RL Notes

Huu Duc Nguyen M.Sc.

2 May 2022

Contents

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
CS	Computer Science
CV	Computer Vision
RL	Reinforcement Learning
ICML	International Conference on Machine Learning
prob.	probability
params.	parameters
algor.	algorithms
a.k.a.	also known as
w.r.t.	with regard to
no.	number of
func.	function
vs.	versus
s.t.	subject to
MLE	Maximum Likelihood Estimation
SVM	State Vector Machine
GD	Gradient Descent
KL	Kullback–Leibler
IG	Information Gain
LQR	Linear Quadratic Regulator
iLQR	Iterative Linear Quadratic Regulator
MPC	Model Predictive Control
FLM	Fitted Local Model
BPTT	Backpropagation through time
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
CGAN	Conditional Generative Adversarial Network
DOF	degrees of freedom
HMM	Hidden Markov Model
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
A3C	Asynchronous advantage actor-critic

SAC	Soft actor-critic
DQN	Deep Q-learning
DDP	Differential Dynamic Programming
DAgger	Dataset Aggregation
CEM	Cross-entropy Method
MCTS	Monte-Carlo Tree Search
MBA	Model-based Acceleration
MVE	Model-based Value Expansion
MBPO	Model-based Policy Optimization
UCB	Upper Confidence Bounce
PAC	Probably Approximately Correct
CQL	Conservative Q-learning
MOPO	Model-Based Offline Policy Optimization
IRL	Inverse Reinforcement Learning
MaxEnt	Maximum Entropy
MAML	Model-Agnostic Meta-Learning
OPE	Off-policy evaluation
LSTD	Least-squares temporal difference
LSPI	Least-squares policy iteration
US	unconditioned stimuli
CS	conditioned stimuli
CR	conditioned response

1 Overview

This if my learning notes for RL. RL are approaches for learning decision making from experience. In the Artificial Intelligence (AI) context, many RL algorithms handle the scarcity of available (human-annotated) data.

> *Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education, one would obtain the adult brain.* - Alan Turing -

A RL problem has 3 major blocks as follows:

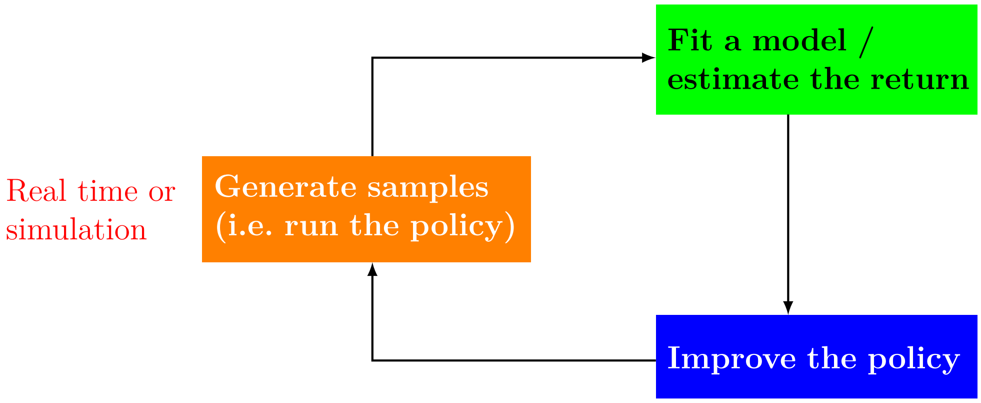


Figure 1.1: Structure of RL algorithms.

1.1 Learning resources

- Deep RL - CS285, UC Berkeley - Sergey Levine
- CS188 Berkeley AI
- The book *Reinforcement learning: An introduction* [SB18]
- `TensorFlow` Agents' tutorial
- A Free course in Deep RL

1.2 Terminology & Notation

- Check out Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) in the robotic notes or the RL book by Sutton and Barto (2018) [SB18].

- $\mathbf{s}_t$ - state
- $\mathbf{o}_t$ - observation
- $\mathbf{a}_t$ - action
- $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ - policy (or $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ for fully observed scenario)
- $r(\mathbf{s}_t, \mathbf{a}_t)$ - reward or $c(\mathbf{s}_t, \mathbf{a}_t)$ - cost
- τ - trajectory (as sequence of states and actions)

$$p_\theta(\tau) = p_\theta(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

- The Q-function is the expectation of total reward, from the q-state $(\mathbf{s}_t, \mathbf{a}_t)$, under policy π_θ.

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta} \left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t \right] \tag{1.1}$$

- The value function is the expectation of total reward, from the state $\mathbf{s}_t$, under policy π_θ.

$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta} \left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t \right] = \mathbb{E}_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)} Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \tag{1.2}$$

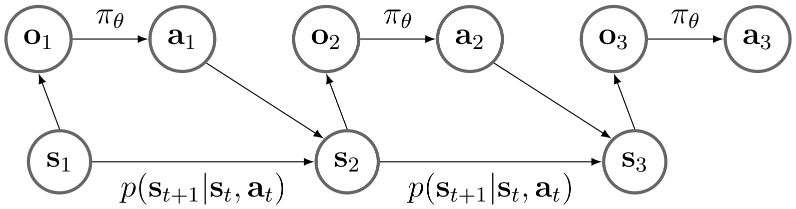


Figure 1.2: The relationship between state $\mathbf{s}_t$, observation $\mathbf{o}_t$ and action $\mathbf{a}_t$.

1.3 Overview on RL Algorithms

RL problem revolves around maximizing the expectation of total rewards. We aim to find the parameters (params.) to maximize the expected value of the sum of rewards, under the trajectory distribution.

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{1.3}$$

There are many methods / algorithms with their trade-offs and assumptions:

- Sampling efficiency & stability and ease of use
- Stochastic or deterministic
- Continuous or discrete
- Episode or infinite horizon

Tab. 1.1 gives an overview and comparison between algorithms.

1.4 Challenges

- Humans can learn incredibly quickly
- Humans can reuse past knowledge
  Transfer learning in Deep RL is an open problem
- Not clear what the reward function should be
- Not clear what the role of prediction should be

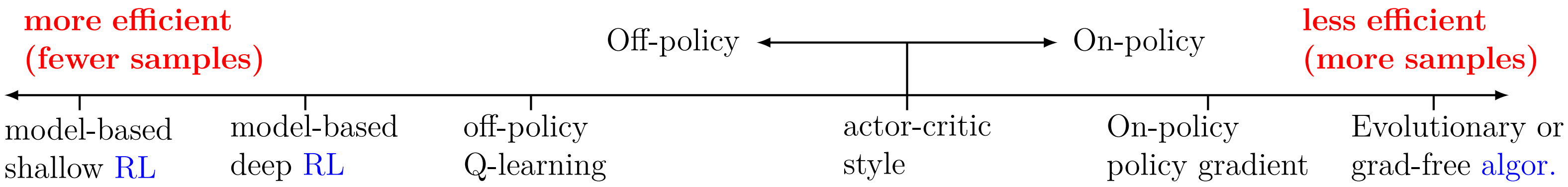
	Model-based	Value function fitting	Actor-critic	Policy Gradient
Sample efficiency (How many samples do we need to get good policy?)	 more efficient (fewer samples) Off-policy On-policy less efficient (more samples) model-based shallow RL model-based deep RL off-policy Q-learning actor-critic style On-policy policy gradient Evolutionary or grad-free algor. - Sometimes, with simulated experiences, we can use **less efficient** algor. **Wall clock time ≠ efficiency** - More assumptions, as we go to the left			
Stability & ease of use - Does it converge? - If yes, to what? - Does it **always** converge?	**RL is often not GD** Model will converge. ***BUT***, better model do **NOT GUARANTEE** better policy	Minimize error of fit At worst case, doesn't optimize anything. Un-provable convergence ⇒ **more like heuristics**		**is GD** least efficient + assumptions
Assumption	By some: episode learning	***Generally***: full observability. By some continuous method: continuity / smoothness		By pure policy gradient methods: ***Often:*** episode learning
Example	- Dyna - Guided policy search	- Q-learning - DQN - Temporal difference - Fitted value iteration	- A3C [MBM+16] - SAC [HZA+18]	- REINFORCE [Wil92] - Natural policy gradient [Kak01] - Trusted Region policy optimization [SLA+

Table 1.1: Different RL algorithms.

2 Markov Decision Process

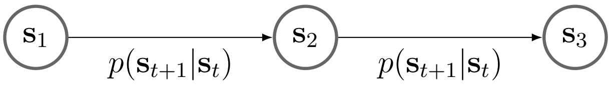
2.1 Markov Chain

A Markov Chain $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$ (discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}\|s_t)$ or $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$ ($\mathcal{T}$ is a matrix)

Markov chain is a process **without memories**. In other words, the next state s_{t+1} depends only on the current state s_t, not the previous state s_{t-1}.


$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$

2.2 Hidden Markov Model

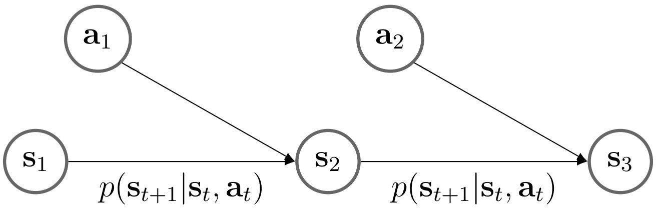
A Markov chain becomes a Hidden Markov Model (HMM) when the states are not completely observable.

2.3 Markov Decision Process (MDP)

A MDP $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$ (discrete or continuous)
$\mathcal{A}$ – action space	$a \in \mathcal{A}$ (discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}\|s_t)$ (now a tensor)
r – reward function	$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad r(s_t, a_t)$
γ – discount factor	$\gamma \in [0, 1]$ (optional)


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

There are definitions relating to MDP:

- Policy: choice of action (at each state): $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
- Utility: sum of (discounted) rewards

A MDP can also be considered as a Markov chain on the augmented state (**s, a**), also known as (a.k.a.) Q-state. Knowing the state transition $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, the transition of these Q-states can be derived as follows:

$$p((\mathbf{s}_{t+1}, \mathbf{a}_{t+1})|(\mathbf{s}_t, \mathbf{a}_t)) = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}) \tag{2.1}$$

MDP state projects an expectimax-like search tree [**TODO: add image**]

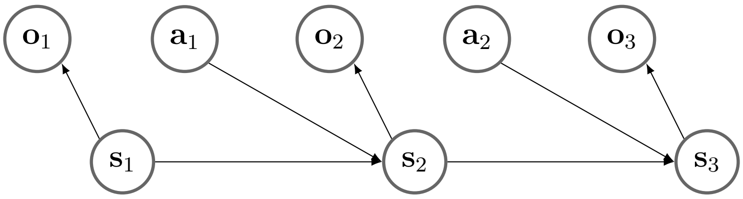
NOTE: Solving a MDP means to find a policy that maximize the received reward.

2.4 Partially Observable Markov Decision Process (POMDP)

A POMDP $\mathcal{M}$ is a tuple consisting of:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r, \gamma\}$$

$\mathcal{S}$ – state space	$s \in \mathcal{S}$	(discrete or continuous)
$\mathcal{A}$ – action space	$a \in \mathcal{A}$	(discrete or continuous)
$\mathcal{O}$ – observation space	$o \in \mathcal{O}$	(discrete or continuous)
$\mathcal{T}$ – transition operator	$p(s_{t+1}|s_t)$	
$\mathcal{E}$ – emission prob.	$p(o_t|s_t)$	
r – reward function	$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$	
γ – discount factor	$\gamma \in [0, 1]$	(optional)



Finite versus (vs.) infinite horizon: [**TODO:**]

To solve infinite utilities:

- Finite horizon
- Discounting
- Absorbing state (like the fire hole, overheating)

POMDP is defined with a tuple: $< S, A, O, P, R, Z, \gamma >$:

- S: state
- A: action
- O: **observation**
- P: transition matrix
- R: reward
- Z: **observation function (func.)**
- γ: discount factor (optional)

$$Z^a_{s'o} = P\left[O_{t+1} = o | S_{t+1} = s', A_t = a\right] \tag{2.2}$$

$$\Rightarrow \quad \begin{cases} V^*(b) \leftarrow \max\limits_a \left[R^a_b + \gamma \sum\limits_{s'} P^a_{bb'} V(b')\right] \\ \pi^*(b) = \arg\max\limits_a \left[R^a_b + \gamma \sum\limits_{s'} P^a_{bb'} V(b')\right] \end{cases} \tag{2.3}$$

Markov Models		Do we have control over the state transition?	
		No	Yes
Are the states completely observable?	Yes	Markov Chain	MDP
	No	HMM	POMDP

2.5 Bellman equations

- $T(s, a, s') = P(s'|s, a)$ the prob. of reaching state s' from taking action a at state s
- $R(s, a, s')$ the reward of making the transition
- $Q^*(s, a)$: expected utility starting in state s and having taken action a, then act optimally

  $$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right] \tag{2.4}$$

  It is the sum over possible next state s', because there is uncertainty of which state s' will be reached, even with the same starting state s and action a.
- $V^*(s)$: value of a state - expected utility starting in state s and acting optimally

  $$V^*(s) = \max_a Q^*(s, a) \tag{2.5}$$

  $$\Rightarrow \quad V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s')\right] \tag{2.6}$$

- $\pi^*(s)$: optimal action / policy from state s

[**TODO: Explain this??**]

$$\mathbf{V(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1}) | S_t = s\right] = R_s + \gamma \sum_{s'} P_{ss'} V(s')} \tag{2.7}$$

3 Imitation Learning

Also known as *Behavior Cloning*, essentially, this is ***supervised learning***. One problem might arise: applying the learned policy π_θ might lead to different action $\mathbf{a}_t$, which then leads to different observations and states, comparing to the given dataset: $p_{data}(\mathbf{o}_t) \neq p_{\pi_\theta}(\mathbf{o}_t)$. This ***distribution mismatch*** can be tackled by adding on-policy data.

3.1 DAgger

Dataset Aggregation (DAgger) aggregates training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of just $p_{data}(\mathbf{o}_t)$. Without DAgger, it is proven that the error will grow quadratically with the number of time steps $\mathcal{O}(\epsilon T^2)$. [RGB11]

1. Train $p_{\pi_\theta}(\mathbf{o}_t)$ from human data $\mathcal{D} = \{(\mathbf{o}_t, \mathbf{a}_t)_i\}$
2. Run $p_{\pi_\theta}(\mathbf{o}_t)$ to get data set $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with action $\mathbf{a}_t$
4. Aggregate $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_\pi$

The major problem with DAgger is that it requires human input again in step 3.

3.2 Recap

- Requires human to annotate the data
- Often (but not always) insufficient by itself (distribution mismatch problem)
- Sometimes works well

Problems:

- Non-Markovian behavior
- Multimodal behavior

Solutions:

- Output a mixture of Gaussians
- Latent variable models
- Auto-regressive discretization (src)

4 Policy Gradient

The Policy Gradient is a model-free RL approach. A model-free RL approach assumes that we don't know the transition model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ or the initial state prob. $p(\mathbf{s}_1)$. However, we can either interact with the real world or run simulation to sample the data. With regard to the general structure for a RL algorithm, this approach has a neural network (Fig. 4.1) to learn and optimize the policy (the blue box in Fig. 1.1).

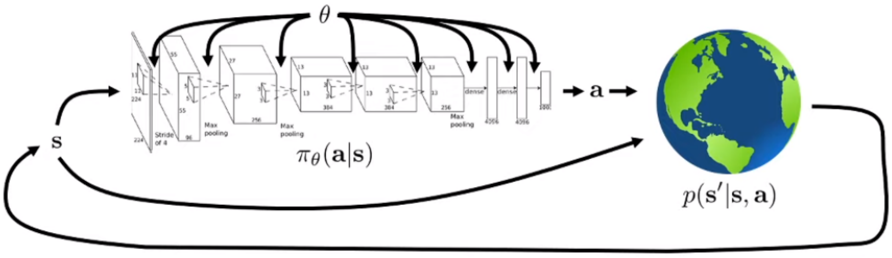

θ
$\mathbf{s}$
$\mathbf{a}$
dense
Max pooling
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

Figure 4.1: The policy network $\pi_\theta(\mathbf{a}|\mathbf{s})$ with params. θ. The network takes the current state $\mathbf{s}_t$ as input, learn the policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ by optimizing params. θ, and output the action $\mathbf{a}_t$.

4.1 Approach

Goal: to maximize the expectation of total rewards, which will be denoted as $J(\theta)$

$$\tau = \{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\} \qquad \text{– denotes the trajectory} \tag{4.1}$$

$$p_\theta(\tau) = p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) \qquad \text{– prob. of the trajectory} \tag{4.2}$$

$$= p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \tag{4.3}$$

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] = \arg\max_\theta J(\theta) \qquad \text{– RL goal} \tag{4.4}$$

$$= \arg\max_\theta \mathbb{E}_{(\mathbf{s},\mathbf{a})\sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s},\ \mathbf{a})] \qquad \text{– } \textbf{infinite} \text{ horizon case} \tag{4.5}$$

$$= \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] \qquad \text{– } \textbf{finite} \text{ horizon case} \tag{4.6}$$

$$J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{4.7}$$

$$= \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau \tag{4.8}$$

Even though we do not know the initial state prob. $p(\mathbf{s}_1)$ and the transition model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, we do have the ability to interact with the world and take samples from it. Thus, we can simply take N trajectory samples τ_i and take the average of them to approximate the expectation of $J(\theta)$. The higher the number of sample trajectories N is, the better the approximation accuracy.

$$J(\theta) \approx \frac{1}{N}\sum_i^N \sum_t^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \qquad \text{sum over samples and time steps} \tag{4.9}$$

Using a convenient identity transformation, we can derive ***the Policy Gradient***:

$$\nabla_\theta p_\theta(\tau) = p_\theta(\tau)\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) \qquad \text{[Wil92]} \tag{4.10}$$

$$\Rightarrow \nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau)\nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \tag{4.11}$$

$$= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log p_\theta(\tau) r(\tau)\right] \tag{4.12}$$

$$= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t)\right)\right] \qquad \text{(substitute Eq. 4.3)} \tag{4.13}$$

$$\boxed{\begin{aligned} \nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^N\left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)\right] && \text{(4.14a)} \\ \Rightarrow \quad & \theta \leftarrow \theta + \alpha\nabla_\theta J(\theta) && \text{(4.14b)} \end{aligned}}$$

REMARKS: some what like Maximum Likelihood Estimation (MLE), makes good stuff happen more, bad stuff happens less

4.2 Partial Observability

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^N\left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|o_{i,t})\right)\left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)\right]$$

$$o_{i,t} \rightarrow \underset{\pi_\theta(\mathbf{a}_{i,t}|o_{i,t})}{\text{network}} \rightarrow \mathbf{a}_{i,t}$$

Markov property is not actually used! ⇒ can use policy gradient for POMDPs without modification

4.3 High Variance Problem

In general, when we add a constant (either positive or negative) to the rewards, the policy should be the same. However, this is not the case for the above derivation of policy gradient.

The change of policy distribution varies depends on the value of the total rewards $r(\tau)$. In other words, the problem is **HIGH VARIANCE with** $r(\tau)$.

- Different samples ⇒ different gradient estimate
- For a small finite number of (no.) samples ⇒ noisy gradient
  (At the beginning, policy θ is not so good ⇒ random action ⇒ the not-so-good action results accumulate ⇒ high variance in the end)

Solution: reducing variance

- *Causality:* policy at later time step t' cannot affect reward at previous time step t ($t' > t$)
$$\Rightarrow \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)\right] \tag{4.15}$$
$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\widehat{Q}_{i,t} \tag{4.16}$$
  This leads to smaller variance, because $\widehat{Q}_{i,t}$ (the reward to-go) is smaller than the total rewards, and the expectation of smaller number has smaller variance. [SMS+99; BB01]
- *Baselines:* the average of total rewards over different trajectories. It is proven that subtracting the baseline is unbiased in expectation. This is not the optimal baseline to reduce the variance, but it's simple and good enough.
$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau) \tag{4.17}$$
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b] \tag{4.18}$$
$$r(\tau) - b = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t) \tag{4.19}$$
  The complex optimal baselines are derived for each element g_i of the gradient $\mathbf{g}$ [PS08]
$$b_h = \frac{\frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_{\theta_h}\log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)^2 r(\tau)\right]}{\frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_{\theta_h}\log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)^2\right]} \tag{4.20}$$

4.4 Off-policy Policy Gradient

Since we have $\tau \sim p_\theta(\tau)$, vanilla Policy Gradient is on-policy. This poses a problem, since the neural networks change only a bit with each gradient step. Off-policy Policy Gradient can be derived with **Important sampling**. θ' is the *new* params. and θ is the *old* params.

$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] && (4.21)\\
\Rightarrow \quad \nabla_{\theta'} J(\theta') &= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p_\theta(\tau)} r(\tau)\right] && (4.22)\\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau)\right] && (4.23)\\
&\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \widehat{Q}_{i,t} && (4.24)
\end{aligned}
$$

4.5 REINFORCE Algorithm

```
1. Sample $\{\tau_i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ policy
2. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$        [Wil92]
```

When coding, use the pseudo loss as a weighted maximum likelihood:

$$
\widetilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \widehat{Q}_{i,t} \tag{4.25}
$$

