RL Notes

Huu Duc Nguyen M.Sc.

2 May 2022

Contents

Abbreviations

AI	Artificial Intelligence
RL	Reinforcement 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
GD	Gradient Descent
K-L	Kullback–Leibler
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
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

1 Overview

Reinforcement Learning (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:

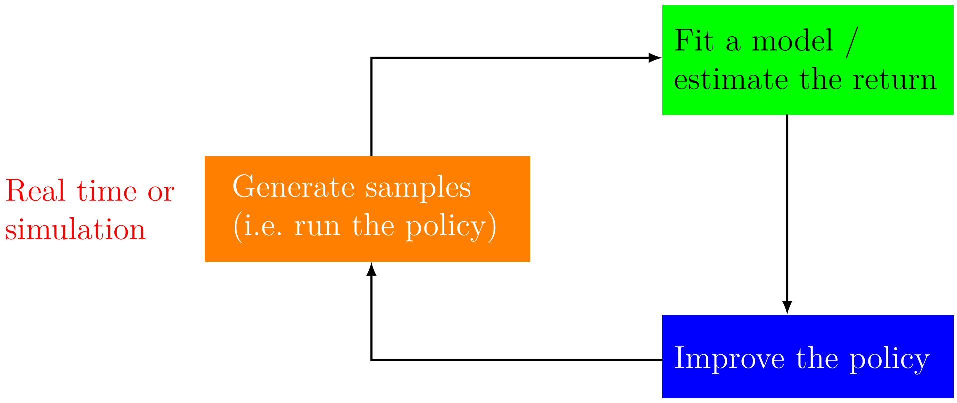

Fit a model / estimate the return
Real time or simulation
Generate samples (i.e. run the policy)
Improve the policy

Figure 1.1: Structure of RL algorithms.

1.1 Learning resources

- Deep RL - CS285, UC Berkeley - Sergey Levine
- CS188 Berkeley AI

1.2 Terminology & Notation

- Check out Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) in the robotic notes.
- $\mathbf{s}_t$ - 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)

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

- 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.

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

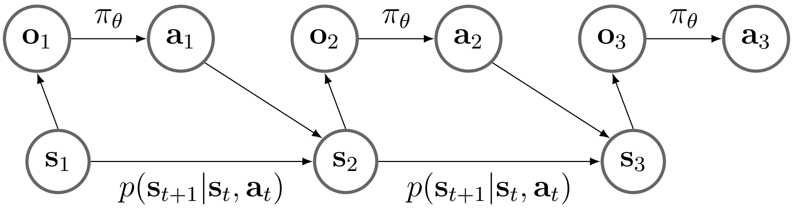


Figure 1.2: The relationship between state $\mathbf{s}_t$, observation $\mathbf{o}_t$ and action $\mathbf{a}_t$.

1.3 Overview on RL Algorithms

RL problem revolves around maximizing the expectation of total rewards. Thus, 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 because they have 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

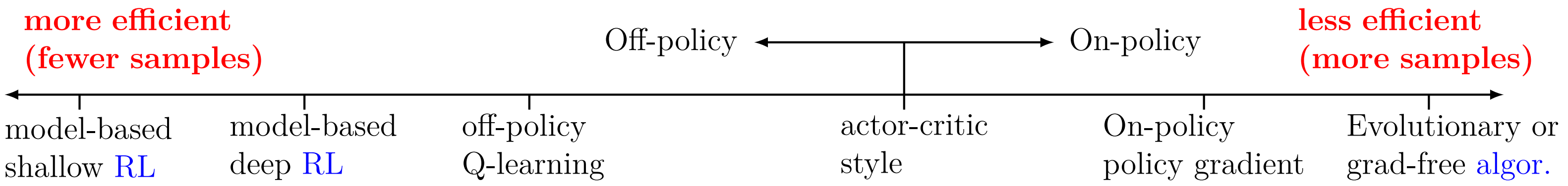
	Model-based approaches	Value function fitting methods	Actor-critic algorithms	Policy Gradient
Sample efficiency (How many samples do we need to get good policy?)	 **more efficient** **(fewer samples)** Off-policy On-policy **less efficient** **(more samples)** model-based shallow RL model-based deep 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 - SAC	- REINFORCE - Natural policy gradient - Trusted Region policy optimization

Table 1.1: Different RL algorithms.

