a$ are the parameters for arm a.

Contextual bandits have many applications. For example, consider an **online advertising system**. In this case, the state s_t represents features of the web page that the user is currently looking at, and the action a_t represents the identity of the ad which the system chooses to show. Since the relevance of the ad depends on the page, the reward function has the form $R(s_t, a_t)$, and hence the problem is contextual. The goal is to maximize the expected reward, which is equivalent to the expected number of times people click on ads; this is known as the **click through rate** or **CTR**. (See e.g., [Gra+10; Li+10; McM+13; Aga+14; Du+21; YZ22] for more information about this application.) Another application of contextual bandits arises in **clinical trials** [VBW15]. In this case, the state s_t are features of the current patient we are treating, and the action a_t is the treatment the doctor chooses to give them (e.g., a new drug or a **placebo**).

For more details on bandits, see e.g., [LS19; Sli19].

1.2.5 Belief state MDPs

In this section, we describe a kind of MDP where the state represents a probability distribution, known as a **belief state** or **information state**, which is updated by the agent ("in its head") as it receives information from the environment.[6] More precisely, consider a contextual bandit problem, where the agent approximates the unknown reward by a function $R(o, a) = f(o, a; \boldsymbol{w})$. Let us denote the posterior over the unknown parameters by $\boldsymbol{b}_t = p(\boldsymbol{w}|\boldsymbol{h}_t)$, where $\boldsymbol{h}_t = \{o_{1:t}, a_{1:t}, r_{1:t}\}$ is the history of past observations, actions and rewards. This belief state can be updated deterministically using Bayes' rule; we denote this operation by $\boldsymbol{b}_{t+1} = \text{BayesRule}(\boldsymbol{b}_t, o_{t+1}, a_{t+1}, r_{t+1})$. (This corresponds to the state update SU defined earlier.) Using this, we can define the following **belief state MDP**, with deterministic dynamics given by

$$p(\boldsymbol{b}_{t+1}|\boldsymbol{b}_t, o_{t+1}, a_{t+1}, r_{t+1}) = \mathbb{I}\left(\boldsymbol{b}_{t+1} = \text{BayesRule}(\boldsymbol{b}_t, o_{t+1}, a_{t+1}, r_{t+1})\right) \tag{1.20}$$

and reward function given by

$$p(r_t|o_t, a_t, \boldsymbol{b}_t) = \int p_R(r_t|o_t, a_t; \boldsymbol{w}) p(\boldsymbol{w}|\boldsymbol{b}_t) d\boldsymbol{w} \tag{1.21}$$

---

[5]The terminology arises by analogy to a slot machine (sometimes called a "bandit") in a casino. If there are K slot machines, each with different rewards (payout rates), then the agent (player) must explore the different machines until they have discovered which one is best, and can then stick to exploiting it.

[6]Technically speaking, this is a POMDP, where we assume the states are observed, and the parameters are the unknown hidden random variables. This is in contrast to Section 1.2.1, where the states were not observed, and the parameters were assumed to be known.
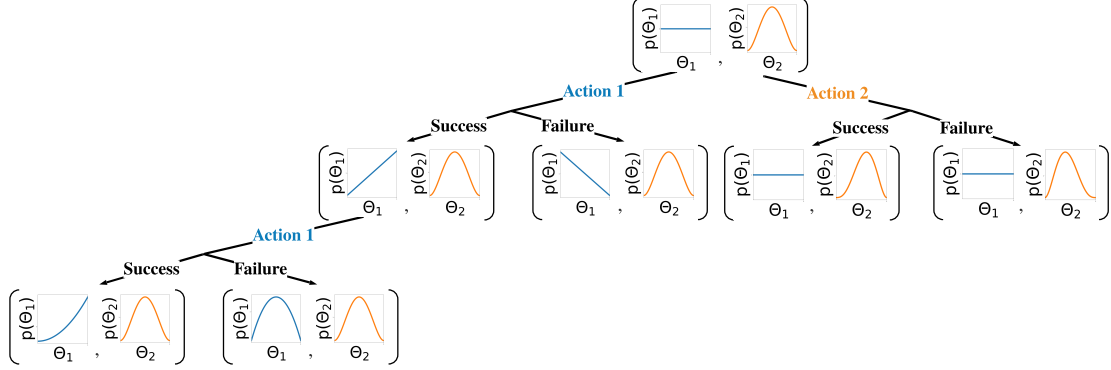
Figure 1.4: Illustration of sequential belief updating for a two-armed beta-Bernoulli bandit. The prior for the reward for action 1 is the (blue) uniform distribution Beta(1, 1)*; the prior for the reward for action 2 is the (orange) unimodal distribution* Beta(2, 2)*. We update the parameters of the belief state based on the chosen action, and based on whether the observed reward is success (1) or failure (0).*

If we can solve this (PO)MDP, we have the optimal solution to the exploration-exploitation problem.

As a simple example, consider a context-free **Bernoulli bandit**, where $p_R(r|a) = \text{Ber}(r|\mu_a)$, and $\mu_a = p_R(r = 1|a) = R(a)$ is the expected reward for taking action a. The only unknown parameters are $\boldsymbol{w} = \mu_{1:A}$. Suppose we use a factored beta prior

$$p_0(\boldsymbol{w}) = \prod_a \text{Beta}(\mu_a|\alpha_0^a, \beta_0^a) \tag{1.22}$$

where $\boldsymbol{w} = (\mu_1, \dots, \mu_K)$. We can compute the posterior in closed form to get

$$p(\boldsymbol{w}|\mathcal{D}_t) = \prod_a \text{Beta}(\mu_a|\underbrace{\alpha_0^a + N_t^0(a)}_{\alpha_t^a}, \underbrace{\beta_0^a + N_t^1(a)}_{\beta_t^a}) \tag{1.23}$$

where

$$N_t^r(a) = \sum_{i=1}^{t-1} \mathbb{I}(a_i = a, r_i = r) \tag{1.24}$$

This is illustrated in Figure 1.4 for a two-armed Bernoulli bandit. We can use a similar method for a **Gaussian bandit**, where $p_R(r|a) = \mathcal{N}(r|\mu_a, \sigma_a^2)$.

In the case of contextual bandits, the problem is conceptually the same, but becomes more complicated computationally. If we assume a **linear regression bandit**, $p_R(r|s, a; \boldsymbol{w}) = \mathcal{N}(r|\boldsymbol{\phi}(s, a)^\mathsf{T}\boldsymbol{w}, \sigma^2)$, we can use Bayesian linear regression to compute $p(\boldsymbol{w}|\mathcal{D}_t)$ exactly in closed form. If we assume a **logistic regression bandit**, $p_R(r|s, a; \boldsymbol{w}) = \text{Ber}(r|\sigma(\boldsymbol{\phi}(s, a)^\mathsf{T}\boldsymbol{w}))$, we have to use approximate methods for approximate Bayesian logistic regression to compute $p(\boldsymbol{w}|\mathcal{D}_t)$. If we have a **neural bandit** of the form $p_R(r|s, a; \boldsymbol{w}) = \mathcal{N}(r|f(s, a; \boldsymbol{w}))$ for some nonlinear function f, then posterior inference is even more challenging (this is equivalent to the problem of inference in Bayesian neural networks, see e.g., [Arb+23] for a review paper for the offline case, and [DMKM22; JCM24] for some recent online methods).

We can generalize the above methods to compute the belief state for the parameters of an MDP in the obvious way, but modeling both the reward function and state transition function.

Once we have computed the belief state, we can derive a policy with optimal regret using the methods like UCB (Section 1.4.3) or Thompson sampling (Section 1.4.4).

1.2.6 Optimization problems

The bandit problem is an example of a problem where the agent must interact with the world in order to collect information, but it does not otherwise affect the environment. Thus the agents internal belief state

changes over time, but the environment state does not.[7] Such problems commomly arise when we are trying to optimize a fixed but unknown function R. We can "query" the function by evaluating it at different points (parameter values), and in some cases, the resulting observation may also include gradient information. The agent's goal is to find the optimum of the function in as few steps as possible. We give some examples of this problem setting below.

1.2.6.1 Best-arm identification

In the standard multi-armed bandit problem our goal is to maximize the sum of expected rewards. However, in some cases, the goal is to determine the best arm given a fixed budget of T trials; this variant is known as **best-arm identification** [ABM10]. Formally, this corresponds to optimizing the **final reward** criterion:

$$V_{\pi,\pi_T} = \mathbb{E}_{p(a_{1:T}, r_{1:T}|s_0,\pi)}\left[R(\hat{a})\right] \tag{1.25}$$

where $\hat{a} = \pi_T(a_{1:T}, r_{1:T})$ is the estimated optimal arm as computed by the **terminal policy** π_T applied to the sequence of observations obtained by the exploration policy π. This can be solved by a simple adaptation of the methods used for standard bandits.

1.2.6.2 Bayesian optimization

Bayesian optimization is a gradient-free approach to optimizing expensive blackbox functions. That is, we want to find

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, R(\boldsymbol{w}) \tag{1.26}$$

for some unknown function R, where $\boldsymbol{w} \in \mathbb{R}^N$, using as few actions (function evaluations of R) as possible. This is essentially an "infinite arm" version of the best-arm identification problem [Tou14], where we replace the discrete choice of arms $a \in \{1, \ldots, K\}$ with the parameter vector $\boldsymbol{w} \in \mathbb{R}^N$. In this case, the optimal policy can be computed if the agent's state s_t is a belief state over the unknown function, i.e., $s_t = p(R|\boldsymbol{h}_t)$. A common way to represent this distribution is to use Gaussian processes. We can then use heuristics like expected improvement, knowledge gradient or Thompson sampling to implement the corresponding policy, $\boldsymbol{w}_t = \pi(s_t)$. For details, see e.g., [Gar23].

1.2.6.3 Active learning

Active learning is similar to BayesOpt, but instead of trying to find the point at which the function is largest (i.e., $\boldsymbol{w}^*$), we are trying to learn the whole function R, again by querying it at different points $\boldsymbol{w}_t$. Once again, the optimal strategy again requires maintaining a belief state over the unknown function, but now the best policy takes a different form, such as choosing query points to reduce the entropy of the belief state. See e.g., [Smi+23].

1.2.6.4 Stochastic Gradient Descent (SGD)

Finally we discuss how to interpret SGD as a sequential decision making process, following [Pow22]. The action space consists of querying the unknown function R at locations $\boldsymbol{a}_t = \boldsymbol{w}_t$, and observing the function value $r_t = R(\boldsymbol{w}_t)$; however, unlike BayesOpt, now we also observe the corresponding gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{w}} R(\boldsymbol{w})|_{\boldsymbol{w}_t}$, which gives non-local information about the function. The environment state contains the true function R which is used to generate the observations given the agent's actions. The agent state contains the current parameter estimate $\boldsymbol{w}_t$, and may contain other information such as first and second moments $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$, needed by methods such as Adam. The update rule (for vanilla SGD) takes the form $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \boldsymbol{g}_t$, where the stepsize α_t is chosen by the policy, $\alpha_t = \pi(s_t)$. The terminal policy has the form $\pi(s_T) = \boldsymbol{w}_T$.

---

[7]In the contextual bandit problem, the environment state (context) does change, but not in response to the agent's actions. Thus $p(o_t)$ is usually assumed to be a static distribution.

Approach	Method	Functions learned	On/Off	Section
Value-based	SARSA	$Q(s, a)$	On	Section 2.4
Value-based	Q-learning	$Q(s, a)$	Off	Section 2.5
Policy-based	REINFORCE	$\pi(a|s)$	On	Section 3.2
Policy-based	A2C	$\pi(a|s)$, $V(s)$	On	Section 3.3.1
Policy-based	TRPO/PPO	$\pi(a|s)$, $A(s, a)$	On	Section 3.4.3
Policy-based	DDPG	$a = \pi(s)$, $Q(s, a)$	Off	Section 3.6.1
Policy-based	Soft actor-critic	$\pi(a|s)$, $Q(s, a)$	Off	Section 3.5.4
Model-based	MBRL	$p(s'|s, a)$	Off	Chapter 4

Table 1.1: Summary of some popular methods for RL. On/off refers to on-policy vs off-policy methods.

Although in principle it is possible to learn the learning rate (stepsize) policy using RL (see e.g., [Xu+17]), the policy is usually chosen by hand, either using a **learning rate schedule** or some kind of manually designed **adaptive learning rate** policy (e.g., based on second order curvature information).

1.3 Reinforcement Learning

In this section, we give a brief overview of how to compute optimal policies when the model of the environment is unknown; this is the core problem tackled by RL. We mostly focus on the MDP case, but discuss the POMDP case in Section 1.3.4.

We may categorize RL methods along two main dimensions: (1) by what the agent represents and learns: the value function, and/or the policy, and/or the model; (2) and by how actions are selected: **on-policy** (actions must be selected by the agent's current policy), and **off-policy** (actions can be select by any kind of policy, including human demonstrations). Table 1.1 lists a few representative examples. More details are given in the subsequent sections.

1.3.1 Value-based RL (Approximate Dynamic Programming)

In this section, we give a brief introduction to **value-based RL**, also called **Approximate Dynamic Programming** or **ADP**; see Chapter 2 for more details.

We introduced the value function $V_\pi(s)$ in Equation (1.1), which we repeat here for convenience:

$$V_\pi(s) \triangleq \mathbb{E}_\pi \left[G_0 | s_0 = s \right] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right] \tag{1.27}$$

The value function for the optimal policy π^* is known to satisfy the following recursive condition, known as **Bellman's equation**:

$$V_*(s) = \max_a R(s, a) + \gamma \mathbb{E}_{p_S(s'|s,a)} \left[V_*(s') \right] \tag{1.28}$$

This follows from the principle of **dynamic programming**, which computes the optimal solution to a problem (here the value of state s by combining the optimal solution of various subproblems (here the values of the next states s'). This can be used to derive the following learning rule:

$$V(s) \leftarrow V(s) + \eta[r + \gamma V(s') - V(s)] \tag{1.29}$$

where $s' \sim p_S(\cdot|s, a)$ is the next state sampled from the environment, and $r = R(s, a)$ is the observed reward. This is called **Temporal Difference** or **TD** learning (see Section 2.3.2 for details). Unfortunately, it is not clear how to derive a policy if all we know is the value function. We now describe a solution to this problem.

We first generalize the notion of value function to assigning a value to a state and action pair, by defining the **Q function** as follows:

$$Q_\pi(s, a) \triangleq \mathbb{E}_\pi [G_0 | s_0 = s, a_0 = a] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right] \tag{1.30}$$

This quantity represents the expected return obtained if we start by taking action a in state s, and then follow π to choose actions thereafter. The Q function for the optimal policy satisfies a modified Bellman equation

$$Q_*(s, a) = R(s, a) + \gamma \mathbb{E}_{p_S(s'|s,a)} \left[\max_{a'} Q_*(s', a') \right] \tag{1.31}$$

This gives rise to the following TD update rule:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') - Q(s, a) \tag{1.32}$$

where we sample $s' \sim p_S(\cdot|s, a)$ from the environment. The action is chosen at each step from the implicit policy

$$a = \underset{a'}{\operatorname{argmax}} Q(s, a') \tag{1.33}$$

This is called **Q learning** (see Section 2.5 for details),

1.3.2 Policy-based RL

In this section we give a brief introductin to **Policy-based RL**; for details see Chapter 3.

In policy-based methods, we try to directly maximize $J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{p(s_0)} [V_\pi(s_0)]$ wrt the parameter's $\boldsymbol{\theta}$; this is called **policy search**. If $J(\pi_{\boldsymbol{\theta}})$ is differentiable wrt $\boldsymbol{\theta}$, we can use stochastic gradient ascent to optimize $\boldsymbol{\theta}$, which is known as **policy gradient** (see Section 3.1).

Policy gradient methods have the advantage that they provably converge to a local optimum for many common policy classes, whereas Q-learning may diverge when approximation is used (Section 2.5.2.4). In addition, policy gradient methods can easily be applied to continuous action spaces, since they do not need to compute $\operatorname{argmax}_a Q(s, a)$. Unfortunately, the score function estimator for $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ can have a very high variance, so the resulting method can converge slowly.

One way to reduce the variance is to learn an approximate value function, $V_{\boldsymbol{w}}(s)$, and to use it as a baseline in the score function estimator. We can learn $V_{\boldsymbol{w}}(s)$ using using TD learning. Alternatively, we can learn an advantage function, $A_{\boldsymbol{w}}(s, a)$, and use it as a baseline. These policy gradient variants are called **actor critic** methods, where the actor refers to the policy $\pi_{\boldsymbol{\theta}}$ and the critic refers to $V_{\boldsymbol{w}}$ or $A_{\boldsymbol{w}}$. See Section 3.3 for details.

1.3.3 Model-based RL

In this section, we give a brief introduction to **model-based RL**; for more details, see Chapter 4.

Value-based methods, such as Q-learning, and policy search methods, such as policy gradient, can be very **sample inefficient**, which means they may need to interact with the environment many times before finding a good policy, which can be problematic when real-world interactions are expensive. In model-based RL, we first learn the MDP, including the $p_S(s'|s, a)$ and $R(s, a)$ functions, and then compute the policy, either using approximate dynamic programming on the learned model, or doing lookahead search. In practice, we often interleave the model learning and planning phases, so we can use the partially learned policy to decide what data to collect, to help learn a better model.

1.3.4 Dealing with partial observability

In an MDP, we assume that the state of the environment s_t is the same as the observation o_t obtained by the agent. But in many problems, the observation only gives partial information about the underlying state of the world (e.g., a rodent or robot navigating in a maze). This is called **partial observability**. In this case, using a policy of the form $a_t = \pi(o_t)$ is suboptimal, since o_t does not give us complete state information. Instead we need to use a policy of the form $a_t = \pi(\boldsymbol{h}_t)$, where $\boldsymbol{h}_t = (a_1, o_1, \ldots, a_{t-1}, o_t)$ is the entire past history of observations and actions, plus the current observation. Since depending on the entire past is not tractable for a long-lived agent, various approximate solution methods have been developed, as we summarize below.

1.3.4.1 Optimal solution

If we know the true latent structure of the world (i.e., both $p(o|z)$ and $p(z'|z, a)$, to use the notation of Section 1.1.2), then we can use solution methods designed for POMDPs, discussed in Section 1.2.1. This requires using Bayesian inference to compute a belief state, $\boldsymbol{b}_t = p(\boldsymbol{z}_t|\boldsymbol{h}_t)$ (see Section 1.2.5), and then using this belief state to guide our decisions. However, learning the parameters of a POMDP (i.e., the generative latent world model) is very difficult, as is recursively computing and updating the belief state, as is computing the policy given the belief state. Indeed, optimally solving POMDPs is known to be computationally very difficult for any method [PT87; KLC98]. So in practice simpler approximations are used. We discuss some of these below. (For more details, see [Mur00].)

Note that it is possible to marginalize out the POMDP latent state z_t, to derive a prediction over the next observable state, $p(\boldsymbol{o}_{t+1}|\boldsymbol{h}_t, \boldsymbol{a}_t)$. This can then become a learning target for a model, that is trained to directly predict future observations, without explicitly invoking the concept of latent state. This is called a **predictive state representation** or **PSR** [LS01]. This is related to the idea of **observable operator models** [Jae00], and to the concept of successor representations which we discuss in Section 4.4.2.

1.3.4.2 Finite observation history

The simplest solution to the partial observability problem is to define the state to be a finite history of the last k observations, $\boldsymbol{s}_t = \boldsymbol{h}_{t-k:t}$; when the observations $\boldsymbol{o}_t$ are images, this is often called **frame stacking**. We can then use standard MDP methods. Unfortunately, this cannot capture long-range dependencies in the data.

1.3.4.3 Stateful (recurrent) policies

A more powerful approach is to use a stateful policy, that can remember the entire past, and not just respond to the current input or last k frames. For example, we can represent the policy by an RNN (recurrent neural network), as proposed in the **R2D2** paper [Kap+18], and used in many other papers. Now the hidden state $\boldsymbol{z}_t$ of the RNN will implicitly summarize the past observations, $\boldsymbol{h}_t$, and can be used in lieu of the state s_t in any standard RL algorithm.

RNNs policies are widely used, and this method is often effective in solving partially observed problems. However, they typically will not plan to perform information-gathering actions, since there is no explicit notion of belief state or uncertainty. However, such behavior can arise via meta-learning [Mik+20].

1.3.5 Software

Implementing RL algorithms is much trickier than methods for supervised learning, or generative methods such as language modeling and diffusion, all of which have stable (easy-to-optimize) loss functions. Therefore it is often wise to build on existing software rather than starting from scratch. We list some useful libraries in Section 1.3.5.

In addition, RL experiments can be very high variance, making it hard to draw valid conclusions. See [Aga+21b; Pat+24; Jor+24] for some recommended experimental practices. For example, when reporting performance across different environments, with different intrinsic difficulties (e.g., different kinds of Atari

URL	Language	Comments
Stoix	Jax	Mini-library with many methods (including MBRL)
PureJaxRL	Jax	Single files with DQN; PPO, DPO
JaxRL	Jax	Single files with AWAC, DDPG, SAC, SAC+REDQ
Stable Baselines Jax	Jax	Library with DQN, CrossQ, TQC; PPO, DDPG, TD3, SAC
Jax Baselines	Jax	Library with many methods
Rejax	Jax	Library with DDQN, PPO, (discrete) SAC, DDPG
Dopamine	Jax/TF	Library with many methods
Rlax	Jax	Library of RL utility functions (used by Acme)
Acme	Jax/TF	Library with many methods (uses rlax)
CleanRL	PyTorch	Single files with many methods
Stable Baselines 3	PyTorch	Library with DQN; A2C, PPO, DDPG, TD3, SAC, HER
TianShou	PyTorch	Library with many methods (including offline RL)

Table 1.2: Some open source RL software.

games), [Aga+21b] recommend reporting the **interquartile mean** (IQM) of the performance metric, which is the mean of the samples between the 0.25 and 0.75 percentiles, (this is a special case of a trimmed mean). Let this estimate be denoted by $\hat{\mu}(\mathcal{D}_i)$, where $\mathcal{D}$ is the empirical data (e.g., reward vs time) from the i'th run. We can estimate the uncertainty in this estimate using a nonparametric method, such as bootstrap resampling, or a parametric approximation, such as a Gaussian approximation. (This requires computing the standard error of the mean, $\frac{\hat{\sigma}}{\sqrt{n}}$, where n is the number of trials, and $\hat{\sigma}$ is the estimated standard deviation of the (trimmed) data.)

1.4 Exploration-exploitation tradeoff

A fundamental problem in RL with unknown transition and reward models is to decide between choosing actions that the agent knows will yield high reward, or choosing actions whose reward is uncertain, but which may yield information that helps the agent get to parts of state-action space with even higher reward. This is called the **exploration-exploitation tradeoff**. In this section, we discuss various solutions.

1.4.1 Simple heuristics

We start with a policy based on pure exploitation. This is known as the **greedy policy**, $a_t = \text{argmax}_a Q(s, a)$. We can add exploration to this by sometimes picking some other, non-greedy action.

One approach is to use an ϵ**-greedy** policy π_ϵ, parameterized by $\epsilon \in [0, 1]$. In this case, we pick the greedy action wrt the current model, $a_t = \text{argmax}_a \hat{R}_t(s_t, a)$ with probability $1 - \epsilon$, and a random action with probability ϵ. This rule ensures the agent's continual exploration of all state-action combinations. Unfortunately, this heuristic can be shown to be suboptimal, since it explores every action with at least a constant probability $\epsilon/|\mathcal{A}|$, although this can be solved by annealing ϵ to 0 over time.

Another problem with ϵ-greedy is that it can result in "dithering", in which the agent continually changes its mind about what to do. In [DOB21] they propose a simple solution to this problem, known as ϵz**-greedy**, that often works well. The idea is that with probability $1 - \epsilon$ the agent exploits, but with with probability ϵ the agent explores by repeating the sampled action for $n \sim z()$ steps in a row, where $z(n)$ is a distribution over the repeat duration. This can help the agent escape from local minima.

Another approach is to use **Boltzmann exploration**, which assigns higher probabilities to explore more promising actions, taking itno account the reward function. That is, we use a polocy of the form

$$\pi_\tau(a|s) = \frac{\exp(\hat{R}_t(s_t, a)/\tau)}{\sum_{a'} \exp(\hat{R}_t(s_t, a')/\tau)} \tag{1.34}$$

where $\tau > 0$ is a temperature parameter that controls how entropic the distribution is. As τ gets close to 0, π_τ becomes close to a greedy policy. On the other hand, higher values of τ will make $\pi(a|s)$ more uniform,

$\hat{R}(s, a_1)$	$\hat{R}(s, a_2)$	$\pi_\epsilon(a\|s_1)$	$\pi_\epsilon(a\|s_2)$	$\pi_\tau(a\|s_1)$	$\pi_\tau(a\|s_2)$
1.00	9.00	0.05	0.95	0.00	1.00
4.00	6.00	0.05	0.95	0.12	0.88
4.90	5.10	0.05	0.95	0.45	0.55
5.05	4.95	0.95	0.05	0.53	0.48
7.00	3.00	0.95	0.05	0.98	0.02
8.00	2.00	0.95	0.05	1.00	0.00

Table 1.3: Comparison of ϵ-greedy policy (with $\epsilon = 0.1$) and Boltzmann policy (with $\tau = 1$) for a simple MDP with 6 states and 2 actions. Adapted from Table 4.1 of [GK19].

and encourage more exploration. Its action selection probabilities can be much "smoother" with respect to changes in the reward estimates than ϵ-greedy, as illustrated in Table 1.3.

The Boltzmann policy explores equally widely in all states. An alternative approach is to try to explore (state,action) combinations where the consequences of the outcome might be uncertain. This can be achived using an **exploration bonus** $R_t^b(s, a)$, which is large if the number of times we have tried actioon a in state s is small. We can then add R^b to the regular reward, to bias the behavior in a way that will hopefully cause the agent to learn useful information about the world. This is called an **intrinsic reward** function (Section 5.2.4).

1.4.2 Methods based on the belief state MDP

We can compute an optimal solution to the exploration-exploitation tradeoff by adopting a Bayesian approach to the problem. We start by computing the belief state MDP, as discussed in Section 1.2.5. We then compute the optimal policy, as we explain below.

1.4.2.1 Bandit case (Gittins indices)

Suppose we have a way to compute the recursively compute the belief state over model parameters, $p(\boldsymbol{\theta}_t|\mathcal{D}_{1:t})$. How do we use this to solve for the policy in the resulting belief state MDP?

In the special case of context-free bandits with a finite number of arms, the optimal policy of this belief state MDP can be computed using dynamic programming. The result can be represented as a table of action probabilities, $\pi_t(a_1, \ldots, a_K)$, for each step; this are known as **Gittins indices** [Git89] (see [PR12; Pow22] for a detailed explanation). However, computing the optimal policy for general contextual bandits is intractable [PT87].

1.4.2.2 MDP case (Bayes Adaptive MDPs)

We can extend the above techniques to the MDP case by constructing a **BAMDP**, which stands for "Bayes-Adaptive MDP" [Duf02]. However, this is computationally intractable to solve, so various approximations are made (see e.g., [Zin+21; AS22; Mik+20]).

1.4.3 Upper confidence bounds (UCBs)

The optimal solution to explore-exploit is intractable. However, an intuitively sensible approach is based on the principle known as "**optimism in the face of uncertainty**" (OFU). The principle selects actions greedily, but based on optimistic estimates of their rewards. The optimality of this approach is proved in the **R-Max** paper of [Ten02], which builds on the earlier **E3** paper of [KS02].

The most common implementation of this principle is based on the notion of an **upper confidence bound** or **UCB**. We will initially explain this for the bandit case, then extend to the MDP case.

1.4.3.1 Basic idea

To use a UCB strategy, the agent maintains an optimistic reward function estimate $\tilde{R}_t$, so that $\tilde{R}_t(s_t, a) \geq R(s_t, a)$ for all a with high probability, and then chooses the greedy action accordingly:

$$a_t = \operatorname*{argmax}_a \tilde{R}_t(s_t, a) \tag{1.35}$$

UCB can be viewed a form of **exploration bonus**, where the optimistic estimate encourages exploration. Typically, the amount of optimism, $\tilde{R}_t - R$, decreases over time so that the agent gradually reduces exploration. With properly constructed optimistic reward estimates, the UCB strategy has been shown to achieve near-optimal regret in many variants of bandits [LS19]. (We discuss regret in Section 1.1.4.)

The optimistic function $\tilde{R}$ can be obtained in different ways, sometimes in closed forms, as we discuss below.

1.4.3.2 Bandit case: Frequentist approach

A frequentist approach to computing a confidence bound can be based on a **concentration inequality** [BLM16] to derive a high-probability upper bound of the estimation error: $|\hat{R}_t(s, a) - R_t(s, a)| \leq \delta_t(s, a)$, where $\hat{R}_t$ is a usual estimate of R (often the MLE), and δ_t is a properly selected function. An optimistic reward is then obtained by setting $\tilde{R}_t(s, a) = \hat{R}_t(s, a) + \delta_t(s, a)$.

As an example, consider again the context-free Bernoulli bandit, $R(a) \sim \text{Ber}(\mu(a))$. The MLE $\hat{R}_t(a) = \hat{\mu}_t(a)$ is given by the empirical average of observed rewards whenever action a was taken:

$$\hat{\mu}_t(a) = \frac{N_t^1(a)}{N_t(a)} = \frac{N_t^1(a)}{N_t^0(a) + N_t^1(a)} \tag{1.36}$$

where $N_t^r(a)$ is the number of times (up to step $t-1$) that action a has been tried and the observed reward was r, and $N_t(a)$ is the total number of times action a has been tried:

$$N_t(a) = \sum_{s=1}^{t-1} \mathbb{I}(a_t = a) \tag{1.37}$$

Then the **Chernoff-Hoeffding inequality** [BLM16] leads to $\delta_t(a) = c/\sqrt{N_t(a)}$ for some constant c, so

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + \frac{c}{\sqrt{N_t(a)}} \tag{1.38}$$

1.4.3.3 Bandit case: Bayesian approach

We can also derive an upper confidence about using Bayesian inference. If we use a beta prior, we can compute the posterior in closed form, as shown in Equation (1.23). The posterior mean is $\hat{\mu}_t(a) = \mathbb{E}[\mu(a)|\boldsymbol{h}_t] = \frac{\alpha_t^a}{\alpha_t^a + \beta_t^a}$, and the posterior standard deviation is approximately

$$\hat{\sigma}_t(a) = \sqrt{\mathbb{V}[\mu(a)|\boldsymbol{h}_t]} \approx \sqrt{\frac{\hat{\mu}_t(a)(1 - \hat{\mu}_t(a))}{N_t(a)}} \tag{1.39}$$

We can use similar techniques for a Gaussian bandit, where $p_R(R|a, \boldsymbol{\theta}) = \mathcal{N}(R|\mu_a, \sigma_a^2)$, μ_a is the expected reward, and σ_a^2 the variance. If we use a conjugate prior, we can compute $p(\mu_a, \sigma_a|\mathcal{D}_t)$ in closed form. Using an uninformative version of the conjugate prior, we find $\mathbb{E}[\mu_a|\boldsymbol{h}_t] = \hat{\mu}_t(a)$, which is just the empirical mean of rewards for action a. The uncertainty in this estimate is the standard error of the mean, i.e., $\sqrt{\mathbb{V}[\mu_a|\boldsymbol{h}_t]} = \hat{\sigma}_t(a)/\sqrt{N_t(a)}$, where $\hat{\sigma}_t(a)$ is the empirical standard deviation of the rewards for action a.

Once we have computed the mean and posterior standard deviation, we define the optimistic reward estimate as

$$\tilde{R}_t(a) = \hat{\mu}_t(a) + c\hat{\sigma}_t(a) \tag{1.40}$$

for some constant c that controls how greedy the policy is. See Figure 1.5 for an illustration. We see that this is similar to the frequentist method based on concentration inequalities, but is more general.
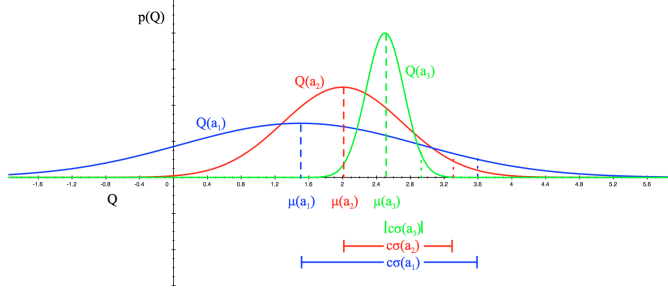
Figure 1.5: *Illustration of the reward distribution $Q(a)$ for a Gaussian bandit with 3 different actions, and the corresponding lower and upper confidence bounds. We show the posterior means $Q(a) = \mu(a)$ with a vertical dotted line, and the scaled posterior standard deviations $c\sigma(a)$ as a horizontal solid line. From [Sil18]. Used with kind permission of David Silver.*

1.4.3.4 MDP case

The UCB idea (especially in its frequentist form) has been extended to the MDP case in several works. (The Bayesian version is discussed in Section 1.4.4.) For example, [ACBF02] proposes to combine UCB with Q learning, by defining the policy as

$$\pi(a|s) = \mathbb{I}\left(a = \operatorname*{argmax}_{a'} Q(s, a') + c\sqrt{\log(t)/N_t(s, a')}\right) \tag{1.41}$$

[AJO08] presents the more sophisticated **UCRL2** algorithm, which computes confidence intervals on all the MDP model parameters at the start of each episode; it then computes the resulting **optimistic MDP** and solves for the optimal policy, which it uses to collect more data.

1.4.4 Thompson sampling

A common alternative to UCB is to use **Thompson sampling** [Tho33], also called **probability matching** [Sco10]. We start by describing this in the bandit case, then extend to the MDP case. For more details, see [Rus+18]. (See also [Ger18] for some evidence that humans use Thompson-sampling like mechanisms.)

1.4.4.1 Bandit case

In Thompson sampling, we define the policy at step t to be $\pi_t(a|s_t, \boldsymbol{h}_t) = p_a$, where p_a is the probability that a is the optimal action. This can be computed using

$$p_a = \Pr(a = a_* | s_t, \boldsymbol{h}_t) = \int \mathbb{I}\left(a = \operatorname*{argmax}_{a'} R(s_t, a'; \boldsymbol{\theta})\right) p(\boldsymbol{\theta}|\boldsymbol{h}_t) d\boldsymbol{\theta} \tag{1.42}$$

If the posterior is uncertain, the agent will sample many different actions, automatically resulting in exploration. As the uncertainty decreases, it will start to exploit its knowledge.

To see how we can implement this method, note that we can compute the expression in Equation (1.42) by using a single Monte Carlo sample $\tilde{\boldsymbol{\theta}}_t \sim p(\boldsymbol{\theta}|\boldsymbol{h}_t)$. We then plug in this parameter into our reward model, and greedily pick the best action:

$$a_t = \operatorname*{argmax}_{a'} R(s_t, a'; \tilde{\boldsymbol{\theta}}_t) \tag{1.43}$$

This sample-then-exploit approach will choose actions with exactly the desired probability, since

$$p_a = \int \mathbb{I}\left(a = \operatorname*{argmax}_{a'} R(s_t, a'; \tilde{\boldsymbol{\theta}}_t)\right) p(\tilde{\boldsymbol{\theta}}_t|\boldsymbol{h}_t) = \Pr_{\tilde{\boldsymbol{\theta}}_t \sim p(\boldsymbol{\theta}|\boldsymbol{h}_t)}(a = \operatorname*{argmax}_{a'} R(s_t, a'; \tilde{\boldsymbol{\theta}}_t)) \tag{1.44}$$
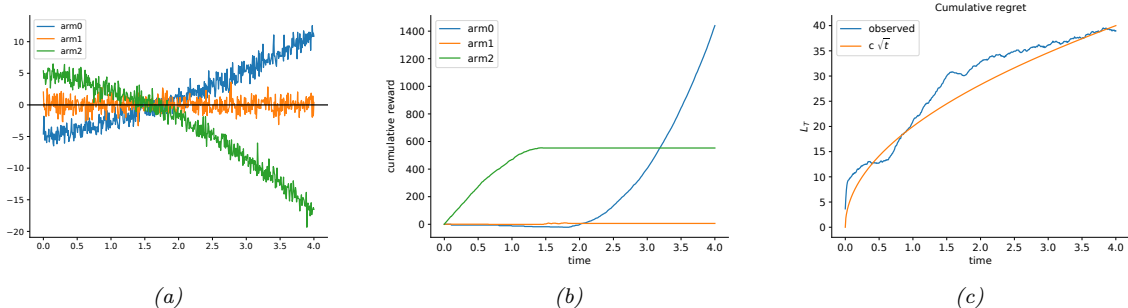
Figure 1.6: Illustration of Thompson sampling applied to a linear-Gaussian contextual bandit. The context has the form $\boldsymbol{s}_t = (1, t, t^2)$. (a) True reward for each arm vs time. (b) Cumulative reward per arm vs time. (c) Cumulative regret vs time. Generated by thompson_sampling_linear_gaussian.ipynb.

Despite its simplicity, this approach can be shown to achieve optimal regret (see e.g., [Rus+18] for a survey). In addition, it is very easy to implement, and hence is widely used in practice [Gra+10; Sco10; CL11].

In Figure 1.6, we give a simple example of Thompson sampling applied to a linear regression bandit. The context has the form $\boldsymbol{s}_t = (1, t, t^2)$. The true reward function has the form $R(\boldsymbol{s}_t, a) = \boldsymbol{w}_a^\mathsf{T} \boldsymbol{s}_t$. The weights per arm are chosen as follows: $\boldsymbol{w}_0 = (-5, 2, 0.5)$, $\boldsymbol{w}_1 = (0, 0, 0)$, $\boldsymbol{w}_2 = (5, -1.5, -1)$. Thus we see that arm 0 is initially worse (large negative bias) but gets better over time (positive slope), arm 1 is useless, and arm 2 is initially better (large positive bias) but gets worse over time. The observation noise is the same for all arms, $\sigma^2 = 1$. See Figure 1.6(a) for a plot of the reward function. We use a conjugate Gaussian-gamma prior and perform exact Bayesian updating. Thompson sampling quickly discovers that arm 1 is useless. Initially it pulls arm 2 more, but it adapts to the non-stationary nature of the problem and switches over to arm 0, as shown in Figure 1.6(b). In Figure 1.6(c), we show that the empirical cumulative regret in blue is close to the optimal lower bound in red.

1.4.4.2 MDP case (posterior sampling RL)

We can generalize Thompson sampling to the (episodic) MDP case by maintaining a posterior over all the model parameters (reward function and transition model), sampling an MDP from this belief state at the start of each episode, solving for the optimal policy corresponding to the sampled MDP, using the resulting policy to collect new data, and then updating the belief state at the end of the episode. This is called **posterior sampling RL** [Str00; ORVR13; RR14; OVR17; WCM24].

As a more computationally efficient alternative, it is also possible to maintain a posterior over policies or Q functions instead of over world models; see e.g., [Osb+23a] for an implementation of this idea based on **epistemic neural networks** [Osb+23b]. Another approach is to use successor features (Section 4.4.4), where the Q function is assumed to have the form $Q^\pi(s, a) = \boldsymbol{\psi}^\pi(s, a)^\mathsf{T} \boldsymbol{w}$. In particular, [Jan+19b] proposes **Sucessor Uncertainties**, in which they model the uncertainty over $\boldsymbol{w}$ as a Gaussian, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu_w}, \boldsymbol{\Sigma_w})$. From this they can derive the posterior distribution over Q values as

$$p(Q(s, a)) = \mathcal{N}(\boldsymbol{\Psi}^\pi \boldsymbol{\mu_w}, \boldsymbol{\Psi}^\pi \boldsymbol{\Sigma_w} (\boldsymbol{\Psi}^\pi)^\mathsf{T}) \tag{1.45}$$

where $\boldsymbol{\Psi}^\pi = [\boldsymbol{\psi}^\pi(s, a)]^\mathsf{T}$ is a matrix of features, one per state-action pair.

1.5 RL as a posterior inference problem

In this section, we discuss an approach to policy optimization that reduces it to probabilistic inference. This is called **control as inference** or **RL as inference**, and has been discussed in numerous works (see e.g., [Att03; TS06; Tou09; ZABD10; RTV12; BT12; KGO12; HR17; Lev18; Wat+21]). The resulting framework
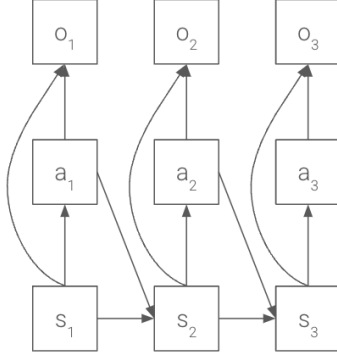
Figure 1.7: A graphical model for optimal control.

also forms the foundation of the SAC method discussed in Section 3.5.4, the MPO discussed in Section 3.4.4, and the MPC method discussed in Section 4.1.5.

1.5.1 Modeling assumptions

Figure 1.7 gives a probabilistic model, which not only captures state transitions as in a standard MDP, but also introduces a new variable, $\mathcal{O}_t$. This variable is binary, indicating whether the action at time t is optimal or not, and has the following probability distribution:

$$p(\mathcal{O}_t = 1|s_t, a_t) = \exp(R(s_t, a_t)) \tag{1.46}$$

In the above, we have assumed that $R(s, a) < 0$, so that Equation (1.46) gives a valid probability. However, this is not required, since we can simply replace the likelihood term $p(\mathcal{O}_t = 1|s_t, a_t)$ with an unnormalized potential, $\phi_t(s_t, a_t)$; this will not affect the results of inference. For brevity, we will just write $p(\mathcal{O}_t)$ rather than $p(\mathcal{O}_t = 1)$, since 1 is just a dummy value.

To simplify notation, we assume a uniform action prior, $p(a_t|s_t) = 1/|\mathcal{A}|$; this is without loss of generality, since we can always push an informative action prior $p(a_t|s_t)$ into the potential function $\phi_t(s_t, a_t)$. (We call this an "action prior" rather than a policy, since we are going to derive the policy using posterior inference, as we explain below.) Under these assumptions, the posterior probability of observing a length-T trajectory $\boldsymbol{\tau}$, when optimality achieved in every step, is

$$p(\boldsymbol{\tau}|\mathcal{O}_{1:T}) \propto p(\boldsymbol{\tau}, \mathcal{O}_{1:T}) \propto \left[p(s_1) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t) \right] \left[\prod_{t=1}^{T} p(\mathcal{O}_t|s_t, a_t) \right]$$

$$= p(s_1) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t) \exp\left(\sum_{t=1}^{T} R(s_t, a_t) \right) \tag{1.47}$$

(Typically $p(s_1)$ is a delta function at the observed initial state s_1.) The intuition of Equation (1.47) is clearest when the state transitions are deterministic. In this case, $p_S(s_{t+1}|s_t, a_t)$ is either 1 or 0, depending on whether the transition is dynamically feasible or not. Hence we have

$$p(\boldsymbol{\tau}|\mathcal{O}_{1:T}) \propto \mathbb{I}\left(p(\boldsymbol{\tau}) \neq 0 \right) \exp(\sum_{t=1}^{T} R(s_t, a_t)) \tag{1.48}$$

where the first term determines if $\boldsymbol{\tau}$ is feasible or not. In this case, find the action sequence that maximizes the sum of rewards is equivalent to inferring the MAP sequence of actions, which we denote by $\hat{\boldsymbol{a}}_{1:T}(s_1)$. (The case of stochastic transitions is more complicated, and will be discussed later.)

For deterministic environments, the optimal policy is **open loop**, and corresponds to following the optimal action sequence $\hat{a}_{1:T}(s_1)$. (This is like a shortest path planning problem.) However, in the stochastic case, we need to compute a **closed loop** policy, $\pi(a_t|s_t)$, that conditions on the observed state. To compute this, let us define the following quantities:

$$\beta_t(s_t, a_t) \triangleq p(\mathcal{O}_{t:T}|s_t, a_t) \tag{1.49}$$

$$\beta_t(s_t) \triangleq p(\mathcal{O}_{t:T}|s_t) \tag{1.50}$$

(These terms are analogous to the **backwards messages** in the forwards-backwards algorithm for HMMs [Rab89].) Using this notation, we can write the optimal policy using

$$p(a_t|s_t, \mathcal{O}_{t:T}) = \frac{p(s_t, a_t|\mathcal{O}_{t:T})}{p(s_t|\mathcal{O}_{t:T})} = \frac{p(\mathcal{O}_{t:T}|s_t, a_t)p(a_t|s_t)p(s_t)}{p(\mathcal{O}_{t:T}|s_t)p(s_t)} \propto \frac{\beta_t(s_t, a_t)}{\beta_t(s)} \tag{1.51}$$

We can compute the backwards messages as follows:

$$\beta_t(s_t, a_t) = \int_{\mathcal{S}} \beta_{t+1}(s_{t+1})p_S(s_{t+1}|s_t, a_t)p(\mathcal{O}_t|s_t, a_t)ds_{t+1} \tag{1.52}$$

$$\beta_s(s_t) = \int_{\mathcal{A}} \beta_t(s_t, a_t)p(a_t|s_t)da_t \propto \int_{\mathcal{A}} \beta_t(s_t, a_t)da_t \tag{1.53}$$

where we have assumed the action prior $p(a_t|s_t) = 1/|\mathcal{A}|$ for notational simplicty. (Recall that the action prior is distinct from the optimal policy, which is given by $p(a_t|s_t, \mathcal{O}_{t:T})$.)

1.5.2   Soft value functions

We can gain more insight into what is going on by working in log space. Let us define

$$Q(s_t, a_t) = \log \beta_t(s_t, a_t) \tag{1.54}$$

$$V(s_t) = \log \beta_t(s_t) \tag{1.55}$$

The update for V becomes

$$V(s_t) = \log \sum_{a_t} \exp(Q(s_t, a_t)) \tag{1.56}$$

This is a standard log-sum-exp computation, and is similar to the softmax operation. Thus we call it a **soft value function**. When the values of $Q(s_t, a_t)$ are large (which can be ensure by scaling up all the rewards), this approximates the standard hard max operation:

$$V(s_t) = \log \sum_{a_t} \exp(Q(s_t, a_t)) \approx \max_{a_t} Q(s_t, a_t) \tag{1.57}$$

For the deterministic case, the backup for Q becomes the usual

$$Q(s_t, a_t) = \log p(\mathcal{O}_t|s_t, a_t) + \log \beta_{t+1}(s_{t+1}) = r(s_t, a_t) + V(s_{t+1}) \tag{1.58}$$

where $s_{t+1} = f(s_t, a_t)$ is the next state. However, for the stochastic case, we get

$$Q(s_t, a_t) = r(s_t, a_t) + \log \mathbb{E}_{p_S(s_{t+1}|s_t, a_t)}[\exp(V(s_{t+1}))] \tag{1.59}$$

This replaces the standard expectation over the next state with a softmax. This can result in Q functions that are optimistic, since if there is one next state with particularly high reward (e.g., you win the lottery), it will dominate the backup, even if on average it is unlikely. This can result in risk seeking behavior, and is known as the **optimism bias** (see e.g., [Mad+17; Cha+21] for discussion). We will discuss a solution to this below.

1.5.3 Maximum entropy RL

Recall that the true posterior is given by

$$p(\boldsymbol{\tau}|\mathcal{O}_{1:T}) \triangleq p^*(\boldsymbol{\tau}) \propto p(s_1) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t) \exp\left(\sum_{t=1}^{T} R(s_t, a_t)\right) \tag{1.60}$$

In the sections above, we derived the exact posterior over states and actions conditioned on the optimality variables. However, in general we will have to approximate it.

Let us denote the approximate posterior by $q(\boldsymbol{\tau})$. Variational inference corresponds to the minimizing (wrt q) the following objective:

$$D_{\mathbb{KL}}(q(\boldsymbol{\tau}) \parallel p^*(\boldsymbol{\tau})) = -\mathbb{E}_{q(\boldsymbol{\tau})}[\log p^*(\boldsymbol{\tau}) - \log q(\boldsymbol{\tau})] \tag{1.61}$$

We can drive this loss to its minimum value of 0 by performing exact inference, which sets $q(\boldsymbol{\tau}) = p^*(\boldsymbol{\tau})$, which is given by

$$p^*(\boldsymbol{\tau}) = p(s_1|\mathcal{O}_{1:T}) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t, \mathcal{O}_{1:T})p(a_t|s_t, \mathcal{O}_{1:T})) \tag{1.62}$$

Unfortunately, this uses an optimistic form of the dynamics, $p_S(s_{t+1}|s_t, a_t, \mathcal{O}_{1:T})$, in which the agent plans assuming it directly controls the state distribution itself, rather than just the action distribution. We can solve this optimism bias problem by instead using a "causal" variational posterior of the following form:[8]

$$q(\boldsymbol{\tau}) = p(s_1) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t)p(a_t|s_t, \mathcal{O}_{1:T}) = p(s_1) \prod_{t=1}^{T-1} p_S(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{1.63}$$

where $\pi(a_t|s_t)$ is the policy we wish to learn. In the case of deterministic transitions, where $p_S(s_{t+1}|s_t, a_t) = \delta(s_{t+1} - f(s_t, a_t))$, we do not need this simplification, since $p_S(s_{t+1}|s_t, a_t, \mathcal{O}_{1:T}) = p_S(s_{t+1}|s_t, a_t)$. (And in both cases $p(s_1|\mathcal{O}_{1:T}) = p(s_1)$, which is assumed to be a delta function.) We can now write the (negative of) the objective as follows:

$$-D_{\mathbb{KL}}(q(\boldsymbol{\tau}) \parallel p^*(\boldsymbol{\tau})) = \mathbb{E}_{q(\boldsymbol{\tau})}\left[\log p(s_1) + \sum_{t=1}^{T}(\log p_S(s_{t+1}|s_t, a_t) + R(s_t, a_t)) - \tag{1.64}\right.$$

$$\left. - \log p(s_1) - \sum_{t=1}^{T}(\log p_S(s_{t+1}|s_t, a_t) + \log \pi(a_t|s_t))\right] \tag{1.65}$$

$$= \mathbb{E}_{q(\boldsymbol{\tau})}\left[\sum_{t=1}^{T} R(s_t, a_t) - \log \pi(a_t|s_t)\right] \tag{1.66}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{q(s_t, a_t)}[R(s_t, a_t)] + \mathbb{E}_{q(s_t)} \mathbb{H}(\pi(\cdot|s_t)) \tag{1.67}$$

This is known as the **maximum entropy RL** objective [ZABD10].We can optimize this using the soft actor critic algorithm which we discuss in Section 3.5.4.

Note that we can tune the magnitude of the entropy regularizer by defining the optimality variable using $p(\mathcal{O}_t = 1|s_t, a_t) = \exp(\frac{1}{\alpha}R(s_t, a_t))$. This gives the objective

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{q(s_t, a_t)}[R(s_t, a_t)] + \alpha \mathbb{E}_{q(s_t)} \mathbb{H}(\pi(\cdot|s_t)) \tag{1.68}$$

As $\alpha \to 0$ (equivalent to scaling up the rewards), this approaches the standard (unregularized) RL objective.

---

[8]Unfortunately, this trick is specific to variational inference, which means that other posterior inference methods, such as sequential Monte Carlo [Pic+19; Lio+22], will still suffer from the optimism bias in the stochastic case (see e.g., [Mad+17] for discussion).

1.5.4 Active inference

Control as inference is closely related to a technique known as **active inference**, as we explain below. For more details on the connection, see [Mil+20; WIP20; LÖW21; Saj+21; Tsc+20].

The active inference technique was developed in the neuroscience community, that has its own vocabulary for standard ML concepts. We start with the **free energy principle** [Fri09; Buc+17; SKM18; Ger19; Maz+22]. The FEP is equivalent to using variational inference to perform state estimation (perception) and parameter estimation (learning) in a latent variable model. In particular, consider an LVM $p(\boldsymbol{z}, \boldsymbol{o}|\boldsymbol{\theta})$ with hidden states $\boldsymbol{z}$, observations $\boldsymbol{o}$, and parameters $\boldsymbol{\theta}$. We define the variational free energy to be

$$\mathcal{F}(\boldsymbol{o}|\boldsymbol{\theta}) = D_{\mathbb{KL}}\left(q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta}) \parallel p(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta})\right) - \log p(\boldsymbol{o}|\boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta})}\left[\log q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta}) - \log p(\boldsymbol{o}, \boldsymbol{z}|\boldsymbol{\theta})\right] \geq -\log p(\boldsymbol{o}|\boldsymbol{\theta}) \tag{1.69}$$

which is the KL between the approximate variational posterior q and the true posterior p, minus a normalization constant, $\log p(\boldsymbol{o}|\boldsymbol{\theta})$, which is known as the free energy. State estimation (perception) corresponds to solving $\min_{q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta})} \mathcal{F}(\boldsymbol{o}|\boldsymbol{\theta})$, and parameter estimation (model fitting) corresponds to solving $\min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{o}|\boldsymbol{\theta})$, just as in the EM (expectation maximization) algorithm. (We can also be Bayesian about $\boldsymbol{\theta}$, as in variational Bayes EM, instead of just computing a point estimate.) This EM procedure will minimize the VFE, which is an upper bound on the negative log marginal likelihood of the data. In other words, it adjusts the model (belief state and parameters) so that it better predicts the observations, so the agent is less surprised (minimizes prediction errors).

To extend the above FEP to decision making problems, we define the **expected free energy** as follows

$$\mathcal{G}(\boldsymbol{a}) = \mathbb{E}_{q(\boldsymbol{o}|\boldsymbol{a})}\left[\mathcal{F}(\boldsymbol{o})\right] = \mathbb{E}_{q(\boldsymbol{o}, \boldsymbol{z}|\boldsymbol{a})}\left[\log q(\boldsymbol{z}|\boldsymbol{o}) - \log p(\boldsymbol{o}, \boldsymbol{z})\right] \tag{1.70}$$

where $q(\boldsymbol{o}|\boldsymbol{a})$ is the posterior predictive distribution over future observations given action sequence $\boldsymbol{a}$. (We can also condition on any observed history or agent state $\boldsymbol{h}$, but we omit this (and the model parameters $\boldsymbol{\theta}$) from the notation for brevity.) We can decompose the EFE (which the agent wants to minimize) into two terms. First there is the **intrinsic value**, known as the **epistemic drive**:

$$\mathcal{G}_{\text{epistemic}}(\boldsymbol{a}) = \mathbb{E}_{q(\boldsymbol{o}, \boldsymbol{z}|\boldsymbol{a})}\left[\log q(\boldsymbol{z}|\boldsymbol{o}) - \log q(\boldsymbol{z})\right] \tag{1.71}$$

Minimizing this will encourage the agent to choose actions which maximize the mutual information between the observations $\boldsymbol{o}$ and the hidden states $\boldsymbol{z}$, thus reducing uncertainty about the hidden states. (This is called **epistemic foraging**.) The **extrinsic value**, known as the **exploitation term**, is given by

$$\mathcal{G}_{\text{extrinsic}}(\boldsymbol{a}) = -\mathbb{E}_{q(\boldsymbol{o}|\boldsymbol{a})}\left[\log p(\boldsymbol{o})\right] \tag{1.72}$$

Minimizing this will encourage the agent to choose actions that result in observations that match its prior. For example, if the agent predicts that the world will look brighter when it flips a light switch, it can take the action of flipping the switch to fulfill this prediction. This prior can be related to a reward function by defining as $p(\boldsymbol{o}) \propto e^{R(\boldsymbol{o})}$, encouraging goal directed behavior, exactly as in control-as-inference. However, the active inference approach provides a way of choosing actions without needing to specify a reward. Since solving to the optimal action at each step can be slow, it is possible to amortize this cost by training a policy network to compute $\pi(\boldsymbol{a}|\boldsymbol{h}) = \operatorname{argmin}_{\boldsymbol{a}} \mathcal{G}(\boldsymbol{a}|\boldsymbol{h})$, where $\boldsymbol{h}$ is the observation history (or current state), as shown in [Mil20; HL20]; this is called **"deep active inference"**.

Overall, we see that this framework provides a unified theory of both perception and action, both of which try to minimize some form of free energy. In particular, minimizing the expected free energy will cause the agent to pick actions to reduce its uncertainty about its hidden states, which can then be used to improve its predictive model $p_{\boldsymbol{\theta}}$ of observations; this in turn will help minimize the VFE of future observations, by updating the internal belief state $q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta})$ to explain the observations. In other words, the agent acts so it can learn so it becomes less surprised by what it sees. This ensures the agent is in **homeostasis** with its environment.

Note that active inference is often discussed in the context of **predictive coding**. This is equivalent to a special case of FEP where two assumptions are made: (1) the generative model $p(\boldsymbol{z}, \boldsymbol{o}|\boldsymbol{\theta})$ is a a nonlinear

hierarchical Gaussian model (similar to a VAE decoder), and (2) the variational posterior approximation uses a diagonal Laplace approximation, $q(\boldsymbol{z}|\boldsymbol{o}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{z}|\hat{\boldsymbol{z}}, \mathbf{H})$ with the mode $\hat{\boldsymbol{z}}$ being computed using gradient descent, and $\mathbf{H}$ being the Hessian at the mode. This can be considered a non-amortized version of a VAE, where inference (E step) is done with iterated gradient descent, and parameter estimation (M step) is also done with gradient descent. (A more efficient incremental EM version of predictive coding, which updates $\{\hat{\boldsymbol{z}}_n : n = 1 : N\}$ and $\boldsymbol{\theta}$ in parallel, was recently presented in [Sal+24], and an amortized version in [Tsc+23].) For more details on predictive coding, see [RB99; Fri03; Spr17; HM20; MSB21; Mar21; OK22; Sal+23; Sal+24].

Chapter 2

Value-based RL

2.1 Basic concepts

In this section we introduce some definitions and basic concepts.

2.1.1 Value functions

Let π be a given policy. We define the **state-value function**, or **value function** for short, as follows (with $\mathbb{E}_\pi [\cdot]$ indicating that actions are selected by π):

$$V_\pi(s) \triangleq \mathbb{E}_\pi [G_0|s_0 = s] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right] \tag{2.1}$$

This is the expected return obtained if we start in state s and follow π to choose actions in a continuing task (i.e., $T = \infty$).

Similarly, we define the **state-action value function**, also known as the Q-**function**, as follows:

$$Q_\pi(s, a) \triangleq \mathbb{E}_\pi [G_0|s_0 = s, a_0 = a] = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right] \tag{2.2}$$

This quantity represents the expected return obtained if we start by taking action a in state s, and then follow π to choose actions thereafter.

Finally, we define the **advantage function** as follows:

$$A_\pi(s, a) \triangleq Q_\pi(s, a) - V_\pi(s) \tag{2.3}$$

This tells us the benefit of picking action a in state s then switching to policy π, relative to the baseline return of always following π. Note that $A_\pi(s, a)$ can be both positive and negative, and $\mathbb{E}_{\pi(a|s)} [A_\pi(s, a)] = 0$ due to a useful equality: $V_\pi(s) = \mathbb{E}_{\pi(a|s)} [Q_\pi(s, a)]$.