Pseudo code: (`tensorflow`)

```
logits = policy.predictions(states)
negative_likelihoods = tf.nn.softmax_cross_entropy(labels = actions, logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

with `q_values` already taking causality and baselines into account.

4.6 Natural Policy Gradient

Natural Policy gradients, a.k.a., covariant policy gradient, apply a trick to change the learning rate for different parameters. The high-level idea is such that some params. change prob. a lot more than others. In other words, the vanilla policy gradient apply a constraint on the params. space rather than the policy space. But with every gradient step, we rather want a constant step in the policy space. [Kak01; PS08]

$$\max_{\Delta\theta} J(\theta + \Delta\theta) \approx J(\theta) + \Delta\theta^T \nabla_\theta J \tag{4.26}$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta), \quad ||\theta' - \theta||^2 \leq \epsilon \qquad \text{(params. space)} \tag{4.27}$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta), \quad D(\pi_{\theta'}, \pi_\theta) \leq \epsilon \qquad \text{(policy space)} \tag{4.28}$$

A good choice for $D(\pi_{\theta'}, \pi_\theta)$ is the Kullback–Leibler (KL)-divergence. To simplify the process, the KL-divergence is approximated with Fisher information matrix $\mathbf{F}$ [Ama98]

$$D_{KL}(\pi_{\theta'} || \pi_\theta) \approx \frac{1}{2}(\theta' - \theta)^T \mathbf{F} (\theta' - \theta) = \frac{1}{2} \Delta\theta^T \mathbf{F} \Delta\theta \tag{4.29}$$

$$\mathbf{F} = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})^T] \tag{4.30}$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \quad \text{s.t. } ||\theta' - \theta||^2_{\mathbf{F}} \leq \epsilon \tag{4.31}$$

$$\theta \leftarrow \theta + \alpha_n \mathbf{F}^{-1} \nabla_\theta J(\theta) \tag{4.32}$$

$$\alpha_n = \left[\epsilon \left(\nabla J(\theta)^T \mathbf{F}_\theta^{-1} \nabla J(\theta) \right)^{-1} \right]^{\frac{1}{2}} \tag{4.33}$$

- Convergence to local minimum is guaranteed
- Faster convergence and avoids premature convergence of vanilla gradients
- Requires fewer data points for a good gradient estimate than vanilla gradients

4.7 Examples

[TODO:]

4.8 References

- Williams (1992) [Wil92]. *"Simple statistical gradient-following algorithms for connectionist reinforcement learning"*.
- Peters et al. (2008) [PS08]. *"Reinforcement learning of motor skills with policy gradients"*.
- Levine et al. (2013) [LK13]. *"Guided policy search"*.
- Schulman et al. (2015) [SLA+15]. *"Trust region policy optimization"*.
- Schulman et al. (2017) [SWD+17]. *"Proximal policy optimization algorithms"*.

5 Actor-Critic

Actor-Critic is kind of hybrid between policy gradient and value function based approach. Compared to deep Policy gradient, deep Actor-Critic has an extra network to learn the value function (the green box in Fig. 1.1).

- **the Actor is the policy**
- **the Critic is the value function (a.k.a. policy evaluation)**

5.1 Approach

We continue with the Eq. 4.16, where we multiply with the estimate of the expected reward $\widehat{Q}_{i,t}$. The estimate $\widehat{Q}_{i,t}$ is currently calculated as the approximation of the sum of the reward afterward, from roll-outs. There are different ways we could go better than that single-sample estimate.

- $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$: the Q-function, a.k.a., the state-action value function, represents the total reward from taking $\mathbf{a}_t$ at state $\mathbf{s}_t$, the *true expected* reward-to-go.
$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(\mathbf{s}_{t'}, a_{t'})|\mathbf{s}_t, \mathbf{a}_t] \tag{5.1}$$
- $V^\pi(\mathbf{s}_t)$: the state value function represents the total reward from state $\mathbf{s}_t$.
$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t, \mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)] \tag{5.2}$$
- $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: the advantage function: represents how much action $\mathbf{a}_t$ is better than average
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t) \tag{5.3}$$

Using these value functions, we would have a better estimate for the policy gradients. Thus, we can rewrite the gradient as:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \tag{5.4}$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} V^\pi(\mathbf{s}_{t+1}) \qquad \text{(Eq. 5.1)} \tag{5.5}$$

$$\approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) \qquad \text{(with 1 sample)} \tag{5.6}$$

$$\Rightarrow \quad A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t) \qquad \text{(Eq. 5.3)} \tag{5.7}$$

From the above derivation, let just fit the value function $V^\pi(\mathbf{s})$ with a neural network. There are two possible approaches:

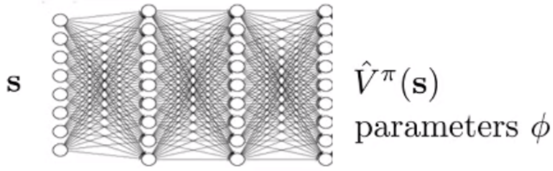


Figure 5.1: The network for value function $V_\phi^\pi(s)$ with params. ϕ.

- **Monte-Carlo:** just as with policy gradient, we approximate by the result from a single sample roll-out.

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{5.8}$$

The average of multiple samples would be a better approximation for the true expectation. However, we could not simply stop at one state within the trajectories and try out different actions. Single sample estimation is still pretty good!

$$\text{Better (but not possible)} \qquad V^\pi(\mathbf{s}_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{5.9}$$

$$\text{Training data:} \qquad \{(\mathbf{s}_{i,t}, y_{i,t})\} = \left\{\left(\mathbf{s}_{i,t}, \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right)\right\} \tag{5.10}$$

$$\text{Supervised regression:} \qquad \mathcal{L}(\phi) = \frac{1}{2}\sum_i \left|\left|\widehat{V}_\phi^\pi(\mathbf{s}_i) - y_i\right|\right|^2 \tag{5.11}$$

- **Bootstrapped estimate:** use the previous fitted value function

$$y_{i,t} = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(\mathbf{s}_{t'}, a_{t'}|\mathbf{s}_{i,t})] \qquad \text{ideal target} \tag{5.12}$$

$$\approx r(\mathbf{s}_{i,t}, a_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) \tag{5.13}$$

$$\approx r(\mathbf{s}_{i,t}, a_{i,t}) + \widehat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) \tag{5.14}$$

$$\text{Training data:} \qquad \{(\mathbf{s}_{i,t}, y_{i,t})\} = \left\{\left(\mathbf{s}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \widehat{V}_\phi^\pi(\mathbf{s}_{i,t+1})\right)\right\} \tag{5.15}$$

$$\text{Supervised regression:} \qquad \mathcal{L}(\phi) = \frac{1}{2}\sum_i \left|\left|\widehat{V}_\phi^\pi(\mathbf{s}_i) - y_i\right|\right|^2 \tag{5.16}$$

5.2 Batch Actor-Critic

1. Sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
2. Fit $V_\phi^\pi(\mathbf{s})$ to sampled reward sums (either Monte-Carlo [SML+15] or bootstrapped estimate)
3. Evaluate $\widehat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \widehat{V}_\phi^\pi(\mathbf{s}_{i'}) - \widehat{V}_\phi^\pi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\widehat{A}^\pi(\mathbf{a}_i, \mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

With **discount factor** $\gamma \in [0, 1]$ (0.99 works well)

3. Evaluate $\widehat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma\widehat{V}_\phi^\pi(\mathbf{s}_{i'}) - \widehat{V}_\phi^\pi(\mathbf{s}_i)$ [Tho14]

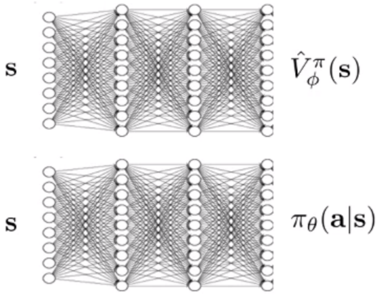
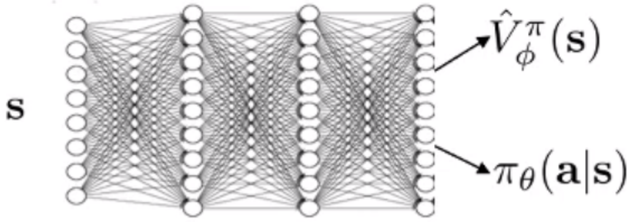
5.3 Online Actor-Critic

1. Take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get sample $(\mathbf{s},\ \mathbf{a},\ \mathbf{s'}, r)$
2. Update $\widehat{V}^\pi_\phi$ using target value $r + \gamma \widehat{V}^\pi_\phi(\mathbf{s}')$
3. Evaluate $\widehat{A}^\pi(\mathbf{s},\ \mathbf{a}) = r(\mathbf{s},\ \mathbf{a}) + \widehat{V}^\pi_\phi(\mathbf{s}') - \widehat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \widehat{A}^\pi(\mathbf{s},\ \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Problem: single batch ⇒ need parallel actor-critic (synchronous / asynchronous) [MBM+16]

5.4 Design Decisions

- Architecture design:

Two separate networks	Shared network design
+simple and stable −no shared features between actor & critic  s $\hat{V}^\pi_\phi(\mathbf{s})$ s $\pi_\theta(\mathbf{a}\|\mathbf{s})$	+could be more efficient in practice −two different gradients which need to tuned  s $\hat{V}^\pi_\phi(\mathbf{s})$ $\pi_\theta(\mathbf{a}\|\mathbf{s})$

- Critic as state-dependent baselines

  Actor-critic: +lower variance (due to critic)
  −not unbiased (if the critic is not perfect)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \widehat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) - \widehat{V}^\pi_\phi(\mathbf{s}_{i,t}) \right)$$

  Policy gradient: +no bias
  −higher variance (because of single-sample estimate)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - b \right)$$

  ⇒ Critic as baseline: +no bias
  +lower variance (baseline is closer to rewards)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right) - \widehat{V}^\pi_\phi(\mathbf{s}_{i,t}) \right)$$

  This doesn't lower the variance as much as in the actor-critic algorithm. But it is much lower than using a constant baseline, and it's still unbiased.

- Control variates: action-dependent baselines [GLG+16]
  We could go further with a state dependent baseline, and have a action-and-state-dependent baselines. The variance is now even lower, but it's getting much more complicated.

  $$\widehat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - V_{\phi}^{\pi}(\mathbf{s}_t)$$

  +no bias (state dependent baseline)
  −still high variance (compared to actor-critic)

  $$\widehat{A}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - Q_{\phi}^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$$

  +goes to 0 in expectation if critic is correct
  −not correct

  The second one lead to wrong policy gradient, thus must be corrected with an error term:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\widehat{Q}_{i,t} - Q_{\phi}^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right) + \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \mathbb{E}_{\mathbf{a} \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_{i,t})}[Q_{\phi}^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})] \tag{5.17}$$

- Eligibility traces & n-step returns: reduces the bias [MBM+16]

  Actor-critic: $$\widehat{A}_{C}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \widehat{V}_{\phi}^{\pi}(\mathbf{s}_{t+1}) - \widehat{V}_{\phi}^{\pi}(\mathbf{s}_t) \tag{5.18}$$

  +low variance
  −but biased

  Monte-Carlo: $$\widehat{A}_{MC}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, a_{t'}) - \widehat{V}_{\phi}^{\pi}(\mathbf{s}_t) \tag{5.19}$$

  +no bias
  −higher variance

  $$\Rightarrow \qquad \widehat{A}_{n}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'}, a_{t'}) - \widehat{V}_{\phi}^{\pi}(\mathbf{s}_t) + \gamma^{n} \widehat{V}_{\phi}^{\pi}(\mathbf{s}_{t+n}) \tag{5.20}$$

  Simply put, the further the states are in the future, the higher the variance of those states. E.g., where would you/the robot be in 5 minutes versus where would you/the robot be in 20 years? The larger n is, the lower the bias, the higher the variance.

- Generalized advantage estimation: extension of n-step returns [SML+15]
  To have many n-step returns, then take weighted average of them.

  $$\widehat{A}_{GAE}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{n=1}^{\infty} \omega_n \widehat{A}_{n}^{\pi}(\mathbf{s}_t, \mathbf{a}_t), \qquad \omega_n \propto \lambda^{n-1} \tag{5.21}$$

  $$\Rightarrow \widehat{A}_{GAE}^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t} \delta_{t'}, \qquad \delta_{t'} = r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) + \gamma \widehat{V}_{\phi}^{\pi}(\mathbf{s}_{t'+1}) - \widehat{V}_{\phi}^{\pi}(\mathbf{s}_{t'}) \tag{5.22}$$

5.5 Natural Actor-Critic

Natural Actor-Critic [PVS05] is the extension of natural policy gradient [Kak01] to actor-critic approach. [PS08]

5.6 Off-policy Actor-Critic

SAC algorithm: [HZA+18]

1. Take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s},\ \mathbf{a},\ \mathbf{s'}, r)$, store in buffer $\mathcal{B}$
2. Sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
3. Update $\widehat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma \widehat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i), \quad \mathbf{a}'_i \sim \pi_\theta(\mathbf{a}'_i|\mathbf{s}'_i)$
4. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^\pi_i|\mathbf{s})\widehat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i), \quad \mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

NOTE: In step 3 and 4, the actions are not from the replay buffer $\mathcal{B}$, but from our current policy $\mathbf{a}'_i \sim \pi_\theta(\mathbf{a}'_i|\mathbf{s}'_i)$ and $\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)$

5.7 References

- Sutton et al. (1999) [SMS+99]. *"Policy gradient methods for reinforcement learning with function approximation"*.
- Mnih et al. (2016) [MBM+16]. *"Asynchronous methods for deep reinforcement learning"*.
- Schulman et al. (2015) [SML+15]. *"High-dimensional continuous control using generalized advantage estimation"*.
- Gu et al. (2016) [GLG+16]. *"Q-prop: Sample-efficient policy gradient with an off-policy critic"*.

6 Value Function Based Algorithms

6.1 Approach

Knowing the value functions, we could just remove the policy gradient completely. The advantage function $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$ tells how much better is $\mathbf{a}_t$ than the average action according to policy π, regardless of what $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is. We could have a policy π' by simply choosing the current best action. π' would be as good as π (probably better).

High-level idea for Policy Iteration:

1. Evaluate $A^\pi(s, a)$ (policy evaluation)
2. Set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

6.2 Policy Iteration with Dynamic Programming

Dynamic Programming:

- Assume we know $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$
- $\mathbf{s}$ and $\mathbf{a}$ are both discrete and small
- $V^\pi(\mathbf{s})$ can be stored in a lookup table
- $\mathcal{T}$ is a tensor

Algorithm:

1. $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\pi(\mathbf{s}))}[V^\pi(\mathbf{s}')]$
2. Set $\pi \leftarrow \pi'$

6.3 Value Iteration Algorithm

Since $\arg\max_{\mathbf{a}'_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \arg\max_{\mathbf{a}'_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$, we can simplify above algor.:

1. Set $Q^\pi(\mathbf{s},\ \mathbf{a}) \leftarrow r(\mathbf{s},\ \mathbf{a}) + \gamma \mathbb{E}[V^\pi(\mathbf{s}')]$
2. Set $V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s},\ \mathbf{a})$

6.4 Fitted Value Iteration

The above two approaches still have a table to fit the value functions. For larger state space (either continuous or discrete), when facing the curse of dimensionality (for s), we shall use neural network to evaluate the value functions.

1. $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i} r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \mathbb{E}\left[V_\phi(\mathbf{s}_i')\right]$
2. $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i ||V_\phi(\mathbf{s}_i) - \mathbf{y}_i||^2$

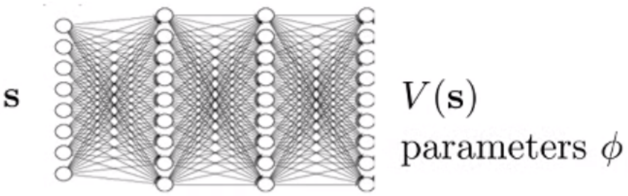


Figure 6.1: The network for value function $V_\phi(s)$ with params. ϕ.

Problem: still need to know transition dynamic.

6.5 Fully fitted Q-Iteration

Policy evaluation:

- $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))}[V^\pi(\mathbf{s}')]$ needs to know the transition models
- $Q^\pi(\mathbf{s},\ \mathbf{a}) \leftarrow r(\mathbf{s},\ \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\ \mathbf{a})}[Q^\pi(\mathbf{s}', \pi(\mathbf{s}'))]$ needs only a sample tuple $\{\mathbf{s},\ \mathbf{a},\ \mathbf{s}'\}$

Replacing since $\mathbb{E}[V(\mathbf{s}_i')] \approx \max_{\mathbf{a}'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$ into fitted value iteration algorithm, we have Fully fitted Q-iteration:

1. Collect dataset: $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)\}$ using some policy

$K\times$

2. Set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$
3. Set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i||^2$

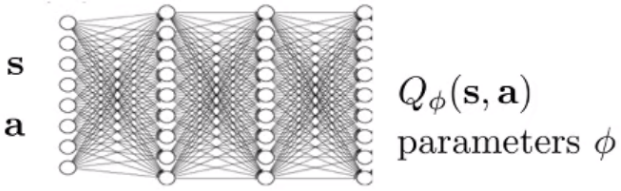


Figure 6.2: The network for Q-value function $Q_\phi(s, a)$ with params. ϕ.

+Off-policy (unlike actor-critic)

+Single network, no high-variance policy gradient

−Not really converge

6.6 Online Q-Iteration Algorithm

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ **1 sample off policy**
2. $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \dfrac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\left[Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\right]$ **1 gradient step**

Problems:

- Sequential states are strongly correlated ⇒ Replay buffer
- Target value is always changing ⇒ Target network

6.7 Exploration vs Exploitation

- Epsilon greedy

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1-\epsilon & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ \dfrac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases} \tag{6.2}$$

- Boltzmann exploration **(very large or continuous action-space)**

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t)) \tag{6.3}$$

6.8 Q-learning

Q-learning with replay buffer $\mathcal{B}$ and target network ϕ':

1. Save target network params. $\phi' \leftarrow \phi$
2. Collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
3. Sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \dfrac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \left[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{a'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)\right]\right)$

$N\times$ $K\times$

$K \in [1,4], N \approx 10,000$

6.9 Deep Q-learning

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. Sample mini-batch $\{(\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j)\}$ from $\mathcal{B}$ uniformly
3. Compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)\left(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j\right)$
5. Update ϕ': copy ϕ every N steps

The above *"Classic"* Deep Q-learning (DQN) is essentially Q-learning with $K = 1$ [MKS+15]

Improving DQN:

- Alternative: Step 5. Update $\phi' \leftarrow \tau\phi' + (1-\tau)\phi, \quad \tau = 0.999$ (Polyak averaging)
- Double Q-learning: **helps a lot, solve over-estimate problem, no downside $\Rightarrow$ should always use**

$$\text{Standard Q-learning:} \qquad y = r + \gamma Q_{\phi'}\left(s', \arg\max_{a'} Q_{\phi'}(s', a')\right) \tag{6.4}$$

$$\text{Double Q-learning:} \qquad y = r + \gamma Q_{\phi'}\left(s', \arg\max_{a'} Q_{\phi}(s', a')\right) \tag{6.5}$$

- Multi-Step returns: **helps a lot, have DOWNSIDE $\Rightarrow$ frequently use** [MSH+16]

$$\text{Q-learning target:} \qquad y_{i,t} = r_{i,t} + \gamma \max_{a_{i,t+1}} Q_{\phi'}(\mathbf{s}_{i,t+1}, a_{i,t+1}) \tag{6.6}$$

$$\text{Multi-step target:} \qquad y_{i,t} = \sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{i,t'} + \gamma^N \max_{a_{i,t+N}} Q_{\phi'}(\mathbf{s}_{i,t+N}, a_{i,t+N}) \tag{6.7}$$

+less biased target values when Q-values are inaccurate

+typically faster learning, especially early on

−Only actually CORRECT when learning on-policy

6.10 Q-learning with continuous action-space

[**TODO:**]

6.11 Tips for Q-learning

- Large replay buffer helps improve stability (1 Million)
- It takes time, be patient - might be no better than random for a while
- Start with high exploration $\Rightarrow$ gradually reduce
- Bellman error gradient can be quite large $\Rightarrow$ clip gradients / use Huber loss

$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \le \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases} \qquad \text{(Huber loss)} \tag{6.8}$$

- Run multiple random seeds, it's very ***inconsistent*** between runs.

6.12 Policy Gradient as Policy Iteration

[**TODO: math stuffs**]

6.13 References

- Watkins (1989) [Wat89]. *"Learning from delayed rewards"*.
- Riedmiller (2005) [Rie05]. *"Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method"*.
- Lange et al. (2010) [LR10]. *"Deep auto-encoder neural networks in reinforcement learning"*.
- Mnih et al. (2015) [MKS+15]. *"Human-level Control Through Deep Reinforcement Learning"*.
- Van Hasselt et al. (2016) [VHGS16]. *"Deep reinforcement learning with double q-learning"*.
- Lillicrap et al. (2015) [LHP+15]. *"Continuous control with deep reinforcement learning"*.
- Gu et al. (2016) [GLS+16]. *"Continuous deep q-learning with model-based acceleration"*.
- Wang et al. (2016) [WSH+16]. *"Dueling network architectures for deep reinforcement learning"*.
- Kalashnikov et al. (2018) [KIP+18]. *"Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation"*.

7 Optimal Control

Prior approaches are all model-free algorithms. They either assume the dynamics model is unknown or don't even attempt to learn it. On the other hands, there are times that we either do know the dynamics transition or can learn it. **Knowing the dynamics model actually does make thing easier.** These section is about what we do, how to plan through the action sequence to maximize the reward, **IF we already know the model.**

- Deterministic case vs Stochastic:

$$\mathbf{a}_1,\ldots,\mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t) \quad \text{s.t. } \mathbf{s}_{t+1} = f(\mathbf{s}_t,\mathbf{a}_t) \quad \text{(Deterministic case)} \tag{7.1}$$

$$\mathbf{a}_1,\ldots,\mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \mathbb{E}\left[\sum_{t} r(\mathbf{s}_t,\mathbf{a}_t)|\mathbf{a}_1,\ldots,\mathbf{a}_T\right] \quad \text{(Stochastic case)} \tag{7.2}$$

$$p_\theta(\mathbf{s}_1,\ldots,\mathbf{s}_T|\mathbf{a}_1,\ldots,\mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t) \quad \text{(stochastic dynamics)} \tag{7.3}$$

- Open-loop case vs Closed-loop:
  Open-loop case: we are only given $\mathbf{s}_1$ and have to plan through the whole sequence of actions $\mathbf{a}_1,\ldots,\mathbf{a}_T$. In deterministic case, it's still possible to come up with a good action plan. But for stochastic case, the randomness would probably drive us to a bad result.
  Closed-loop case: we plan once action $\mathbf{a}_t$ at a time and observe the state transition $\mathbf{s}_{t+1}$

7.1 Open-Loop Planning

Maximize objective through sequence of actions:

$$\mathbf{a}_1,\ldots,\mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} J(\mathbf{a}_1,\ldots,\mathbf{a}_T) \quad \Rightarrow \quad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A}) \tag{7.4}$$

Some stochastic optimization:

- Guess and Check (***Random Shooting Method***)
  1. Pick $\mathbf{A}_1,\ldots,\mathbf{A}_N$ from some distribution (e.g., uniform)
  2. Choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$
- Cross-entropy Method (CEM)
  1. Sample $\mathbf{A}_1,\ldots,\mathbf{A}_N$ from $p(\mathbf{A})$
  2. Evaluate $J(\mathbf{A}_1),\ldots,J(\mathbf{A}_N)$
  3. Pick M *elites* $\mathbf{A}_{i_1},\ldots,\mathbf{A}_{i_M}$ with highest values $M < N$ (usually 10%)
  4. Refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1},\ldots,\mathbf{A}_{i_M}$

NOTE: Check out CMA-ES (CEM with momentum)

The two above approaches are:

+very fast if parallelized
+extremely simple
−very harsh dimensionality limit
−only open-loop planning

- Discrete case: Monte-Carlo Tree Search (MCTS) [BPW+12]
  1. Find a leaf s_l using $TreePolicy(s_1)$
  2. Evaluate the leaf using $DefaultPolicy(s_l)$
  3. Update all values in tree between s_1 and s_l, take the best action from s_1.
     UCT Tree Policy(s_t): if s_t is not fully expanded, choose new a_t, else choose child with the highest Score(s_{t+1})
     $$Score(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$
     With $Q(s_t)$ as the reward, $N(s_t)$ as the no. times the leaf is visited.

7.2 Trajectory Optimization with Derivatives

Derivatives are hard to come by, but SOMETIMES you **CAN**, through Physics equation.

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{7.5}$$

$$= \min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, , \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T) \tag{7.6}$$

NOTE: Shooting methods vs. collocation:

- Shooting methods: optimize over actions only
  $$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, , \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$
  tends to be very sensitive with early actions and leads to numerical instability.
- Collocation: optimize over actions and states, with constraints
  $$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

Linear case: Linear Quadratic Regulator (LQR): $f(\cdot)$ has special structure.

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t \qquad \underline{\underline{\text{linear}}}\ \text{dynamics} \tag{7.7}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t \qquad \underline{\underline{\text{quadratic}}}\ \text{cost} \tag{7.8}$$

$$\mathbf{C}_T = \begin{bmatrix} \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \\ \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \end{bmatrix}; \quad \mathbf{c}_T = \begin{bmatrix} c_{\mathbf{x}_T} \\ c_{\mathbf{u}_T} \end{bmatrix} \tag{7.9}$$

NOTE: LQR and its extensions are also an open-loop plannings.

Solve for $\mathbf{u}_T$ only:

$$Q(\mathbf{x}_T, \mathbf{u}_T) = const + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_T \tag{7.10}$$

$$\text{Set } \nabla_{\mathbf{u}_T} Q(\mathbf{x}_T, \mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0 \tag{7.11}$$

$$\Rightarrow \quad \mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}.\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}) = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \tag{7.12}$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \tag{7.13}$$

$$\mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{c}_{\mathbf{u}_T} \tag{7.14}$$

$$\tag{7.15}$$

Replace Eq. 7.12 into Eq. 7.10:

$$\Rightarrow \quad V(\mathbf{x}_T) = const + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_t \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T \tag{7.16}$$

$$= const + \frac{1}{2}\mathbf{x}_T^T\mathbf{V}_T\mathbf{x}_T + \mathbf{x}_T^T\mathbf{v}_T \tag{7.17}$$

Minimize objective on last action u_T, based on x_T

u_{T-1} affects objectives through linear dynamics func. to x_T

Backward recursion:

$$\text{for } t = T \rightarrow 1 : \tag{7.18}$$

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t \tag{7.19}$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1} \tag{7.20}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = const + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t \tag{7.21}$$

$$\mathbf{u}_t \leftarrow \arg\min_{u_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t \tag{7.22}$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} \tag{7.23}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t} \tag{7.24}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t \tag{7.25}$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t \tag{7.26}$$

$$V(\mathbf{x}_t) = const + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t \tag{7.27}$$

Forward recursion:

$$\text{for } t = 1 \rightarrow T : \tag{7.28}$$

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t \tag{7.29}$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{7.30}$$

LQR for Stochastic and Non-Linear Systems

- Stochastic dynamics:

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) \tag{7.31}$$

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t \right) \tag{7.32}$$

  Solution: choose actions according to $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$
- Non-linear case: Differential Dynamic Programming (DDP) or Iterative Linear Quadratic Regulator (iLQR)

At every iteration: we ***linearize local nonlinear dynamic*** (with Taylor expansion) as a linear-quadratic system

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} \tag{7.33}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} \tag{7.34}$$

We can run LQR with dynamics $\bar{f}$, cost $\bar{c}$, state $\delta\mathbf{x}_t$ and action $\delta\mathbf{u}_t$:

$$\delta\mathbf{x}_t = \mathbf{x}_t - \widehat{\mathbf{x}}_t \tag{7.35}$$

$$\delta\mathbf{u}_t = \mathbf{u}_t - \widehat{\mathbf{u}}_t \tag{7.36}$$

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \qquad \text{with} \quad \mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \tag{7.37}$$

$$\bar{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t \qquad \text{with} \quad \begin{aligned} \mathbf{C}_t &= \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \\ \mathbf{c}_t &= \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \end{aligned} \tag{7.38}$$

⇒ iLQR (simplified pseudo code):