2 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.

2.1 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 $\mathcal{D}_\pi = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
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.

2.2 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)

3 Policy Gradient

The Policy Gradient approach has a neural network (Fig. 3.1) to learn and optimize the policy (the blue box in Fig. 1.1). This is a model-free RL approach. For most model-free RL approach, we assume that we don't know the transition model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ or the initial state probability (prob.) $p(\mathbf{s}_1)$. However, we assume that we can interact with the real world to sample the data.

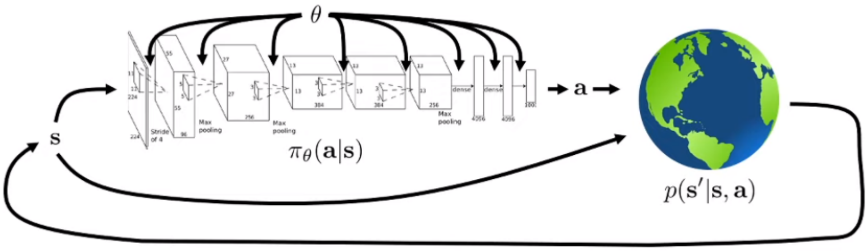

θ
$\mathbf{a}$
$\mathbf{s}$
$\pi_\theta(\mathbf{a}|\mathbf{s})$
$p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

Figure 3.1: 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 $\mathbf{a}_t$.

3.1 Approach

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

$$
\begin{aligned}
\tau &= \{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\} && \text{denotes the trajectory} \quad (3.1)\\
p_\theta(\tau) &= p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) && \text{prob. of the trajectory} \quad (3.2)\\
&= p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) && (3.3)\\
\theta^* &= \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] && \text{RL goal} \quad (3.4)\\
&= \arg\max_\theta \mathbb{E}_{(\mathbf{s},\mathbf{a}) \sim p_\theta(\mathbf{s},\mathbf{a})}[r(\mathbf{s},\ \mathbf{a})] && \textbf{infinite} \text{ horizon case} \quad (3.5)\\
&= \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t) \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)] && \textbf{finite} \text{ horizon case} \quad (3.6)\\
&= \arg\max_\theta J(\theta) && (3.7)\\
J(\theta) &= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right] && (3.8)\\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau && (3.9)
\end{aligned}
$$

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.

$$J(\theta) \approx \frac{1}{N} \sum_i^N \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \qquad \text{sum over samples and time steps} \tag{3.10}$$

The Policy Gradient:

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau \tag{3.11}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) r(\tau) \right] \tag{3.12}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \tag{3.13}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left[\left(\sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right) \right] \tag{3.14a}$$

$$\underline{\text{then}} \quad \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{3.14b}$$

REMARKS:

- some what like Maximum Likelihood Estimation (MLE), makes good stuff happen more, bad stuff happens less
- Transition in Eq. 3.11 happens due to a convenient identity transformation:
$$p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$$
- Taking the log of $p_\theta(\tau)$ in Eq. 3.2 then replacing it into Eq. 3.12 leads to Eq. 3.13

3.2 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

3.3 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 time t' cannot affect reward at time t, when $t' < t$

$$\Rightarrow \ \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)\right] \tag{3.15}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\widehat{Q}_{i,t} \qquad (\widehat{Q}_{i,t} \text{ - the reward to-go}) \tag{3.16}$$

  This leads to smaller variance, because $\widehat{Q}_{i,t}$ is smaller than the total rewards, and the expectation of smaller number has smaller variance.

- *Baselines:* the average of total rewards over different trajectories

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau) \tag{3.17}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log\pi_\theta(\tau)[r(\tau)-b] \tag{3.18}$$

$$r(\tau) - b = Q^\pi(\mathbf{s}_t,\mathbf{a}_t) - V^\pi(\mathbf{s}_t) \tag{3.19}$$

  It is proven that subtracting the baseline is unbiased in expectation. This is not the best baseline to reduce the variance, but it's simple and good enough.

3.4 Off-policy Policy Gradient

Vanilla Policy Gradient is on-policy, since we have $\tau \sim \ p_\theta(\tau)$. 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**:

$$\begin{aligned}
&& \mathbb{E}_{x \sim p(x)}[f(x)] &= \mathbb{E}_{x \sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right] && (3.20)\\
\Rightarrow && \nabla_{\theta'} J(\theta') &= \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p_\theta(\tau)} r(\tau)\right] && (3.21)\\
&& &= \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau)\right] && (3.22)\\
&& &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \widehat{Q}_{i,t} && (3.23)
\end{aligned}$$

with θ' as the *new* params. and θ as the *old* params.