2.1.2 Bellman's equations

Suppose π_* is a policy such that $V_{\pi_*} \geq V_\pi$ for all $s \in \mathcal{S}$ and all policy π, then it is an **optimal policy**. There can be multiple optimal policies for the same MDP, but by definition their value functions must be the same, and are denoted by V_* and Q_*, respectively. We call V_* the **optimal state-value function**, and Q_* the **optimal action-value function**. Furthermore, any finite MDP must have at least one deterministic optimal policy [Put94].

A fundamental result about the optimal value function is **Bellman's optimality equations**:

$$V_*(s) = \max_a R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)} \left[V_*(s') \right] \tag{2.4}$$

$$Q_*(s,a) = R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)} \left[\max_{a'} Q_*(s',a') \right] \tag{2.5}$$

Conversely, the optimal value functions are the only solutions that satisfy the equations. In other words, although the value function is defined as the expectation of a sum of infinitely many rewards, it can be characterized by a recursive equation that involves only one-step transition and reward models of the MDP. Such a recursion play a central role in many RL algorithms we will see later.

Given a value function (V or Q), the discrepancy between the right- and left-hand sides of Equations (2.4) and (2.5) are called **Bellman error** or **Bellman residual**. We can define the **Bellman operator** $\mathcal{B}$ given an MDP $M = (R,T)$ and policy π as a function that takes a value function V and derives a few value function V' that satisfies

$$V'(s) = \mathcal{B}_M^\pi V(s) \triangleq \mathbb{E}_{\pi(a|s)} \left[R(s,a) + \gamma \mathbb{E}_{T(s'|s,a)} \left[V(s') \right] \right] \tag{2.6}$$

This reduces the Bellman error. Applying the Bellman operator to a state is called a **Bellman backup**. If we iterate this process, we will converge to the optimal value function V_*, as we discuss in Section 2.2.1.

Given the optimal value function, we can derive an optimal policy using

$$\pi_*(s) = \operatorname*{argmax}_a Q_*(s,a) \tag{2.7}$$

$$= \operatorname*{argmax}_a \left[R(s,a) + \gamma \mathbb{E}_{p_S(s'|s,a)} \left[V_*(s') \right] \right] \tag{2.8}$$

Following such an optimal policy ensures the agent achieves maximum expected return starting from any state.

The problem of solving for V_*, Q_* or π_* is called **policy optimization**. In contrast, solving for V_π or Q_π for a given policy π is called **policy evaluation**, which constitutes an important subclass of RL problems as will be discussed in later sections. For policy evaluation, we have similar Bellman equations, which simply replace $\max_a\{\cdot\}$ in Equations (2.4) and (2.5) with $\mathbb{E}_{\pi(a|s)}[\cdot]$.

In Equations (2.7) and (2.8), as in the Bellman optimality equations, we must take a maximum over all actions in $\mathcal{A}$, and the maximizing action is called the **greedy action** with respect to the value functions, Q_* or V_*. Finding greedy actions is computationally easy if $\mathcal{A}$ is a small finite set. For high dimensional continuous spaces, see Section 2.5.4.1.

2.1.3   Example: 1d grid world

In this section, we show a simple example, to make some of the above concepts more concrete. Consider the 1d **grid world** shown in Figure 2.1(a). There are 5 possible states, among them S_{T1} and S_{T2} are absorbing states, since the interaction ends once the agent enters them. There are 2 actions, $\uparrow$ and $\downarrow$. The reward function is zero everywhere except at the goal state, S_{T2}, which gives a reward of 1 upon entering. Thus the optimal action in every state is to move down.

Figure 2.1(b) shows the Q_* function for $\gamma = 0$. Note that we only show the function for non-absorbing states, as the optimal Q-values are 0 in absorbing states by definition. We see that $Q_*(s_3, \downarrow) = 1.0$, since the agent will get a reward of 1.0 on the next step if it moves down from s_3; however, $Q_*(s,a) = 0$ for all other state-action pairs, since they do not provide nonzero immediate reward. This optimal Q-function reflects the fact that using $\gamma = 0$ is completely myopic, and ignores the future.

Figure 2.1(c) shows Q_* when $\gamma = 1$. In this case, we care about all future rewards equally. Thus $Q_*(s,a) = 1$ for all state-action pairs, since the agent can always reach the goal eventually. This is infinitely far-sighted. However, it does not give the agent any short-term guidance on how to behave. For example, in s_2, it is not clear if it is should go up or down, since both actions will eventually reach the goal with identical Q_*-values.

Figure 2.1(d) shows Q_* when $\gamma = 0.9$. This reflects a preference for near-term rewards, while also taking future reward into account. This encourages the agent to seek the shortest path to the goal, which is usually

Q*(s, a)

R(s)

s_{T1}	s_{T1}	0
s_1	s_1	0
s_2	s_2	0
s_3	s_3	0
s_{T2}	s_{T2}	1

Up a1

Down a2

γ = 0

Up	Down
0	0
0	0
0	1.0

γ = 1

Up	Down
0	1.0
1.0	1.0
1.0	1.0

γ = 0.9

Up	Down
0	0.81
0.73	0.9
0.81	1.0

(a)　　　(b)　　　(c)　　　(d)
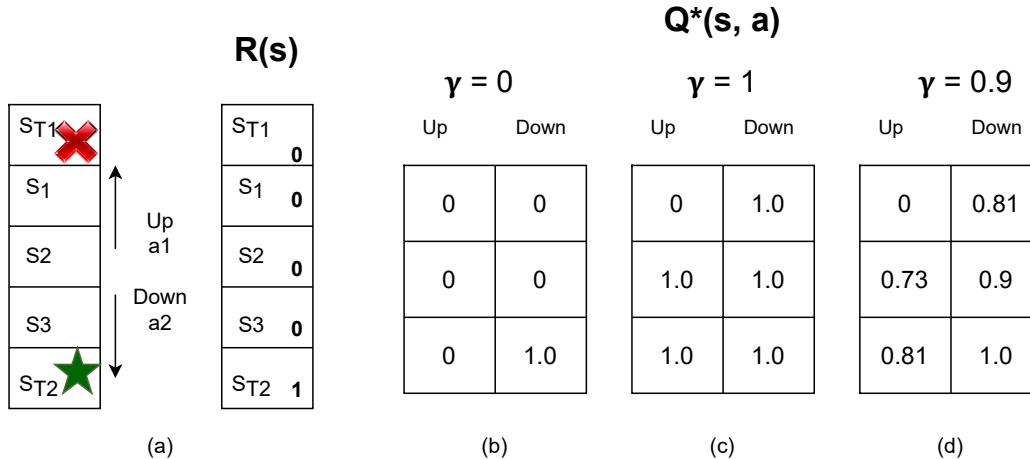
Figure 2.1: Left: illustration of a simple MDP corresponding to a 1d grid world of 3 non-absorbing states and 2 actions. Right: optimal Q-functions for different values of γ. Adapted from Figures 3.1, 3.2, 3.4 of [GK19].

what we desire. A proper choice of γ is up to the agent designer, just like the design of the reward function, and has to reflect the desired behavior of the agent.

2.2 Computing the value function and policy given a known world model

In this section, we discuss how to compute the optimal value function (the **prediction problem**) and the optimal policy (the **control problem**) when the MDP model is known. (Sometimes the term **planning** is used to refer to computing the optimal policy, given a known model, but planning can also refer to computing a sequence of actions, rather than a policy.) The algorithms we discuss are based on **dynamic programming** (DP) and **linear programming** (LP).

For simplicity, in this section, we assume discrete state and action sets with $\gamma < 1$. However, exact calculation of optimal policies often depends polynomially on the sizes of $\mathcal{S}$ and $\mathcal{A}$, and is intractable, for example, when the state space is a Cartesian product of several finite sets. This challenge is known as the **curse of dimensionality**. Therefore, approximations are typically needed, such as using parametric or nonparametric representations of the value function or policy, both for computational tractability and for extending the methods to handle MDPs with general state and action sets. This requires the use of **approximate dynamic programming** (ADP) and **approximate linear programming** (ALP) algorithms (see e.g., [Ber19]).

2.2.1 Value iteration

A popular and effective DP method for solving an MDP is **value iteration** (VI). Starting from an initial value function estimate V_0, the algorithm iteratively updates the estimate by

$$V_{k+1}(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} p(s'|s, a) V_k(s') \right] \tag{2.9}$$

Note that the update rule, sometimes called a **Bellman backup**, is exactly the right-hand side of the Bellman optimality equation Equation (2.4), with the unknown V_* replaced by the current estimate V_k. A

fundamental property of Equation (2.9) is that the update is a **contraction**: it can be verified that

$$\max_s |V_{k+1}(s) - V_*(s)| \le \gamma \max_s |V_k(s) - V_*(s)| \tag{2.10}$$

In other words, every iteration will reduce the maximum value function error by a constant factor.

V_k will converge to V_*, after which an optimal policy can be extracted using Equation (2.8). In practice, we can often terminate VI when V_k is close enough to V_*, since the resulting greedy policy wrt V_k will be near optimal. Value iteration can be adapted to learn the optimal action-value function Q_*.

2.2.2   Real-time dynamic programming (RTDP)

In value iteration, we compute $V_*(s)$ and $\pi_*(s)$ for all possible states s, averaging over all possible next states s' at each iteration, as illustrated in Figure 2.2(right). However, for some problems, we may only be interested in the value (and policy) for certain special starting states. This is the case, for example, in **shortest path problems** on graphs, where we are trying to find the shortest route from the current state to a goal state. This can be modeled as an episodic MDP by defining a transition matrix $p_S(s'|s, a)$ where taking edge a from node s leads to the neighboring node s' with probability 1. The reward function is defined as $R(s, a) = -1$ for all states s except the goal states, which are modeled as absorbing states.

In problems such as this, we can use a method known as **real-time dynamic programming** or **RTDP** [BBS95], to efficiently compute an **optimal partial policy**, which only specifies what to do for the reachable states. RTDP maintains a value function estimate V. At each step, it performs a Bellman backup for the current state s by $V(s) \leftarrow \max_a \mathbb{E}_{p_S(s'|s,a)} [R(s, a) + \gamma V(s')]$. It picks an action a (often with some exploration), reaches a next state s', and repeats the process. This can be seen as a form of the more general **asynchronous value iteration**, that focuses its computational effort on parts of the state space that are more likely to be reachable from the current state, rather than synchronously updating all states at each iteration.

2.2.3   Policy iteration

Another effective DP method for computing π_* is **policy iteration**. It is an iterative algorithm that searches in the space of deterministic policies until converging to an optimal policy. Each iteration consists of two steps, **policy evaluation** and **policy improvement**.

The policy evaluation step, as mentioned earlier, computes the value function for the current policy. Let π represent the current policy, $\boldsymbol{v}(s) = V_\pi(s)$ represent the value function encoded as a vector indexed by states, $\boldsymbol{r}(s) = \sum_a \pi(a|s)R(s, a)$ represent the reward vector, and $\mathbf{T}(s'|s) = \sum_a \pi(a|s)p(s'|s, a)$ represent the state transition matrix. Bellman's equation for policy evaluation can be written in the matrix-vector form as

$$\boldsymbol{v} = \boldsymbol{r} + \gamma \mathbf{T} \boldsymbol{v} \tag{2.11}$$

This is a linear system of equations in $|\mathcal{S}|$ unknowns. We can solve it using matrix inversion: $\boldsymbol{v} = (\mathbf{I} - \gamma \mathbf{T})^{-1}\boldsymbol{r}$. Alternatively, we can use value iteration by computing $\boldsymbol{v}_{t+1} = \boldsymbol{r} + \gamma \mathbf{T} \boldsymbol{v}_t$ until near convergence, or some form of asynchronous variant that is computationally more efficient.

Once we have evaluated V_π for the current policy π, we can use it to derive a better policy π', thus the name policy improvement. To do this, we simply compute a deterministic policy π' that acts greedily with respect to V_π in every state, using

$$\pi'(s) = \underset{a}{\operatorname{argmax}}\{R(s, a) + \gamma \mathbb{E}\left[V_\pi(s')\right]\} \tag{2.12}$$

We can guarantee that $V_{\pi'} \ge V_\pi$. This is called the **policy improvement theorem**. To see this, define $\boldsymbol{r}'$, $\mathbf{T}'$ and $\boldsymbol{v}'$ as before, but for the new policy π'. The definition of π' implies $\boldsymbol{r}' + \gamma \mathbf{T}' \boldsymbol{v} \ge \boldsymbol{r} + \gamma \mathbf{T} \boldsymbol{v} = \boldsymbol{v}$, where the equality is due to Bellman's equation. Repeating the same equality, we have

$$\boldsymbol{v} \le \boldsymbol{r}' + \gamma \mathbf{T}' \boldsymbol{v} \le \boldsymbol{r}' + \gamma \mathbf{T}'(\boldsymbol{r}' + \gamma \mathbf{T}' \boldsymbol{v}) \le \boldsymbol{r}' + \gamma \mathbf{T}'(\boldsymbol{r}' + \gamma \mathbf{T}'(\boldsymbol{r}' + \gamma \mathbf{T}' \boldsymbol{v})) \le \cdots \tag{2.13}$$

$$= (\mathbf{I} + \gamma \mathbf{T}' + \gamma^2 \mathbf{T}'^2 + \cdots)\boldsymbol{r}' = (\mathbf{I} - \gamma \mathbf{T}')^{-1}\boldsymbol{r}' = \boldsymbol{v}' \tag{2.14}$$
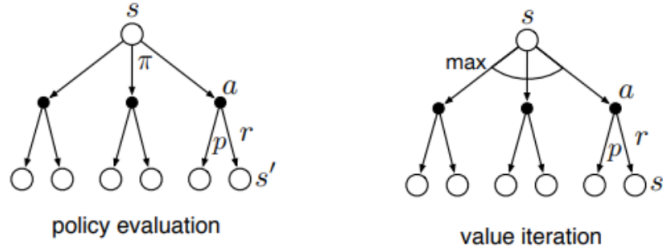
Figure 2.2: Policy iteration vs value iteration represented as backup diagrams. Empty circles represent states, solid (filled) circles represent states and actions. Adapted from Figure 8.6 of [SB18].

Starting from an initial policy π_0, policy iteration alternates between policy evaluation (E) and improvement (I) steps, as illustrated below:

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} V_* \tag{2.15}$$

The algorithm stops at iteration k, if the policy π_k is greedy with respect to its own value function V_{π_k}. In this case, the policy is optimal. Since there are at most $|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies, and every iteration strictly improves the policy, the algorithm must converge after finite iterations.

In PI, we alternate between policy evaluation (which involves multiple iterations, until convergence of V_π), and policy improvement. In VI, we alternate between one iteration of policy evaluation followed by one iteration of policy improvement (the "max" operator in the update rule). We are in fact free to intermix any number of these steps in any order. The process will converge once the policy is greedy wrt its own value function.

Note that policy evaluation computes V_π whereas value iteration computes V_*. This difference is illustrated in Figure 2.2, using a **backup diagram**. Here the root node represents any state s, nodes at the next level represent state-action combinations (solid circles), and nodes at the leaves representing the set of possible resulting next state s' for each possible action. In PE, we average over all actions according to the policy, whereas in VI, we take the maximum over all actions.

2.3 Computing the value function without knowing the world model

In the rest of this chapter, we assume the agent only has access to samples from the environment, $(s', r) \sim p(s', r|s, a)$. We will show how to use these samples to learn optimal value function and Q-function, even without knowing the MDP dynamics.

2.3.1 Monte Carlo estimation

Recall that $V_\pi(s) = \mathbb{E}[G_t|s_t = s]$ is the sum of expected (discounted) returns from state s if we follow policy π. A simple way to estimate this is to rollout the policy, and then compute the average sum of discounted rewards. The trajectory ends when we reach a terminal state, if the task is episodic, or when the discount factor γ^t becomes negligibly small, whichever occurs first. This is called **Monte Carlo estimation**. We can use this to update our estimate of the value function as follows:

$$V(s_t) \leftarrow V(s_t) + \eta[G_t - V(s_t)] \tag{2.16}$$

where η is the learning rate, and the term in brackets is an error term. We can use a similar technique to estimate $Q_\pi(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a]$ by simply starting the rollout with action a.

We can use MC estimation of Q, together with policy iteration (Section 2.2.3), to learn an optimal policy. Specifically, at iteration k, we compute a new, improved policy using $\pi_{k+1}(s) = \text{argmax}_a Q_k(s, a)$, where Q_k

is approximated using MC estimation. This update can be applied to all the states visited on the sampled trajectory. This overall technique is called **Monte Carlo control**.

To ensure this method converges to the optimal policy, we need to collect data for every (state, action) pair, at least in the tabular case, since there is no generalization across different values of $Q(s, a)$. One way to achieve this is to use an ϵ-greedy policy (see Section 1.4.1). Since this is an on-policy algorithm, the resulting method will converge to the optimal ϵ-soft policy, as opposed to the optimal policy. It is possible to use importance sampling to estimate the value function for the optimal policy, even if actions are chosen according to the ϵ-greedy policy. However, it is simpler to just gradually reduce ϵ.

2.3.2   Temporal difference (TD) learning

The Monte Carlo (MC) method in Section 2.3.1 results in an estimator for $V(s)$ with very high variance, since it has to unroll many trajectories, whose returns are a sum of many random rewards generated by stochastic state transitions. In addition, it is limited to episodic tasks (or finite horizon truncation of continuing tasks), since it must unroll to the end of the episode before each update step, to ensure it reliably estimates the long term return.

In this section, we discuss a more efficient technique called **temporal difference** or **TD** learning [Sut88]. The basic idea is to incrementally reduce the Bellman error for sampled states or state-actions, based on transitions instead of a long trajectory. More precisely, suppose we are to learn the value function V_π for a fixed policy π. Given a state transition (s_t, a_t, r_t, s_{t+1}), where $a_t \sim \pi(s_t)$, we change the estimate $V(s_t)$ so that it moves towards the **target value** $q_t = r_t + \gamma V(s_{t+1}) \approx G_{t:t+1}$:

$$V(s_t) \leftarrow V(s_t) + \eta \left[\underbrace{r_t + \gamma V(s_{t+1}) - V(s_t)}_{\delta_t} \right] \tag{2.17}$$

where η is the learning rate. (See [RFP15] for ways to adaptively set the learning rate.) The $\delta_t = y_t - V(s_t)$ term is known as the **TD error**. A more general form of TD update for parametric value function representations is

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left[r_t + \gamma V_{\boldsymbol{w}}(s_{t+1}) - V_{\boldsymbol{w}}(s_t) \right] \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t) \tag{2.18}$$

we see that Equation (2.16) is a special case. The TD update rule for evaluating Q_π is similar, except we replace states with states and actions.

It can be shown that TD learning in the tabular case, Equation (2.16), converges to the correct value function, under proper conditions [Ber19]. However, it may diverge when using nonlinear function approximators, as we discuss in Section 2.5.2.4. The reason is that this update is a "**semi-gradient**", which refers to the fact that we only take the gradient wrt the value function, $\nabla_{\boldsymbol{w}} V(\boldsymbol{s}_t, \boldsymbol{w}_t)$, treating the target U_t as constant.

The potential divergence of TD is also consistent with the fact that Equation (2.18) does not correspond to a gradient update on any objective function, despite having a very similar form to SGD (stochastic gradient descent). Instead, it is an example of **bootstrapping**, in which the estimate, $V_{\boldsymbol{w}}(s_t)$, is updated to approach a target, $r_t + \gamma V_{\boldsymbol{w}}(s_{t+1})$, which is defined by the value function estimate itself. This idea is shared by DP methods like value iteration, although they rely on the complete MDP model to compute an exact Bellman backup. In contrast, TD learning can be viewed as using sampled transitions to approximate such backups. An example of a non-bootstrapping approach is the Monte Carlo estimation in the previous section. It samples a complete trajectory, rather than individual transitions, to perform an update; this avoids the divergence issue, but is often much less efficient. Figure 2.3 illustrates the difference between MC, TD, and DP.

2.3.3   Combining TD and MC learning using TD(λ)

A key difference between TD and MC is the way they estimate returns. Given a trajectory $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, \ldots, s_T)$, TD estimates the return from state s_t by one-step lookahead, $G_{t:t+1} = r_t + \gamma V(s_{t+1})$, where the return from
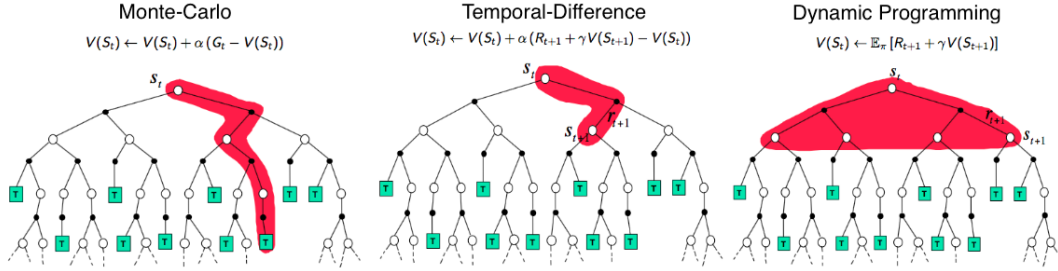
Figure 2.3: Backup diagrams of $V(s_t)$ for Monte Carlo, temporal difference, and dynamic programming updates of the state-value function. Used with kind permission of Andy Barto.

time $t + 1$ is replaced by its value function estimate. In contrast, MC waits until the end of the episode or until T is large enough, then uses the estimate $G_{t:T} = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t-1} r_{T-1}$. It is possible to interpolate between these by performing an n-step rollout, and then using the value function to approximate the return for the rest of the trajectory, similar to heuristic search (Section 4.1.2). That is, we can use the **n-step return**

$$G_{t:t+n} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) \tag{2.19}$$

For example, the 1-step and 2-step returns are given by

$$G_{t:t+1} = r_t + \gamma v_{t+1} \tag{2.20}$$
$$G_{t:t+1} = r_t + \gamma r_{t+1} + \gamma^2 v_{t+2} \tag{2.21}$$

The corresponding n-step version of the TD update becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left[G_{t:t+n} - V_{\boldsymbol{w}}(s_t) \right] \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t) \tag{2.22}$$

Rather than picking a specific lookahead value, n, we can take a weighted average of all possible values, with a single parameter $\lambda \in [0, 1]$, by using

$$G_t^\lambda \triangleq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n} \tag{2.23}$$

This is called the **lambda return**. Note that these coefficients sum to one (since $\sum_{t=0}^{\infty}(1-\lambda)\lambda^t = \frac{1-\lambda}{1-\lambda} = 1$, for $\lambda < 1$), so the return is a convex combination of n-step returns. See Figure 2.4 for an illustration. We can now use G_t^λ inside the TD update instead of $G_{t:t+n}$; this is called **TD(λ)**.

Note that, if a terminal state is entered at step T (as happens with episodic tasks), then all subsequent n-step returns are equal to the conventional return, G_t. Hence we can write

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{2.24}$$

From this we can see that if $\lambda = 1$, the λ-return becomes equal to the regular MC return G_t. If $\lambda = 0$, the λ-return becomes equal to the one-step return $G_{t:t+1}$ (since $0^{n-1} = 1$ iff $n = 1$), so standard TD learning is often called **TD(0) learning**. This episodic form also gives us the following recursive equation

$$G_t^\lambda = r_t + \gamma[(1 - \lambda)v_{t+1} + \lambda G_{t+1}^\lambda] \tag{2.25}$$

which we initialize with $G_T = v_t$.
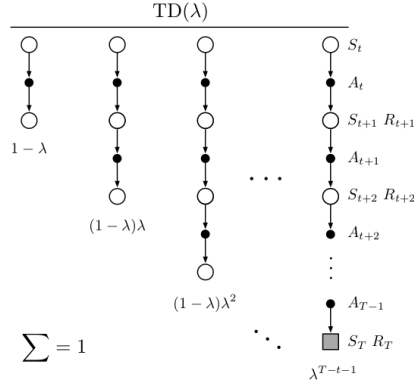
Figure 2.4: The backup diagram for TD(λ). Standard TD learning corresponds to $\lambda = 0$, and standard MC learning corresponds to $\lambda = 1$. From Figure 12.1 of [SB18]. Used with kind permission of Richard Sutton.

2.3.4 Eligibility traces

An important benefit of using the geometric weighting in Equation (2.23), as opposed to the n-step update, is that the corresponding TD learning update can be efficiently implemented through the use of **eligibility traces**, even though G_t^λ is a sum of infinitely many terms. The eligibility term is a weighted sum of the gradients of the value function:

$$z_t = \gamma \lambda z_{t-1} + \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t) \tag{2.26}$$

(This trace term gets reset to 0 at the start of each episode.) We replace the TD(0) update of $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \delta_t \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t)$ with the TD(λ) version to get

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \eta \delta_t \boldsymbol{z}_t \tag{2.27}$$

See [Sei+16] for more details.

2.4 SARSA: on-policy TD control

TD learning is for policy evaluation, as it estimates the value function for a fixed policy. In order to find an optimal policy, we may use the algorithm as a building block inside generalized policy iteration (Section 2.2.3). In this case, it is more convenient to work with the action-value function, Q, and a policy π that is greedy with respect to Q. The agent follows π in every step to choose actions, and upon a transition (s, a, r, s') the TD update rule is

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + \gamma Q(s', a') - Q(s, a) \right] \tag{2.28}$$

where $a' \sim \pi(s')$ is the action the agent will take in state s'. After Q is updated (for policy evaluation), π also changes accordingly as it is greedy with respect to Q (for policy improvement). This algorithm, first proposed by [RN94], was further studied and renamed to **SARSA** by [Sut96]; the name comes from its update rule that involves an augmented transition (s, a, r, s', a').

In order for SARSA to converge to Q_*, every state-action pair must be visited infinitely often, at least in the tabular case, since the algorithm only updates $Q(s, a)$ for (s, a) that it visits. One way to ensure this condition is to use a "greedy in the limit with infinite exploration" (**GLIE**) policy. An example is the ϵ-greedy policy, with ϵ vanishing to 0 gradually. It can be shown that SARSA with a GLIE policy will converge to Q_* and π_* [Sin+00].

2.5 Q-learning: off-policy TD control

SARSA is an on-policy algorithm, which means it learns the Q-function for the policy it is currently using, which is typically not the optimal policy, because of the need to perform exploration. However, with a simple modification, we can convert this to an off-policy algorithm that learns Q_*, even if a suboptimal or exploratory policy is used to choose actions.

2.5.1 Tabular Q learning

Suppose we modify SARSA by replacing the sampled next action $a' \sim \pi(s')$ in Equation (2.28) with a greedy action: $a' = \text{argmax}_b Q(s', b)$. This results in the following update when a transition (s, a, r, s') happens

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \qquad (2.29)$$

This is the update rule of **Q-learning** for the tabular case [WD92].

Since it is off-policy, the method can use (s, a, r, s') triples coming from any data source, such as older versions of the policy, or log data from an existing (non-RL) system. If every state-action pair is visited infinitely often, the algorithm provably converges to Q_* in the tabular case, with properly decayed learning rates [Ber19]. Algorithm 1 gives a vanilla implementation of Q-learning with ϵ-greedy exploration.

Algorithm 1: Tabular Q-learning with ϵ-greedy exploration

---

1 Initialize value function Q
2 **repeat**
3   Sample starting state s of new episode
4   **repeat**
5   Sample action $a = \begin{cases} \text{argmax}_b Q(s, b), & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{with probability } \epsilon \end{cases}$
6         $(s', r) = \text{env.step}(a)$
7   Compute the TD error: $\delta = r + \gamma \max_{a'} Q(s', a') - Q(s, a)$
8         $Q(s, a) \leftarrow Q(s, a) + \eta \delta$
9         $s \leftarrow s'$
10   **until** *state s is terminal*
11 **until** *converged*

For terminal states, $s \in \mathcal{S}^+$, we know that $Q(s, a) = 0$ for all actions a. Consequently, for the optimal value function, we have $V^*(s) = \max_{a'} Q^*(s, a) = 0$ for all terminal states. When performing online learning, we don't usually know which states are terminal. Therefore we assume that, whenever we take a step in the environment, we get the next state s' and reward r, but also a binary indicator done(s') that tells us if s' is terminal. In this case, we set the target value in Q-learning to $V^*(s') = 0$ yielding the modified update rule:

$$Q(s, a) \leftarrow Q(s, a) + \eta \left[r + (1 - \text{done}(s'))\gamma \max_{a'} Q(s', a') - Q(s, a) \right] \qquad (2.30)$$

For brevity, we will usually ignore this factor in the subsequent equations, but it needs to be implemented in the code.

Figure 2.5 gives an example of Q-learning applied to the simple 1d grid world from Figure 2.1, using $\gamma = 0.9$. We show the Q-functon at the start and end of each episode, after performing actions chosen by an ϵ-greedy policy. We initialize $Q(s, a) = 0$ for all entries, and use a step size of $\eta = 1$. At convergence, we have $Q_*(s, a) = r + \gamma Q_*(s', a_*)$, where $a_* = \downarrow$ for all states.

Q-function episode start			Episode	Time Step	Action	(s,α,r , s')	r + γ Q*(s' , α)	Q-function episode end		
	UP	DOWN							UP	DOWN
Q₁ S₁	0	0	1	1	↓	(S₁, D,0,S₂)	0 + 0.9 X 0 = 0	S₁	0	0
			1	2	↑	(S₂,U,0,S₁)	0 + 0.9 X 0 = 0			
S₂	0	0	1	3	↓	(S₁, D,0,S₂)	0 + 0.9 X 0 = 0	S₂	0	0
			1	4	↓	(S₂, U,0,S₁)	0 + 0.9 X 0 = 0			
S₃	0	0	1	5	↓	(S₃, D,1,S_T2)	1	S₃	0	1
Q₂ S₁	0	0						S₁	0	0
			2	1	↓	(S₁ , D,0,S₂)	0 + 0.9 x 0 = 0			
S₂	0	0	2	2	↓	(S₂ , D,0,S₃)	0 + 0.9 x 1 = 0.9	S₂	0	0.9
			2	3	↓	(S₃ , D,0,S_T2)	1			
S₃	0	1						S₃	0	1
Q₃ S₁	0	0	3	1	↓	(S₁ , D,0,S₂)	0 + 0.9 x 0.9 = 0.81	S₁	0	0.81
			3	2	↓	(S₂ , D,0,S₃)	0 + 0.9 x 1 = 0.9			
S₂	0	0.9	3	3	↑	(S₃ , D,0,S₂)	0 + 0.9 x 0.9 = 0.81	S₂	0	0.9
			3	4	↓	(S₂ , D,0,S₃)	0 + 0.9 x 1 = 0.9			
S₃	0	1	3	5	↓	(S₃ , D,0,S_T2)	1	S₃	0.81	1
Q₄ S₁	0	0.81	4	1	↓	(S₁ , D,0,S₂)	0 + 0.9 x 0.9 = 0.81	S₁	0	0.81
			4	2	↑	(S₂ , U,0,S₁)	0 + 0.9 x 0.81 = 0.73			
S₂	0	0.9	4	3	↓	(S₁ , D,0,S₂)	0 + 0.9 x 0.9 = 0.81	S₂	0.73	0.9
			4	4	↑	(S₂ , U,0,S₃)	0 + 0.9 x 0.81 = 0.73			
S₃	0.81	1	4	5	↓	(S₁ , D,0,S₃)	0 + 0.9 x 0.9 = 0.81	S₃	0.81	1
			4	6	↓	(S₂ , D,0,S₃)	0 + 0.9 x 1 = 0.9			
			4	7	↓	(S₂ , D,0,S₃)	1			
Q₅ S₁	0	0.81						S₁	0	0.81
			5	1	↑	(S1 , U, 0,S_T1)	0			
S₂	0.73	0.9						S₂	0.73	0.9
S₃	0.81	1						S₃	0.81	1

Figure 2.5: *Illustration of Q learning for one random trajectory in the 1d grid world in Figure 2.1 using ε-greedy exploration. At the end of episode 1, we make a transition from S_3 to S_{T2} and get a reward of $r = 1$, so we estimate $Q(S_3, \downarrow) = 1$. In episode 2, we make a transition from S_2 to S_3, so S_2 gets incremented by $\gamma Q(S_3, \downarrow) = 0.9$. Adapted from Figure 3.3 of [GK19].*

2.5.2   Q learning with function approximation

To make Q learning work with high-dimensional state spaces, we have to replace the tabular (non-parametric) representation with a parametric approximation, denoted $Q_{\boldsymbol{w}}(s, a)$. We can update this function using one or more steps of SGD on the following loss function

$$\mathcal{L}(\boldsymbol{w}|\boldsymbol{s}, a, r, \boldsymbol{s}') = \left((r + \gamma \max_{a'} Q_{\boldsymbol{w}}(\boldsymbol{s}', a')) - Q_{\boldsymbol{w}}(\boldsymbol{s}, a)\right)^2 \tag{2.31}$$

Since nonlinear functions need to be trained on minibatches of data, we compute the average loss over multiple randomly sampled experience tuples (see Section 2.5.2.3 for discussion) to get

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{s}, a, r, \boldsymbol{s}') \sim U(\mathcal{D})} \left[\mathcal{L}(\boldsymbol{w}|\boldsymbol{s}, a, r, \boldsymbol{s}')\right] \tag{2.32}$$

See Algorithm 2 for the pseudocode.

Algorithm 2: Q learning with function approximation and replay buffers

1 Initialize environment state $\boldsymbol{s}$, network parameters $\boldsymbol{w}_0$, replay buffer $\mathcal{D} = \emptyset$, discount factor γ, step size η, policy $\pi_0(a|s) = \epsilon\text{Unif}(a) + (1 - \epsilon)\delta(a = \text{argmax}_a Q_{\boldsymbol{w}_0}(s, a))$
2 **for** *iteration* $k = 0, 1, 2, \ldots$ **do**
3   **for** *environment step* $s = 0, 1, \ldots, S - 1$ **do**
4   Sample action: $a \sim \pi_k(a|s)$
5   Interact with environment: $(s', r) = \text{env.step}(a)$
6   Update buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, s', r)\}$
7     $\boldsymbol{w}_{k,0} \leftarrow \boldsymbol{w}_k$
8   **for** *gradient step* $g = 0, 1, \ldots, G - 1$ **do**
9   Sample batch: $B \subset \mathcal{D}$
10   Compute error: $\mathcal{L}(B, \boldsymbol{w}_{k,g}) = \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left[Q_{\boldsymbol{w}_{k,g}}(s, a) - (r + \gamma \max_{a'} Q_{\boldsymbol{w}_k}(s', a'))\right]^2$
11   Update parameters: $\boldsymbol{w}_{k,g} \leftarrow \boldsymbol{w}_{k,g} - \eta \nabla_{\boldsymbol{w}_{k,g}} \mathcal{L}(B, \boldsymbol{w}_{k,g})$
12     $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_{k,G}$

2.5.2.1   Neural fitted Q

The first approach of this kind is known as **neural fitted Q iteration** [Rie05], which corresponds to fully optimizing $\mathcal{L}(\boldsymbol{w})$ at each iteration (equivalent to using $G = \infty$ gradient steps).

2.5.2.2   DQN

The influential deep Q-network or DQN paper of [Mni+15] also used neural nets to represent the Q function, but performed a smaller number of gradient updates per iteration. Furthermore, they proposed to modify the target value when fitting the Q function in order to avoid instabilities during training (see Section 2.5.2.4 for details).

The DQN method became famous since it was able to train agents that can outperform humans when playing various Atari games from the **ALE** (Atari Learning Environment) benchmark [Bel+13]. Here the input is a small color image, and the action space corresponds to moving left, right, up or down, plus an optional shoot action.[1]

Since 2015, many more extensions to DQN have been proposed, with the goal of improving performance in various ways, either in terms of peak reward obtained, or sample efficiency (e.g., reward obtained after only

---

[1]For more discussion of ALE, see [Mac+18a], and for a recent extension to continuous actions (representing joystick control), see the CALE benchmark of [FC24]. Note that DQN was not the first deep RL method to train an agent from pixel input; that honor goes to [LR10], who trained an autoencoder to embed images into low-dimensional latents, and then used neural fitted Q learning (Section 2.5.2.1) to fit the Q function.
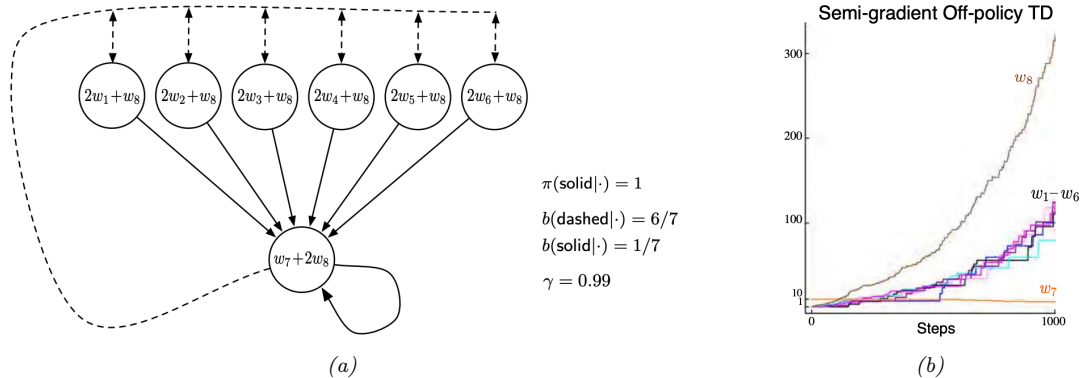
Figure 2.6: (a) A simple MDP. (b) Parameters of the policy diverge over time. From Figures 11.1 and 11.2 of [SB18]. Used with kind permission of Richard Sutton.

100k steps in the environment, as proposed in the **Atari-100k** benchmark [Kai+19]), or training stability, or all of the above. We discuss some of these extensions in Section 2.5.4.

2.5.2.3 Experience replay

Since Q learning is an off-policy method, we can update the Q function using any data source. This is particularly important when we use nonlinear function approximation (see Section 2.5.2), which often needs a lot of data for model fitting. A natural source of data is data collected earlier in the trajectori of the agent; this is called an **experience replay** buffer, which stores (s, a, r, s') transition tuples into a buffer. This can improve the stability and sample efficiency of learning, and was originally proposed in [Lin92].

This modification has two advantages. First, it improves data efficiency as every transition can be used multiple times. Second, it improves stability in training, by reducing the correlation of the data samples that the network is trained on, since the training tuples do not have to come from adjacent moments in time. (Note that experience replay requires the use of off-policy learning methods, such as Q learning, since the training data is sampled from older versions of the policy, not the current policy.)

It is possible to replace the uniform sampling from the buffer with one that favors more important transition tuples that may be more informative about Q. This idea is formalized in [Sch+16a], who develop a technique known as **prioritized experience replay**.

2.5.2.4 The deadly triad

The problem with the naive Q learning objective in Equation (2.31) is that it can lead to instability, since the target we are regressing towards uses the same parameters w as the function we are updating. So the network is "chasing its own tail". Although this is fine for tabular models, it can fail for nonlinear models, as we discuss below.

In general, an RL algorithm can become unstable when it has these three components: function approximation (such as neural networks), bootstrapped value function estimation (i.e., using TD-like methods instead of MC), and off-policy learning (where the actions are sampled from some distribution other than the policy that is being optimized). This combination is known as **the deadly triad** [Sut15; van+18]).

A classic example of this is the simple MDP depicted in Figure 2.6a, due to [Bai95]. (This is known as **Baird's counter example**.) It has 7 states and 2 actions. Taking the dashed action takes the environment to the 6 upper states uniformly at random, while the solid action takes it to the bottom state. The reward is 0 in all transitions, and $\gamma = 0.99$. The value function V_w uses a linear parameterization indicated by the expressions shown inside the states, with $w \in \mathbb{R}^8$. The target policies π always chooses the solid action in every state. Clearly, the true value function, $V_\pi(s) = 0$, can be exactly represented by setting $w = 0$.

Suppose we use a behavior policy b to generate a trajectory, which chooses the dashed and solid actions with probabilities 6/7 and 1/7, respectively, in every state. If we apply TD(0) on this trajectory, the parameters diverge to ∞ (Figure 2.6b), even though the problem appears simple. In contrast, with on-policy data (that is, when b is the same as π), TD(0) with linear approximation can be guaranteed to converge to a good value function approximate [TR97]. The difference is that with on-policy learning, as we improve the value function, we also improve the policy, so the two become self-consistent, whereas with off-policy learning, the behavior policy may not match the optimal value function that is being learned, leading to inconsistencies.

The divergence behavior is demonstrated in many value-based bootstrapping methods, including TD, Q-learning, and related approximate dynamic programming algorithms, where the value function is represented either linearly (like the example above) or nonlinearly [Gor95; TVR97; OCD21]. The root cause of these divergence phenomena is that bootstrapping methods typically are not minimizing a fixed objective function. Rather, they create a learning target using their own estimates, thus potentially creating a self-reinforcing loop to push the estimates to infinity. More formally, the problem is that the contraction property in the tabular case (Equation (2.10)) may no longer hold when V is approximated by $V_{\boldsymbol{w}}$.

We discuss some solutions to the deadly triad problem below.

2.5.2.5 Target networks

One heuristic solution to the deadly triad, proposed in the DQN paper, is to use a "frozen" **target network** computed at an earlier iteration to define the target value for the DQN updates, rather than trying to chase a constantly moving target. Specifically, we maintain an extra copy the Q-network, $Q_{\boldsymbol{w}^-}$, with the same structure as $Q_{\boldsymbol{w}}$. This new Q-network is used to compute bootstrapping targets

$$q(r, \boldsymbol{s}'; \boldsymbol{w}^-) = r + \gamma \max_{a'} Q_{\boldsymbol{w}^-}(\boldsymbol{s}', a') \qquad (2.33)$$

for training $Q_{\boldsymbol{w}}$. We can periodically set $\boldsymbol{w}^- \leftarrow \text{sg}(\boldsymbol{w})$, usually after a few episodes, where the stop gradient operator is used to prevent autodiff propagating gradients back to $\boldsymbol{w}$. Alternatively, we can use an exponential moving average (EMA) of the weights, i.e., we use $\overline{\boldsymbol{w}} = \rho\overline{\boldsymbol{w}} + (1 - \rho)\text{sg}(\boldsymbol{w})$, where $\rho \ll 1$ ensures that $Q_{\overline{\boldsymbol{w}}}$ slowly catches up with $Q_{\boldsymbol{w}}$. (If $\rho = 0$, we say that this is a **detached target**, since it is just a frozen copy of the current weights.) The final loss has the form

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{s}, a, r, \boldsymbol{s}') \sim U(\mathcal{D})} \left[\mathcal{L}(\boldsymbol{w} | \boldsymbol{s}, a, r, \boldsymbol{s}') \right] \qquad (2.34)$$

$$\mathcal{L}(\boldsymbol{w} | \boldsymbol{s}, a, r, \boldsymbol{s}') = (q(r, \boldsymbol{s}'; \overline{\boldsymbol{w}}) - Q_{\boldsymbol{w}}(\boldsymbol{s}, a))^2 \qquad (2.35)$$

Theoretical work justifying this technique is given in [FSW23; Che+24a].

2.5.2.6 Two time-scale methods

A general way to ensure convergence in off-policy learning is to construct an objective function, the minimization of which leads to a good value function approximation. This is the basis of the **gradient TD method** of [SSM08; Mae+09; Ghi+20]. In practice, this can be achieved by updating the target value in the TD update more quickly than the value function itself; this is known as a **two timescale optimization** (see e.g., [Yu17; Zha+19; Hon+23]). It is also possible to use a standard single timescale method provided the target value is computed using a frozen target network, as discussed in Section 2.5.2.5. See [FSW23; Che+24a] for details.

2.5.2.7 Layer norm

More recently, [Gal+24] proved that just adding LayerNorm [BKH16] to the penultimate layer of the critic network, just before the linear head, is sufficient to provably yield convergence of TD learning even in the off-policy setting. In particular, suppose the network has the form $Q(s, a | \boldsymbol{w}, \boldsymbol{\theta}) = \boldsymbol{w}^T \text{ReLU}(\text{LayerNorm}(f(s, a; \boldsymbol{\theta})))$. Since $||\text{LayerNorm}(f(s, a; \boldsymbol{\theta}))|| \leq 1$, we have $||Q(s, a | \boldsymbol{w}, \boldsymbol{\theta}) \leq ||\boldsymbol{w}||$, which means the magnitude of the output is always bounded, as shown in Figure 2.7. In [Gal+24], they prove this (plus ℓ_2 regularization on $\boldsymbol{w}$, and a sufficiently wide penultimate layer) is sufficient to ensure convergence of the value function estimate.
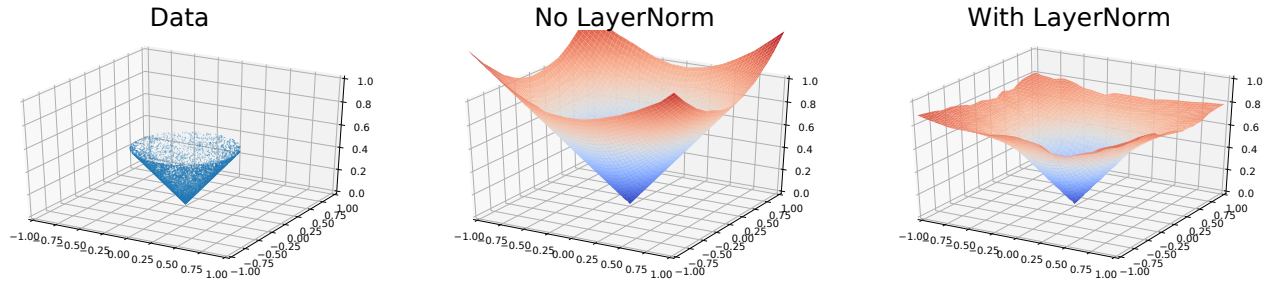
Figure 2.7: We generate a dataset (left) with inputs $\boldsymbol{x}$ distributed in a circle with radius 0.5 and labels $y = ||\boldsymbol{x}||$. We then fit a two-layer MLP without LayerNorm (center) and with LayerNorm (right). LayerNorm bounds the values and prevents catastrophic overestimation when extrapolating. From Figure 3 of [Bal+23]. Used with kind permission of Philip Ball.
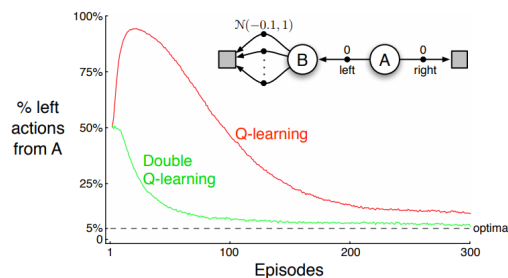
Figure 2.8: Comparison of Q-learning and double Q-learning on a simple episodic MDP using ϵ-greedy action selection with $\epsilon = 0.1$. The initial state is A, and squares denote absorbing states. The data are averaged over 10,000 runs. From Figure 6.5 of [SB18]. Used with kind permission of Richard Sutton.

2.5.3 Maximization bias

Standard Q-learning suffers from a problem known as the **optimizer's curse** [SW06], or the **maximization bias**. The problem refers to the simple statistical inequality: $\mathbb{E}[\max_a X_a] \geq \max_a \mathbb{E}[X_a]$, for a set of random variables $\{X_a\}$. Thus, if we pick actions greedily according to their random scores $\{X_a\}$, we might pick a wrong action just because random noise makes it appealing.

Figure 2.8 gives a simple example of how this can happen in an MDP. The start state is A. The right action gives a reward 0 and terminates the episode. The left action also gives a reward of 0, but then enters state B, from which there are many possible actions, with rewards drawn from $\mathcal{N}(-0.1, 1.0)$. Thus the expected return for any trajectory starting with the left action is -0.1, making it suboptimal. Nevertheless, the RL algorithm may pick the left action due to the maximization bias making B appear to have a positive value.

2.5.3.1 Double Q-learning

One solution to avoid the maximization bias is to use two separate Q-functions, Q_1 and Q_2, one for selecting the greedy action, and the other for estimating the corresponding Q-value. In particular, upon seeing a transition (s, a, r, s'), we perform the following update for $i = 1 : 2$:

$$Q_i(s, a) \leftarrow Q_i(s, a) + \eta(q_i(s, a) - Q_i(s, a)) \tag{2.36}$$

$$q_i(s, a) = r + \gamma Q_i(s', \underset{a'}{\operatorname{argmax}} Q_{-i}(s', a')) \tag{2.37}$$

So we see that Q_1 uses Q_2 to choose the best action but uses Q_1 to evaluate it, and vice versa. This technique is called **double Q-learning** [Has10]. Figure 2.8 shows the benefits of the algorithm over standard Q-learning

in a toy problem.

2.5.3.2 Double DQN

In [HGS16], they combine double Q learning with deep Q networks (Section 2.5.2.2) to get **double DQN**. This modifies Equation (2.37) to its gradient form, and then the current network for action proposals, but the target network for action evaluation. Thus the training target becomes

$$q(r, \boldsymbol{s}'; \boldsymbol{w}, \overline{\boldsymbol{w}}) = r + \gamma Q_{\overline{\boldsymbol{w}}}(\boldsymbol{s}', \underset{a'}{\operatorname{argmax}} \, Q_{\boldsymbol{w}}(\boldsymbol{s}', a')) \tag{2.38}$$

In Section 3.6.2 we discuss an extension called **clipped double DQN** which uses two Q networks and their frozen copies to define the following target:

$$q(r, \boldsymbol{s}'; \boldsymbol{w}_{1:2}, \overline{\boldsymbol{w}}_{1:2}) = r + \gamma \min_{i=1,2} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}', \underset{a'}{\operatorname{argmax}} \, Q_{\boldsymbol{w}_i}(\boldsymbol{s}', a')) \tag{2.39}$$

where $Q_{\overline{\boldsymbol{w}}_i}$ is the target network for $Q_{\boldsymbol{w}_i}$.

2.5.3.3 Randomized ensemble DQN

The double DQN method is extended in the **REDQ** (randomized ensembled double Q learning) method of [Che+20], which uses an ensemble of $N > 2$ Q-networks. Furthermore, at each step, it draws a random sample of $M \le N$ networks, and takes the minimum over them when computing the target value. That is, it uses the following update (see Algorithm 2 in appendix of [Che+20]):

$$q(r, \boldsymbol{s}'; \boldsymbol{w}_{1:N}, \overline{\boldsymbol{w}}_{1:N}) = r + \gamma \max_{a'} \min_{i \in \mathcal{M}} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}', a') \tag{2.40}$$

where $\mathcal{M}$ is a random subset from the N value functions. The ensemble reduces the variance, and the minimum reduces the overestimation bias.[2] If we set $N = M = 2$, we get a method similar to clipped double Q learning. (Note that REDQ is very similiar to the **Random Ensemble Mixture** method of [ASN20], which was designed for offline RL.)

2.5.4 DQN extensions

In this section, we discuss various extensions of DQN.

2.5.4.1 Q learning for continuous actions

Q learning is not directly applicable to continuous actions due to the need to compute the argmax over actions. An early solution to this problem, based on neural fitted Q learning (see Section 2.5.2.1), is proposed in [HR11]. This became the basis of the DDPG algorithm of Section 3.6.1, which learns a policy to predict the argmax.

An alternative approach is to use gradient-free optimizers such as the cross-entropy method to approximate the argmax. The **QT-Opt** method of [Kal+18] treats the action vector $\boldsymbol{a}$ as a sequence of actions, and optimizes one dimension at a time [Met+17]. The **CAQL** (continuous action Q-learning) method of [Ryu+20]) uses mixed integer programming to solve the argmax problem, leveraging the ReLU structure of the Q-network. The method of [Sey+22] quantizes each action dimension separately, and then solves the argmax problem using methods inspired by multi-agent RL.

---

[2]In addition, REDQ performs $G \gg 1$ updates of the value functions for each environment step; this high **Update-To-Data** (UTD) ratio (also called **Replay Ratio**) is critical for sample efficiency, and is commonly used in model-based RL.

2.5.4.2 Dueling DQN

The **dueling DQN** method of [Wan+16], learns a value function and an advantage function, and derives the Q function, rather than learning it directly. This is helpful when there are many actions with similar Q-values, since the advantage $A(s, a) = Q(s, a) - V(s)$ focuses on the differences in value relative to a shared baseline.

In more detail, we define a network with $|A| + 1$ output heads, which computes $A_{\boldsymbol{w}}(\boldsymbol{s}, a)$ for $a = 1 : A$ and $V_{\boldsymbol{w}}(\boldsymbol{s})$. We can then derive

$$Q_{\boldsymbol{w}}(\boldsymbol{s}, a) = V_{\boldsymbol{w}}(\boldsymbol{s}) + A_{\boldsymbol{w}}(\boldsymbol{s}, a) \tag{2.41}$$

However, this naive approach ignores the following constraint that holds for any policy π:

$$\mathbb{E}_{\pi(a|s)} \left[A^{\pi}(s, a) \right] = \mathbb{E}_{\pi(a|s)} \left[Q^{\pi}(s, a) - V^{\pi}(s) \right] \tag{2.42}$$
$$= V^{\pi}(s) - V^{\pi}(s) = 0 \tag{2.43}$$

Fortunately, for the optimal policy $\pi^*(s) = \operatorname{argmax}_{a'} Q^*(s, a')$ we have

$$0 = \mathbb{E}_{\pi^*(a|s)} \left[Q^*(s, a) \right] - V^*(s) \tag{2.44}$$
$$= Q^*(s, \operatorname*{argmax}_{a'} Q^*(s, a')) - V^*(s) \tag{2.45}$$
$$= \max_{a'} Q^*(s, a') - V^*(s) \tag{2.46}$$
$$= \max_{a'} A^*(s, a') \tag{2.47}$$

Thus we can satisfy the constraint for the optimal policy by subtracting off $\max_a A(s, a)$ from the advantage head. Equivalently we can compute the Q function using

$$Q_{\boldsymbol{w}}(\boldsymbol{s}, a) = V_{\boldsymbol{w}}(\boldsymbol{s}) + A_{\boldsymbol{w}}(\boldsymbol{s}, a) - \max_{a'} A_{\boldsymbol{w}}(\boldsymbol{s}, a') \tag{2.48}$$

In practice, the max is replaced by an average, which seems to work better empirically.

2.5.4.3 Noisy nets and exploration

Standard DQN relies on the epsilon-greedy strategy to perform exploration. However, this will explore equally in all states, whereas we would like to the amount of exploration to be state dependent, to reflect the amount of uncertainty in the outcomes of trying each action in that state due to lack of knowledge (i.e., **epistemic uncertainty** rather than aleatoric or irreducile uncertainty). An early approach to this, known as **noisy nets** [For+18], added random noise to the network weights to encourage exploration which is temporally consistent within episodes. More recent methods for exploration are discussed in Section 1.4.

2.5.4.4 Multi-step DQN

As we discussed in Section 2.3.3, we can reduce the bias introduced by bootstrapping by replacing TD(1) updates with TD(n) updates, where we unroll the value computation for n MC steps, and then plug in the value function at the end. We can apply this to the DQN context by defining the target

$$q(s_0, a_0) = \sum_{t=1}^{n} \gamma^{t-1} r_t + \gamma^n \max_{a_n} Q_{\boldsymbol{w}}(s_n, a_n) \tag{2.49}$$

This can be implemented for episodic environments by storing experience tuples of the form

$$\tau = (s, a, \sum_{k=1}^{n} \gamma^{k-1} r_k, s_n, \text{done}) \tag{2.50}$$

where done $= 1$ if the trajectory ended at any point during the n-step rollout.
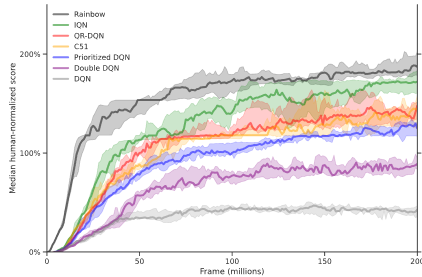
Figure 2.9: Plot of median human-normalized score over all 57 Atari games for various DQN agents. The yellow, red and green curves are distributional RL methods (Section 5.1), namely categorical DQN (C51) (Section 5.1.2) Quantile Regression DQN (Section 5.1.1), and Implicit Quantile Networks [Dab+18]. Figure from https://github.com/google-deepmind/dqn_zoo.

Theoretically this method is only valid if all the intermediate actions, $a_{2:n-1}$, are sampled from the current optimal policy derived from $Q_{\boldsymbol{w}}$, as opposed to some behavior policy, such as epsilon greedy or some samples from the replay buffer from an old policy. In practice, we can just restrict sampling to recent samples from the replay buffer, making the resulting method approximately on-policy.

Instead of using a fixed n, it is possible to use a weighted combination of returns; this is known as the $Q(\lambda)$ algorithm [PW94; Koz+21].

2.5.4.5 Rainbow

The **Rainbow** method of [Hes+18] combined 6 improvements to the vanilla DQN method, as listed below. (The paper is called "Rainbow" due to the color coding of their results plot, a modified version of which is shown in Figure 2.9.) At the time it was published (2018), this produced SOTA results on the Atari-200M benchmark. The 6 improvements are as follows:

- Use double DQN, as in Section 2.5.3.2.

- Use prioritized experience replay, as in Section 2.5.2.3.

- Use the categorical DQN (C51) (Section 5.1.2) distributional RL method.

- Use n-step returns (with $n = 3$), as in Section 2.5.4.4.

- Use dueling DQN, as in Section 2.5.4.2.

- Use noisy nets, as in Section 2.5.4.3.

Each improvement gives diminishing returns, as can be see in Figure 2.9.

Recently the "Beyond the Rainbow" paper [Unk24] proposed several more extensions:

- Use a larger CNN with residual connections, namely the Impala network from [Esp+18] with the modifications (including the use of spectral normalization) proposed in [SS21].

- Replace C51 with Implicit Quantile Networks [Dab+18].

- Use **Munchausen RL** [VPG20], which modifies the Q learning update rule by adding an entropy-like penalty.

- Collect 1 environment step from 64 parallel workers for each minibatch update (rather than taking many steps from a smaller number of workers).

2.5.4.6 Bigger, Better, Faster

At the time of writing this document (2024), the SOTA on the 100k sample-efficient Atari benchmark [Kai+19] is obtained by the **BBF** algorithm of [Sch+23b]. (BBF stands for "Bigger, Better, Faster".) It uses the following tricks, in order of decreasing importance:

- Use a larger CNN with residual connections, namely a modified version of the Impala network from [Esp+18].

- Increase the **update-to-data** (UTD) ratio (number of times we update the Q function for every observation that is observed), in order to increase sample efficiency [HHA19].

- Use a periodic soft reset of (some of) the network weights to avoid loss of elasticity due to increased network updates, following the **SR-SPR** method of [D'O+22].

- Use n-step returns, as in Section 2.5.4.4, and then gradually decrease (anneal) the n-step return from $n = 10$ to $n = 3$, to reduce the bias over time.

- Add weight decay.

- Add a self-predictive representation loss (Section 4.3.2.2) to increase sample efficiency.

- Gradually increase the discount factor from $\gamma = 0.97$ to $\gamma = 0.997$, to encourage longer term planning once the model starts to be trained.[3]

- Drop noisy nets (which requires multiple network copies and thus slows down training due to increased memory use), since it does not help.

- Use dueling DQN (see Section 2.5.4.2).

- Use distributional DQN (see Section 5.1).

2.5.4.7 Other methods

Many other methods have been proposed to reduce the sample complexity of value-based RL while maintaining performance, see e.g., the **MEME** paper of [Kap+22].