```
until convergence:                                                          (7.39)
    $\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                  (7.40)
    $\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                (7.41)
    $\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                  (7.42)
    Run LQR backward pass on state $\delta\mathbf{x}_t$ and action $\delta\mathbf{u}_t$       (7.43)
    Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \widehat{\mathbf{x}}_t) + \alpha\mathbf{k}_t + \widehat{\mathbf{u}}_t$   (7.44)
    Update $\widehat{\mathbf{x}}_t$ and $\widehat{\mathbf{u}}_t$ based on states and actions in forward pass   (7.45)
```

NOTE: For practical improvement, use $\alpha \in [0, 1]$. When running the forward pass, we should run over different α until we find something good.

7.3 References

- Jacobson et al. (1970) [JM70].Differential dynamic programming.
- Tassa et al. (2012) [TET12].*"Synthesis and stabilization of complex behaviors through online trajectory optimization"*.
- Levine et al. (2014) [LA14].*"Learning neural network policies with guided policy search under unknown dynamics"*.

8 Model-based RL

For many current complex situation, it's too optimistic to assume that we will know the precise model e.g., with the task of folding clothes. This section aims to learn the model.

8.1 Model-based RL v.0.5

1. Run based policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})_i\}$
2. Learn dynamic model $f(\mathbf{s},\mathbf{a})$ to minimize $\sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. Plan through $f(\mathbf{s},\mathbf{a})$ to choose actions

Problems: might go beyond to new state distribution $p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$.

- If the states are discrete, we could use cross-entropy loss. If the states are continuous, we could use squared-error loss. Generally, negative log-likelihood loss.
- Good based policy should be taken with good care.
- Particularly effective ***if*** can hand-engineer a dynamics representation with our knowledge of physics ... ⇒ need to fit only a few params.

8.2 Model-based RL v.1.0

Similar solution for distribution mismatch as in Sec. **??**.

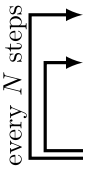
1. Run based policy $\pi_0(\mathbf{a}_t, \mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})_i\}$
2. Learn dynamic model $f(\mathbf{s},\mathbf{a})$ to minimize $\sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. Plan through $f(\mathbf{s},\mathbf{a})$ to choose actions
4. Execute these actions and **add resulting data** $\{(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})_j\}$ to $\mathcal{D}$

Problems: the model would learn faster if we correct the mistake more often.

8.3 Model-based RL v.1.5

This is much more computational expensive compared to the above.

The more you re-plan, the less perfect each individual plan needs to be ⇒ Can use shorter horizons. (YouTube).



1. Run based policy $\pi_0(\mathbf{a}_t, \mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s, a, s'})_i\}$
2. Learn dynamic model $f(\mathbf{s},\mathbf{a})$ to minimize $\sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. Plan through $f(\mathbf{s},\mathbf{a})$ to choose actions
4. Execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. Append $\{(\mathbf{s, a, s'})_j\}$ to $\mathcal{D}$

NOTE: These are all ***open-loop planning***, either stochastic or deterministic, algorithms. In other words, in step 3, given a single current state, we do the planning and output a sequence of actions $\{\mathbf{a}_t, \dots, \mathbf{a}_{t+T}\}$. Possible planning algorithms are presented in Sec. 7, e.g., random shooting, CEM, MCTS, LQR.

8.4 Uncertainty-aware Model

Problem of Model-based RL v.1.5 (Subsec. **??**): overfitting early, especially with high-dimensional data

Solution: Introduce **uncertainty estimation**
Step 3. Take action with **high expected reward**

This goes against exploration, thus depends on problems, use different strategies:

- expected value planning
- optimistic value planning
- pessimistic value planning

8.5 Uncertainty-Aware Neural Net Models

There is two kinds of uncertainty:

- *aleatoric* uncertainty (statistical data uncertainty)
- *epistemic* uncertainty (model uncertainty)

"The model is certain about the data, but we are not certain about the model". Model uncertainty is then the uncertainty about params. θ that represents the model. Usually, we estimate:

$$\arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\mathcal{D}|\theta)$$

Or estimate the exact $p(\theta|\mathcal{D})$, then predict according to $\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta$

- Use output entropy $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$: predicts aleatoric uncertainty, which is the wrong one. Thus, it's ***bad, not going to work***.

- Bayesian neural network: **complicate** [BCK+15], [GHK17].

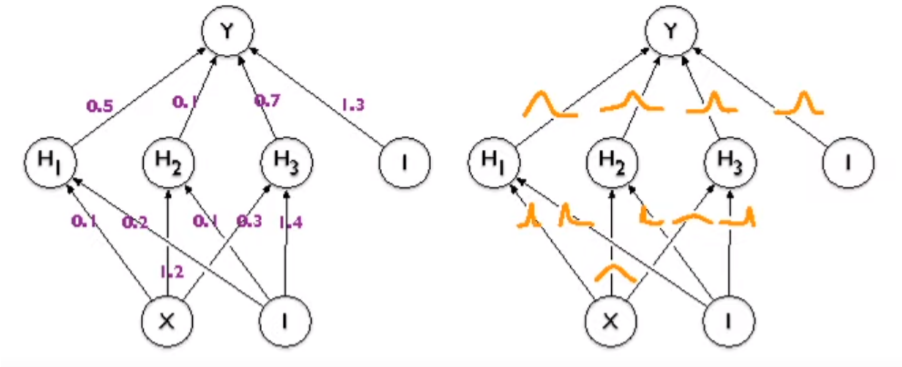


Figure 8.1: Normal neural network (left) and Bayesian neural network (right).

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D}) \qquad \text{common approximation} \tag{8.1}$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i) \qquad \text{common choice for each marginal prob.} \tag{8.2}$$

- Bootstrap ensembles: train ensemble of models (Sec. **??**). Each model (usually < 10 models) with params. θ_i is trained on a dataset $\mathcal{D}_i$, which is sampled with replacement from $\mathcal{D}$.

$$p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i) \qquad \text{mixture of delta func.} \tag{8.3}$$

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) \tag{8.4}$$

8.6 Planning with Uncertainty

As mentioned before, the model uncertainty is used in step 3 of the model-based RL algorithm (Subsec. 8.4). The change is about the reward function that we use to do optimal control for action planning:

- Before: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)$ where $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$
- Now: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} r(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$ where $f(\mathbf{s}_{t,i}, \mathbf{a}_t) = \mathbf{s}_{t+1,i}$ (deterministic case)

General procedure for candidate action sequence $\mathbf{a}_1, \ldots, \mathbf{a}_H$:

1. Sample $\theta \sim p(\theta|\mathcal{D})$
2. At each time step t, sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$
3. Calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$
4. Repeat step 1 → 3 and accumulate the average reward

8.7 References

- Deisenroth et al. (2011) [DR11]. *"PILCO: A model-based and data-efficient approach to policy search"*.
- Chua et al. (2018) [CCM+18]. *"Deep reinforcement learning in a handful of trials using probabilistic dynamics models"*.
- Nagabandi et al. (2020) [NKL+20]. *"Deep dynamics models for learning dexterous manipulation"*.
- Feinberg et al. (2018) [FWS+18a]. *"Model-based value expansion for efficient model-free reinforcement learning"*.
- Buckman et al. (2018) [BHT+18]. *"Sample-efficient reinforcement learning with stochastic ensemble value expansion"*.

9 Model-based RL with Images

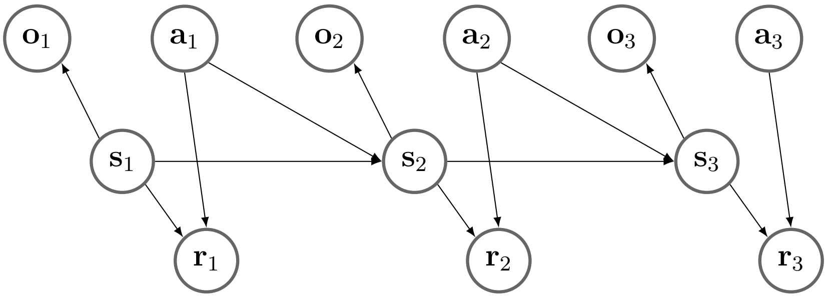
9.1 Latent Space Models

In state space (latent-space) models, the inputs are given observations, not states.

$$p(\mathbf{o}_t|\mathbf{s}_t) \quad \text{- observation model: high-dimensional, but not dynamic} \tag{9.1}$$

$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \quad \text{- dynamics model: low-dimensional, but dynamic} \tag{9.2}$$

$$p(r_t|\mathbf{s}_t, \mathbf{a}_t) \quad \text{- reward model} \tag{9.3}$$



With the above state space model, we now learn a different model:

- Before: standard (fully observable) model
  $$\max_\phi \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$$
- Now: latent space model
  $$\max_\phi \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \mathbb{E}\left[\log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})\right]$$
  the expectation with regard to $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$ **(very complicate)**

The above objective can be learn by the approximate posterior $q_\psi(\mathbf{s}_t|\mathbf{o}_{1:t}, \mathbf{a}_{1:t})$ **"encoder"**. There are also other choices for approximate posterior, depending on each problem settings:

$q_\psi(\mathbf{s}_t, \mathbf{s}_{t+1}\|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$	full smoothing posterior	+most accurate −most complicated (big RNN)
$q_\psi(\mathbf{s}_t\|\mathbf{o}_t)$	single-step encoder	+simplest −least accurate (CNN)

- If the situation is more partially observed, you would want a more accurate approximation.
- If the state can be entirely guessed by one single current observation, then this single-step posterior is a good choice.

9.2 Deterministic Single-Step Encoder

Simple special case: $q(\mathbf{s}_t, \mathbf{o}_t)$ is ***deterministic***

$$q_\psi(\mathbf{s}_t|\mathbf{o}_t) = \delta(\mathbf{s}_t = g_\psi(\mathbf{o}_t)) \Rightarrow \mathbf{s}_t = g_\psi(\mathbf{o}_t) \qquad \text{deterministic encoder} \tag{9.4}$$

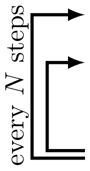
$\Rightarrow$ The goal for model-based RL with latent space is now:

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi \left[g_\psi(\mathbf{o}_{t+1,i}) | g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i} \right] + \log p_\phi \left[\mathbf{o}_{t,i} | g_\psi(\mathbf{o}_{t,i}) \right] \tag{9.5}$$

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underset{\textbf{latent space dynamics}}{\log p_\phi \left[g_\psi(o_{t+1,i}) | g_\psi(o_{t,i}), a_{t,i} \right] +} \quad \underset{\textbf{image reconstruction}}{\log p_\phi \left[o_{t,i} | g_\psi(o_{t,i}) \right] +} \quad \underset{\textbf{reward model}}{\log p_\phi \left[r_{t,i} | g_\psi(o_{t,i}) \right]}$$

9.3 Model-based RL with Latent Space Model

This is the model-based RL with latent space model, assuming deterministic observation model:



1. Run based policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{o},\ \mathbf{a},\ \mathbf{o}')_i\}$
2. Learn $p_\phi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t), p_\phi(r_t, \mathbf{s}_t), p(\mathbf{o}_t|\mathbf{s}_t), g_\psi(\mathbf{o}_t)$
3. Plan through the model to choose actions (e.g., MCTS, LQR, random shooting)
4. Execute the first planned action, observe result $\mathbf{o}'$ (MPC)
5. Append resulting $(\mathbf{o},\ \mathbf{a},\ \mathbf{o}')$ to $\mathcal{D}$

9.4 Learning in Observation Space

Im some situation, there are too many objects, thus, it's complicate to learn/build a compact state space. The better solution would be to directly learn $p(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{a}_t)$ (taken in image → split out image).

References:

- Finn et al. (2017) [FL17]. *"Deep visual foresight for planning robot motion"*.
- Ebert et al. (2017) [EFL+17]. *"Self-Supervised Visual Planning with Temporal Skip Connections."*.

Gigantic model: RNN. **[TODO: ??]**

10 Model-Based Policy Learning

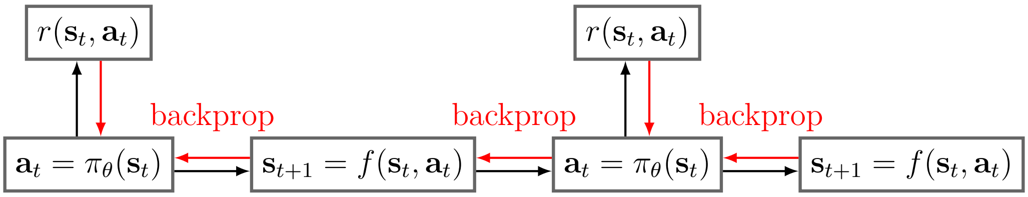
As mentioned before, model-based RL v.1.5 algorithm (Sec. 8.3) is a ***stochastic open-loop*** algorithm. The agent does see the next state, but it doesn't able to reason about the fact that more information will be available and make use out it. It simply plans the whole action sequence at each time step and assumes that it has to commit to that complete action plan. This is, in most case, **sub-optimal**. This section describes the ***closed-loop*** case, implying the agent aware that it will be able to see the state feedback and act upon it. Thus, instead of a complete action plan, the output is now a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$.

- Stochastic open-loop case: $$\begin{cases} p_\theta(\mathbf{s}_1,\dots,\mathbf{s}_T|\mathbf{a}_1,\dots,\mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t) \\ \mathbf{a}_1,\dots,\mathbf{a}_T = \underset{\mathbf{a}_1,\dots,\mathbf{a}_T}{\arg\max}\ \mathbb{E}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)|\mathbf{a}_1,\dots,\mathbf{a}_T\right] \end{cases}$$
- Stochastic closed-loop case: $$\begin{cases} p(\mathbf{s}_1,\mathbf{a}_1,\dots,\mathbf{s}_T,\mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t) \\ \pi = \underset{\pi}{\arg\max}\ \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right] \end{cases}$$

For the above policy π, there are possibly different forms for it:

- Neural net: **global policy**, which would tell us what to do regardless of the state the agent is in the whole state space.
- Time-varying linear $\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t$: **local policy**, which would be simple but only sufficient around particular area of a known trajectory

10.1 Model-based RL v.2.0


$r(\mathbf{s}_t,\mathbf{a}_t)$
$r(\mathbf{s}_t,\mathbf{a}_t)$
backprop
backprop
backprop
$\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$
$\mathbf{s}_{t+1} = f(\mathbf{s}_t,\mathbf{a}_t)$
$\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$
$\mathbf{s}_{t+1} = f(\mathbf{s}_t,\mathbf{a}_t)$

1. Run based policy $\pi_0(\mathbf{a}_t,\mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})_i\}$
2. Learn dynamic model $f(\mathbf{s},\mathbf{a})$ to minimize $\sum_i ||f(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{s}'_i||^2$
3. Back-propagate through $f(\mathbf{s},\mathbf{a})$ into the policy to optimize $\pi_\theta(\mathbf{a}_t,\mathbf{s}_t)$
4. Run $\pi_\theta(\mathbf{a}_t,\mathbf{s}_t)$, appending the tuples $(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})$ to $\mathcal{D}$

Problem:

- Similar parameter sensitivity problems as shooting methods
  The first action is way more important the the later ones.
- Similar problems to training long RNNs with Backpropagation through time (BPTT)
  Vanishing and exploding gradients

⇒ ***Solutions:***

- Use derivative-free ("model-free") planning algorithms with the model used to generate synthetic samples
  E.g.: Policy gradients has high variance, which can be reduced with lots of data, which can be generated by learned model
- Use simpler policies than neural nets
  - LQR with learned models (LQR-Fitted Local Model (FLM))
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

10.2 Model-Free Learning With a Model

This is one of the solutions for Model-based RL v.2.0 (Sec. 10.1): use the learned model to generate synthetic data for "model-free" RL algorithms, e.g., policy gradient. [PRP+18]

"Classic" Dyna [Sut90]: online Q-learning algor. that performs model-free RL with a model

1. Given state s, pick action a using exploration policy
2. Observe s' and r, to get transition (s, a, s', r)
3. Update model $\widehat{p}(s'|s, a)$ and $\widehat{r}(s, a)$ using (s, a, s')
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha \mathbb{E}_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. Repeat K times:
   6. Sample $(s, a) \sim \mathcal{B}$ from buffer of past states and actions
   7. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha \mathbb{E}_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

General "Dyna-style" model-based RL:

1. Collect some data, consisting of transitions (**s, a, s’**, r) **(1-million steps)**
2. Learn model $\widehat{p}(s'|s, a)$ (and optionally, $\widehat{r}(s, a)$)
3. Repeat K times:
   4. Sample $s \sim \mathcal{B}$ from buffer
   5. Choose action a (from $\mathcal{B}$, from π, or random)
   6. Simulate $s' \sim \widehat{p}(s'|s, a)$ (and $r = \widehat{r}(s, a)$)
   7. Train on (s, a, s', r) with model-free RL
   8. (optional) Take N more model-based steps

The above approach is:

+only requires short (as few as one step) rollouts from model

+still sees diverse states

Problem: if your model is inaccurate (which always is), the longer we roll-out the model, the more these errors compound. This leads to distribution shift, either in the model or the policy. This is also why this is suited for mostly short rollouts of the model. ⇒ Not very nice for Policy Gradients, but is okay for value-based approaches, actor-critic, etc.

Note:

- In Classic Dyna, step 5 is to choose action from buffer
- This general procedure is the basis for:
  - Model-based Acceleration (MBA) [GLS+16]
  - Model-based Value Expansion (MVE) [FWS+18b]
  - Model-based Policy Optimization (MBPO) [JFZ+19]

10.3 Local Models

This is the second solution for model-based RL v.2.0 (Sec. 10.1): instead of using neural network, we use simple policies, which is time-varying linear controller, i.e., LQR-FLM.

In order to use LQR (Sec. 7.2), we need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$, in which, knowing the model would give us $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}$. If continuous system sufficiently smooth and initial state distribution quite tight ⇒ do linearization regression at every time step

⇒ ***Idea:*** fit $\frac{df}{d\mathbf{x}_t}$ and $\frac{df}{d\mathbf{u}_t}$ around ***current trajectory / policy***

LQR-FLM Algorithm:

1. Run $p(\mathbf{u}_t|\mathbf{x}_t)$ on robot, collect $\mathcal{D} = \{\tau_i\}$
2. Fit dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t$$

$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \quad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$

3. Improve controller $p(\mathbf{u}_t|\mathbf{x}_t)$ (LQR)

Which controller to run? $p(\mathbf{u}_t|\mathbf{x}_t)$

- Version 0.5: $p(\mathbf{u}_t|\mathbf{x}_t) = \delta(\mathbf{u}_t = \hat{\mathbf{u}}_t)$ doesn't correct deviations or drift

- Version 1.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \delta\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \widehat{\mathbf{x}}_t) + \mathbf{k}_t + \widehat{\mathbf{u}}_t$
  Better, but a little too good. When fitting the dynamics, we need data to be a little bit cluster, but not too much. **Still need to be varied, for exploration and fitting.**
- Version 2.0: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \widehat{\mathbf{x}}_t) + \mathbf{k}_t + \widehat{\mathbf{u}}_t, \Sigma_t)$
  Set $\Sigma_t = \mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}$

How to fit the dynamics? $p(x_{t+1}|x_t, u_t)$ given $\{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})_i\}$

- Version 1.0: At each time step using linear regression
  $$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{A}_t\mathbf{x}_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{c}, \mathbf{N}_t); \quad \mathbf{A}_t \approx \frac{df}{d\mathbf{x}_t}; \mathbf{B}_t \approx \frac{df}{d\mathbf{u}_t}$$
  Problems: linear regression requires number of samples that scale with dimensional states
- Version 2.0: fit using Bayesian linear regression
  Use your favorite global model as prior
  ⇒ Can get away with fewer samples

How to stay close to old controller?

We want to stay close around local region of trajectories where we have linearize to approximation
⇒ Keep KL divergence small (between old and new trajectories) [LA14]

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \widehat{\mathbf{x}}_t) + \mathbf{k}_t + \widehat{\mathbf{u}}_t, \Sigma_t) \qquad \text{the controller} \tag{10.1}$$

$$p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \qquad \text{the resulting trajectory} \tag{10.2}$$

$$D_{KL}(p(\tau)||\bar{p}(\tau)) \leq \epsilon \qquad \text{constraint on KL divergence} \tag{10.3}$$

10.4 Guided Policy Search

This is the extension of local policies to global policies. However, the idea behind this, which is similar to distillation of ensemble (Sec. 5.8, AI notes), is also important in other settings.

Given many local policies, we take the data from these local policies and treat them as expert's demonstrations and combine them into a global policy. The global policy can be learned as a neural net with supervised learning from these local data.

Guided Policy Search Algorithm: [LFD+16]

1. Optimize each local policy $\pi_{LQR,i}(\mathbf{u}_t|\mathbf{x}_t)$ on initial state $\mathbf{x}_{0,i}$, w.r.t. $\widetilde{c}_{k,i}(\mathbf{x}_t, \mathbf{u}_t)$