3.5 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]

Pseudo code: (`tensorflow`)
When coding, use the pseudo loss as a weighted maximum likelihood:

$$\widetilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \widehat{Q}_{i,t} \qquad (3.24)$$

```
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.

3.6 Natural Policy Gradient

Natural Policy gradients, also known as (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. [PS08]

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \quad \text{s.t. } ||\theta' - \theta||^2 \leq \epsilon \quad \text{(params. space)} \tag{3.25}$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \quad \text{s.t. } D(\pi_{\theta'}, \pi_\theta) \leq \epsilon \quad \text{(policy space)} \tag{3.26}$$

A good choice for $D(\pi_{\theta'}, \pi_\theta)$ is the Kullback–Leibler (K-L)-divergence. To simplify the process, the K-L-divergence is approximated with Fisher information matrix

$$D_{KL}(\pi_{\theta'}||\pi_\theta) \approx (\theta' - \theta)^T \mathbf{F}(\theta' - \theta) \tag{3.27}$$

$$\mathbf{F} = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})^T] \quad \text{Fisher information matrix} \tag{3.28}$$

$$\theta' \leftarrow \arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta J(\theta) \quad \text{s.t. } ||\theta' - \theta||^2_{\mathbf{F}} \leq \epsilon \tag{3.29}$$

$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta) \tag{3.30}$$

3.7 References

- Peters and Schaal (2008) [PS08]. Reinforcement learning of motor skills with policy gradients.
- Levine and Koltun (2013) [LK13]. Guided policy search: deep RL with importance sampled policy gradient.
- Schulman et al. (2015) [SLA+15]. Trust region policy optimization: deep RL with natural policy gradient and adaptive step size.
- Schulman et al. (2017) [SWD+17]. Proximal policy optimization algorithms: deep RL with importance sampled policy gradient

4 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)**

4.1 Approach

We continue with the Eq. 3.16, 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 sum of the reward afterward, in a single run. 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.
$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(\mathbf{s}_{t'}, a_{t'})|\mathbf{s}_t, \mathbf{a}_t] \tag{4.1}$$
- $V^\pi(\mathbf{s}_t)$: the state value function represents the total reward from state $\mathbf{s}_t$.
$$V^\pi(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t, \mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)] \tag{4.2}$$
- $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: the advantage function: represents how much action $\mathbf{a}_t$ is better than average
$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t) \tag{4.3}$$

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

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \tag{4.4}$$

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \mathbb{E}_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} V^\pi(\mathbf{s}_{t+1}) \qquad \text{(Eq. 4.1)} \tag{4.5}$$

$$\approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) \qquad \text{(with 1 sample)} \tag{4.6}$$

$$\Rightarrow \quad A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t) \qquad \text{(Eq. 4.3)} \tag{4.7}$$

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

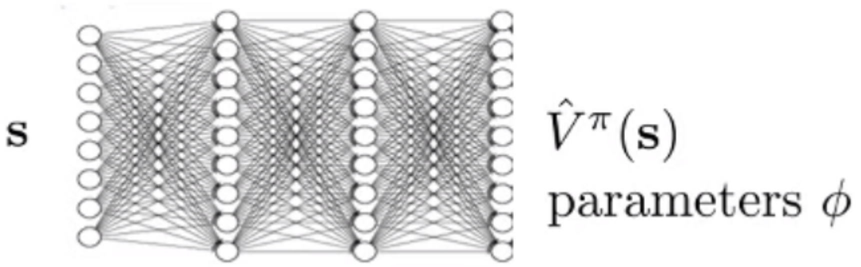


Figure 4.1: The network for value function $V^\pi_\phi(s)$ with params. ϕ.

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

$$V^\pi(\mathbf{s}_t) \approx \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{4.8}$$

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!

Better (but not possible)
$$V^\pi(\mathbf{s}_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \tag{4.9}$$

Training data:
$$\{(\mathbf{s}_{i,t}, y_{i,t})\} = \left\{\left(\mathbf{s}_{i,t}, \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right)\right\} \tag{4.10}$$

Supervised regression:
$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left|\left|\widehat{V}^\pi_\phi(\mathbf{s}_i) - y_i\right|\right|^2 \tag{4.11}$$

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

$$\begin{aligned} y_{i,t} &= \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}[r(\mathbf{s}_{t'}, a_{t'}|\mathbf{s}_{i,t})] && \text{ideal target} \quad (4.12)\\ &\approx r(\mathbf{s}_{i,t}, a_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) && (4.13)\\ &\approx r(\mathbf{s}_{i,t}, a_{i,t}) + \widehat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) && (4.14)\end{aligned}$$

Training data:
$$\{(\mathbf{s}_{i,t}, y_{i,t})\} = \left\{\left(\mathbf{s}_{i,t}, r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \widehat{V}^\pi_\phi(\mathbf{s}_{i,t+1})\right)\right\} \tag{4.15}$$

Supervised regression:
$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left|\left|\widehat{V}^\pi_\phi(\mathbf{s}_i) - y_i\right|\right|^2 \tag{4.16}$$

4.2 Batch Actor-Critic