---

[3]The **Agent 57** method of [Bad+20] automatically learns the exploration rate and discount factor using a multi-armed bandit stratey, which lets it be more exploratory or more exploitative, depending on the game. This resulted in super human performance on all 57 Atari games in ALE. However, it required 80 billion frames (environment steps)! This was subsequently reduced to the "standard" 200M frames in the **MEME** method of [Kap+22].

Chapter 3

Policy-based RL

In the previous section, we considered methods that estimate the action-value function, $Q(s, a)$, from which we derive a policy. However, these methods have several disadvantages: (1) they can be difficult to apply to continuous action spaces; (2) they may diverge if function approximation is used (see Section 2.5.2.4); (3) the training of Q, often based on TD-style updates, is not directly related to the expected return garnered by the learned policy; (4) they learn deterministic policies, whereas in stochastic and partially observed environments, stochastic policies are provably better [JSJ94].

In this section, we discuss **policy search** methods, which directly optimize the parameters of the policy so as to maximize its expected return. We mostly focus on **policy gradient** methods, that use the gradient of the loss to guide the search. As we will see, these policy methods often benefit from estimating a value or advantage function to reduce the variance in the policy search process, so we will also use techniques from Chapter 2. The parametric policy will be denoted by $\pi_{\boldsymbol{\theta}}(a|s)$. For discrete actions, this can be a DNN with a softmax output. For continuous actions, we can use a Gaussian output layer, or a diffusion policy [Ren+24].

For more details on policy gradient methods, see [Wen18b; Leh24].

3.1 The policy gradient theorem

We start by defining the objective function for policy learning, and then derive its gradient. The objective, which we aim to maximize, is defined as

$$J(\pi) \triangleq \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \right] \tag{3.1}$$

$$= \sum_{t=0}^{\infty} \gamma^t \sum_s \left(\sum_{s_0} p_0(s_0) p^{\pi}(s_0 \to s, t) \right) \sum_a \pi(a|s) R(s, a) \tag{3.2}$$

$$= \sum_s \left(\sum_{s_0} \sum_{t=0}^{\infty} \gamma^t p_0(s_0) p^{\pi}(s_0 \to s, t) \right) \sum_a \pi(a|s) R(s, a) \tag{3.3}$$

$$= \sum_s \rho^{\pi}(s) \sum_a \pi(a|s) R(s, a) \tag{3.4}$$

where we have defined the discounted state visitation measure

$$\rho_{\pi}^{\gamma}(s) \triangleq \sum_{t=0}^{\infty} \gamma^t \underbrace{\sum_{s_0} p_0(s_0) p^{\pi}(s_0 \to s, t)}_{p_t^{\pi}(s)} \tag{3.5}$$

where $p^\pi(s_0 \to s, t)$ is the probability of going from s_0 to s in t steps, and $p_t^\pi(s)$ is the marginal probability of being in state s at time t (after each episodic reset). Note that ρ_π^γ is a measure of time spent in non-terminal states, but it is not a probability measure, since it is not normalized, i.e., $\sum_s \rho_\pi^\gamma(s) \neq 1$. However, we may abuse notation and still treat it like a probability, so we can write things like

$$\mathbb{E}_{\rho_\pi^\gamma(s)}[f(s)] = \sum_s \rho_\pi^\gamma(s) f(s) \tag{3.6}$$

Using this notation, we can define the objective as

$$J(\pi) = \mathbb{E}_{\rho_\pi^\gamma(s), \pi(a|s)}[R(s, a)] \tag{3.7}$$

We can also define a normalized version of the measure ρ by noting that $\sum_{t=0}^\infty \gamma^t = \frac{1}{1-\gamma}$ for $\gamma < 1$. Hence the normalized discounted state visitation distribution is given by

$$p_\pi^\gamma(s) = (1 - \gamma)\rho_\pi^\gamma(s) = (1 - \gamma) \sum_{t=0}^\infty \gamma^t p_t(s) \tag{3.8}$$

(Note the change from ρ to p.)

Note that in [SB18, Sec 13.2], they use slightly different notation. In particular, they assume $\gamma = 1$, and define the non-discounted state visitation measure as $\eta(s)$ and the corresponding normalized version by $\mu(s)$. This is equivalent to ignoring the discount factor γ^t when defining $\rho_\pi(s)$. This is standard practice in many implementations, since we can just average over (unweighted) trajectories when estimating the objective and its gradient, even though it results in a biased estimate [NT20; CVRM23].

It can be shown that the gradient of the above objective is given by

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_s \rho_\pi^\gamma(s) \sum_a Q^\pi(s, a) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) \tag{3.9}$$

$$= \sum_s \rho_\pi^\gamma(s) \sum_a Q^{\pi_{\boldsymbol{\theta}}}(s, a) \pi_{\boldsymbol{\theta}}(a|s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \tag{3.10}$$

$$= \mathbb{E}_{\rho_\pi^\gamma(s) \pi_{\boldsymbol{\theta}}(a|s)}[Q^{\pi_{\boldsymbol{\theta}}}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)] \tag{3.11}$$

This is known as the **policy gradient theorem** [Sut+99]. In statistics, the term $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ is called the (Fisher) **score function**[1], so sometimes Equation (3.11) is called the **score function estimator** or **SFE** [Fu15; Moh+20].

3.2   REINFORCE

One way to apply the policy gradient theorem to optimize a policy is to use stochastic gradient ascent. Theoretical results concerning the convergence and sample complexity of such methods can be found in [Aga+21a].

To implement such a method, let $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, \ldots, s_T)$ be a trajectory created by sampling from $s_0 \sim p_0$ and then following $\pi_{\boldsymbol{\theta}}$. Then we have

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \sum_{t=0}^\infty \gamma^t \mathbb{E}_{p_t(s) \pi_{\boldsymbol{\theta}}(a_t|s_t)}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) Q_{\pi_{\boldsymbol{\theta}}}(s_t, a_t)] \tag{3.12}$$

$$\approx \sum_{t=0}^{T-1} \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \tag{3.13}$$

---

[1]This is distinct from the Stein score, which is the gradient wrt the argument of the log probability, $\nabla_{\boldsymbol{a}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$, as used in diffusion.

where the return is defined as follows

$$G_t \triangleq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t-1} r_{T-1} = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k} = \sum_{j=t}^{T-1} \gamma^{j-t} r_j \tag{3.14}$$

See Algorithm 3 for the pseudocode.

Algorithm 3: REINFORCE (episodic version)

---

1 Initialize policy parameters $\boldsymbol{\theta}$
2 repeat
3   Sample an episode $\boldsymbol{\tau} = (s_0, a_0, r_0, s_1, \ldots, s_T)$ using $\pi_{\boldsymbol{\theta}}$
4   **for** $t = 0, 1, \ldots, T-1$ **do**
5 $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
6 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t)$
7 until *converged*

In practice, estimating the policy gradient using Equation (3.11) can have a high variance. A **baseline** function $b(s)$ can be used for variance reduction to get

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\rho_{\boldsymbol{\theta}}(s)\pi_{\boldsymbol{\theta}}(a|s)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s)(Q_{\pi_{\boldsymbol{\theta}}}(s, a) - b(s)) \right] \tag{3.15}$$

Any function that satisfies $\mathbb{E}\left[\nabla_{\boldsymbol{\theta}} b(s)\right] = 0$ is a valid baseline. This follows since

$$\sum_a \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)(Q(s, a) - b(s)) = \nabla_{\boldsymbol{\theta}} \sum_a \pi_{\boldsymbol{\theta}}(a|s) Q(s, a) - \nabla_{\boldsymbol{\theta}} [\sum_a \pi_{\boldsymbol{\theta}}(a|s)] b(s) = \nabla_{\boldsymbol{\theta}} \sum_a \pi_{\boldsymbol{\theta}}(a|s) Q(s, a) - 0 \tag{3.16}$$

A common choice for the baseline is $b(s) = V_{\pi_{\boldsymbol{\theta}}}(s)$. This is a good choice since $V_{\pi_{\boldsymbol{\theta}}}(s)$ and $Q(s, a)$ are correlated and have similar magnitudes, so the scaling factor in front of the gradient term will be small.

Using this we get an update of the following form

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \sum_{t=0}^{T-1} \gamma^t (G_t - b(s_t)) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \tag{3.17}$$

This is is called the **REINFORCE** estimator [Wil92].[2] The update equation can be interpreted as follows: we compute the sum of discounted future rewards induced by a trajectory, compared to a baseline, and if this is positive, we increase $\boldsymbol{\theta}$ so as to make this trajectory more likely, otherwise we decrease $\boldsymbol{\theta}$. Thus, we reinforce good behaviors, and reduce the chances of generating bad ones.

3.3 Actor-critic methods

An **actor-critic** method [BSA83] uses the policy gradient method, but where the expected return G_t is estimated using temporal difference learning of a value function instead of MC rollouts. (The term "actor" refers to the policy, and the term "critic" refers to the value function.) The use of bootstrapping in TD updates allows more efficient learning of the value function compared to MC, and further reduces the variance. In addition, it allows us to develop a fully online, incremental algorithm, that does not need to wait until the end of the trajectory before updating the parameters.

---

[2]The term "REINFORCE" is an acronym for "REward Increment = nonnegative Factor x Offset Reinforcement x Characteristic Eligibility". The phrase "characteristic eligibility" refers to the $\nabla \log \pi_{\boldsymbol{\theta}}(a_t|s_t)$ term; the phrase "offset reinforcement" refers to the $G_t - b(s_t)$ term; and the phrase "nonnegative factor" refers to the learning rate η of SGD.

3.3.1 Advantage actor critic (A2C)

Concretely, consider the use of the one-step TD method to estimate the return in the episodic case, i.e., we replace G_t with $G_{t:t+1} = r_t + \gamma V_{\boldsymbol{w}}(s_{t+1})$. If we use $V_{\boldsymbol{w}}(s_t)$ as a baseline, the REINFORCE update in Equation (3.17) becomes

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \sum_{t=0}^{T-1} \gamma^t \left(G_{t:t+1} - V_{\boldsymbol{w}}(s_t)\right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \tag{3.18}$$

$$= \boldsymbol{\theta} + \eta \sum_{t=0}^{T-1} \gamma^t \left(r_t + \gamma V_{\boldsymbol{w}}(s_{t+1}) - V_{\boldsymbol{w}}(s_t)\right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \tag{3.19}$$

Note that $\delta_t = r_{t+1} + \gamma V_{\boldsymbol{w}}(s_{t+1}) - V_{\boldsymbol{w}}(s_t)$ is a single sample approximation to the advantage function $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$. This method is therefore called **advantage actor critic** or **A2C**. See Algorithm 4 for the pseudo-code.[3] (Note that $V_{\boldsymbol{w}}(s_{t+1}) = 0$ if s_t is a done state, representing the end of an episode.) Note that this is an on-policy algorithm, where we update the value function $V_{\boldsymbol{w}}^\pi$ to reflect the value of the current policy π. See Section 3.3.3 for further discussion of this point.

Algorithm 4: Advantage actor critic (A2C) algorithm (episodic)

---

1 Initialize actor parameters $\boldsymbol{\theta}$, critic parameters $\boldsymbol{w}$
2 repeat
3   Sample starting state s_0 of a new episode
4   **for** $t = 0, 1, 2, \dots$ **do**
5   Sample action $a_t \sim \pi_{\boldsymbol{\theta}}(\cdot|s_t)$
6 $(s_{t+1}, r_t, \text{done}_t) = \text{env.step}(s_t, a_t)$
7 $q_t = r_t + \gamma(1 - \text{done}_t)V_{\boldsymbol{w}}(s_{t+1})$ // Target
8 $\delta_t = q_t - V_{\boldsymbol{w}}(s_t)$ // Advantage
9 $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta_{\boldsymbol{w}}\delta_t \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t)$ // Critic
10 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}}\gamma^t \delta_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)$ // Actor
11   **if** $done_t = 1$ **then**
12   break
13 until *converged*

In practice, we should use a stop-gradient operator on the target value for the TD update, for reasons explained in Section 2.5.2.4. Furthermore, it is common to add an entropy term to the policy, to act as a regularizer (to ensure the policy remains stochastic, which smoothens the loss function — see Section 3.5.4). If we use a shared network with separate value and policy heads, we need to use a single loss function for training all the parameters $\boldsymbol{\phi}$. Thus we get the following loss, for each trajectory, where we want to minimize TD loss, maximize the policy gradient (expected reward) term, and maximize the entropy term.

$$\mathcal{L}(\boldsymbol{\phi}; \tau) = \frac{1}{T} \sum_{t=1}^{T} [\lambda_{TD}\mathcal{L}_{TD}(s_t, a_t, r_t, s_{t+1}) - \lambda_{PG}J_{PG}(s_t, a_t, r_t, s_{t+1}) - \lambda_{ent}J_{ent}(s_t)] \tag{3.20}$$

$$q_t = r_t + \gamma(1 - \text{done}(s_t))V_{\boldsymbol{\phi}}(s_{t+1}) \tag{3.21}$$

$$\mathcal{L}_{TD}(s_t, a_t, r_t, s_{t+1}) = (\text{sg}(q_t) - V_\phi(s))^2 \tag{3.22}$$

$$J_{PG}(s_t, a_t, r_t, s_{t+1}) = (\text{sg}(q_t - V_\phi(s_t)) \log \pi_{\boldsymbol{\phi}}(a_t|s_t) \tag{3.23}$$

$$J_{ent}(s_t) = -\sum_a \pi_{\boldsymbol{\phi}}(a|s_t) \log \pi_{\boldsymbol{\phi}}(a|s_t) \tag{3.24}$$

---

[3]In [Mni+16], they proposed a distributed version of A2C known as **A3C** which stands for "asynchrononous advantage actor critic".

To handle the dynamically varying scales of the different loss functions, we can use the **PopArt** method of [Has+16; Hes+19] to allow for a fixed set of hyper-parameter values for λ_i. (PopArt stands for "Preserving Outputs Precisely, while Adaptively Rescaling Targets".)

3.3.2 Generalized advantage estimation (GAE)

In A2C, we replaced the high variance, but unbiased, MC return G_t with the low variance, but biased, one-step bootstrap return $G_{t:t+1} = r_t + \gamma V_{\boldsymbol{w}}(s_{t+1})$. More generally, we can compute the n-step estimate

$$G_{t:t+n} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V_{\boldsymbol{w}}(s_{t+n}) \tag{3.25}$$

and thus obtain the (truncated) n-step advantage estimate as follows:

$$A_{\boldsymbol{w}}^{(n)}(s_t, a_t) = G_{t:t+n} - V_{\boldsymbol{w}}(s_t) \tag{3.26}$$

Unrolling to infinity, we get

$$A_t^{(1)} = r_t + \gamma v_{t+1} - v_t \tag{3.27}$$

$$A_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 v_{t+2} - v_t \tag{3.28}$$

$$\vdots \tag{3.29}$$

$$A_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - v_t \tag{3.30}$$

$A_t^{(1)}$ is high bias but low variance, and $A_t^{(\infty)}$ is unbiased but high variance.

Instead of using a single value of n, we can take a weighted average. That is, we define

$$A_t = \frac{\sum_{n=1}^{T} w_n A_t^{(n)}}{\sum_{n=1}^{T} w_n} \tag{3.31}$$

If we set $w_n = \lambda^{n-1}$ we get the following simple recursive calculation:

$$\delta_t = r_t + \gamma v_{t+1} - v_t \tag{3.32}$$

$$A_t = \delta_t + \gamma \lambda \delta_{t+1} + \cdots + (\gamma \lambda)^{T-t+1} \delta_{T-1} = \delta_t + \gamma \lambda A_{t+1} \tag{3.33}$$

Here $\lambda \in [0, 1]$ is a parameter that controls the bias-variance tradeoff: larger values decrease the bias but increase the variance. This is called **generalized advantage estimation** (**GAE**) [Sch+16b]. See Algorithm 5 for some pseudocode. Using this, we can define a general actor-critic method, as shown in Algorithm 6.

Algorithm 5: Generalized Advantage Estimation

1 def GAE($r_{1:T}, v_{1:T}, \gamma, \lambda$)
2 $A' = 0$
3 **for** $t = T : 1$ **do**
4     $\delta_t = r_t + \gamma v_{t+1} - v_t$
5     $A' = \delta_t + \gamma \lambda A'$
6     $A_t = A'$ // advantage
7     $q_t = A_t + v_t$ // TD target
8 Return $(A_{1:T}), q_{1:T}$

We can generalize this approach even further, by using gradient estimators of the form

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E} \left[\sum_{t=0}^{\infty} \Psi_t \nabla \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \right] \tag{3.34}$$

Algorithm 6: Actor critic with GAE
---
1 Initialize parameters ϕ, environment state s
2 repeat
3 $(s_1, a_1, r_1, \ldots, s_T) = \text{rollout}(s, \pi_\phi)$
4 $v_{1:T} = V_\phi(s_{1:T})$
5 $(A_{1:T}, q_{1:T}) = \text{sg}(\text{GAE}(r_{1:T}, v_{1:T}, \gamma, \lambda))$
6 $\mathcal{L}(\phi) = \frac{1}{T} \sum_{t=1}^{T} \left[\lambda_{TD}(V_\phi(s_t) - q_t)^2 - \lambda_{PG} A_t \log \pi_\phi(a_t|s_t) - \lambda_{ent} \mathbb{H}(\pi_\phi(\cdot|s_t)) \right]$
7 $\phi := \phi - \eta \nabla \mathcal{L}(\phi)$
8 until *converged*
---

where Ψ_t may be any of the following:

$$\Psi_t = \sum_{i=t}^{\infty} \gamma^i r_i \qquad\qquad \text{Monte Carlo target} \qquad (3.35)$$

$$\Psi_t = \sum_{i=t}^{\infty} \gamma^i r_i - V_{\boldsymbol{w}}(s_t) \qquad\qquad \text{MC with baseline} \qquad (3.36)$$

$$\Psi_t = A_{\boldsymbol{w}}(s_t, a_t) \qquad\qquad \text{advantage function} \qquad (3.37)$$

$$\Psi_t = Q_{\boldsymbol{w}}(s_t, a_t) \qquad\qquad \text{Q function} \qquad (3.38)$$

$$\Psi_t = r_t + V_{\boldsymbol{w}}(s_{t+1}) - V_{\boldsymbol{w}}(s_t) \qquad\qquad \text{TD residual} \qquad (3.39)$$

See [Sch+16b] for details.

3.3.3   Two-time scale actor critic algorithms

In standard AC, we update the actor and critic in parallel. However, it is better to let critic $V_{\boldsymbol{w}}$ learn using a faster learning rate (or more updates), so that it reflects the value of the current policy $\pi_{\boldsymbol{\theta}}$ more accurately, in order to get better gradient estimates for the policy update. This is known as two timescale learning or **bilevel optimization** [Yu17; Zha+19; Hon+23; Zhe+22; Lor24]. (See also Section 4.2.1, where we discuss RL from a game theoretic perspective.)

3.3.4   Natural policy gradient methods

In this section, we discuss an improvement to policy gradient methods that uses preconditioning to speedup convergence. In particular, we replace gradient descent with **natural gradient descent** (**NGD**) [Ama98; Mar20], which we explain below. We then show how to combine it with actor-critic.

3.3.4.1   Natural gradient descent

NGD is a second order method for optimizing the parameters of (conditional) probability distributions, such as policies, $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$. It typically converges faster and more robustly than SGD, but is computationally more expensive.

Before we explain NGD, let us review standard SGD, which is an update of the following form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \boldsymbol{g}_k \qquad (3.40)$$

where $\boldsymbol{g}_k = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_k)$ is the gradient of the loss at the previous parameter values, and η_k is the learning rate. It can be shown that the above update is equivalent to minimizing a locally linear approximation to the loss, $\hat{\mathcal{L}}_k$, subject to the constraint that the new parameters do not move too far (in Euclidean distance) from the
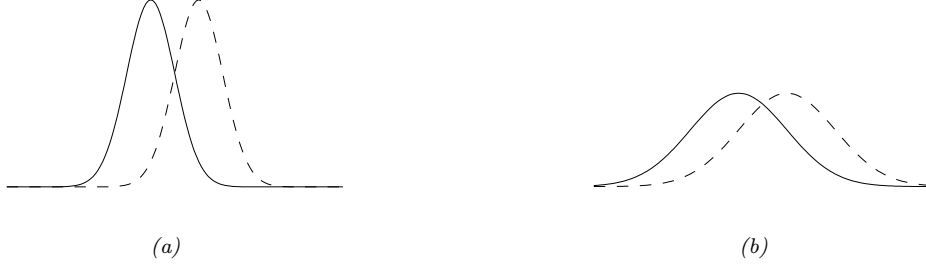
(a)   *(b)*

Figure 3.1: Changing the mean of a Gaussian by a fixed amount (from solid to dotted curve) can have more impact when the (shared) variance is small (as in a) compared to when the variance is large (as in b). Hence the impact (in terms of prediction accuracy) of a change to μ depends on where the optimizer is in (μ, σ) space. From Figure 3 of [Hon+10], reproduced from [Val00]. Used with kind permission of Antti Honkela.

previous parameters:

$$\boldsymbol{\theta}_{k+1} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \, \hat{\mathcal{L}}_k(\boldsymbol{\theta}) \;\; \text{s.t.} \;\; ||\boldsymbol{\theta} - \boldsymbol{\theta}_k||_2^2 \leq \epsilon \tag{3.41}$$

$$\hat{\mathcal{L}}_k(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_k) + \boldsymbol{g}_k^{\mathsf{T}}(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \tag{3.42}$$

where the step size η_k is proportional to ϵ. This is called a **proximal update** [PB+14].

One problem with the SGD update is that Euclidean distance in parameter space does not make sense for probabilistic models. For example, consider comparing two Gaussians, $p_{\boldsymbol{\theta}} = p(y|\mu, \sigma)$ and $p_{\boldsymbol{\theta}'} = p(y|\mu', \sigma')$. The (squared) Euclidean distance between the parameter vectors decomposes as $||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2^2 = (\mu - \mu')^2 + (\sigma - \sigma')^2$. However, the predictive distribution has the form $\exp(-\frac{1}{2\sigma^2}(y - \mu)^2)$, so changes in μ need to be measured relative to σ. This is illustrated in Figure 3.1(a-b), which shows two univariate Gaussian distributions (dotted and solid lines) whose means differ by ϵ. In Figure 3.1(a), they share the same small variance σ^2, whereas in Figure 3.1(b), they share the same large variance. It is clear that the difference in μ matters much more (in terms of the effect on the distribution) when the variance is small. Thus we see that the two parameters interact with each other, which the Euclidean distance cannot capture.

The key to NGD is to measure the notion of distance between two probability distributions in terms of the KL divergence. This can be approximated in terms of the **Fisher information matrix** (FIM). In particular, for any given input $\boldsymbol{x}$, we have

$$D_{\mathbb{KL}}\left(p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \, \| \, p_{\boldsymbol{\theta}+\boldsymbol{\delta}}(\boldsymbol{y}|\boldsymbol{x})\right) \approx \frac{1}{2}\boldsymbol{\delta}^{\mathsf{T}}\mathbf{F}_{\boldsymbol{x}}\boldsymbol{\delta} \tag{3.43}$$

where $\mathbf{F}_{\boldsymbol{x}}$ is the FIM

$$\mathbf{F}_{\boldsymbol{x}}(\boldsymbol{\theta}) = -\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[\nabla^2 \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\right] = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})}\left[(\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}))(\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}))^{\mathsf{T}}\right] \tag{3.44}$$

We now replace the Euclidean distance between the parameters, $d(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k+1}) = ||\boldsymbol{\delta}||_2^2$, with

$$d(\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k+1}) = \boldsymbol{\delta}^{\mathsf{T}}\mathbf{F}_k\boldsymbol{\delta}_k \tag{3.45}$$

where $\boldsymbol{\delta} = \boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$ and $\mathbf{F}_k = \mathbf{F}_{\boldsymbol{x}}(\boldsymbol{\theta}_k)$ for a randomly chosen input $\boldsymbol{x}$. This gives rise to the following constrained optimization problem:

$$\boldsymbol{\delta}_k = \underset{\boldsymbol{\delta}}{\operatorname{argmin}} \, \hat{\mathcal{L}}_k(\boldsymbol{\theta}_k + \boldsymbol{\delta}) \;\; \text{s.t.} \;\; \boldsymbol{\delta}^{\mathsf{T}}\mathbf{F}_k\boldsymbol{\delta} \leq \epsilon \tag{3.46}$$

If we replace the constraint with a Lagrange multiplier, we get the unconstrained objective:

$$J_k(\boldsymbol{\delta}) = \mathcal{L}(\boldsymbol{\theta}_k) + \boldsymbol{g}_k^{\mathsf{T}}\boldsymbol{\delta} + \eta_k\boldsymbol{\delta}^{\mathsf{T}}\mathbf{F}_k\boldsymbol{\delta} \tag{3.47}$$

Solving $J_k(\boldsymbol{\delta}) = 0$ gives the update

$$\boldsymbol{\delta} = -\eta_k \mathbf{F}_k^{-1} \boldsymbol{g}_k \tag{3.48}$$

The term $\mathbf{F}^{-1}\boldsymbol{g}$ is called the **natural gradient**. This is equivalent to a preconditioned gradient update, where we use the inverse FIM as a preconditioning matrix. We can compute the (adaptive) learning rate using

$$\eta_k = \sqrt{\frac{\epsilon}{\boldsymbol{g}_k^\mathsf{T} \mathbf{F}_k^{-1} \boldsymbol{g}_k}} \tag{3.49}$$

Computing the FIM can be hard. A simple approximation is to replace the model's distribution with the empirical distribution. In particular, define $p_\mathcal{D}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^N \delta_{\boldsymbol{x}_n}(\boldsymbol{x}) \delta_{\boldsymbol{y}_n}(\boldsymbol{y})$, $p_\mathcal{D}(\boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^N \delta_{\boldsymbol{x}_n}(\boldsymbol{x})$ and $p_\theta(\boldsymbol{x}, \boldsymbol{y}) = p_\mathcal{D}(\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$. Then we can compute the **empirical Fisher** [Mar16] as follows:

$$\mathbf{F}(\boldsymbol{\theta}) = \mathbb{E}_{p_\theta(\boldsymbol{x},\boldsymbol{y})} \left[\nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})^\mathsf{T} \right] \tag{3.50}$$

$$\approx \mathbb{E}_{p_\mathcal{D}(\boldsymbol{x},\boldsymbol{y})} \left[\nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})^\mathsf{T} \right] \tag{3.51}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D}} \nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \nabla \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})^\mathsf{T} \tag{3.52}$$

3.3.4.2   Natural actor critic

To apply NGD to RL, we can adapt the A2C algorithm in Algorithm 6. In particular, define

$$\boldsymbol{g}_{kt} = \nabla_{\boldsymbol{\theta}_k} A_t \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t|\boldsymbol{s}_t) \tag{3.53}$$

where A_t is the advantage function at step t of the random trajectory generated by the policy at iteration k. Now we compute

$$\boldsymbol{g}_k = \frac{1}{T} \sum_{t=1}^T \boldsymbol{g}_{kt}, \ \mathbf{F}_k = \frac{1}{T} \sum_{t=1}^T \boldsymbol{g}_{kt} \boldsymbol{g}_{kt}^\mathsf{T} \tag{3.54}$$

and compute $\boldsymbol{\delta}_{k+1} = -\eta_k \mathbf{F}_k^{-1} \boldsymbol{g}_k$. This approach is called **natural policy gradient** [Kak01; Raj+17].

We can compute $\mathbf{F}_k^{-1} \boldsymbol{g}_k$ without having to invert $\mathbf{F}_k$ by using the conjugate gradient method, where each CG step uses efficient methods for Hessian-vector products [Pea94]. This is called **Hessian free optimization** [Mar10]. Similarly, we can efficiently compute $\boldsymbol{g}_k^\mathsf{T}(\mathbf{F}_k^{-1} \boldsymbol{g}_k)$.

As a more accurate alternative to the empirical Fisher, [MG15] propose the **KFAC** method, which stands for "Kronecker factored approximate curvature"; this approximates the FIM of a DNN as a block diagonal matrix, where each block is a Kronecker product of two small matrices. This was applied to policy gradient learning in [Wu+17].

3.4   Policy improvement methods

In this section, we discuss methods that try to monotonically improve performance of the policy at each step, rather than just following the gradient, which can result in a high variance estimate where performance can increase or decrease at each step. These are called **policy improvement** methods. Our presentation is based on [QPC24].

3.4.1   Policy improvement lower bound

We start by stating a useful result from [Ach+17]. Let π_k be the current policy at step k, and let π be any other policy (e.g., a candidate new one). Let $p_{\pi_k}^\gamma$ be the normalized discounted state visitation distribution for π_k, defined in Equation (3.8). Let $A^{\pi_k}(s, a) = Q^{\pi_k}(s, a) - V^{\pi_k}(s)$ be the advantage function. Finally, let the total variation distance between two distributions be given by

$$\mathrm{TV}(p, q) \triangleq \frac{1}{2} ||\boldsymbol{p} - \boldsymbol{q}||_1 = \frac{1}{2} \sum_s |p(s) - q(s)| \tag{3.55}$$

Then one can show [Ach+17] that

$$J(\pi) - J(\pi_k) \geq \frac{1}{1-\gamma} \underbrace{\mathbb{E}_{p_{\pi_k}^\gamma(s)\pi_k(a|s)}\left[\frac{\pi(a|s)}{\pi_k(a|s)}A^{\pi_k}(s,a)\right]}_{L(\pi,\pi_k)} - \frac{2\gamma C^{\pi,\pi_k}}{(1-\gamma)^2}\mathbb{E}_{p_{\pi_k}^\gamma(s)}\left[\mathrm{TV}(\pi(\cdot|s),\pi_k(\cdot|s))\right] \quad (3.56)$$

where $C^{\pi,\pi_k} = \max_s |\mathbb{E}_{\pi(a|s)}[A^{\pi_k}(s,a)]|$. In the above, $L(\pi,\pi_k)$ is a surrogate objective, and the second term is a penalty term.

If we can optimize this lower bound (or a stochastic approximation, based on samples from the current policy π_k), we can guarantee monotonic policy improvement (in expectation) at each step. We will replace this objective with a trust-region update that is easier to optimize:

$$\pi_{k+1} = \operatorname*{argmax}_\pi L(\pi,\pi_k) \ \ \text{s.t.} \ \ \mathbb{E}_{p_{\pi_k}^\gamma(s)}\left[\mathrm{TV}(\pi,\pi_k)(s)\right] \leq \epsilon \quad (3.57)$$

The constraint bounds the worst-case performance decline at each update. The overall procedure becomes an approximate policy improvement method. There are various ways of implementing the above method in practice, some of which we discuss below. (See also [GDWF22], who propose a framework called **mirror learning**, that justifies these "approximations" as in fact being the optimal thing to do for a different kind of objective.)

3.4.2 Trust region policy optimization (TRPO)

In this section, we describe the **trust region policy optimization** (**TRPO**) method of [Sch+15b]. This implements an approximation to Equation (3.57). First, it leverages the fact that if

$$\mathbb{E}_{p_{\pi_k}^\gamma(s)}\left[D_{\mathbb{KL}}\left(\pi_k \parallel \pi\right)(s)\right] \leq \delta \quad (3.58)$$

then π also satisfies the TV constraint with $\delta = \frac{\epsilon^2}{2}$. Next it considers a first-order expansion of the surrogate objective to get

$$L(\pi,\pi_k) = \mathbb{E}_{p_{\pi_k}^\gamma(s)\pi_k(a|s)}\left[\frac{\pi(a|s)}{\pi_k(a|s)}A^{\pi_k}(s,a)\right] \approx \boldsymbol{g}_k^\mathsf{T}(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \quad (3.59)$$

where $\boldsymbol{g}_k = \nabla_{\boldsymbol{\theta}} L(\pi_{\boldsymbol{\theta}},\pi_k)|_{\boldsymbol{\theta}_k}$. Finally it considers a second-order expansion of the KL term to get the approximate constraint

$$\mathbb{E}_{p_{\pi_k}^\gamma(s)}\left[D_{\mathbb{KL}}\left(\pi_k \parallel \pi\right)(s)\right] \approx \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^\mathsf{T}\mathbf{F}_k(\boldsymbol{\theta} - \boldsymbol{\theta}_k) \quad (3.60)$$

where $\mathbf{F}_k = \boldsymbol{g}_k\boldsymbol{g}_k^\mathsf{T}$ is an approximation to the Fisher information matrix (see Equation (3.54)). We then use the update

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \eta_k\boldsymbol{v}_k \quad (3.61)$$

where $\boldsymbol{v}_k = \mathbf{F}_k^{-1}\boldsymbol{g}_k$ is the natural gradient, and the step size is initialized to $\eta_k = \sqrt{\frac{2\delta}{\boldsymbol{v}_k^\mathsf{T}\mathbf{F}_k\boldsymbol{v}_k}}$. (In practice we compute $\boldsymbol{v}_k$ by approximately solving the linear system $\mathbf{F}_k\boldsymbol{v} = \boldsymbol{g}_k$ using conjugate gradient methods, which just require matrix vector multiplies.) We then use a backtracking line search procedure to ensure the trust region is satisfied.

3.4.3 Proximal Policy Optimization (PPO)

In this section, we describe the the **proximal policy optimization** or **PPO** method of [Sch+17], which is a simplification of TRPO.

We start by noting the following result:

$$\mathbb{E}_{p_{\pi_k}^\gamma(s)}\left[\mathrm{TV}(\pi,\pi_k)(s)\right] = \frac{1}{2}\mathbb{E}_{(s,a)\sim p_{\pi_k}^\gamma}\left[\left|\frac{\pi(a|s)}{\pi_k(a|s)} - 1\right|\right] \quad (3.62)$$

This holds provided the support of π is contained in the support of π_k at every state. We then use the following update:

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{(s,a) \sim p_{\pi_k}^\gamma} \left[\min \left(\rho_k(s,a) A^{\pi_k}(s,a), \tilde{\rho}_k(s,a) A^{\pi_k}(s,a) \right) \right] \tag{3.63}$$

where $\rho_k(s,a) = \frac{\pi(a|s)}{\pi_k(a|s)}$ is the likelihood ratio, and $\tilde{\rho}_k(s,a) = \operatorname{clip}(\rho_k(s,a), 1-\epsilon, 1+\epsilon)$, where $\operatorname{clip}(x,l,u) = \min(\max(x,l),u)$. See [GDWF22] for a theoretical justification for these simplifications. Furthermore, this can be modified to ensure monotonic improvement as discussed in [WHT19], making it a true bound optimization method.

Some pseudocode for PPO (with GAE) is given in Algorithm 7. It is basically identical to the AC code in Algorithm 6, except the policy loss has the form $\min(\rho_t A_t, \tilde{\rho}_t A_t)$ instead of $A_t \log \pi_\phi(a_t|s_t)$, and we perform multiple policy updates per rollout, for increased sample efficiency. For all the implementation details, see https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Algorithm 7: PPO with GAE

1 Initialize parameters ϕ, environment state s
2 **for** *iteration* $k = 1, 2, \ldots$ **do**
3 $(\tau, s) = \operatorname{rollout}(s, \pi_\phi)$
4 $(s_1, a_1, r_1, \ldots, s_T) = \tau$
5 $v_t = V_\phi(s_t)$ for $t = 1 : T$
6 $(A_{1:T}, q_{1:T}) = \operatorname{GAE}(r_{1:T}, v_{1:T}, \gamma, \lambda)$
7 $\phi_{\text{old}} \leftarrow \phi$
8   **for** $m = 1 : M$ **do**
9 $\rho_t = \frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}$ for $t = 1 : T$
10 $\tilde{\rho}_t = \operatorname{clip}(\rho_t)$ for $t = 1 : T$
11 $\mathcal{L}(\phi) = \frac{1}{T} \sum_{t=1}^T \left[\lambda_{TD} (V_\phi(s_t) - q_t)^2 - \lambda_{PG} \min(\rho_t A_t, \tilde{\rho}_t A_t) - \lambda_{ent} \mathbb{H}(\pi_\phi(\cdot|s_t)) \right]$
12 $\phi := \phi - \eta \nabla_\phi \mathcal{L}(\phi)$

3.4.4   VMPO

In this section, we discuss the **VMPO** algorithm of [FS+19], which is an on-policy extenson of the earlier on-policy **MPO** algorithm (MAP policy optimization) from [Abd+18]. It was originally explained in terms of "control as inference" (see Section 1.5), but we can also view it as a contrained policy improvement method, based on Equation (3.57). In particular, VMPO leverages the fact that if

$$\mathbb{E}_{p_{\pi_k}^\gamma(s)} \left[D_{\mathbb{KL}} \left(\pi \parallel \pi_k \right)(s) \right] \leq \delta \tag{3.64}$$

then π also satisfies the TV constraint with $\delta = \frac{\epsilon^2}{2}$.

Note that here the KL is reversed compared to TRPO in Section 3.4.2. This new version will encourage π to be mode-covering, so it will naturally have high entropy, which can result in improved robustness. Unfortunately, this kind of KL is harder to compute, since we are taking expectations wrt the unknown distribution π.

To solve this problem, VMPO adopts an EM-type approach. In the E step, we compute a non-parametric version of the state-action distribution given by the unknown new policy:

$$\psi(s,a) = \pi(a|s) p_{\pi_k}^\gamma(s) \tag{3.65}$$

The optimal new distribution is given by

$$\psi_{k+1} = \underset{\psi}{\operatorname{argmax}} \, \mathbb{E}_{\psi(s,a)} \left[A^{\pi_k}(s,a) \right] \quad \text{s.t.} \quad D_{\mathbb{KL}} \left(\psi \parallel \psi_k \right) \leq \delta \tag{3.66}$$

where $\psi_k(s,a) = \pi_k(a|s)p_{\pi_k}^{\gamma}(s)$. The solution to this is

$$\psi_{k+1}(s,a) = p_{\pi_k}^{\gamma}(s)\pi_k(a|s)w(s,a) \tag{3.67}$$

$$w(s,a) = \frac{\exp(A^{\pi_k}(s,a)/\lambda^*)}{Z(\lambda^*)} \tag{3.68}$$

$$Z(\lambda) = \mathbb{E}_{(s,a)\sim p_{\pi_k}^{\gamma}}\left[\exp(A^{\pi_k}(s,a)/\lambda)\right] \tag{3.69}$$

$$\lambda^* = \underset{\lambda \geq 0}{\operatorname{argmin}}\, \lambda\delta + \lambda \log Z(\lambda) \tag{3.70}$$

In the M step, we project this target distribution back onto the space of parametric policies, while satisfying the KL trust region constraint:

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(s,a)\sim p_{\pi_k}^{\gamma}}\left[w(s,a)\log\pi(a|s)\right] \quad \text{s.t.} \quad \mathbb{E}_{p_{\pi_k}^{\gamma}}\left[D_{\mathbb{KL}}\left(\psi_k \,\|\, \psi\right)(s)\right] \leq \delta \tag{3.71}$$

3.5 Off-policy methods

In many cases, it is useful to train a policy using data collected from a distinct **behavior policy** $\pi_b(a|s)$ that is not the same as the **target policy** $\pi(a|s)$ that is being learned. For example, this could be data collected from earlier trials or parallel workers (with different parameters $\boldsymbol{\theta'}$) and stored in a **replay buffer**, or it could be **demonstration data** from human experts. This is known as **off-policy RL**, and can be much more sample efficient than the on-policy methods we have discussed so far, since these methods can use data from multiple sources. However, off-policy methods are more complicated, as we will explain below.

The basic difficulty is that the target policy that we want to learn may want to try an action in a state that has not been experienced before in the existing data, so there is no way to predict the outcome of this new (s,a) pair. In this section, we tackle this problem by assuming that the target policy is not too different from the behavior policy, so that the ratio $\pi(a|s)/\pi_b(a|s)$ is bounded, which allows us to use methods based on importance sampling. In the online learning setting, we can ensure this property by using conservative incremental updates to the policy. Alternatively we can use policy gradient methods with various regularization methods, as we discuss below.

In Section 5.5, we discuss offline RL, which is an extreme instance of off-policy RL where we have a fixed behavioral dataset, possibly generated from an unknown behavior policy, and can never collect any new data.

3.5.1 Policy evaluation using importance sampling

Assume we have a dataset of the form $\mathcal{D} = \{\boldsymbol{\tau}^{(i)}\}_{1 \leq i \leq n}$, where each trajectory is a sequence $\boldsymbol{\tau}^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)} \ldots)$, where the actions are sampled according to a behavior policy π_b, and the reward and next states are sampled according to the reward and transition models. We want to use this offline dataset to evaluate the performance of some target policy π; this is called **off-policy policy evaluation** or **OPE**. If the trajectories $\boldsymbol{\tau}^{(i)}$ were sampled from π. we could use the standard Monte Carlo estimate:

$$\hat{J}(\pi) \triangleq \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1}\gamma^t r_t^{(i)} \tag{3.72}$$

However, since the trajectories are sampled from π_b, we use **importance sampling** (IS) to correct for the distributional mismatch, as first proposed in [PSS00]. This gives

$$\hat{J}_{\mathrm{IS}}(\pi) \triangleq \frac{1}{n}\sum_{i=1}^{n}\frac{p(\boldsymbol{\tau}^{(i)}|\pi)}{p(\boldsymbol{\tau}^{(i)}|\pi_b)}\sum_{t=0}^{T-1}\gamma^t r_t^{(i)} \tag{3.73}$$

It can be verified that $\mathbb{E}_{\pi_b}\left[\hat{J}_{\mathrm{IS}}(\pi)\right] = J(\pi)$, that is, $\hat{J}_{\mathrm{IS}}(\pi)$ is **unbiased**, provided that $p(\boldsymbol{\tau}|\pi_b) > 0$ whenever $p(\boldsymbol{\tau}|\pi) > 0$. The **importance ratio**, $\frac{p(\boldsymbol{\tau}^{(i)}|\pi)}{p(\boldsymbol{\tau}^{(i)}|\pi_b)}$, is used to compensate for the fact that the data is sampled

from π_b and not π. It can be simplified as follows:

$$\frac{p(\boldsymbol{\tau}|\pi)}{p(\boldsymbol{\tau}|\pi_b)} = \frac{p(s_0)\prod_{t=0}^{T-1}\pi(a_t|s_t)p_S(s_{t+1}|s_t,a_t)p_R(r_t|s_t,a_t,s_{t+1})}{p(s_0)\prod_{t=0}^{T-1}\pi_b(a_t|s_t)p_S(s_{t+1}|s_t,a_t)p_R(r_t|s_t,a_t,s_{t+1})} = \prod_{t=0}^{T-1}\frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)} \tag{3.74}$$

This simplification makes it easy to apply IS, as long as the target and behavior policies are known. (If the behavior policy is unknown, we can estimate it from $\mathcal{D}$, and replace π_b by its estimate $\hat{\pi}_b$. For convenience, define the **per-step importance ratio** at time t by

$$\rho_t(\boldsymbol{\tau}) \triangleq \pi(a_t|s_t)/\pi_b(a_t|s_t) \tag{3.75}$$

We can reduce the variance of the estimator by noting that the reward r_t is independent of the trajectory beyond time t. This leads to a **per-decision importance sampling** variant:

$$\hat{J}_{\text{PDIS}}(\pi) \triangleq \frac{1}{n}\sum_{i=1}^{n}\sum_{t=0}^{T-1}\prod_{t'\leq t}\rho_{t'}(\boldsymbol{\tau}^{(i)})\gamma^t r_t^{(i)} \tag{3.76}$$

3.5.2 Off-policy actor critic methods

In this section, we discuss how to extend actor-critic methods to work with off-policy data.

3.5.2.1 Learning the critic using V-trace

In this section we build on Section 3.5.1 to develop a practical method, known as **V-trace** [Esp+18], to estimate the value function for a target policy using off-policy data. (This is an extension of the earlier **Retrace** algorithm [Mun+16], which estimates the Q function using off-policy data.)

First consider the n-step target value for $V(s_i)$ in the on-policy case:

$$V_i = V(s_i) + \sum_{t=i}^{i+n-1}\gamma^{t-i}r_t + \gamma^n V(s_{i+n}) \tag{3.77}$$

$$= V(s_i) + \sum_{t=i}^{i+n-1}\gamma^{t-i}\underbrace{(r_t + \gamma V(s_{t+1}) - V(s_t))}_{\delta_t} \tag{3.78}$$

where we define $\delta_t = (r_t + \gamma V(s_{t+1}) - V(s_t))$ as the TD error at time t. To extend this to the off-policy case, we use the per-step importance ratio trick. However, to bound the variance of the estimator, we truncate the IS weights. In particular, we define

$$c_t = \min\left(\bar{c}, \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}\right), \ \rho_t = \min\left(\bar{\rho}, \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}\right) \tag{3.79}$$

where $\bar{c}$ and $\bar{\rho}$ are hyperparameters. We then define the V-trace target value for $V(s_i)$ as

$$v_i = V(s_i) + \sum_{t=i}^{i+n-1}\gamma^{t-i}\left(\prod_{t'=i}^{t-1}c_{t'}\right)\rho_t\delta_t \tag{3.80}$$

Note that we can compute these targets recursively using

$$v_i = V(s_i) + \rho_i\delta_i + \gamma c_i(v_{i+1} - V(s_{i+1})) \tag{3.81}$$

The product of the weights $c_i\ldots c_{t-1}$ (known as the "trace") measures how much a temporal difference δ_t at time t impacts the update of the value function at earlier time i. If the policies are very different, the

variance of this product will be large. So the truncation parameter $\bar{c}$ is used to reduce the variance. In [Esp+18], they find $\bar{c} = 1$ works best.

The use of the target $\rho_t \delta_t$ rather than δ_t means we are evaluating the value function for a policy that is somewhere between π_b and π. For $\bar{\rho} = \infty$ (i.e., no truncation), we converge to the value function V^π, and for $\bar{\rho} \to 0$, we converge to the value function V^{π_b}. In [Esp+18], they find $\bar{\rho} = 1$ works best.

Note that if $\bar{c} = \bar{\rho}$, then $c_i = \rho_i$. This gives rise to the simplified form

$$v_t = V(s_t) + \sum_{j=0}^{n-1} \gamma^j \left(\prod_{m=0}^{j} c_{t+m} \right) \delta_{t+j} \tag{3.82}$$

We can use the above V-trace targets to learn an approximate value function by minimizing the usual ℓ_2 loss:

$$\mathcal{L}(\boldsymbol{w}) = \mathbb{E}_{t \sim \mathcal{D}} \left[(v_t - V_{\boldsymbol{w}}(s_t))^2 \right] \tag{3.83}$$

the gradient of which has the form

$$\nabla \mathcal{L}(\boldsymbol{w}) = 2 \mathbb{E}_{t \sim \mathcal{D}} \left[(v_t - V_{\boldsymbol{w}}(s_t)) \nabla_{\boldsymbol{w}} V_{\boldsymbol{w}}(s_t) \right] \tag{3.84}$$

3.5.2.2 Learning the actor

We now discuss how to update the actor using an off-policy estimate of the policy gradient. We start by defining the objective to be the expected value of the new policy, where the states are drawn from the behavior policy's state distribution, but the actions are drawn from the target policy:

$$J_{\pi_b}(\pi_{\boldsymbol{\theta}}) = \sum_s p_{\pi_b}^\gamma(s) V_\pi(s) = \sum_s p_{\pi_b}^\gamma(s) \sum_a \pi_{\boldsymbol{\theta}}(a|s) Q_\pi(s, a) \tag{3.85}$$

Differentiating this and ignoring the term $\nabla_{\boldsymbol{\theta}} Q_\pi(s, a)$, as suggested by [DWS12], gives a way to (approximately) estimate the **off-policy policy-gradient** using a one-step IS correction ratio:

$$\nabla_{\boldsymbol{\theta}} J_{\pi_b}(\pi_{\boldsymbol{\theta}}) \approx \sum_s \sum_a p_{\pi_b}^\gamma(s) \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) Q_\pi(s, a) \tag{3.86}$$

$$= \mathbb{E}_{p_{\pi_b}^\gamma(s), \pi_b(a|s)} \left[\frac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_b(a|s)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q_\pi(s, a) \right] \tag{3.87}$$

In practice, we can approximate $Q_\pi(s_t, a_t)$ by $q_t = r_t + \gamma v_{t+1}$, where v_{t+1} is the V-trace estimate for state s_{t+1}. If we use $V(s_t)$ as a baseline, to reduce the variance, we get the following gradient estimate for the policy:

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{t \sim \mathcal{D}} \left[\rho_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)(r_t + \gamma v_t - V_{\boldsymbol{w}}(s_t)) \right] \tag{3.88}$$

We can also replace the 1-step IS-weighted TD error $\rho_t(r_t + \gamma v_t - V_{\boldsymbol{w}}(s_t))$ with an IS-weighted GAE value by modifying the generalized advantage estimation method in Section 3.3.2. In particular, we just need to define $\lambda_t = \lambda \min(1, \rho_t)$. We denote the IS-weighted GAE estimate as A_t^ρ.[4]

3.5.2.3 IMPALA

As an example of an off-policy AC method, we consider **IMPALA**, which stands for "Importance Weighted Actor-Learning Architecture". [Esp+18]. This uses shared parameters for the policy and value function (with different output heads), and adds an entropy bonus to ensure the policy remains stochastic. Thus we end up with the following objective, which is very similar to on-policy actor-critic shown in Algorithm 6:

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{t \sim \mathcal{D}} \left[\lambda_{TD}(V_{\boldsymbol{\phi}}(s_t) - v_t)^2 - \lambda_{PG} A_t^\rho \log \pi_{\boldsymbol{\phi}}(a_t|s_t) - \lambda_{ent} \mathbb{H}(\pi_{\boldsymbol{\phi}}(\cdot|s_t)) \right] \tag{3.89}$$

---

[4]For an implementation, see https://github.com/google-deepmind/rlax/blob/master/rlax/_src/multistep.py#L39

The only difference from standard A2C is that we need to store the probabilities of each action, $\pi_b(a_t|s_t)$, in addition to (s_t, a_t, r_t, s_{t+1}) in the dataset $\mathcal{D}$, which can be used to compute ρ_t. [Esp+18] was able to use this method to train a single agent (using a shared CNN and LSTM for both value and policy) to play all 57 games at a high level. Furthermore, they showed that their method — thanks to its off-policy corrections — outperformed the A3C method (a parallel version of A2C) in Section 3.3.1.

3.5.3 Off-policy policy improvement methods

So far we have focused on actor-critic methods. However, policy improvement methods, such as PPO, are often preferred to AC methods, since they monotonically improve the objective. In [QPC21] they propose one way to extend PPO to the off-policy case. This method was generalized in [QPC24] to cover a variety of policy improvement algorithms, including TRPO and VMPO. We give a brief summary below.

The key insight is to realize that we can generalize the lower bound in Equation (3.56) to any reference policy

$$J(\pi) - J(\pi_k) \geq \frac{1}{1-\gamma} \mathbb{E}_{p^\gamma_{\pi_{\mathrm{ref}}}(s)\pi_k(a|s)} \left[\frac{\pi(a|s)}{\pi_{\mathrm{ref}}(a|s)} A^{\pi_k}(s,a) \right] - \frac{2\gamma C^{\pi,\pi_k}}{(1-\gamma)^2} \mathbb{E}_{p^\gamma_{\pi_{\mathrm{ref}}}(s)} \left[\mathrm{TV}(\pi(\cdot|s), \pi_{\mathrm{ref}}(\cdot|s)) \right] \quad (3.90)$$

The reference policy can be any previous policy, or a convex combination of them. In particular, if π_k is the current policy, we can consider the reference policy to be $\pi_{\mathrm{ref}} = \sum_{i=1}^M \nu_i \pi_{k-i}$, where $0 \leq \nu_i \leq 1$ and $\sum_i \nu_i = 1$ are mixture weights. We can approximate the expectation by sampling from the replay buffer, which contains samples from older policies. That is, $(s,a) \sim p^\gamma_{\pi_{\mathrm{ref}}}$ can be implemented by $i \sim \nu$ and $(s,a) \sim p^\gamma_{\pi_{k-i}}$.

To compute the advantage function A^{π_k} from off policy data, we can adapt the V-trace method of Equation (3.82) to get

$$A^{\pi_k}_{\mathrm{trace}}(s_t, a_t) = \delta_t + \sum_{j=0}^{n-1} \gamma^j \left(\prod_{m=1}^j c_{t+m} \right) \delta_{t+j} \quad (3.91)$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, and $c_t = \min\left(\bar{c}, \frac{\pi_k(a_t|s_t)}{\pi_{k-i}(a_t|s_t)}\right)$ is the truncated importance sampling ratio.

To compute the TV penalty term from off policy data, we need to choose between the PPO (Section 3.4.3), VMPO (Section 3.4.4) and TRPO (Section 3.4.2) approach. We discuss each of these cases below.

3.5.3.1 Off-policy PPO

The simplest is to use off-policy PPO, which gives an update of the following form (known as **Generalized PPO**):

$$\pi_{k+1} = \underset{\pi}{\mathrm{argmax}} \, \mathbb{E}_{i \sim \nu} \left[\mathbb{E}_{(s,a) \sim p^\gamma_{\pi_{k-i}}} \left[\min(\rho_{k-i}(s,a) A^{\pi_k}(s,a), \tilde{\rho}_{k-i}(s,a) A^{\pi_k}(s,a)) \right] \right] \quad (3.92)$$

where $\rho_{k-i}(s,a) = \frac{\pi(a|s)}{\pi_{k-i}(a|s)}$ and $\tilde{\rho}_{k-i}(s,a) = \mathrm{clip}(\frac{\pi(a|s)}{\pi_{k-i}(a|s)}, l, u)$, where $l = \frac{\pi_k(a|s)}{\pi_{k-i}(a|s)} - \epsilon$ and $u = \frac{\pi_k(a|s)}{\pi_{k-i}(a|s)} + \epsilon$. (For other off-policy variants of PPO, see e.g., [Men+23; LMW24].)

3.5.3.2 Off-policy VMPO

For an off-policy version of VMPO, see the original **MPO** method of [Abd+18]; this is derived using an EM framework, but EM is just another bound optimization algorithm [HL04], and the result is equivalent to the version presented in [QPC24].

3.5.3.3 Off-policy TRPO

For details on the off-policy version of TRPO, see [QPC24].

3.5.4 Soft actor-critic (SAC)

The **soft actor-critic** (**SAC**) algorithm [Haa+18a; Haa+18b] is an off-policy actor-critic method based on a framework known as maximum entropy RL, which we introduced in Section 1.5.3. Crucially, even though SAC is off-policy and utilizes a replay buffer to sample past experiences, the policy update is done using the actor's own probability distribution, eliminating the need to use importance sampling to correct for discrepancies between the behavior policy (used to collect data) and the target policy (used for updating), as we will see below.

We start by slightly rewriting the maxent RL objective from Equation (1.67) using modified notation:

$$J^{\text{SAC}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{p_{\pi_{\boldsymbol{\theta}}}^{\gamma}(s)\pi_{\boldsymbol{\theta}}(a|s)} \left[R(s,a) + \alpha \, \mathbb{H}(\pi_{\boldsymbol{\theta}}(\cdot|s)) \right] \tag{3.93}$$

Note that the entropy term makes the objective easier to optimize, and encourages exploration.

To optimize this, we can perform a soft policy evaluation step, and then a soft policy improvement step. In the policy evaluation step, we can repeatedly apply a modified Bellman backup operator $\mathcal{T}^{\pi}$ defined as

$$\mathcal{T}^{\pi} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim p} \left[V(\boldsymbol{s}_{t+1}) \right] \tag{3.94}$$

where

$$V(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi} \left[Q(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \right] \tag{3.95}$$

is the **soft value function**. If we iterate $Q^{k+1} = \mathcal{T}^{\pi} Q^k$,, this will converge to the soft Q function for π.

In the policy improvement step, we derive the new policy based on the soft Q function by softmaxing over the possible actions for each state. We then project the update back on to the policy class Π:

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{\mathbb{KL}} \left(\pi'(\cdot|\boldsymbol{s}_t) \, \| \, \frac{\exp(\frac{1}{\alpha} Q^{\pi_{\text{old}}}(\boldsymbol{s}_t, \cdot))}{Z^{\pi_{\text{old}}}(\boldsymbol{s}_t)} \right) \tag{3.96}$$

(The partition function $Z^{\pi_{\text{old}}}(\boldsymbol{s}_t)$ may be intractable to compute for a continuous action space, but it cancels out when we take the derivative of the objective, so this is not a problem, as we show below.) After solving the above optimization problem, we are guaranteed to satisfy the soft policy improvement theorem, i.e., $Q^{\pi_{\text{new}}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \geq Q^{\pi_{\text{old}}}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for all $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$.

The above equations are intractable in the non-tabular case, so we now extend to the setting where we use function approximation.

3.5.4.1 Policy evaluation

For policy evaluation, we hold the policy parameters π fixed and optimize the parameters $\boldsymbol{w}$ of the Q function by minimizing the soft Bellman residual

$$J_Q(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_{t+1}, \boldsymbol{s}_{t+1}) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\boldsymbol{w}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - q(r_{t+1}, \boldsymbol{s}_{t+1}) \right)^2 \right] \tag{3.97}$$

where $\mathcal{D}$ is a replay buffer,

$$q(r_{t+1}, \boldsymbol{s}_{t+1}) = r_{t+1} + \gamma V_{\overline{\boldsymbol{w}}}(\boldsymbol{s}_{t+1}) \tag{3.98}$$

is the frozen target value, and and $V_{\overline{\boldsymbol{w}}}(\boldsymbol{s})$ is a frozen version of the soft value function from Equation (3.95):

$$V_{\overline{\boldsymbol{w}}}(\boldsymbol{s}_t) = \mathbb{E}_{\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)} \left[Q_{\overline{\boldsymbol{w}}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) \right] \tag{3.99}$$

where $\overline{\boldsymbol{w}}$ is the EMA version of $\boldsymbol{w}$. (The use of a frozen target is to avoid bootstrapping instablilities discussed in Section 2.5.2.4.)

To avoid the positive overestimation bias that can occur with actor-critic methods, [Haa+18a], suggest fitting two soft Q functions, by optimizing $J_Q(\boldsymbol{w}_i)$, for $i = 1, 2$, independently. Inspired by clipped double Q learning, used in TD3 (Section 3.6.2), the targets are defined as

$$q(r_{t+1}, \boldsymbol{s}_{t+1}; \overline{\boldsymbol{w}}_{1:2}, \boldsymbol{\theta}) = r_{t+1} + \gamma \left[\min_{i=1,2} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{a}}_{t+1}) - \alpha \log \pi_{\boldsymbol{\theta}}(\tilde{\boldsymbol{a}}_{t+1}|\boldsymbol{s}_{t+1}) \right] \tag{3.100}$$

where $\tilde{\boldsymbol{a}}_{t+1} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1})$ is a sampled next action. In [Che+20], they propose the REDQ method (Section 2.5.3.3) which uses a random ensemble of $N \geq 2$ networks instead of just 2.