2. Use samples from step (1) to train $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to mimic all $\pi_{LQR,i}(\mathbf{u}_t|\mathbf{x}_t)$
3. Update cost function $\widetilde{c}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = c(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

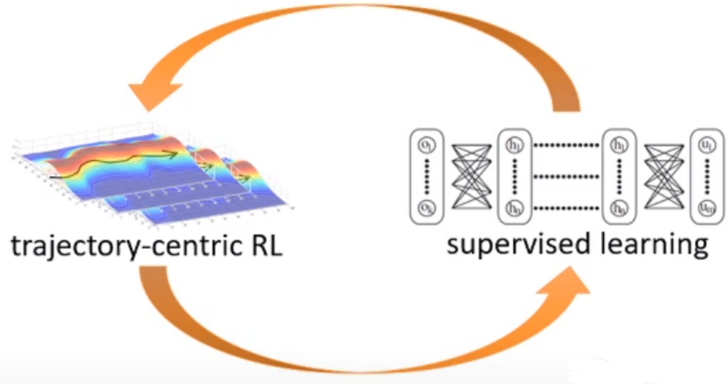


Figure 10.1: Guided policy search: algorithm sketch (src).

in which, i indexes the initial state and the local solution, k the iteration, $\widetilde{c}_{k,i}(\mathbf{x}_t, \mathbf{u}_t)$ is the modified cost, including the task reward, and the KL between $\pi_{LQR,i}$ and $\pi_\theta PP$

Divide and Conquer RL algorithm:

1. Optimize each local policy $\pi_{\theta,i}(\mathbf{u}_t|\mathbf{x}_t)$ on initial state $\mathbf{x}_{0,i}$, w.r.t. $\widetilde{r}_{k,i}(\mathbf{x}_t, \mathbf{u}_t)$
2. Use samples from step (1) to train $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to mimic all $\pi_{\theta,i}(\mathbf{u}_t|\mathbf{x}_t)$
3. Update cost function $\widetilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

10.5 References

- Levine et al. (2016) [LFD+16]. *"End-to-end training of deep visuomotor policies"*.
- Rusu et al. (2015) [RCG+15]. *"Policy distillation"*.
- Parisotto et al. (2015) [PBS15]. *"Actor-mimic: Deep multitask and transfer reinforcement learning"*.
- Ghosh et al. (2017) [GSR+17]. *"Divide-and-conquer reinforcement learning"*.
- Teh et al. (2017) [TBC+17]. *"Distral: Robust multitask reinforcement learning"*.

11 Exploration

11.1 Overview

In the setting of delayed reward, not knowing which actions would give more reward, we would want the agent to *explore*. The concerns are:

- How can an agent discover high-reward strategies that require a temporally extended sequence of complex behaviors that, individually, are not rewarding?
- How can an agent decide whether to attempt new behaviors (to discover ones with higher reward) or continue to do the best things it knows so far?

This poses a exploration and exploitation dilemma:

- *Exploration*: doing things you haven't done before, in the hopes of getting even higher reward
- *Exploitation*: doing what you know will yield the highest reward

11.1.1 Regret

Exploration can be viewed as a POMDP. Solving the POMDP to know the exact underlying distribution is overkill. We instead want a simpler strategy that is not too bad from the optimal ones. The *regret* is the metric to measure how good or bad an exploration strategy.

Definition: *Regret* is the reward difference from optimal policy at time step T:

$$\underbrace{Reg(T)}_{\text{regret at } T} = T. \underbrace{\mathbb{E}[r(a^*)]}_{\text{expected reward of the best action}} - \underbrace{\sum_{t=1}^{T} r(a_t)}_{\text{the actual sum of reward}}$$

Example: Consider the following multi-armed bandit problem:
A professor moves to a small town for work. He will stay there for 300 days. Each day, he will eat at one of three restaurants in the town. Eating at each restaurant has a different happiness distribution. Let say, the happiness distributions of each restaurant are as follows:

- Restaurant 1: $\mu = 10, \sigma = 5$
- Restaurant 2: $\mu = 8, \sigma = 8$
- Restaurant 3: $\mu = 5, \sigma = 25$

Not knowing the true happiness distribution, which strategy should the professor follow to maximize the expected happiness score?

The regrets for some exploration strategies:

- Optimal reward: Knowing the true distribution, the optimal action is to always go to restaurant 1.

  $$\mathbb{E}[r] = 300 \times 10 = 3000$$

- Explore only: the professor spends 100 days at each restaurant.

  $$\mathbb{E}[r] = 100 \times 10 + 100 \times 8 + 100 \times 5 = 2300$$
  $$\Rightarrow \rho = 3000 - 2300 = 700 \qquad \text{(regret)}$$

- Exploit only: visit each restaurant once, then stick with the one with the highest value. Assume the receive reward after 3 days are: $r_1 = 7, r_2 = 8, r_3 = 5$. After that, the expected reward would be:

  $$\mathbb{E}[r] = 7 + 8 + 5 + (300 - 3) \times 8 = 2396$$

  However, this is not the actual expected reward for the exploit only strategy.

  $$\rho = 3000 - 2670 = 330$$

- Zero regret strategy: As time goes on $T \to \infty$, the regret will approach to 0 $\rho \to 0$

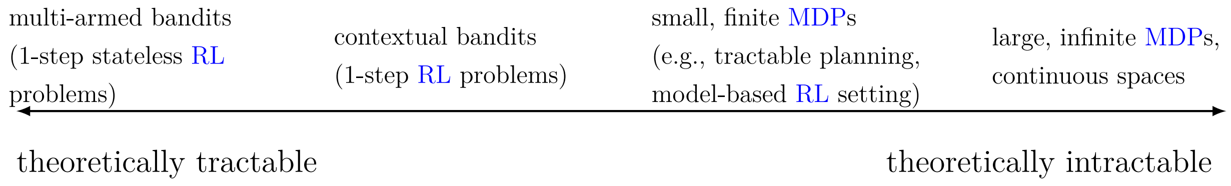
11.1.2 Optimality

Optimality: An exploration strategy is *optimal* when we compared the regret vs. the Bayes-optimal strategy. [**TODO:**]

[**TODO: Optimal exploration in small MDPs**]

11.1.3 Tractability

- With *theoretically tractable* exploration strategy, we can quantify or understand whether the given exploration strategy is optimal or sub-optimal
- With *theoretically intractable* exploration strategy, we cannot make the above estimate exactly.


multi-armed bandits (1-step stateless RL problems)
contextual bandits (1-step RL problems)
small, finite MDPs (e.g., tractable planning, model-based RL setting)
large, infinite MDPs, continuous spaces
theoretically tractable
theoretically intractable

- multi-armed bandits: can formalize exploration as POMDP identification
- contextual bandits: policy learning is trivial even with POMDP
- small, finite MDPs: can frame as Bayesian model identification, reason explicitly about value of information
- large or infinite MDPs: optimal methods don't work, but can take inspiration from optimal methods in smaller settings

11.2 Bandits Problems

11.2.1 One-Armed Bandit

One armed bandit is the slot machine. It can be represent as a MDP with one single action. The prob. distribution of the reward is unknown.

$$\mathcal{A} = \{\text{pull arm}\} \tag{11.1}$$

$$r(\text{pull arm}) = ? \tag{11.2}$$

Figure 11.1: One-armed bandit (src).

11.2.2 Multi-Armed Bandit

Multi armed bandit is a bank of multiple one-armed bandit slot machines. This problem is a 1-step stateless MDP. Different machines have different reward distributions, which are unknown, but can be learned by trials.

$$\mathcal{A} = \{\text{pull}_1, \text{pull}_2, \ldots, \text{pull}_n\} \tag{11.3}$$

$$r(a_n) = ? \tag{11.4}$$

$$\text{assume } r(a_n) \sim p(r|a_n) \tag{11.5}$$

We can define the bandits as a POMDP with the state as params. that represents the reward models. Solving the POMDP leads to the optimal exploration strategy.

$$\mathbf{s} = [\theta_1, \ldots, \theta_n]$$

But the belief state of θ is large, thus, doing this is overkill. We can do well with simpler strategies.

11.2.3 Contextual Bandits

- the reward distribution depends on some external measurable variable.
- bandits with state, essentially 1-step MDP
- **[TODO:**]

11.2.4 Bandit Variants

- Infinite Arms: there are more slot machines.
- Variable Arms: the reward distribution varies for each slot machine.
- Combinatorial Bandits: the agent has to pull more than one arm at once.
- Dueling Bandits: agent always pulls two arms, is never told about the reward, ...
- Continuous Bandits: agent has to choose interval value, like the force to the arm.
- Adversarial Arms: the agent plays against an opponent. Thus if the agent uses the same strategy, the opponent will adapt, and the Q-value of that action will change over time. E.g.: chess, tic-tac-toe.
- Strategic Arms
- and more!

11.2.5 Gradient Bandits

[TODO:]

11.2.6 Applications

There are various applications in:

- Ad serving: arms - possible ads, reward - a click
- Website optimization: arms: possible website options, reward - user engagement
- Clinical trials: arms: possible medications, reward - health outcomes

11.3 Exploration Strategies for Bandits Problem

11.3.1 ϵ-first

[TODO:]

11.3.2 ϵ-greedy

[TODO: formula]

With the example in Sec. 11.1.1, assuming $\epsilon = 10\%$, the professor spend 10% of the days (30 days) to try non-optimal restaurants, and the rest to exploit the current belief about the best restaurant.

$\rho \approx 100$

11.3.3 Optimistic Exploration

The high-level idea is to try each arm until you are sure it's not great.

- Keep track of average reward $\widehat{\mu}_a$ for each action a
- Optimistic estimate: $a = \arg\max_a[\widehat{\mu}_a + C\sigma_a]$.
  Choose the action with either current high average reward, or with high covariance σ_a

Upper Confidence Bounce (UCB):

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right], \qquad \begin{cases} Q_t(a) - \text{current reward belief} \\ N_t(a) - \text{no. times taking action } a \\ t - \text{current no. time step} \end{cases}$$

UCB-1 use Chernoff - Hoeffding Inequality, with $Reg(T) = \mathcal{O}(\log T)$:

$$C_j(t) = \sqrt{\frac{\log(n)}{T_j(t)}}$$

$$a = \arg\max_a \left[\widehat{\mu}_a + \sqrt{\frac{2\ln T}{N(a)}} \right]$$

NOTE:

- lots of other functions work as well, as long as they decrease with $N(a)$
- UCB is more difficult than ϵ-greedy to extend beyond bandits to more general RL problems (nonstationary problems, large state spaces)

11.3.4 Probability Matching / Posterior Sampling

Optimistic strategy doesn't try to model the uncertainty. It is model-free approach, simply keeps the average rewards and the number of times an action is taken. For the multi bandits problem, assuming some reward model $r(a_i) \sim p_{\theta_i}(r_i)$, we could instead keep a belief $\widehat{p}(\theta_1, \ldots, \theta_n)$ over the params. and solve the POMDP with $\mathbf{s} = [\theta_1, \ldots, \theta_n]$.

Idea: Thompson sampling [CL11]

1. Sample $\theta_1, \ldots, \theta_n \sim \widehat{p}(\theta_1, \ldots, \theta_n)$
2. Pretend the model $\theta_1, \ldots, \theta_n$ is correct
3. Take the optimal action $a = \arg\max_{a} \mathbb{E}_{\theta_a}[r(a)]$
4. Update the model

- Harder to analyze theoretically
- Can work very well empirically

11.3.5 Information Gain

This method is even more explicitly model-based.

Bayesian experimental design: aim to determine some unknown latent variable z but can only take actions to learn about it.

$$\begin{aligned}
&\mathcal{H}(\widehat{p}(z)) && \text{– the current entropy of } z \text{ estimate}\\
&\mathcal{H}(\widehat{p}(z)|y) && \text{– the entropy of } z \text{ estimate after observation } y\\
&IG(z,y) = \mathbb{E}_y\Big[\mathcal{H}(\widehat{p}(z)) - \mathcal{H}(\widehat{p}(z)|y)\Big] && \text{– the } \textit{information gain} \text{ about } z \text{ after } y\\
&IG(z,y|a) = \mathbb{E}_y\Big[\mathcal{H}(\widehat{p}(z)) - \mathcal{H}(\widehat{p}(z)|y)|a\Big] && \text{– usually condition on action } a
\end{aligned}$$

- e.g., y might be optimal action a^* or the reward $r(a)$)
- We want to take actions and receive y such that we get high Information Gain (IG) about z

E.g., bandit algor. [RVR14]:

$$y = r_a \qquad \text{– reward for action } a \tag{11.6}$$

$$z = \theta_a \qquad \text{– params. of model } p(r_a) \tag{11.7}$$

$$g(a) = IG(\theta_a, r_a|a) \qquad \text{– information gain of } a \tag{11.8}$$

$$\Delta(a) = \mathbb{E}[r(a^*) - r(a)] \qquad \text{– expected suboptimality of } a \tag{11.9}$$

$$\Rightarrow a = \arg\min_{a} \frac{\Delta(a)^2}{g(a)} \tag{11.10}$$

the high-level idea is to take the least suboptimal action that give us more information.

[TODO: Bayesian Model-based RL]

[TODO: Probably Approximately Correct (PAC) exploration]

11.4 Exploration Strategies in RL

11.4.1 Information Theoretic Quantities in RL

$$\begin{aligned}
&\pi(\mathbf{s}) = p_\pi(\mathbf{s}) && \text{– state } \textit{marginal} \text{ distribution of policy } \pi\\
&\mathcal{H}(\pi(\mathbf{s})) && \text{– state } \textit{marginal} \text{ entropy of policy } \pi\\
&\mathcal{I}(\mathbf{x};\mathbf{y}) = D_{KL}\Big(p(\mathbf{x},\mathbf{y})||p(\mathbf{x})p(\mathbf{y})\Big) && \text{– mutual information (math notes)}\\
&\mathcal{I}(\mathbf{s}_{t+1};\mathbf{a}_t) = \mathcal{H}(\mathbf{s}_{t+1}) - \mathcal{H}(\mathbf{s}_{t+1}|\mathbf{a}_t) && \text{– } \textit{empowerment}
\end{aligned}$$

Intuition: we want a large empowerment because: A large entropy $\mathcal{H}(\mathbf{s}_{t+1})$ implies that there are many possible next states. A small entropy $\mathcal{H}(\mathbf{s}_{t+1}|\mathbf{a}_t)$ implies that given current action, it's easy to determine where the state will landed.

11.4.2 Optimistic Exploration in Deep RL

For bandits problem, UCB chooses the action with $a = \arg\max_a \hat{\mu}_a + \sqrt{\frac{2\ln T}{N(a)}}$ (Subsec. 11.3.3). The term $\sqrt{\frac{2\ln T}{N(a)}}$ can be considered as "exploration bonus". Lots of functions work, as long as they decrease with the count $N(a)$.

For MDPs, we use similar state counts $N(\mathbf{s},\ \mathbf{a})$ or $N(\mathbf{s})$ to add *exploration bonus*:

$$r^+(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \mathcal{B}(N(s)) \qquad \text{– updated reward function} \tag{11.11}$$

$$\mathcal{B}(N(s)) \qquad \text{– bonus that decreases with } N(\mathbf{s}) \tag{11.12}$$

+simple addition to any model-free RL algor.

−need to tune bonus weight

In most settings of continuous state and action space, the agent might not literally visit the exact state again.
⇒ ***Pseudo-counts:*** to count similar states $\hat{N}(\mathbf{s})$ or $\hat{N}(\mathbf{s},\mathbf{a})$ [BSO+16]

1. fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far
2. take a step i and observe $\mathbf{s}_i$
3. fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \bigcup \mathbf{s}_i$
4. use $p_\theta(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$
5. set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_\theta(\mathbf{s}_i), \qquad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_\theta(\mathbf{s}_i)} p_\theta(\mathbf{s}_i)$$

Types of exploration bonus:

- UCB: $$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{2 \ln n}{N(\mathbf{s})}} \tag{11.13}$$
- MBIE-EB: [SL08; BSO+16] $$\mathcal{B}(N(\mathbf{s})) = \sqrt{\frac{1}{N(\mathbf{s})}} \tag{11.14}$$
- BEB: [KN09] $$\mathcal{B}(N(\mathbf{s})) = \frac{1}{N(\mathbf{s})} \tag{11.15}$$

Kind of model: $p_\theta(\mathbf{s})$ need to output densities, but not necessarily produce great samples

- CTS model [BSO+16]
- stochastic neural networks
- compression length
- EX2

There are similar count style approaches:

- Counting with hashes: counts states but in different space [THF+17].
- Implicit density modeling with exemplar models: Use a classifier to classify whether a state is **novel** or not. [FCRL17]
- Heuristics estimation of counts via errors: Given buffer $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$
  - Fit $\hat{f}_\theta(\mathbf{s},\mathbf{a})$ to **target** function $f^*(\mathbf{s},\mathbf{a})$
  - Use $\mathcal{E}(\mathbf{s},\mathbf{a}) = \left|\left|\hat{f}_\theta(\mathbf{s},\mathbf{a}) - f^*(\mathbf{s},\mathbf{a})\right|\right|^2$ as exploration bonus

  Choice for $f^*(\mathbf{s},\mathbf{a})$: [BES+18]
  - $f^*(\mathbf{s},\mathbf{a}) = \mathbf{s}'$ as next state transition
  - $f^*(\mathbf{s},\mathbf{a}) = f_\phi(\mathbf{s},\mathbf{a})$, where ϕ is a *random* params. vector

11.4.3 Posterior Sampling in Deep RL

The MDP analog for θ of the bandits problem (Subsec. 11.3.4) is the Q-function.

Idea:

1. sample Q-function Q from $p(Q)$
2. act according to Q for one episode
3. update $p(Q)$

Bootstrapped DQN algorithm: to represent the distribution of a function [OBP+16]

- given a dataset $\mathcal{D}$, resample with replacement N times → $\mathcal{D}_1, \ldots, \mathcal{D}_N$

- train each model f_{θ_i} on $\mathcal{D}$ (**NOTE:** shared network with multi heads)
- to sample from $p(Q)$, sample $i \in [1, \ldots, N]$ and use model f_{θ_i}

  +no change to original reward function

  −very good bonuses often do better

11.4.4 Information Gain in Deep RL

Extending Subsec. 11.3.5 in the context of deep RL:

- IG on reward $r(\mathbf{s,a})$: not very useful if reward is sparse
- IG on state density $p(\mathbf{s})$: strange, but makes sense!
- IG on dynamics $p(\mathbf{s}'|\mathbf{s,a})$: good proxy for learning MDP, though still heuristics
- Generally intractable to use IG exactly, thus, we will ***have to approximate something***

Few options for approximation of IG are:

- Prediction gain: $\log p_{\theta'}(\mathbf{s}) - \log p_{\theta}(\mathbf{s})$ [BSO+16]

  Intuition: if the density change a lot, the state was novel
- Variational inference: $D_{KL}\Big(p(z|y)||p(z)\Big)$

  Learn about *transitions* $p_\theta(s_{t+1}|s_t, a_t)$, with $z = \theta$ and $y = (s_{t+1}|s_t, a_t)$

  $$\Rightarrow D_{KL}\Big(p(\theta|h, s_t, a_t, s_{t+1})||p(\theta|h)\Big) \qquad \text{– IG from a new transition} \tag{11.16}$$

  $$h \qquad \text{– history of all prior transitions} \tag{11.17}$$

  Intuition: a transition is more informative if it causes belief over θ to change a lot.

  Idea: [HCD+16]
  - Use variational inference to estimate $q(\theta|\phi) \approx p(\theta|h)$
  - Given new transition (s, a, s'), update ϕ to get ϕ'
  - Use $D_{KL}\Big(q(\theta|\phi')||q(\theta|\phi)\Big)$ as approximate bonus of IG

  +appealing mathematical formalism

  −models are more complex, generally harder to use effectively

Similar works: exploration with model errors: [Sch10; SLA15]

11.5 Unsupervised Learning of Diverse Behaviors

The three above approaches are modifications to existing RL approaches to boost exploration behaviors, which is in the context of given existing task and reward. This section concerns situations there is no specified task, no sparse or delay reward, but ***no reward at all***.

- Learn skills without supervision, then use them to accomplish goals
- Learn sub-skills to use with hierarchical RL
- Explore the space of possible behaviors

11.5.1 Goal-proposed Mechanism

Intuition: To prepare for an unknown goal in the future, the robot will have an unsupervised learning phase, in which it trains itself with self-proposed goals. The *goal*, which is to reach an underlying state z, are inferred from current observation x, using variational inference models.

Algorithm: unsupervised goal-proposed mechanism

1. Propose goal: $z_g \sim p(z), x_g \sim p_\theta(x_g, z_g)$
2. Attempt to reach goal using $\pi(a|x, x_g)$, reach $\bar{x}$
3. Use data o update π
4. Use data to update $p_\theta(x_g|z_g), q_\phi(z_g|x_g)$

How to have diverse goals?

The initial policy π might ends up with a specific state distribution $\pi(\mathbf{s}) = p_\pi(\mathbf{s})$, which is not the marginal state distribution $p(\mathbf{s}) \neq \pi(\mathbf{s})$. If we use these samples to update the variational inference model, the model will learn and generate more samples similar to the seen samples. We don't want this to happen, thus, apply exploration ideas. [NPD+18; PDL+19]

- Standard MLE: $\theta, \phi \leftarrow \underset{\theta,\phi}{\arg\max} \mathbb{E}[\log p(\bar{x})]$
- Weighted MLE: $\theta, \phi \leftarrow \underset{\theta,\phi}{\arg\max} \mathbb{E}[w(\bar{x}) \log p(\bar{x})], \quad w(\bar{x}) = p_\theta(\bar{x})^\alpha, \quad \alpha \in [-1, 0)$

NOTE: the goal of unsupervised learning of diverse behaviors

- Updating the variational inference model implies maximizing the entropy $\mathcal{H}(p(G))$ to have diverse goals
- Updating the policy $\pi(a|S, G)$ is minimizing $\mathcal{H}(p(G|S))$, because as π gets better, the final state S gets closer to G
- Thus, this algorithm is maximizing the empowerment $\max \mathcal{I}(S; G)$, good exploration $\mathcal{H}(p(G))$ and effective goal reaching $\mathcal{H}(p(G|S))$.

11.5.2 State Distribution-Matching Formulation

The first three types of algor.s, especially optimistic and IG, have an intrinsic motivation to reward visiting ***novel*** states by adding an exploration bonus. This bonus concerns with whether the state has been visited *often* before.

$$\tilde{r}(\mathbf{s}) = r(\mathbf{s}) - \log p_\pi(\mathbf{s}) = r(\mathbf{s}) - \log \pi(\mathbf{s}) \tag{11.18}$$

A general procedure could be described as:

1. update $\pi(\mathbf{a}|\mathbf{s})$ to maximize $\mathbb{E}_\pi[\tilde{r}(\mathbf{s})]$
2. update $p_\pi(\mathbf{s})$ to fit state marginal

In the case of unsupervised learning with no reward $r(\mathbf{s})$, the goal will then to simply to maximize the exploration bonus $\tilde{r}(\mathbf{s}) = -\log p_\pi(\mathbf{s})$ (Eq. 11.18).

Problem: the state prob. is under expectation of the current policy π. Thus, the policy π will keep changing to visit states that it hasn't been often before.