```
1. Sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
2. Fit $V^\pi_\phi(\mathbf{s})$ to sampled reward sums (either Monte-Carlo or bootstrapped estimate)
3. Evaluate $\widehat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \widehat{V}^\pi_\phi(\mathbf{s}_{i'}) - \widehat{V}^\pi_\phi(\mathbf{s}_i)$
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 $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i'}) - \hat{V}^\pi_\phi(\mathbf{s}_i)$ [Tho14]

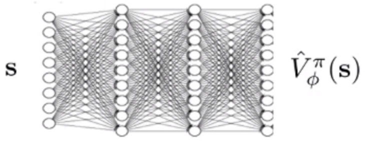
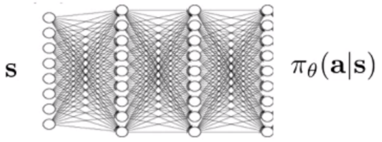
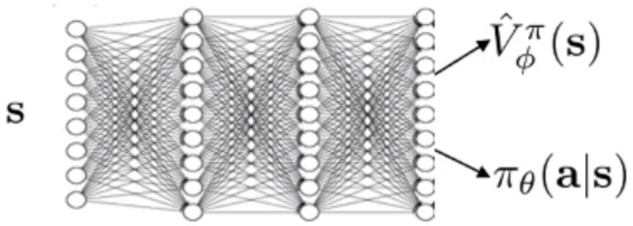
4.3 Online Actor-Critic

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

Problem: single batch $\Rightarrow$ need parallel actor-critic (synchronous / asynchronous)

4.4 Design Decisions

- Architecture design:

Two separate networks	Shared network design
+simple and stable −no shared features between actor & critic	+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)

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

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

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

$\Rightarrow$ Critic as baseline: +no bias
+lower variance (baseline is closer to rewards)

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

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:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\widehat{Q}_{i,t} - Q_\phi^\pi(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right) + \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \mathbb{E}_{\mathbf{a}\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_{i,t})}[Q_\phi^\pi(\mathbf{s}_{i,t},\mathbf{a}_{i,t})] \tag{4.17}$$

- Eligibility traces & n-step returns: reduces the bias

Actor-critic: $$\widehat{A}_C^\pi(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma \widehat{V}_\phi^\pi(\mathbf{s}_{t+1}) - \widehat{V}_\phi^\pi(\mathbf{s}_t) \tag{4.18}$$

+low variance
−but biased

Monte-Carlo: $$\widehat{A}_{MC}^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'},a_{t'}) - \widehat{V}_\phi^\pi(\mathbf{s}_t) \tag{4.19}$$

+no bias
−higher variance

$$\Rightarrow \qquad \widehat{A}_n^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(\mathbf{s}_{t'},a_{t'}) - \widehat{V}_\phi^\pi(\mathbf{s}_t) + \gamma^n \widehat{V}_\phi^\pi(\mathbf{s}_{t+n}) \tag{4.20}$$

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
  To have many n-step returns, then take weighted average of them.

$$\widehat{A}_{GAE}^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{n=1}^{\infty} \omega_n \widehat{A}_n^\pi(\mathbf{s}_t,\mathbf{a}_t), \qquad \omega_n \propto \lambda^{n-1} \tag{4.21}$$

$$\Rightarrow \widehat{A}_{GAE}^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{\infty} (\gamma\lambda)^{t'-t}\delta_{t'}, \qquad \delta_{t'} = r(\mathbf{s}_{t'},\mathbf{a}_{t'}) + \gamma \widehat{V}_\phi^\pi(\mathbf{s}_{t'+1}) - \widehat{V}_\phi^\pi(\mathbf{s}_{t'}) \tag{4.22}$$

4.5 References

- Sutton, McAllester, Singh, and Mansour (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.