3.5.4.2 Policy improvement: Gaussian policy

For policy improvement, we hold the value function parameters $\boldsymbol{w}$ fixed and optimize the parameters $\boldsymbol{\theta}$ of the policy by minimizing the objective below, which is derived from the KL term by multiplying by α and dropping the constant Z term:

$$J_{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}} \left[\mathbb{E}_{\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}} \left[\alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t) - Q_{\boldsymbol{w}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \right] \tag{3.101}$$

Since we are taking gradients wrt $\boldsymbol{\theta}$, which affects the inner expectation term, we need to either use the REINFORCE estimator from Equation (3.15) or the **reparameterization trick** (see e.g., [Moh+20]). The latter is much lower variance, so is preferable.

To explain this in more detail, let us assume the policy distribution has the form $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{s}_t), \sigma^2 \mathbf{I})$. We can write the random action as $\boldsymbol{a}_t = f_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{\epsilon}_t)$, where f is a deterministic function of the state and a noise variable $\boldsymbol{\epsilon}_t$, since $\boldsymbol{a}_t = \boldsymbol{\mu}(\boldsymbol{s}_t) + \sigma^2 \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The objective now becomes

$$J_{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}, \boldsymbol{\epsilon}_t \sim \mathcal{N}} \left[\alpha \log \pi_{\boldsymbol{\theta}}(f_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{\epsilon}_t) | \boldsymbol{s}_t) - Q_{\boldsymbol{w}}(\boldsymbol{s}_t, f_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{\epsilon}_t)) \right] \tag{3.102}$$

where we have replaced the expectation of $\boldsymbol{a}_t$ wrt $\pi_{\boldsymbol{\theta}}$ with an expectation of $\boldsymbol{\epsilon}_t$ wrt its noise distribution $\mathcal{N}$. Hence we can now safely take stochastic gradients. See Algorithm 8 for the pseudocode. (Note that, for discrete actions, we can avoid the need for the reparameterization trick by computing the expectations explicitly, as discussed in Section 3.5.4.3.)

3.5.4.3 Policy improvement: softmax policy

For discrete actions, we can replace the Gaussian reparameterization with the gumbel-softmax reparameterization [JGP16; MMT17]. Alternatively, we can eschew sampling and compute the expectations over the actions explicitly, to derive lower variance versions of the equations; this is known as **SAC-Discrete** [Chr19]. The J_{π} objective can now be computed as

$$J'_{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}} \left[\sum_a \pi_{\boldsymbol{\theta}}(a | \boldsymbol{s}_t) [\alpha \log \pi_{\boldsymbol{\theta}}(a | \boldsymbol{s}_t) - Q_{\boldsymbol{w}}(\boldsymbol{s}_t, a)] \right] \tag{3.103}$$

which avoids the need for reparameterization. (In [Zho+22], they propose to augment J'_{π} with an entropy penalty, adding a term of the form $\frac{1}{2}(\mathbb{H}_{\text{old}} - \mathbb{H}_{\pi})^2$, to prevent drastic changes in the policy, where the entropy of the policy can be computed analytically per sampled state.) The J_Q term is similar to before

$$J'_Q(\boldsymbol{w}) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_{t+1}, \boldsymbol{s}_{t+1}) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_{\boldsymbol{w}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - q'(r_{t+1}, \boldsymbol{s}_{t+1}))^2 \right) \right] \tag{3.104}$$

where now the frozen target function is given by

$$q'(r_{t+1}, \boldsymbol{s}_{t+1}) = r_{t+1} + \gamma \left(\sum_{a_{t+1}} \pi_{\boldsymbol{\theta}}(a_{t+1} | \boldsymbol{s}_{t+1}) [\min_{i=1,2} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}_{t+1}, a_{t+1}) - \alpha \log \pi_{\boldsymbol{\theta}}(a_{t+1} | \boldsymbol{s}_{t+1})] \right) \tag{3.105}$$

3.5.4.4 Adjusting the temperature

In [Haa+18b] they propose to automatically adjust the temperature parameter α by optimizing

$$J(\alpha) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}, \boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}} \left[-\alpha(\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t) + \overline{H}) \right]$$

where $\overline{H}$ is the target entropy (a hyper-parameters). This objective is approximated by sampling actions from the replay buffer.

Algorithm 8: SAC
***

1 Initialize environment state s, policy parameters $\boldsymbol{\theta}$, N critic parameters $\boldsymbol{w}_i$, target parameters $\overline{\boldsymbol{w}}_i = \boldsymbol{w}_i$, replay buffer $\mathcal{D} = \emptyset$, discount factor γ, EMA rate ρ, step size η_w, η_π.

2 repeat

3   Take action $\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s})$

4 $(\boldsymbol{s}', r) = \text{step}(\boldsymbol{a}, \boldsymbol{s})$

5 $\mathcal{D} := \mathcal{D} \cup \{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')\}$

6 $\boldsymbol{s} \leftarrow \boldsymbol{s}'$

7   **for** G *updates* **do**

8   Sample a minibatch $\mathcal{B} = \{(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)\}$ from $\mathcal{D}$

9 $\boldsymbol{w} = \text{update-critics}(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$

10   Sample a minibatch $\mathcal{B} = \{(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)\}$ from $\mathcal{D}$

11 $\boldsymbol{\theta} = \text{update-policy}(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$

12 until *converged*

13 .

14 def update-critics$(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$:

15 Let $(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)_{j=1}^B = \mathcal{B}$

16 $q_j = q(r_j, \boldsymbol{s}'_j; \overline{\boldsymbol{w}}_{1:N}, \boldsymbol{\theta})$ for $j = 1 : B$

17 for $i = 1 : N$ **do**

18 $\mathcal{L}(\boldsymbol{w}_i) = \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')_j \in \mathcal{B}} (Q_{\boldsymbol{w}_i}(\boldsymbol{s}_j, \boldsymbol{a}_j) - \text{sg}(q_j))^2$

19 $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta_{\boldsymbol{w}} \nabla \mathcal{L}(\boldsymbol{w}_i)$ // Descent

20 $\overline{\boldsymbol{w}}_i := \rho \overline{\boldsymbol{w}}_i + (1 - \rho)\boldsymbol{w}_i$ //Update target networks

21 Return $\boldsymbol{w}_{1:N}, \overline{\boldsymbol{w}}_{1:N}$

22 .

23 def update-actor$(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$:

24 $\hat{Q}(s, a) \triangleq \frac{1}{N} \sum_{i=1}^N Q_{\boldsymbol{w}_i}(s, a)$ // Average critic

25 $J(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{s} \in \mathcal{B}} \left(\hat{Q}(\boldsymbol{s}, \tilde{\boldsymbol{a}}_{\boldsymbol{\theta}}(\boldsymbol{s})) - \alpha \log \pi_{\boldsymbol{\theta}}(\tilde{\boldsymbol{a}}(\boldsymbol{s})|\boldsymbol{s}) \right), \ \tilde{\boldsymbol{a}}_{\boldsymbol{\theta}}(\boldsymbol{s}) \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s})$

26 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \nabla J(\boldsymbol{\theta})$ // Ascent

27 Return $\boldsymbol{\theta}$
***

For discrete actions, temperature objective is given by

$$J'(\alpha) = \mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}} \left[\sum_a \pi_t(a|\boldsymbol{s}_t)[-\alpha(\log \pi_t(\boldsymbol{a}_t|\boldsymbol{s}_t) + \overline{H})] \right] \tag{3.106}$$

3.6 Deterministic policy gradient methods

In this section, we consider the case of a deterministic policy, that predicts a unique action for each state, so $a_t = \mu_{\boldsymbol{\theta}}(s_t)$, rather than $a_t \sim \pi_{\boldsymbol{\theta}}(s_t)$. (We require that the actions are continuous, because we will take the Jacobian of the Q function wrt the actions; if the actions are discrete, we can just use DQN.) The advantage of using a deterministic policy is that we can modify the policy gradient method so that it can work off policy without needing importance sampling, as we will see.

Following Equation (3.7), we define the value of a policy as the expected discounted reward per state:

$$J(\mu_{\boldsymbol{\theta}}) \triangleq \mathbb{E}_{\rho_{\mu_{\boldsymbol{\theta}}}(s)} [R(s, \mu_{\boldsymbol{\theta}}(s))] \tag{3.107}$$

The **deterministic policy gradient theorem** [Sil+14] tells us that the gradient of this expression is given by

$$\nabla_{\boldsymbol{\theta}} J(\mu_{\boldsymbol{\theta}}) = \mathbb{E}_{\rho_{\mu_{\boldsymbol{\theta}}}(s)} [\nabla_{\boldsymbol{\theta}} Q_{\mu_{\boldsymbol{\theta}}}(s, \mu_{\boldsymbol{\theta}}(s))] \tag{3.108}$$

$$= \mathbb{E}_{\rho_{\mu_{\boldsymbol{\theta}}}(s)} \left[\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_a Q_{\mu_{\boldsymbol{\theta}}}(s, a)|_{a=\mu_{\boldsymbol{\theta}}(s)} \right] \tag{3.109}$$

where $\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s)$ is the $M \times N$ Jacobian matrix, and M and N are the dimensions of $\mathcal{A}$ and $\boldsymbol{\theta}$, respectively. For stochastic policies of the form $\pi_{\boldsymbol{\theta}}(a|s) = \mu_{\boldsymbol{\theta}}(s) + \text{noise}$, the standard policy gradient theorem reduces to the above form as the noise level goes to zero.

Note that the gradient estimate in Equation (3.109) integrates over the states but not over the actions, which helps reduce the variance in gradient estimation from sampled trajectories. However, since the deterministic policy does not do any exploration, we need to use an off-policy method for training. This collects data from a stochastic behavior policy π_b, whose stationary state distribution is $p_{\pi_b}^{\gamma}$. The original objective, $J(\mu_{\boldsymbol{\theta}})$, is approximated by the following:

$$J_b(\mu_{\boldsymbol{\theta}}) \triangleq \mathbb{E}_{p_{\pi_b}^{\gamma}(s)} [V_{\mu_{\boldsymbol{\theta}}}(s)] = \mathbb{E}_{p_{\pi_b}^{\gamma}(s)} [Q_{\mu_{\boldsymbol{\theta}}}(s, \mu_{\boldsymbol{\theta}}(s))] \tag{3.110}$$

with the off-policy deterministic policy gradient from [DWS12] is approximated by

$$\nabla_{\boldsymbol{\theta}} J_b(\mu_{\boldsymbol{\theta}}) \approx \mathbb{E}_{p_{\pi_b}^{\gamma}(s)} [\nabla_{\boldsymbol{\theta}} [Q_{\mu_{\boldsymbol{\theta}}}(s, \mu_{\boldsymbol{\theta}}(s))]] = \mathbb{E}_{p_{\pi_b}^{\gamma}(s)} \left[\nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s) \nabla_a Q_{\mu_{\boldsymbol{\theta}}}(s, a)|_{a=\mu_{\boldsymbol{\theta}}(s)} \right] \tag{3.111}$$

where we have a dropped a term that depends on $\nabla_{\boldsymbol{\theta}} Q_{\mu_{\boldsymbol{\theta}}}(s, a)$ and is hard to estimate [Sil+14].

To apply Equation (3.111), we may learn $Q_{\boldsymbol{w}} \approx Q_{\mu_{\boldsymbol{\theta}}}$ with TD, giving rise to the following updates:

$$\delta = r_t + \gamma Q_{\boldsymbol{w}}(s_{t+1}, \mu_{\boldsymbol{\theta}}(s_{t+1})) - Q_{\boldsymbol{w}}(s_t, a_t) \tag{3.112}$$

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \eta_{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} Q_{\boldsymbol{w}}(s_t, a_t) \tag{3.113}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \eta_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s_t) \nabla_a Q_{\boldsymbol{w}}(s_t, a)|_{a=\mu_{\boldsymbol{\theta}}(s_t)} \tag{3.114}$$

So we learn both a state-action critic $Q_{\boldsymbol{w}}$ and an actor $\boldsymbol{\mu_{\theta}}$. This method avoids importance sampling in the actor update because of the deterministic policy gradient, and we avoids it in the critic update because of the use of Q-learning.

If $Q_{\boldsymbol{w}}$ is linear in $\boldsymbol{w}$, and uses features of the form $\phi(s, a) = \boldsymbol{a}^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \mu_{\boldsymbol{\theta}}(s)$, then we say the function approximator for the critic is **compatible** with the actor; in this case, one can show that the above approximation does not bias the overall gradient.

The basic off-policy DPG method has been extended in various ways, some of which we describe below.

3.6.1 DDPG

The **DDPG** algorithm of [Lil+16], which stands for "deep deterministic policy gradient", uses the DQN method (Section 2.5.2.2) to update Q that is represented by deep neural networks. In more detail, the actor tries to minimize the output of the critic by optimize

$$\mathcal{L}_{\boldsymbol{\theta}}(s) = Q_{\boldsymbol{w}}(s, \mu_{\boldsymbol{\theta}}(s)) \tag{3.115}$$

averaged over states s drawn from the replay buffer. The critic tries to minimize the 1-step TD loss

$$\mathcal{L}_{\boldsymbol{w}}(s, a, r, s') = [Q_{\boldsymbol{w}}(s, a) - (r + \gamma Q_{\overline{\boldsymbol{w}}}(s', \mu_{\boldsymbol{\theta}}(s')))]^2 \tag{3.116}$$

where $Q_{\overline{\boldsymbol{w}}}$ is the target critic network, and the samples (s, a, r, a') are drawn from a replay buffer. (See Section 2.5.2.5 for a discussion of target networks.)

The **D4PG** algorithm [BM+18], which stands for "distributed distributional DDPG", extends DDPG to handle distributed training, and to handle distributional RL (see Section 5.1).

3.6.2 Twin Delayed DDPG (TD3)

The **TD3** ("twin delayed deep deterministic") method of [FHM18] extends DDPG in 3 main ways. First, it uses **target policy smoothing**, in which noise is added to the action, to encourage generalization:

$$\tilde{\boldsymbol{a}} = \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{s}) + \text{noise} = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}) \tag{3.117}$$

Second it uses **clipped double Q learning**, which is an extension of the double Q-learning discussed in Section 2.5.3.1 to avoid over-estimation bias. In particular, the target values for TD learning are defined using

$$q(r, \boldsymbol{s}'; \overline{\boldsymbol{w}}_{1:2}, \overline{\boldsymbol{\theta}}) = r + \gamma \min_{i=1,2} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}', \pi_{\overline{\boldsymbol{\theta}}}(\boldsymbol{s}')) \tag{3.118}$$

Third, it uses **delayed policy updates**, in which it only updates the policy after the value function has stabilized. (See also Section 3.3.3.) See Algorithm 9 for the pseudcode.

Algorithm 9: TD3

1 Initialize environment state s, policy parameters $\boldsymbol{\theta}$, target policy parameters $\overline{\boldsymbol{\theta}}$, critic parameters $\boldsymbol{w}_i$, target critic parameters $\overline{\boldsymbol{w}}_i = \boldsymbol{w}_i$, replay buffer $\mathcal{D} = \emptyset$, discount factor γ, EMA rate ρ, step size $\eta_{\boldsymbol{w}}$, $\eta_{\boldsymbol{\theta}}$.

2 **repeat**

3      $\boldsymbol{a} = \mu_{\boldsymbol{\theta}}(\boldsymbol{s}) + \text{noise}$

4      $(\boldsymbol{s}', r) = \text{step}(\boldsymbol{a}, \boldsymbol{s})$

5      $\mathcal{D} := \mathcal{D} \cup \{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')\}$

6      $\boldsymbol{s} \leftarrow \boldsymbol{s}'$

7   **for** G *updates* **do**

8   Sample a minibatch $\mathcal{B} = \{(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)\}$ from $\mathcal{D}$

9          $\boldsymbol{w} = \text{update-critics}(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$

10   Sample a minibatch $\mathcal{B} = \{(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)\}$ from $\mathcal{D}$

11      $\boldsymbol{\theta} = \text{update-policy}(\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B})$

12 **until** *converged*

13 .

14 def update-critics($\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B}$):

15 Let $(\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j)_{j=1}^{B} = \mathcal{B}$

16 **for** $j = 1 : B$ **do**

17      $\tilde{\boldsymbol{a}}_j = \mu_{\overline{\boldsymbol{\theta}}}(\boldsymbol{s}'_j) + \text{clip}(\text{noise}, -c, c)$

18      $q_j = r_j + \gamma \min_{i=1,2} Q_{\overline{\boldsymbol{w}}_i}(\boldsymbol{s}'_j, \tilde{\boldsymbol{a}}_j)$

19 **for** $i = 1 : 2$ **do**

20      $\mathcal{L}(\boldsymbol{w}_i) = \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')_j \in \mathcal{B}} (Q_{\boldsymbol{w}_i}(\boldsymbol{s}_j, \boldsymbol{a}_j) - \text{sg}(q_j))^2$

21      $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i - \eta_{\boldsymbol{w}} \nabla \mathcal{L}(\boldsymbol{w}_i)$ // Descent

22      $\overline{\boldsymbol{w}}_i := \rho \overline{\boldsymbol{w}}_i + (1 - \rho) \boldsymbol{w}_i$ //Update target networks with EMA

23 Return $\boldsymbol{w}_{1:N}, \overline{\boldsymbol{w}}_{1:N}$

24 .

25 def update-actor($\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{B}$):

26 $J(\boldsymbol{\theta}) = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{s} \in \mathcal{B}} (Q_{\boldsymbol{w}_1}(\boldsymbol{s}, \mu_{\boldsymbol{\theta}}(\boldsymbol{s})))^2$

27 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \nabla J(\boldsymbol{\theta})$ // Ascent

28 $\overline{\boldsymbol{\theta}} := \rho \overline{\boldsymbol{\theta}} + (1 - \rho) \boldsymbol{\theta}$ //Update target policy network with EMA

29 Return $\boldsymbol{\theta}, \overline{\boldsymbol{\theta}}$

Chapter 4

Model-based RL

Model-free approaches to RL typically need a lot of interactions with the environment to achieve good performance. For example, state of the art methods for the Atari benchmark, such as rainbow (Section 2.5.2.2), use millions of frames, equivalent to many days of playing at the standard frame rate. By contrast, humans can achieve the same performance in minutes [Tsi+17]. Similarly, OpenAI's robot hand controller [And+20] needs 100 years of simulated data to learn to manipulate a rubiks cube.

One promising approach to greater sample efficiency is **model-based RL** (**MBRL**). In the simplest approach to MBRL, we first learn the state transition or dynamics model $p_S(s'|s,a)$ — also called a **world model** — and the reward function $R(s,a)$, using some offline trajectory data, and then we use these models to compute a policy (e.g., using dynamic programming, as discussed in Section 2.2, or using some model-free policy learning method on simulated data, as discussed in Chapter 3). It can be shown that the sample complexity of learning the dynamics is less than the sample complexity of learning the policy [ZHR24].

However, the above two-stage approach — where we first learn the model, and then plan with it — can suffer from the usual problems encountered in offline RL (Section 5.5), i.e., the policy may query the model at a state for which no data has been collected, so predictions can be unreliable, causing the policy to learn the wrong thing. To get better results, we have to interleave the model learning and policy learning, so that one helps the other (since the policy determines what data is collected).

There are two main ways to perform MBRL. In the first approach, known as **decision-time planning** or **model predictive control**, we use the model to choose the next action by searching over possible future trajectories. We then score each trajectory, pick the action corresponding to the best one, take a step in the environment, and repeat. (We can also optionally update the model based on the rollouts.) This is discussed in Section 4.1.

The second approach is to use the current model and policy to rollout imaginary trajectories, and to use this data (optionally in addition to empirical data) to improve the policy using model-free RL; this is called **background planning**, and is discussed in Section 4.2.

The advantage of decision-time planning is that it allows us to train a world model on reward-free data, and then use that model to optimize any reward function. This can be particularly useful if the reward contains changing constraints, or if it is an intrinsic reward (Section 5.2.4) that frequently changes based on the knowledge state of the agent. The downside of decision-time planning is that it is much slower. However, it is possible to combine the two methods, as we discuss below. For an empirical comparison of background planning and decision-time planning, see [AP24].

Some generic pseudo-code for an MBRL agent is given in Algorithm 10. (The `rollout` function is defined in Algorithm 11; some simple code for model learning is shown in Algorithm 12, although we discuss other loss functions in Section 4.3; finally, the code for the policy learning is given in other parts of this manuscript.) For more details on general MBRL, see e.g., [Wan+19; Moe+23; PKP21].

Algorithm 10: MBRL agent

---

1 def MBRL-agent($M_{\text{env}}; T, H, N$):

2 Initialize state $s \sim M_{\text{env}}$

3 Initialize data buffer $\mathcal{D} = \emptyset$, model $\hat{M}$

4 Initialize value function V, policy proposal π

5 **repeat**

6 $\quad$ // Collect data from environment

7 $\quad$ $\tau_{\text{env}} = \text{rollout}(s, \pi, T, M_{\text{env}})$,

8 $\quad$ $s = \tau_{\text{env}}[-1]$,

9 $\quad$ $\mathcal{D} = \mathcal{D} \cup \tau_{\text{env}}$

10 $\quad$ // Update model

11 $\quad$ **if** *Update model online* **then**

12 $\quad\quad$ $\hat{M} = \text{update-model}(\hat{M}, \tau_{\text{env}})$

13 $\quad$ **if** *Update model using replay* **then**

14 $\quad\quad$ $\tau_{\text{replay}}^n = \text{sample-trajectory}(\mathcal{D}), n = 1 : N$

15 $\quad\quad$ $\hat{M} = \text{update-model}(\hat{M}, \tau_{\text{replay}}^{1:N})$

16 $\quad$ // Update policy

17 $\quad$ **if** *Update on-policy with real* **then**

18 $\quad\quad$ $(\pi, V) = \text{update-on-policy}(\pi, V, \tau_{\text{env}})$

19 $\quad$ **if** *Update on-policy with imagination* **then**

20 $\quad\quad$ $\tau_{\text{imag}}^n = \text{rollout}(\text{sample-init-state}(\mathcal{D}), \pi, T, \hat{M}), n = 1 : N$

21 $\quad\quad$ $(\pi, V) = \text{update-on-policy}(\pi, V, \tau_{\text{imag}}^{1:N})$

22 $\quad$ **if** *Update off-policy with real* **then**

23 $\quad\quad$ $\tau_{\text{replay}}^n = \text{sample-trajectory}(\mathcal{D}), n = 1 : N$

24 $\quad\quad$ $(\pi, V) = \text{update-off-policy}(\pi, V, \tau_{\text{replay}}^{1:N})$

25 $\quad$ **if** *Update off-policy with imagination* **then**

26 $\quad\quad$ $\tau_{\text{imag}}^n = \text{rollout}(\text{sample-state}(\mathcal{D}), \pi, T, \hat{M}), n = 1 : N$

27 $\quad\quad$ $(\pi, V) = \text{update-off-policy}(\pi, V, \tau_{\text{imag}}^{1:N})$

28 **until** *until converged*

Algorithm 11: Rollout

---

1 def rollout(s_1, π, T, M)

2 $\tau = [s_1]$

3 **for** $t = 1 : T$ **do**

4 $\quad$ $a_t = \pi(s_t)$

5 $\quad$ $(s_{t+1}, r_{t+1}) \sim M(s_t, a_t)$

6 $\quad$ $\tau + = [a_t, r_{t+1}, s_{t+1}]$

7 Return τ

Algorithm 12: Model learning

---

1 def update-model($M, \tau^{1:N}$) :

2 $\ell(M) = -\frac{1}{NT} \sum_{n=1}^{N} \sum_{(s_t, a_t, r_{t+1}, s_{t+1}) \in \tau^n} \log M(s_{t+1}, r_{t+1} | s_t, a_t)$ // NLL

3 $M = M - \eta_M \nabla_M \ell(M)$

4 Return M

4.1 Decision-time planning

If the model is known, and the state and action space is discrete and low dimensional, we can use exact techniques based on dynamic programming to compute the policy, as discussed in Section 2.2. However, for the general case, approximate methods must be used for planning, whether the model is known (e.g., for board games like Chess and Go) or learned.

One approach to approximate planning is to be lazy, and just wait until we know what state we are in, call it s_t, and then decide what to do, rather than trying to learn a policy that maps any state to the best action. This is called **decision time planning** or "**planning in the now**" [KLP11]. We discuss some variants of this approach below.

4.1.1 Model predictive control (MPC)

We now describe a method known as **receeding horizon control** or **model predictive control** (**MPC**) [MM90; CA13; RMD22]: We use the world model to predict future states and rewards that might follow from the current state for each possible sequence of future actions we might pursue, and we then take the action from the sequence that looks most promising. More precisely, at each step, we compute

$$\boldsymbol{a}_{t:t+H-1}^* = \text{planning}(s_t, M, R, \hat{V}, H) \tag{4.1}$$

$$= \underset{\boldsymbol{a}_{t:t+H-1}}{\operatorname{argmax}} \mathbb{E}_{s_{t+1:t+H} \sim M(\cdot|s_t, a_{t:t+H-1})} \left[\sum_{h=0}^{H-1} R(s_{t+h}, a_{t+h}) + \hat{V}(s_{t+H}) \right] \tag{4.2}$$

$$\pi_{\text{MPC}}(s_t) = a_t^* \tag{4.3}$$

Here, H is called the **planning horizon**, and $\hat{V}(s_{t+H})$ is an estimate of the reward-to-go at the end of this H-step look-ahead process. We can often speed up the optimization process by using a pre-trained proposal policy $a_t = \pi(s_t)$, which can be used to guide the search process, as we discuss below.

Note that MPC computes a fixed sequence of actions, $\boldsymbol{a}_{t:t+H-1}$, also called a plan, given the current state $\boldsymbol{s}_t$; since the future actions $\boldsymbol{a}_{t'}$ for $t' > t$ are independent of the future states $\boldsymbol{s}_{t'}$, this is called an **open loop controller**. Such a controller can work well in deterministic environments (where $\boldsymbol{s}_{t'}$ can be computed from $\boldsymbol{s}_t$ and the action sequence), but in general, we will need to replan at each step, as the actual next state is observed. Thus MPC is a way of creating a **closed loop controller**.

We can combine MPC with model and policy/proposal learning using the pseudocode in Algorithm 10, where the decision policy $a_t = \pi_{\text{MPC}}(s_t)$ is implemented by Equation (4.2). If we want to learn the propsoal policy $a_t = \pi(s_t)$, we should use off-policy methods, since the training data (even if imaginary) will be collected by π_{MPC} rather than by π. When learning the world model, we only need it to be locally accurate, around the current state, which means we can often use simpler models in MPC than in background planning approaches.

In the sections below, we discuss particular kinds of MPC methods. Further connections between MPC and RL are discussed in [Ber24].

4.1.2 Heuristic search

If the state and action spaces are finite, we can solve Equation (4.2) exactly, although the time complexity will typically be exponential in H. However, in many situations, we can prune off unpromising trajectories, thus making the approach feasible in large scale problems.

In particular, consider a discrete, deterministic MDP where reward maximization corresponds to finding a shortest path to a goal state. We can expand the successors of the current state according to all possible actions, trying to find the goal state. Since the search tree grows exponentially with depth, we can use a **heuristic function** to prioritize which nodes to expand; this is called **best-first search**, as illustrated in Figure 4.1.

If the heuristic function is an optimistic lower bound on the true distance to the goal, it is called **admissible**. If we aim to maximize total rewards, admissibility means the heuristic function is an upper
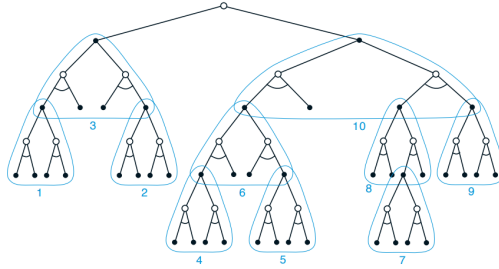
Figure 4.1: Illustration of heuristic search. In this figure, the subtrees are ordered according to a depth-first search procedure. From Figure 8.9 of [SB18]. Used with kind permission of Richard Sutton.

bound of the true value function. Admissibility ensures we will never incorrectly prune off parts of the search space. In this case, the resulting algorithm is known as A^* **search**, and is optimal. For more details on classical AI **heuristic search** methods, see [Pea84; RN19].

4.1.3 Monte Carlo tree search

Monte Carlo tree search or **MCTS** is similar to heuristic search, but learns a value function for each encountered state, rather than relying on a manually designed heuristic (see e.g., [Mun14] for details). MCTS is inspired by the upper confidence bound (UCB) method for bandits, but works for general MDPs [KS06].

4.1.3.1 AlphaGo and AlphaZero

The famous **AlphaGo** system [Sil+16], which was the first AI system to beat a human grandmaster at the board game Go, used the MCTS method, combined with a value function learned using RL, and a policy that was initialized using supervised learning from human demonstrations. This was followed up by **AlphaGoZero** [Sil+17a], which had a much simpler design, and did not train on any human data, i.e., it was trained entirely using RL and self play. It significantly outperformed the original AlphaGo. This was generalized to **AlphaZero** [Sil+18], which can play expert-level Go, chess, and shogi (Japanese chess), using a known model of the environment.

4.1.3.2 MuZero

AlphaZero assumes the world model is known. The **MuZero** method of [Sch+20] learns a world model, by training a latent representation of the observations, $z_t = \phi(o_t)$, and a corresponding latent dynamics model $z_t = M(z_t, a_t)$. The world model is trained to predict the immediate reward, the future reward (i..e, the value), and the optimal policy, where the optimal policy is computed using MCTS.

In more detail, to learn the model, MuZero uses a sum of 3 loss terms applied to each (z_{t-1}, a_t, z_t, r_t) tuple in the replay buffer. The first loss is $\mathcal{L}(r_t, \hat{r}_t)$, where r_t is the observed reward and $\hat{r}_t = R(z_t)$ is the predicted reward. The second loss is $\mathcal{L}(\pi_t^{\text{MCTS}}, \pi_t)$, where π_t^{MCTS} is the target policy from MCTS search (see below) and $\pi_t = f(z_t)$ is the predicted policy. The third loss is $\mathcal{L}(G_t^{\text{MCTS}}, v_t)$, where $G_t^{\text{MCTS}} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^k v_{t+n}$ is the n-step bootstrap target value derived from MCTS search (see below), and $v_t = V(z_t)$ is the predicted value from the current model.

To pick an action, MuZero does not use the policy directly. Instead it uses MCTS to rollout a search tree using the dynamics model, starting from the current state z_t. It uses the predicted policy π_t and value v_t as heuristics to limit the breadth and depth of the search. Each time it expands a node in the tree, it assigns it a unique integer id (since we are assuming the dynamics are deterministic), thus lazily creating a discrete MDP. It then partially solves for the tabular Q function for this MDP using Monte Carlo rollouts, similar to real-time dynamic programming (Section 2.2.2).

In more detail, the MCTS process is as follows. Let $s^k = \boldsymbol{z}_t$ be the root node, for $k = 0$. We initialize $Q(s^k, a) = 0$ and $P(s^k, a) = \boldsymbol{\pi}_t(a|s^k)$, where the latter is the prior for each action. To select the action a^k to perform next (in the rollout), we use the UCB heuristic (Section 1.4.3) based on the empirical counts $N(s, a)$ combined with the prior policy, $P(s, a)$, which act as pseudocounts. After expanding this node, we create the child node $s^{k+1} = M(s^k, a^k)$; we initialize $Q(s^{k+1}, a) = 0$ and $P(s^{k+1}, a) = \boldsymbol{\pi}_t(a|s^{k+1})$, and repeat the process until we reach a maximum depth, where we apply the value function to the corresponding leaf node. We then compute the empirical sum of discounted rewards along each of the explored paths, and use this to update the $Q(s, a)$ and $N(s, a)$ values for all visited nodes. After performing 50 such rollouts, we compute the empirical distribution over actions at the root node to get the MCTS visit count policy, $\boldsymbol{\pi}_t^{\mathrm{MCTS}}(a) = [N(s^0, a)/(\sum_b N(s^0, b))]^{1/\tau}$, where τ is a temperature. Finally we sample an action a_t from $\boldsymbol{\pi}_t^{\mathrm{MCTS}}$, take a step, add $(o_t, a_t, r_t, \boldsymbol{\pi}_t^{\mathrm{MCTS}}, G_t^{\mathrm{MCTS}})$ to the replay buffer, compute the losses, update the model and policy parameters, and repeat.

The **Stochastic MuZero** method of [Ant+22] extends MuZero to allow for stochastic environments. The **Sampled MuZero** method of [Hub+21] extends MuZero to allow for large action spaces.

4.1.3.3 EfficientZero

The **Efficient Zero** paper [Ye+21] extends MuZero by adding an additional self-prediction loss to help train the world model. (See Section 4.3.2.2 for a discussion of such losses.) It also makes several other changes. In particular, it replaces the empirical sum of instantaneous rewards, $\sum_{i=0}^{n-1} \gamma^i r_{t+i}$, used in computing G_t^{MCTS}, with an LSTM model that predicts the sum of rewards for a trajectory starting at $\boldsymbol{z}_t$; they call this the value prefix. In addition, it replaces the stored value at the leaf nodes of trajectories in the replay buffer with new values, by rerunning MCTS using the current model applied to the leaves. They show that all three changes help, but the biggest gain is from the self-prediction loss. The recent **Efficient Zero V2** [Wan+24b] extends this to also work with continuous actions, by replacing tree search with sampling-based Gumbel search, amongst other changes.

4.1.4 Trajectory optimization for continuous actions

For continuous actions, we cannot enumerate all possible branches in the search tree. Instead, we can view Equation (4.2) as a standard optimization problem over the real valued sequence of vectors $\boldsymbol{a}_{t:t+H-1}$.

4.1.4.1 Random shooting

For general nonlinear models (such as neural networks), a simple approach is to pick a sequence of random actions to try, evaluate the reward for each trajectory, and pick the best. This is called **random shooting** [Die+07; Rao10].

4.1.4.2 LQG

If the system dynamics are linear and the reward function corresponds to negative quadratic cost, the optimal action sequence can be solved mathematically, as in the **linear-quadratic-Gaussian** (**LQG**) controller (see e.g., [AM89; HR17]).

If the model is nonlinear, we can use **differential dynamic programming** (**DDP**) [JM70; TL05]. In each iteration, DDP starts with a reference trajectory, and linearizes the system dynamics around states on the trajectory to form a locally quadratic approximation of the reward function. This system can be solved using LQG, whose optimal solution results in a new trajectory. The algorithm then moves to the next iteration, with the new trajectory as the reference trajectory.

4.1.4.3 CEM

It common to use black-box (gradient-free) optimization methods like the **cross-entropy method** or **CEM** in order to find the best action sequence. The CEM method is a simple derivative-free optimization method for

continuous black-box functions $f : \mathbb{R}^D \to \mathbb{R}$. We start with a multivariate Gaussian, $\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, representing a distribution over possible action $\boldsymbol{a}$. We sample from this, evaluate all the proposals, pick the top K, then refit the Gaussian to these top K, and repeat, until we find a sample with sufficiently good score (or we perform moment matching on the top K scores). For details, see [Rub97; RK04; Boe+05].

In Section 4.1.4.4, we discuss the MPPI method, which is a common instantiation of CEM method. Another example is in the **TD-MPC** paper [HSW22a]. They learn the world model (dynamics model) in a latent space so as to predict future value and reward using temporal difference learning, and then use CEM to implement MPC for this world model. In [BXS20] they discuss how to combine CEM with gradient-based planning.

4.1.4.4 MPPI

The **model predictive path integral** or **MPPI** approach [WAT17] is a version of CEM. Originally MPPI was limited to models with linear dynamics, but it was extended to general nonlinear models in [Wil+17]. The basic idea is that the initial mean of the Gaussian at step t, namely $\boldsymbol{\mu}_t = \boldsymbol{a}_{t:t+H}$, is computed based on shifting $\hat{\boldsymbol{\mu}}_{t-1}$ forward by one step. (Here $\boldsymbol{\mu}_t$ is known as a reference trajectory.)

In [Wag+19], they apply this method for robot control. They consider a state vector of the form $\boldsymbol{s}_t = (\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$, where $\boldsymbol{q}_t$ is the configuration of the robot. The deterministic dynamics has the form

$$\boldsymbol{s}_{t+1} = F(\boldsymbol{s}_t, \boldsymbol{a}_t) = \begin{pmatrix} \boldsymbol{q}_t + \dot{\boldsymbol{q}}_t \Delta t \\ \dot{\boldsymbol{q}}_t + f(\boldsymbol{s}_t, \boldsymbol{a}_t) \Delta t \end{pmatrix} \tag{4.4}$$

where f is a 2 layer MLP. This is trained using the **Dagger** method of [RGB11], which alternates between fitting the model (using supervised learning) on the current replay buffer (initialized with expert data), and then deploying the model inside the MPPI framework to collect new data.

4.1.4.5 GP-MPC

[KD18] proposed **GP-MPC**, which combines a Gaussian process dynamics model with model predictive control. They compute a Gaussian approximation to the future state trajectory given a candidate action trajectory, $p(\boldsymbol{s}_{t+1:t+H}|\boldsymbol{a}_{t:t+H-1}, \boldsymbol{s}_t)$, by moment matching, and use this to deterministically compute the expected reward and its gradient wrt $\boldsymbol{a}_{t:t+H-1}$. Using this, they can solve Equation (4.2) to find $\boldsymbol{a}_{t:t+H-1}^*$; finally, they execute the first step of this plan, a_t^*, and repeat the whole process.

The key observation is that moment matching is a deterministic operator that maps $p(\boldsymbol{s}_t|\boldsymbol{a}_{1:t-1})$ to $p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_{1:t})$, so the problem becomes one of deterministic optimal control, for which many solution methods exist. Indeed the whole approach can be seen as a generalization of the **LQG** method from classical control, which assumes a (locally) linear dynamics model, a quadratic cost function, and a Gaussian distribution over states [Rec19]. In GP-MPC, the moment matching plays the role of local linearization.

The advantage of GP-MPC over the earlier method known as **PILCO** ("probabilistic inference for learning control"), which learns a policy by maximizing the expected reward from rollouts (see [DR11; DFR15] for details), is that GP-MPC can handle constraints more easily, and it can be more data efficient, since it continually updates the GP model after every step (instead of at the end of an trajectory).

4.1.5 SMC for MPC

A general way to tackle MPC — which supports discrete and continuous actions, as well as discrete and continuous states and linear and nonlinear world models — is to formulate it as the problem of posterior inference over state-action sequences with high reward. That is, following the control as inference framework discussed in Section 1.5, we define the goal as computing the following posterior:

$$p(\boldsymbol{x}_{1:T}|\boldsymbol{s}_1, O_{1:T}) \propto p(\boldsymbol{x}_{1:T}, O_{1:T}|\boldsymbol{s}_1) = \prod_{t=1}^{T-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t) \exp \left(\sum_{t=1}^{T} R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \log p(\boldsymbol{a}_t) \right) \tag{4.5}$$

where $\boldsymbol{x}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t)$, and O_t is the "optimality variable" which is clamped to the value 1, with distribution $p(O_t = 1 | \boldsymbol{s}_t, \boldsymbol{a}_t) = \exp(R(s_t, a_t))$. (Henceforth we will assume a uniform prior over actions, so $p(\boldsymbol{a}_t) \propto 1$.) If we can sample from this distribution, we can find state-action sequences with high expected reward, and then we can just extract the first action from one of these sampled trajectories.[1]

In practice we only compute the posterior for h steps into the future, although we still condition on optimality out to the full horizon T. Thus we define our goal as computing

$$p(\boldsymbol{x}_{1:h} | O_{1:T}) \propto \underbrace{p(\boldsymbol{x}_{1:h} | O_{1:h})}_{\alpha_h(\boldsymbol{x}_{1:h})} \underbrace{p(O_{h+1:T} | \boldsymbol{x}_h)}_{\beta_h(\boldsymbol{x}_h)} \tag{4.6}$$

where $p(O_t = 1 | s_t, a_t) = \exp(R(s_t, a_t))$ is the probability that the "optimality variable" obtains its observed (clamped) value of 1. We have decomposed the posterior as a forwards filtering term, $\alpha_h(\boldsymbol{x}_{1:h})$, and a backwards likelihood or smoothing term, $\beta_h(\boldsymbol{x}_h)$, as is standard in the literature on inference in state-space models (see e.g., [Mur23, Ch.8-9]). Note that if we define the value function as $V(\boldsymbol{s}_h) = \log p(O_{h:T} | \boldsymbol{s}_h)$, then the backwards message can be rewritten as follows [Pic+19]:

$$p(O_{h+1:T} | \boldsymbol{x}_h) = \mathbb{E}_{p(\boldsymbol{s}_{h+1} | \boldsymbol{x}_h)} [\exp(V(\boldsymbol{s}_{h+1}))] \tag{4.7}$$

A standard way to perform posterior inference in models such as these is to use **Sequential Monte Carlo** or **SMC**, which is an extension of particle filtering (i.e., sequential importance sampling with resampling) to a general sequence of distributions over a growing state space (see e.g., [Mur23, Ch 13.]). When combined with an approximation to the backwards message, the approach is called **twisted SMC** [BDM10; WL14; AL+16; Law+22; Zha+24]. This was applied to MPC in [Pic+19]. In particular, they suggest using SAC to learn a value function V, analogous to the backwards twist function, and policy π, which can be used to create the forwards proposal. More precisely, the policy can be combined with the world model $M(\boldsymbol{s}_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$ to give a (Markovian) proposal disribution over the next state and action:

$$q(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1}) = M(\boldsymbol{s}_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}) \pi(\boldsymbol{a}_t | \boldsymbol{s}_t) \tag{4.8}$$

This can then be used inside of an SMC algorithm to sample trajectories from the posterior in Equation (4.6). In particular, at each step, we sample from the proposal to extend each previous particle (sampled trajectory) by one step, and then reweight the corresponding particle using

$$w_t = \frac{p(\boldsymbol{x}_{1:T} | O_{1:T})}{q(\boldsymbol{x}_{1:t})} = \frac{p(\boldsymbol{x}_{1:t-1} | O_{1:T}) p(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1}, O_{1:T})}{q(\boldsymbol{x}_{1:t-1}) q(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1})} \tag{4.9}$$

$$= w_{t-1} \frac{p(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1}, O_{1:T})}{q(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1})} \propto w_{t-1} \frac{1}{q(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1})} \frac{p(\boldsymbol{x}_{1:t} | O_{1:T})}{p(\boldsymbol{x}_{1:t-1} | O_{1:T})} \tag{4.10}$$

Now plugging in the forward-backward equation from Equation (4.6), and doing some algebra, we get the following (see [Pic+19, App. A.4] for the detailed derivation):

$$w_t \propto w_{t-1} \frac{1}{q(\boldsymbol{x}_t | \boldsymbol{x}_{1:t-1})} \frac{p(\boldsymbol{x}_{1:t} | O_{1:t}) p(O_{t+1:T} | \boldsymbol{x}_t)}{p(\boldsymbol{x}_{1:t-1} | O_{1:t-1}) p(O_{t:T} | \boldsymbol{x}_{t-1})} \tag{4.11}$$

$$\propto w_{t-1} \mathbb{E}_{p(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, a_t)} [\exp(A(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}))] \tag{4.12}$$

where
$$A(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}) = r_t - \log \pi(\boldsymbol{a}_t | \boldsymbol{s}_t) + V(\boldsymbol{s}_{t+1}) - \mathbb{E}_{p(\boldsymbol{s}_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})} [\exp(V(\boldsymbol{s}_t))] \tag{4.13}$$

is a maximum entropy version of an advantage function. We show the overall pseudocode in Algorithm 13.

An improved version of the above method, called **Critic SMC**, is presented in [Lio+22]. The main difference is that they first extend each of the N particles (sampled trajectories) by K possible "putative actions" a_i^{nk}, then score them using a learned heuristic function $Q(s_i^n, a_i^{nk})$, then resample N winners a_i^n from

---

[1]We should really marginalize over the state sequences, and then find the maximum marginal probability action sequence, as in Equation (4.2), but we approximate this by joint sampling, for simplicity. For more discussion on this point, see [LG+24].

Algorithm 13: SMC for MPC

---

1 def SMC-MPC($\boldsymbol{s}_t, M, \pi, V, H$)
2 Initialize particles: $\{\boldsymbol{s}_t^n = \boldsymbol{s}_t\}_{n=1}^N$
3 Initialize weights: $\{w_t^n = 1\}_{n=1}^N$
4 **for** $i = t : t + H$ **do**
5 $\quad$ // Propose one-step extension
6 $\quad$ $\{\boldsymbol{a}_i^n \sim \pi(\cdot|\boldsymbol{s}_i^n)\}$
7 $\quad$ $\{(\boldsymbol{s}_{i+1}^n, r_i^n) \sim M(\cdot|\boldsymbol{s}_i^n, \boldsymbol{a}_i^n)\}$
8 $\quad$ // Update weights
9 $\quad$ $\{w_i^n \propto w_{i-1}^n \exp(A(\boldsymbol{s}_i^n, \boldsymbol{a}_i^n, \boldsymbol{s}_{i+1}^n))\}$
10 $\quad$ // Resampling
11 $\quad$ $\{\boldsymbol{x}_{1:i}^n\} \sim \text{Multinom}(n; w_i^1, \dots, w_i^N)$
12 $\quad$ $\{w_i^n = 1\}$
13 Sample $n \sim \text{Unif}(1 : N)$ // Pick one of the top samples
14 Return $\boldsymbol{a}_t^n$

---

this set of $N \times K$ particles, and then push these winners through the dynamics model to get $s_{i+1}^n \sim M(\cdot|s_i^n, a_i^n)$. Finally, they reweight the N particles by the advantage and resample, as before. This can be advantageous if the dynamics model is slow to evaluate, since we can evaluate K possible extensions just using the heuristic function. We can think of this as a form of stochastic beam search, where the beam has N candidates, and you expand each one using K possible actions, and then reduce the population (beam) back to N

4.2 Background planning

In Section 4.1, we discussed how to use models to perform decision time planning. However, this can be slow. Fortunately, we can amortize the planning process into a reactive policy. To do this, we can use the model to generate synthetic trajectories "in the background" (while executing the current policy), and use this imaginary data to train the policy; this is called "**background planning**". We discuss a game theoretic formulation of this setup in Section 4.2.1. Then in Section 4.2.2, we discuss ways to combine model-based and model-free learning. Finally, in Section 4.2.3, we discuss ways to deal with model errors, that might lead the policy astray.

4.2.1 A game-theoretic perspective on MBRL

In this section, we discuss a game-theoretic framework for MBRL, as proposed in [RMK20]. This provides a theoretical foundation for many of the more heuristic methods in the literature.

We denote the true world model by M_{env}. To simplify the notation, we assume an MDP setup with a known reward function, so all that needs to be learned is the world model, $\hat{M}$, representing $p(s'|s, a)$. (It is trivial to also learn the reward function.) We define the value of a policy π when rolled out in some model M' as the (discounted) sum of expected rewards:

$$J(\pi, M') = \mathbb{E}_{M', \pi} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right]$$

We define the loss of a model $\hat{M}$ given a distribution $\mu(s, a)$ over states and actions as

$$\ell(\hat{M}, \mu) = \mathbb{E}_{(s,a) \sim \mu} \left[D_{\mathbb{KL}} \left(M_{\text{env}}(\cdot|s, a) \parallel \hat{M}(\cdot|s, a) \right) \right]$$

We now define MBRL as a two-player general-sum game:

$$\overbrace{\max_{\pi} J(\pi, \hat{M})}^{\text{policy player}}, \overbrace{\min_{\hat{M}} \ell(\hat{M}, \mu^{\pi}_{M_{\text{env}}})}^{\text{model player}}$$

where $\mu^{\pi}_{M_{\text{env}}} = \frac{1}{T}\sum_{t=0}^{T} M_{\text{env}}(s_t = s, a_t = a)$ as the induced state visitation distribution when policy π is applied in the real world M_{env}, so that minimizing $\ell(\hat{M}, \mu^{\pi}_{M_{\text{env}}})$ gives the **maximum likelihood estimate** for $\hat{M}$.

Now consider a **Nash equilibrium** of this game, that is a pair $(\pi, \hat{M})$ that satisfies $\ell(\hat{M}, \mu^{\pi}_{M_{\text{env}}}) \leq \epsilon_{M_{\text{env}}}$ and $J(\pi, \hat{M}) \geq J(\pi', \hat{M}) - \epsilon_{\pi}$ for all π'. (That is, the model is accurate when predicting the rollouts from π, and π cannot be improved when evaluated in $\hat{M}$). In [RMK20] they prove that the Nash equilibirum policy π is near optimal wrt the real world, in the sense that $J(\pi^*, M_{\text{env}}) - J(\pi, M_{\text{env}})$ is bounded by a constant, where π^* is an optimal policy for the real world M_{env}. (The constant depends on the ϵ parameters, and the TV distance between $\mu^{\pi^*}_{M_{\text{env}}}$ and $\mu^{\pi^*}_{\hat{M}}$.)

A natural approach to trying to find such a Nash equilibrium is to use **gradient descent ascent** or **GDA**, in which each player updates its parameters simultaneously, using

$$\pi_{k+1} = \pi_k + \eta_{\pi} \nabla_{\pi} J(\pi_k, \hat{M}_k)$$
$$\hat{M}_{k+1} = \hat{M}_k - \eta_M \nabla_{\hat{M}} \ell(\hat{M}_k, \mu^{\pi_k}_{M_{\text{env}}})$$

Unfortunately, GDA is often an unstable algorithm, and often needs very small learning rates η. In addition, to increase sample efficiency in the real world, it is better to make multiple policy improvement steps using synthetic data from the model $\hat{M}_k$ at each step.

Rather than taking small steps in parallel, the **best response** strategy fully optimizes each player given the previous value of the other player, in parallel:

$$\pi_{k+1} = \underset{\pi}{\text{argmax}}\, J(\pi, \hat{M}_k)$$
$$\hat{M}_{k+1} = \underset{\hat{M}}{\text{argmin}}\, \ell(\hat{M}, \mu^{\pi_k}_{M_{\text{env}}})$$

Unfortunately, making such large updates in parallel can often result in a very unstable algorithm.

To avoid the above problems, [RMK20] propose to replace the min-max game with a **Stackelberg game**, which is a generalization of min-max games where we impose a specific player ordering. In particular, let the players be A and B, let their parameters be θ_A and θ_B, and let their losses be $\mathcal{L}_A(\theta_A, \theta_B)$ and $\mathcal{L}_B(\theta_A, \theta_B)$. If player A is the leader, the Stackelberg game corresponds to the following nested optimization problem, also called a bilevel optimization problem:

$$\min_{\theta_A} \mathcal{L}_A(\theta_A, \theta_B^*(\theta_A)) \quad \text{s.t.} \quad \theta_B^*(\theta_A) = \underset{\theta}{\text{argmin}}\, \mathcal{L}_B(\theta_A, \theta)$$

Since the follower B chooses the best response to the leader A, the follower's parameters are a function of the leader's. The leader is aware of this, and can utilize this when updating its own parameters.

The main advantage of the Stackelberg approach is that one can derive gradient-based algorithms that will provably converge to a local optimum [CMS07; ZS22]. In particular, suppose we choose the **policy as leader** (**PAL**). We then just have to solve the following optimization problem:

$$\hat{M}_{k+1} = \underset{\hat{M}}{\text{argmin}}\, \ell(\hat{M}, \mu^{\pi_k}_{M_{\text{env}}})$$
$$\pi_{k+1} = \pi_k + \eta_{\pi} \nabla_{\pi} J(\pi_k, \hat{M}_{k+1})$$

We can solve the first step by executing π_k in the environment to collect data $\mathcal{D}_k$ and then fitting a local (policy-specific) dynamics model by solving $\hat{M}_{k+1} = \text{argmin}\, \ell(\hat{M}, \mathcal{D}_k)$. (For example, this could be a locally

linear model, such as those used in trajectory optimization methods discussed in Section 4.1.4.4.) We then (slightly) improve the policy to get π_{k+1} using a conservative update algorithm, such as natural actor-critic (Section 3.3.4) or TRPO (Section 3.4.2), on "imaginary" model rollouts from $\hat{M}_{k+1}$.

Alternatively, suppose we choose the **model as leader** (**MAL**). We now have to solve

$$\pi_{k+1} = \underset{\pi}{\operatorname{argmax}} J(\pi, \hat{M}_k)$$
$$\hat{M}_{k+1} = \hat{M}_k - \eta_M \nabla_{\hat{M}} \ell(\hat{M}, \mu_{M_{\text{env}}}^{\pi_{k+1}})$$

We can solve the first step by using any RL algorithm on "imaginary" model rollouts from $\hat{M}_k$ to get π_{k+1}. We then apply this in the real world to get data $\mathcal{D}_{k+1}$, which we use to slightly improve the model to get $\hat{M}_{k+1}$ by using a conservative model update applied to $\mathcal{D}_{k+1}$. (In practice we can implement a conservative model update by mixing $\mathcal{D}_{k+1}$ with data generated from earlier models, an approach known as **data aggregation** [RB12].) Compared to PAL, the resulting model will be a more global model, since it is trained on data from a mixture of policies (including very suboptimal ones at the beginning of learning).

4.2.2   Dyna

The **Dyna** paper [Sut90] proposed an approach to MBRL that is related to the approach discussed in Section 4.2.1, in the sense that it trains a policy and world model in parallel, but it differs in one crucial way: the policy is also trained on real data, not just imaginary data. That is, we define $\pi_{k+1} = \pi_k + \eta_\pi \nabla_\pi J(\pi_k, \hat{D}_k \cup \mathcal{D}_k)$, where $\mathcal{D}_k$ is data from the real environment and $\hat{D}_k = \text{rollout}(\pi_k, \hat{M}_k)$ is imaginary data from the model. This makes Dyna a hybrid model-free and model-based RL method, rather than a "pure" MBRL method.

In more detail, at each step of Dyna, the agent collects new data from the environment and adds it to a real replay buffer. This is then used to do an off-policy update. It also updates its world model given the real data. Then it simulates imaginary data, starting from a previously visited state (see `sample-init-state` function in Algorithm 10), and rolling out the current policy in the learned model. The imaginary data is then added to the imaginary replay buffer and used by an on-policy learning algorithm. This process continue until the agent runs out of time and must take the next step in the environment.

4.2.2.1   Tabular Dyna

The original Dyna paper was developed under the assumption that the world model $s' = M(s, a)$ is deterministic and tabular, and the Q function is also tabular. See Algorithm 14 for the simplified pseudocode for this case. Since we assume a deterministic world model of the form $s' = M(s, a)$, then sampling a single step from this starting at a previously visited state is equivalent to experience replay (Section 2.5.2.3). Thus we can think of ER as a kind of non-parametric world model [HHA19].

4.2.2.2   Dyna with function approximation

It is easy to extend Dyna to work with function approximation and policy gradient methods. The code is identical to the MBRL code in Algorithm 10, where now we train the policy on real as well as imaginary data. ([Lai+21] argues that we should gradually increase the fraction of real data that is used to train the policy, to avoid suboptimal performance due to model limitations.) If we use real data from the replay buffer, we have to use an off-policy learner, since the replay buffer contains trajectories that may have been generated from old policies. (The most recent real trajectory, and all imaginary trajectories, are always from the current policy.)

We now mention some examples of this "generalized Dyna" framework. In [Sut+08] they extended Dyna to the case where the Q function is linear, and in [HTB18] they extended it to the DQN case. In [Jan+19a], they present the **MBPO** (model based policy optimization) algorithm, which uses Dyna with the off-policy SAC method. Their world model is an **ensemble of DNNs**, which generates diverse predictions (an approach which was originally proposed in the **PETS** (probabilistic ensembles with trajectory sampling) paper of [Chu+18]). In [Kur+19], they combine Dyna with TRPO (Section 3.4.2) and ensemble world models, and

---

Algorithm 14: Tabular Dyna-Q

---

1 def dyna-Q-agent($s, M_{\text{env}}; \epsilon, \eta, \gamma$):
2 Initialize data buffer $\mathcal{D} = \emptyset$, $Q(s, a) = 0$ and $\hat{M}(s, a) = 0$
3 **repeat**
4 $\quad$ // Collect real data from environment
5 $\quad$ $a = \text{eps-greedy}(Q, \epsilon)$
6 $\quad$ $(s', r) = \text{env.step}(s, a)$
7 $\quad$ $\mathcal{D} = \mathcal{D} \cup \{(s, a, r, s')\}$
8 $\quad$ // Update policy on real data
9 $\quad$ $Q(s, a) := Q(s, a) + \eta[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
10 $\quad$ // Update model on real data
11 $\quad$ $\hat{M}(s, a) = (s', r)$
12 $\quad$ $s := s'$
13 $\quad$ // Update policy on imaginary data
14 $\quad$ **for** $n=1{:}N$ **do**
15 $\quad\quad$ Select (s, a) from $\mathcal{D}$
16 $\quad\quad$ $(s', r) = \hat{M}(s, a)$
17 $\quad\quad$ $Q(s, a) := Q(s, a) + \eta[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
18 **until** *until converged*

---

in [Wu+23] they combine Dyna with PPO and GP world models. (Technically speaking, these on-policy approaches are not valid with Dyna, but they can work if the replay buffer used for policy training is not too stale.)

4.2.3 Dealing with model errors and uncertainty

The theory in Section 4.2.1 tells us that the model-as-leader approach, which trains a new policy in imagination at each inner iteration while gradually improving the model in the outer loop, will converge to the optimal policy, provided the model converges to the true model (or one that is value equivalent to it, see Section 4.3.2.1). This can be assured provided the model is sufficiently powerful, and the policy explores sufficiently widely to collect enough diverse but task-relevant data. Nevertheless, models will inevitably have errors, and it can be useful for the policy learning to be aware of this. We discuss some approaches to this below.

4.2.3.1 Avoiding compounding errors in rollouts

In MBRL, we have to rollout imaginary trajectories to use for training the policy. It makes intuitive sense to start from a previously visited real-world state, since the model will likely be reliable there. We should start rollouts from different points along each real trajectory, to ensure good state coverage, rather than just expanding around the initial state [Raj+17]. However, if we roll out too far from a previously seen state, the trajectories are likely to become less realistic, due to **compounding errors** from the model [LPC22].

In [Jan+19a], they present the MBPO method, which uses short rollouts (inside Dyna) to prevent compounding error (an approach which is justified in [Jia+15]). [Fra+24] is a recent extension of MBPO which dynamically decides how much to roll out, based on model uncertainty.

Another approach to mitigating compounding errors is to learn a trajectory-level dynamics model, instead of a single-step model, see e.g., [Zho+24] which uses diffusion to train $p(s_{t+1:t+H}|s_t, a_{t:t+H-1})$, and uses this inside an MPC loop.

If the model is able to predict a reliable distribution over future states, then we can leverage this uncertainty estimate to compute an estimate of the expected reward. For example, PILCO [DR11; DFR15] uses Gaussian processes as the world model, and uses this to analytically derive the expected reward over trajectories as a function of policy parameters, which are then optimized using a deterministic second-order

gradient-based solver. In [Man+19], they combine the MPO algorithm (Section 3.4.4) for continuous control with **uncertainty sets** on the dynamics to learn a policy that optimizes for a worst case expected return objective.

4.2.3.2 End-to-end differentiable learning of model and planner

One solution to the mismatch problem between model fitting and policy learning is to use **differentiable planning**, in which we learn the model so as to minimize the planning loss. This bilevel optimization problem was first proposed in the **Value Iteration Network** paper of [Tam+16] and extended in the **TreeQN** paper of [Far+18]. In [AY20] they proposed a version of this for continuous actions based on the differentiable cross entropy method. In [Nik+22; Ban+23] they propose to use implicit differentation to avoid explicitly unrolling the inner optimization.