Solution: the state marginal matching problem - learn $\pi(\mathbf{a}|\mathbf{s})$ to minimize $D_{KL}\Big(p_\pi(\mathbf{s})||p^*(\mathbf{s})\Big)$

$$\tilde{r}(\mathbf{s}) = \log p^*(\mathbf{s}) - \log p_\pi(\mathbf{s}) \tag{11.19}$$

1. learn $\pi_k(\mathbf{a}|\mathbf{s})$ to maximize $\mathbb{E}_\pi[\tilde{r}(\mathbf{s})]$
2. update $p_{\pi_k}(\mathbf{s})$ to fit *all states seen so far* (not just the state of the current policy π_k)
3. return $\pi^*(\mathbf{a}|\mathbf{s}) = \sum_k \pi_k(\mathbf{a}|\mathbf{s})$

Proof: [LEP+19; HKS+19]

11.5.3 State Coverage

Is coverage of valid states a good exploration objective?

- Skew-Fit: $\max \mathcal{H}(p(G)) - \mathcal{H}(p(G|S)) = \max \mathcal{I}(S:G)$ [PDL+19]
- SMM (special case where $p^*(\mathbf{s}) = C$): $\max \mathcal{H}(p_\pi(S))$ [LEP+19]

"Eysenbach's Theorem": If at test time, an *adversary* will possibly choose the *hardest/worst* goal G, which goal distribution should we use for *training*?

Answer: Choose $p(G) = \arg\max_{p} \mathcal{H}(p(G))$, which is the uniform distribution to maximize state entropy. [HKS+19; GEF+18]

11.5.4 Covering the Space of Skills

Reaching diverse **goals** is not the same as performing diverse **tasks**. Not all behaviors can be captured by **goal-reaching**. [GRW16; EGI+18]

Intuition: different **skills** should visit different **state-space regions**.

$$\pi(\mathbf{a}|\mathbf{s}, z) = \arg\max_{\pi} \sum_{z} \mathbb{E}_{\mathbf{s}\sim\pi(\mathbf{s}|z)}[r(\mathbf{s}, z)] \tag{11.20}$$

$$r(\mathbf{s}, z) = \log p(z|\mathbf{s}) \qquad \text{reward states that are unlikely for other } z' \neq z \tag{11.21}$$

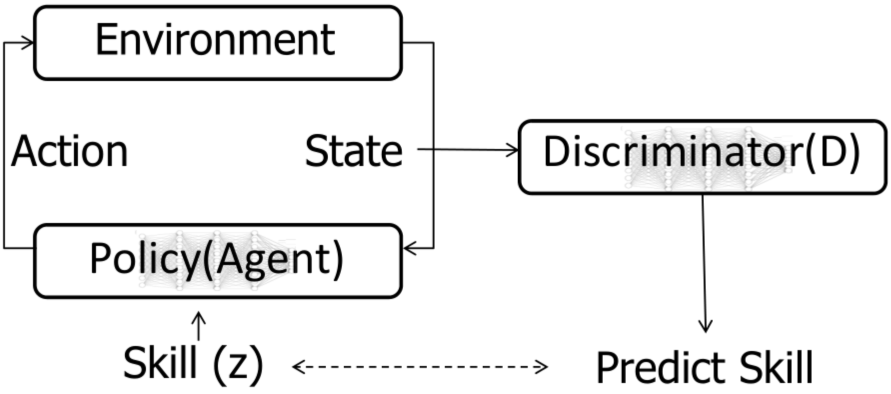


Figure 11.2: Diversity is All You Need. [EGI+18].

Connection to mutual information

$$\mathcal{I}(z, \mathbf{s}) = \mathcal{H}(z) - \mathcal{H}(z|\mathbf{s}) \tag{11.22}$$

- $\mathcal{H}(z)$: maximized by using uniform prior $p(z)$
- $\mathcal{H}(z|\mathbf{s})$: minimized by maximizing $r(\mathbf{s}, z) = \log p(z|\mathbf{s})$

11.6 References

- Schmidhuber (1991) [Sch91]. *"A possibility for implementing curiosity and boredom in model-building neural controllers"*.
- Stadie et al. (2015) [SLA15]. *"Incentivizing exploration in reinforcement learning with deep predictive models"*.
- Osband et al. (2016) [OBP+16]. *"Deep exploration via bootstrapped DQN"*.
- Houthooft et al. (2016) [HCD+16]. *"Vime: Variational information maximizing exploration"*.
- Bellemare et al. (2016) [BSO+16]. *"Unifying count-based exploration and intrinsic motivation"*.
- Tang et al. (2017) [THF+17]. *"# exploration: A study of count-based exploration for deep reinforcement learning"*.

- Fu et al. (2017) [FCRL17]. *“EX2: Exploration with exemplar models for deep reinforcement learning”.*

12 Control as Inference

- *Inference* means the process of inferring something, or a conclusion reached on the basis of evidence and reasoning.
- *Probabilistic inference* is the task of deriving the prob. of one or more random variables taking a specific value or set of values.
- *Variational inference* lets us approximate a high-dimensional Bayesian posterior with a simpler variational distribution by solving an optimization problem.

12.1 Probabilistic Graphical Model

Good behavior most of the time has minor mistakes here and there, though overall it is optimal. Thus, good behavior is rather suboptimal. Traditional optimal control doesn't not consider stochastic suboptimal behaviors like that.

- Traditional probabilistic model of optimal control

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

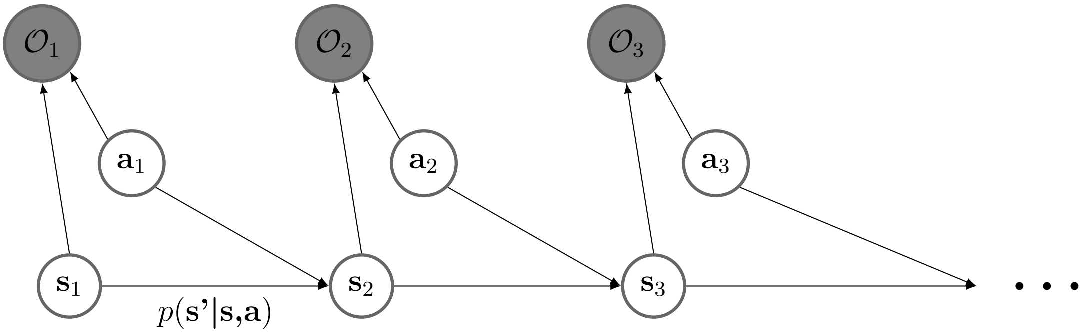
- Probabilistic graphical model: $\mathcal{O}_t$ as the optimality variable

$$p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = p(\tau) \tag{12.1}$$

$$p(\mathcal{O}|\mathbf{s}_t, \mathbf{a}_t) = \exp(r(\mathbf{s}_t, \mathbf{a}_t)) \tag{12.2}$$

$$p(\tau|\mathcal{O}_{1:T}) = \frac{p(\tau, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} \tag{12.3}$$

$$\propto p(\tau) \prod_t \exp(r(\mathbf{s}_t, \mathbf{a}_t)) = p(\tau) \exp\left(\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right) \tag{12.4}$$



12.2 Control as Inference

Here, *inference* implies planning. To do inference, we compute the three following terms:

$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t)$ – backward messages

$p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{1:T})$ – policy

$\alpha_t(\mathbf{s}_t) = p(\mathbf{s}_t|\mathcal{O}_{1:t-1})$ – forward messages

12.2.1 Backward Messages Computation

Backward messages $\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t)$ is the prob. of optimality from the current time step t till the end T, given the current state. $\mathbf{s}_t$ and action $\mathbf{a}_t$.

$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t)$ – the state-action backward message

$\beta_t(\mathbf{s}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t)$ – the state backward message

Given:

$p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) \propto \exp(r(\mathbf{s}_t, \mathbf{a}_t))$ prob. of current optimality given state and action

$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ transition prob.

Formulation:

$$\begin{aligned}
\beta_t(\mathbf{s}_t, \mathbf{a}_t) &= p(\mathcal{O}_{t:T}|\mathbf{s}_t, \mathbf{a}_t) && (12.5)\\
&= \int p(\mathcal{O}_{t:T}, \mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} && (12.6)\\
&= \int p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} && (12.7)\\
&= \int p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}) \underbrace{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}_{\text{known}} \underbrace{p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t)}_{\text{known}} d\mathbf{s}_{t+1} && (12.8)\\
&= p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) \int p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_{t+1} && (12.9)\\
\Rightarrow \beta_t(\mathbf{s}_t, \mathbf{a}_t) &= p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[\beta_{t+1}(\mathbf{s}_{t+1})\right] && (12.10)\\
p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}) &= \int \underbrace{p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}, \mathbf{a}_{t+1})}_{\text{backward message } \beta_{t+1}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})} \underbrace{p(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})}_{\text{action prior}} d\mathbf{a}_{t+1} && (12.11)\\
\Rightarrow \beta_t(\mathbf{s}_t) &= \mathbb{E}_{\mathbf{a}_t \sim p(\mathbf{a}_t|\mathbf{s}_t)} \left[\beta_t(\mathbf{s}_t, \mathbf{a}_t)\right] \quad \text{(assuming uniform action prior } p(\mathbf{a}_t|\mathbf{s}_t)) && (12.12)
\end{aligned}$$

Recursive algorithm:

$$\beta_T(\mathbf{s}_T, \mathbf{a}_T) = p(\mathcal{O}_T|\mathbf{s}_T, \mathbf{a}_T) = \exp(r(\mathbf{s}_T, \mathbf{a}_T)) \tag{12.13}$$

$$\text{for } t = T-1 \rightarrow 1: \tag{12.14}$$

$$\beta_t(\mathbf{s}_t, \mathbf{a}_t) = p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t)\mathbb{E}_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\left[\beta_{t+1}(\mathbf{s}_{t+1})\right] \tag{12.15}$$

$$\beta_t(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim p(\mathbf{a}_t|\mathbf{s}_t)}\left[\beta_t(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{12.16}$$

Further examination:

let $V_t(\mathbf{s}_t) = \log \beta_t(\mathbf{s}_t)$

let $Q_t(\mathbf{s}_t, \mathbf{a}_t) = \log \beta_t(\mathbf{s}_t, \mathbf{a}_t)$

$V_t(\mathbf{s}_t) = \log \int \exp\Big(Q_t(\mathbf{s}_t, \mathbf{a}_t)\Big) d\mathbf{a}_t$

$V_t(\mathbf{s}_t) \rightarrow \max_{\mathbf{a}_t} Q_t(\mathbf{s}_t, \mathbf{a}_t)$ as $Q_t(\mathbf{s}_t, \mathbf{a}_t)$ gets bigger! ⇒ "soft max"

$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log \mathbb{E}[\exp V_{t+1}(\mathbf{s}_{t+1})]$

$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + V_{t+1}(\mathbf{s}_{t+1})$ in case of deterministic transition

The above looks a lot like Bellman's equation ⇒ log of β_t is "Q-function-like"

NOTE: Value functions are backward messages.

12.2.2 The Action Prior

Re-examine the action prior when computing the backward messages. Before, we assume it was uniform (as a constant to be ignored). If it's not uniform:

$$p(\mathcal{O}_{t+1:T}|\mathbf{s}_{t+1}) = \int \beta_{t+1}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})p(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})d\mathbf{a}_{t+1} = \beta_{t+1}(\mathbf{s}_{t+1}) \tag{12.17}$$

$$\Rightarrow V(\mathbf{s}_t) = \log \int \exp\Big(Q_t(\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t|\mathbf{s}_t)\Big) d\mathbf{a}_t = \log \beta_t(\mathbf{s}_t) \tag{12.18}$$

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log \mathbb{E}[\exp V_{t+1}(\mathbf{s}_{t+1})] \qquad \text{(before)} \tag{12.19}$$

$$\tilde{Q}_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \log p(\mathbf{a}_t|\mathbf{s}_t) + \log \mathbb{E}[\exp V_{t+1}(\mathbf{s}_{t+1})] \qquad \text{(modified)} \tag{12.20}$$

$$\Rightarrow V(\mathbf{s}_t) = \log \int \exp\Big(\tilde{Q}_t(\mathbf{s}_t, \mathbf{a}_t)\Big) d\mathbf{a}_t \tag{12.21}$$

Thus, a simple modification to the reward will take account of the non-uniform action prior.
⇒ we can assume uniform action prior without loss of generality.

12.2.3 Policy Computation

The policy $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{1:T})$ is the current action prob. given the current state $\mathbf{s}_t$ that the whole trajectory is optimal.

Given:

$$
\begin{aligned}
&p(\mathcal{O}_t|\mathbf{s}_t,\mathbf{a}_t) \propto \exp(r(\mathbf{s}_t,\mathbf{a}_t)) && \text{– prob. of current optimality given state and action}\\
&p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t) && \text{– transition prob.}\\
&\beta_t(\mathbf{s}_t,\mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t,\mathbf{a}_t) && \text{– backward messages}\\
&\beta_t(\mathbf{s}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t) && \text{– backward state messages}
\end{aligned}
$$

Formulation:

$$
\begin{aligned}
p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{1:T}) = \pi(\mathbf{a}_t|\mathbf{s}_t) &= p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{t:T}) && (12.22)\\
&= \frac{p(\mathbf{a}_t,\mathbf{s}_t|\mathcal{O}_{t:T})}{p(\mathbf{s}_t|\mathcal{O}_{t:T})} && (12.23)\\
&= \frac{p(\mathcal{O}_{t:T}|\mathbf{a}_t,\mathbf{s}_t)p(\mathbf{a}_t,\mathbf{s}_t)/p(\mathcal{O}_{t:T})}{p(\mathcal{O}_{t:T}|\mathbf{s}_t)p(\mathbf{s}_t)/p(\mathcal{O}_{t:T})} \qquad \text{(Bayes rule)} && (12.24)\\
&= \frac{p(\mathcal{O}_{t:T}|\mathbf{a}_t,\mathbf{s}_t)}{p(\mathcal{O}_{t:T}|\mathbf{s}_t)}\frac{p(\mathbf{a}_t,\mathbf{s}_t)}{p(\mathbf{s}_t)} = \frac{\beta_t(\mathbf{s}_t,\mathbf{a}_t)}{\beta_t(\mathbf{s}_t)}p(\mathbf{a}_t|\mathbf{s}_t) && (12.25)\\
\Rightarrow \pi(\mathbf{a}_t|\mathbf{s}_t) &= \frac{\beta_t(\mathbf{s}_t,\mathbf{a}_t)}{\beta_t(\mathbf{s}_t)} \qquad \text{(assume uniform action prior } p(\mathbf{a}_t|\mathbf{s}_t)\text{=const)} && (12.26)
\end{aligned}
$$

Recursive algorithm:

$$
\begin{aligned}
&\text{for } t = T-1 \to 1: && (12.27)\\
&\quad Q_t(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \log \mathbb{E}\left[\exp(V_{t+1}(\mathbf{s}_{t+1}))\right] && (12.28)\\
&\quad V_t(\mathbf{s}_t) = \log \int \exp\Big(Q_t(\mathbf{s}_t,\mathbf{a}_t)\Big)d\mathbf{a}_t && (12.29)
\end{aligned}
$$

Policy computation with value functions:

$$
\begin{aligned}
&\pi(\mathbf{a}_t|\mathbf{s}_t) = \frac{\beta_t(\mathbf{s}_t,\mathbf{a}_t)}{\beta_t(\mathbf{s}_t)} \quad \text{with} \begin{cases} V_t(\mathbf{s}_t) = \log \beta_t(\mathbf{s}_t) \\ Q_t(\mathbf{s}_t,\mathbf{a}_t) = \log \beta_t(\mathbf{s}_t,\mathbf{a}_t) \end{cases} && (12.30)\\
&\Rightarrow \pi(\mathbf{a}_t|\mathbf{s}_t) = \exp\Big(Q_t(\mathbf{s}_t,\mathbf{a}_t) - V_t(\mathbf{s}_t)\Big) = \exp\Big(A_t(\mathbf{s}_t,\mathbf{a}_t)\Big) && (12.31)
\end{aligned}
$$

⇒ Soft advantage function with temperature:

$$
\pi(\mathbf{a}_t|\mathbf{s}_t) = \exp\left(\frac{1}{\alpha}A_t(\mathbf{s}_t,\mathbf{a}_t)\right) \qquad \begin{cases} \alpha \to 0: & \text{more deterministic} \\ \alpha \to 1: & \text{classical inference framework} \end{cases} \tag{12.32}
$$

12.2.4 Forward Messages Computation

The forward messages $\alpha_t(\mathbf{s}_t) = p(\mathbf{s}_t|\mathcal{O}_{1:t-1})$ is the prob. of the state $\mathbf{s}_t$ given that all previous trajectory steps are so far optimal.

Given:

$p(\mathcal{O}_t|\mathbf{s}_t,\mathbf{a}_t) \propto \exp(r(\mathbf{s}_t,\mathbf{a}_t))$ – prob. of current optimality given state and action

$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$ – transition prob.

$\alpha(\mathbf{s}_1) = p(\mathbf{s}_1)$ – usually known

Formulation:

$$\alpha(\mathbf{s}_t) = p(\mathbf{s}_t|\mathcal{O}_{1:t-1}) \tag{12.33}$$

$$= \int p(\mathbf{s}_t,\mathbf{s}_{t-1},\mathbf{a}_{t-1}|\mathcal{O}_{1:t-1})d\mathbf{s}_{t-1}d\mathbf{a}_{t-1} \tag{12.34}$$

$$= \int p(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_{t-1},\mathcal{O}_{1:t-1})p(\mathbf{a}_{t-1}|\mathbf{s}_{t-1},\mathcal{O}_{1:t-1})p(\mathbf{s}_{t-1}|\mathcal{O}_{1:t-1})d\mathbf{s}_{t-1}d\mathbf{a}_{t-1} \tag{12.35}$$

$$= \int \underbrace{p(\mathbf{s}_t|\mathbf{s}_{t-1},\mathbf{a}_{t-1})}_{\text{given}} p(\mathbf{a}_{t-1}|\mathbf{s}_{t-1},\mathcal{O}_{t-1})p(\mathbf{s}_{t-1}|\mathcal{O}_{1:t-1})d\mathbf{s}_{t-1}d\mathbf{a}_{t-1} \tag{12.36}$$

$$p(\mathbf{a}_{t-1}|\mathbf{s}_{t-1},\mathcal{O}_{t-1})p(\mathbf{s}_{t-1}|\mathcal{O}_{1:t-1}) = \frac{p(\mathcal{O}_{t-1}|\mathbf{s}_{t-1},\mathbf{a}_{t-1})p(\mathbf{a}_{t-1}|\mathbf{s}_{t-1})}{p(\mathcal{O}_{t-1}|\mathbf{s}_{t-1})}\frac{p(\mathcal{O}_{t-1}|\mathbf{s}_{t-1})p(\mathbf{s}_{t-1}|\mathcal{O}_{1:t-2})}{p(\mathcal{O}_{t-1}|\mathcal{O}_{1:t-2})} \tag{12.37}$$

$$= \underbrace{p(\mathcal{O}_{t-1}|\mathbf{s}_{t-1},\mathbf{a}_{t-1})}_{\text{given}}\underbrace{p(\mathbf{a}_{t-1}|\mathbf{s}_{t-1})}_{\text{action prior}}\frac{\overbrace{p(\mathbf{s}_{t-1}|\mathcal{O}_{1:t-2})}^{\alpha_{t-1}(\mathbf{s}_{t-1})}}{p(\mathcal{O}_{t-1}|\mathcal{O}_{1:t-2})} \tag{12.38}$$

The state marginal:

$$p(\mathbf{s}_t|\mathcal{O}_{1:T}) = \frac{p(\mathbf{s}_t,\mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} = \frac{\overbrace{p(\mathcal{O}_{t:T}|\mathbf{s}_t)}^{\beta_t(\mathbf{s}_t)}p(\mathbf{s}_t,\mathcal{O}_{1:t-1})}{p(\mathcal{O}_{1:T})} \tag{12.39}$$

$$\propto \beta_t(\mathbf{s}_t)\underbrace{p(\mathbf{s}_t|\mathcal{O}_{1:t-1})}_{\alpha_t(\mathbf{s}_t)}p(\mathcal{O}_{1:t-1}) \propto \beta_t(\mathbf{s}_t)\alpha_t(\mathbf{s}_t) \tag{12.40}$$

12.2.5 Forward / Backward Message Intersection

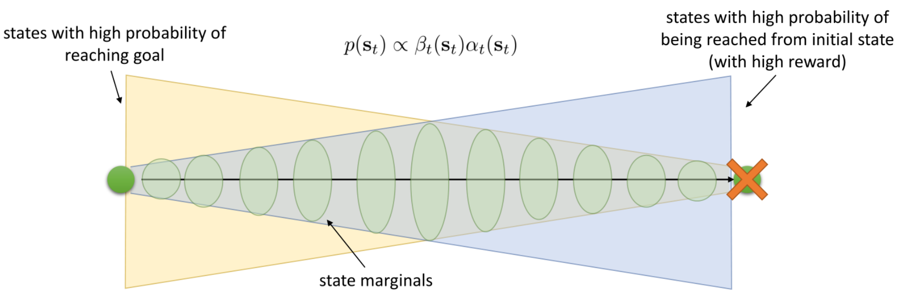

states with high probability of reaching goal
$p(\mathbf{s}_t) \propto \beta_t(\mathbf{s}_t)\alpha_t(\mathbf{s}_t)$
states with high probability of being reached from initial state (with high reward)
state marginals

Figure 12.1: The intersection of backward and forward messages as state marginal (UC Berkeley).

12.3 Control as Variational Inference

12.3.1 Optimism Problem

The problem is due to the fact that given that you obtained high reward, the transition probability changes $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathcal{O}_{1:T}) \neq p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \underbrace{\log \mathbb{E}\left[\exp(V_{t+1}(\mathbf{s}_{t+1}))\right]}_{\text{"optimistic" transition}} \tag{12.41}$$

Unlike in classic MDP, the log of expectation of exp value function is overly optimistic, in the sense that a single large reward will overwhelm other values, while the average reward in expectation is low.

Problem: given that you obtained high reward, what was your action probability, *given that your transition probability did not change?*

Idea: find a distribution $q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) \approx p(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}|\mathcal{O}_{1:T})$ but has dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

12.3.2 Control via Variational Inference

- Let $\mathbf{x} = \mathcal{O}_{1:T}$ and $z = (\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$
  $\Rightarrow$ find $q(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{x})$
- Let $q(\mathbf{z}) = q(\mathbf{s}_{1:T}, \mathbf{a}_{1:T}) = p(\mathbf{s}_1) \prod_t p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) q(\mathbf{a}_t|\mathbf{s}_t)$

The variational lower bound: $\log p(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}[\log p(\mathbf{x,z}) - \log q(\mathbf{z})]$ (AI notes)

$$\begin{aligned}
\Rightarrow \log p(\mathcal{O}_{1:T}) \geq \mathbb{E}_{(\mathbf{s}_{1:T},\mathbf{a}_{1:T})\sim q}\Bigg[&\log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) + \sum_{t=1}^{T} \log p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t) \\
&- \log p(\mathbf{s}_1) - \sum_{t=1}^{T} \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) - \sum_{t=1}^{T} \log q(\mathbf{a}_t|\mathbf{s}_t)\Bigg] \\
= \mathbb{E}_{(\mathbf{s}_{1:T},\mathbf{a}_{1:T})\sim q}&\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) - \log q(\mathbf{a}_t|\mathbf{s}_t)\right] \\
= \sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim q}&[r(\mathbf{s}_t, \mathbf{a}_t) + \mathcal{H}(q(\mathbf{a}_t|\mathbf{s}_t))]
\end{aligned}$$

$\Rightarrow$ This objective is the RL objective, plus the action entropy. The additional action entropy term will give us more sub-optimal actions.

Optimization with dynamic programming:

- Base case: solve for the last time step $q(\mathbf{a}_T|\mathbf{s}_T)$

$$\begin{aligned}
q(\mathbf{a}_T|\mathbf{s}_T) &= \arg\max \mathbb{E}_{\mathbf{s}_T\sim q(\mathbf{s}_T)}\left[\mathbb{E}_{\mathbf{a}_T\sim q(\mathbf{a}_T|\mathbf{s}_T)}[r(\mathbf{s}_T, \mathbf{a}_T)] + \mathcal{H}(q(\mathbf{a}_T|\mathbf{s}_T))\right] \\
&= \arg\max \mathbb{E}_{\mathbf{s}_T\sim q(\mathbf{s}_T)}\left[\mathbb{E}_{\mathbf{a}_T\sim q(\mathbf{a}_T|\mathbf{s}_T)}[r(\mathbf{s}_T, \mathbf{a}_T) - \log q(\mathbf{a}_T|\mathbf{s}_T)]\right]
\end{aligned}$$

Taking the derivative will result in: $q(\mathbf{a}_T|\mathbf{s}_T) \propto \exp(r(\mathbf{s}_T, \mathbf{a}_T))$

$$q(\mathbf{a}_T|\mathbf{s}_T) = \frac{\exp(r(\mathbf{s}_T, \mathbf{a}_T))}{\int \exp(r(\mathbf{s}_T, \mathbf{a}))d\mathbf{a}} = \exp\Big(Q(\mathbf{s}_T, \mathbf{a}_T) - V(\mathbf{s}_T)\Big) \tag{12.42}$$

$$V(\mathbf{s}_T) = \log \int \exp(Q(\mathbf{s}_T, \mathbf{a}_T))d\mathbf{a}_T \tag{12.43}$$

$$\Rightarrow \mathbb{E}_{\mathbf{s}_T \sim q(\mathbf{s}_T)}\Big[\mathbb{E}_{\mathbf{a}_T \sim q(\mathbf{a}_T|\mathbf{s}_T)}[r(\mathbf{s}_T, \mathbf{a}_T) - \log q(\mathbf{a}_T|\mathbf{s}_T)]\Big] = \mathbb{E}_{\mathbf{s}_T \sim q(\mathbf{s}_T)}\Big[\mathbb{E}_{\mathbf{a}_T \sim q(\mathbf{a}_T|\mathbf{s}_T)}[V(\mathbf{s}_T)]\Big] \tag{12.44}$$

- At any time step:

$$Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}[V_{t+1}(\mathbf{s}_{t+1})] \qquad (\textit{regular} \text{ Bellman backup - } \textbf{not} \text{ optimistic})$$

$$\begin{aligned} q(\mathbf{a}_t|\mathbf{s}_t) &= \arg\max \mathbb{E}_{\mathbf{s}_t \sim q(\mathbf{s}_t)}\Big[\mathbb{E}_{\mathbf{a}_t \sim q(\mathbf{a}_t|\mathbf{s}_t)}[r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}[V(\mathbf{s}_{t+1})]] + \mathcal{H}(q(\mathbf{a}_t|\mathbf{s}_t))\Big] \\ &= \arg\max \mathbb{E}_{\mathbf{s}_t \sim q(\mathbf{s}_t)}\Big[\mathbb{E}_{\mathbf{a}_t \sim q(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t)] + \mathcal{H}(q(\mathbf{a}_t|\mathbf{s}_t))\Big] \\ &= \arg\max \mathbb{E}_{\mathbf{s}_t \sim q(\mathbf{s}_t)}\Big[\mathbb{E}_{\mathbf{a}_t \sim q(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t) - \log q(\mathbf{a}_t|\mathbf{s}_t)]\Big] \end{aligned}$$

optimized when: $q(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t))$

$$V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t))d\mathbf{a}_t$$

$$q(\mathbf{a}_t|\mathbf{s}_t) = \exp\Big(Q_t(\mathbf{s}_t, \mathbf{a}_t) - V_t(\mathbf{s}_t)\Big)$$

Now we have a dynamic programming algor. for backward pass, a.k.a. *soft* value iteration algor., which is similar to value iteration algor.:

```
for $t = T - 1 \rightarrow 1$ :
    $Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}[V_{t+1}(\mathbf{s}_{t+1})]$
    $V_t(\mathbf{s}_t) = \log \int \exp(Q_t(\mathbf{s}_t, \mathbf{a}_t))d\mathbf{a}_t$
```

Value iteration algor. (Sec. 6.3)

1. set $Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_a Q(\mathbf{s},\mathbf{a})$

Soft value iteration algor. [Lev18]

1. set $Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma\mathbb{E}[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \text{soft max}_a Q(\mathbf{s},\mathbf{a})$

Variants:

- Discounted SOC: $Q_t(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\mathbb{E}[V_{t+1}(\mathbf{s}_{t+1})]$
- Explicit temperature: $V_t(\mathbf{s}_t) = \alpha \log \int \exp(\frac{1}{\alpha}Q_t(\mathbf{s}_t, \mathbf{a}_t))d\mathbf{a}_t$

12.4 RL Algorithms as Inference

Benefits of soft optimality

- Improve exploration and prevent entropy collapse
- Easier to specialize (finetune) policies for more specific tasks

- Principled approach to break ties
- Better robustness (due to wider coverage of states)
- Can reduce to hard optimality as reward magnitude increases
- Good model for modeling human behavior

12.4.1 Soft Q-Learning

Q-Learning with Soft Optimality simply changes to the soft max:

- Standard Q-learning (Sec. 6.9):

$$\phi \leftarrow \phi + \alpha \nabla_\phi Q_\phi(\mathbf{s},\mathbf{a})\Big(r(\mathbf{s},\mathbf{a}) + \gamma V(\mathbf{s}') - Q_\phi(\mathbf{s},\mathbf{a})\Big)$$

$$V(\mathbf{s}') = \max_{\mathbf{a}'} Q_\phi(\mathbf{s}',\mathbf{a}') \qquad \text{(target value)}$$

- Soft Q-learning:

$$\phi \leftarrow \phi + \alpha \nabla_\phi Q_\phi(\mathbf{s},\mathbf{a})\Big(r(\mathbf{s},\mathbf{a}) + \gamma V(\mathbf{s}') - Q_\phi(\mathbf{s},\mathbf{a})\Big)$$

$$V(\mathbf{s}') = \text{soft max}_{\mathbf{a}'} Q_\phi(\mathbf{s}',\mathbf{a}') = \log \int \exp\Big(Q_\phi(\mathbf{s}',\mathbf{a}')\Big) d\mathbf{a}' \qquad \text{(target value)}$$

$$\pi(\mathbf{a}|\mathbf{s}) = \exp\Big(Q_\phi(\mathbf{s},\mathbf{a}) - V(\mathbf{s})\Big) = \exp\Big(A_\phi(\mathbf{s},\mathbf{a})\Big)$$

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{R}$
2. Sample mini-batch $\{(\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j)\}$ from $\mathcal{R}$ uniformly
3. Compute $y_j = r_j + \gamma \text{soft max}_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)\,(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. Update ϕ': copy ϕ every N steps, or Polyak average $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$

12.4.2 Entropy Regularized Policy Gradient

Policy gradient with soft optimality:
$\pi(\mathbf{a}|\mathbf{s}) = \exp\Big(Q_\phi(\mathbf{s},\mathbf{a}) - V(\mathbf{s})\Big)$ optimizes $J(\theta) = \sum_t \mathbb{E}_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] + \mathbb{E}_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]$

Intuition:

$\pi(\mathbf{a}|\mathbf{s}) \propto \exp(Q_\phi(\mathbf{s},\mathbf{a}))$ when π minimizes $D_{KL}\left(\pi(\mathbf{a}|\mathbf{s}) \Big|\Big| \frac{1}{Z}\exp(Q(\mathbf{s},\mathbf{a}))\right)$

$$D_{KL}\left(\pi(\mathbf{a}|\mathbf{s}) \Big|\Big| \frac{1}{Z}\exp(Q(\mathbf{s},\mathbf{a}))\right) = \mathbb{E}_{\pi(\mathbf{a}|\mathbf{s})}[Q(\mathbf{s},\mathbf{a})] - \mathcal{H}(\pi)$$

12.4.3 Soft Policy Gradient vs Soft Q-Learning

Soft policy gradient derivation:

$$
\begin{aligned}
J(\theta) &= \sum_t \mathbb{E}_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] + \mathbb{E}_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))] && (12.45)\\
&= \sum_t \mathbb{E}_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t) - \log\pi(\mathbf{a}_t|\mathbf{s}_t)] && (12.46)\\
\nabla_\theta J &= \nabla_\theta \left[\sum_t \mathbb{E}_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t) - \log\pi(\mathbf{a}_t|\mathbf{s}_t)]\right] && (12.47)\\
&\approx \frac{1}{N}\sum_i\sum_t \nabla_\theta \log\pi(\mathbf{a}_t|\mathbf{s}_t)\left(r(\mathbf{s}_t,\mathbf{a}_t) + \underbrace{\left(\sum_{t'=t+1}^{T} r(\mathbf{s}_{t'},\mathbf{a}_{t'}) - \log\pi(\mathbf{a}_{t'}|\mathbf{s}_{t'})\right)}_{\approx Q(\mathbf{s}_{t+1},\mathbf{a}_{t+1})} - \log\pi(\mathbf{a}_t|\mathbf{s}_t) - \cancel{1}\right) && (12.48)
\end{aligned}
$$

We can ignore the -1 in the end (baseline).

Recall $\pi(\mathbf{a}_t|\mathbf{s}_t) = \exp\big(Q_t(\mathbf{s}_t,\mathbf{a}_t) - V_t(\mathbf{s}_t)\big)$ (Subsec. 12.2.3):

$$
\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i\sum_t \big(\nabla_\theta Q_t(\mathbf{s}_t,\mathbf{a}_t) - \nabla_\theta V_t(\mathbf{s}_t)\big)\big(r(\mathbf{s}_t,\mathbf{a}_t) + Q(\mathbf{s}_{t+1},\mathbf{a}_{t+1}) - Q_t(\mathbf{s}_t,\mathbf{a}_t) + \cancel{V(\mathbf{s}_t)}\big) \tag{12.49}
$$

Because of the baseline properties, any state dependent baseline can be removed, thus, we can ignore the $V(\mathbf{s}_t)$.

Soft Q-learning (gradient descent, instead of gradient ascent in policy gradient):

$$
\nabla_\theta J(\theta) \approx -\frac{1}{N}\sum_i\sum_t \nabla_\theta Q_t(\mathbf{s}_t,\mathbf{a}_t)\big(r(\mathbf{s}_t,\mathbf{a}_t) + \operatorname{soft}\max_{\mathbf{a}_{t+1}} Q(\mathbf{s}_{t+1},\mathbf{a}_{t+1}) - Q_t(\mathbf{s}_t,\mathbf{a}_t)\big) \tag{12.50}
$$

12.5 Soft Actor-Critic

Algorithm overview [HZA+18]:

1. Q-function update to evaluate current policy: this converges to Q^π
$$
Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \mathbb{E}_{\mathbf{s}'\sim p_\mathbf{s},\mathbf{a}'\sim\pi}[Q(\mathbf{s}',\mathbf{a}') - \log\pi(\mathbf{a}'|\mathbf{s}')] \tag{12.51}
$$
2. Update policy with gradient of information projection
$$
\pi_{new} = \arg\min_{\pi'} D_{KL}\left(\pi'(\cdot|\mathbf{s})\middle\|\frac{1}{Z}\exp Q^{\pi_{old}}(\mathbf{s},\cdot)\right) \tag{12.52}
$$
In practice, only take one gradient step on this objective
3. Interact with the world, collect more data

12.6 References

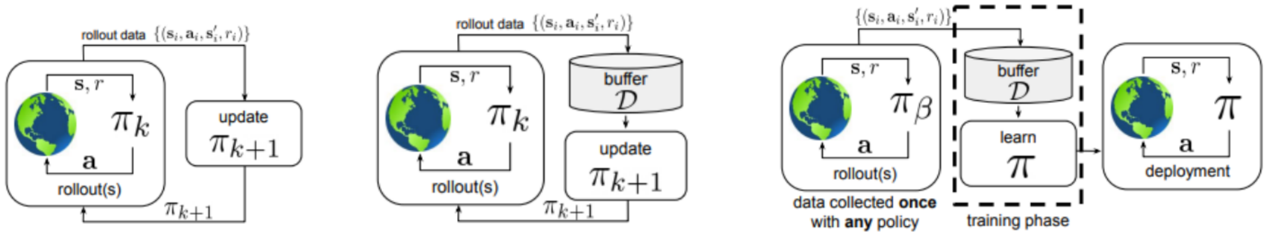
- Todorov (2006) [Tod06]. *"Linearly-solvable Markov decision problems"*.
- Todorov (2008) [Tod08]. *"General duality between optimal control and estimation"*.
- Kappen et al. (2012) [KGO12]. *"Optimal control as a graphical model inference problem"*.
- Ziebart et al. (2010) [ZBD10]. *"Modeling interaction via the principle of maximum causal entropy"*.
- Rawlik et al. (2012) [RTV12]. *"On stochastic optimal control and reinforcement learning by approximate inference"*.
- Haarnoja et al. (2017) [HTA+17]. *"Reinforcement learning with deep energy-based policies"*.
- Nachum et al. (2017) [NNX+17]. *"Bridging the gap between value and policy based reinforcement learning"*.
- Schulman et al. (2017) [SCA17]. *"Equivalence between policy gradients and soft q-learning"*.
- Haarnoja et al. (2018) [HZA+18]. *"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"*.
- Levine (2018) [Lev18]. *"Reinforcement learning and control as probabilistic inference: Tutorial and review"*.

13 Offline Reinforcement Learning

One major challenge in RL is the ability to generalize to different tasks, unlike with supervised learning. E.g., supervised learning in Computer Vision (CV), a model can be trained on a large dataset (e.g., ImageNet), then retrained on a smaller dataset to adapt with a specific task. This is possible because it was proven that early layers manage to learn different feature representation What RL differs from more conventional supervised learning are large diverse dataset, training procedure, how and when the data is collected, etc. Offline RL is one of proposed solution for RL generalization problem. This chapter examines how we can formulate RL similar to supervised learning.

13.1 Formulation

Offline RL is also known as batch RL, fully off-policy RL.



(a) On-policy RL

(b) Off-policy RL

(c) Offline RL

Figure 13.1: Different settings for RL.

Formal problem definition: Offline RL

$$\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, \mathbf{r}_i)\} \qquad \text{– dataset of transitions} \tag{13.1}$$

$$\mathbf{s} \sim d^{\pi_\beta}(\mathbf{s}) \tag{13.2}$$

$$\mathbf{a} \sim \pi_\beta(\mathbf{a}|\mathbf{s}) \qquad \text{– unknown policy} \tag{13.3}$$

$$\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a}) \tag{13.4}$$

$$r \leftarrow r(\mathbf{s},\mathbf{a}) \tag{13.5}$$

$$\max_\pi \sum_{t=0}^{T} \mathbb{E}_{\mathbf{s}_t \sim d^{\pi_\beta}(\mathbf{s}), \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)] \qquad \text{– RL objective} \tag{13.6}$$

- Off-policy evaluation (OPE): given $\mathcal{D}$, estimate $J(\pi_\beta) = \mathbb{E}_{\pi_\beta}\left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right]$

- Offline RL: given $\mathcal{D}$, learn the ***best possible*** policy π_θ, which might not be the best possible policy for the MDP

We hope this is possible because: we expect the model can

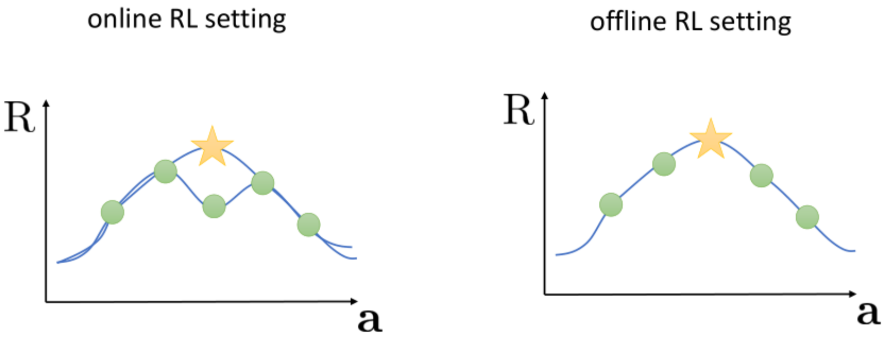
- Find the *"good stuff"* in a dataset full of good and bad behaviors
- Generalization: good behavior in one place may suggest good behavior in another place
- *"Stitching"*: parts of good behaviors can be recombined [SYY+20]

13.2 Challenges

- Little amount of online tuning has significantly more impact than offline training [KIP+18]
- Distribution shift

$$\begin{aligned}
&\theta \leftarrow \arg\max_\theta \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}), y\sim p(y|\mathbf{x})}[(f_\theta(\mathbf{x}) - y)^2] && \text{– normal regression problem}\\
&\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x})}[(f_\theta(\mathbf{x}') - y)^2] && \text{– is low in expectation}\\
&\mathbb{E}_{\mathbf{x}'\sim \bar{p}(\mathbf{x})}[(f_\theta(\mathbf{x}') - y)^2] && \text{– is not, for general } \bar{p}(\mathbf{x}) \neq p(\mathbf{x})\\
&\mathbb{E}_{\mathbf{x}'\sim p(\mathbf{x})}[(f_\theta(\mathbf{x}') - y)^2] && \text{– is } \underline{\boldsymbol{NOT}} \text{ low if } \mathbf{x}' \leftarrow \arg\max_{\mathbf{x}} f_\theta(\mathbf{x})
\end{aligned}$$

  This is exactly what happens for value-based RL approaches, in which we decide the action based on maximizing the value function ⇒ Over-estimation of value function [KFS+19]
- Counterfactual queries: must account for out-of-distribution actions, ideally in a safe way, while still making use of generalization to come up with behaviors that are better than the best thing seen in the data [LKT+20]
- Issues with generalization are not corrected:



13.3 Classic Offline RL

If you want to use offline RL today, you probably would not use approaches in this section.

13.3.1 Batch RL via Importance Sampling

Batch RL is just an old name for offline RL (around 2000s). Using importance sampling, we can use samples from one policy to learn another one (Sec. 4.4)

$$
\begin{aligned}
\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{\mathbf{s}_t \sim d^{\pi}(\mathbf{s}), \mathbf{a}_t \sim \pi(\mathbf{a}|\mathbf{s})} [\gamma^t r(\mathbf{s}_t, \mathbf{a}_t)] & \qquad \text{– RL objective} \\
\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi(\mathbf{a}_t|\mathbf{s}_t) \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right] & \qquad \text{– policy gradient} \\
\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) & \qquad \text{requires sampling from } \pi_\theta \\
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \underbrace{\frac{\pi_\theta(\tau_i)}{\pi_\beta(\tau_i)}}_{\text{importance weight}} \sum_{t=0}^{T} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) & \qquad \text{use samples from } \pi_\beta \\
\frac{\pi_\theta(\tau)}{\pi_\beta(\tau)} = \frac{\cancel{p(\mathbf{s}_1)} \prod_t \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\cancel{p(\mathbf{s}_1)} \prod_t \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \pi_\beta(\mathbf{a}_t|\mathbf{s}_t)} & \qquad \text{exponential in } T \\
\hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) = \mathbb{E}_{\pi_\theta} \left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \right] \approx \sum_{t'=t}^{T} \gamma^{t'-t} r_{t',i} & \qquad \text{single sample estimate} \\
\Rightarrow \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \underbrace{\cancel{\left(\prod_{t'=0}^{t-1} \frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})} \right)}}_{\substack{\text{accounts for difference in} \\ \text{prob. of landing in } \mathbf{s}_{t,i}, \\ \text{we have } \mathbf{s}_t \sim d^{\pi_\beta}(\mathbf{s}_t), \\ \text{but want } \mathbf{s}_t \sim d^{\pi_\theta}(\mathbf{s}_t)}} \nabla_\theta \gamma^t \log \pi_\theta(\mathbf{a}_{t,i}|\mathbf{s}_{t,i}) \underbrace{\left(\prod_{t'=t}^{T} \frac{\pi_\theta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})}{\pi_\beta(\mathbf{a}_{t',i}|\mathbf{s}_{t',i})} \right)}_{\substack{\text{accounts for having} \\ \text{incorrect } \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i})}} \hat{Q}(\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) &
\end{aligned}
$$

[**TODO: Ignore for now, too many complicated equations**]

13.3.2 Batch RL via Linear Fitted Value Functions

Let's go back to the model-based RL setting with tabular linear fitted value functions. This method generally doesn't concern with distributional shift, maybe it was not such a big problem with linear models.

High level idea procedure:

1. Estimate the reward
2. Estimate the transitions
3. Recover the value function
4. Improve the policy

$$\begin{aligned}
&\Phi && \text{– feature matrix, size}|S| \times K && (13.7)\\
&\Phi \mathbf{w}_r \approx r && \text{– reward model, } \mathbf{w}_r \in \mathbb{R}^{K\times 1}, r \in \mathbb{R}^{|S|\times 1} && (13.8)\\
\Rightarrow &\mathbf{w}_r = (\Phi^T\Phi)^{-1}\Phi^T\vec{\mathbf{r}} && \text{– least square solution (assume knowing } \vec{\mathbf{r}}\text{)} && (13.9)\\
&\Phi\mathbf{P}_\Phi \approx \mathbf{P}^\pi\Phi && \text{– transition model} && (13.10)\\
&\mathbf{P}^\pi \in \mathbb{R}^{|S|\times|S|} && \text{– real transition matrix (on states)} && (13.11)\\
&\mathbf{P}_\Phi \in \mathbb{R}^{K\times K} && \text{– estimated feature-space transition matrix} && (13.12)\\
\Rightarrow &\mathbf{P}_\Phi = \left(\Phi^T\Phi\right)^{-1}\Phi^T\mathbf{P}^\pi\Phi && \text{– least squares} && (13.13)\\
\Rightarrow &V^\pi \approx V^\pi_\Phi = \Phi\mathbf{w}_V && \text{– value functions} && (13.14)
\end{aligned}$$

We can find V^π in terms of $\mathbf{P}^\pi$ and $\mathbf{r}$:

$$V^\pi = \mathbf{r} + \gamma\mathbf{P}^\pi V^\pi \tag{13.15}$$

$$\Rightarrow V^\pi = (\mathbf{I} - \gamma\mathbf{P}^\pi)^{-1}\mathbf{r} \tag{13.16}$$

Same derivation in feature space:

$$\begin{aligned}
&\mathbf{w}_V = (\mathbf{I} - \gamma\mathbf{P}_\Phi)^{-1}\mathbf{w}_r && && (13.17)\\
&\quad\;\; = (\mathbf{I} - \gamma(\Phi^T\Phi)^{-1}\Phi^T\mathbf{P}^\pi\Phi)^{-1}(\Phi^T\Phi)^{-1}\Phi^T\vec{\mathbf{r}} && \text{– substitute in } \mathbf{w}_r \text{ and } \mathbf{P}_\Phi && (13.18)\\
\Rightarrow\; &\mathbf{w}_V = (\Phi^T\Phi - \gamma\Phi^T\mathbf{P}^\pi\Phi)^{-1}\Phi^T\vec{\mathbf{r}} && \text{– LSTD} && (13.19)
\end{aligned}$$

The above equations are of Least-squares temporal difference (LSTD). However it requires us to know transition matrix $\mathbf{P}^\pi$ and reward vector $\vec{\mathbf{r}}$. To get a model-free approach, we will use samples. We replace $\mathbf{P}^\pi\Phi$ with the feature matrix of the next time step Φ':

$$\Phi'_i = \phi(\mathbf{s}'_i) \tag{13.20}$$

LSTD Algorithm: not going to work for offline RL, because the equation requires samples from π for $\mathbf{P}^\pi$

1. $\pi'(\mathbf{s}) \leftarrow \text{Greedy}(\Phi\mathbf{w}_V)$
2. estimate $V^{\pi'}$

Least-squares policy iteration (LSPI) Algorithm: LSTDQ - LSTD for Q-functions.

1. compute $\mathbf{w}_Q$ for π_k
2. $\pi_{k+1}(\mathbf{s}) = \arg\max_{\mathbf{a}}\phi(\mathbf{s},\mathbf{a})\mathbf{w}_Q$
3. set $\Phi'_i = \phi(\mathbf{s}'_i, \pi_{k+1}(\mathbf{s}'_i))$

Now $\Phi \in \mathbb{R}^{|S||A|\times K'}$, $\mathbf{w}_Q = (\Phi^T\Phi - \gamma\Phi^T\Phi')^{-1}\Phi^T\vec{\mathbf{r}}$, $\vec{\mathbf{r}}_i = r(\mathbf{s}_i, \mathbf{a}_i)$ [LP03]

13.4 Conservative Q-Learning

Conservative Q-learning (CQL)

13.5 Model-Based Offline RL

Model-Based Offline Policy Optimization (MOPO)

13.6 Summary

13.7 Applications

13.8 Open Questions

13.9 References

Readings for importance sampling

- Classic work on importance sampled policy gradients and return estimation:
  - Precup (2000) [Pre00]. *"Eligibility traces for off-policy policy evaluation"*.
  - Peshkin et al. (2002) [PS02]. *"Learning from scarce experience"*.
- Doubly robust estimators and other improved importance-sampling estimators:
  - Jiang et al. (2016) [JL16]. *"Doubly robust off-policy value evaluation for reinforcement learning"*.
  - Thomas et al. (2016) [TB16]. *"Data-efficient off-policy policy evaluation for reinforcement learning"*.
- Analysis and theory:
  - Thomas et al. (2015) [TTG15]. *"High-confidence off-policy evaluation"*.
- Marginalized importance sampling:
  - Hallak et al. (2017) [HM17]. *"Consistent on-line off-policy evaluation"*.
  - Liu et al. (2019) [LSA+19]. *"Off-policy policy gradient with state distribution correction"*.

Modern offline RL:

- Levine et al. (2020) [LKT+20]. *"Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems"*.

14 Inverse Reinforcement Learning

Prior to this, we have been manually design the reward function, which defines the task. There are cases, the reward function is unavailable or difficult to specify. The idea behind inverse RL is to use human / expert's experience to *learn* the reward function, then use it for RL as a goal to optimize.

There is a difference between standard imitation learning and human imitation learning:

- Standard imitation learning:
  - copy the *actions* performed by the expert
  - no reasoning about outcomes of actions
- Human imitation learning:
  - copy the *intent* of the expert
  - might take very different actions!

14.1 Definition

Inverse RL vs. "forward" RL:

"Forward" RL	Inverse RL
given: states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$ (sometimes) transitions $p(\mathbf{s}'\|\mathbf{s},\mathbf{a})$ reward function $r(\mathbf{s},\mathbf{a})$	given: states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$ (sometimes) transitions $p(\mathbf{s}'\|\mathbf{s},\mathbf{a})$ samples $\{\tau_i\}$ sampled from $\pi^*(\tau)$
learn $\pi^*(\mathbf{a}\|\mathbf{s})$	learn $r_\psi(\mathbf{s},\mathbf{a})$ . . . and then use it to learn $\pi^*(\mathbf{a}\|\mathbf{s})$

Choices for ψ:

- linear reward function: $r_\psi(\mathbf{s},\mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s},\mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s},\mathbf{a})$
- neural net reward function: $r_\psi(\mathbf{s},\mathbf{a})$

14.2 Feature matching IRL

- Linear reward function: $r_\psi(\mathbf{s},\mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s},\mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s},\mathbf{a})$
- Let π^{r_ψ} be the optimal policy for r_ψ, π^* be the unknown optimal policy
- Pick ψ such that $\mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}(\mathbf{s},\mathbf{a})] = \mathbb{E}_{\pi^*}[\mathbf{f}(\mathbf{s},\mathbf{a})]$

Problem: still ambiguous

Maximum Margin Principle:

$$\max_{\psi,m} m \qquad \text{such that } \psi^T \mathbb{E}_{\pi^*}[\mathbf{f}(\mathbf{s},\mathbf{a})] \geq \max_{\pi \in \Pi} \psi^T \mathbb{E}_{\pi}[\mathbf{f}(\mathbf{s},\mathbf{a})] + m$$

$$\Leftrightarrow \min_{\psi} \frac{1}{2}||\psi||^2 \quad \text{such that } \psi^T \mathbb{E}_{\pi^*}[\mathbf{f}(\mathbf{s},\mathbf{a})] \geq \max_{\pi \in \Pi} \psi^T \mathbb{E}_{\pi}[\mathbf{f}(\mathbf{s},\mathbf{a})] + D(\pi, \pi^*) \quad \text{(SVM trick)}$$

Issues: [AN04; RBZ06]

- Maximizing the margin is a bit arbitrary
- No clear model of expert suboptimality (can add slack variables. . .)
- Messy constrained optimization problem – not great for deep learning!

14.3 MaxEnt IRL Algorithm

Using the graphical model (Sec. 12.1), we approach the problem as maximum likelihood learning: learning the params. ψ to maximize the prob. of expert's trajectories given that they are (sub)optimal.

$$p(\mathcal{O}_t|\mathbf{s}_t, \mathbf{a}_t, \psi) = \exp(r_\psi(\mathbf{s}_t, \mathbf{a}_t)) \tag{14.1}$$

$$p(\tau|\mathcal{O}_{1:T}) = \frac{p(\tau, \mathcal{O}_{1:T})}{p(\mathcal{O}_{1:T})} \propto \cancel{p(\tau)} \exp\left(\sum_t r_\psi(\mathbf{s}_t, \mathbf{a}_t)\right) \qquad (p(\tau) \text{ is independent of } \psi) \tag{14.2}$$

The Inverse Reinforcement Learning (IRL)'s goal:

$$\psi = \arg\max_\psi \frac{1}{N}\sum_{i=1}^{N} \log p(\tau_i|\mathcal{O}_{1:T}, \psi) = \arg\max_\psi \frac{1}{N}\sum_{i=1}^{N} r_\psi(\tau_i) - \log Z = \arg\max_\psi \mathcal{L}(\psi) \tag{14.3}$$

The IRL partition function:

$$Z = \int p(\tau)\exp(r_\psi(\tau))d\tau \qquad (Z - \text{partition function}) \tag{14.4}$$

$$\nabla_\psi \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \underbrace{\frac{1}{Z}\int p(\tau)\exp(r_\psi(\tau))}_{p(\tau|\mathcal{O}_{1:T}, \psi)} \nabla_\psi r_\psi(\tau) d\tau \tag{14.5}$$

$$= \underset{\text{expert samples}}{\mathbb{E}_{\tau\sim\pi^*(\tau)}[\nabla_\psi r_\psi(\tau_i)]} - \underset{\text{soft optimal policy}}{\mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)]} \tag{14.6}$$

The gradient is the difference between the expectation of the derivative of trajectory's reward between the expert's and the one from rolling out the current policy.

Estimation of the 2nd term:

$$\mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)] = \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}\left[\nabla_\psi \sum_{t=1}^{T} r_\psi(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{14.7}$$

$$= \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim p(\mathbf{s}_t,\mathbf{a}_t|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\mathbf{s}_t,\mathbf{a}_t)] \tag{14.8}$$

$$p(\mathbf{s}_t,\mathbf{a}_t|\mathcal{O}_{1:T},\psi) = \underbrace{p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{1:T},\psi)}_{\text{the policy}} \quad \underbrace{p(\mathbf{s}_t|\mathcal{O}_{1:T},\psi)}_{\text{the state marginal prob.}} \tag{14.9}$$

$$p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{1:T},\psi) = \frac{\beta(\mathbf{s}_t,\mathbf{a}_t)}{\beta(\mathbf{s}_t)} \qquad p(\mathbf{s}_t|\mathcal{O}_{1:T},\psi) \propto \alpha(\mathbf{s}_t)\beta(\mathbf{s}_t) \qquad \text{(Sec. 12.2)}$$

$$\Rightarrow p(\mathbf{s}_t,\mathbf{a}_t|\mathcal{O}_{1:T},\psi) \propto \beta(\mathbf{s}_t,\mathbf{a}_t)\alpha(\mathbf{s}_t) \qquad \text{let } \mu_t(\mathbf{s}_t,\mathbf{a}_t) \propto \beta(\mathbf{s}_t,\mathbf{a}_t)\alpha(\mathbf{s}_t)$$

$$\Rightarrow \mathbb{E}_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)] = \sum_{t=1}^{T} \int\int \mu_t(\mathbf{s}_t,\mathbf{a}_t)\nabla_\psi r_\psi(\mathbf{s}_t,\mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t = \sum_{t=1}^{T} \vec{\mu}_t \nabla_\psi \vec{r}_\psi \tag{14.10}$$

Maximum Entropy (MaxEnt) IRL Algorithm:

1. Given ψ, compute backward message $\beta(\mathbf{s}_t,\mathbf{a}_t)$
2. Given ψ, compute forward message $\alpha(\mathbf{s}_t)$
3. Compute $\mu(\mathbf{s}_t,\mathbf{a}_t) \propto \beta(\mathbf{s}_t,\mathbf{a}_t)\alpha(\mathbf{s}_t)$
4. Evaluate $\nabla_\psi \mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\psi r_\psi(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) - \sum_{t=1}^{T}\int\int \mu(\mathbf{s}_t,\mathbf{a}_t)\nabla_\psi r_\psi(\mathbf{s}_t,\mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t$
5. $\psi \leftarrow \psi + \eta\nabla_\psi\mathcal{L}$

The name MaxEnt: when $r_\psi(\mathbf{s}_t,\mathbf{a}_t) = \psi^T\mathbf{f}(\mathbf{s}_t,\mathbf{a}_t)$, the procedure optimizes $\max_\psi \mathcal{H}(\pi^{r_\psi})$ such that $\mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}] = \mathbb{E}_{\pi^*}[\mathbf{f}]$. It is as random as possible while matching features. [ZMB+08]

14.4 Guided Cost Learning Algorithm

Problems: MaxEnt IRL so far requires:

- Solving for (soft) optimal policy in the inner loop
- Enumerating all state-action tuples for visitation frequency and gradient

Practical problem settings:

- Large and continuous state and action spaces
- States obtained via sampling only
- Unknown dynamics

Idea: learn $p(\mathbf{a}_t|\mathbf{s}_t, \mathcal{O}_{1:T}, \psi)$ using any max entropy RL algor., then run that policy to sample trajectories $\{\tau_j\}$

$$\nabla_\psi \mathcal{L} \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \frac{1}{M}\sum_{j=1}^{M} \nabla_\psi r_\psi(\tau_j) \tag{14.11}$$

If instead of expensively learning the policy, we just improve the policy (a little), we could use ***importance sampling*** to handle the biased estimator. [FLA16]

$$\nabla_\psi \mathcal{L} \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\psi r_\psi(\tau_i) - \frac{1}{\sum_j w_j}\sum_{j=1}^{M} w_j \nabla_\psi r_\psi(\tau_j) \tag{14.12}$$

$$w_j = \frac{p(\tau)\exp(r_\psi(\tau_j))}{\pi(\tau_j)} \tag{14.13}$$

$$= \frac{\cancel{p(\mathbf{s}_1)} \prod_t \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \exp(r_\psi(\mathbf{s}_t, \mathbf{a}_t))}{\cancel{p(\mathbf{s}_1)} \prod_t \cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \pi(\mathbf{a}_t|\mathbf{s}_t)} = \frac{\exp\left(\sum_t r_\psi(\mathbf{s}_t, \mathbf{a}_t)\right)}{\prod_t \pi(\mathbf{a}_t|\mathbf{s}_t)} \tag{14.14}$$

14.5 IRL and GANs

Similarity between IRL and Generative Adversarial Network (GAN):

IRL	GAN [GPAM+14]
Player 1: policy π_θ Player 2: reward function r_ψ	Player 1: the generator $p_\theta(\mathbf{x}|\mathbf{z})$ Player 2: the discriminator $p_\psi(\mathbf{z}|\mathbf{x})$
- The policy improves itself to make it ***harder*** to distinguish its samples and the expert demonstrations - The reward model improves itself to make it ***easier*** to distinguish the expert demonstrations and the policy's samples (Eq. 14.6)	- The generator improves itself to make it ***harder*** to distinguish its samples and the real images - The discriminator improves itself to make it ***easier*** to distinguish the real images and the generator's samples

14.5.1 IRL as a GAN

The optimal discriminator for GAN is: $D^*(\mathbf{x}) = \dfrac{p^*(\mathbf{x})}{p_\theta(\mathbf{x}) + p^*(\mathbf{x})}$. For IRL, optimal policy approaches $\pi_\theta(\tau) \propto p(\tau)\exp(r_\psi(\tau))$. Thus choosing parameterization for discriminator in the following way can remove importance weights (they are subsumed into Z) [FCA+16]

$$
\begin{aligned}
D_\psi(\tau) &= \frac{p(\tau)\frac{1}{Z}\exp(r(\tau))}{p_\theta(\tau) + p(\tau)\frac{1}{Z}\exp(r(\tau))} = \frac{\cancel{p(\tau)}\frac{1}{Z}\exp(r(\tau))}{\cancel{p(\tau)}\prod_t \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{p(\tau)}\frac{1}{Z}\exp(r(\tau))} && (14.15)\\
&= \frac{\frac{1}{Z}\exp(r(\tau))}{\prod_t \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \frac{1}{Z}\exp(r(\tau))} && (14.16)\\
\psi &\leftarrow \arg\max_\psi \mathbb{E}_{\tau\sim p^*}[\log D_\psi(\tau)] + \mathbb{E}_{\tau\sim\pi_\theta}[\log(1 - D_\psi(\tau))] && (14.17)
\end{aligned}
$$

14.5.2 Generative adversarial imitation learning