5 Value Function Based Algorithms

5.1 Approach

Knowing the value functions, we could just remove the policy gradient completely. The advantage value 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 $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

5.2 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 $\pi \leftarrow \pi'$

5.3 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 $V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s},\ \mathbf{a})$

5.4 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. $\phi \leftarrow \arg\min_{\phi} \frac{1}{2}\sum_i ||V_\phi(\mathbf{s}_i) - \mathbf{y}_i||^2$
```

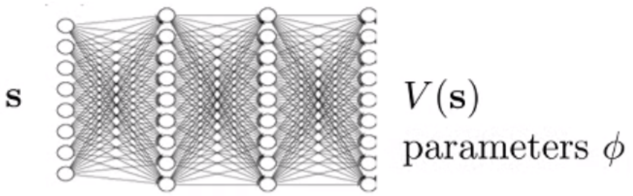


Figure 5.1: The network for value function $V_\phi(s)$ with params. ϕ.

Problem: still need to know transition dynamic.

5.5 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×  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 $\phi \leftarrow \arg\min_{\phi} \frac{1}{2}\sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i||^2$
```

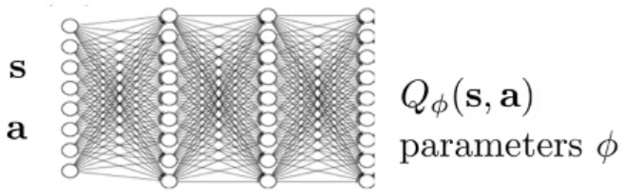


Figure 5.2: The network for Q-value function $Q_\phi(s, a)$ with params. ϕ.

+Off-policy (unlike actor-critic)

+Single network, no high-variance policy gradient

−Not really converge

5.6 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. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\,[Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i]$ **1 gradient step**

Problems:

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

5.7 Exploration vs Exploitation

- Epsilon greedy

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1-\epsilon & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ \dfrac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases} \tag{5.2}$$

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

$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t)) \tag{5.3}$$

5.8 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. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\left(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \left[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{a'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)\right]\right)$

$N\times$ $K\times$

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

5.9 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. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)\,(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. Update ϕ': copy ϕ every N steps

The above *"Classic"* Deep Q-learning (DQN) is essentially Q-learning with $K = 1$

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 ⇒ should always use**

$$\text{Standard Q-learning:} \qquad y = r + \gamma Q_{\phi'}\left(s', \arg\max_{a'} Q_{\phi'}(s', a')\right) \tag{5.4}$$

$$\text{Double Q-learning:} \qquad y = r + \gamma Q_{\phi'}\left(s', \arg\max_{a'} Q_{\phi}(s', a')\right) \tag{5.5}$$

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

$$\text{Q-learning target:} \qquad y_{i,t} = r_{i,t} + \gamma \max_{a_{i,t+1}} Q_{\phi'}(\mathbf{s}_{i,t+1}, a_{i,t+1}) \tag{5.6}$$

$$\text{Multi-step target:} \qquad y_{i,t} = \sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{i,t'} + \gamma^N \max_{a_{i,t+N}} Q_{\phi'}(\mathbf{s}_{i,t+N}, a_{i,t+N}) \tag{5.7}$$

+less biased target values when Q-values are inaccurate

+typically faster learning, especially early on

−Only actually CORRECT when learning on-policy

5.10 Q-learning with continuous action-space

[TODO:]

5.11 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 ⇒ gradually reduce
- Bellman error gradient can be quite large ⇒ clip gradients / use Huber loss

$$L(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \le \delta \\ \delta|x| - \frac{\delta^2}{2} & \text{otherwise} \end{cases} \qquad \text{(Huber loss)} \tag{5.8}$$

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

5.12 Policy Gradient as Policy Iteration

[TODO: math stuffs]

5.13 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 and Riedmiller (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.