4.2.3.3 Unified model and planning variational lower bound

In [Eys+22], they propose a method called **Mismatched No More** (MNM) to solve the objective mismatch problem. They define an optimality variable (see Section 1.5) based on the entire trajectory, $p(O = 1|\tau) = R(\tau) = \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)$. This gives rise to the following variational lower bound on the log probability of optimality:

$$\log p(O = 1) = \log \int_{\tau} P(O = 1, \tau) = \log \mathbb{E}_{P(\tau)}\left[P(O = 1|\tau)\right] \geq \mathbb{E}_{Q(\tau)}\left[\log R(\tau) + \log P(\tau) - \log Q(\tau)\right]$$

where $P(\tau)$ is the distribution over trajectories induced by policy applied to the true world model, $P(\tau) = \mu(s_0) \prod_{t=0}^{\infty} M(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$, and $Q(\tau)$ is the distribution over trajectories using the estimated world model, $Q(\tau) = \mu(s_0) \prod_{t=0}^{\infty} \hat{M}(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$. They then maximize this bound wrt π and $\hat{M}$.

In [Ghu+22] they extend MNM to work with images (and other high dimensional states) by learning a latent encoder $\hat{E}(z_t|o_t)$ as well as latent dynamics $\hat{M}(z_{t+1}|z_t, a_t)$, similar to other self-predictive methods (Section 4.3.2.2). They call their method **Aligned Latent Models**.

4.2.3.4 Dynamically switching between MFRL and MBRL

One problem with the above methods is that, if the model is of limited capacity, or if it learns to model "irrelevant" aspects of the environment, then any MBRL method may be dominated by a MFRL method that directly optimizes the true expected reward. A safer approach is to use a model-based policy only when the agent is confident it is better, but otherwise to fall back to a model-free policy. This is the strategy proposed in the **Unified RL** method of [Fre+24].

4.3 World models

In this section, we discuss various kinds of world models that have been proposed in the literature. These can be used for decision-time planning or for background planning

4.3.1 Generative world models

In this section, we discuss different kinds of world model $M(s'|s, a)$. We can use this to generate imaginary trajectories by sampling from the following joint distribution:

$$p(s_{t+1:T}, r_{t+1:T}, a_{t:T-1}|s_t) = \prod_{i=t}^{T-1} \pi(a_i|s_i)M(s_{i+1}|s_i, a_i)R(r_{i+1}|s_i, a_i) \tag{4.14}$$

4.3.1.1 Observation-space world models

The simplest approach is to define $M(s'|s, a)$ as a conditional generative model over states. If the state space is high dimensional (e.g., images), we can use standard techniques for image generation such as diffusion (see e.g., the **Diamond** method of [Alo+24]). If the observed states are low-dimensional vectors, such as proprioceptive states, we can use transformers (see e.g., the **Transformer Dynamics Model** of [Sch+23a]).

4.3.1.2 Factored models

In some cases, the dimensions of the state vector s represent distinct variables, and the joint Markov transition matrix $p(s'|s, a)$ has conditional independence properties which can be represented as a sparse graphical model, This is called a **factored MDP** [BDG00].

4.3.1.3 Latent-space world models

In this section, we describe some methods that use latent variables as part of their world model. We let z_t denote the latent (or hidden) state at time t; this can be a discrete or continuous variable (or vector of variables). The generative model has the form of a controlled HMM:

$$p(\boldsymbol{o}_{t+1:T}, \boldsymbol{z}_{t+1:T}, \boldsymbol{r}_{t+1:T}, \boldsymbol{a}_{t:T-1}|\boldsymbol{z}_t) = \prod_{i=t}^{T-1} [\pi(\boldsymbol{a}_i|\boldsymbol{z}_i)M(\boldsymbol{z}_{i+1}|\boldsymbol{z}_i, \boldsymbol{a}_i)R(r_i|\boldsymbol{z}_{i+1}, \boldsymbol{a}_i)D(\boldsymbol{o}_i|\boldsymbol{z}_{i+1})] \tag{4.15}$$

where $p(\boldsymbol{o}_t|\boldsymbol{z}_t) = D(\boldsymbol{o}_t|\boldsymbol{z}_t)$ is the decoder, or likelihood function, and $\pi(\boldsymbol{a}_t|\boldsymbol{z}_t)$ is the policy.

The world model is usually trained by maximizing the marginal likelihood of the observed outputs given an action sequence. (We discuss non-likelihood based loss functions in Section 4.3.2.) Computing the marginal likelihood requires marginalizing over the hidden variables $\boldsymbol{z}_{t+1:T}$. To make this computationally tractable, it is common to use amortized variational inference, in which we train an encoder network, $p(\boldsymbol{z}_t|\boldsymbol{o}_t)$, to approximate the posterior over the latents. Many papers have followed this basic approach, such as the **"world models"** paper [HS18], and the methods we discuss below.

4.3.1.4 Dreamer

In this section, we summarize the approach used in **Dreamer** paper [Haf+20] and its recent extensions, such as DreamerV2 [Haf+21] and DreamerV3 [Haf+23]. These are all based on the background planning approach, in which the policy is trained on imaginary trajectories generated by a latent variable world model. (Note that Dreamer is based on an earlier approach called **PlaNet** [Haf+19], which used MPC instead of background planning.)

In Dreamer, the stochastic dynamic latent variables in Equation (4.15) are replaced by deterministic dynamic latent variables $\boldsymbol{h}_t$, since this makes the model easier to train. (We will see that $\boldsymbol{h}_t$ acts like the posterior over the hidden state at time $t-1$; this is also the prior predictive belief state before we see $\boldsymbol{o}_t$.) A "static" stochastic variable $\boldsymbol{\epsilon}_t$ is now generated for each time step, and acts like a "random effect" in order to help generate the observations, without relying on $\boldsymbol{h}_t$ to store all of the necessary information. (This simplifies the recurrent latent state.) In more detail, Dreamer defines the following functions:[2]

- A hidden dynamics (sequence) model: $\boldsymbol{h}_{t+1} = U(\boldsymbol{h}_t, \boldsymbol{a}_t, \boldsymbol{\epsilon}_t)$

- A latent state prior: $\hat{\boldsymbol{\epsilon}}_t \sim P(\hat{\boldsymbol{\epsilon}}_t|\boldsymbol{h}_t)$

- A latent state decoder (observation predictor): $\hat{\boldsymbol{o}}_t \sim D(\hat{\boldsymbol{o}}_t|\boldsymbol{h}_t, \hat{\boldsymbol{\epsilon}}_t)$.

- A reward predictor: $\hat{r}_t \sim R(\hat{r}_t|\boldsymbol{h}_t, \hat{\boldsymbol{\epsilon}}_t)$

- A latent state encoder: $\boldsymbol{\epsilon}_t \sim E(\boldsymbol{\epsilon}_t|\boldsymbol{h}_t, \boldsymbol{o}_t)$.

---

[2]To map from our notation to the notation in the paper, see the following key: $\boldsymbol{o}_t \rightarrow x_t$, $U \rightarrow f_\phi$ (sequence model), $P \rightarrow p_\phi(\hat{z}_t|h_t)$ (dynamics predictor),ion model), $D \rightarrow p_\phi(\hat{x}_t|h_t, \hat{z}_t)$ (decoder), $E \rightarrow q_\phi(\boldsymbol{\epsilon}_t|h_t, x_t)$ (encoder).
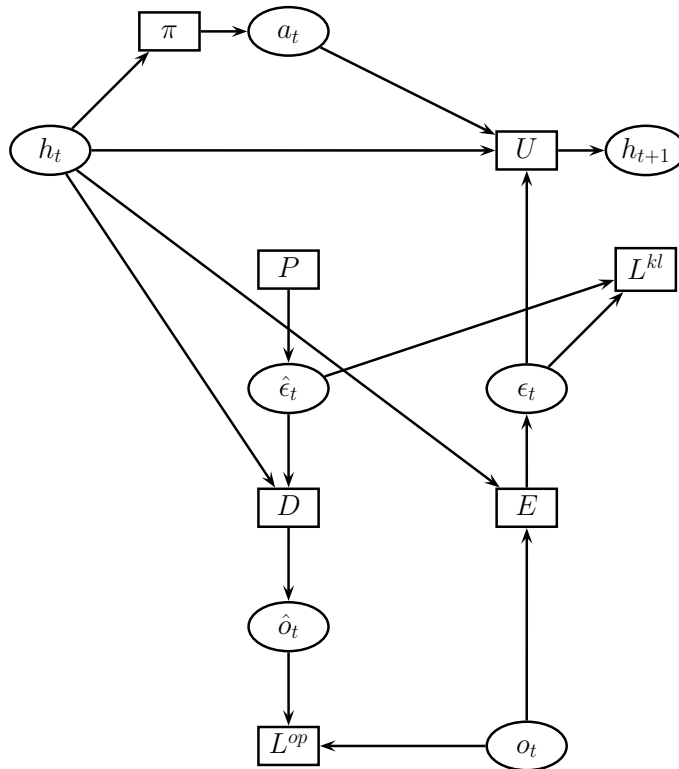
Figure 4.2: Illustration of Dreamer world model as a factor graph (so squares are functions, circles are variables). We have unrolled the forwards prediction for only 1 step. Also, we have omitted the reward prediction loss.

- A policy function: $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t | \boldsymbol{h}_t)$

See Figure 4.2 for an illustration of the system.

We now give a simplified explanation of how the world model is trained. The loss has the form

$$\mathcal{L}^{\mathrm{WM}} = \mathbb{E}_{q(\boldsymbol{\epsilon}_{1:T})} \left[\sum_{t=1}^{T} \beta_o \mathcal{L}^o(\boldsymbol{o}_t, \hat{\boldsymbol{o}}_t) + \beta_z \mathcal{L}^z(\boldsymbol{\epsilon}_t, \hat{\boldsymbol{\epsilon}}_t) \right] \tag{4.16}$$

where the β terms are different weights for each loss, and q is the posterior over the latents, given by

$$q(\boldsymbol{\epsilon}_{1:T} | \boldsymbol{h}_0, \boldsymbol{o}_{1:T}, \boldsymbol{a}_{1:T}) = \prod_{t=1}^{T} E(\boldsymbol{\epsilon}_t | \boldsymbol{h}_t, \boldsymbol{o}_t) \delta(\boldsymbol{h}_t - U(\boldsymbol{h}_{t-1}, \boldsymbol{a}_{t-1}, \boldsymbol{\epsilon}_{t-1})) \tag{4.17}$$

The loss terms are defined as follows:

$$\mathcal{L}^o = -\ln D(\boldsymbol{o}_t | \boldsymbol{\epsilon}_t, \boldsymbol{h}_t) \tag{4.18}$$
$$\mathcal{L}^z = D_{\mathbb{KL}} \left(E(\boldsymbol{\epsilon}_t | \boldsymbol{h}_t, \boldsymbol{o}_t) \parallel P(\boldsymbol{\epsilon}_t | \boldsymbol{h}_t)) \right) \tag{4.19}$$

where we abuse notation somewhat, since $\mathcal{L}^z$ is a function of two distributions, not of the variables $\boldsymbol{\epsilon}_t$ and $\hat{\boldsymbol{\epsilon}}_t$.

In addition to the world model loss, we have the following actor-critic losses

$$\mathcal{L}^{\mathrm{critic}} = \sum_{t=1}^{T} (V(\boldsymbol{h}_t) - \mathrm{sg}(G_t^\lambda))^2 \tag{4.20}$$

$$\mathcal{L}^{\mathrm{actor}} = -\sum_{t=1}^{T} \mathrm{sg}((G_t^\lambda - V(\boldsymbol{h}_t))) \log \pi(\boldsymbol{a}_t | \boldsymbol{h}_t) \tag{4.21}$$

where G_t^λ is the GAE estimate of the reward to go:

$$G_t^\lambda = r_t + \gamma \left((1 - \lambda) V(\boldsymbol{h}_t) + \lambda G_{t+1}^\lambda \right) \tag{4.22}$$

There have been several extensions to the original Dreamer paper. DreamerV2 [Haf+21] adds categorical (discrete) latents and KL balancing between prior and posterior estimates. This was the first imagination-based agent to outperform humans in Atari games. DayDreamer [Wu+22] applies DreamerV2 to real robots. DreamerV3 [Haf+23] builds upon DreamerV2 using various tricks — such as symlog encodings[3] for the reward, critic, and decoder — to enable more stable optimization and domain independent choice of hyper-parameters. It was the first method to create diamonds in the Minecraft game without requiring human demonstration data. (However, reaching this goal took 17 days of training.) [Lin+24a] extends DreamerV3 to also model language observations.

Variants of Dreamer such as TransDreamer [Che+21a] and STORM [Zha+23b] have also been explored, where transformers replace the recurrent network. The DreamingV2 paper of [OT22] replaces the generative loss with a non-generative self-prediction loss (see Section 4.3.2.2).

4.3.1.5 Iris

The **Iris** method of [MAF22] follows the MBRL paradigm, in which it alternates beween (1) learning a world model using real data D_r and then generate imaginary rollouts D_i using the WM, and (2) learning the policy given D_i and collecting new data D_r' for learning. In the model learning stage, Iris learns a discrete latent encoding using the VQ-VAE method, and then fits a transformer dynamics model to the latent codes. In the policy learning stage, it uses actor critic methods. The **Delta-Iris** method of [MAF24] extends this by training the model to only predict the delta between neighboring frames. Note that, in both cases, the policy has the form $a_t = \pi(\boldsymbol{o}_t)$, where $\boldsymbol{o}_t$ is an image, so the the rollouts need to ground to pixel space, and cannot only be done in latent space.

---

[3]The symlog function is defined as $\mathrm{symlog}(x) = \mathrm{sign}(x) \ln(|x| + 1)$, and its inverse is $\mathrm{symexp}(x) = \mathrm{sign}(x)(\exp(|x|) - 1)$. The symlog function squashes large positive and negative values, while preserving small values.

Loss	Policy	Usage	Examples
OP	Observables	Dyna	Diamond [Alo+24], Delta-Iris [MAF24]
OP	Observables	MCTS	TDM [Sch+23a]
OP	Latents	Dyna	Dreamer [Haf+23]
RP, VP, PP	Latents	MCTS	MuZero [Sch+20]
RP, VP, PP, ZP	Latents	MCTS	EfficientZero [Ye+21]
RP, VP, ZP	Latents	MPC-CEM	TD-MPC [HSW22b]
VP, ZP	Latents	Aux.	Minimalist [Ni+24]
VP, ZP	Latents	Dyna	DreamingV2 [OT22]
VP, ZP, OP	Latents	Dyna	AIS [Sub+22]
POP	Latents	Dyna	Denoised MDP [Wan+22]

Table 4.1: Summary of some world-modeling methods. The "loss" column refers to the loss used to train the latent encoder (if present) and the dynamics model (OP = observation prediction, ZP = latent state prediction, RP = reward prediction, VP = value prediction, PP = policy prediction, POP = partial observation prediction). The "policy" column refers to the input that is passed to the policy. (For MCTS methods, the policy is just used as a proposal over action sequences to initialize the search/ optimization process.) The "usage" column refers to how to the world model is used: for background planning (which we call "Dyna"), or for decision-time planning (which we call "MCTS"), or just as an auxiliary loss on top of standard policy/value learning (which we call "Aux"). Thus Aux methods are single-stage ("end-to-end"), whereas the other methods alternate are two-phase, and alternate between improving the world model and then using it for improving the policy (or searching for the optimal action).

4.3.2 Non-generative world models

In Section 4.2.1, we argued that, if we can learn a sufficiently accurate world model, then solving for the optimal policy in simulation will give a policy that is close to optimal in the real world. However, a simple agent may not be able to capture the full complexity of the true environment; this is called the "**small agent, big world**" problem [DVRZ22; Lu+23; Aru+24a; Kum+24].

Consider what happens when the agent's model is misspecified (i.e., it cannot represent the true world model), which is nearly always the case. The agent will train its model to reduce state (or observation) prediction error, by minimizing $\ell(\hat{M}, \mu_M^\pi)$. However, not all features of the state are useful for planning. For example, if the states are images, a dynamics model with limited representational capacity may choose to focus on predicting the background pixels rather than more control-relevant features, like small moving objects, since predicting the background reliably reduces the MSE more. This is due to "**objective mismatch**" [Lam+20; Wei+24], which refers to the discrepancy between the way a model is usually trained (to predict the observations) vs the way its representation is used for control. To tackle this problem, in this section we discuss methods for learning representations and models that don't rely on predicting all the observations. Our presentation is based in part on [Ni+24] (which in turn builds on [Sub+22]). See Table 4.1 for a summary of some of the methods we will discuss.

4.3.2.1 Value prediction

Let $\mathcal{D}_t = (\mathcal{D}_{t-1}, \boldsymbol{a}_{t-1}, r_{t-1}, \boldsymbol{o}_t)$ be the observed history at time t, and let $\boldsymbol{z}_t = \phi(\mathcal{D}_t)$ be a latent representation (compressed encoding) of this history, where ϕ is called an encoder or a **state abstraction** function. We will train the policy $\boldsymbol{a}_t = \pi(\boldsymbol{z}_t)$ in the usual way, so our focus will be on how to learn good latent representations.

An optimal representation $\boldsymbol{z}_t = \phi(\mathcal{D}_t)$ is a sufficient statistic for the optimal action-value function Q^*. Thus it satifies the **value equivalence** principle [LWL06; Cas11; Gri+20; GBS22; AP23; ARKP24], which says that two states s_1 and s_2 are value equivalent (given a policy) if $V^\pi(s_1) = V^\pi(s_2)$. In particular, if the representation is optimal, it will satisfy value equivalence wrt the optimal policy, i.e., if $\phi(\mathcal{D}_i) = \phi(\mathcal{D}_j)$ then $Q^*(\mathcal{D}_i, a) = Q^*(\mathcal{D}_j, a)$. We can train such a representation function by using its output $\boldsymbol{z} = \phi(\mathcal{D})$ as input to the Q function or to the policy. (We call such a loss **VP**, for value prediction.) This will cause the model to focus its representational power on the relevant parts of the observation history.

Note that there is a stronger property than value equivalence called **bisimulation** [GDG03]. This says
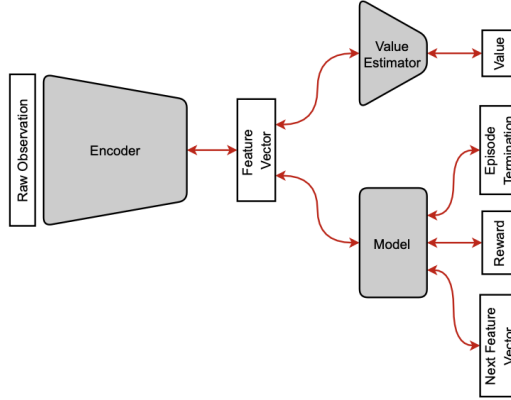
Figure 4.3: Illustration of an encoder $z_t = E(o_t)$, which is passed to a value estimator $v_t = V(z_t)$, and a world model, which predicts the next latent state $\hat{z}_{t+1} = M(z_t, a_t)$, the reward $r_t = R(z_t, a_t)$, and the termination (done) flag, $d_t = done(z_t)$. From Figure C.2 of [AP23]. Used with kind permission of Doina Precup.

that two states s_1 and s_2 are bisimiliar if $P(s'|s_1, a) \approx P(s'|s_2, a)$ and $R(s_1, a) = R(s_2, a)$. From this, we can derive a continuous measure called the **bisimulation metric** [FPP04]. This has the advantage (compared to value equivalence) of being policy independent, but the disadvantage that it can be harder to compute [Cas20; Zha+21], although there has been recent progress on computaitonally efficient methods such as MICo [Cas+21] and KSMe [Cas+23].

4.3.2.2   Self prediction

Unfortunately, in problems with sparse reward, predicting the value may not provide enough of a feedback signal to learn quickly. Consequently it is common to augment the training with a **self-prediction** loss where we train ϕ to ensure the following condition hold:

$$\exists M \quad \text{s.t.} \quad \mathbb{E}_{M^*}\left[z'|\mathcal{D}, a\right] = \mathbb{E}_M\left[z'|\phi(\mathcal{D}), a)\right] \quad \forall \mathcal{D}, a \tag{4.23}$$

where the LHS is the predicted mean of the next latent state under the true model, and the RHS is the predicted mean under the learned dynamics model. We call this the **EZP**, which stands for expected z prediction.[4]

A trivial way to minimize the (E)ZP loss is for the embedding to map everything to a constant vector, say $E(\mathcal{D}) = \mathbf{0}$, in which case z_{t+1} will be trivial for the dynamics model M to predict. However this is not a useful representation. This problem is **representational collapse** [Jin+22]. Fortunately, we can provably prevent collapse (at least for linear encoders) by using a frozen target network [Tan+23; Ni+24]. That is, we use the following auxiliary loss

$$\mathcal{L}_{\text{EZP}}(\phi, \theta; \mathcal{D}, a, \mathcal{D}') = ||M_\theta(E_\phi(\mathcal{D}, a)) - E_{\overline{\phi}}(\mathcal{D}')||_2^2 \tag{4.24}$$

where

$$\overline{\phi} = \rho\overline{\phi} + (1 - \rho)\text{sg}(\phi) \tag{4.25}$$

is the (stop-gradient version of) the EMA of the encoder weights. (If we set $\rho = 0$, this is called a detached network.)

---

[4]In [Ni+24], they also describe the ZP loss, which requires predicting the full distribution over z' using a stochastic transition model. This is strictly more powerful, but somewhat more complicated, so we omit it for simplicity.

We can also train the latent encoder to predict the reward. Formally, we want to ensure we can satisfy the following condition, which we call **RP** for "reward prediction":

$$\exists R \quad \text{s.t.} \quad \mathbb{E}_{R^*}[r|\mathcal{D}, a] = \mathbb{E}_R[r|\phi(\mathcal{D}), a)] \quad \forall \mathcal{D}, a \tag{4.26}$$

See Figure 4.3 for an illustration. In [Ni+24], they prove that a representation that satisfies ZP and RP is enough to satisfy value equivalence (sufficiency for Q^*).

Methods that optimize ZP and VP loss have been used in many papers, such as **Predictron** [Sil+17b], **Value Prediction Networks** [OSL17], **Self Predictive Representations** (SPR) [Sch+21], **Efficient Zero** (Section 4.1.3.3), **BYOL-Explore** (Section 4.3.2.6), etc.

4.3.2.3 Policy prediction

The value function and reward losses may be too sparse to learn efficiently. Although self-prediction loss can help somewhat, it does not use any extra information from the environment as feedback. Consequently it is natural to consider other kinds of prediction targets for learning the latent encoder (and dynamics). When using MCTS, it is possible compute what the policy should be for a given state, and this can be used as a prediction target for the reactive policy $a_t = \pi(\boldsymbol{z}_t)$, which in turn can be used as a feedback signal for the latent state. This method is used by MuZero (Section 4.1.3.2) and EfficientZero (Section 4.1.3.3).

4.3.2.4 Observation prediction

Another natural target to use for learning the encoder and dynamics is the next observation, using a one-step version of Equation (4.14). Indeed, [Ni+24] say that a representation ϕ satsifies the **OP** (observation prediction) criterion if it satisfies the following condition:

$$\exists D \quad \text{s.t.} \quad p^*(\boldsymbol{o}'|\mathcal{D}, a) = D(\boldsymbol{o}'|\phi(\mathcal{D}), a) \quad \forall \mathcal{D}, a \tag{4.27}$$

where D is the decoder. In order to repeatedly apply this, we need to be able to update the encoding $\boldsymbol{z} = \phi(\mathcal{D})$ in a recursive or online way. Thus we must also satisfify the following recurrent encoder condition, which [Ni+24] call **Rec**:

$$\exists U \quad \text{s.t.} \quad \phi(\mathcal{D}') = U(\phi(\mathcal{D}), a, \boldsymbol{o}') \quad \forall \mathcal{D}, a, \boldsymbol{o}' \tag{4.28}$$

where U is the update operator. Note that belief state updates (as in a POMDP) satisfy this property. Furthermore, belief states are a sufficient statistic to satisfy the OP condition. See Section 4.3.1.3 for a discussion of generative models of this form. However, there are other approaches to partial observability which work directly in prediction space (see Section 4.4.2).

4.3.2.5 Partial observation prediction

We have argued that predicting all the observations is problematic, but not predicting them is also problematic. A natural compromise is to predict some of the observations, or at least sone function of them. This is known as a **partial world model** (see e.g., [AP23]).

The best way to do this is an open research problem. A simple approach would be to predict all the observations, but put a penalty on the resulting OP loss term. A more sophisticated approach would be to structure the latent space so that we distinguish latent variables that are useful for learning Q^* (i.e., which affect the reward and which are affected by the agent's actions) from other latent variables that are needed to explain parts of the observation but otherwise are not useful. We can then impose an information bottleneck penalty on the latter, to prevent the agent focusing on irrelevant observational details. (See e.g., the **denoised MDP** method of [Wan+22].)
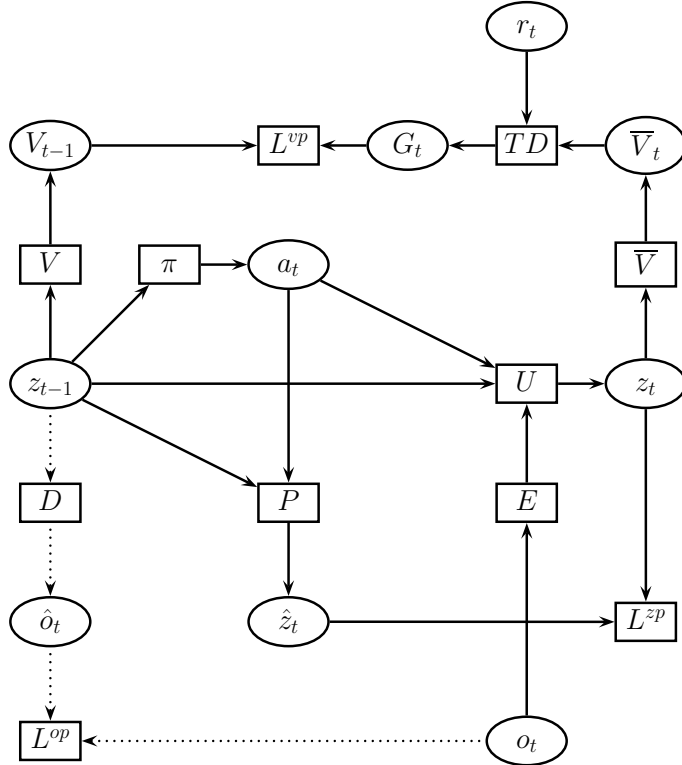
Figure 4.4: Illustration of (a simplified version of) the BYOL-Explore architecture, represented as a factor graph (so squares are functions, circles are variables). The dotted lines represent an optional observation prediction loss. The map from notation in this figure to the paper is as follows: $U \to h^c$ (closed-loop RNN update), $P \to h^o$ (open-loop RNN update), $D \to g$ (decoder), $E \to f$ (encoder). We have unrolled the forwards prediction for only 1 step. Also, we have omitted the reward prediction loss. The $\overline{V}$ node is the EMA version of the value function. The TD node is the TD operator.

4.3.2.6 BYOL-Explore

As an example of the above framework, consider the **BYOL-Explore** paper [Guo+22a], which uses a non-generative world model trained with ZP and VP loss. (BYOL stands for "build your own latent".) See Figure 4.4 for the computation graph, which we see is slightly simpler than the Dreamer computation graph in Figure 4.2 due to the lack of stochastic latents. In addition to using self-prediction loss to help train the latent representation, the error in this loss can be used to define an intrinsic reward, to encourage the agent to explore states where the model is uncertain. See Section 5.2.4 for further discussion of this topic.

4.4 Beyond one-step models: predictive representations

The "world models" we described in Section 4.3 are **one-step models** of the form $p(s'|s,a)$, or $p(z'|z,a)$ for $z = \phi(s)$, where ϕ is a state-abstraction function. However, such models are problematic when it comes to predicting many kinds of future events, such as "will a car pull in front of me?" or "when will it start raining?", since it is hard to predict exactly when these events will occur, and these events may correspond to many different "ground states". In principle we can roll out many possible long term futures, and apply some abstraction function to the resulting generated trajectories to extract features of interest, and thus derive a predictive model of the form $p(t', \phi(s_{t+1:t'})|s_t, \pi)$, where t' is the random duration of the sampled trajectory. However, it would be more efficient if we could directly predict this distribution without having to know the value of t', and without having to predict all the details of all the intermediate future states, many of which will be irrelevant given the abstraction function ϕ. This motivates the study of multi-step world models, that predict multiple steps into the future, either at the state level, or at the feature level. These are called **predictive representations**, and are a compromise between standard model-based RL and model-free RL, as we will see. Our presentation on this topic is based on [Car+24]. (See also Section 5.3, where we discuss the related topic of temporal abstraction from a model-free perspective.)

4.4.1 General value functions

The value function is based on predicting the sum of expected discounted future rewards. But the reward is just one possible signal of interest we can extract from the environment. We can generalize this by considering a **cumulant** $C_t \in \mathbb{R}$, which is some scalar of interest derived from the state or observation (e.g., did a loud bang just occur? is there a tree visible in the image?). We then define the **general value function** or **GVF** as follows [Sut95]:

$$V^{\pi,C,\gamma}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t C(s_{t+1})|s_0 = s, a_{0:\infty} \sim \pi\right] \tag{4.29}$$

If $C(s_{t+1}) = R_{t+1}$, this reduces to the value function.[5] However, we can also define the GVF to predict components of the observation vector; this is called **nexting** [MWS14], since it refers to next state prediction at different timescales.

4.4.2 Successor representations

In this section we consider a variant of GVF where the cumulant corresponds to a state occupancy vector $C(s_{t+1}) = \mathbb{I}(s_{t+1} = \tilde{s})$, which provides a dense feedback signal. This give us the **successor representation** or **SR** [Day93]:

$$M^{\pi}(s, \tilde{s}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}(s_{t+1} = \tilde{s})|S_0 = s\right] \tag{4.30}$$

---

[5]This follows the convention of [SB18], where we write $(s_t, a_t, r_{t+1}, s_{t+1})$ to represent the transitions, since r_{t+1} and s_{t+1} are both generated by applying a_t in state s_t.

If we define the policy-dependent state-transition matrix by

$$T^\pi(s, s') = \sum_a \pi(a|s) T(s'|s, a) \tag{4.31}$$

then the SR matrix can be rewritten as

$$\mathbf{M}^\pi = \sum_{t=0}^\infty \gamma^t [\mathbf{T}^\pi]^{t+1} = \mathbf{T}^\pi (\mathbf{I} - \gamma \mathbf{T}^\pi)^{-1} \tag{4.32}$$

Thus we see that the SR replaces information about individual transitions with their cumulants, just as the value function replaces individual rewards with the reward-to-go.

Like the value function, the SR obeys a Bellman equation

$$M^\pi(s, \tilde{s}) = \sum_a \pi(a|s) \sum_{s'} T(s'|s, a) \left(\mathbb{I}(s' = \tilde{s}) + \gamma M^\pi(s', \tilde{s}) \right) \tag{4.33}$$

$$= \mathbb{E} \left[\mathbb{I}(s' = \tilde{s}) + \gamma M^\pi(s', \tilde{s}) \right] \tag{4.34}$$

Hence we can learn an SR using a TD update of the form

$$M^\pi(s, \tilde{s}) \leftarrow M^\pi(s, \tilde{s}) + \eta \underbrace{\left(\mathbb{I}(s' = \tilde{s}) + \gamma M^\pi(s', \tilde{s}) - M^\pi(s, \tilde{s}) \right)}_{\delta} \tag{4.35}$$

where s' is the next state sampled from $T(s'|s, a)$. Compare this to the value-function TD update in Equation (2.16):

$$V^\pi(s) \leftarrow V^\pi(s) + \eta \underbrace{\left(R(s') + \gamma V^\pi(s') - V^\pi(s) \right)}_{\delta} \tag{4.36}$$

However, with an SR, we can easily compute the value function for any reward function (as approximated by a given policy) as follows:

$$V^{R,\pi} = \sum_{\tilde{s}} M^\pi(s, \tilde{s}) R(\tilde{s}) \tag{4.37}$$

See Figure 4.5 for an example.

We can also make a version of SR that depends on the action as well as the state to get

$$M^\pi(s, a, \tilde{s}) = \mathbb{E} \left[\sum_{t=0}^\infty \gamma^t \mathbb{I}(s_{t+1} = \tilde{s}) | s_0 = s, a_0 = a, a_{1:\infty} \sim \pi \right] \tag{4.38}$$

$$= \mathbb{E} \left[\mathbb{I}(s' = \tilde{s}) + \gamma M^\pi(s', a, \tilde{s}) | s_0 = s, a_0 = a, a_{1:\infty} \sim \pi \right] \tag{4.39}$$

This gives rise to a TD update of the form

$$M^\pi(s, a, \tilde{s}) \leftarrow M^\pi(s, a, \tilde{s}) + \eta \underbrace{\left(\mathbb{I}(s' = \tilde{s}) + \gamma M^\pi(s', a', \tilde{s}) - M^\pi(s, a, \tilde{s}) \right)}_{\delta} \tag{4.40}$$

where s' is the next state sampled from $T(s'|s, a)$ and a' is the next action sampled from $\pi(s')$. Compare this to the (on-policy) SARSA update from Equation (2.28):

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \eta \underbrace{\left(R(s') + \gamma Q^\pi(s', a') - Q^\pi(s, a) \right)}_{\delta} \tag{4.41}$$

However, from an SR, we can compute the state-action value function for any reward function:

$$Q^{R,\pi}(s, a) = \sum_{\tilde{s}} M^\pi(s, a, \tilde{s}) R(\tilde{s}) \tag{4.42}$$
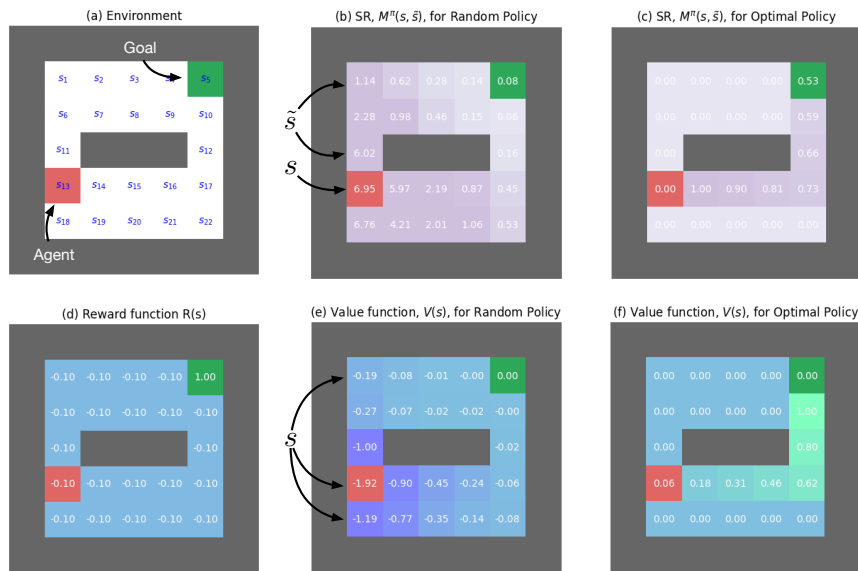
Figure 4.5: Illustration of successor representation for the 2d maze environment shown in (a) with reward shown in (d), which assings all states a reward of -0.1 except for the goal state which has a reward of 1.0. In (b-c) we show the SRs for a random policy and the optimal policy. In (e-f) we show the corresponding value functons. In (b), we see that the SR under the random policy assigns high state occupancy values to states which are close (in Manhattan distance) to the current state s_{13} (e.g., $M^\pi(s_{13}, s_{14}) = 5.97$) and low values to states that are further away (e.g., $M^\pi(s_{13}, s_{12}) = 0.16$). In (c), we see that the SR under the optimal policy assigns high state occupancy values to states which are close to the optimal path to the goal (e.g., $M^\pi(s_{13}, s_{14}) = 1.0$) and which fade with distance from the current state along that path (e.g., $M^\pi(s_{13}, s_{12}) = 0.66$). From Figure 3 of [Car+24]. Used with kind permission of Wilka Carvalho. Generated by https://github.com/wcarvalho/jaxneurorl/blob/main/successor_representation.ipynb.

90

This can be used to improve the policy as we discuss in Section 4.4.4.1.

We see that the SR representation has the computational advantages of model-free RL (no need to do explicit planning or rollouts in order to compute the optimal action), but also the flexibility of model-based RL (we can easily change the reward function without having to learn a new value function). This latter property makes SR particularly well suited to problems that use intrinsic reward (see Section 5.2.4), which often changes depending on the information state of the agent.

Unfortunately, the SR is limited in several ways: (1) it assumes a finite, discrete state space; (2) it depends on a given policy. We discuss ways to overcome limitation 1 in Section 4.4.3, and limitation 2 in Section 4.4.4.1.

4.4.3 Successor models

In this section, we discuss the **successor model** (also called a γ-model), which is a probabilistic extension of SR [JML20; Eys+21]. This allows us to generalize SR to work with continuous states and actions, and to simulate future state trajectories. The approach is to define the cumulant as the k-step conditional distribution $C(s_{k+1}) = P(s_{k+1} = \tilde{s}|s_0 = s, \pi)$, which is the probability of being in state $\tilde{s}$ after following π for k steps starting from state s. (Compare this to the SR cumulant, which is $C(s_{k+1}) = \mathbb{I}(s_{k+1} = \tilde{s})$.) The SM is then defined as

$$\boldsymbol{\mu}^\pi(\tilde{s}|s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_{t+1} = \tilde{s}|s_0 = s) \tag{4.43}$$

where the $1-\gamma$ term ensures that $\boldsymbol{\mu}^\pi$ integrates to 1. (Recall that $\sum_{t=0}^{\infty}\gamma^t = \frac{1}{1-\gamma}$ for $\gamma < 1$.) In the tabular setting, the SM is just the normalized SR, since

$$\boldsymbol{\mu}^\pi(\tilde{s}|s) = (1-\gamma)M^\pi(s,\tilde{s}) \tag{4.44}$$

$$= (1-\gamma)\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\mathbb{I}(s_{t+1} = \tilde{s})\,|s_0 = s, a_{0:\infty} \sim \pi\right] \tag{4.45}$$

$$= (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_{t+1} = \tilde{s}|s_0 = s, \pi) \tag{4.46}$$

Thus $\boldsymbol{\mu}^\pi(\tilde{s}|s)$ tells us the probability that $\tilde{s}$ can be reached from s within a horizon determined by γ when following π, even though we don't know exactly when we will reach $\tilde{s}$.

SMs obey a Bellman-like recursion

$$\boldsymbol{\mu}^\pi(\tilde{s}|s) = \mathbb{E}\left[(1-\gamma)T(\tilde{s}|s,a) + \gamma\boldsymbol{\mu}^\pi(\tilde{s}|s')\right] \tag{4.47}$$

We can use this to perform policy evaluation by computing

$$V^\pi(s) = \frac{1}{1-\gamma}\mathbb{E}_{\boldsymbol{\mu}^\pi(\tilde{s}|s)}\left[R(\tilde{s})\right] \tag{4.48}$$

We can also define an action-conditioned SM

$$\boldsymbol{\mu}^\pi(\tilde{s}|s,a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_{t+1} = \tilde{s}|s_0 = s, a_0 = a) \tag{4.49}$$

$$= (1-\gamma)T(\tilde{s}|s,a) + \gamma\mathbb{E}\left[\boldsymbol{\mu}^\pi(\tilde{s}|s',a',\pi)\right] \tag{4.50}$$

Hence we can learn an SM using a TD update of the form

$$\boldsymbol{\mu}^\pi(\tilde{s}|s,a) \leftarrow \boldsymbol{\mu}^\pi(\tilde{s}|s,a) + \eta\underbrace{\left((1-\gamma)T(s'|s,a) + \gamma\boldsymbol{\mu}^\pi(\tilde{s}|s',a') - \boldsymbol{\mu}^\pi(\tilde{s}|s,a)\right)}_{\delta} \tag{4.51}$$

where s' is the next state sampled from $T(s'|s, a)$ and a' is the next action sampled from $\pi(s')$. With an SM, we can compute the state-action value for any reward:

$$Q^{R,\pi}(s, a) = \frac{1}{1 - \gamma} \mathbb{E}_{\boldsymbol{\mu}^\pi(\tilde{s}|s,a)} [R(\tilde{s})] \tag{4.52}$$

This can be used to improve the policy as we discuss in Section 4.4.4.1.

4.4.3.1 Learning SMs

Although we can learn SMs using the TD update in Equation (4.51), this requires evaluating $T(s'|s, a)$ to compute the target update δ, and this one-step transition model is typically unknown. Instead, since $\boldsymbol{\mu}^\pi$ is a conditional density model, we will optimize the cross-entropy TD loss [JML20], defined as follows

$$\mathcal{L}_\mu = \mathbb{E}_{(s,a)\sim p(s,a), \tilde{s}\sim(T^\pi\boldsymbol{\mu}^\pi)(\cdot|s,a)} [\log \boldsymbol{\mu}_{\boldsymbol{\theta}}(\tilde{s}|s, a)] \tag{4.53}$$

where $(T^\pi\boldsymbol{\mu}^\pi)(\cdot|s, a)$ is the Bellman operator applied to $\boldsymbol{\mu}^\pi$ and then evaluated at (s, a), i.e.,

$$(T^\pi\boldsymbol{\mu}^\pi)(\tilde{s}|s, a) = (1 - \gamma)T(s'|s, a) + \gamma \sum_{s'} T(s'|s, a) \sum_{a'} \pi(a'|s'(\boldsymbol{\mu}^\pi(\tilde{s}|s', a') \tag{4.54}$$

We can sample from this as follows: first sample $s' \sim T(s'|s, a)$ from the environment and then with probability $1 - \gamma$ set $\tilde{s} = s'$ and terminate. Otherwise sample $a' \sim \pi(a'|s')$ and then create a bootstrap sample from the model using $\tilde{s} \sim \boldsymbol{\mu}^\pi(\tilde{s}|s', a')$.

There are many possible density models we can use for $\boldsymbol{\mu}^\pi$. In [Tha+22], they use a VAE. In [Tom+24], they use an autoregressive transformer applied to a set of discrete latent tokens, which are learned using VQ-VAE or a non-reconstructive self-supervised loss. They call their method **Video Occcupancy Models**.

An alternative approach to learning SMs, that avoids fitting a normalized density model over states, is to use contrastive learning to estimate how likely $\tilde{s}$ is to occur after some number of steps, given (s, a), compared to some randomly sampled negative state [ESL21; ZSE24]. Although we can't sample from the resulting learned model (we can only use it for evaluation), we can use it to improve a policy that achieves a target state (an approach known as goal-conditioned policy learning, discussed in Section 5.3.1).

4.4.3.2 Jumpy models using geometric policy composition

In [Tha+22], they propose **geometric policy composition** or GPC as a way to learn a new policy by sequencing together a set of N policies, as opposed to taking N primitive actions in a row. This can be thought of as a **jumpy model**, since it predicts multiple steps into the future, instead of one step at a time (c.f., [Zha+23a]).

In more detail, in GPC, the agent picks a sequence of n policies π_i for $i = 1 : n$, and then samples states according to their corresponding SMs: starting with (s_0, a_0), we sample $s_1 \sim \boldsymbol{\mu}_\gamma^{\pi_1}(\cdot|s_0, a_0)$, then $a_1 \sim \pi_1(\cdot|s_1)$, then $s_2 \sim \boldsymbol{\mu}_\gamma^{\pi_2}(\cdot|s_1, a_1)$, etc. This continues for $n - 1$ steps. Finally we sample $s_n \sim \boldsymbol{\mu}_{\gamma'}^{\pi_n}(\cdot|s_{n-1}, a_{n-1})$, where $\gamma' > \gamma$ represents a longer horizon SM. The reward estimates computed along this sampled path can then be combined to compute the value of each candidate policy sequence.

4.4.4 Successor features

Both SRs and SMs require defining expectations or distributions over the entire future state vector, which can be problematic in high dimensional spaces. In [Bar+17] they introduced **successor features**, that generalize SRs by working with features $\boldsymbol{\phi}(s)$ instead of primitive states. In particular, if we define the cumulant to be $C(s_{t+1}) = \boldsymbol{\phi}(s_{t+1})$, we get the following definition of SF:

$$\boldsymbol{\psi}^{\pi,\boldsymbol{\phi}}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(s_{t+1})|s_0 = s, a_{0:\infty} \sim \pi\right] \tag{4.55}$$

We will henceforth drop the ϕ superscript from the notation, for brevity. SFs obey a Bellman equation

$$\psi(s) = \mathbb{E}\left[\phi(s') + \gamma\psi(s')\right] \tag{4.56}$$

If we assume the reward function can be written as

$$R(s, \boldsymbol{w}) = \phi(s)^{\mathsf{T}}\boldsymbol{w} \tag{4.57}$$

then we can derive the value function for any reward as follows:

$$V^{\pi,\boldsymbol{w}}(s) = \mathbb{E}\left[R(s_1) + \gamma R(s_2) + \cdots | s_0 = s\right] \tag{4.58}$$

$$= \mathbb{E}\left[\phi(s_1)^{\mathsf{T}}\boldsymbol{w} + \gamma\phi(s_2)^{\mathsf{T}}\boldsymbol{w} + \cdots | s_0 = s\right] \tag{4.59}$$

$$= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\phi(s_{t+1})| s_0 = s\right]^{\mathsf{T}}\boldsymbol{w} = \psi^{\pi}(s)^{\mathsf{T}}\boldsymbol{w} \tag{4.60}$$

Similarly we can define an action-conditioned version of SF as

$$\psi^{\pi,\phi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty}\phi(s_{t+1})| s_0 = s, a_0 = a, a_{1:\infty} \sim \pi\right] \tag{4.61}$$

$$= \mathbb{E}\left[\phi(s') + \gamma\psi(s', a')\right] \tag{4.62}$$

We can learn this using a TD rule

$$\psi^{\pi}(s, a) \leftarrow \psi^{\pi}(s, a) + \eta\underbrace{\left(\phi(s') + \gamma\psi^{\pi}(s', a') - \psi^{\pi}(s, a)\right)}_{\delta} \tag{4.63}$$

And we can use it to derive a state-action value function:

$$Q^{\pi,\boldsymbol{w}}(s) = \psi^{\pi}(s, a)^{\mathsf{T}}\boldsymbol{w} \tag{4.64}$$

This allows us to define multiple Q functions (and hence policies) just by changing the weight vector $\boldsymbol{w}$, as we discuss in Section 4.4.4.1.

4.4.4.1 Generalized policy improvement

So far, we have discussed how to compute the value function for a new reward function but using the SFs from an existing known policy. In this section we discuss how to create a new policy that is better than an existing set of policies, by using **Generalized Policy Improvement** or **GPI** [Bar+17; Bar+20].

Suppose we have learned a set of N (potentially optimal) policies π_i and their corresponding SFs ψ^{π_i} for maximizing rewards defined by $\boldsymbol{w}_i$. When presented with a new task $\boldsymbol{w}_{\text{new}}$, we can compute a new policy using GPI as follows:

$$a^*(s; \boldsymbol{w}_{\text{new}}) = \underset{a}{\operatorname{argmax}}\max_i Q^{\pi_i}(s, a, \boldsymbol{w}_{\text{new}}) = \underset{a}{\operatorname{argmax}}\max_i \psi^{\pi_i}(s, a)^{\mathsf{T}}\boldsymbol{w}_{\text{new}} \tag{4.65}$$

If $\boldsymbol{w}_{\text{new}}$ is in the span of the training tasks (i.e., there exist weights α_i such that $\boldsymbol{w}_{\text{new}} \sum_i \alpha_i\boldsymbol{w}_i$), then the GPI theorem states that $\pi(a|s) = \mathbb{I}(a = a^*(s, \boldsymbol{w}_{\text{new}}))$ will perform at least as well as any of the existing policies, i.e., $Q^{\pi}(s, a) \geq \max_i Q^{\pi_i}(s, a)$ (c.f., policy improvement in Section 3.4). See Figure 4.6 for an illustration.

Note that GPI is a model-free approach to computing a new policy, based on an existing library of policies. In [Ale+23], they propose an extension that can also leverage a (possibly approximate) world model to learn better policies that can outperform the library of existing policies by performing more decision-time search.
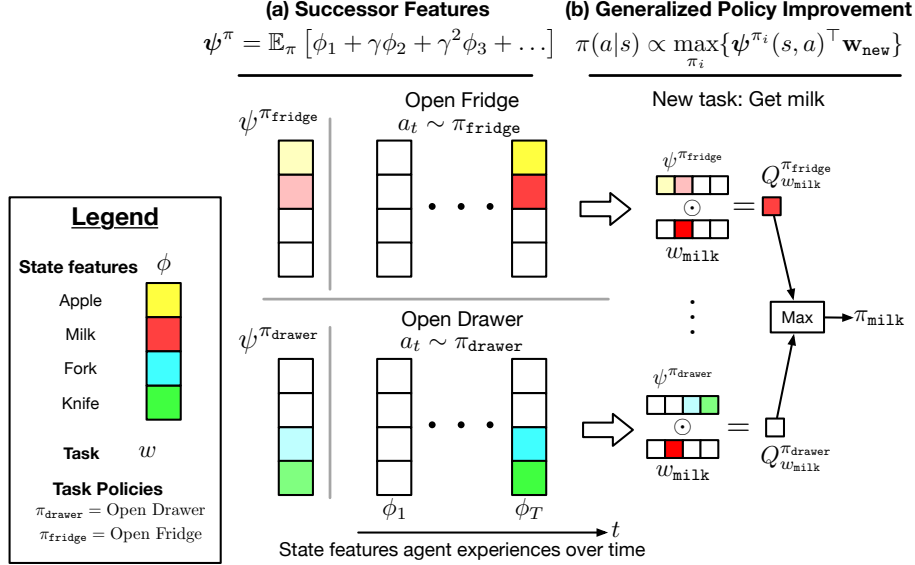
(a) Successor Features
$$\boldsymbol{\psi}^\pi = \mathbb{E}_\pi \left[\phi_1 + \gamma\phi_2 + \gamma^2\phi_3 + \ldots \right]$$

(b) Generalized Policy Improvement
$$\pi(a|s) \propto \max_{\pi_i}\{\boldsymbol{\psi}^{\pi_i}(s,a)^\top \mathbf{w}_{\texttt{new}}\}$$

Figure 4.6: Illustration of successor features representation. (a) Here $\boldsymbol{\phi}_t = \boldsymbol{\phi}(s_t)$ is the vector of features for the state at time t, and $\boldsymbol{\psi}^\pi$ is the corresponding SF representation, which depends on the policy π. (b) Given a set of existing policies and their SFs, we can create a new one by specifying a desired weight vector $\boldsymbol{w}_{new}$ and taking a weighted combination of the existing SFs. From Figure 5 of [Car+24]. Used with kind permission of Wilka Carvalho.

4.4.4.2   Option keyboard

One limitation of GPI is that it requires that the reward function, and the resulting policy, be defined in terms of a fixed weight vector $\boldsymbol{w}_{\text{new}}$, where the preference over features is constant over time. However, for some tasks we might want to initially avoid a feature or state and then later move towards it. To solve this, [Bar+19; Bar+20] introduced the **option keyboard**, in which the weight vector for a task can be computed dynamically in a state-dependent way, using $\boldsymbol{w}_s = g(s, \boldsymbol{w}_{\text{new}})$. (Options are discussed in Section 5.3.2.) Actions can then be chosen as follows:

$$a^*(s; \boldsymbol{w}_{\text{new}}) = \operatorname*{argmax}_a \max_i \boldsymbol{\psi}^{\pi_i}(s,a)^\top \boldsymbol{w}_s \tag{4.66}$$

Thus the policy π_i that is chosen depends in the current state. Thus $\boldsymbol{w}_s$ induces a set of policies that are active for a period of time, similar to playing a chord on a piano.

4.4.4.3   Learning SFs

A key question when using SFs is how to learn the cumulants or state-features $\boldsymbol{\phi}(s)$. Various approaches have been suggested, including leveraging meta-gradients [Vee+19], image reconstruction [Mac+18b], and maximizing the mutual information between task encodings and the cumulants that an agent experiences when pursuing that task [Han+19]. The cumulants are encouraged to satisfies the linear reward constraint by minimizing

$$\mathcal{L}_r = ||r - \boldsymbol{\phi}_{\boldsymbol{\theta}}(s)^\top \boldsymbol{w}||_2^2 \tag{4.67}$$

Once the cumulant function is known, we have to learn the corresponding SF. The standard approach learns a different SF for every policy, which is limiting. In [Bor+19] they introduced **Universal Successor Feature Approximators** which takes an input a policy encoding $\boldsymbol{z_w}$, representing a policy $\pi_{\boldsymbol{w}}$ (typically we set $\boldsymbol{z_w} = \boldsymbol{w}$). We then define

$$\boldsymbol{\psi}^{\pi_{\boldsymbol{w}}}(s,a) = \boldsymbol{\psi}_{\boldsymbol{\theta}}(s,a,\boldsymbol{z_w}) \tag{4.68}$$

The GPI update then becomes

$$a^*(s; \boldsymbol{w}_{\text{new}}) = \operatorname*{argmax}_a \max_{\boldsymbol{z_w}} \boldsymbol{\psi_\theta}(s, a, \boldsymbol{z_w})^\mathsf{T} \boldsymbol{w}_{\text{new}} \tag{4.69}$$

so we replace the discrete max over a finite number of policies with a continuous optimization problem (to be solved per state).

If we want to learn the policies and SFs at the same time, we can optimize the following losses in parallel:

$$\mathcal{L}_Q = ||\boldsymbol{\psi_\theta}(s, a, \boldsymbol{z}_w)^\mathsf{T} \boldsymbol{w} - \boldsymbol{y}_Q||, \ \boldsymbol{y}_Q = R(s'; \boldsymbol{w}) + \gamma \boldsymbol{\psi_\theta}(s', a^*, \boldsymbol{z}_w)^\mathsf{T} \boldsymbol{w} \tag{4.70}$$

$$\mathcal{L}_\psi = ||\boldsymbol{\psi_\theta}(s, a, \boldsymbol{z_w}) - \boldsymbol{y}_\psi||, \ \boldsymbol{y}_\psi = \boldsymbol{\phi}(s') + \gamma \boldsymbol{\psi_\theta}(s', a^*, \boldsymbol{z_w}) \tag{4.71}$$

where $a^* = \operatorname{argmax}_{a'} \boldsymbol{\psi_\theta}(s', a', \boldsymbol{z_w})^\mathsf{T} \boldsymbol{w}$. The first equation is standard Q learning loss, and the second is the TD update rule in Equation (4.63) for the SF. In [Car+23], they present the **Successor Features Keyboard**, that can learn the policy, the SFs and the task encoding $\boldsymbol{z_w}$, all simultaneously. They also suggest replacing the squared error regression loss in Equation (4.70) with a cross-entropy loss, where each dimension of the SF is now a discrete probability distribution over M possible values of the corresponding feature. (c.f. Section 5.1.2).

4.4.4.4 Choosing the tasks

A key advantage of SFs is that they provide a way to compute a value function and policy for any given reward, as specified by a task-specific weight vector $\boldsymbol{w}$. But how do we choose these tasks? In [Han+19] they sample $\boldsymbol{w}$ from a distribution at the start of each task, to encourage the agent to learn to explore different parts of the state space (as specified by the feature function $\boldsymbol{\phi}$). In [LA21] they extend this by adding an intrinsic reward that favors exploring parts of the state space that are surprising (i.e., which induce high entropy), c.f., Section 5.2.4. In [Far+23], they introduce **proto-value networks**, which is a way to define auxiliary tasks based on successor measures.

Chapter 5

Other topics in RL

In this section, we briefly mention some other important topics in RL.

5.1 Distributional RL

The **distributional RL** approach of [BDM17; BDR23], predicts the distribution of (discounted) returns, not just the expected return. More precisely, let $Z^\pi = \sum_{t=0}^{T} \gamma^t r_t$ be a random variable representing the reward-to-go. The standard value function is defined to compute the expectation of this variable: $V^\pi(s) = \mathbb{E}\left[Z^\pi | s_0 = s\right]$. In DRL, we instead attempt to learn the full distribution, $p(Z^\pi | s_0 = s)$. For a general review of distributional regression, see [KSS23]. Below we briefly mention a few algorithms in this class that have been explored in the context of RL.

5.1.1 Quantile regression methods

An alternative to predicting a full distribution is to predict a fixed set of quantiles. This is called quantile regression, and has been used with DQN in [Dab+17] to get **QR-DQN**, and with SAC in [Wur+22] to get **QR-SAC**. (The latter was used in Sony's **GTSophy** Gran Turismo AI racing agent.)

5.1.2 Replacing regression with classification

An alternative to quantile regression is to approximate the distribution over returns using a histogram, and then fit it using cross entropy loss (see Figure 5.1). This approach was first suggested in [BDM17], who called it **categorical DQN**. (In their paper, they use 51 discrete categories (atoms), giving rise to the name **C51**.)
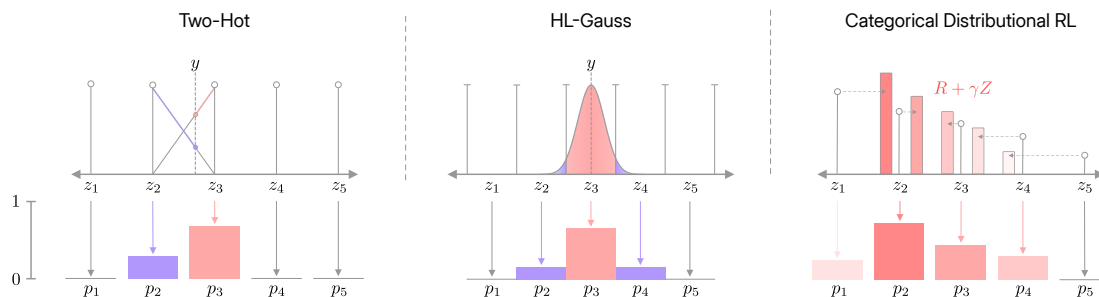
Figure 5.1: Illustration of how to encode a scalar target y or distributional target Z using a categorical distribution. From Figure 1 of [Far+24]. Used with kind permission of Jesse Farebrother.

An even simpler approach is to replace the distributional target with the standard scalar target (representing the mean), and then discretize this target and use cross entropy loss instead of squared error.[1] Unfortunately, this encoding is lossy. In [Sch+20], they proposed the **two-hot** transform, that is a lossless encoding of the target based on putting appropriate weight on the nearest two bins (see Figure 5.1). In [IW18], they proposed the **HL-Gauss** histogram loss, that convolves the target value y with a Gaussian, and then discretizes the resulting continuous distribution. This is more symetric than two-hot encoding, as shown in Figure 5.1. Regardless of how the discrete target is chosen, predictions are made using $\hat{y}(s; \boldsymbol{\theta}) = \sum_k p_k(s) b_k$, where $p_k(s)$ is the probability of bin k, and b_k is the bin center.