We could also use regular discriminator ($D_\psi(\tau)$ as standard binary classifier) [HE16].

+often simpler to set up optimization, fewer moving parts

−discriminator knows nothing at convergence

−generally cannot re-optimize the "reward"

14.5.3 Generalization via IRL

We need to decouple the **goal** from the **dynamics!** [FLL17]

14.6 References

Classic Papers:

- Abbeel et al. (2004) [AN04]. *"Apprenticeship learning via inverse reinforcement learning"*.
- Ziebart et al. (2008) [ZMB+08]. *"Maximum entropy inverse reinforcement learning."*.

Modern Papers:

- Finn et al. (2016) [FLA16]. *"Guided cost learning: Deep inverse optimal control via policy optimization"*.
- Wulfmeier et al. (2015) [WOP15]. *"Maximum entropy deep inverse reinforcement learning"*.
- Ho et al. (2016) [HE16]. *"Generative adversarial imitation learning"*.
- Fu et al. (2017) [FLL17]. *"Learning robust rewards with adversarial inverse reinforcement learning"*.

15 Transfer and Multi-Task Learning

If we've solved prior tasks, we might acquire useful knowledge for solving a new task. In RL, knowledge is stored in:

- Q-function: tells us which actions or states are good
- Policy: tells us which actions are potentially useful. Some actions are never useful!
- Models: what are the laws of physics that govern the world?
- Features/hidden states: provide us with a good representation

15.1 Definitions

- *Transfer learning* is using experience from ***one set of tasks*** for faster learning and better performance on a ***new task*** in RL.
- In RL, task = MDP
- *Shot*: number of attempts in the target domain
  - 0-shot: just run the policy trained in the source domain
  - 1-shot: try the task once
  - few shot: try the task a few times

No single solution!

- Forward transfer: train on one task, transfer to a new task
  - Transferring visual representations & domain adaptation
  - Domain adaptation in reinforcement learning
  - Randomization
- Multi-task transfer: train on many tasks, transfer to a new task
  - Sharing representations and layers across tasks in multi-task learning
  - Contextual policies
  - Optimization challenges for multi-task learning
  - Algorithms
- Transferring models and value functions
  - Model-based RL as a mechanism for transfer
  - Successor features & representations

15.2 Forward Transfer

Forward transfer: train on one task, transfer to a new task.

- Transferring visual representations & domain adaptation
- Domain adaptation in reinforcement learning
- Randomization

Common challenges:

- Domain shift: representations learned in the source domain might not work well in the target domain
- Difference in the MDP: some things that are possible to do in the source domain are not possible to do in the target domain
- Finetuning issues: if pretraining & finetuning, the finetuning process may still need to explore, but optimal policy during finetuning may be deterministic!

15.2.1 Domain Adaptation

Invariance assumption: everything that is **different** between domains is **irrelevant**.

Idea: force a representation layer (a layer) to be domain invariant. $p(x)$ is different, exists some $z = f(x)$ such that $p(y|z) = p(y|x)$, but $p(z)$ is the same in both source and target domains. [TDH+20]

Invariance is not enough when the dynamics don't match. [EAC+20]

$$\begin{aligned}
\tilde{r}(s,a) =& r(s,a) + \Delta r(s,a) \\
\Delta r(s_t, a_t, s_{t+1}) =& \log p_{target}(s_{t+1}|s_t, a_t) - \log p_{source}(s_{t+1}|s_t, a_t) \\
\Delta r(s_t, a_t, s_{t+1}) =& \log p(target|s_t, a_t, s_{t+1}) - \log p(target|s_t, a_t) \\
& - \log p(source|s_t, a_t, s_{t+1}) - \log p(source|s_t, a_t)
\end{aligned}$$

Problems:

- The above approach considers limits in the target domain that doesn't present in source domain.
- But it doesn't consider things that can happen in target domain but limited in source domain.

15.2.2 Finetuning

Challenges:

- RL tasks are generally much less diverse
  - Features are less general
  - Policies & value functions become overly specialized
- Optimal policies in fully observed MDPs are deterministic
  - Loss of exploration at convergence
  - Low-entropy policies adapt very slowly to new settings

Finetuning with maximum-entropy policies: act ***as randomly as possible*** while collecting high rewards. [HTA+17]

15.2.3 References

Domain adaptation:

- Tzeng et al. (2014) [THZ+14]. *"Deep domain confusion: Maximizing for domain invariance"*.
- Ganin et al. (2016) [GUA+16]. *"Domain-adversarial training of neural networks"*.
- Tzeng et al. (2020) [TDH+20]. *"Adapting deep visuomotor representations with weak pairwise constraints"*.
- Eysenbach et al. (2020) [EAC+20]. *"Off-dynamics reinforcement learning: Training for transfer with domain classifiers"*.

Finetuning:

- Haarnoja et al. (2017) [HTA+17]. *"Reinforcement learning with deep energy-based policies"*.
- Andreas et al. (2017) [AKL17]. *"Modular multitask reinforcement learning with policy sketches"*.
- Florensa et al. (2017) [FDA17]. *"Stochastic neural networks for hierarchical reinforcement learning"*.
- Kumar et al. (2020) [KKL+20]. *"One solution is not all you need: Few-shot extrapolation via structured maxent rl"*.

15.3 Forward Transfer with Randomization

Idea: If we can design the source domain, then vary it to prepare for the difficult in target domain.

- Randomizing physical params. [RGR+16]
- Explicit system identification [YTL+17]
- Randomization for real-world control [SL16]

15.3.1 References

- Rajeswaran et al. (2016) [RGR+16]. *"EpOpt: Learning robust neural network policies using model ensembles"*.
- Yu et al. (2017) [YTL+17]. *"Preparing for the unknown: Learning a universal policy with online system identification"*.
- Sadeghi et al. (2016) [SL16]. *"CAD2RL: Real single-image flight without a single real image"*.
- Tobin et al. (2017) [TFR+17]. *"Domain randomization for transferring deep neural networks from simulation to the real world"*.
- James et al. (2017) [JDJ17]. *"Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task"*.
- Bousmalis et al. (2018) [BIW+18]. *"Using simulation and domain adaptation to improve efficiency of deep robotic grasping"*.
- Rao et al. (2020) [RHI+20]. *"Rl-CycleGAN: Reinforcement learning aware simulation-to-real"*.

15.4 Multi-Task Transfer

Definition: train on many tasks, transfer to a new task.

- Sharing representations and layers across tasks in multi-task learning
- Contextual policies
- Optimization challenges for multi-task learning
- Algorithms

Multi-task RL corresponds to a single-task RL in a ***joint MDP***.

Challenges:

- ***Gradient interference:*** becoming better on one task can make you worse on another *Negative transfer* implies the transfer actually hurting the task's performance.
- ***Winner-take-all problem:*** imagine one task starts getting good algorithm is likely to prioritize that task (to increase average expected reward) at the expensive of others.

The two approaches is not really multi-task transfer. Not really fasten the learning speed of other tasks

15.4.1 Actor-Mimic

- Train multiple single-task policy, then use supervised learning to combine them into one single policy. [RCG+15]
- Analogous to guided policy search (Sec. 10.4), but for transfer learning. [LK13]

15.4.2 Policy Distillation

Divide and Conquer does has some transfer between central policy and local policies, but not ***much***. [PBS15]

15.4.3 Contextual Policies

Can do (discern) multiple things in the same environment.

- Standard policy: $\pi_\theta(\mathbf{a}|\mathbf{s})$
- Contextual policy: $\pi_\theta(\mathbf{a}|\mathbf{s}, \omega)$
- Augmented state space: $\tilde{\mathbf{s}} = \begin{bmatrix} \mathbf{s} \\ \omega \end{bmatrix} \qquad \tilde{\mathcal{S}} = \mathcal{S} \times \Omega$

15.5 Transferring Models and Value Functions

Assumption:

- the **dynamics** is the same in both domains
- but the **reward function** is different

E.g.:

- Autonomous car learns how to drive to a few destinations, and then has to navigate to a new one.
- A kitchen robot learns to cook many different recipes, and

15.5.1 Transferring Models

Due to distribution shift, zero-shot might not always work. Although the dynamics is the same, they could be in different contexts: things present in source domain might not be present in target domain, and vice versa.

Unless the samples in the source domain is broad enough that they covers target domain's.

15.5.2 Transferring Value Functions

Value functions couple dynamics, rewards, and policies, but in linearity.

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \underbrace{r(\mathbf{s},\mathbf{a})}_{\text{rewards}} + \gamma \mathbb{E}_{\mathbf{s}' \sim \underbrace{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}_{\text{dynamics}}, \mathbf{a}' \sim \underbrace{\pi(\mathbf{a}'|\mathbf{s}')}_{\text{policies}}}[Q^{\pi}(\mathbf{s}',\mathbf{a}')] \tag{15.1}$$

Let $\mathbf{P}^{\pi}\mathbf{v}$ denote a vector $\mathbf{w}$ of length $|S||A|$:

$$\mathbf{w}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a}), \mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')}[\mathbf{v}(\mathbf{s}',\mathbf{a}')] \tag{15.2}$$

$$\mathbf{w}_{|S||A| \times 1} = \mathbf{P}^{\pi}_{|S||A| \times |S||A|} \mathbf{v}_{|S||A| \times 1} \tag{15.3}$$

$$Q^{\pi} = r + \gamma \mathbf{P}^{\pi} Q^{\pi} \tag{15.4}$$

$$Q^{\pi} = (\mathbf{I} - \gamma \mathbf{P}^{\pi})^{-1} r \tag{15.5}$$

Let:

- ϕ be an $|S||A| \times N$ *feature* matrix
- ψ be an $|S||A| \times N$ matrix such that $\psi = (\mathbf{I} - \mathbf{P}^{\pi})^{-1}\phi$
- if $r = \phi w$, then $Q^{\pi} = \psi w$, with w is a $1 \times N$ row vector
  In other words, if r is a linear combination of ϕ, then Q^{π} is a linear combination of ψ
- ψ_i is a "successor feature" for ϕ_i
- for any *new* reward function, if we can fit $r \approx \phi w$, we get $Q^{\pi} \approx \psi w$
  NOTE: this holds for Q^{π}, not Q^*

  $$Q^*(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[\max_{\mathbf{a}'} Q^{\pi}(\mathbf{s}',\mathbf{a}')\right] \qquad \text{(no longer linear)}$$

[**TODO:**] how to find ψ, ϕ given $\mathbf{P}^{\pi}$

Simplest use: evaluation

1. get small amount of data $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)$ in new MDP
2. fit w such that $\phi(\mathbf{s}_i, \mathbf{a}_i) w \approx r_i$ (linear regression)
3. initialize $Q^{\pi}(\mathbf{s},\mathbf{a}) = \psi(\mathbf{s},\mathbf{a}) w$
4. finetune π and Q^{π} with any RL method

More sophisticated use: train multiple ψ^{π_i} functions for different π_i

- choose initial policy $\pi(\mathbf{s},\mathbf{a}) = \arg\max_{\mathbf{a}} \psi^{\pi_i}(\mathbf{s},\mathbf{a}) w$
- this provides a better initial policy in general

Additional notes: successor representations with $\phi = \mathbf{I}$ . . . [Day93]

16 Meta-Learning

16.1 Scaffolding

Scaffolding is a term that commonly used in the construction field, but also has a relation to infant's learning process. Scaffolding is a temporary structure that supports the construction, maintenance of a large building (wiki). In the context of learning, it implies the structured assistance, or guidance, for better accomplishment of new skills and tasks. As the learner's *autonomy* (learning abilities) increases, the support decrease accordingly . [Par; ZBC20]

Figure 16.1: Scaffolding in construction.

Extending the idea behind scaffolding to Machine Learning (ML) and RL: [ZBC20]

- Visual imitation learning [FYZ+17; SPG19]
- Learning from demonstration [**TODO:**]
- Collective learning: learn from / guided by other agents
- Active learning

[**TODO: Read [SSR18; AZ17; BEP+19]**]

[**TODO: Read A Generalist Agent**]

[**TODO: Read meta**]

16.2 Learning Resource

- Learning to learn: An Introduction to Meta Learning || International Conference on Machine Learning (ICML) 2019

16.3 Definitions

Conventional supervised learning relies on large and diverse dataset for broad generalization. There are, however, problems with limited labeled data. These scenarios would need a general purpose AI system to adapt and learn on the job.

Mechanistic view	Probabilistic view
- Deep neural network model that can read in an entire dataset and make predictions for new datapoints - Training this network uses a meta-dataset, which itself consists of many datasets, each for a different task - Easier to implement meta-learning	- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks - Learning a new task uses this prior and (small) training set to infer most likely posterior parameters - Easier to understand meta-learning

- If you've learned 100 tasks already, can you figure out how to learn more efficiently?
- Meta-learning = learning to learn
- In practice, very closely related to multi-task
- A meta-learned learner can:
  - Explore more intelligently
  - Avoid trying actions that are known to be useless
  - Acquire the right features more quickly

16.4 Supervised Meta-learning

The problem setup (Fig. 16.2):

- Meta-training: acquire your learned learning algorithm
- Meta-testing: use/adapt your learned learning algorithm
- Meta-training and meta-testing have separated training data and test data

Supervised learning:

$$f(x) \rightarrow y$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D})$$

Supervised meta-learning:

$$f(\mathcal{D}^{tr}, x) \rightarrow y$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{meta-train})$$

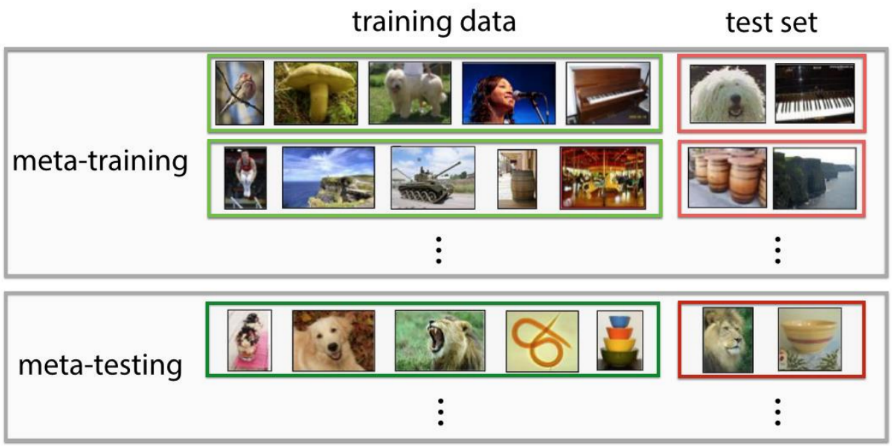


Figure 16.2: Supervised meta-learning pipeline (Ravi & Larochelle ’17)

$$
\begin{aligned}
&x && \text{– input}\\
&y && \text{– output}\\
&\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\} && \text{– dataset}\\
&\mathcal{D}_{meta-train} = \{(\mathcal{D}_1^{tr}, \mathcal{D}_1^{ts}), \ldots, (\mathcal{D}_n^{tr}, \mathcal{D}_n^{ts})\} && \text{– meta dataset}\\
&\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}
\end{aligned}
$$

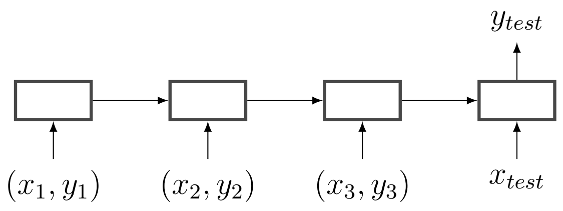


Figure 16.3: Using RNN to input a training set $\mathcal{D}^{tr}$, instead of just x.

The knowledge from the meta training dataset $\mathcal{D}_{meta-train}$ is extracted into parameter θ:

$$\theta^* = \arg\max_{\theta} \log p(\theta | \mathcal{D}_{meta-train}) \qquad \text{– meta-learning} \tag{16.1}$$

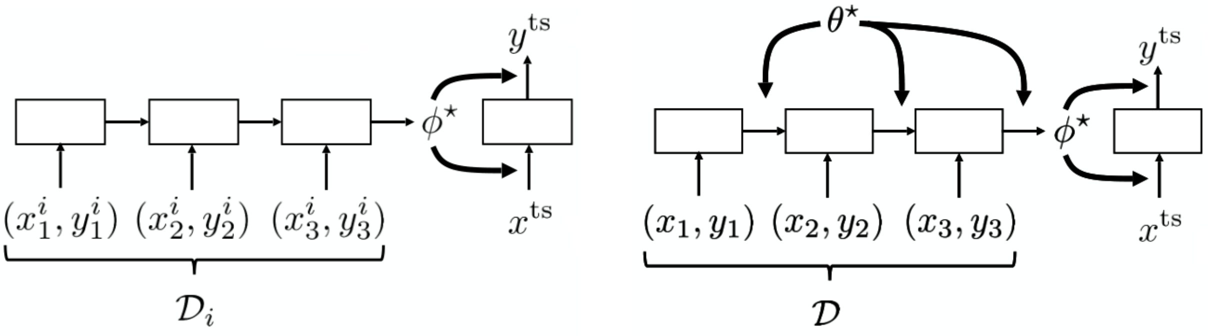
$$\log p(\phi | \mathcal{D}, \mathcal{D}_{meta-train}) \approx \log p(\phi | \mathcal{D}, \theta^*) \tag{16.2}$$

$$\phi^* = \arg\max_{\phi} \log p(\phi | \mathcal{D}, \theta^*) \qquad \text{– adaptation} \tag{16.3}$$

$$\phi^* = f_{\theta^*}(\mathcal{D}^{tr}) \tag{16.4}$$

[TODO: still confusing]

- In meta-training, each task has a training set $\mathcal{D}_i^{tr}$ and a test set $\mathcal{D}_i^{ts}$.
- ϕ_i is the params. learned from the task’s training set $\mathcal{D}_i^{tr}$ (Fig. 16.5).
  $\phi_i = [h_i, \theta_p]$ with h_i - RNN hidden state, θ_p - meta-learned weights.


y^{ts}
$\phi^\star$
x^{ts}
(x_1^i, y_1^i) (x_2^i, y_2^i) (x_3^i, y_3^i)
$\mathcal{D}_i$
$\theta^\star$
(x_1, y_1) (x_2, y_2) (x_3, y_3)
$\mathcal{D}$

(a) Meta-training: $(x^{ts}, y^{ts}) \sim \mathcal{D}_i^{ts}$ **(b)** Meta-testing

Figure 16.4: Matching test and train conditions.

- In meta-testing, we optimize θ, which is the arg min of the average over all the tasks (in meta-training), of the loss of that task on its test set $\mathcal{D}_i^{ts}$.

NOTE: The test set in meta-testing is not available (of course) for learning θ, but the test set for meta-training is (Fig. 16.2).

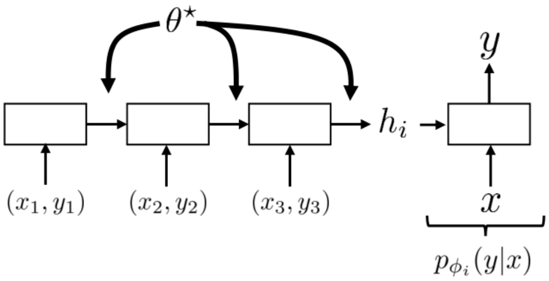


Figure 16.5: $\phi_i = [h_i, \theta_p]$ (src)

16.5 Meta Reinforcement Learning

Reinforcement learning:

$$\begin{aligned} \theta^* &= \arg\min_\theta \mathbb{E}_{\pi_\theta(\tau)}[R(\tau)] \\ &= f_{RL}(\mathcal{M}) \\ \mathcal{M} &= \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\} : \text{ the MDP} \end{aligned}$$

Meta-reinforcement learning:

$$\begin{aligned} &\theta^* = \arg\min_\theta \sum_{i=1}^{n} \mathbb{E}_{\pi_{\phi_i}(\tau)}[R(\tau)] \\ &\text{where } \phi_i = f_\theta(\mathcal{M}_i) \\ &\mathcal{M}_i : \text{ the MDP for task } i \end{aligned}$$

$$\mathcal{M}_i \sim p(\mathcal{M}) \tag{16.5}$$

$$\mathcal{M}_{test} \sim p(\mathcal{M}) \tag{16.6}$$

- In multi-task RL, the context is typically given.
- In meta-RL, the *context* is inferred from experience from $\mathcal{M}_i$: $\pi_\theta(a_t|s_t, \phi_i)$

16.6 Gradient-based Meta Learning

NOTE: The word *agnostic* implies that some characteristics are irrelevant and that something is widely applicable. E.g., a pick and place strategy is object-agnostic, if it doesn't matter what characteristics the object has, classes, geometries, materials, etc. In other words, that strategy is compatible to a variety of objects.

Idea: pretraining as a type of meta-learning.
In Model-Agnostic Meta-Learning (MAML), $f_\theta(\mathcal{M}_i)$ is itself an RL algor. [FAL17]

$$\theta^* = \arg\max \sum_{i=1}^{n} \mathbb{E}_{\pi_{\theta_i}}[r(\tau)] \tag{16.7}$$

$$\text{where } \phi_i = f_\theta(\mathcal{M}_i) \tag{16.8}$$

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta) \tag{16.9}$$

$$\cancel{\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)} \tag{16.10}$$

$$\theta \leftarrow \theta + \beta \sum_i \nabla_\theta J_i[\theta + \alpha \nabla_\theta J(\theta)] \tag{16.11}$$

16.7 Meta-RL as a POMDP

16.7.1 PEARL

16.7.2 MELD

16.8 Model-Based Meta-RL

16.9 Meta-RL and Emergent Phenomena

Humans and animals learn behaviors in a variety of ways. Perhaps each is a separate algorithm in the brain. But maybe these are all emergent phenomena resulting from meta-RL:

- Highly efficient but (apparently) model-free RL
- Episodic recall
- Model-based RL
- Causal inference
- etc.

References:

- Ritter et al. (2018) [RWKN+18]. *"Been there, done that: Meta-learning with episodic recall"*.
- Wang et al. (2018) [WKNK+18]. *"Prefrontal cortex as a meta-reinforcement learning system"*.
- Dasgupta et al. (2019) [DWC+19]. *"Causal reasoning from meta-reinforcement learning"*.

16.10 References

Gradient-based Meta-RL

MAML meta-policy gradient estimators:

- Finn et al. (2017) [FAL17]. *"Model-agnostic meta-learning for fast adaptation of deep networks"*.
- Foerster et al. (2018) [FFAS+18]. *"DICE: The infinitely differentiable monte carlo estimator"*.
- Rothfuss et al. (2018) [RLC+18]. *"ProMP: Proximal meta-policy search"*.

Improving exploration:

- Gupta et al. (2018) [GML+18]. *"Meta-reinforcement learning of structured exploration strategies"*.
- Stadie et al. (2018) [SYH+18]. *"Some considerations on learning to explore via meta-reinforcement learning"*.

Hybrid algorithms (not necessarily gradient-based):

- Houthooft et al. (2018) [HCI+18]. *"Evolved policy gradients"*.
- Fernando et al. (2018) [FSO+18]. *"Meta-learning by the Baldwin effect"*.

Meta-RL, Inference, and POMDPs

- Rakelly et al. (2019) [RZF+19]. *"Efficient off-policy meta-reinforcement learning via probabilistic context variables"*.
- Zintgraf et al. (2019) [ZIS+19]. *"Variational task embeddings for fast adaptation in deep reinforcement learning"*.
- Humplik et al. (2019) [HGH+19]. *"Meta reinforcement learning as task inference"*.
- Liu et al. (2020) [LRL+20]. *"Explore then Execute: Adapting without Rewards via Factorized Meta-Reinforcement Learning"*.

17 Continual Learning

18 RL for Robotics

18.1 Motor Primitive Policy

Not the same as path primitive from robotic trajectory planning. [INS02]. Motor plans are trajectory plans for each degrees of freedom (DOF):

- Spline-based trajectory plans: the desired movement is parameterized by its spline nodes and the duration of each spline node. [MGG+95; SKK08]
- Nonlinear dynamic motor primitives: the movement plans $(q_d, \dot{q}_d)$ are represented in terms of the time evolution of the nonlinear dynamical systems [INS02]

  $\ddot{q}_d = f(q_d, z, g, \tau, \theta)$ in which:

$(q_d, \dot{q}_d)$	– the desired position and velocity profile of a joint
z	– the internal state $\ddot{z} = f_c(z, \tau)$
θ	– params. for f
τ	– the time duration shared by all DOFs

We include exploration by adding a small perturbation to the desired accelerations

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \qquad \text{– small perturbation} \tag{18.1}$$

$$\ddot{\hat{q}}_d = \ddot{q}_d + \epsilon \qquad \text{– the perturbed target output} \tag{18.2}$$

$$\pi(\ddot{\hat{q}}_d | \ddot{q}_d) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\ddot{\hat{q}}_d - \ddot{q}_d)^2}{2\sigma^2}\right) \qquad \text{– a stochastic policy} \tag{18.3}$$

18.2 Requirements

Requirements for motor primitive learning in robotics: [PS08]

- Any change to the policy parameterization has to be smooth, because
  - drastic changes can be hazardous for the robot and its environment
  - rendering initialization of the policy based on domain knowledge or imitation learning useless, as these would otherwise vanish after a single update step due to out-of-distribution problem [Sch96].
- Need to guarantee that the policy is improved in the update steps at least on average. This rules out greedy value function based methods with approximated value functions, as these methods are frequently problematic in regard of this property [Kak03]

18.3 References

Resources:

- EPHE 245 Motor Learning
- Schaal (1996) [Sch96]. *"Learning from demonstration"*.
- Ijspeert et al. (2002) [INS02]. *"Movement imitation with nonlinear dynamical systems in humanoid robots"*.
- Kakade (2003) [Kak03].On the sample complexity of reinforcement learning.
- Peters et al. (2008) [PS08]. *"Reinforcement learning of motor skills with policy gradients"*.

19 Challenges and Open Problems

19.1 References

- Deep Reinforcement Learning Doesn't Work Yet [Irp18]
- CS 285 at UC Berkeley, Deep RL

Multi-task transfer learning: deeper explanation for gradient interference and winner-take-all problem

19.2 Stability

19.2.1 Problem

- Devising stable RL algorithms is very hard
- Q-learning/value function estimation
  - Fitted Q/fitted value methods with deep network function estimators are typically not contractions, hence no guarantee of convergence
  - Lots of parameters for stability: target network delay, replay buffer size, clipping, sensitivity to learning rates, etc.
- Policy gradient/likelihood ratio/REINFORCE
  - Very high variance gradient estimator
  - Lots of samples, complex baselines, etc.
  - Parameters: batch size, learning rate, design of baseline
- Model-based RL algorithms
  - Model class and fitting method
  - Optimizing policy w.r.t. model non-trivial due to back-propagation
  - More subtle issue: policy tends to exploit the model
- How representative is your simulator? Usually the answer is "not very"

19.2.2 Directions

Can we develop more stable algorithms that are less sensitive to hyperparameters?

- Algorithms with favorable improvement and convergence properties
  - Trust region policy optimization [SLA+15]

  - Safe RL, high-confidence policy improvement
- Algorithms that adaptively adjust parameters
  - Q-Prop: Sample-efficient policy gradient with an off-policy critic [GLG+16]

19.3 Sample Complexity

19.3.1 Problem

Efficiency: how long does it take to converge? (how many samples)

- Real-world learning becomes difficult or impractical
- Precludes the use of expensive, high-fidelity simulators
- Limits applicability to real-world problems

[Irp18]

19.3.2 Directions

- Better model-based RL algorithms
- Design faster algorithms
  - Addressing Function Approximation Error in Actor-Critic Algorithms [FHM18]
  - Soft Actor-Critic [HZA+18]
- Reuse prior knowledge to accelerate reinforcement learning
  - RL2: Fast reinforcement learning via slow reinforcement learning [DSC+16]
  - Learning to reinforcement learning [WKNT+16]
  - Model-agnostic meta-learning [FAL17]
  - DREAM [LRL+20]

19.4 Scaling and Generalization

Off-policy and Offline RL

19.5 Problem Formulation

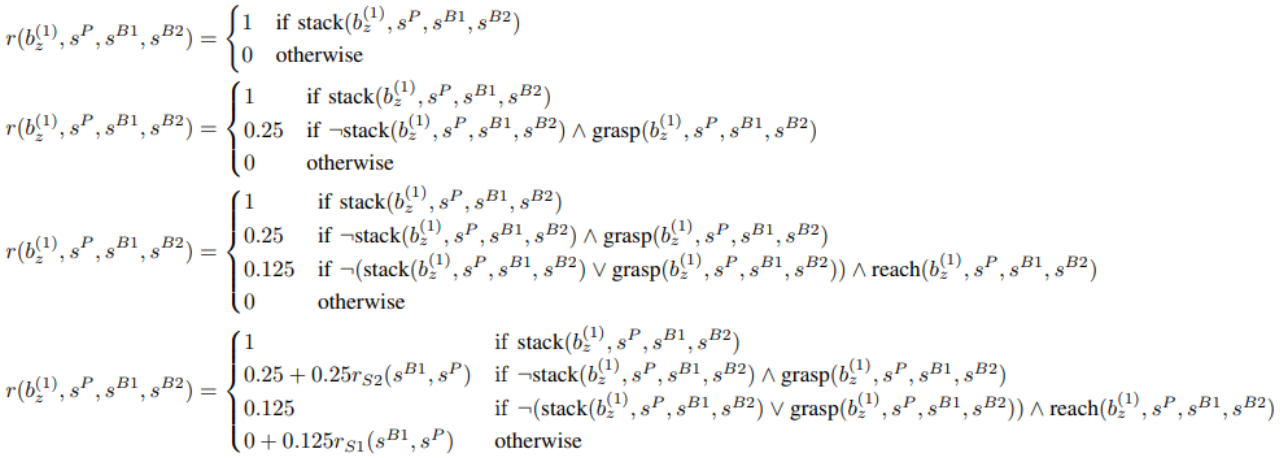
19.5.1 Single Task or Multi-Task

We have the assumption that the trained tasks and the test task are from the same distribution. As the result, with enough trials, the agent will visit every possible situations. But in real world, it's never like that.

Prior works on generalizing from multi-task learning:

- Train on multiple tasks, then try to generalize or finetune
  - Policy distillation [RCG+15]
  - Actor-mimic [PBS15]
  - Model-agnostic meta-learning [FAL17]
- Unsupervised or weakly supervised learning of diverse behaviors
  - Stochastic neural networks [FDA17]
  - Reinforcement learning with deep energy-based policies [HTA+17]
  - Unsupervised information-theoretic exploration

19.5.2 Supervision



$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 & \text{otherwise} \end{cases}$$

$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.25 & \text{if } \neg\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \wedge \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 & \text{otherwise} \end{cases}$$

$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.25 & \text{if } \neg\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \wedge \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.125 & \text{if } \neg(\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \vee \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2})) \wedge \text{reach}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 & \text{otherwise} \end{cases}$$

$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.25 + 0.25 r_{S2}(s^{B1}, s^P) & \text{if } \neg\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \wedge \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.125 & \text{if } \neg(\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \vee \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2})) \wedge \text{reach}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 + 0.125 r_{S1}(s^{B1}, s^P) & \text{otherwise} \end{cases}$$

Figure 19.1: Complex reward function [PHL+17]

- If you want to learn from many different tasks, you need to get those tasks somewhere!
- Supervision comes from reward.
- Learn objectives/rewards from demonstration (inverse RL)
- Generate objectives automatically?

Unsupervised RL

- Interact with the world, without a reward function
- Learn something about the world (what?)
- Use what you learned to quickly solve new tasks

20 Technical Tools

This chapter talks about or at least lists out helpful tools, platforms for working with RL

20.1 Supporting Platforms & Tools

Supporting platforms for simulation

- MuJoCo
- OpenAI `Gym`: is a standard API for RL, and a diverse collection of reference environments.
- `PyBullet`: physics simulation for games, visual effects, robotics and RL.

20.2 Coding Libraries

- Acme: DeepMind's research framework for RL
- TensorFlow Agents
- OpenAI Baselines: is a set of high-quality implementations of RL algorithms.

21 Research Proposal