In [Far+24], they show that the HL-Gauss trick works much better than MSE, two-hot and C51 across a variety of problems (both offline and online), especially when they scale to large networks. They conjecture that the reason it beats MSE is that cross entropy is more robust to noisy targets (e.g., due to stochasticity) and nonstationary targets. They also conjecture that the reason HL works better than two-hot is that HL is closer to ordinal regression, and reduces overfitting by having a softer (more entropic) target distribution (similiar to label smoothing in classification problems).

5.2   Reward functions

Sequential decision making relies on the user to define the reward function in order to encourage the agent to exhibit some desired behavior. In this section, we discuss this crucial aspect of the problem.

5.2.1   Reward hacking

In some cases, the reward function may be misspecified, so even though the agent may maximize the reward, this might turn out not to be what the user desired. For example, suppose the user rewards the agent for making as many paper clips as possible. An optimal agent may convert the whole world into a paper clip factory, because the user forgot to specify various constraints, such as not killing people or not destroying the environment. In the **AI alignment** community, this example is known as the **paperclip maximizer problem**, and is due to Nick Bostrom [Bos16]. (See e.g., https://openai.com/index/faulty-reward-functions/ for some examples that have occurred in practice.) This is an example of a more general problem known as **reward hacking** [Ska+22]. For a potential solution, based on the assistance game paradigm, see Section 5.6.1.2.

5.2.2   Sparse reward

Even if the reward function is correct, optimizing it is not always easy. In particular, many problems suffer from **sparse reward**, in which $R(s, a) = 0$ for almost all states and actions, so the agent only every gets feedback (either positive or negative) on the rare occasions when it achieves some unknown goal. This requires **deep exploration** [Osb+19] to find the rewarding states. One approach to this is use to use PSRL (Section 1.4.4.2). However, various other heuristics have been developed, some of which we discuss below.

5.2.3   Reward shaping

In **reward shaping**, we add prior knowledge about what we believe good states should look like, as a way to combat the difficulties of learning from sparse reward. That is, we define a new reward function $r' = r + F$, where F is called the shaping function. In general, this can affect the optimal policy. For example, if a soccer playing agent is "artificially" rewarded for making contact with the ball, it might learn to repeatedly touch and untouch the ball (toggling between s and s'), rather than trying to win the original game. But in [NHR99], the prove that if the shaping function has the form

$$F(s, a, s') = \gamma \Phi(s') - \Phi(s) \tag{5.1}$$

---

[1]Technically speaking, this is no longer a distributional RL method, since the prediction target is the mean, but the mechanism for predicting the mean leverages a distribution, for robustness and ease of optimization.

where $\Phi : \mathcal{S} \to \mathbb{R}$ is a **potential function**, then we can guarantee that the sum of shaped rewards will match the sum of original rewards plus a constant. This is called **Potential-Based Reward Shaping**.

In [Wie03], they prove that (in the tabular case) this approach is equivalent to initializing the value function to $V(s) = \Phi(s)$. In [TMM19], they propose an extension called potential-based advice, where they show that a potential of the form $F(s, a, s', a') = \gamma \Phi(s', a') - \Phi(s, a)$ is also valid (and more expressive). In [Hu+20], they introduce a reward shaping function z which can be used to down-weight or up-weight the shaping function:

$$r'(s, a) = r(s, a) + z_\phi(s, a) F(s, a) \tag{5.2}$$

They use bilevel optimization to optimize ϕ wrt the original task performance.

5.2.4 Intrinsic reward

When the extrinsic reward is sparse, it can be useful to (also) reward the agent for solving "generally useful" tasks, such as learning about the world. This is called **intrinsically motivated RL** [AMH23; Lin+19; Ami+21; Yua22; Yua+24; Col+22]. It can be thought of as a special case of reward shaping, where the shaping function is dynamically computed.

We can classify these methods into two main types: **knowledge-based intrinsic motivation**, or **artificial curiosity**, where the agent is rewarded for learning about its environment; and **competence-based intrinsic motivation**, where the agent is rewarded for achieving novel goals or mastering new skills.

5.2.4.1 Knowledge-based intrinsic motivation

One simple approach to knowledge-based intrinsic motivation is to add to the extrinsic reward an intrinsic **exploration bonus** $R_t^i(s_t)$, which is high when the agent visits novel states. For tabular environments, we can just count the number of visits to each state, $N_t(s)$, and define $R_t^i(s) = 1/N_t(s)$ or $R_t^i(s) = 1/\sqrt{N_t(s)}$, which is similar to the UCB heuristic used in bandits (see Section 1.4.3). We can extend exploration bonuses to high dimensional states (e.g. images) using density models [Bel+16]. Alternatively, [MBB20] propose to use the ℓ_1 norm of the successor feature (Section 4.4.4) representation as an alternative to the visitation count, giving rise to an intrinsic reward of the form $R^i(s) = 1/||\boldsymbol{\psi}^\pi(s)||_1$. Recently [Yu+23] extended this to combine SFs with *predecessor* representations, which encode retrospective information about the previous state (c.f., inverse dynamics models, mentioned below). This encourages exploration towards bottleneck states.

Another approach is the **Random Network Distillation** or **RND** method of [Bur+18]. This uses a fixed random neural network feature extractor $\boldsymbol{z}_t = f(\boldsymbol{s}_t; \boldsymbol{\theta}^*)$ to define a target, and then trains a predictor $\hat{\boldsymbol{z}}_t = f(\boldsymbol{s}_t; \hat{\boldsymbol{\theta}}_t)$ to predict these targets. If $\boldsymbol{s}_t$ is similar to previously seen states, then the trained model will have low prediction error. We can thus define the intrinsic reward as proportional to the squared error $||\hat{\boldsymbol{z}}_t - \boldsymbol{z}_t||_2^2$. The **BYOL-Explore** method of [Guo+22b] goes beyond RND by learning the target representation (for the next state), rather than using a fixed random projection, but is still based on prediction error.

We can also define an intrinsic reward in terms of the information theoretic **surprise** of the next state given the current one:

$$R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') = -\log q(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) \tag{5.3}$$

This is the same as methods based on rewarding states for prediction error. Unfortunately such methods can suffer from the **noisy TV problem** (also called a **stochastic trap**), in which an agent is attracted to states which are intrinsically to predict. To see this, note that by averaging over future states we see that the above reward reduces to

$$R(\boldsymbol{s}, \boldsymbol{a}) = -\mathbb{E}_{p^*(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})} \left[\log q(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) \right] = \mathbb{H}_{ce}(p^*, q) \tag{5.4}$$

where p^* is the true model and q is the learned dynamics model, and $\mathbb{H}_{ce}$ is the cros -entropy. As we learn the optimal model, $q = p^*$, this reduces to the conditional entropy of the predictive distribution, which can be non-zero for inherently unpredictable states.

To help filter out such random noise, [Pat+17] proposes an **Intrinsic Curiosity Module**. This first learns an **inverse dynamics model** of the form $a = f(s, s')$, which tries to predict which action was used, given that the agent was in s and is now in s'. The classifier has the form $\text{softmax}(g(\phi(s), \phi(s'), a))$, where $z = \phi(s)$ is a representation function that focuses on parts of the state that the agent can control. Then the agent learns a forwards dynamics model in z-space. Finally it defines the intrinsic reward as

$$R(s, a, s') = -\log q(\phi(s')|\phi(s), a) \tag{5.5}$$

Thus the agent is rewarded for visiting states that lead to unpredictable consequences, where the difference in outcomes is measured in a (hopefully more meaningful) latent space.

Another solution is to replace the cross entropy with the KL divergence, $R(s, a) = D_{\mathbb{KL}}(p||q) = \mathbb{H}_{ce}(p, q) - \mathbb{H}(p)$, which goes to zero once the learned model matches the true model, even for unpredictable states. This has the desired effect of encouraging exploration towards states which have epistemic uncertainty (reducible noise) but not aleatoric uncertainty (irreducible noise) [MP+22]. The **BYOL-Hindsight** method of [Jar+23] is one recent approach that attempts to use the $R(s, a) = D_{\mathbb{KL}}(p||q)$ objective. Unfortunately, computing the $D_{\mathbb{KL}}(p||q)$ term is much harder than the usual variational objective of $D_{\mathbb{KL}}(q||p)$. A related idea, proposed in the RL context by [Sch10], is to use the **information gain** as a reward. This is defined as $R_t(s_t, a_t) = D_{\mathbb{KL}}(q(s_t|h_t, a_t, \theta_t)||q(s_t|h_t, a_t, \theta_{t-1}))$, where h_t is the history of past observations, and $\theta_t = \text{update}(\theta_{t-1}, h_t, a_t, s_t)$ are the new model parameters. This is closely related to the BALD (Bayesian Active Learning by Disagreement) criterion [Hou+11; KAG19], and has the advantage of being easier to compute, since it is does not reference the true distribution p.

5.2.4.2 Goal-based intrinsic motivation

We will discuss goal-conditioned RL in Section 5.3.1. If the agent creates its own goals, then it provides a way to explore the environment. The question of when and how an agent to switch to pursuing a new goal is studied in [Pis+22] (see also [BS23]). Some other key work in this space includes the **scheduled auxiliary control** method of [Rie+18], and the **Go Explore** algorithm in [Eco+19; Eco+21] and its recent LLM extension [LHC24].

5.3 Hierarchical RL

So far we have focused on MDPs that work at a single time scale. However, this is very limiting. For example, imagine planning a trip from San Francisco to New York: we need to choose high level actions first, such as which airline to fly, and then medium level actions, such as how to get to the airport, followed by low level actions, such as motor commands. Thus we need to consider actions that operate multiple levels of **temporal abstraction**. This is called **hierarchical RL** or **HRL**. This is a big and important topic, and we only brief mention a few key ideas and methods. Our summary is based in part on [Pat+22]. (See also Section 4.4 where we discuss multi-step predictive models; by contrast, in this section we focus on model-free methods.)

5.3.1 Feudal (goal-conditioned) HRL

In this section, we discuss an approach to HRL known as **feudal RL** [DH92]. Here the action space of the higher level policy consists of **subgoals** that are passed down to the lower level policy. See Figure 5.2 for an illustration. The lower level policy learns a **universal policy** $\pi(a|s, g)$, where g is the goal passed into it [Sch+15a]. This policy optimizes an MDP in which the reward is define as $R(s, a|g) = 1$ iff the goal state is achieved, i.e., $R(s, a|s) = \mathbb{I}(s = g)$. (We can also define a dense reward signal using some state abstraction function ϕ, by defining $R(s, a|g) = \text{sim}(\phi(s), \phi(g))$ for some similarity metric.) This approach to RL is known as **goal-conditioned RL** [LZZ22].
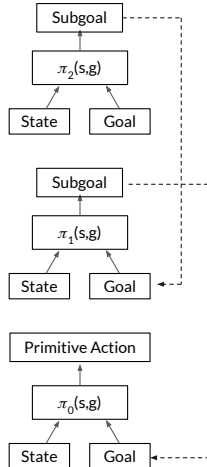
Figure 5.2: *Illustration of a 3 level hierarchical goal-conditioned controller. From* `http://bigai.cs.brown.edu/2019/09/03/hac.html`. *Used with kind permission of Andrew Levy.*

5.3.1.1 Hindsight Experience Relabeling (HER)

In this section, we discuss an approach to efficiently learning goal-conditioned policies, in the special case where the set of goal states $\mathcal{G}$ is the same as the set of original states $\mathcal{S}$. We will extend this to the hierarchical case below.

The basic idea is as follows. We collect various trajectores in the environment, from a starting state s_0 to some terminal state s_T, and then define the goal of each trajectory as being $g = s_T$; this trajectory then serves as a demonstration of how to achieve this goal. This is called **hindsight experience relabeling** or **HER** [And+17]. This can be used to relabel the trajectories stored in the replay buffer. That is, if we have $(s, a, R(s|g), s', g)$ tuples, we replace them with $(s, a, R(s|g'), g')$ where $g' = s_T$. We can then use any off-policy RL method to learn $\pi(a|s, g)$. In [Eys+20], they show that HER can be viewed as a special case of maximum-entropy inverse RL, since it is estimating the reward for which the corresponding trajectory was optimal.

5.3.1.2 Hierarchical HER

We can leverage HER to learn a hierarchical controller in several ways. In [Nac+18] they propose **HIRO** (Hierarchical Reinforcement Learning with Off-policy Correction) as a way to train a two-level controller. (For a two-level controller, the top level is often called the **manager**, and the low level the **worker**.) The data for the manager are transition tuples of the form $(s_t, g_t, \sum r_{t:t+c}, s_{t+c})$, where c is the time taken for the worker to reach the goal (or some maximum time), and r_t is the main task reward function at step t. The data for the worker are transition tuples of the form $(s_{t+i}, g_t, a_{t+i}, r_{t+i}^{g_t}, s_{t+i+1})$ for $i = 0 : c$, where r_t^g is the reward wrt reaching goal g. This data can be used to train the two policies. However, if the worker fails to achieve the goal in the given time limit, all the rewards will be 0, and no learning will take place. To combat this, if the worker does not achieve g_t after c timesteps, the subgoal is relabeled in the transition data with another subgoal g_t' which is sampled from $p(g|\tau)$, where τ is the observed trajectory. Thus both policies treat g_t' as the goal in hindsight, so they can use the actually collected data for training

The **hierarchical actor critic** (HAC) method of [Lev+18] is a simpler version of HIRO that can be extended to multiple levels of hierarchy, where the lowest level corresponds to primitive actions (see Figure 5.2). In the HAC approach, the output subgoal in the higher level data, and the input subgoal in the lower-level data, are replaced with the actual state that was achieved in hindsight. This allows the training of each level of the hierarchy independently of the lower levels, by assuming the lower level policies are already optimal (since they achieved the specified goal). As a result, the distribution of (s, a, s') tuples experienced by a higher level

will be stable, providing a stationary learning target. By contrast, if all policies are learned simultaneously, the distribution becomes **non-stationary**, which makes learning harder. For more details, see the paper, or the corresponding blog post (with animations) at `http://bigai.cs.brown.edu/2019/09/03/hac.html`.

5.3.1.3 Learning the subgoal space

In the previous approaches, the subgoals are defined in terms of the states that were achieved at the end of each trajectory, $g' = s_T$. This can be generalized by using a state abstraction function to get $g' = \phi(s_T)$. The methods in Section 5.3.1.2) assumed that ϕ was manually specified. We now mention some ways to learn ϕ.

In [Vez+17], they present **Feudal Networks** for learning a two level hierarchy. The manager samples subgoals in a learned latent subgoal space. The worker uses distance to this subgoal as a reward, and is trained in the usual way. The manager uses the "transition gradient" as a reward, which is derived from the task reward as well as the distance between the subgoal and the actual state transition made by the worker. This reward signal is used to learn the manager policy and the latent subgoal space.

Feudal networks do not guarantee that the learned subgoal space will result in optimal behavior. In [Nac+19], they present a method to optimize the policy and ϕ function so as to minimize a bound on the suboptimality of the hierarchical policy. This approach is combined with HIRO (Section 5.3.1.2) to tackle the non-stationarity issue.

5.3.2 Options

The feudal approach to HRL is somewhat limited, since not all subroutines or skills can be defined in terms of reaching a goal state (even if it is a partially specified one, such as being in a desired location but without specifying the velocity). For example, consider the skill of "driving in a circle", or "finding food". The **options** framework is a more general framework for HRL first proposed in [SPS99]. We discuss this below.

5.3.2.1 Definitions

An option $\omega = (I, \pi, \beta)$ is a tuple consisting of: the **initiation set** $I_\omega \subset S$, which is a subset of states that this option can start from (also called the **affordances** of each state [Khe+20]); the **subpolicy** $\pi_\omega(a|s) \in [0, 1]$; and the **termination condition** $\beta_\omega(s) \in [0, 1]$, which gives the probability of finishing in state s. (This induces a geometric distribution over option durations, which we denote by $\tau \sim \beta_\omega$.) The set of all options is denoted Ω.

To execute an option at step t entails choosing an action using $a_t = \pi_\omega(s_t)$ and then deciding whether to terminate at step $t + 1$ with probability $1 - \beta_\omega(s_{t+1})$ or to continue following the option at step $t + 1$. (This is an example of a **semi-Markov decision process** [Put94].) If we define $\pi_\omega(s) = a$ and $\beta_\omega(s) = 0$ for all s, then this option corresponds to primitive action a that terminates in one step. But with options we can expand the repertoire of actions to include those that take many steps to finish.

To create an MDP with options, we need to define the reward function and dynamics model. The reward is defined as follows:

$$R(s, \omega) = \mathbb{E}\left[R_1 + \gamma R^2 + \cdots + \gamma^{\tau-1} R_\tau | S_0 = s, A_{0:\tau-1} \sim \pi_\omega, \tau \sim \beta_\omega\right] \tag{5.6}$$

The dynamics model is defined as follows:

$$p_\gamma(s'|s, \omega) = \sum_{k=1}^{\infty} \gamma^k \Pr\left(S_k = s', \tau = k | S_0 = s, A_{0:k-1} \sim \pi_\omega, \tau \sim \beta_\omega\right) \tag{5.7}$$

Note that $p_\gamma(s'|s, \omega)$ is not a conditional probability distribution, because of the γ^k term, but we can usually treat it like one. Note also that a dynamics model that can predict multiple steps ahead is sometimes called a **jumpy model** (see also Section 4.4.3.2).

We can use these definitions to define the value function for a hierarchical policy using a generalized Bellman equation, as follows:

$$V_\pi(s) = \sum_{\omega \in \Omega(s)} \pi(\omega|s) \left[R(s, \omega) + \sum_{s'} p_\gamma(s'|s, \omega) V_\pi(s') \right] \tag{5.8}$$

We can compute this using value iteration. We can then learn a policy using policy iteration, or a policy gradient method. In other words, once we have defined the options, we can use all the standard RL machinery.

Note that GCRL can be considered a special case of options where each option corresponds to a different goal. Thus the reward function has the form $R(s, \omega) = \mathbb{I}(s = \omega)$, the termination function is $\beta_\omega(s) = \mathbb{I}(s = \omega)$, and the initiation set is the entire state space.

5.3.2.2 Learning options

The early work on options, including the **MAXQ** approach of [Die00], assumed that the set of options was manually specified. Since then, many methods for learning options have been proposed. We mention a few of these below.

The first set of methods for option learning rely on two stage training. In the first stage, exploration methods are used to collect trajectories. Then this data is analysed, either by inferring hidden segments using EM applied to a latent variable model [Dan+16], or by using the **skill chaining** method of [KB09], which uses classifiers to segment the trajectories. The labeled data can then be used to define a set of options, which can be trained using standard methods.

The second set of methods for option learning use end-to-end training, i.e., the options and their policies are jointly learned online. For example, [BHP17] propose the **option-critic** architecture. The number of options is manually specified, and all policies are randomly initialized. Then they are jointly trained using policy gradient methods designed for semi-MDPs. (See also [RLT18] for a hierarchical extension of option-critic to support options calling options.) However, since the learning signal is just the main task reward, the method can work poorly in problems with sparse reward compared to subgoal methods (see discussion in [Vez+17; Nac+19]).

Another problem with option-critic is that it requires specialized methods that are designed for optimizing semi-MDPs. In [ZW19], they propose **double actor critic**, which allows the use of standard policy gradient methods. This works by defining two parallel **augmented MDPs**, where the state space of each MDP is the cross-product of the original state space and the set of options. The manager learns a policy over options, and the worker learns a policy over states for each option. Both MDPs just use task rewards, without subgoals or subtask rewards.

It has been observed that option learning using option-critic or double actor-critic can fail, in the sense that the top level controller may learn to switch from one option to the next at almost every time step [ZW19; Har+18]. The reason is that the optimal policy does not require the use of temporally extended options, but instead can be defined in terms of primitive actions (as in standard RL). Therefore in [Har+18] they propose to add a regularizer called the **deliberation cost**, in which the higher level policy is penalized whenever it switches options. This can speed up learning, at the cost of a potentially suboptimal policy.

Another possible failure mode in option learning is if the higher level policy selects a single option for the entire task duration. To combat this, [KP19] propose the **Interest Option Critic**, which learns the initiation condition I_ω so that the option is selected only in certain states of interest, rather than the entire state space.

In [Mac+23], they discuss how the successor representation (discussed in Section 4.4) can be used to define options, using a method they call the **Representation-driven Option Discovery** (ROD) cycle.

In [Lin+24b] they propose to represent options as programs, which are learned using LLMs.

5.4   Imitation learning

In previous sections, an RL agent is to learn an optimal sequential decision making policy so that the total reward is maximized. **Imitation learning** (IL), also known as **apprenticeship learning** and **learning from demonstration** (LfD), is a different setting, in which the agent does not observe rewards, but has access to a collection $\mathcal{D}_{\exp}$ of trajectories generated by an expert policy $\pi_{\exp}$; that is, $\boldsymbol{\tau} = (s_0, a_0, s_1, a_1, \ldots, s_T)$ and $a_t \sim \pi_{\exp}(s_t)$ for $\boldsymbol{\tau} \in \mathcal{D}_{\exp}$. The goal is to learn a good policy by imitating the expert, in the absence of reward signals. IL finds many applications in scenarios where we have demonstrations of experts (often humans) but designing a good reward function is not easy, such as car driving and conversational systems. (See also Section 5.5, where we discuss the closely related topic of offline RL, where we also learn from a collection of trajectories, but no longer assume they are generated by an optimal policy.)

5.4.1   Imitation learning by behavior cloning

A natural method is **behavior cloning**, which reduces IL to supervised learning; see [Pom89] for an early application to autonomous driving. It interprets a policy as a classifier that maps states (inputs) to actions (labels), and finds a policy by minimizing the imitation error, such as

$$\min_{\pi} \mathbb{E}_{p^{\gamma}_{\pi_{\exp}}(s)} \left[D_{\mathbb{KL}} \left(\pi_{\exp}(s) \parallel \pi(s) \right) \right] \tag{5.9}$$

where the expectation wrt $p^{\gamma}_{\pi_{\exp}}$ may be approximated by averaging over states in $\mathcal{D}_{\exp}$. A challenge with this method is that the loss does not consider the sequential nature of IL: future state distribution is not fixed but instead depends on earlier actions. Therefore, if we learn a policy $\hat{\pi}$ that has a low imitation error under distribution $p^{\gamma}_{\pi_{\exp}}$, as defined in Equation (5.9), it may still incur a large error under distribution $p^{\gamma}_{\hat{\pi}}$ (when the policy $\hat{\pi}$ is actually run). This problem has been tackled by the offline RL literature, which we discuss in Section 5.5.

5.4.2   Imitation learning by inverse reinforcement learning

An effective approach to IL is **inverse reinforcement learning** (IRL) or **inverse optimal control** (IOC). Here, we first infer a reward function that "explains" the observed expert trajectories, and then compute a (near-)optimal policy against this learned reward using any standard RL algorithms studied in earlier sections. The key step of reward learning (from expert trajectories) is the opposite of standard RL, thus called inverse RL [NR00].

It is clear that there are infinitely many reward functions for which the expert policy is optimal, for example by several optimality-preserving transformations [NHR99]. To address this challenge, we can follow the maximum entropy principle, and use an energy-based probability model to capture how expert trajectories are generated [Zie+08]:

$$p(\boldsymbol{\tau}) \propto \exp \Big(\sum_{t=0}^{T-1} R_{\boldsymbol{\theta}}(s_t, a_t) \Big) \tag{5.10}$$

where $R_{\boldsymbol{\theta}}$ is an unknown reward function with parameter $\boldsymbol{\theta}$. Abusing notation slightly, we denote by $R_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \sum_{t=0}^{T-1} R_{\boldsymbol{\theta}}(s_t, a_t))$ the cumulative reward along the trajectory $\boldsymbol{\tau}$. This model assigns exponentially small probabilities to trajectories with lower cumulative rewards. The partition function, $Z_{\boldsymbol{\theta}} \triangleq \int_{\boldsymbol{\tau}} \exp(R_{\boldsymbol{\theta}}(\boldsymbol{\tau}))$, is in general intractable to compute, and must be approximated. Here, we can take a sample-based approach. Let $\mathcal{D}_{\exp}$ and $\mathcal{D}$ be the sets of trajectories generated by an expert, and by some known distribution q, respectively. We may infer $\boldsymbol{\theta}$ by maximizing the likelihood, $p(\mathcal{D}_{\exp}|\boldsymbol{\theta})$, or equivalently, minimizing the negative log-likelihood loss

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}_{\exp}|} \sum_{\boldsymbol{\tau} \in \mathcal{D}_{\exp}} R_{\boldsymbol{\theta}}(\boldsymbol{\tau}) + \log \frac{1}{|\mathcal{D}|} \sum_{\boldsymbol{\tau} \in \mathcal{D}} \frac{\exp(R_{\boldsymbol{\theta}}(\boldsymbol{\tau}))}{q(\boldsymbol{\tau})} \tag{5.11}$$
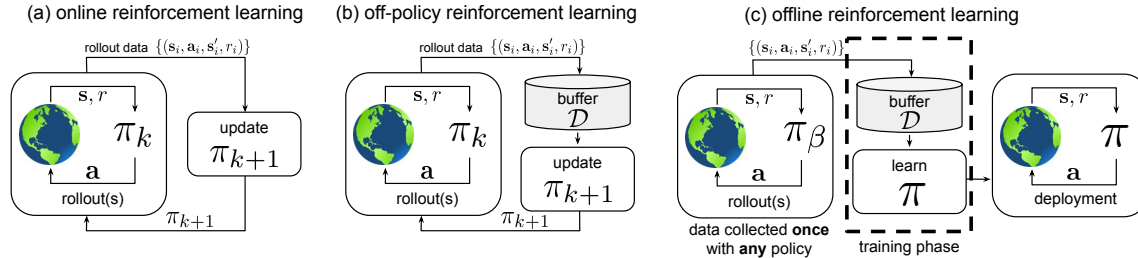
Figure 5.3: Comparison of online on-policy RL, online off-policy RL, and offline RL. From Figure 1 of [Lev+20a]. Used with kind permission of Sergey Levine.

The term inside the log of the loss is an importance sampling estimate of Z that is unbiased as long as $q(\boldsymbol{\tau}) > 0$ for all $\boldsymbol{\tau}$. However, in order to reduce the variance, we can choose q adaptively as $\boldsymbol{\theta}$ is being updated. The optimal sampling distribution, $q_*(\boldsymbol{\tau}) \propto \exp(R_{\boldsymbol{\theta}}(\boldsymbol{\tau}))$, is hard to obtain. Instead, we may find a policy $\hat{\pi}$ which induces a distribution that is close to q_*, for instance, using methods of maximum entropy RL discussed in Section 1.5.3. Interestingly, the process above produces the inferred reward $R_{\boldsymbol{\theta}}$ as well as an approximate optimal policy $\hat{\pi}$. This approach is used by **guided cost learning** [FLA16], and found effective in robotics applications.

5.4.3 Imitation learning by divergence minimization

We now discuss a different, but related, approach to IL. Recall that the reward function depends only on the state and action in an MDP. It implies that if we can find a policy π, so that $p_{\pi}^{\gamma}(s, a)$ and $p_{\pi_{\exp}}^{\gamma}(s, a)$ are close, then π receives similar long-term reward as $\pi_{\exp}$, and is a good imitation of $\pi_{\exp}$ in this regard. A number of IL algorithms find π by minimizing the divergence between p_{π}^{γ} and $p_{\pi_{\exp}}^{\gamma}$. We will largely follow the exposition of [GZG19]; see [Ke+19] for a similar derivation.

Let f be a convex function, and D_f be the corresponding f-divergence [Mor63; AS66; Csi67; LV06; CS04]. From the above intuition, we want to minimize $D_f\left(p_{\pi_{\exp}}^{\gamma} \middle\| p_{\pi}^{\gamma}\right)$. Then, using a variational approximation of D_f [NWJ10], we can solve the following optimization problem for π:

$$\min_{\pi} \max_{\boldsymbol{w}} \mathbb{E}_{p_{\pi_{\exp}}^{\gamma}(s,a)} \left[T_{\boldsymbol{w}}(s, a)\right] - \mathbb{E}_{p_{\pi}^{\gamma}(s,a)} \left[f^*(T_{\boldsymbol{w}}(s, a))\right] \tag{5.12}$$

where f^* is the convex conjugate of f, and $T_{\boldsymbol{w}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is some function parameterized by $\boldsymbol{w}$. We can think of π as a generator (of actions) and $T_{\boldsymbol{w}}$ as an adversarial critic that is used to compare the generated (s, a) pairs to the real ones. Thus the first expectation can be estimated using $\mathcal{D}_{\exp}$, as in behavior cloning, and the second can be estimated using trajectories generated by policy π. Furthermore, to implement this algorithm, we often use a parametric policy representation $\pi_{\boldsymbol{\theta}}$, and then perform stochastic gradient updates to find a saddle-point to Equation (5.12). With different choices of the convex function f, we can obtain many existing IL algorithms, such as **generative adversarial imitation learning** (**GAIL**) [HE16] and **adversarial inverse RL** (**AIRL**) [FLL18], etc.

5.5 Offline RL

Offline reinforcement learning (also called **batch reinforcement learning** [LGR12]) is concerned with learning a reward maximizing policy from a fixed, static dataset, collected by some existing policy, known as the **behavior policy**. Thus no interaction with the environment is allowed (see Figure 5.3). This makes policy learning harder than the online case, since we do not know the consequences of actions that were not taken in a given state, and cannot test any such "counterfactual" predictions by trying them. (This is the same problem as in off-policy RL, which we discussed in Section 3.5.) In addition, the policy will be deployed

on new states that it may not have seen, requiring that the policy generalize out-of-distribution, which is the main bottleneck for current offline RL methods [Par+24b].

A very simple and widely used offline RL method is known as behavior cloning or BC. This amounts to training a policy to predict the observed output action a_t associated with each observed state s_t, so we aim to ensure $\pi(s_t) \approx a_t$, as in supervised learning. This assumes the offline dataset was created by an expert, and so falls under the umbrella of imitation learning (see Section 5.4.1 for details). By contrast, offline RL methods can leverage suboptimal data. We give a brief summary of some of these methods below. For more details, see e.g., [Lev+20b; Che+24b; Cet+24]. For some offline RL benchmarks, see DR4L [Fu+20], RL Unplugged [Gul+20], OGBench (Offline Goal-Conditioned benchmark) [Par+24a], and D5RL [Raf+24].

5.5.1 Offline model-free RL

In principle, we can tackle offline RL using the off-policy methods that we discussed in Section 3.5. These use some form of importance sampling, based on $\pi(a|s)/\pi_b(a|s)$, to reweight the data in the replay buffer $\mathcal{D}$, which was collected by the behavior policy, towards the current policy (the one being evaluated/ learned). Unfortunately, such methods only work well if the behavior policy is is close to the new policy. In the online RL case, this can be ensured by gradually updating the new policy away from the behavior policy, and then sampling new data from the updated policy (which becomes the new behavior policy). Unfortunately, this is not an option in the offline case. Thus we need to use other strategies to control the discrepancy between the behavior policy and learned policy, as we discuss below. (Besides the algorithmic techniques we discuss, another reliable way to get better offline RL performance is to train on larger, more diverse datasets, as shown in [Kum+23].)

5.5.1.1 Policy constraint methods

In the **policy constraint** method, we use a modified form of actor-critic, which, at iteration k, uses an update of the form

$$Q_{k+1}^\pi \leftarrow \underset{Q}{\operatorname{argmin}} \, \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left(Q(s,a) - (R(s,a) + \gamma \mathbb{E}_{\pi_k(a'|s')} \left[Q_k^\pi(s',a') \right]) \right)^2 \right] \tag{5.13}$$

$$\pi_{k+1} \leftarrow \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{s\sim\mathcal{D}} \left[\mathbb{E}_{\pi(a|s)} \left[Q_{k+1}^\pi(s,a) \right] \right] \quad \text{s.t.} \quad D(\pi, \pi_b) \leq \epsilon \tag{5.14}$$

where $D(\pi(\cdot|s), \pi_b(\cdot|s))$ is a divergence measure on distributions, such as KL divergence or another f-divergence. This ensures that we do not try to evaluate the Q function on actions a' that are too dissimilar from those seen in the data buffer (for each sampled state s), which might otherwise result in artefacts similar an adversarial attack.

As an alternative to adding a constraint, we can add a penalty of $\alpha D(\pi(\cdot|s), \pi_b(\cdot|s))$ to the target Q value and the actor objective, resulting in the following update:

$$Q_{k+1}^\pi \leftarrow \underset{Q}{\operatorname{argmin}} \, \mathbb{E}_{(s,a,s')\sim\mathcal{D}} \left[\left(Q(s,a) - (R(s,a) + \gamma \mathbb{E}_{\pi_k(a'|s')} \left[Q_k^\pi(s',a') - \alpha\gamma D(\pi_k(\cdot|s'), \pi_b(\cdot|s')) \right]) \right)^2 \right] \tag{5.15}$$

$$\pi_{k+1} \leftarrow \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{s\sim\mathcal{D}} \left[\mathbb{E}_{\pi(a|s)} \left[Q_{k+1}^\pi(s,a) \right] - \alpha D(\pi(\cdot|s'), \pi_b(\cdot|s')) \right] \tag{5.16}$$

One problem with the above method is that we have to fit a parametric model to $\pi_b(a|s)$ in order to evaluate the divergence term. Fortunately, in the case of KL, the divergence can be enforced implicitly, as in the **advantage weighted regression** or **AWR** method of [Pen+19], the **reward weighted regression** method of [PS07], the **advantage weighted actor critic** or **AWAC** method of [Nai+20], the **advantage weighted behavior model** or **ABM** method of [Sie+20], In this approach, we first solve (nonparametrically) for the new policy under the KL divergence constraint to get $\overline{\pi}_{k+1}$, and then we project this into the required

policy function class via supervised regression, as follows:

$$\overline{\pi}_{k+1}(a|s) \leftarrow \frac{1}{Z}\pi_b(a|s)\exp\left(\frac{1}{\alpha}Q_k^\pi(s,a)\right) \tag{5.17}$$

$$\pi_{k+1} \leftarrow \operatorname*{argmin}_\pi D_{\mathbb{KL}}\left(\overline{\pi}_{k+1} \parallel \pi\right) \tag{5.18}$$

In practice the first step can be implemented by weighting samples from $\pi_b(a|s)$ (i.e., from the data buffer) using importance weights given by $\exp\left(\frac{1}{\alpha}Q_k^\pi(s,a)\right)$, and the second step can be implemented via supervised learning (i.e., maximum likelihood estimation) using these weights.

It is also possible to replace the KL divergence with an integral probability metric (IPM), such as the maximum mean discrepancy (MMD) distance, which can be computed from samples, without needing to fit a distribution $\pi_b(a|s)$. This approach is used in [Kum+19]. This has the advantage that it can constrain the support of the learned policy to be a subset of the behavior policy, rather than just remaining close to it. To see why this can be advantageous, consider the case where the behavior policy is uniform. In this case, constraining the learned policy to remain close (in KL divergence) to this distribution could result in suboptimal behavior, since the optimal policy may just want to put all its mass on a single action (for each state).

5.5.1.2 Behavior-constrained policy gradient methods

Recently a class of methods has been developed that is simple and effective: we first learn a baseline policy $\pi(a|s)$ (using BC) and a Q function (using Bellman minimization) on the offline data, and then update the policy parameters to pick actions that have high expected value according to Q and which are also likely under the BC prior. An early example of this is the $Q^\dagger$ algorithm of [Fuj+19]. In [FG21], they present the **DDPG+BC** method, which optimizes

$$\max_\pi J(\pi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[Q(s,\mu^\pi(s)) + \alpha\log\pi(a|s)\right] \tag{5.19}$$

where $\mu^\pi(s) = \mathbb{E}_{\pi(a|s)}[a]$ is the mean of the predicted action, and α is a hyper-parameter. As another example, the **DQL** method of [WHZ23] optimizes a diffusion policy using

$$\min_\pi \mathcal{L}(\pi) = \mathcal{L}_{\text{diffusion}}(\pi) + \mathcal{L}_q(\pi) = \mathcal{L}_{\text{diffusion}}(\pi) - \alpha\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(\cdot|s)}\left[Q(s,a)\right] \tag{5.20}$$

Finally, [Aga+22b] discusses how to transfer the policy from a previous agent to a new agent by combining BC with Q learning.

5.5.1.3 Uncertainty penalties

An alternative way to avoid picking out-of-distribution actions, where the Q function might be unreliable, is to add a penalty term to the Q function based on the estimated epistemic uncertainty, given the dataset $\mathcal{D}$, which we denote by $\text{Unc}(P_D(Q^\pi))$, where $P_D(Q^\pi)$ is the distribution over Q functions, and Unc is some metric on distributions. For example, we can use a deep ensemble to represent the distribution, and use the variance of $Q(s,a)$ across ensemble members as a measure of uncertainty. This gives rise to the following policy improvement update:

$$\pi_{k+1} \leftarrow \operatorname*{argmax}_\pi \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{\pi(a|s)}\left[\mathbb{E}_{P_D(Q_{k+1}^\pi)}\left[Q_{k+1}^\pi(s,a)\right]\right] - \alpha\text{Unc}(P_D(Q_{k+1}^\pi))\right] \tag{5.21}$$

For examples of this approach, see e.g., [An+21; Wu+21; GGN22].

5.5.1.4 Conservative Q-learning and pessimistic value functions

An alternative to explicitly estimating uncertainty is to add a **conservative penalty** directly to the Q-learning error term. That is, we minimize the following wrt $\boldsymbol{w}$ using each batch of data $\mathcal{B}$:

$$\overline{\mathcal{E}}(\mathcal{B},\boldsymbol{w}) = \alpha\mathcal{C}(\mathcal{B},\boldsymbol{w}) + \mathcal{E}(\mathcal{B},\boldsymbol{w}) \tag{5.22}$$

where $\mathcal{E}(\mathcal{B}, \boldsymbol{w}) = \mathbb{E}_{(s,a,s') \in \mathcal{B}} \left[(Q_{\boldsymbol{w}}(s,a) - (r + \gamma \max_{a'} Q_{\boldsymbol{w}}(s',a')))^2 \right]$ is the usual loss for Q-learning, and $\mathcal{C}(\mathcal{B}, \boldsymbol{w})$ is some conservative penalty. In the **conservative Q learning** or **CQL** method of [Kum+20], we use the following penalty term:

$$\mathcal{C}(\mathcal{B}, \boldsymbol{w}) = \mathbb{E}_{s \sim \mathcal{B}, a \sim \pi(\cdot|s)} [Q_{\boldsymbol{w}}(s,a)] - \mathbb{E}_{(s,a) \sim \mathcal{B}} [Q_{\boldsymbol{w}}(s,a)] \tag{5.23}$$

If π is the behavior policy, this penalty becomes 0.

5.5.2 Offline model-based RL

In Chapter 4, we discussed model-based RL, which can train a dynamics model given a fixed dataset, and then use this to generate synthetic data to evaluate and then optimize different possible policies. However, if the model is wrong, the method may learn a suboptimal policy, as we discussed in Section 4.2.3. This problem is particularly severe in the offline RL case, since we cannot recover from any errors by collecting more data. Therefore various conservative MBRL algorithms have been developed, to avoid exploiting model errors. For example, [Kid+20] present the **MOREL** algorithm, and [Yu+20] present the **MOPO** algorithm. Unlike the value function uncertainty method of Section 5.5.1.3, or the conservative value function method of Section 5.5.1.4, these model-based methods add a penalty for visiting states where the model is likely to be incorrect.

In more detail, let $u(s,a)$ be an estimate of the uncertainty of the model's predictions given input (s,a). In MOPO, they define a conservative reward using $\overline{R}(s,a) = R(s,a) - \lambda u(s,a)$, and in MOREL, they modify the MDP so that the agent enters an absorbing state with a low reward when $u(s,a)$ is sufficiently large. In both cases, it is possible to prove that the model-based estimate of the policy's performance under the modified reward or dynamics is a lower bound of the performance of the policy's true performance in the real MDP, provided that the uncertainty function u is an error oracle, which means that is satisfies $D(M_{\boldsymbol{\theta}}(s'|s,a), M^*(s'|s,a)) \leq u(s,a)$, where M^* is the true dynamics, and $M_{\boldsymbol{\theta}}$ is the estimated dynamics.

For more information on offline MBRL methods, see [Che+24c].

5.5.3 Offline RL using reward-conditioned sequence modeling

Recently an approach to offline RL based on sequence modeling has become very popular. The basic idea — known as **upside down RL** [Sch19] or **RvS** (RL via Supervised learning) [KPL19; Emm+21] — is to train a generative model over future states and/or actions conditioned on the observed reward, rather than predicting the reward given a state-action trajectory. At test time, the conditioning is changed to represent the desired reward, and futures are sampled from the model. The implementation of this idea then depends on what kind of generative model to use, as we discuss below.

The **trajectory transformer** method of [JLL21] learns a joint model of the form $p(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, \boldsymbol{r}_{1:T})$ using a transformer, and then samples from this using beam search, selecting the ones with high reward (similar to MPC, Section 4.1.1). The **decision transformer** [Che+21b] is related, but just generates action sequences, and conditions on the past observations and the future reward-to-go. That is, it fits

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \, \mathbb{E}_{p_{\mathcal{D}}} \left[\log \pi_{\boldsymbol{\theta}}(a_t | s_{0:t}, a_{0:t-1}, \mathrm{RTG}_{0:t}) \right] \tag{5.24}$$

where $\mathrm{RTG}_t = \sum_{k=t}^{T} r_t$ is the return to go. (For a comparison of decision transformers to other offline RL methods, see [Bha+24].)

The **diffuser** method of [Jan+22] is a diffusion version of trajectory transformer, so it fits $p(\boldsymbol{s}_{1:T}, \boldsymbol{a}_{1:T}, \boldsymbol{r}_{1:T})$ using diffusion, where the action space is assumed to be continuous. They also replace beam search with classifier guidance. The **decision diffuser** method of [Aja+23] extends diffuser by using classifer-free guidance, where the conditioning signal is the reward-to-go, simlar to decision transformer. However, unlike diffuser, the decision diffuser just models the future state trajectories (rather than learning a joint distribution over states and actions), and infers the actions using an **inverse dynamics model** $a_t = \pi(s_t, s_{t+1})$, which is trained using supervised learning.

One problem with the above approaches is that conditioning on a desired return and taking the predicted action can fail dramatically in stochastic environments, since trajectories that result in a return may have only achieved that return due to chance [PMB22; Yan+23; Bra+22; Vil+22]. (This is related to the optimism bias in the control-as-inference approach discussed in Section 1.5.)

5.5.4 Hybrid offline/online methods

Despite the progress in offline RL, it is fundamentally more limited in what it can learn compared to online RL [OCD21]. Therefore, various hybrids of offline and online RL have been proposed, such as [Bal+23] and [Nak+23].

For example, [Nak+23] suggest pre-training with offline RL (specifically CQL) followed by online finetuning. Naively this does not work that well, because CQL can be too conservative, requiring the online learning to waste some time at the beginning fixing the pessimism. So they propose a small modification to CQL, known as **calibrated Q learning**. This simply prevents CQL from being too conservative, by replacing the CQL regularizer with

$$\min_Q \max_\pi J(Q,\pi) + \alpha \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi(a|s)} \left[\max(Q(s,a), V^{\pi_\beta}(s)) - \alpha \mathbb{E}_{(s,a) \sim \mathcal{D}} \left[Q(s,a) \right] \right] \tag{5.25}$$

where the $Q(s,a)$ term inside the max ensures conservatism (so Q lower bounds the value of the learned policy), and the $V^{\pi_\beta}(s)$ term ensures "calibration" (so Q upper bounds the value of the behavior policy). Then online finetuning is performed in the usual way.

5.6 LLMs and RL

In this section, we discuss some connections between RL and "**foundation models**" (see e.g., [Cen21]). These are large pretrained generative models of text and/or images such as **large language models** (**LLMs**) and their multimodal extension, sometimes called **vision language models** (**VLMs**). Note that this is a very fast growing field, so we only briefly mention a few highlights. For more details, see e.g. https://github.com/WindyLab/LLM-RL-Papers.

5.6.1 RL for LLMs

We can think of LLMs as agents, where the state s_t is the entire sequence of previous words, $s_t = (w_1, \ldots, w_{t-1})$, the action is the next word w_t[2], the stochastic policy $\pi(a_t|s_t)$ is the LLM, and the transition model is the determistic function $p(s_{t+1}|s_t, a_t) = \delta(s_t = \text{concat}(s_t, a_t))$. We see that the size of the state grows linearly over time, which is a standard way to capture non-local dependencies in a Markov model.

We discuss how to train these models below. Once trained, they are used in a semi-MDP fashion, in which at round t the agent generates an answer $a_t = (a_{t,1}, \ldots, a_{t,N_t})$, which is a sequence of N_t tokens, in response to a prompt from the user, $p_t = (p_{t,1}, \ldots, p_{t,M_t})$, and the previous context (dialog history), $c_t = (p_{1,1:M_1}, a_{1,1:N_1}, p_{2,1:M_2}, \ldots, a_{t-1,1:N_{t-1}})$. We can now define the state as the sequence of tokens $s_t = (c_t, p_t)$. Similarly, the action sequence a_t can be flattened into a single atomic (string-valued) action, since there is no intermediate feedback from the environment after each token is produced.[3] Note that, if there is a single round of prompting and answering (as is often assumed during training), then this is a contextual bandit problem rather than a full MDP. In particular, the context is the string p_t and the action is the string a_t. However, in multi-turn dialog situations, the agent's actions will affect the environment (i.e., the user's mental state, and hence subsequent prompt p_{t+1}), turning it into a full MDP.

---

[2]When using VLMs, the "words" are a tokenized representation of the visual input and/or output. Even when using language, the elementary components w_t are sub-words (which allows for generalization), not words. So a more precise term would be "tokens" instead of "words".

[3]The fact that the action (token) sequence is generated by an autoregressive policy inside the agent's head is an implementation detail, and not part of the problem specification; for example, the agent could instead use discrete diffusion to generate $a_t = (a_{t,1}, \ldots, a_{t,N_t})$.

5.6.1.1 RLHF

LLMs are usually trained with behavior cloning, i.e., MLE on a fixed dataset, such as a large text (and tokenized image) corpus scraped from the web. This is called "**pre-training**". We can then improve their performance using RL, as we describe below; this is called "**post-training**".

A common way to perform post-training is to use **reinforcement learning from human feedback** or **RLHF**. This technique, which was first introduced in the InstructGPT paper [Ouy+22], works as follows. First a large number of (context, answer0, answer1) tuples are generated, either by a human or an LLM. Then human raters are asked if they prefer answer 0 or answer 1. Let $y = 0$ denote the event that they prefer answer 0, and $y = 1$ the event that they prefer answer 1. We can then fit a model of the form

$$p(y = 0|a_0, a_1, c) = \frac{\exp(\phi(c, a_0))}{\exp(\phi(c, a_0)) + \exp(\phi(c, a_1))} \tag{5.26}$$

using binary cross entropy loss, where $\phi(c, a)$ is some function that maps text to a scalar (interpreted as logits). Typically $\phi(c, a)$ is a shallow MLP on top of the last layer of a pretrained LLM. Finally, we define the reward function as $R(s, a) = \phi(s, a)$, where s is the context (e.g., a prompt or previous dialog state), and a is the action (answer generated by LLM). We then use this reward to fine-tune the LLM using a policy gradient method such as PPO (Section 3.4.3), or a simpler method such as **RLOO** [Ahm+24], which is based on REINFORCE (Section 3.2).

Note that this form of training assumes the agent just interacts with a single action (answer) in response to a single prompt, so is learning the reward for a bandit problem, rather than the full MDP. Also, the learned reward function is a known parametric model (since it is fit to the human feedback data), whereas in RL, the reward is an unknown non-differentiable blackbox function. When viewed in this light, it becomes clear that one can also use non-RL algorithms to improve performance of LLMs, such as **DPO** [Raf+23] or the density estimation methods of [Dum+24]. For more details on RL for LLMs, see e.g., [Kau+23].

5.6.1.2 Assistance game

In general, any objective-maximizing agent may suffer from reward hacking (Section 5.2.1), even if the reward has been learned using lots of RLHF data. In [Rus19], Stuart Russell proposed a clever solution to this problem. Specifically, the human and machine are both treated as agents in a two-player cooperative game, called an **assistance game**, where the machine's goal is to maximize the user's utility (reward) function, which is inferred based on the human's behavior using inverse RL. That is, instead of trying to learn a point estimate of the reward function using RLHF, and then optimizing that, we treat the reward function as an unknown part of the environment. If we adopt a Bayesian perspective on this, we can maintain a posterior belief over the model parameters. This will incentivize the agent to perform information gathering actions. For example, if the machine is uncertain about whether something is a good idea or not, it will proceed cautiously (e.g., by asking the user for their preference), rather than blindly solving the wrong problem. For more details on this framework, see [Sha+20].

5.6.1.3 Run-time inference as MPC

Recently the LLM community has investigated ways to improve the "reasoning" performance of LLMs by using MCTS-like methods (see Section 4.1.3). The basic idea is to perform Monte Carlo rollouts of many possible action sequences (by generating different "chains of thought" in response to the context so far), and then applying a value function to the leaves of this search tree to decide on which trajectory to return as the final "decision". The value function is usually learned using policy gradient methods, such as REINFORCE (see e.g., [Zel+24]). It is believed that OpenAI's recently released **o1** (aka **Strawberry**) model[4] uses similar techniques, most likely pre-training on large numbers of human reasoning traces.

Note that the resulting policy is an instance of MPC (Section 4.1.1) or decision time planning. This means that, as in MPC, the agent must replan after every new state observation (which incorporates the

---

[4]See https://openai.com/index/learning-to-reason-with-llms/.

response from the user), making the method much slower than "reactive" LLM policies, that does not use look-ahead search (but still conditions on the entire past context). However, once trained, it may be possible to distill this slower "system 2" policy into a faster reactive "system 1" policy.

5.6.2 LLMs for RL

There are many ways that (pretrained) LLMs can be used for RL, by leveraging their prior knowledge, their ability to generate code, their "reasoning" ability, and their ability to perform **in-context learning** (which can be viewed as a form of Bayesian inference [PAG24], which is a "gradient-free" way of optimally learning that is well suited to rapid learning from limited data). The survey in [Cao+24] groups the literature into four main categories: LLMs for pre-processing the inputs, LLMs for rewards, LLMs for world models, and LLMs for decision making or policies. In our brief presentation below, we follow this categorization. (See also [Spi+24] for a similar grouping.)

5.6.2.1 LLMs for pre-processing the input

If the input observations o_t sent to the agent are in natural language (or some other textual representation, such as JSON), it is natural to use an LLM to process them, in order to compute a more compact representation, $s_t = \phi(o_t)$, where ϕ can the hidden state of the last layer of an LLM. This encoder can either be frozen, or fine-tuned with the policy network. Note that we can also pass in the entire past observation history, $o_{1:t}$, as well as static "side information", such as instruction manuals or human hints; these can all be concatenated to form the LLM prompt.

For example, the **AlphaProof** system[5] uses an LLM (called the "formalizer network") to translate an informal specification of a math problem into the formal Lean representation, which is then passed to an agent (called the "solver network") which is trained, using the AlphaZero method (see Section 4.1.3.1), to generate proofs inside the Lean theorem proving environment. In this environment, the reward is 0 or 1 (proof is correct or not), the state space is a structured set of previously proved facts and the current goal, and the action space is a set of proof tactics. The agent itself is a separate transformer policy network (distincy from the formalizer network) that is trained from scratch in an incremental way, based on the AlphaZero method.

If the observations are images, it it is traditional to use a CNN to proccess the input, so $s_t \in \mathbb{R}^N$ would be an embedding vector. However, we could alternatively use a VLM to compute a structured representation, where s_t might be a set of tokens describing the scene at a high level. We then proceed as in the text case.

Note that the information that is extracted will heavily depend on the prompt that is used. Thus we should think of an LLM/VLM as an **active sensor** that we can control via prompts. Choosing how to control this sensor requires expanding the action space of the agent to include computational actions [Che+24d]. Note also that these kinds of "sensors" are very expensive to invoke, so an agent with some limits on its time and compute (which is all practical agents) will need to reason about the value of information and the cost of computation. This is called **metareasoning** [RW91]. Devising good ways to train agents to perform both computational actions (e.g., invoking an LLM or VLM) and environment actions (e.g., taking a step in the environment or calling a tool) is an open research problem.

5.6.2.2 LLMs for rewards

It is difficult to design a reward function to cause an agent to exhibit some desired behavior, as we discussed in Section 5.2. Fortunately LLMs can often help with this task. We discuss a few approaches below.

In [Kli+24], they present the **Motif** system, that uses an LLM in lieu of a human to provide preference judgements to an RLHF system. In more detail, a pre-trained policy is used to collect trajectories, from which pairs of states, (o, o'), are selected at random. The LLM is then asked which state is preferable, thus generating (o, o', y) tuples, which can be used to train a binary classifier from which a reward model is extracted, as in Section 5.6.1.1. In [Kli+24], the observations o are text captions generated by the NetHack game, but the same method could be applied to images if we used a VLM instead of an LLM for learning the

---

[5]See https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/.

reward. The learned reward model is then used as a shaping function (Section 5.2.3) when training an agent in the NetHack environment, which has very sparse reward.

In [Ma+24], they present the **Eureka** system, that learns the reward using bilevel optimization, with RL on the inner loop and LLM-powered evolutionary search on the outer loop. In particular, in the inner loop, given a candidate reward function R_i, we use PPO to train a policy, and then return a scalar quality score $S_i = S(R_i)$. In the outer loop, we ask an LLM to generate a new set of reward functions, R_i', given a population of old reward functions and their scores, (R_i, S_i), which have been trained and evaluated in parallel on a fleet of GPUs. The prompt also includes the source code of the environment simulator. Each generated reward function R_i is represented as a Python function, that has access to the ground truth state of the underlying robot simulator. The resulting system is able to learn a complex reward function that is sufficient to train a policy (using PPO) that can control a simulated robot hand to perform various dexterous manipulation tasks, including spinning a pen with its finger tips. In [Li+24], they present a somewhat related approach and apply it to Minecraft.

In [Ven+24], they propose **code as reward**, in which they prompt a VLM with an initial and goal image, and ask it to describe the corresponding sequence of tasks needed to reach the goal. They then ask the LLM to synthesize code that checks for completion of each subtask (based on processing of object properties, such as relative location, derived from the image). These reward functions are then "verified" by applying them to an offline set of expert and random trajectories; a good reward function should allocate high reward to the expert trajectories and low reward to the random ones. Finally, the reward functions are used as auxiliary rewards inside an RL agent.

There are of course many other ways an LLM could be used to help learn reward functions, and this remains an active area of research.

5.6.2.3   LLMs for world models

There are many papers that use transformers or diffusion models to represent the world model $p(s'|s, a)$, and learn them from data collected by the agent, as we discussed in Section 4.3. Here we focus our attention on ways to use pre-trained foundation models as world models (WM).

[Yan+24] presents **UniSim**, which is an action-conditioned video diffusion model trained on large amounts of robotics and visual navigation data. Combined with a VLM reward model, this can be used for decision-time planning as follows: sample candidate action trajectories from a proposal, generate the corresponding images, feed them to the reward model, score the rollouts, and then pick the best action from this set. (Note that this is just standard MPC in image space with a diffusion WM and a random shooting planning algorithm.)

[TKE24] presents **WorldCoder**, which takes a very different approach. It prompts a frozen LLM to generate code to represent the WM $p(s'|s, a)$, which it then uses inside of a planning algorithm. The agent then executes this in the environment, and passes back failed predictions to the LLM, asking it to improve the WM. (This is related to the Eureka reward-learning system mentioned in Section 5.6.2.2.)

There are of course many other ways an LLM could be used to help learn world models, and this remains an active area of research.

5.6.2.4   LLMs for policies

Finally we turn to LLMs as policies.

One approach is to pre-train a special purpose foundation model on state-action sequences (using behavior cloning), then sample the next action from it using $a_t \sim p(a_t|o_t, h_{t-1})$, where o_t is the latest observation and $h_{t-1} = (o_{1:t-1}, a_{1:t-1})$ is the history. See e.g., **Gato** model [Ree+22] **RT-2** [Zit+23], and **RoboCat** [Bou+23].

More recently it has become popular to leverage pre-trained LLMs that are trained on web data, and then to repurpose them as "agents" using in-context learning. We can then sample an action from the policy $\pi(a_t|p_t, o_t, h_{t-1})$, where p_t is a manually chosen prompt. This approach is used by the **ReAct** paper [Yao+22] which works by prompting the LLM to "think step-by-step" ("reasoning") and then to predict an action ("acting"). This approach can be extended by prompting the LLM to first retrieve relevant past examples
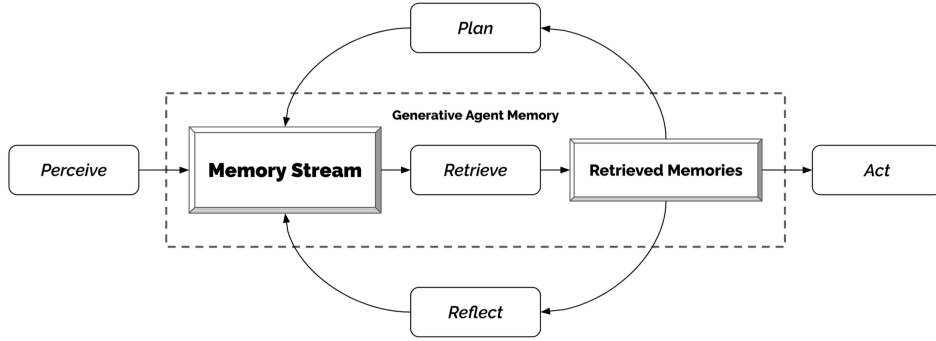
Figure 5.4: Illustration of how to use a pretrained LLM (combined with RAG) as a policy. From Figure 5 of [Par+23]. Used with kind permission of Joon Park.

from an external "memory", rather than explicitly storing the entire history h_t in the context (this is called **retrieval augmented generation** or **RAG**); see Figure 5.4 for an illustration. Note that no explicit learning (in the form of parametric updates) is performed in these systems; instead they rely entirely on in-context learning (and prompt engineering).

An alternative approach is to enumerate all possible discrete actions, and use the LLM to score them in terms of their likelihoods given the goal, and their suitability given a learned value function applied to the current state, i.e. $\pi(a_t = k | g, p_t, o_t, h_t) \propto \text{LLM}(w_k | g_t, p_t, h_t) V_k(o_t)$, where g_t is the current goal, w_k is a text description of action k, and V_k is the value function for action k. This is the approach used in the robotics **SayCan** approach [Ich+23], where the primitive actions a_k are separately trained goal-conditioned policies.

Calling the LLM at every step is very slow, so an alternative is to use the LLM to generate code that represents (parts of) the policy. For example, the **Voyager** system in [Wan+24a] builds up a reusable skill library (represented as Python functions), by alternating between environment exploration and prompting the (frozen) LLM to generate new tasks and skills, given the feedback collected so far.

There are of course many other ways an LLM could be used to help learn policies, and this remains an active area of research.

5.7 General RL, AIXI and universal AGI

The term "**general RL**" (see e.g., [Hut05; LHS13; HQC24; Maj21]) refers to the setup in which an agent receives a stream of observations $o_1, o_2, \ldots$ and rewards $r_1, r_2, \ldots$, and performs a sequence of actions in response, $a_1, a_2, \ldots$, but where we do not make any Markovian (or even stationarity) assumptions about the environment that generates the observation stream. Instead, we assume that the environment is a computable function or program p^*, which generated the observations $o_{1:t}$ and $r_{1:t}$ seen so far in response to the actions taken, $a_{1:t-1}$. We denote this by $U(p^*, \boldsymbol{a}_{1:t}) = (o_1 r_1 \cdots o_t r_t)$, where U is a universal Turing machine. If we use the receding horizon control strategy (see Section 4.1.1), the optimal action at each step is the one that maximizes the posterior expected reward-to-go (out to some horizon m steps into the future). If we assume the agent represents the unknown environment as a program $p \in \mathcal{M}$, then the optimal action is given by the following **expectimax** formula:

$$a_t = \underset{a_t}{\operatorname{argmax}} \sum_{o_t, r_t} \cdots \max_{a_m} \sum_{o_m, r_m} [r_t + \cdots + r_m] \sum_{p:U(p, \boldsymbol{a}_{1:m})=(o_1 r_1 \cdots o_m r_m)} \Pr(p) \tag{5.27}$$

where $\Pr(p)$ is the prior probability of p, and we assume the likelihood is 1 if p can generate the observations given the actions, and is 0 otherwise.

The key question is: what is a reasonable prior over programs? In [Hut05], Marcus Hutter proposed to apply the idea of **Solomonoff induction** [Sol64] to the case of an online decision making agent. This

amounts to using the prior $\Pr(p) = 2^{-\ell(p)}$, where $\ell(p)$ is the length of program p. This prior favors shorter programs, and the likelihood filters out programs that cannot explain the data.

The resulting agent is known as **AIXI**, where "AI" stands for "Artificial Intelligence" and "XI" referring to the Greek letter ξ used in Solomonoff induction. The AIXI agent has been called the "most intelligent general-purpose agent possible" [HQC24], and can be viewed as the theoretical foundation of (universal) **artificial general intelligence** or **AGI**.

Unfortunately, the AIXI agent is intractable to compute, since it relies on Solomonoff induction and Kolmogorov complexity, both of which are intractable, but various approximations can be devised. For example, we can approximate the expectimax with MCTS (see Section 4.1.3). Alternatively, [GM+24] showed that it is possible to use meta learning to train a generic sequence predictor, such as a transformer or LSTM, on data generated by random Turing machines, so that the transformer learns to approximate a universal predictor. Another approach is to learn a policy (to avoid searching over action sequences) using TD-learning (Section 2.3.2); the weighting term in the policy mixture requires that the agent predict its own future actions, so this approach is known as **self-AIXI** [Cat+23].

Note that AIXI is a normative theory for optimal agents, but is not very practical, since it does not take computational limitations into account. In [Aru+24a; Aru+24b], they describe an approach which extends the above Bayesian framework, while also taking into account the data budget (due to limited environment interactions) that real agents must contend with (which prohibits modeling the entire environment or finding the optimal action). This approach, known as **Capacity-Limited Bayesian RL** (CBRL), combines Bayesian inference, RL, and rate distortion theory, and can be seen as a normative theoretical foundation for computationally bounded rational agents.

Bibliography

[Abd+18]   A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. "Maximum a Posteriori Policy Optimisation". In: *International Conference on Learning Representations*. Feb. 2018. URL: https://openreview.net/pdf?id=S1ANxQW0b.

[ABM10]   J.-Y. Audibert, S. Bubeck, and R. Munos. "Best Arm Identification in Multi-Armed Bandits". In: *COLT*. 2010, pp. 41–53.

[ACBF02]   P. Auer, N. Cesa-Bianchi, and P. Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *MLJ* 47.2 (May 2002), pp. 235–256. URL: http://mercurio.srv.di.unimi.it/~cesabian/Pubblicazioni/ml-02.pdf.

[Ach+17]   J. Achiam, D. Held, A. Tamar, and P. Abbeel. "Constrained Policy Optimization". In: *ICML*. 2017. URL: http://arxiv.org/abs/1705.10528.

[Aga+14]   D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. "LASER: a scalable response prediction platform for online advertising". In: *WSDM*. 2014.

[Aga+21a] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan. "On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift". In: *JMLR* 22.98 (2021), pp. 1–76. URL: http://jmlr.org/papers/v22/19-736.html.

[Aga+21b] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *NIPS*. Aug. 2021. URL: http://arxiv.org/abs/2108.13264.

[Aga+22a] A. Agarwal, N. Jiang, S. Kakade, and W. Sun. *Reinforcement Learning: Theory and Algorithms*. 2022. URL: https://rltheorybook.github.io/rltheorybook_AJKS.pdf.

[Aga+22b] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. "Reincarnating Reinforcement Learning: Reusing Prior Computation to Accelerate Progress". In: *NIPS*. Vol. 35. 2022, pp. 28955–28971. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/ba1c5356d9164bb64c446a4b690226b0-Abstract-Conference.html.

[Ahm+24]   A. Ahmadian, C. Cremer, M. Gallé, M. Fadaee, J. Kreutzer, A. Üstün, and S. Hooker. "Back to basics: Revisiting REINFORCE style optimization for learning from Human Feedback in LLMs". In: *arXiv [cs.LG]* (Feb. 2024). URL: http://arxiv.org/abs/2402.14740.

[Aja+23]   A. Ajay, Y. Du, A. Gupta, J. B. Tenenbaum, T. S. Jaakkola, and P. Agrawal. "Is Conditional Generative Modeling all you need for Decision Making?" In: *ICLR*. 2023. URL: https://openreview.net/forum?id=sP1fo2K9DFG.

[AJO08]   P. Auer, T. Jaksch, and R. Ortner. "Near-optimal Regret Bounds for Reinforcement Learning". In: *NIPS*. Vol. 21. 2008. URL: https://proceedings.neurips.cc/paper_files/paper/2008/file/e4a6222cdb5b34375400904f03d8e6a5-Paper.pdf.

[AL+16]   J. Ala-Luhtala, N. Whiteley, K. Heine, and R. Piche. "An Introduction to Twisted Particle Filters and Parameter Estimation in Non-linear State-space Models". In: *IEEE Trans. Signal Process.* 64.18 (2016), pp. 4875–4890. URL: http://arxiv.org/abs/1509.09175.

[Ale+23] L. N. Alegre, A. L. C. Bazzan, A. Nowé, and B. C. da Silva. "Multi-step generalized policy improvement by leveraging approximate models". In: *NIPS*. Vol. 36. Curran Associates, Inc., 2023, pp. 38181–38205. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/77c7faab15002432ba1151e8d5cc389a-Abstract-Conference.html.

[Alo+24] E. Alonso, A. Jelley, V. Micheli, A. Kanervisto, A. Storkey, T. Pearce, and F. Fleuret. "Diffusion for world modeling: Visual details matter in Atari". In: *arXiv [cs.LG]* (May 2024). URL: http://arxiv.org/abs/2405.12399.

[AM89] B. D. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice-Hall International, Inc., 1989.

[Ama98] S Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Comput.* 10.2 (1998), pp. 251–276. URL: http://dx.doi.org/10.1162/089976698300017746.

[AMH23] A. Aubret, L. Matignon, and S. Hassas. "An information-theoretic perspective on intrinsic motivation in reinforcement learning: A survey". en. In: *Entropy* 25.2 (Feb. 2023), p. 327. URL: https://www.mdpi.com/1099-4300/25/2/327.

[Ami+21] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup. "A survey of exploration methods in reinforcement learning". In: *arXiv [cs.LG]* (Aug. 2021). URL: http://arxiv.org/abs/2109.00157.

[An+21] G. An, S. Moon, J.-H. Kim, and H. O. Song. "Uncertainty-Based Offline Reinforcement Learning with Diversified Q-Ensemble". In: *NIPS*. Vol. 34. Dec. 2021, pp. 7436–7447. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/3d3d286a8d153a4a58156d0e02d8570c-Paper.pdf.

[And+17] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. "Hindsight Experience Replay". In: *arXiv [cs.LG]* (July 2017). URL: http://arxiv.org/abs/1707.01495.

[And+20] O. M. Andrychowicz et al. "Learning dexterous in-hand manipulation". In: *Int. J. Rob. Res.* 39.1 (2020), pp. 3–20. URL: https://doi.org/10.1177/0278364919887447.

[Ant+22] I. Antonoglou, J. Schrittwieser, S. Ozair, T. K. Hubert, and D. Silver. "Planning in Stochastic Environments with a Learned Model". In: *ICLR*. 2022. URL: https://openreview.net/forum?id=X6D9bAHhBQ1.

[AP23] S. Alver and D. Precup. "Minimal Value-Equivalent Partial Models for Scalable and Robust Planning in Lifelong Reinforcement Learning". en. In: *Conference on Lifelong Learning Agents*. PMLR, Nov. 2023, pp. 548–567. URL: https://proceedings.mlr.press/v232/alver23a.html.

[AP24] S. Alver and D. Precup. "A Look at Value-Based Decision-Time vs. Background Planning Methods Across Different Settings". In: *Seventeenth European Workshop on Reinforcement Learning*. Oct. 2024. URL: https://openreview.net/pdf?id=Vx2ETvHId8.

[Arb+23] J. Arbel, K. Pitas, M. Vladimirova, and V. Fortuin. "A Primer on Bayesian Neural Networks: Review and Debates". In: *arXiv [stat.ML]* (Sept. 2023). URL: http://arxiv.org/abs/2309.16314.

[ARKP24] S. Alver, A. Rahimi-Kalahroudi, and D. Precup. "Partial models for building adaptive model-based reinforcement learning agents". In: *COLLAS*. May 2024. URL: https://arxiv.org/abs/2405.16899.

[Aru+17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. "A Brief Survey of Deep Reinforcement Learning". In: *IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding* (2017). URL: http://arxiv.org/abs/1708.05866.

[Aru+24a]   D. Arumugam, M. K. Ho, N. D. Goodman, and B. Van Roy. "Bayesian Reinforcement Learning With Limited Cognitive Load". en. In: *Open Mind* 8 (Apr. 2024), pp. 395–438. URL: https://direct.mit.edu/opmi/article-pdf/doi/10.1162/opmi_a_00132/2364075/opmi_a_00132.pdf.

[Aru+24b]   D. Arumugam, S. Kumar, R. Gummadi, and B. Van Roy. "Satisficing exploration for deep reinforcement learning". In: *Finding the Frame Workshop at RLC*. July 2024. URL: https://openreview.net/forum?id=tHCpsrzehb.

[AS22]   D. Arumugam and S. Singh. "Planning to the information horizon of BAMDPs via epistemic state abstraction". In: *NIPS*. Oct. 2022.

[AS66]   S. M. Ali and S. D. Silvey. "A General Class of Coefficients of Divergence of One Distribution from Another". In: *J. R. Stat. Soc. Series B Stat. Methodol.* 28.1 (1966), pp. 131–142. URL: http://www.jstor.org/stable/2984279.

[ASN20]   R. Agarwal, D. Schuurmans, and M. Norouzi. "An Optimistic Perspective on Offline Reinforcement Learning". en. In: *ICML*. PMLR, Nov. 2020, pp. 104–114. URL: https://proceedings.mlr.press/v119/agarwal20c.html.

[Att03]   H. Attias. "Planning by Probabilistic Inference". In: *AI-Stats*. 2003. URL: http://research.goldenmetallic.com/aistats03.pdf.

[AY20]   B. Amos and D. Yarats. "The Differentiable Cross-Entropy Method". In: *ICML*. 2020. URL: http://arxiv.org/abs/1909.12830.

[Bad+20]   A. P. Badia, B. Piot, S. Kapturowski, P Sprechmann, A. Vitvitskyi, D. Guo, and C Blundell. "Agent57: Outperforming the Atari Human Benchmark". In: *ICML* 119 (Mar. 2020), pp. 507–517. URL: https://proceedings.mlr.press/v119/badia20a/badia20a.pdf.

[Bai95]   L. C. Baird. "Residual Algorithms: Reinforcement Learning with Function Approximation". In: *ICML*. 1995, pp. 30–37.

[Bal+23]   P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. "Efficient Online Reinforcement Learning with Offline Data". en. In: *ICML*. PMLR, July 2023, pp. 1577–1594. URL: https://proceedings.mlr.press/v202/ball23a.html.

[Ban+23]   D. Bansal, R. T. Q. Chen, M. Mukadam, and B. Amos. "TaskMet: Task-driven metric learning for model learning". In: *NIPS*. Ed. by A Oh, T Naumann, A Globerson, K Saenko, M Hardt, and S Levine. Vol. abs/2312.05250. Dec. 2023, pp. 46505–46519. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91a5742235f70ae846436d9780e9f1d4-Abstract-Conference.html.

[Bar+17]   A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. "Successor Features for Transfer in Reinforcement Learning". In: *NIPS*. Vol. 30. 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf.

[Bar+19]   A. Barreto et al. "The Option Keyboard: Combining Skills in Reinforcement Learning". In: *NIPS*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/251c5ffd6b62cc21c446c963c76cf214-Paper.pdf.

[Bar+20]   A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup. "Fast reinforcement learning with generalized policy updates". en. In: *PNAS* 117.48 (Dec. 2020), pp. 30079–30087. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1907370117.

[BBS95]   A. G. Barto, S. J. Bradtke, and S. P. Singh. "Learning to act using real-time dynamic programming". In: *AIJ* 72.1 (1995), pp. 81–138. URL: http://www.sciencedirect.com/science/article/pii/000437029400011O.

[BDG00]   C. Boutilier, R. Dearden, and M. Goldszmidt. "Stochastic dynamic programming with factored representations". en. In: *Artif. Intell.* 121.1-2 (Aug. 2000), pp. 49–107. URL: http://dx.doi.org/10.1016/S0004-3702(00)00033-3.

[BDM10]   M. Briers, A. Doucet, and S. Maskel. "Smoothing algorithms for state-space models". In: *Annals of the Institute of Statistical Mathematics* 62.1 (2010), pp. 61–89.

[BDM17]   M. G. Bellemare, W. Dabney, and R. Munos. "A Distributional Perspective on Reinforcement Learning". In: *ICML*. 2017. URL: http://arxiv.org/abs/1707.06887.

[BDR23]   M. G. Bellemare, W. Dabney, and M. Rowland. *Distributional Reinforcement Learning*. http://www.distributional-rl.org. MIT Press, 2023.

[Bel+13]   M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *JAIR* 47 (2013), pp. 253–279.

[Bel+16]   M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *NIPS*. 2016. URL: http://arxiv.org/abs/1606.01868.

[Ber19]   D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019. URL: http://www.mit.edu/~dimitrib/RLbook.html.

[Ber24]   D. P. Bertsekas. "Model Predictive Control and Reinforcement Learning: A unified framework based on Dynamic Programming". In: *arXiv [eess.SY]* (June 2024). URL: http://arxiv.org/abs/2406.00592.

[Bha+24]   P. Bhargava, R. Chitnis, A. Geramifard, S. Sodhani, and A. Zhang. "When should we prefer Decision Transformers for Offline Reinforcement Learning?" In: *ICLR*. 2024. URL: https://arxiv.org/abs/2305.14550.

[BHP17]   P.-L. Bacon, J. Harb, and D. Precup. "The Option-Critic Architecture". In: *AAAI*. 2017.

[BKH16]   J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer Normalization". In: (2016). arXiv: 1607.06450 [stat.ML]. URL: http://arxiv.org/abs/1607.06450.

[BLM16]   S. Boucheron, G. Lugosi, and P. Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, 2016.

[BM+18]   G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, T. B. Dhruva, A. Muldal, N. Heess, and T. Lillicrap. "Distributed Distributional Deterministic Policy Gradients". In: *ICLR*. 2018. URL: https://openreview.net/forum?id=SyZipzbCb&noteId=SyZipzbCb.

[BMS11]   S. Bubeck, R. Munos, and G. Stoltz. "Pure Exploration in Finitely-armed and Continuous-armed Bandits". In: *Theoretical Computer Science* 412.19 (2011), pp. 1832–1852.

[Boe+05]   P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. "A Tutorial on the Cross-Entropy Method". en. In: *Ann. Oper. Res.* 134.1 (2005), pp. 19–67. URL: https://link.springer.com/article/10.1007/s10479-005-5724-z.

[Bor+19]   D. Borsa, A. Barreto, J. Quan, D. J. Mankowitz, H. van Hasselt, R. Munos, D. Silver, and T. Schaul. "Universal Successor Features Approximators". In: *ICLR*. 2019. URL: https://openreview.net/pdf?id=S1VWjiRcKX.

[Bos16]   N. Bostrom. *Superintelligence: Paths, Dangers, Strategies*. en. London, England: Oxford University Press, Mar. 2016. URL: https://www.amazon.com/Superintelligence-Dangers-Strategies-Nick-Bostrom/dp/0198739834.

[Bou+23]   K. Bousmalis et al. "RoboCat: A Self-Improving Generalist Agent for Robotic Manipulation". In: *TMLR* (June 2023). URL: http://arxiv.org/abs/2306.11706.

[Bra+22]   D. Brandfonbrener, A. Bietti, J. Buckman, R. Laroche, and J. Bruna. "When does return-conditioned supervised learning work for offline reinforcement learning?" In: *NIPS*. June 2022. URL: http://arxiv.org/abs/2206.01079.

[BS23]   A. Bagaria and T. Schaul. "Scaling goal-based exploration via pruning proto-goals". en. In: *IJCAI*. Aug. 2023, pp. 3451–3460. URL: https://dl.acm.org/doi/10.24963/ijcai.2023/384.

[BSA83]   A. G. Barto, R. S. Sutton, and C. W. Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *SMC* 13.5 (1983), pp. 834–846. URL: http://dx.doi.org/10.1109/TSMC.1983.6313077.

[BT12]   M. Botvinick and M. Toussaint. "Planning as inference". en. In: *Trends Cogn. Sci.* 16.10 (2012), pp. 485–488. URL: https://pdfs.semanticscholar.org/2ba7/88647916f6206f7fcc137fe7866c58e6211e.pdf.

[Buc+17]   C. L. Buckley, C. S. Kim, S. McGregor, and A. K. Seth. "The free energy principle for action and perception: A mathematical review". In: *J. Math. Psychol.* 81 (2017), pp. 55–79. URL: https://www.sciencedirect.com/science/article/pii/S0022249617300962.

[Bur+18]   Y. Burda, H. Edwards, A Storkey, and O. Klimov. "Exploration by random network distillation". In: *ICLR*. Vol. abs/1810.12894. Sept. 2018.

[BXS20]   H. Bharadhwaj, K. Xie, and F. Shkurti. "Model-Predictive Control via Cross-Entropy and Gradient-Based Optimization". en. In: *Learning for Dynamics and Control*. PMLR, July 2020, pp. 277–286. URL: https://proceedings.mlr.press/v120/bharadhwaj20a.html.

[CA13]   E. F. Camacho and C. B. Alba. *Model predictive control*. Springer, 2013.

[Cao+24]   Y. Cao, H. Zhao, Y. Cheng, T. Shu, G. Liu, G. Liang, J. Zhao, and Y. Li. "Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods". In: *arXiv [cs.LG]* (Mar. 2024). URL: http://arxiv.org/abs/2404.00282.

[Car+23]   W. C. Carvalho, A. Saraiva, A. Filos, A. Lampinen, L. Matthey, R. L. Lewis, H. Lee, S. Singh, D. Jimenez Rezende, and D. Zoran. "Combining Behaviors with the Successor Features Keyboard". In: *NIPS*. Vol. 36. 2023, pp. 9956–9983. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/1f69928210578f4cf5b538a8c8806798-Abstract-Conference.html.

[Car+24]   W. Carvalho, M. S. Tomov, W. de Cothi, C. Barry, and S. J. Gershman. "Predictive representations: building blocks of intelligence". In: *Neural Comput.* (Feb. 2024). URL: https://gershmanlab.com/pubs/Carvalho24.pdf.

[Cas11]   P. S. Castro. "On planning, prediction and knowledge transfer in Fully and Partially Observable Markov Decision Processes". en. PhD thesis. McGill, 2011. URL: https://www.proquest.com/openview/d35984acba38c072359f8a8d5102c777/1?pq-origsite=gscholar&cbl=18750.

[Cas20]   P. S. Castro. "Scalable methods for computing state similarity in deterministic Markov Decision Processes". In: *AAAI*. 2020.

[Cas+21]   P. S. Castro, T. Kastner, P. Panangaden, and M. Rowland. "MICo: Improved representations via sampling-based state similarity for Markov decision processes". In: *NIPS*. Nov. 2021. URL: https://openreview.net/pdf?id=wFp6kmQELgu.

[Cas+23]   P. S. Castro, T. Kastner, P. Panangaden, and M. Rowland. "A kernel perspective on behavioural metrics for Markov decision processes". In: *TMLR* abs/2310.19804 (Oct. 2023). URL: https://openreview.net/pdf?id=nHfPXl1ly7.

[Cat+23]   E. Catt, J. Grau-Moya, M. Hutter, M. Aitchison, T. Genewein, G. Delétang, K. Li, and J. Veness. "Self-Predictive Universal AI". In: *NIPS*. Vol. 36. 2023, pp. 27181–27198. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/56a225639da77e8f7c0409f6d5ba996b-Abstract-Conference.html.

[Cen21]   Center for Research on Foundation Models (CRFM). "On the Opportunities and Risks of Foundation Models". In: (2021). arXiv: 2108.07258 [cs.LG]. URL: http://arxiv.org/abs/2108.07258.

[Cet+24]   E. Cetin, A. Tirinzoni, M. Pirotta, A. Lazaric, Y. Ollivier, and A. Touati. "Simple ingredients for offline reinforcement learning". In: *arXiv [cs.LG]* (Mar. 2024). URL: http://arxiv.org/abs/2403.13097.

[Cha+21]   A. Chan, H. Silva, S. Lim, T. Kozuno, A Mahmood, and M. White. "Greedification operators for policy optimization: Investigating forward and reverse KL divergences". In: *JMLR* abs/2107.08285.253 (July 2021), pp. 1–79. URL: http://jmlr.org/papers/v23/21-054.html.

[Che+20]   X. Chen, C. Wang, Z. Zhou, and K. W. Ross. "Randomized Ensembled Double Q-Learning: Learning Fast Without a Model". In: *ICLR*. Oct. 2020. URL: https://openreview.net/pdf?id=AY8zfZm0tDd.

[Che+21a]   C. Chen, Y.-F. Wu, J. Yoon, and S. Ahn. "TransDreamer: Reinforcement Learning with Transformer World Models". In: *Deep RL Workshop NeurIPS*. 2021. URL: http://arxiv.org/abs/2202.09481.

[Che+21b]   L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *arXiv [cs.LG]* (June 2021). URL: http://arxiv.org/abs/2106.01345.

[Che+24a]   F. Che, C. Xiao, J. Mei, B. Dai, R. Gummadi, O. A. Ramirez, C. K. Harris, A. R. Mahmood, and D. Schuurmans. "Target networks and over-parameterization stabilize off-policy bootstrapping with function approximation". In: *ICML*. May 2024.

[Che+24b]   J. Chen, B. Ganguly, Y. Xu, Y. Mei, T. Lan, and V. Aggarwal. "Deep Generative Models for Offline Policy Learning: Tutorial, Survey, and Perspectives on Future Directions". In: *TMLR* (Feb. 2024). URL: https://openreview.net/forum?id=Mm2cMDl9r5.

[Che+24c]   J. Chen, B. Ganguly, Y. Xu, Y. Mei, T. Lan, and V. Aggarwal. "Deep Generative Models for Offline Policy Learning: Tutorial, Survey, and Perspectives on Future Directions". In: *TMLR* (Feb. 2024). URL: https://openreview.net/forum?id=Mm2cMDl9r5.

[Che+24d]   W. Chen, O. Mees, A. Kumar, and S. Levine. "Vision-language models provide promptable representations for reinforcement learning". In: *arXiv [cs.LG]* (Feb. 2024). URL: http://arxiv.org/abs/2402.02651.

[Chr19]   P. Christodoulou. "Soft Actor-Critic for discrete action settings". In: *arXiv [cs.LG]* (Oct. 2019). URL: http://arxiv.org/abs/1910.07207.

[Chu+18]   K. Chua, R. Calandra, R. McAllister, and S. Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *NIPS*. 2018. URL: http://arxiv.org/abs/1805.12114.

[CL11]   O. Chapelle and L. Li. "An empirical evaluation of Thompson sampling". In: *NIPS*. 2011.

[CMS07]   B. Colson, P. Marcotte, and G. Savard. "An overview of bilevel optimization". en. In: *Ann. Oper. Res.* 153.1 (Sept. 2007), pp. 235–256. URL: https://link.springer.com/article/10.1007/s10479-007-0176-2.

[Cob+19]   K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. "Quantifying Generalization in Reinforcement Learning". en. In: *ICML*. May 2019, pp. 1282–1289. URL: https://proceedings.mlr.press/v97/cobbe19a.html.

[Col+22]   C. Colas, T. Karch, O. Sigaud, and P.-Y. Oudeyer. "Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: A short survey". en. In: *JAIR* 74 (July 2022), pp. 1159–1199. URL: https://www.jair.org/index.php/jair/article/view/13554.

[CS04]   I. Csiszár and P. C. Shields. "Information theory and statistics: A tutorial". In: (2004).

[Csi67]   I. Csiszar. "Information-Type Measures of Difference of Probability Distributions and Indirect Observations". In: *Studia Scientiarum Mathematicarum Hungarica* 2 (1967), pp. 299–318.

[CVRM23]   F. Che, G. Vasan, and A Rupam Mahmood. "Correcting discount-factor mismatch in on-policy policy gradient methods". en. In: *ICML*. PMLR, July 2023, pp. 4218–4240. URL: https://proceedings.mlr.press/v202/che23a.html.

[Dab+17]   W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. "Distributional reinforcement learning with quantile regression". In: *arXiv [cs.AI]* (Oct. 2017). URL: http://arxiv.org/abs/1710.10044.

[Dab+18]   W. Dabney, G. Ostrovski, D. Silver, and R. Munos. "Implicit quantile networks for distributional reinforcement learning". In: *arXiv [cs.LG]* (June 2018). URL: http://arxiv.org/abs/1806.06923.

[Dan+16]   C. Daniel, H. van Hoof, J. Peters, and G. Neumann. "Probabilistic inference for determining options in reinforcement learning". en. In: *Mach. Learn.* 104.2-3 (Sept. 2016), pp. 337–357. URL: https://link.springer.com/article/10.1007/s10994-016-5580-x.

[Day93]   P. Dayan. "Improving generalization for temporal difference learning: The successor representation". en. In: *Neural Comput.* 5.4 (July 1993), pp. 613–624. URL: https://ieeexplore.ieee.org/abstract/document/6795455.

[DFR15]   M. P. Deisenroth, D. Fox, and C. E. Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". en. In: *IEEE PAMI* 37.2 (2015), pp. 408–423. URL: http://dx.doi.org/10.1109/TPAMI.2013.218.

[DH92]   P. Dayan and G. E. Hinton. "Feudal Reinforcement Learning". In: *NIPS* 5 (1992). URL: https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

[Die00]   T. G. Dietterich. "Hierarchical reinforcement learning with the MAXQ value function decomposition". en. In: *JAIR* 13 (Nov. 2000), pp. 227–303. URL: https://www.jair.org/index.php/jair/article/view/10266.

[Die+07]   M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber. "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control". In: *Lecture Notes in Control and Inform. Sci.* 340 (2007). URL: https://www.researchgate.net/publication/29603798_Fast_Direct_Multiple_Shooting_Algorithms_for_Optimal_Robot_Control.

[DMKM22]   G. Duran-Martin, A. Kara, and K. Murphy. "Efficient Online Bayesian Inference for Neural Bandits". In: *AISTATS*. 2022. URL: http://arxiv.org/abs/2112.00195.

[D'O+22]   P. D'Oro, M. Schwarzer, E. Nikishin, P.-L. Bacon, M. G. Bellemare, and A. Courville. "Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier". In: *Deep Reinforcement Learning Workshop NeurIPS 2022*. Dec. 2022. URL: https://openreview.net/pdf?id=4GBGwVIEYJ.

[DOB21]   W. Dabney, G. Ostrovski, and A. Barreto. "Temporally-Extended epsilon-Greedy Exploration". In: *ICLR*. 2021. URL: https://openreview.net/pdf?id=ONBPHFZ7zG4.

[DR11]   M. P. Deisenroth and C. E. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *ICML*. 2011. URL: http://www.icml-2011.org/papers/323_icmlpaper.pdf.

[Du+21]   C. Du, Z. Gao, S. Yuan, L. Gao, Z. Li, Y. Zeng, X. Zhu, J. Xu, K. Gai, and K.-C. Lee. "Exploration in Online Advertising Systems with Deep Uncertainty-Aware Learning". In: *KDD*. KDD '21. Association for Computing Machinery, 2021, pp. 2792–2801. URL: https://doi.org/10.1145/3447548.3467089.

[Duf02]   M. Duff. "Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes". PhD thesis. U. Mass. Dept. Comp. Sci., 2002. URL: http://envy.cs.umass.edu/People/duff/diss.html.

[Dum+24]   V. Dumoulin, D. D. Johnson, P. S. Castro, H. Larochelle, and Y. Dauphin. "A density estimation perspective on learning from pairwise human preferences". In: *Trans. on Machine Learning Research* 2024 (2024). URL: https://openreview.net/pdf?id=YH3oERVYjF.

[DVRZ22] S. Dong, B. Van Roy, and Z. Zhou. "Simple Agent, Complex Environment: Efficient Reinforcement Learning with Agent States". In: *J. Mach. Learn. Res.* (2022). URL: https://www.jmlr.org/papers/v23/21-0773.html.

[DWS12] T. Degris, M. White, and R. S. Sutton. "Off-Policy Actor-Critic". In: *ICML*. 2012. URL: http://arxiv.org/abs/1205.4839.

[Eco+19] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. "Go-Explore: a New Approach for Hard-Exploration Problems". In: (2019). arXiv: 1901.10995 [cs.LG]. URL: http://arxiv.org/abs/1901.10995.

[Eco+21] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. "First return, then explore". en. In: *Nature* 590.7847 (Feb. 2021), pp. 580–586. URL: https://www.nature.com/articles/s41586-020-03157-9.

[Emm+21] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. "RvS: What is essential for offline RL via Supervised Learning?" In: *arXiv [cs.LG]* (Dec. 2021). URL: http://arxiv.org/abs/2112.10751.

[ESL21] B. Eysenbach, R. Salakhutdinov, and S. Levine. "C-Learning: Learning to Achieve Goals via Recursive Classification". In: *ICLR*. 2021. URL: https://openreview.net/pdf?id=tc5qisoB-C.

[Esp+18] L. Espeholt et al. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". en. In: *ICML*. PMLR, July 2018, pp. 1407–1416. URL: https://proceedings.mlr.press/v80/espeholt18a.html.

[Eys+20] B. Eysenbach, X. Geng, S. Levine, and R. Salakhutdinov. "Rewriting History with Inverse RL: Hindsight Inference for Policy Improvement". In: *NIPS*. Feb. 2020.

[Eys+21] B. Eysenbach, A. Khazatsky, S. Levine, and R. Salakhutdinov. "Mismatched No More: Joint Model-Policy Optimization for Model-Based RL". In: (2021). arXiv: 2110.02758 [cs.LG]. URL: http://arxiv.org/abs/2110.02758.

[Eys+22] B. Eysenbach, A. Khazatsky, S. Levine, and R. Salakhutdinov. "Mismatched No More: Joint Model-Policy Optimization for Model-Based RL". In: *NIPS*. 2022.

[Far+18] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson. "TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning". In: *ICLR*. Feb. 2018. URL: https://openreview.net/pdf?id=H1dh6Ax0Z.

[Far+23] J. Farebrother, J. Greaves, R. Agarwal, C. Le Lan, R. Goroshin, P. S. Castro, and M. G. Bellemare. "Proto-Value Networks: Scaling Representation Learning with Auxiliary Tasks". In: *ICLR*. 2023. URL: https://openreview.net/pdf?id=oGDKSt9JrZi.

[Far+24] J. Farebrother et al. "Stop regressing: Training value functions via classification for scalable deep RL". In: *arXiv [cs.LG]* (Mar. 2024). URL: http://arxiv.org/abs/2403.03950.

[FC24] J. Farebrother and P. S. Castro. "CALE: Continuous Arcade Learning Environment". In: *NIPS*. Oct. 2024. URL: https://arxiv.org/abs/2410.23810.

[FG21] S. Fujimoto and S. s. Gu. "A Minimalist Approach to Offline Reinforcement Learning". In: *NIPS*. Vol. 34. Dec. 2021, pp. 20132–20145. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/a8166da05c5a094f7dc03724b41886e5-Paper.pdf.

[FHM18] S. Fujimoto, H. van Hoof, and D. Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *ICLR*. 2018. URL: http://arxiv.org/abs/1802.09477.

[FL+18] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3 (2018). URL: http://arxiv.org/abs/1811.12560.

[FLA16] C. Finn, S. Levine, and P. Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *ICML*. 2016, pp. 49–58.

[FLL18]   J. Fu, K. Luo, and S. Levine. "Learning Robust Rewards with Adverserial Inverse Reinforcement Learning". In: *ICLR*. 2018.

[For+18]   M. Fortunato et al. "Noisy Networks for Exploration". In: *ICLR*. 2018. URL: http://arxiv.org/abs/1706.10295.

[FPP04]   N. Ferns, P. Panangaden, and D. Precup. "Metrics for finite Markov decision processes". en. In: *UAI*. 2004. URL: https://dl.acm.org/doi/10.5555/1036843.1036863.

[Fra+24]   B. Frauenknecht, A. Eisele, D. Subhasish, F. Solowjow, and S. Trimpe. "Trust the Model Where It Trusts Itself - Model-Based Actor-Critic with Uncertainty-Aware Rollout Adaption". In: *ICML*. June 2024. URL: https://openreview.net/pdf?id=N0ntTjTfHb.

[Fre+24]   B. Freed, T. Wei, R. Calandra, J. Schneider, and H. Choset. "Unifying Model-Based and Model-Free Reinforcement Learning with Equivalent Policy Sets". In: *RL Conference*. 2024. URL: https://rlj.cs.umass.edu/2024/papers/RLJ_RLC_2024_37.pdf.

[Fri03]   K. Friston. "Learning and inference in the brain". en. In: *Neural Netw.* 16.9 (2003), pp. 1325–1352. URL: http://dx.doi.org/10.1016/j.neunet.2003.06.005.

[Fri09]   K. Friston. "The free-energy principle: a rough guide to the brain?" en. In: *Trends Cogn. Sci.* 13.7 (2009), pp. 293–301. URL: http://dx.doi.org/10.1016/j.tics.2009.04.005.

[FS+19]   H Francis Song et al. "V-MPO: On-Policy Maximum a Posteriori Policy Optimization for Discrete and Continuous Control". In: *arXiv [cs.AI]* (Sept. 2019). URL: http://arxiv.org/abs/1909.12238.

[FSW23]   M. Fellows, M. J. A. Smith, and S. Whiteson. "Why Target Networks Stabilise Temporal Difference Methods". en. In: *ICML*. PMLR, July 2023, pp. 9886–9909. URL: https://proceedings.mlr.press/v202/fellows23a.html.

[Fu15]   M. Fu, ed. *Handbook of Simulation Optimization*. 1st ed. Springer-Verlag New York, 2015. URL: http://www.springer.com/us/book/9781493913831.

[Fu+20]   J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*. arXiv:2004.07219. 2020.

[Fuj+19]   S. Fujimoto, E. Conti, M. Ghavamzadeh, and J. Pineau. "Benchmarking batch deep reinforcement learning algorithms". In: *Deep RL Workshop NeurIPS*. Oct. 2019. URL: https://arxiv.org/abs/1910.01708.

[Gal+24]   M. Gallici, M. Fellows, B. Ellis, B. Pou, I. Masmitja, J. N. Foerster, and M. Martin. "Simplifying deep temporal difference learning". In: *ICML*. July 2024.

[Gar23]   R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023. URL: https://bayesoptbook.com/.

[GBS22]   C. Grimm, A. Barreto, and S. Singh. "Approximate Value Equivalence". In: *NIPS*. Oct. 2022. URL: https://openreview.net/pdf?id=S2Awu3Zn04v.

[GDG03]   R. Givan, T. Dean, and M. Greig. "Equivalence notions and model minimization in Markov decision processes". en. In: *Artif. Intell.* 147.1-2 (July 2003), pp. 163–223. URL: https://www.sciencedirect.com/science/article/pii/S0004370202003764.

[GDWF22]   J. Grudzien, C. A. S. De Witt, and J. Foerster. "Mirror Learning: A Unifying Framework of Policy Optimisation". In: *ICML*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 7825–7844. URL: https://proceedings.mlr.press/v162/grudzien22a/grudzien22a.pdf.

[Ger18]   S. J. Gershman. "Deconstructing the human algorithms for exploration". en. In: *Cognition* 173 (Apr. 2018), pp. 34–42. URL: https://www.sciencedirect.com/science/article/abs/pii/S0010027717303359.

[Ger19]   S. J. Gershman. "What does the free energy principle tell us about the brain?" In: *Neurons, Behavior, Data Analysis, and Theory* (2019). URL: http://arxiv.org/abs/1901.07945.

[GGN22]   S. K. S. Ghasemipour, S. S. Gu, and O. Nachum. "Why So Pessimistic? Estimating Uncertainties for Offline RL through Ensembles, and Why Their Independence Matters". In: *NIPS*. Oct. 2022. URL: https://openreview.net/pdf?id=z64kN1h1-rR.

[Ghi+20]   S. Ghiassian, A. Patterson, S. Garg, D. Gupta, A. White, and M. White. "Gradient temporal-difference learning with Regularized Corrections". In: *ICML*. July 2020.

[Gho+21]   D. Ghosh, J. Rahme, A. Kumar, A. Zhang, R. P. Adams, and S. Levine. "Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability". In: *NIPS*. Vol. 34. Dec. 2021, pp. 25502–25515. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/d5ff135377d39f1de7372c95c74dd962-Paper.pdf.

[Ghu+22]   R. Ghugare, H. Bharadhwaj, B. Eysenbach, S. Levine, and R. Salakhutdinov. "Simplifying Model-based RL: Learning Representations, Latent-space Models, and Policies with One Objective". In: *ICLR*. Sept. 2022. URL: https://openreview.net/forum?id=MQcmfgRxf7a.

[Git89]   J. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley, 1989.

[GK19]   L. Graesser and W. L. Keng. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. en. 1 edition. Addison-Wesley Professional, 2019. URL: https://www.amazon.com/Deep-Reinforcement-Learning-Python-Hands/dp/0135172381.

[GM+24]   J. Grau-Moya et al. "Learning Universal Predictors". In: *arXiv [cs.LG]* (Jan. 2024). URL: https://arxiv.org/abs/2401.14953.

[Gor95]   G. J. Gordon. "Stable Function Approximation in Dynamic Programming". In: *ICML*. 1995, pp. 261–268.

[Gra+10]   T. Graepel, J. Quinonero-Candela, T. Borchert, and R. Herbrich. "Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine". In: *ICML*. 2010.

[Gri+20]   C. Grimm, A. Barreto, S. Singh, and D. Silver. "The Value Equivalence Principle for Model-Based Reinforcement Learning". In: *NIPS* 33 (2020), pp. 5541–5552. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/3bb585ea00014b0e3ebe4c6dd165a358-Paper.pdf.

[Gul+20]   C. Gulcehre et al. *RL Unplugged: Benchmarks for Offline Reinforcement Learning*. arXiv:2006.13888. 2020.

[Guo+22a]   Z. D. Guo et al. "BYOL-Explore: Exploration by Bootstrapped Prediction". In: *Advances in Neural Information Processing Systems*. Oct. 2022. URL: https://openreview.net/pdf?id=qHGCH75usg.

[Guo+22b]   Z. D. Guo et al. "BYOL-Explore: Exploration by bootstrapped prediction". In: *NIPS*. June 2022. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/ced0d3b92bb83b15c43ee32c7f57d Abstract-Conference.html.

[GZG19]   S. K. S. Ghasemipour, R. S. Zemel, and S. Gu. "A Divergence Minimization Perspective on Imitation Learning Methods". In: *CORL*. 2019, pp. 1259–1277.

[Haa+18a]   T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *ICML*. 2018. URL: http://arxiv.org/abs/1801.01290.

[Haa+18b]   T. Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: (2018). arXiv: 1812.05905 [cs.LG]. URL: http://arxiv.org/abs/1812.05905.

[Haf+19]   D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. "Learning Latent Dynamics for Planning from Pixels". In: *ICML*. 2019. URL: http://arxiv.org/abs/1811.04551.

[Haf+20]   D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. "Dream to Control: Learning Behaviors by Latent Imagination". In: *ICLR*. 2020. URL: https://openreview.net/forum?id=S1lOTC4tDS.

[Haf+21]   D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. "Mastering Atari with discrete world models". In: *ICLR*. 2021.

[Haf+23]   D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. "Mastering Diverse Domains through World Models". In: *arXiv [cs.AI]* (Jan. 2023). URL: http://arxiv.org/abs/2301.04104.

[Han+19]   S. Hansen, W. Dabney, A. Barreto, D. Warde-Farley, T. Van de Wiele, and V. Mnih. "Fast Task Inference with Variational Intrinsic Successor Features". In: *ICLR*. Sept. 2019. URL: https://openreview.net/pdf?id=BJeAHkrYDS.

[Har+18]   J. Harb, P.-L. Bacon, M. Klissarov, and D. Precup. "When waiting is not an option: Learning options with a deliberation cost". en. In: *AAAI* 32.1 (Apr. 2018). URL: https://ojs.aaai.org/index.php/AAAI/article/view/11831.

[Has10]   H. van Hasselt. "Double Q-learning". In: *NIPS*. Ed. by J. D. Lafferty, C. K. I. Williams, J Shawe-Taylor, R. S. Zemel, and A Culotta. Curran Associates, Inc., 2010, pp. 2613–2621. URL: http://papers.nips.cc/paper/3964-double-q-learning.pdf.

[Has+16]   H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver. "Learning values across many orders of magnitude". In: *NIPS*. Feb. 2016.

[HDCM15]   A. Hallak, D. Di Castro, and S. Mannor. "Contextual Markov decision processes". In: *arXiv [stat.ML]* (Feb. 2015). URL: http://arxiv.org/abs/1502.02259.

[HE16]   J. Ho and S. Ermon. "Generative Adversarial Imitation Learning". In: *NIPS*. 2016, pp. 4565–4573.

[Hes+18]   M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *AAAI*. 2018. URL: http://arxiv.org/abs/1710.02298.

[Hes+19]   M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt. "Multi-task deep reinforcement learning with PopArt". In: *AAAI*. 2019.

[HGS16]   H. van Hasselt, A. Guez, and D. Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *AAAI*. AAAI'16. AAAI Press, 2016, pp. 2094–2100. URL: http://dl.acm.org/citation.cfm?id=3016100.3016191.

[HHA19]   H. van Hasselt, M. Hessel, and J. Aslanides. "When to use parametric models in reinforcement learning?" In: *NIPS*. 2019. URL: http://arxiv.org/abs/1906.05243.

[HL04]   D. R. Hunter and K. Lange. "A Tutorial on MM Algorithms". In: *The American Statistician* 58 (2004), pp. 30–37.

[HL20]   O. van der Himst and P. Lanillos. "Deep active inference for partially observable MDPs". In: *ECML workshop on active inference*. Sept. 2020. URL: https://arxiv.org/abs/2009.03622.

[HM20]   M. Hosseini and A. Maida. "Hierarchical Predictive Coding Models in a Deep-Learning Framework". In: (2020). arXiv: 2005.03230 [cs.CV]. URL: http://arxiv.org/abs/2005.03230.

[Hon+10]   A. Honkela, T. Raiko, M. Kuusela, M. Tornio, and J. Karhunen. "Approximate Riemannian Conjugate Gradient Learning for Fixed-Form Variational Bayes". In: *JMLR* 11.Nov (2010), pp. 3235–3268. URL: http://www.jmlr.org/papers/volume11/honkela10a/honkela10a.pdf.

[Hon+23]   M. Hong, H.-T. Wai, Z. Wang, and Z. Yang. "A two-timescale stochastic algorithm framework for bilevel optimization: Complexity analysis and application to actor-critic". en. In: *SIAM J. Optim.* 33.1 (Mar. 2023), pp. 147–180. URL: https://epubs.siam.org/doi/10.1137/20M1387341.

[Hou+11]   N. Houlsby, F. Huszár, Z. Ghahramani, and M. Lengyel. "Bayesian active learning for classification and preference learning". In: *arXiv [stat.ML]* (Dec. 2011). URL: http://arxiv.org/abs/1112.5745.

[HQC24]   M. Hutter, D. Quarel, and E. Catt. *An introduction to universal artificial intelligence*. Chapman and Hall, 2024. URL: http://www.hutter1.net/ai/uaibook2.htm.

[HR11]   R. Hafner and M. Riedmiller. "Reinforcement learning in feedback control: Challenges and benchmarks from technical process control". en. In: *Mach. Learn.* 84.1-2 (July 2011), pp. 137–169. URL: https://link.springer.com/article/10.1007/s10994-011-5235-x.

[HR17]   C. Hoffmann and P. Rostalski. "Linear Optimal Control on Factor Graphs — A Message Passing Perspective". In: *Intl. Federation of Automatic Control* 50.1 (2017), pp. 6314–6319. URL: https://www.sciencedirect.com/science/article/pii/S2405896317313800.

[HS18]   D. Ha and J. Schmidhuber. "World Models". In: *NIPS*. 2018. URL: http://arxiv.org/abs/1803.10122.

[HSW22a]   N. A. Hansen, H. Su, and X. Wang. "Temporal Difference Learning for Model Predictive Control". en. In: *ICML*. PMLR, June 2022, pp. 8387–8406. URL: https://proceedings.mlr.press/v162/hansen22a.html.

[HSW22b]   N. A. Hansen, H. Su, and X. Wang. "Temporal Difference Learning for Model Predictive Control". en. In: *ICML*. PMLR, June 2022, pp. 8387–8406. URL: https://proceedings.mlr.press/v162/hansen22a.html.

[HTB18]   G. Z. Holland, E. J. Talvitie, and M. Bowling. "The effect of planning shape on Dyna-style planning in high-dimensional state spaces". In: *arXiv [cs.AI]* (June 2018). URL: http://arxiv.org/abs/1806.01825.

[Hu+20]   Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan. "Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping". In: *NIPS* 33 (2020), pp. 15931–15941. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/b710915795b9e9c02cf10d6d2bdb688c-Paper.pdf.

[Hub+21]   T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver. "Learning and planning in complex action spaces". In: *arXiv [cs.LG]* (Apr. 2021). URL: http://arxiv.org/abs/2104.06303.

[Hut05]   M. Hutter. *Universal Artificial Intelligence: Sequential Decisions Based On Algorithmic Probability*. en. 2005th ed. Springer, 2005. URL: http://www.hutter1.net/ai/uaibook.htm.

[Ich+23]   B. Ichter et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". en. In: *Conference on Robot Learning*. PMLR, Mar. 2023, pp. 287–318. URL: https://proceedings.mlr.press/v205/ichter23a.html.

[ID19]   S. Ivanov and A. D'yakonov. "Modern Deep Reinforcement Learning algorithms". In: *arXiv [cs.LG]* (June 2019). URL: http://arxiv.org/abs/1906.10025.

[IW18]   E. Imani and M. White. "Improving Regression Performance with Distributional Losses". en. In: *ICML*. PMLR, July 2018, pp. 2157–2166. URL: https://proceedings.mlr.press/v80/imani18a.html.

[Jae00]   H Jaeger. "Observable operator models for discrete stochastic time series". en. In: *Neural Comput.* 12.6 (June 2000), pp. 1371–1398. URL: https://direct.mit.edu/neco/article-pdf/12/6/1371/814514/089976600300015411.pdf.

[Jan+19a] M. Janner, J. Fu, M. Zhang, and S. Levine. "When to Trust Your Model: Model-Based Policy Optimization". In: *NIPS*. 2019. URL: http://arxiv.org/abs/1906.08253.

[Jan+19b] D. Janz, J. Hron, P. Mazur, K. Hofmann, J. M. Hernández-Lobato, and S. Tschiatschek. "Successor Uncertainties: Exploration and Uncertainty in Temporal Difference Learning". In: *NIPS*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/1b113258af3968aaf3969ca67e744ff8-Paper.pdf.

[Jan+22]   M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine. "Planning with Diffusion for Flexible Behavior Synthesis". In: *ICML*. May 2022. URL: http://arxiv.org/abs/2205.09991.

[Jar+23]   D. Jarrett, C. Tallec, F. Altché, T. Mesnard, R. Munos, and M. Valko. "Curiosity in Hindsight: Intrinsic Exploration in Stochastic Environments". In: *ICML*. June 2023. URL: https://openreview.net/pdf?id=fIH2G4fnSy.

[JCM24]   M. Jones, P. Chang, and K. Murphy. "Bayesian online natural gradient (BONG)". In: May 2024. URL: http://arxiv.org/abs/2405.19681.

[JGP16]   E. Jang, S. Gu, and B. Poole. "Categorical Reparameterization with Gumbel-Softmax". In: (2016). arXiv: 1611.01144 [stat.ML]. URL: http://arxiv.org/abs/1611.01144.

[Jia+15]   N. Jiang, A. Kulesza, S. Singh, and R. Lewis. "The Dependence of Effective Planning Horizon on Model Accuracy". en. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, May 2015, pp. 1181–1189. URL: https://dl.acm.org/doi/10.5555/2772879.2773300.

[Jin+22]   L. Jing, P. Vincent, Y. LeCun, and Y. Tian. "Understanding Dimensional Collapse in Contrastive Self-supervised Learning". In: *ICLR*. 2022. URL: https://openreview.net/forum?id=YevsQ05DEN7.

[JLL21]   M. Janner, Q. Li, and S. Levine. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". In: *NIPS*. June 2021.

[JM70]   D. H. Jacobson and D. Q. Mayne. *Differential Dynamic Programming*. Elsevier Press, 1970.

[JML20]   M. Janner, I. Mordatch, and S. Levine. "Gamma-Models: Generative Temporal Difference Learning for Infinite-Horizon Prediction". In: *NIPS*. Vol. 33. 2020, pp. 1724–1735. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/12ffb0968f2f56e51a59a6beb37b2859-Paper.pdf.

[Jor+24]   S. M. Jordan, A. White, B. C. da Silva, M. White, and P. S. Thomas. "Position: Benchmarking is Limited in Reinforcement Learning Research". In: *ICML*. June 2024. URL: https://arxiv.org/abs/2406.16241.

[JSJ94]   T. Jaakkola, S. Singh, and M. Jordan. "Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems". In: *NIPS*. 1994.

[KAG19]   A. Kirsch, J. van Amersfoort, and Y. Gal. "BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning". In: *NIPS*. 2019. URL: http://arxiv.org/abs/1906.08158.

[Kai+19]   L. Kaiser et al. "Model-based reinforcement learning for Atari". In: *arXiv [cs.LG]* (Mar. 2019). URL: http://arxiv.org/abs/1903.00374.

[Kak01]   S. M. Kakade. "A Natural Policy Gradient". In: *NIPS*. Vol. 14. 2001. URL: https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[Kal+18]   D. Kalashnikov et al. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *CORL*. 2018. URL: http://arxiv.org/abs/1806.10293.

[Kap+18]   S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. "Recurrent Experience Replay in Distributed Reinforcement Learning". In: *ICLR*. Sept. 2018. URL: https://openreview.net/pdf?id=r1lyTjAqYX.

[Kap+22]   S. Kapturowski, V. Campos, R. Jiang, N. Rakicevic, H. van Hasselt, C. Blundell, and A. P. Badia. "Human-level Atari 200x faster". In: *ICLR*. Sept. 2022. URL: https://openreview.net/pdf?id=JtC6yOHRoJJ.

[Kau+23]   T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier. "A survey of reinforcement learning from human feedback". In: *arXiv [cs.LG]* (Dec. 2023). URL: http://arxiv.org/abs/2312.14925.

[KB09]   G. Konidaris and A. Barto. "Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining". In: *Advances in Neural Information Processing Systems* 22 (2009). URL: https://proceedings.neurips.cc/paper_files/paper/2009/file/e0cf1f47118daebc5b16269099ad7347-Paper.pdf.

[KD18]   S. Kamthe and M. P. Deisenroth. "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: *AISTATS*. 2018. URL: http://proceedings.mlr.press/v84/kamthe18a/kamthe18a.pdf.

[Ke+19]   L. Ke, S. Choudhury, M. Barnes, W. Sun, G. Lee, and S. Srinivasa. *Imitation Learning as f-Divergence Minimization*. arXiv:1905.12888. 2019.

[KGO12]   H. J. Kappen, V. Gómez, and M. Opper. "Optimal control as a graphical model inference problem". In: *Mach. Learn.* 87.2 (2012), pp. 159–182. URL: https://doi.org/10.1007/s10994-012-5278-7.

[Khe+20] K. Khetarpal, Z. Ahmed, G. Comanici, D. Abel, and D. Precup. "What can I do here? A Theory of Affordances in Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by H. D. Iii and A. Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 5243–5253. URL: https://proceedings.mlr.press/v119/khetarpal20a.html.

[Kid+20] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. "MOReL: Model-Based Offline Reinforcement Learning". In: *NIPS*. Vol. 33. 2020, pp. 21810–21823. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/f7efa4f864ae9b88d43527f4b14f750f-Paper.pdf.

[Kir+21] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. "A survey of zero-shot generalisation in deep Reinforcement Learning". In: *JAIR* (Nov. 2021). URL: http://jair.org/index.php/jair/article/view/14174.

[KLC98]   L. P. Kaelbling, M. Littman, and A. Cassandra. "Planning and acting in Partially Observable Stochastic Domains". In: *AIJ* 101 (1998).

[Kli+24] M. Klissarov, P. D'Oro, S. Sodhani, R. Raileanu, P.-L. Bacon, P. Vincent, A. Zhang, and M. Henaff. "Motif: Intrinsic motivation from artificial intelligence feedback". In: *ICLR*. 2024.

[KLP11]   L. P. Kaelbling and T Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *ICRA*. 2011, pp. 1470–1477. URL: http://dx.doi.org/10.1109/ICRA.2011.5980391.

[Koz+21] T. Kozuno, Y. Tang, M. Rowland, R Munos, S. Kapturowski, W. Dabney, M. Valko, and D. Abel. "Revisiting Peng's Q-lambda for modern reinforcement learning". In: *ICML* 139 (Feb. 2021). Ed. by M. Meila and T. Zhang, pp. 5794–5804. URL: https://proceedings.mlr.press/v139/kozuno21a/kozuno21a.pdf.

[KP19]   K. Khetarpal and D. Precup. "Learning options with interest functions". en. In: *AAAI* 33.01 (July 2019), pp. 9955–9956. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5114.

[KPL19]   A. Kumar, X. B. Peng, and S. Levine. "Reward-Conditioned Policies". In: *arXiv [cs.LG]* (Dec. 2019). URL: http://arxiv.org/abs/1912.13465.

[KS02]   M. Kearns and S. Singh. "Near-Optimal Reinforcement Learning in Polynomial Time". en. In: *MLJ* 49.2/3 (Nov. 2002), pp. 209–232. URL: https://link.springer.com/article/10.1023/A:1017984413808.

[KS06]   L. Kocsis and C. Szepesvári. "Bandit Based Monte-Carlo Planning". In: *ECML*. 2006, pp. 282–293.

[KSS23]   T. Kneib, A. Silbersdorff, and B. Säfken. "Rage Against the Mean – A Review of Distributional Regression Approaches". In: *Econometrics and Statistics* 26 (Apr. 2023), pp. 99–123. URL: https://www.sciencedirect.com/science/article/pii/S2452306221000824.

[Kum+19] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: *NIPS*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c2073ffa77b5357a498057413bb09d3a-Paper.pdf.

[Kum+20] A. Kumar, A. Zhou, G. Tucker, and S. Levine. "Conservative Q-Learning for Offline Reinforcement Learning". In: *NIPS*. June 2020.

[Kum+23]   A. Kumar, R. Agarwal, X. Geng, G. Tucker, and S. Levine. "Offline Q-Learning on Diverse Multi-Task Data Both Scales And Generalizes". In: *ICLR*. 2023. URL: http://arxiv.org/abs/2211.15144.

[Kum+24]   S. Kumar, H. J. Jeon, A. Lewandowski, and B. Van Roy. "The Need for a Big World Simulator: A Scientific Challenge for Continual Learning". In: *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*. July 2024. URL: https://openreview.net/pdf?id=10XMwt1nMJ.

[Kur+19]   T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. "Model-Ensemble Trust-Region Policy Optimization". In: *ICLR*. 2019. URL: http://arxiv.org/abs/1802.10592.

[LA21]   H. Liu and P. Abbeel. "APS: Active Pretraining with Successor Features". en. In: *ICML*. PMLR, July 2021, pp. 6736–6747. URL: https://proceedings.mlr.press/v139/liu21b.html.

[Lai+21]   H. Lai, J. Shen, W. Zhang, Y. Huang, X. Zhang, R. Tang, Y. Yu, and Z. Li. "On effective scheduling of model-based reinforcement learning". In: *NIPS* 34 (Nov. 2021). Ed. by M Ranzato, A Beygelzimer, Y Dauphin, P. S. Liang, and J. W. Vaughan, pp. 3694–3705. URL: https://proceedings.neurips.cc/paper_files/paper/2021/hash/1e4d36177d71bbb3558e43af9577d70e-Abstract.html.

[Lam+20]   N. Lambert, B. Amos, O. Yadan, and R. Calandra. "Objective Mismatch in Model-based Reinforcement Learning". In: *Conf. on Learning for Dynamics and Control (L4DC)*. Feb. 2020.

[Law+22]   D. Lawson, A. Raventós, A. Warrington, and S. Linderman. "SIXO: Smoothing Inference with Twisted Objectives". In: *NIPS*. June 2022.

[Leh24]   M. Lehmann. "The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations". In: *arXiv [cs.LG]* (Jan. 2024). URL: http://arxiv.org/abs/2401.13662.

[Lev18]   S. Levine. "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review". In: (2018). arXiv: 1805.00909 [cs.LG]. URL: http://arxiv.org/abs/1805.00909.

[Lev+18]   A. Levy, G. Konidaris, R. Platt, and K. Saenko. "Learning Multi-Level Hierarchies with Hindsight". In: *ICLR*. Sept. 2018. URL: https://openreview.net/pdf?id=ryzECoAcY7.

[Lev+20a]   S. Levine, A. Kumar, G. Tucker, and J. Fu. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: (2020). arXiv: 2005.01643 [cs.LG]. URL: http://arxiv.org/abs/2005.01643.

[Lev+20b]   S. Levine, A. Kumar, G. Tucker, and J. Fu. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. arXiv:2005.01643. 2020.

[LG+24]   M. Lázaro-Gredilla, L. Y. Ku, K. P. Murphy, and D. George. "What type of inference is planning?" In: *NIPS*. June 2024.

[LGR12]   S. Lange, T. Gabel, and M. Riedmiller. "Batch reinforcement learning". en. In: *Adaptation, Learning, and Optimization*. Adaptation, learning, and optimization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 45–73. URL: https://link.springer.com/chapter/10.1007/978-3-642-27645-3_2.