One major challenge in RL is the ability to generalize to different tasks. For CV with supervised learning, a model can be trained on a large dataset (e.g., ImageNet), then retrained on a smaller dataset to adapt with a specific task. This is possible because it was proven that early layers manage to learn different feature representations from the image. Thus, the feature extraction and inference process (based on that feature learning) can be separated. It is uncertain though, what a RL model manages to learn in early layers.

What RL differs from more conventional supervised learning are large diverse dataset, training procedure, how and when the data is collected, etc. There are various proposed solution for RL generalization problem: transfer learning, multi-task learning, meta-RL. Offline RL is one of those. It's strange though, as offline RL aims to limit the interaction between the model and the world. Yet this interaction is what differs RL with other AI algorithms.

21.1 Goal-oriented RL

Why the dataset is packed in the way transitions goes with rewards $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$. Prior works on single task have an underlying goal, from that goal, we have rewards for each state-action transition in the dataset. With different tasks, the rewards should be different, should the transition then goes with a certain goal. This is obvious in the multi-task setting. But even for single task setting, it's a bit strange because human's actions aren't often based on maximize rewards (keep running as far as we can, etc.). Even in business, we divide to quarterly objectives. Is it weird, even with discount factor or constrained reward windows? $\Rightarrow$ conditional RL or task-oriented RL. What if we remove the reward completely and just decide our actions based on desired goal and current state?

completely removing the reward is also the core idea behind imagined-goal-based exploration strategy, etc. But what if we apply this idea to common RL training and testing

NOTE:

- The reward can be seen as a representation for the goal.
- Eliminate all problems with value functions (overconfidence, etc.)
- Having a natural extension to multi-task learning

21.1.1 Goal-oriented Imitation Learning

A policy π_θ that is conditional on the goal g, take in state $\mathbf{s}$ and output action a (preferably joint configurations). The policy is optimized to copy the expert behaviors

$$\pi_\theta(\mathbf{a}|\mathbf{s}, g) \tag{21.1}$$

21.1.2 Goal-oriented RL

Same as above, but the policy is optimized not by maximize the reward, but by how far / different the goal state is, from the current state.

The prior reward is basically a representation of how different the goal state from the current state.

Would these solve the instability, overestimation problem of value-based RL approaches?

21.1.3 CGAN

Conditional Generative Adversarial Network (CGAN) idea:

- Generator: $\mathbf{a} = G(\mathbf{s}|g)$
- Discriminator: $D(\mathbf{s}, \mathbf{a}, g) = good/bad?$

[**TODO: How is this relates to the notion of *"context"* in the literature?**]

21.2 Connection with Psychology

21.2.1 Classical Conditioning

The Pavlov conditioning: the experiment on dogs has shown conditioned response (CR)s simply adapt to conditioned stimuli (CS). Single task learning perhaps fail to generalize and might lead to overfitting probably because it is just an over-adaptation of the neural network to that single task.

21.2.2 Blocking conditioning

- Experiment 1: CS1 and unconditioned stimuli (US) lead to CR
- Experiment 2: CS2 then CS1 and US lead to CR
- Experiment 3: CS2 alone doesn't lead to CR

The Rescorla–Wagner model (Rescorla and Wagner, 1972) offer an influential explanation for blocking

21.2.3 High-order Conditioning

- Experiment 1: CS1 and US lead to CR
- Experiment 2: CS2 leads to CS1
- Experiment 3: CS2 leads to CR

21.3 Connection with Neuroscience

Bibliography

[AKL17] J. Andreas, D. Klein, and S. Levine. “Modular multitask reinforcement learning with policy sketches”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2017, pp. 166–175.

[Ama98] S.-I. Amari. “Natural gradient works efficiently in learning”. In: *Neural Computation* 10.2 (1998), pp. 251–276.

[AN04] P. Abbeel and A. Y. Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. 2004, p. 1.

[AZ17] R. Arandjelovic and A. Zisserman. “Look, listen and learn”. In: *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*. 2017, pp. 609–617.

[BB01] J. Baxter and P. L. Bartlett. “Infinite-horizon policy-gradient estimation”. In: *Journal of Artificial Intelligence Research* 15 (2001), pp. 319–350.

[BCK+15] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. “Weight uncertainty in neural network”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2015, pp. 1613–1622.

[BEP+19] P. Barros, M. Eppe, G. I. Parisi, X. Liu, and S. Wermter. “Expectation learning for stimulus prediction across modalities improves unisensory classification”. In: *Frontiers in Robotics and AI* 6 (2019), p. 137.

[BES+18] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. “Exploration by random network distillation”. In: *arXiv preprint arXiv:1810.12894* (2018).

[BHT+18] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. “Sample-efficient reinforcement learning with stochastic ensemble value expansion”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 31 (2018).

[BIW+18] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, et al. “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”. In: *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[BPW+12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. “A survey of monte carlo tree search methods”. In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[BSO+16] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. "Unifying count-based exploration and intrinsic motivation". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 29 (2016).

[CCM+18] K. Chua, R. Calandra, R. McAllister, and S. Levine. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 31 (2018).

[CL11] O. Chapelle and L. Li. "An empirical evaluation of thompson sampling". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 24 (2011).

[Day93] P. Dayan. "Improving generalization for temporal difference learning: The successor representation". In: *Neural Computation* 5.4 (1993), pp. 613–624.

[DR11] M. Deisenroth and C. E. Rasmussen. "PILCO: A model-based and data-efficient approach to policy search". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. Citeseer. 2011, pp. 465–472.

[DSC+16] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. "RL2: Fast reinforcement learning via slow reinforcement learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[DWC+19] I. Dasgupta, J. Wang, S. Chiappa, J. Mitrovic, P. Ortega, D. Raposo, E. Hughes, P. Battaglia, M. Botvinick, and Z. Kurth-Nelson. "Causal reasoning from meta-reinforcemen learning". In: *arXiv preprint arXiv:1901.08162* (2019).

[EAC+20] B. Eysenbach, S. Asawa, S. Chaudhari, S. Levine, and R. Salakhutdinov. "Off-dynamics reinforcement learning: Training for transfer with domain classifiers". In: *arXiv preprint arXiv:2006.13916* (2020).

[EFL+17] F. Ebert, C. Finn, A. X. Lee, and S. Levine. "Self-Supervised Visual Planning with Temporal Skip Connections." In: *CoRL*. 2017, pp. 344–356.

[EGI+18] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. "Diversity is all you need: Learning skills without a reward function". In: *arXiv preprint arXiv:1802.06070* (2018).

[FAL17] C. Finn, P. Abbeel, and S. Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2017, pp. 1126–1135.

[FCA+16] C. Finn, P. Christiano, P. Abbeel, and S. Levine. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: *arXiv preprint arXiv:1611.03852* (2016).

[FCRL17] J. Fu, J. Co-Reyes, and S. Levine. “EX2: Exploration with exemplar models for deep reinforcement learning”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 30 (2017).

[FDA17] C. Florensa, Y. Duan, and P. Abbeel. “Stochastic neural networks for hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1704.03012* (2017).

[FFAS+18] J. Foerster, G. Farquhar, M. Al-Shedivat, T. Rocktäschel, E. Xing, and S. Whiteson. “DICE: The infinitely differentiable monte carlo estimator”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2018, pp. 1529–1538.

[FHM18] S. Fujimoto, H. Hoof, and D. Meger. “Addressing function approximation error in actor-critic methods”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2018, pp. 1587–1596.

[FL17] C. Finn and S. Levine. “Deep visual foresight for planning robot motion”. In: *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE. 2017, pp. 2786–2793.

[FLA16] C. Finn, S. Levine, and P. Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 49–58.

[FLL17] J. Fu, K. Luo, and S. Levine. “Learning robust rewards with adversarial inverse reinforcement learning”. In: *arXiv preprint arXiv:1710.11248* (2017).

[FSO+18] C. Fernando, J. Sygnowski, S. Osindero, J. Wang, T. Schaul, D. Teplyashin, P. Sprechmann, A. Pritzel, and A. Rusu. “Meta-learning by the Baldwin effect”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2018, pp. 1313–1320.

[FWS+18a] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. “Model-based value expansion for efficient model-free reinforcement learning”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. 2018.

[FWS+18b] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. “Model-based value expansion for efficient model-free reinforcement learning”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. 2018.

[FYZ+17] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. “One-shot visual imitation learning via meta-learning”. In: *Conference on Robot Learning*. PMLR. 2017, pp. 357–368.

[GEF+18] A. Gupta, B. Eysenbach, C. Finn, and S. Levine. “Unsupervised meta-learning for reinforcement learning”. In: *arXiv preprint arXiv:1806.04640* (2018).

[GHK17] Y. Gal, J. Hron, and A. Kendall. “Concrete dropout”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 30 (2017).

[GLG+16] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine. “Q-prop: Sample-efficient policy gradient with an off-policy critic”. In: *arXiv preprint arXiv:1611.02247* (2016).

[GLS+16] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. “Continuous deep q-learning with model-based acceleration”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 2829–2838.

[GML+18] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. “Meta-reinforcement learning of structured exploration strategies”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 31 (2018).

[GPAM+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 27 (2014).

[GRW16] K. Gregor, D. J. Rezende, and D. Wierstra. “Variational intrinsic control”. In: *arXiv preprint arXiv:1611.07507* (2016).

[GSR+17] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine. “Divide-and-conquer reinforcement learning”. In: *arXiv preprint arXiv:1711.09874* (2017).

[GUA+16] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. “Domain-adversarial training of neural networks”. In: *Journal of Machine Learning Research* 17.1 (2016), pp. 2096–2030.

[HCD+16] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. “Vime: Variational information maximizing exploration”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 29 (2016).

[HCI+18] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel. “Evolved policy gradients”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 31 (2018).

[HE16] J. Ho and S. Ermon. “Generative adversarial imitation learning”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 29 (2016).

[HGH+19] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess. “Meta reinforcement learning as task inference”. In: *arXiv preprint arXiv:1905.06424* (2019).

[HKS+19] E. Hazan, S. Kakade, K. Singh, and A. Van Soest. “Provably efficient maximum entropy exploration”. In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2019, pp. 2681–2691.

[HM17] A. Hallak and S. Mannor. "Consistent on-line off-policy evaluation". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2017, pp. 1372–1383.

[HTA+17] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. "Reinforcement learning with deep energy-based policies". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2017, pp. 1352–1361.

[HZA+18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2018, pp. 1861–1870.

[INS02] A. J. Ijspeert, J. Nakanishi, and S. Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots". In: *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. Vol. 2. IEEE. 2002, pp. 1398–1403.

[Irp18] A. Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. https://www.alexirpan.com/2018/02/14/rl-hard.html. 2018.

[JDJ17] S. James, A. J. Davison, and E. Johns. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task". In: *Conference on Robot Learning*. PMLR. 2017, pp. 334–343.

[JFZ+19] M. Janner, J. Fu, M. Zhang, and S. Levine. "When to trust your model: Model-based policy optimization". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 32 (2019).

[JL16] N. Jiang and L. Li. "Doubly robust off-policy value evaluation for reinforcement learning". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 652–661.

[JM70] D. H. Jacobson and D. Q. Mayne. *Differential dynamic programming*. 24. Elsevier Publishing Company, 1970.

[Kak01] S. M. Kakade. "A natural policy gradient". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 14 (2001).

[Kak03] S. M. Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[KFS+19] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. "Stabilizing off-policy q-learning via bootstrapping error reduction". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 32 (2019).

[KGO12] H. J. Kappen, V. Gómez, and M. Opper. "Optimal control as a graphical model inference problem". In: *Journal of Machine Learning* 87.2 (2012), pp. 159–182.

[KIP+18] D Kalashnikov, A Irpan, P Pastor, J Ibarz, A Herzog, E Jang, D Quillen, E Holly, M Kalakrishnan, V Vanhoucke, et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).

[KKL+20] S. Kumar, A. Kumar, S. Levine, and C. Finn. “One solution is not all you need: Few-shot extrapolation via structured maxent rl”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 33 (2020), pp. 8198–8210.

[KN09] J. Z. Kolter and A. Y. Ng. “Near-Bayesian exploration in polynomial time”. In: *Proc. of the Int. Conf. on Machine Learning (ICML.* 2009, pp. 513–520.

[LA14] S. Levine and P. Abbeel. “Learning neural network policies with guided policy search under unknown dynamics”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 27 (2014).

[LEP+19] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. “Efficient exploration via state marginal matching”. In: *arXiv preprint arXiv:1906.05274* (2019).

[Lev18] S. Levine. “Reinforcement learning and control as probabilistic inference: Tutorial and review”. In: *arXiv preprint arXiv:1805.00909* (2018).

[LFD+16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[LHP+15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[LK13] S. Levine and V. Koltun. “Guided policy search”. In: *Proc. of the Int. Conf. on Machine Learning (ICML.* PMLR. 2013, pp. 1–9.

[LKT+20] S. Levine, A. Kumar, G. Tucker, and J. Fu. “Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems”. In: *arXiv preprint arXiv:2005.01643* (2020).

[LP03] M. G. Lagoudakis and R. Parr. “Least-squares policy iteration”. In: *The Journal of Machine Learning Research* 4 (2003), pp. 1107–1149.

[LR10] S. Lange and M. Riedmiller. “Deep auto-encoder neural networks in reinforcement learning”. In: *The 2010 International Joint Conference on Neural Networks (IJCNN).* IEEE. 2010, pp. 1–8.

[LRL+20] E. Z. Liu, A. Raghunathan, P. Liang, and C. Finn. "Explore then Execute: Adapting without Rewards via Factorized Meta-Reinforcement Learning". In: (2020).

[LSA+19] Y. Liu, A. Swaminathan, A. Agarwal, and E. Brunskill. "Off-policy policy gradient with state distribution correction". In: *arXiv preprint arXiv:1904.08473* (2019).

[MBM+16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 1928–1937.

[MGG+95] H Miyamoto, F Gandolfo, H Gomi, S Schaal, Y Koike, R Osu, E Nakano, Y Wada, and M Kawato. "A kendama learning robot based on a dynamic optimization theory". In: *Proceedings 4th IEEE International Workshop on Robot and Human Communication*. IEEE. 1995, pp. 327–332.

[MKS+15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[MSH+16] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. "Safe and efficient off-policy reinforcement learning". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 29 (2016).

[NKL+20] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. "Deep dynamics models for learning dexterous manipulation". In: *Conference on Robot Learning*. PMLR. 2020, pp. 1101–1112.

[NNX+17] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. "Bridging the gap between value and policy based reinforcement learning". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 30 (2017).

[NPD+18] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. "Visual reinforcement learning with imagined goals". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 31 (2018).

[OBP+16] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. "Deep exploration via bootstrapped DQN". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 29 (2016).

[Par] *What is Scaffolding?* https://youtu.be/rVaRdVt6Ihw. Accessed: 2022/06/30. 2019.

[PBS15] E. Parisotto, J. L. Ba, and R. Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342* (2015).

[PDL+19] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. "Skew-fit: State-covering self-supervised reinforcement learning". In: *arXiv preprint arXiv:1903.03698* (2019).

[PHL+17] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. "Data-efficient deep reinforcement learning for dexterous manipulation". In: *arXiv preprint arXiv:1704.03073* (2017).

[Pre00] D. Precup. "Eligibility traces for off-policy policy evaluation". In: *Computer Science Department Faculty Publication Series* (2000), p. 80.

[PRP+18] P. Parmas, C. E. Rasmussen, J. Peters, and K. Doya. "PIPPS: Flexible model-based policy search robust to the curse of chaos". In: *Proc. of the Int. Conf. on Machine Learning (ICML.* PMLR. 2018, pp. 4065–4074.

[PS02] L. Peshkin and C. R. Shelton. "Learning from scarce experience". In: *arXiv preprint cs/0204043* (2002).

[PS08] J. Peters and S. Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (2008), pp. 682–697.

[PVS05] J. Peters, S. Vijayakumar, and S. Schaal. "Natural Actor-Critic". In: *Proc. of the European Conf. on Machine Learning (ECML.* Springer. 2005, pp. 280–291.

[RBZ06] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. "Maximum margin planning". In: *Proc. of the Int. Conf. on Machine Learning (ICML.* 2006, pp. 729–736.

[RCG+15] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. "Policy distillation". In: *arXiv preprint arXiv:1511.06295* (2015).

[RGB11] S. Ross, G. Gordon, and D. Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics.* Workshop and Conference Proceedings. 2011, pp. 627–635.

[RGR+16] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. "EpOpt: Learning robust neural network policies using model ensembles". In: *arXiv preprint arXiv:1610.01283* (2016).

[RHI+20] K. Rao, C. Harris, A. Irpan, S. Levine, J. Ibarz, and M. Khansari. "Rl-CycleGAN: Reinforcement learning aware simulation-to-real". In: *Proc. of the IEEE/CVF Int. Conf. on Computer Vision and Pattern Recognition (CVPR).* 2020, pp. 11157–11166.

[Rie05] M. Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *Proc. of the European Conf. on Machine Learning (ECML.* Springer. 2005, pp. 317–328.

[RLC+18] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel. "ProMP: Proximal meta-policy search". In: *arXiv preprint arXiv:1810.06784* (2018).

[RTV12] K. Rawlik, M. Toussaint, and S. Vijayakumar. "On stochastic optimal control and reinforcement learning by approximate inference". In: *Proceedings of Robotics: Science and Systems VIII* (2012).

[RVR14] D. Russo and B. Van Roy. "Learning to optimize via information-directed sampling". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 27 (2014).

[RWKN+18] S. Ritter, J. Wang, Z. Kurth-Nelson, S. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick. "Been there, done that: Meta-learning with episodic recall". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2018, pp. 4354–4363.

[RZF+19] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2019, pp. 5331–5340.

[SB18] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[SCA17] J. Schulman, X. Chen, and P. Abbeel. "Equivalence between policy gradients and soft q-learning". In: *arXiv preprint arXiv:1704.06440* (2017).

[Sch10] J. Schmidhuber. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)". In: *IEEE transactions on autonomous mental development* 2.3 (2010), pp. 230–247.

[Sch91] J. Schmidhuber. "A possibility for implementing curiosity and boredom in model-building neural controllers". In: *Proc. of the international conference on simulation of adaptive behavior: From animals to animats.* 1991, pp. 222–227.

[Sch96] S. Schaal. "Learning from demonstration". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 9 (1996).

[SKK08] B. Siciliano, O. Khatib, and T. Kröger. *Springer Gandbook of Robotics.* Vol. 200. Springer, 2008.

[SL08] A. L. Strehl and M. L. Littman. "An analysis of model-based interval estimation for Markov decision processes". In: *Journal of Computer and System Sciences* 74.8 (2008), pp. 1309–1331.

[SL16] F. Sadeghi and S. Levine. "CAD2RL: Real single-image flight without a single real image". In: *arXiv preprint arXiv:1611.04201* (2016).

[SLA+15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. "Trust region policy optimization". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2015, pp. 1889–1897.

[SLA15] B. C. Stadie, S. Levine, and P. Abbeel. "Incentivizing exploration in reinforcement learning with deep predictive models". In: *arXiv preprint arXiv:1507.00814* (2015).

[SML+15] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[SMS+99] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 12 (1999).

[SPG19] P. Sharma, D. Pathak, and A. Gupta. "Third-person visual imitation learning via decoupled hierarchical controller". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 32 (2019).

[SSR18] A. Soltoggio, K. O. Stanley, and S. Risi. "Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks". In: *Neural Networks* 108 (2018), pp. 48–67.

[Sut90] R. S. Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. Elsevier, 1990, pp. 216–224.

[SWD+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[SYH+18] B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. "Some considerations on learning to explore via meta-reinforcement learning". In: *arXiv preprint arXiv:1803.01118* (2018).

[SYY+20] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. "Cog: Connecting new skills to past experience with offline reinforcement learning". In: *arXiv preprint arXiv:2010.14500* (2020).

[TB16] P. Thomas and E. Brunskill. "Data-efficient off-policy policy evaluation for reinforcement learning". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 2139–2148.

[TBC+17] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. "Distral: Robust multitask reinforcement learning". In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 30 (2017).

[TDH+20] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell. “Adapting deep visuomotor representations with weak pairwise constraints”. In: *Algorithmic Foundations of Robotics XII.* Springer, 2020, pp. 688–703.

[TET12] Y. Tassa, T. Erez, and E. Todorov. “Synthesis and stabilization of complex behaviors through online trajectory optimization”. In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).* IEEE. 2012, pp. 4906–4913.

[TFR+17] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).* IEEE. 2017, pp. 23–30.

[THF+17] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. “# exploration: A study of count-based exploration for deep reinforcement learning”. In: *Advances in neural information processing systems* 30 (2017).

[Tho14] P. Thomas. “Bias in natural actor-critic algorithms”. In: *Proc. of the Int. Conf. on Machine Learning (ICML.* PMLR. 2014, pp. 441–448.

[THZ+14] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. “Deep domain confusion: Maximizing for domain invariance”. In: *arXiv preprint arXiv:1412.3474* (2014).

[Tod06] E. Todorov. “Linearly-solvable Markov decision problems”. In: *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)* 19 (2006).

[Tod08] E. Todorov. “General duality between optimal control and estimation”. In: *2008 47th IEEE Conference on Decision and Control.* IEEE. 2008, pp. 4286–4292.

[TTG15] P. Thomas, G. Theocharous, and M. Ghavamzadeh. “High-confidence off-policy evaluation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 29. 1. 2015.

[VHGS16] H. Van Hasselt, A. Guez, and D. Silver. “Deep reinforcement learning with double q-learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 30. 1. 2016.

[Wat89] C. J. C. H. Watkins. “Learning from delayed rewards”. In: (1989).

[Wil92] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Journal of Machine Learning* 8.3 (1992), pp. 229–256.

[WKNK+18] J. X. Wang, Z. Kurth-Nelson, D. Kumaran, D. Tirumala, H. Soyer, J. Z. Leibo, D. Hassabis, and M. Botvinick. "Prefrontal cortex as a meta-reinforcement learning system". In: *Nature neuroscience* 21.6 (2018), pp. 860–868.

[WKNT+16] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. "Learning to reinforcement learn". In: *arXiv preprint arXiv:1611.05763* (2016).

[WOP15] M. Wulfmeier, P. Ondruska, and I. Posner. "Maximum entropy deep inverse reinforcement learning". In: *arXiv preprint arXiv:1507.04888* (2015).

[WSH+16] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. "Dueling network architectures for deep reinforcement learning". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. PMLR. 2016, pp. 1995–2003.

[YTL+17] W. Yu, J. Tan, C. K. Liu, and G. Turk. "Preparing for the unknown: Learning a universal policy with online system identification". In: *arXiv preprint arXiv:1702.02453* (2017).

[ZBC20] L. Zaadnoordijk, T. R. Besold, and R. Cusack. "The next big thing(s) in unsupervised machine learning: Five lessons from infant learning". In: *arXiv preprint arXiv:2009.08497* (2020).

[ZBD10] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. "Modeling interaction via the principle of maximum causal entropy". In: *Proc. of the Int. Conf. on Machine Learning (ICML*. 2010.

[ZIS+19] L. Zintgraf, M. Igl, K. Shiarlis, A. Mahajan, K. Hofmann, and S. Whiteson. "Variational task embeddings for fast adaptation in deep reinforcement learning". In: *International Conference on Learning Representations Workshop (ICLRW)*. 2019.

[ZMB+08] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.