[LHC24]   C. Lu, S. Hu, and J. Clune. "Intelligent Go-Explore: Standing on the shoulders of giant foundation models". In: *arXiv [cs.LG]* (May 2024). URL: http://arxiv.org/abs/2405.15143.

[LHS13]   T. Lattimore, M. Hutter, and P. Sunehag. "The Sample-Complexity of General Reinforcement Learning". en. In: *ICML*. PMLR, May 2013, pp. 28–36. URL: https://proceedings.mlr.press/v28/lattimore13.html.

[Li+10]   L. Li, W. Chu, J. Langford, and R. E. Schapire. "A contextual-bandit approach to personalized news article recommendation". In: *WWW*. 2010.

[Li18]   Y. Li. "Deep Reinforcement Learning". In: (2018). arXiv: 1810.06339 [cs.LG]. URL: http://arxiv.org/abs/1810.06339.

[Li23]   S. E. Li. *Reinforcement learning for sequential decision and optimal control*. en. Singapore: Springer Nature Singapore, 2023. URL: https://link.springer.com/book/10.1007/978-981-19-7784-8.

[Li+24]   H. Li, X. Yang, Z. Wang, X. Zhu, J. Zhou, Y. Qiao, X. Wang, H. Li, L. Lu, and J. Dai. "Auto MC-Reward: Automated Dense Reward Design with Large Language Models for Minecraft". In: *CVPR*. 2024, pp. 16426–16435. URL: https://openaccess.thecvf.com/content/CVPR2024/papers/Li_Auto_MC-Reward_Automated_Dense_Reward_Design_with_Large_Language_Models_CVPR_2024_paper.pdf.

[Lil+16]   T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. "Continuous control with deep reinforcement learning". In: *ICLR*. 2016. URL: http://arxiv.org/abs/1509.02971.

[Lin+19]   C. Linke, N. M. Ady, M. White, T. Degris, and A. White. "Adapting behaviour via intrinsic reward: A survey and empirical study". In: *J. Artif. Intell. Res.* (June 2019). URL: http://arxiv.org/abs/1906.07865.

[Lin+24a]   J. Lin, Y. Du, O. Watkins, D. Hafner, P. Abbeel, D. Klein, and A. Dragan. "Learning to model the world with language". In: *ICML*. 2024.

[Lin+24b]   Y.-A. Lin, C.-T. Lee, C.-H. Yang, G.-T. Liu, and S.-H. Sun. "Hierarchical Programmatic Option Framework". In: *NIPS*. Nov. 2024. URL: https://openreview.net/pdf?id=FeCWZviCeP.

[Lin92]   L.-J. Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Mach. Learn.* 8.3-4 (1992), pp. 293–321. URL: https://doi.org/10.1007/BF00992699.

[Lio+22]   V. Lioutas, J. W. Lavington, J. Sefas, M. Niedoba, Y. Liu, B. Zwartsenberg, S. Dabiri, F. Wood, and A. Scibior. "Critic Sequential Monte Carlo". In: *ICLR*. Sept. 2022. URL: https://openreview.net/pdf?id=ObtGcyKmwna.

[LMW24]   B. Li, N. Ma, and Z. Wang. "Rewarded Region Replay (R3) for policy learning with discrete action space". In: *arXiv [cs.LG]* (May 2024). URL: http://arxiv.org/abs/2405.16383.

[Lor24]   J. Lorraine. "Scalable nested optimization for deep learning". In: *arXiv [cs.LG]* (July 2024). URL: http://arxiv.org/abs/2407.01526.

[LÖW21]   T. van de Laar, A. Özçelikkale, and H. Wymeersch. "Application of the Free Energy Principle to Estimation and Control". In: *IEEE Trans. Signal Process.* 69 (2021), pp. 4234–4244. URL: http://dx.doi.org/10.1109/TSP.2021.3095711.

[LPC22]   N. Lambert, K. Pister, and R. Calandra. "Investigating Compounding Prediction Errors in Learned Dynamics Models". In: *arXiv [cs.LG]* (Mar. 2022). URL: http://arxiv.org/abs/2203.09637.

[LR10]   S. Lange and M. Riedmiller. "Deep auto-encoder neural networks in reinforcement learning". en. In: *IJCNN*. IEEE, July 2010, pp. 1–8. URL: https://ieeexplore.ieee.org/abstract/document/5596468.

[LS01]   M. Littman and R. S. Sutton. "Predictive Representations of State". In: *Advances in Neural Information Processing Systems* 14 (2001). URL: https://proceedings.neurips.cc/paper_files/paper/2001/file/1e4d36177d71bbb3558e43af9577d70e-Paper.pdf.

[LS19]   T. Lattimore and C. Szepesvari. *Bandit Algorithms*. Cambridge, 2019.

[Lu+23]   X. Lu, B. Van Roy, V. Dwaracherla, M. Ibrahimi, I. Osband, and Z. Wen. "Reinforcement Learning, Bit by Bit". In: *Found. Trends® Mach. Learn.* (2023). URL: https://www.nowpublishers.com/article/Details/MAL-097.

[LV06]   F. Liese and I. Vajda. "On divergences and informations in statistics and information theory". In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4394–4412.

[LWL06]   L. Li, T. J. Walsh, and M. L. Littman. "Towards a Unified Theory of State Abstraction for MDPs". In: (2006). URL: https://thomasjwalsh.net/pub/aima06Towards.pdf.

[LZZ22]   M. Liu, M. Zhu, and W. Zhang. "Goal-conditioned reinforcement learning: Problems and solutions". In: *IJCAI*. Jan. 2022.

[Ma+24]   Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. "Eureka: Human-Level Reward Design via Coding Large Language Models". In: *ICLR*. 2024.

[Mac+18a]   M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents". In: *J. Artif. Intell. Res.* (2018). URL: http://arxiv.org/abs/1709.06009.

[Mac+18b]   M. C. Machado, C. Rosenbaum, X. Guo, M. Liu, G. Tesauro, and M. Campbell. "Eigenoption Discovery through the Deep Successor Representation". In: *ICLR*. Feb. 2018. URL: https://openreview.net/pdf?id=Bk8ZcAxR-.

[Mac+23]   M. C. Machado, A. Barreto, D. Precup, and M. Bowling. "Temporal Abstraction in Reinforcement Learning with the Successor Representation". In: *JMLR* 24.80 (2023), pp. 1–69. URL: http://jmlr.org/papers/v24/21-1213.html.

[Mad+17]   C. J. Maddison, D. Lawson, G. Tucker, N. Heess, A. Doucet, A. Mnih, and Y. W. Teh. "Particle Value Functions". In: *ICLR Workshop on RL*. Mar. 2017.

[Mae+09]   H. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton. "Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation". In: *NIPS*. Vol. 22. 2009. URL: https://proceedings.neurips.cc/paper_files/paper/2009/file/3a15c7d0bbe60300a39f76f8a5ba6896-Paper.pdf.

[MAF22]   V. Micheli, E. Alonso, and F. Fleuret. "Transformers are Sample-Efficient World Models". In: *ICLR*. Sept. 2022.

[MAF24]   V. Micheli, E. Alonso, and F. Fleuret. "Efficient world models with context-aware tokenization". In: *ICML*. June 2024.

[Maj21]   S. J. Majeed. "Abstractions of general reinforcement learning: An inquiry into the scalability of generally intelligent agents". PhD thesis. ANU, Dec. 2021. URL: https://arxiv.org/abs/2112.13404.

[Man+19]   D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller. "Robust Reinforcement Learning for Continuous Control with Model Misspecification". In: (2019). arXiv: 1906.07516 [cs.LG]. URL: http://arxiv.org/abs/1906.07516.

[Mar10]   J Martens. "Deep learning via Hessian-free optimization". In: *ICML*. 2010. URL: http://www.cs.toronto.edu/~asamir/cifar/HFO_James.pdf.

[Mar16]   J. Martens. "Second-order optimization for neural networks". PhD thesis. Toronto, 2016. URL: http://www.cs.toronto.edu/~jmartens/docs/thesis_phd_martens.pdf.

[Mar20]   J. Martens. "New insights and perspectives on the natural gradient method". In: *JMLR* (2020). URL: http://arxiv.org/abs/1412.1193.

[Mar21]   J. Marino. "Predictive Coding, Variational Autoencoders, and Biological Connections". en. In: *Neural Comput.* 34.1 (2021), pp. 1–44. URL: http://dx.doi.org/10.1162/neco_a_01458.

[Maz+22]   P. Mazzaglia, T. Verbelen, O. Çatal, and B. Dhoedt. "The Free Energy Principle for Perception and Action: A Deep Learning Perspective". en. In: *Entropy* 24.2 (2022). URL: http://dx.doi.org/10.3390/e24020301.

[MBB20]   M. C. Machado, M. G. Bellemare, and M. Bowling. "Count-based exploration with the successor representation". en. In: *AAAI* 34.04 (Apr. 2020), pp. 5125–5133. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5955.

[McM+13] H. B. McMahan, G. Holt, D Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. "Ad click prediction: a view from the trenches". In: *KDD*. 2013, pp. 1222–1230.

[Men+23] W. Meng, Q. Zheng, G. Pan, and Y. Yin. "Off-Policy Proximal Policy Optimization". en. In: *AAAI* 37.8 (June 2023), pp. 9162–9170. URL: https://ojs.aaai.org/index.php/AAAI/article/view/26099.

[Met+17] L. Metz, J. Ibarz, N. Jaitly, and J. Davidson. "Discrete Sequential Prediction of Continuous Actions for Deep RL". In: (2017). arXiv: 1705.05035 [cs.LG]. URL: http://arxiv.org/abs/1705.05035.

[Mey22]   S. Meyn. *Control Systems and Reinforcement Learning*. Cambridge, 2022. URL: https://meyn.ece.ufl.edu/2021/08/01/control-systems-and-reinforcement-learning/.

[MG15]   J. Martens and R. Grosse. "Optimizing Neural Networks with Kronecker-factored Approximate Curvature". In: *ICML*. 2015. URL: http://arxiv.org/abs/1503.05671.

[Mik+20] V. Mikulik, G. Delétang, T. McGrath, T. Genewein, M. Martic, S. Legg, and P. Ortega. "Meta-trained agents implement Bayes-optimal agents". In: *NIPS* 33 (2020), pp. 18691–18703. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/d902c3ce47124c66ce615d5ad9ba304f-Paper.pdf.

[Mil20]   B. Millidge. "Deep Active Inference as Variational Policy Gradients". In: *J. Mathematical Psychology* (2020). URL: http://arxiv.org/abs/1907.03876.

[Mil+20] B. Millidge, A. Tschantz, A. K. Seth, and C. L. Buckley. "On the Relationship Between Active Inference and Control as Inference". In: *International Workshop on Active Inference*. 2020. URL: http://arxiv.org/abs/2006.12964.

[MM90]   D. Q. Mayne and H Michalska. "Receding horizon control of nonlinear systems". In: *IEEE Trans. Automat. Contr.* 35.7 (1990), pp. 814–824.

[MMT17]   C. J. Maddison, A. Mnih, and Y. W. Teh. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables". In: *ICLR*. 2017. URL: http://arxiv.org/abs/1611.00712.

[MMT24]   S. Mannor, Y. Mansour, and A. Tamar. *Reinforcement Learning: Foundations*. 2024. URL: https://sites.google.com/corp/view/rlfoundations/home.

[Mni+15] V. Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[Mni+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *ICML*. 2016. URL: http://arxiv.org/abs/1602.01783.

[Moe+23] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker. "Model-based Reinforcement Learning: A Survey". In: *Foundations and Trends in Machine Learning* 16.1 (2023), pp. 1–118. URL: https://arxiv.org/abs/2006.16712.

[Moh+20] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih. "Monte Carlo Gradient Estimation in Machine Learning". In: *JMLR* 21.132 (2020), pp. 1–62. URL: http://jmlr.org/papers/v21/19-346.html.

[Mor63]   T. Morimoto. "Markov Processes and the H-Theorem". In: *J. Phys. Soc. Jpn.* 18.3 (1963), pp. 328–331. URL: https://doi.org/10.1143/JPSJ.18.328.

[MP+22]   A. Mavor-Parker, K. Young, C. Barry, and L. Griffin. "How to Stay Curious while avoiding Noisy TVs using Aleatoric Uncertainty Estimation". en. In: *ICML*. PMLR, June 2022, pp. 15220–15240. URL: https://proceedings.mlr.press/v162/mavor-parker22a.html.

[MSB21]   B. Millidge, A. Seth, and C. L. Buckley. "Predictive Coding: a Theoretical and Experimental Review". In: (2021). arXiv: 2107.12979 [cs.AI]. URL: http://arxiv.org/abs/2107.12979.

[Mun14]   R. Munos. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: *Foundations and Trends in Machine Learning* 7.1 (2014), pp. 1–129. URL: http://dx.doi.org/10.1561/2200000038.

[Mun+16]   R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *NIPS*. 2016, pp. 1046–1054.

[Mur00]   K. Murphy. *A Survey of POMDP Solution Techniques*. Tech. rep. Comp. Sci. Div., UC Berkeley, 2000. URL: https://www.cs.ubc.ca/~murphyk/Papers/pomdp.pdf.

[Mur23]   K. P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[MWS14]   J. Modayil, A. White, and R. S. Sutton. "Multi-timescale nexting in a reinforcement learning robot". en. In: *Adapt. Behav.* 22.2 (Apr. 2014), pp. 146–160. URL: https://sites.ualberta.ca/~amw8/nexting.pdf.

[Nac+18]   O. Nachum, S. Gu, H. Lee, and S. Levine. "Data-Efficient Hierarchical Reinforcement Learning". In: *NIPS*. May 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/e6384711491713d29bc63fc5eeb5ba4f-Abstract.html.

[Nac+19]   O. Nachum, S. Gu, H. Lee, and S. Levine. "Near-Optimal Representation Learning for Hierarchical Reinforcement Learning". In: *ICLR*. 2019. URL: https://openreview.net/pdf?id=H1emus0qF7.

[Nai+20]   A. Nair, A. Gupta, M. Dalal, and S. Levine. "AWAC: Accelerating Online Reinforcement Learning with Offline Datasets". In: *arXiv [cs.LG]* (June 2020). URL: http://arxiv.org/abs/2006.09359.

[Nak+23]   M. Nakamoto, Y. Zhai, A. Singh, M. S. Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine. "Cal-QL: Calibrated offline RL pre-training for efficient online fine-tuning". In: *arXiv [cs.LG]* (Mar. 2023). URL: http://arxiv.org/abs/2303.05479.

[NHR99]   A. Ng, D. Harada, and S. Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. 1999.

[Ni+24]   T. Ni, B. Eysenbach, E. Seyedsalehi, M. Ma, C. Gehring, A. Mahajan, and P.-L. Bacon. "Bridging State and History Representations: Understanding Self-Predictive RL". In: *ICLR*. Jan. 2024. URL: http://arxiv.org/abs/2401.08898.

[Nik+22]   E. Nikishin, R. Abachi, R. Agarwal, and P.-L. Bacon. "Control-oriented model-based reinforcement learning with implicit differentiation". en. In: *AAAI* 36.7 (June 2022), pp. 7886–7894. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20758.

[NR00]   A. Ng and S. Russell. "Algorithms for inverse reinforcement learning". In: *ICML*. 2000.

[NT20]   C. Nota and P. S. Thomas. "Is the policy gradient a gradient?" In: *Proc. of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020.

[NWJ10]   X Nguyen, M. J. Wainwright, and M. I. Jordan. "Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization". In: *IEEE Trans. Inf. Theory* 56.11 (2010), pp. 5847–5861. URL: http://dx.doi.org/10.1109/TIT.2010.2068870.

[OCD21]   G. Ostrovski, P. S. Castro, and W. Dabney. "The Difficulty of Passive Learning in Deep Reinforcement Learning". In: *NIPS*. Vol. 34. Dec. 2021, pp. 23283–23295. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/c3e0c62ee91db8dc7382bde7419bb573-Paper.pdf.

[OK22]   A. Ororbia and D. Kifer. "The neural coding framework for learning generative models". en. In: *Nat. Commun.* 13.1 (Apr. 2022), p. 2064. URL: https://www.nature.com/articles/s41467-022-29632-7.

[ORVR13]   I. Osband, D. Russo, and B. Van Roy. "(More) Efficient Reinforcement Learning via Posterior Sampling". In: *NIPS*. 2013. URL: http://arxiv.org/abs/1306.0940.

[Osb+19]   I. Osband, B. Van Roy, D. J. Russo, and Z. Wen. "Deep exploration via randomized value functions". In: *JMLR* 20.124 (2019), pp. 1–62. URL: http://jmlr.org/papers/v20/18-339.html.

[Osb+23a]   I. Osband, Z. Wen, S. M. Asghari, V. Dwaracherla, M. Ibrahimi, X. Lu, and B. Van Roy. "Approximate Thompson Sampling via Epistemic Neural Networks". en. In: *UAI*. PMLR, July 2023, pp. 1586–1595. URL: https://proceedings.mlr.press/v216/osband23a.html.

[Osb+23b]   I. Osband, Z. Wen, S. M. Asghari, V. Dwaracherla, M. Ibrahimi, X. Lu, and B. Van Roy. "Epistemic Neural Networks". In: *NIPS*. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/07fbde96bee50f4e09303fd4f877c2f3-Paper-Conference.pdf.

[OSL17]   J. Oh, S. Singh, and H. Lee. "Value Prediction Network". In: *NIPS*. July 2017.

[OT22]   M. Okada and T. Taniguchi. "DreamingV2: Reinforcement learning with discrete world models without reconstruction". en. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2022, pp. 985–991. URL: https://ieeexplore.ieee.org/abstract/document/9981405.

[Ouy+22]   L. Ouyang et al. "Training language models to follow instructions with human feedback". In: (Mar. 2022). arXiv: 2203.02155 [cs.CL]. URL: http://arxiv.org/abs/2203.02155.

[OVR17]   I. Osband and B. Van Roy. "Why is posterior sampling better than optimism for reinforcement learning?" In: *ICML*. 2017, pp. 2701–2710.

[PAG24]   M. Panwar, K. Ahuja, and N. Goyal. "In-context learning through the Bayesian prism". In: *ICLR*. 2024. URL: https://arxiv.org/abs/2306.04891.

[Par+23]   J. S. Park, J. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. "Generative agents: Interactive simulacra of human behavior". en. In: *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, Oct. 2023. URL: https://dl.acm.org/doi/10.1145/3586183.3606763.

[Par+24a]   S. Park, K. Frans, B. Eysenbach, and S. Levine. "OGBench: Benchmarking Offline Goal-Conditioned RL". In: *arXiv [cs.LG]* (Oct. 2024). URL: http://arxiv.org/abs/2410.20092.

[Par+24b]   S. Park, K. Frans, S. Levine, and A. Kumar. "Is value learning really the main bottleneck in offline RL?" In: *NIPS*. June 2024. URL: https://arxiv.org/abs/2406.09329.

[Pat+17]   D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. "Curiosity-driven Exploration by Self-supervised Prediction". In: *ICML*. 2017. URL: http://arxiv.org/abs/1705.05363.

[Pat+22]   S. Pateria, B. Subagdja, A.-H. Tan, and C. Quek. "Hierarchical Reinforcement Learning: A comprehensive survey". en. In: *ACM Comput. Surv.* 54.5 (June 2022), pp. 1–35. URL: https://dl.acm.org/doi/10.1145/3453160.

[Pat+24]   A. Patterson, S. Neumann, M. White, and A. White. "Empirical design in reinforcement learning". In: *JMLR* (2024). URL: http://arxiv.org/abs/2304.01315.

[PB+14]   N. Parikh, S. Boyd, et al. "Proximal algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 127–239.

[Pea84]   J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984. URL: https://dl.acm.org/citation.cfm?id=525.

[Pea94]   B. A. Pearlmutter. "Fast Exact Multiplication by the Hessian". In: *Neural Comput.* 6.1 (1994), pp. 147–160. URL: https://doi.org/10.1162/neco.1994.6.1.147.

[Pen+19]   X. B. Peng, A. Kumar, G. Zhang, and S. Levine. "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning". In: *arXiv [cs.LG]* (Sept. 2019). URL: http://arxiv.org/abs/1910.00177.

[Pic+19]   A. Piche, V. Thomas, C. Ibrahim, Y. Bengio, and C. Pal. "Probabilistic Planning with Sequential Monte Carlo methods". In: *ICLR*. 2019. URL: https://openreview.net/pdf?id=ByetGn0cYX.

[Pis+22]   M. Pislar, D. Szepesvari, G. Ostrovski, D. L. Borsa, and T. Schaul. "When should agents explore?" In: *ICLR*. 2022. URL: https://openreview.net/pdf?id=dEwfxt14bca.

[PKP21]   A. Plaat, W. Kosters, and M. Preuss. "High-Accuracy Model-Based Reinforcement Learning, a Survey". In: (2021). arXiv: 2107.08241 [cs.LG]. URL: http://arxiv.org/abs/2107.08241.

[Pla22]   A. Plaat. *Deep reinforcement learning, a textbook*. Berlin, Germany: Springer, Jan. 2022. URL: https://link.springer.com/10.1007/978-981-19-0638-1.

[PMB22]   K. Paster, S. McIlraith, and J. Ba. "You can't count on luck: Why decision transformers and RvS fail in stochastic environments". In: *arXiv [cs.LG]* (May 2022). URL: http://arxiv.org/abs/2205.15967.

[Pom89]   D. Pomerleau. "ALVINN: An Autonomous Land Vehicle in a Neural Network". In: *NIPS*. 1989, pp. 305–313.

[Pow22]   W. B. Powell. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. en. 1st ed. Wiley, Mar. 2022. URL: https://www.amazon.com/Reinforcement-Learning-Stochastic-Optimization-Sequential/dp/1119815037.

[PR12]   W. B. Powell and I. O. Ryzhov. *Optimal Learning*. Wiley Series in Probability and Statistics. http://optimallearning.princeton.edu/. Hoboken, NJ: Wiley-Blackwell, Mar. 2012. URL: https://castle.princeton.edu/wp-content/uploads/2019/02/Powell-OptimalLearningWileyMarch112018.pdf.

[PS07]   J. Peters and S. Schaal. "Reinforcement Learning by Reward-Weighted Regression for Operational Space Control". In: *ICML*. 2007, pp. 745–750.

[PSS00]   D. Precup, R. S. Sutton, and S. P. Singh. "Eligibility Traces for Off-Policy Policy Evaluation". In: *ICML*. ICML '00. Morgan Kaufmann Publishers Inc., 2000, pp. 759–766. URL: http://dl.acm.org/citation.cfm?id=645529.658134.

[PT87]   C. Papadimitriou and J. Tsitsiklis. "The complexity of Markov decision processes". In: *Mathematics of Operations Research* 12.3 (1987), pp. 441–450.

[Put94]   M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[PW94]   J. Peng and R. J. Williams. "Incremental Multi-Step Q-Learning". In: *Machine Learning Proceedings*. Elsevier, Jan. 1994, pp. 226–232. URL: http://dx.doi.org/10.1016/B978-1-55860-335-6.50035-0.

[QPC21]   J. Queeney, I. C. Paschalidis, and C. G. Cassandras. "Generalized Proximal Policy Optimization with Sample Reuse". In: *NIPS*. Oct. 2021.

[QPC24]   J. Queeney, I. C. Paschalidis, and C. G. Cassandras. "Generalized Policy Improvement algorithms with theoretically supported sample reuse". In: *IEEE Trans. Automat. Contr.* (2024). URL: http://arxiv.org/abs/2206.13714.

[Rab89]   L. R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: *Proc. of the IEEE* 77.2 (1989), pp. 257–286.

[Raf+23]   R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. "Direct Preference Optimization: Your language model is secretly a reward model". In: *arXiv [cs.LG]* (May 2023). URL: http://arxiv.org/abs/2305.18290.

[Raf+24]   R. Rafailov et al. "D5RL: Diverse datasets for data-driven deep reinforcement learning". In: *RLC*. Aug. 2024. URL: https://arxiv.org/abs/2408.08441.

[Raj+17]   A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade. "Towards generalization and simplicity in continuous control". In: *NIPS*. Mar. 2017.

[Rao10]   A. V. Rao. "A Survey of Numerical Methods for Optimal Control". In: *Adv. Astronaut. Sci.* 135.1 (2010). URL: http://dx.doi.org/.

[RB12]   S. Ross and J. A. Bagnell. "Agnostic system identification for model-based reinforcement learning". In: *ICML*. Mar. 2012.

[RB99]   R. P. Rao and D. H. Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". en. In: *Nat. Neurosci.* 2.1 (1999), pp. 79–87. URL: http://dx.doi.org/10.1038/4580.

[Rec19]   B. Recht. "A Tour of Reinforcement Learning: The View from Continuous Control". In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 253–279. URL: http://arxiv.org/abs/1806.09460.

[Ree+22]   S. Reed et al. "A Generalist Agent". In: *TMLR* (May 2022). URL: https://arxiv.org/abs/2205.06175.

[Ren+24]   A. Z. Ren, J. Lidard, L. L. Ankile, A. Simeonov, P. Agrawal, A. Majumdar, B. Burchfiel, H. Dai, and M. Simchowitz. "Diffusion Policy Policy Optimization". In: *arXiv [cs.RO]* (Aug. 2024). URL: http://arxiv.org/abs/2409.00588.

[RFP15]   I. O. Ryzhov, P. I. Frazier, and W. B. Powell. "A new optimal stepsize for approximate dynamic programming". en. In: *IEEE Trans. Automat. Contr.* 60.3 (Mar. 2015), pp. 743–758. URL: https://castle.princeton.edu/Papers/Ryzhov-OptimalStepsizeforADPFeb242015.pdf.

[RGB11]   S. Ross, G. J. Gordon, and J. A. Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *AISTATS*. 2011.

[Rie05]   M. Riedmiller. "Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method". en. In: *ECML*. Lecture notes in computer science. 2005, pp. 317–328. URL: https://link.springer.com/chapter/10.1007/11564096_32.

[Rie+18]   M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg. "Learning by Playing Solving Sparse Reward Tasks from Scratch". en. In: *ICML*. PMLR, July 2018, pp. 4344–4353. URL: https://proceedings.mlr.press/v80/riedmiller18a.html.

[RJ22]   A. Rao and T. Jelvis. *Foundations of Reinforcement Learning with Applications in Finance*. Chapman and Hall/ CRC, 2022. URL: https://github.com/TikhonJelvis/RL-book.

[RK04]   R. Rubinstein and D. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[RLT18]   M. Riemer, M. Liu, and G. Tesauro. "Learning Abstract Options". In: *NIPS* 31 (2018). URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/cdf28f8b7d14ab02d12a2329d71e4079-Paper.pdf.

[RMD22]   J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory, Computation, and Design (2nd ed)*. en. Nob Hill Publishing, LLC, Sept. 2022. URL: https://sites.engineering.ucsb.edu/~jbraw/mpc/MPC-book-2nd-edition-1st-printing.pdf.

[RMK20]   A. Rajeswaran, I. Mordatch, and V. Kumar. "A game theoretic framework for model based reinforcement learning". In: *ICML*. 2020.

[RN19]   S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 4th edition. Prentice Hall, 2019.

[RN94]   G. A. Rummery and M Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. Cambridge Univ. Engineering Dept., 1994. URL: http://dx.doi.org/.

[RR14]   D. Russo and B. V. Roy. "Learning to Optimize via Posterior Sampling". In: *Math. Oper. Res.* 39.4 (2014), pp. 1221–1243.

[RTV12]   K. Rawlik, M. Toussaint, and S. Vijayakumar. "On stochastic optimal control and reinforcement learning by approximate inference". In: *Robotics: Science and Systems VIII*. Robotics: Science and Systems Foundation, 2012. URL: https://blogs.cuit.columbia.edu/zp2130/files/2019/03/On_Stochasitc_Optimal_Control_and_Reinforcement_Learning_by_Approximate_Inference.pdf.

[Rub97]   R. Y. Rubinstein. "Optimization of computer simulation models with rare events". In: *Eur. J. Oper. Res.* 99.1 (1997), pp. 89–112. URL: http://www.sciencedirect.com/science/article/pii/S0377221796003852.

[Rus+18]   D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen. "A Tutorial on Thompson Sampling". In: *Foundations and Trends in Machine Learning* 11.1 (2018), pp. 1–96. URL: http://dx.doi.org/10.1561/2200000070.

[Rus19]   S. Russell. *Human Compatible: Artificial Intelligence and the Problem of Control.* en. Kindle. Viking, 2019. URL: https://www.amazon.com/Human-Compatible-Artificial-Intelligence-Problem-ebook/dp/B07N5J5FTS/ref=zg_bs_3887_4?_encoding=UTF8&psc=1&refRID=0JE0ST011W4K15PTFZAT.

[RW91]   S. Russell and E. Wefald. "Principles of metareasoning". en. In: *Artif. Intell.* 49.1-3 (May 1991), pp. 361–395. URL: http://dx.doi.org/10.1016/0004-3702(91)90015-C.

[Ryu+20]   M. Ryu, Y. Chow, R. Anderson, C. Tjandraatmadja, and C. Boutilier. "CAQL: Continuous Action Q-Learning". In: *ICLR.* 2020. URL: https://openreview.net/forum?id=BkxXe0Etwr.

[Saj+21]   N. Sajid, P. J. Ball, T. Parr, and K. J. Friston. "Active Inference: Demystified and Compared". en. In: *Neural Comput.* 33.3 (Mar. 2021), pp. 674–712. URL: https://web.archive.org/web/20210628163715id_/https://discovery.ucl.ac.uk/id/eprint/10119277/1/Friston_neco_a_01357.pdf.

[Sal+23]   T. Salvatori, A. Mali, C. L. Buckley, T. Lukasiewicz, R. P. N. Rao, K. Friston, and A. Ororbia. "Brain-inspired computational intelligence via predictive coding". In: *arXiv [cs.AI]* (Aug. 2023). URL: http://arxiv.org/abs/2308.07870.

[Sal+24]   T. Salvatori, Y. Song, Y. Yordanov, B. Millidge, L. Sha, C. Emde, Z. Xu, R. Bogacz, and T. Lukasiewicz. "A Stable, Fast, and Fully Automatic Learning Algorithm for Predictive Coding Networks". In: *ICLR.* Oct. 2024. URL: https://openreview.net/pdf?id=RyUvzda8GH.

[SB18]   R. Sutton and A. Barto. *Reinforcement learning: an introduction (2nd edn).* MIT Press, 2018.

[Sch10]   J. Schmidhuber. "Formal Theory of Creativity, Fun, and Intrinsic Motivation". In: *IEEE Trans. Autonomous Mental Development* 2 (2010). URL: http://people.idsia.ch/~juergen/ieeecreative.pdf.

[Sch+15a]   T. Schaul, D. Horgan, K. Gregor, and D. Silver. "Universal Value Function Approximators". en. In: *ICML.* PMLR, June 2015, pp. 1312–1320. URL: https://proceedings.mlr.press/v37/schaul15.html.

[Sch+15b]   J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust Region Policy Optimization". In: *ICML.* 2015. URL: http://arxiv.org/abs/1502.05477.

[Sch+16a]   T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized Experience Replay". In: *ICLR.* 2016. URL: http://arxiv.org/abs/1511.05952.

[Sch+16b]   J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *ICLR.* 2016. URL: http://arxiv.org/abs/1506.02438.

[Sch+17]   J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal Policy Optimization Algorithms". In: (2017). arXiv: 1707.06347 [cs.LG]. URL: http://arxiv.org/abs/1707.06347.

[Sch19]   J. Schmidhuber. "Reinforcement learning Upside Down: Don't predict rewards – just map them to actions". In: *arXiv [cs.AI]* (Dec. 2019). URL: http://arxiv.org/abs/1912.02875.

[Sch+20]   J. Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *Nature* (2020). URL: http://arxiv.org/abs/1911.08265.

[Sch+21] M. Schwarzer, A. Anand, R. Goel, R Devon Hjelm, A. Courville, and P. Bachman. "Data-Efficient Reinforcement Learning with Self-Predictive Representations". In: *ICLR*. 2021. URL: https://openreview.net/pdf?id=uCQfPZwRaUu.

[Sch+23a] I. Schubert, J. Zhang, J. Bruce, S. Bechtle, E. Parisotto, M. Riedmiller, J. T. Springenberg, A. Byravan, L. Hasenclever, and N. Heess. "A Generalist Dynamics Model for Control". In: *arXiv [cs.AI]* (May 2023). URL: http://arxiv.org/abs/2305.10912.

[Sch+23b] M. Schwarzer, J. Obando-Ceron, A. Courville, M. Bellemare, R. Agarwal, and P. S. Castro. "Bigger, Better, Faster: Human-level Atari with human-level efficiency". In: *ICML*. May 2023. URL: http://arxiv.org/abs/2305.19452.

[Sco10] S. Scott. "A modern Bayesian look at the multi-armed bandit". In: *Applied Stochastic Models in Business and Industry* 26 (2010), pp. 639–658.

[Sei+16] H. van Seijen, A Rupam Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton. "True Online Temporal-Difference Learning". In: *JMLR* (2016). URL: http://jmlr.org/papers/volume17/15-599/15-599.pdf.

[Sey+22] T. Seyde, P. Werner, W. Schwarting, I. Gilitschenski, M. Riedmiller, D. Rus, and M. Wulfmeier. "Solving Continuous Control via Q-learning". In: *ICLR*. Sept. 2022. URL: https://openreview.net/pdf?id=U5XOGxAgccS.

[Sha+20] R. Shah, P. Freire, N. Alex, R. Freedman, D. Krasheninnikov, L. Chan, M. D. Dennis, P. Abbeel, A. Dragan, and S. Russell. "Benefits of Assistance over Reward Learning". In: *NIPS Workshop*. 2020. URL: https://aima.cs.berkeley.edu/~russell/papers/neurips20ws-assistance.pdf.

[Sie+20] N. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, N. Heess, and M. Riedmiller. "Keep Doing What Worked: Behavior Modelling Priors for Offline Reinforcement Learning". In: *ICLR*. 2020. URL: https://openreview.net/pdf?id=rke7geHtwH.

[Sil+14] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. "Deterministic Policy Gradient Algorithms". In: *ICML*. ICML'14. JMLR.org, 2014, pp. I–387–I–395. URL: http://dl.acm.org/citation.cfm?id=3044805.3044850.

[Sil+16] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". en. In: *Nature* 529.7587 (2016), pp. 484–489. URL: http://dx.doi.org/10.1038/nature16961.

[Sil+17a] D. Silver et al. "Mastering the game of Go without human knowledge". en. In: *Nature* 550.7676 (2017), pp. 354–359. URL: http://dx.doi.org/10.1038/nature24270.

[Sil+17b] D. Silver et al. "The predictron: end-to-end learning and planning". In: *ICML*. 2017. URL: https://openreview.net/pdf?id=BkJsCIcgl.

[Sil18] D. Silver. *Lecture 9L Exploration and Exploitation*. 2018. URL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/XX.pdf.

[Sil+18] D. Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". en. In: *Science* 362.6419 (2018), pp. 1140–1144. URL: http://dx.doi.org/10.1126/science.aar6404.

[Sin+00] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. "Convergence Results for Single-Step On-PolicyReinforcement-Learning Algorithms". In: *MLJ* 38.3 (2000), pp. 287–308. URL: https://doi.org/10.1023/A:1007678930559.

[Ska+22] J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger. "Defining and characterizing reward hacking". In: *NIPS*. Sept. 2022.

[SKM18] S. Schwöbel, S. Kiebel, and D. Marković. "Active Inference, Belief Propagation, and the Bethe Approximation". en. In: *Neural Comput.* 30.9 (2018), pp. 2530–2567. URL: http://dx.doi.org/10.1162/neco_a_01108.

[Sli19]   A. Slivkins. "Introduction to Multi-Armed Bandits". In: *Foundations and Trends in Machine Learning* (2019). URL: http://arxiv.org/abs/1904.07272.

[Smi+23]   F. B. Smith, A. Kirsch, S. Farquhar, Y. Gal, A. Foster, and T. Rainforth. "Prediction-Oriented Bayesian Active Learning". In: *AISTATS*. Apr. 2023. URL: http://arxiv.org/abs/2304.08151.

[Sol64]   R. J. Solomonoff. "A formal theory of inductive inference. Part I". In: *Information and Control* 7.1 (Mar. 1964), pp. 1–22. URL: https://www.sciencedirect.com/science/article/pii/S0019995864902232.

[Son98]   E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. 2nd. Vol. 6. Texts in Applied Mathematics. Springer, 1998.

[Spi+24]   B. A. Spiegel, Z. Yang, W. Jurayj, B. Bachmann, S. Tellex, and G. Konidaris. "Informing Reinforcement Learning Agents by Grounding Language to Markov Decision Processes". In: *Workshop on Training Agents with Foundation Models at RLC 2024*. Aug. 2024. URL: https://openreview.net/pdf?id=uFm9e4Ly26.

[Spr17]   M. W. Spratling. "A review of predictive coding algorithms". en. In: *Brain Cogn.* 112 (2017), pp. 92–97. URL: http://dx.doi.org/10.1016/j.bandc.2015.11.003.

[SPS99]   R. S. Sutton, D. Precup, and S. Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artif. Intell.* 112.1 (Aug. 1999), pp. 181–211. URL: http://www.sciencedirect.com/science/article/pii/S0004370299000521.

[SS21]   D. Schmidt and T. Schmied. "Fast and Data-Efficient Training of Rainbow: an Experimental Study on Atari". In: *Deep RL Workshop NeurIPS 2021*. Dec. 2021. URL: https://openreview.net/pdf?id=GvM7A3cv63M.

[SSM08]   R. S. Sutton, C. Szepesvári, and H. R. Maei. "A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation". en. In: *NIPS*. NIPS'08. Red Hook, NY, USA: Curran Associates Inc., Dec. 2008, pp. 1609–1616. URL: https://proceedings.neurips.cc/paper_files/paper/2008/file/e0c641195b27425bb056ac56f8953d24-Paper.pdf.

[Str00]   M. Strens. "A Bayesian Framework for Reinforcement Learning". In: *ICML*. 2000.

[Sub+22]   J. Subramanian, A. Sinha, R. Seraj, and A. Mahajan. "Approximate information state for approximate planning and reinforcement learning in partially observed systems". In: *JMLR* 23.12 (2022), pp. 1–83. URL: http://jmlr.org/papers/v23/20-1165.html.

[Sut+08]   R. S. Sutton, C. Szepesvari, A. Geramifard, and M. P. Bowling. "Dyna-style planning with linear function approximation and prioritized sweeping". In: *UAI*. 2008.

[Sut15]   R. Sutton. *Introduction to RL with function approximation*. NIPS Tutorial. 2015. URL: http://media.nips.cc/Conferences/2015/tutorialslides/SuttonIntroRL-nips-2015-tutorial.pdf.

[Sut88]   R. Sutton. "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1 (1988), pp. 9–44.

[Sut90]   R. S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *ICML*. Ed. by B. Porter and R. Mooney. Morgan Kaufmann, 1990, pp. 216–224. URL: http://www.sciencedirect.com/science/article/pii/B9781558601413500304.

[Sut95]   R. S. Sutton. "TD models: Modeling the world at a mixture of time scales". en. In: *ICML*. Jan. 1995, pp. 531–539. URL: https://www.sciencedirect.com/science/article/abs/pii/B9781558603776500724.

[Sut96]   R. S. Sutton. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding". In: *NIPS*. Ed. by D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo. MIT Press, 1996, pp. 1038–1044. URL: http://papers.nips.cc/paper/1109-generalization-in-reinforcement-learning-successful-examples-using-sparse-coarse-coding.pdf.

[Sut+99]   R. Sutton, D. McAllester, S. Singh, and Y. Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *NIPS*. 1999.

[SW06]   J. E. Smith and R. L. Winkler. "The Optimizer's Curse: Skepticism and Postdecision Surprise in Decision Analysis". In: *Manage. Sci.* 52.3 (2006), pp. 311–322.

[Sze10]   C. Szepesvari. *Algorithms for Reinforcement Learning.* Morgan Claypool, 2010.

[Tam+16]   A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel. "Value Iteration Networks". In: *NIPS*. 2016. URL: http://arxiv.org/abs/1602.02867.

[Tan+23]   Y. Tang et al. "Understanding Self-Predictive Learning for Reinforcement Learning". In: *ICML*. 2023. URL: https://proceedings.mlr.press/v202/tang23d/tang23d.pdf.

[Ten02]   R. B. A. Tennenholtz. "R-max – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning". In: *JMLR* 3 (2002), pp. 213–231. URL: http://www.ai.mit.edu/projects/jmlr/papers/volume3/brafman02a/source/brafman02a.pdf.

[Tha+22]   S. Thakoor, M. Rowland, D. Borsa, W. Dabney, R. Munos, and A. Barreto. "Generalised Policy Improvement with Geometric Policy Composition". en. In: *ICML*. PMLR, June 2022, pp. 21272–21307. URL: https://proceedings.mlr.press/v162/thakoor22a.html.

[Tho33]   W. R. Thompson. "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25.3/4 (1933), pp. 285–294.

[TKE24]   H. Tang, D. Key, and K. Ellis. "WorldCoder, a model-based LLM agent: Building world models by writing code and interacting with the environment". In: *arXiv [cs.AI]* (Feb. 2024). URL: http://arxiv.org/abs/2402.12275.

[TL05]   E. Todorov and W. Li. "A Generalized Iterative LQG Method for Locally-optimal Feedback Control of Constrained Nonlinear Stochastic Systems". In: *ACC*. 2005, pp. 300–306.

[TMM19]   C. Tessler, D. J. Mankowitz, and S. Mannor. "Reward Constrained Policy Optimization". In: *ICLR*. 2019. URL: https://openreview.net/pdf?id=SkfrvsA9FX.

[Tom+22]   T. Tomilin, T. Dai, M. Fang, and M. Pechenizkiy. "LevDoom: A benchmark for generalization on level difficulty in reinforcement learning". In: *2022 IEEE Conference on Games (CoG)*. IEEE, Aug. 2022. URL: https://ieee-cog.org/2022/assets/papers/paper_30.pdf.

[Tom+24]   M. Tomar, P. Hansen-Estruch, P. Bachman, A. Lamb, J. Langford, M. E. Taylor, and S. Levine. "Video Occupancy Models". In: *arXiv [cs.CV]* (June 2024). URL: http://arxiv.org/abs/2407.09533.

[Tou09]   M. Toussaint. "Robot Rrajectory Optimization using Approximate Inference". In: *ICML*. 2009, pp. 1049–1056.

[Tou14]   M. Toussaint. *Bandits, Global Optimization, Active Learning, and Bayesian RL – understanding the common ground.* Autonomous Learning Summer School. 2014. URL: https://www.user.tu-berlin.de/mtoussai/teaching/14-BanditsOptimizationActiveLearningBayesianRL.pdf.

[TR97]   J. Tsitsiklis and B. V. Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Trans. on Automatic Control* 42.5 (1997), pp. 674–690.

[TS06]   M. Toussaint and A. Storkey. "Probabilistic inference for solving discrete and continuous state Markov Decision Processes". In: *ICML*. 2006, pp. 945–952.

[Tsc+20]   A. Tschantz, B. Millidge, A. K. Seth, and C. L. Buckley. "Reinforcement learning through active inference". In: *ICLR workshop on "Bridging AI and Cognitive Science"*. Feb. 2020.

[Tsc+23]   A. Tscshantz, B. Millidge, A. K. Seth, and C. L. Buckley. "Hybrid predictive coding: Inferring, fast and slow". en. In: *PLoS Comput. Biol.* 19.8 (Aug. 2023), e1011280. URL: https://journals.plos.org/ploscompbiol/article/file?id=10.1371/journal.pcbi.1011280&type=printable.

[Tsi+17]   P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman. "Human Learning in Atari". en. In: *AAAI Spring Symposium Series*. 2017. URL: https://www.aaai.org/ocs/index.php/SSS/SSS17/paper/viewPaper/15280.

[TVR97]   J. N. Tsitsiklis and B Van Roy. "An analysis of temporal-difference learning with function approximation". en. In: *IEEE Trans. Automat. Contr.* 42.5 (May 1997), pp. 674–690. URL: https://ieeexplore.ieee.org/abstract/document/580874.

[Unk24]   Unknown. "Beyond The Rainbow: High Performance Deep Reinforcement Learning On A Desktop PC". In: (Oct. 2024). URL: https://openreview.net/pdf?id=0ydseYDKRi.

[Val00]   H. Valpola. "Bayesian Ensemble Learning for Nonlinear Factor Analysis". PhD thesis. Helsinki University of Technology, 2000. URL: https://users.ics.aalto.fi/harri/thesis/valpola_thesis.ps.gz.

[van+18]   H. van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil. *Deep Reinforcement Learning and the Deadly Triad*. arXiv:1812.02648. 2018.

[VBW15]   S. S. Villar, J. Bowden, and J. Wason. "Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges". en. In: *Stat. Sci.* 30.2 (2015), pp. 199–215. URL: http://dx.doi.org/10.1214/14-STS504.

[Vee+19]   V. Veeriah, M. Hessel, Z. Xu, J. Rajendran, R. L. Lewis, J. Oh, H. P. van Hasselt, D. Silver, and S. Singh. "Discovery of Useful Questions as Auxiliary Tasks". In: *NIPS*. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/10ff0b5e85e5b85cc3095d431d8c08b4-Paper.pdf.

[Ven+24]   D. Venuto, S. N. Islam, M. Klissarov, D. Precup, S. Yang, and A. Anand. "Code as reward: Empowering reinforcement learning with VLMs". In: *ICML*. Feb. 2024. URL: https://openreview.net/forum?id=6P88DMUDvH.

[Vez+17]   A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. "FeUdal Networks for Hierarchical Reinforcement Learning". en. In: *ICML*. PMLR, July 2017, pp. 3540–3549. URL: https://proceedings.mlr.press/v70/vezhnevets17a.html.

[Vil+22]   A. R. Villaflor, Z. Huang, S. Pande, J. M. Dolan, and J. Schneider. "Addressing Optimism Bias in Sequence Modeling for Reinforcement Learning". en. In: *ICML*. PMLR, June 2022, pp. 22270–22283. URL: https://proceedings.mlr.press/v162/villaflor22a.html.

[VPG20]   N. Vieillard, O. Pietquin, and M. Geist. "Munchausen Reinforcement Learning". In: *NIPS*. Vol. 33. 2020, pp. 4235–4246. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/2c6a0bae0f071cbbf0bb3d5b11d90a82-Paper.pdf.

[Wag+19]   N. Wagener, C.-A. Cheng, J. Sacks, and B. Boots. "An online learning approach to model predictive control". In: *Robotics: Science and Systems*. Feb. 2019. URL: https://arxiv.org/abs/1902.08967.

[Wan+16]   Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: *ICML*. 2016. URL: http://proceedings.mlr.press/v48/wangf16.pdf.

[Wan+19]   T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. "Benchmarking Model-Based Reinforcement Learning". In: *arXiv [cs.LG]* (July 2019). URL: http://arxiv.org/abs/1907.02057.

[Wan+22]   T. Wang, S. S. Du, A. Torralba, P. Isola, A. Zhang, and Y. Tian. "Denoised MDPs: Learning World Models Better Than the World Itself". In: *ICML*. June 2022. URL: http://arxiv.org/abs/2206.15477.

[Wan+24a]   G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. "Voyager: An Open-Ended Embodied Agent with Large Language Models". In: *TMLR* (2024). URL: https://openreview.net/forum?id=ehfRiF0R3a.

[Wan+24b]   S. Wang, S. Liu, W. Ye, J. You, and Y. Gao. "EfficientZero V2: Mastering discrete and continuous control with limited data". In: *arXiv [cs.LG]* (Mar. 2024). URL: http://arxiv.org/abs/2403.00564.

[WAT17]   G. Williams, A. Aldrich, and E. A. Theodorou. "Model Predictive Path Integral Control: From Theory to Parallel Computation". In: *J. Guid. Control Dyn.* 40.2 (Feb. 2017), pp. 344–357. URL: https://doi.org/10.2514/1.G001921.

[Wat+21]   J. Watson, H. Abdulsamad, R. Findeisen, and J. Peters. "Stochastic Control through Approximate Bayesian Input Inference". In: *arxiv* (2021). URL: http://arxiv.org/abs/2105.07693.

[WCM24]   C. Wang, Y. Chen, and K. Murphy. "Model-based Policy Optimization under Approximate Bayesian Inference". en. In: *AISTATS*. PMLR, Apr. 2024, pp. 3250–3258. URL: https://proceedings.mlr.press/v238/wang24g.html.

[WD92]   C. Watkins and P. Dayan. "Q-learning". In: *Machine Learning* 8.3 (1992), pp. 279–292.

[Wei+24]   R. Wei, N. Lambert, A. McDonald, A. Garcia, and R. Calandra. "A unified view on solving objective mismatch in model-based Reinforcement Learning". In: *Trans. on Machine Learning Research* (2024). URL: https://openreview.net/forum?id=tQVZgvXhZb.

[Wen18a]   L. Weng. "A (Long) Peek into Reinforcement Learning". In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-02-19-rl-overview/.

[Wen18b]   L. Weng. "Policy Gradient Algorithms". In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

[WHT19]   Y. Wang, H. He, and X. Tan. "Truly Proximal Policy Optimization". In: *UAI*. 2019. URL: http://auai.org/uai2019/proceedings/papers/21.pdf.

[WHZ23]   Z. Wang, J. J. Hunt, and M. Zhou. "Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning". In: *ICLR*. 2023. URL: https://openreview.net/pdf?id=AHvFDPi-FA.

[Wie03]   E Wiewiora. "Potential-Based Shaping and Q-Value Initialization are Equivalent". In: *JAIR*. 2003. URL: https://jair.org/index.php/jair/article/view/10338.

[Wil+17]   G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. "Information theoretic MPC for model-based reinforcement learning". In: *ICRA*. IEEE, May 2017, pp. 1714–1721. URL: https://ieeexplore.ieee.org/document/7989202.

[Wil92]   R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *MLJ* 8.3-4 (1992), pp. 229–256.

[WIP20]   J. Watson, A. Imohiosen, and J. Peters. "Active Inference or Control as Inference? A Unifying View". In: *International Workshop on Active Inference*. 2020. URL: http://arxiv.org/abs/2010.00262.

[WL14]   N. Whiteley and A. Lee. "Twisted particle filters". en. In: *Annals of Statistics* 42.1 (Feb. 2014), pp. 115–141. URL: https://projecteuclid.org/journals/annals-of-statistics/volume-42/issue-1/Twisted-particle-filters/10.1214/13-AOS1167.full.

[Wu+17]   Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation". In: *NIPS*. 2017. URL: https://arxiv.org/abs/1708.05144.

[Wu+21]   Y. Wu, S. Zhai, N. Srivastava, J. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh. "Uncertainty Weighted Actor-critic for offline Reinforcement Learning". In: *ICML*. May 2021. URL: https://arxiv.org/abs/2105.08140.

[Wu+22]   P. Wu, A. Escontrela, D. Hafner, K. Goldberg, and P. Abbeel. "DayDreamer: World Models for Physical Robot Learning". In: (June 2022). arXiv: 2206.14176 [cs.RO]. URL: http://arxiv.org/abs/2206.14176.

[Wu+23]   G. Wu, W. Fang, J. Wang, P. Ge, J. Cao, Y. Ping, and P. Gou. "Dyna-PPO reinforcement learning with Gaussian process for the continuous action decision-making in autonomous driving". en. In: *Appl. Intell.* 53.13 (July 2023), pp. 16893–16907. URL: https://link.springer.com/article/10.1007/s10489-022-04354-x.

[Wur+22]   P. R. Wurman et al. "Outracing champion Gran Turismo drivers with deep reinforcement learning". en. In: *Nature* 602.7896 (Feb. 2022), pp. 223–228. URL: https://www.researchgate.net/publication/358484368_Outracing_champion_Gran_Turismo_drivers_with_deep_reinforcement_learning.

[Xu+17]   C. Xu, T. Qin, G. Wang, and T.-Y. Liu. "Reinforcement learning for learning rate control". In: *arXiv [cs.LG]* (May 2017). URL: http://arxiv.org/abs/1705.11159.

[Yan+23]   M. Yang, D Schuurmans, P Abbeel, and O. Nachum. "Dichotomy of control: Separating what you can control from what you cannot". In: *ICLR.* Vol. abs/2210.13435. 2023. URL: https://github.com/google-research/google-research/tree/.

[Yan+24]   S. Yang, Y. Du, S. K. S. Ghasemipour, J. Tompson, L. P. Kaelbling, D. Schuurmans, and P. Abbeel. "Learning Interactive Real-World Simulators". In: *ICLR.* 2024. URL: https://openreview.net/pdf?id=sFyTZEqmUY.

[Yao+22]   S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao. "ReAct: Synergizing Reasoning and Acting in Language Models". In: *ICLR.* Sept. 2022. URL: https://openreview.net/pdf?id=WE_vluYUL-X.

[Ye+21]   W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao. "Mastering Atari games with limited data". In: *NIPS.* Oct. 2021.

[Yu17]   H. Yu. "On convergence of some gradient-based temporal-differences algorithms for off-policy learning". In: *arXiv [cs.LG]* (Dec. 2017). URL: http://arxiv.org/abs/1712.09652.

[Yu+20]   T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma. "MOPO: Model-based Offline Policy Optimization". In: *NIPS.* Vol. 33. 2020, pp. 14129–14142. URL: https://proceedings.neurips.cc/paper_files/paper/2020/hash/a322852ce0df73e204b7e67cbbef0d0a-Abstract.html.

[Yu+23]   C. Yu, N Burgess, M Sahani, and S Gershman. "Successor-Predecessor Intrinsic Exploration". In: *NIPS.* Vol. abs/2305.15277. Curran Associates, Inc., May 2023, pp. 73021–73038. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/e6f2b968c4ee8ba260cd7077e39590dd-Abstract-Conference.html.

[Yua22]   M. Yuan. "Intrinsically-motivated reinforcement learning: A brief introduction". In: *arXiv [cs.LG]* (Mar. 2022). URL: http://arxiv.org/abs/2203.02298.

[Yua+24]   M. Yuan, R. C. Castanyer, B. Li, X. Jin, G. Berseth, and W. Zeng. "RLeXplore: Accelerating research in intrinsically-motivated reinforcement learning". In: *arXiv [cs.LG]* (May 2024). URL: http://arxiv.org/abs/2405.19548.

[YZ22]   Y. Yang and P. Zhai. "Click-through rate prediction in online advertising: A literature review". In: *Inf. Process. Manag.* 59.2 (2022), p. 102853. URL: https://www.sciencedirect.com/science/article/pii/S0306457321003241.

[ZABD10]   B. D. Ziebart, J Andrew Bagnell, and A. K. Dey. "Modeling Interaction via the Principle of Maximum Causal Entropy". In: *ICML.* 2010. URL: https://www.cs.uic.edu/pub/Ziebart/Publications/maximum-causal-entropy.pdf.

[Zel+24]   E. Zelikman, G. Harik, Y. Shao, V. Jayasiri, N. Haber, and N. D. Goodman. "Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking". In: *arXiv [cs.CL]* (Mar. 2024). URL: http://arxiv.org/abs/2403.09629.

[Zha+19]   S. Zhang, B. Liu, H. Yao, and S. Whiteson. "Provably convergent two-timescale off-policy actor-critic with function approximation". In: *ICML* 119 (Nov. 2019). Ed. by H. D. Iii and A. Singh, pp. 11204–11213. URL: https://proceedings.mlr.press/v119/zhang20s/zhang20s.pdf.

[Zha+21]   A. Zhang, R. T. McAllister, R. Calandra, Y. Gal, and S. Levine. "Learning Invariant Representations for Reinforcement Learning without Reconstruction". In: *ICLR*. 2021. URL: https://openreview.net/pdf?id=-2FCwDKRREu.

[Zha+23a]   J. Zhang, J. T. Springenberg, A. Byravan, L. Hasenclever, A. Abdolmaleki, D. Rao, N. Heess, and M. Riedmiller. "Leveraging Jumpy Models for Planning and Fast Learning in Robotic Domains". In: *arXiv [cs.RO]* (Feb. 2023). URL: http://arxiv.org/abs/2302.12617.

[Zha+23b]   W. Zhang, G. Wang, J. Sun, Y. Yuan, and G. Huang. "STORM: Efficient Stochastic Transformer based world models for reinforcement learning". In: *arXiv [cs.LG]* (Oct. 2023). URL: http://arxiv.org/abs/2310.09615.

[Zha+24]   S. Zhao, R. Brekelmans, A. Makhzani, and R. B. Grosse. "Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo". In: *ICML*. June 2024. URL: https://openreview.net/pdf?id=frAONNBS1n.

[Zhe+22]   L. Zheng, T. Fiez, Z. Alumbaugh, B. Chasnov, and L. J. Ratliff. "Stackelberg actor-critic: Game-theoretic reinforcement learning algorithms". en. In: *AAAI* 36.8 (June 2022), pp. 9217–9224. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20908.

[Zho+22]   H. Zhou, Z. Lin, J. Li, Q. Fu, W. Yang, and D. Ye. "Revisiting discrete soft actor-critic". In: *arXiv [cs.LG]* (Sept. 2022). URL: http://arxiv.org/abs/2209.10081.

[Zho+24]   G. Zhou, S. Swaminathan, R. V. Raju, J. S. Guntupalli, W. Lehrach, J. Ortiz, A. Dedieu, M. Lázaro-Gredilla, and K. Murphy. "Diffusion Model Predictive Control". In: *arXiv [cs.LG]* (Oct. 2024). URL: http://arxiv.org/abs/2410.05364.

[ZHR24]   H. Zhu, B. Huang, and S. Russell. "On representation complexity of model-based and model-free reinforcement learning". In: *ICLR*. 2024.

[Zie+08]   B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. "Maximum Entropy Inverse Reinforcement Learning". In: *AAAI*. 2008, pp. 1433–1438.

[Zin+21]   L Zintgraf, S. Schulze, C. Lu, L. Feng, M. Igl, K Shiarlis, Y Gal, K. Hofmann, and S. Whiteson. "VariBAD: Variational Bayes-Adaptive Deep RL via meta-learning". In: *J. Mach. Learn. Res.* 22.289 (2021), 289:1–289:39. URL: https://www.jmlr.org/papers/volume22/21-0657/21-0657.pdf.

[Zit+23]   B. Zitkovich et al. "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control". en. In: *Conference on Robot Learning*. PMLR, Dec. 2023, pp. 2165–2183. URL: https://proceedings.mlr.press/v229/zitkovich23a.html.

[ZS22]   N. Zucchet and J. Sacramento. "Beyond backpropagation: Bilevel optimization through implicit differentiation and equilibrium propagation". en. In: *Neural Comput.* 34.12 (Nov. 2022), pp. 2309–2346. URL: https://direct.mit.edu/neco/article-pdf/34/12/2309/2057431/neco_a_01547.pdf.

[ZSE24]   C. Zheng, R. Salakhutdinov, and B. Eysenbach. "Contrastive Difference Predictive Coding". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/pdf?id=0akLDTFR9x.

[ZW19]   S. Zhang and S. Whiteson. "DAC: The Double Actor-Critic Architecture for Learning Options". In: *NIPS* 32 (2019). URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/4f284803bd0966cc24fa8683a34afc6e-Paper.pdf.