6 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:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \quad \text{s.t. } \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) \quad \text{(Deterministic case)} \tag{6.1}$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \mathbb{E}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right] \quad \text{(Stochastic case)} \tag{6.2}$$

$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \quad \text{(stochastic dynamics)} \tag{6.3}$$

- 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 $\mathbf{s}_{t+1}$

6.1 Open-Loop Planning

Maximize objective through sequence of actions:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} J(\mathbf{a}_1, \ldots, \mathbf{a}_T) \quad \Rightarrow \quad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A}) \tag{6.4}$$

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.

6.2 Trajectory Optimization with Derivatives

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

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \tag{6.5}$$

$$= \min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, , \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)\dots), \mathbf{u}_T) \tag{6.6}$$

NOTE: Shooting methods versus (vs.) collocation:

- Shooting methods: optimize over actions only

  $$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, , \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)\dots), \mathbf{u}_T)$$

  tends to be very sensitive with early actions and leads to numerical instability.
- Collocation: optimize over actions and states, with constraints

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

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

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t \qquad \underline{\underline{\text{linear}}}\ \textbf{dynamics} \tag{6.7}$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_t \qquad \underline{\underline{\text{quadratic}}}\ \textbf{cost} \tag{6.8}$$

$$\mathbf{C}_T = \begin{bmatrix}\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\\ \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\end{bmatrix}; \quad \mathbf{c}_T = \begin{bmatrix}c_{\mathbf{x}_T}\\ c_{\mathbf{u}_T}\end{bmatrix} \tag{6.9}$$

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

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

$$Q(\mathbf{x}_T,\mathbf{u}_T) = const + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_T \tag{6.10}$$

$$\text{Set } \nabla_{\mathbf{u}_T} Q(\mathbf{x}_T,\mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0 \tag{6.11}$$

$$\Rightarrow \quad \mathbf{u}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}.\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}) = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \tag{6.12}$$

$$\mathbf{K}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \tag{6.13}$$

$$\mathbf{k}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{c}_{\mathbf{u}_T} \tag{6.14}$$

$$\tag{6.15}$$

Replace Eq. 6.12 into Eq. 6.10:

$$\Rightarrow \quad V(\mathbf{x}_T) = const + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_t\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T\end{bmatrix}^T \mathbf{c}_T \tag{6.16}$$

$$= const + \frac{1}{2}\mathbf{x}_T^T\mathbf{V}_T\mathbf{x}_T + \mathbf{x}_T^T\mathbf{v}_T \tag{6.17}$$

Minimize objective on last action u_T, based on x_T

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

Backward recursion:

$$\text{for } t = T \rightarrow 1: \tag{6.18}$$

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t \tag{6.19}$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1} \tag{6.20}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = const + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t \tag{6.21}$$

$$\mathbf{u}_t \leftarrow \underset{u_t}{\arg\min} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t \tag{6.22}$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} \tag{6.23}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t} \tag{6.24}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t \tag{6.25}$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t \tag{6.26}$$

$$V(\mathbf{x}_t) = const + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t \tag{6.27}$$

Forward recursion:

$$\text{for } t = 1 \rightarrow T: \tag{6.28}$$

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t \tag{6.29}$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \tag{6.30}$$

LQR for Stochastic and Non-Linear Systems

- Stochastic dynamics:

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) \tag{6.31}$$

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t \right) \tag{6.32}$$

  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

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} \tag{6.33}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} \tag{6.34}$$

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

$$\delta\mathbf{x}_t = \mathbf{x}_t - \widehat{\mathbf{x}}_t \tag{6.35}$$

$$\delta\mathbf{u}_t = \mathbf{u}_t - \widehat{\mathbf{u}}_t \tag{6.36}$$

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \qquad \text{with} \quad \mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \tag{6.37}$$

$$\bar{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t \qquad \text{with} \quad \begin{aligned} \mathbf{C}_t &= \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \\ \mathbf{c}_t &= \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t) \end{aligned} \tag{6.38}$$

⇒ iLQR (simplified pseudo code):

```
until convergence:                                                              (6.39)
    $\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                                    (6.40)
    $\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                                  (6.41)
    $\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\widehat{\mathbf{x}}_t, \widehat{\mathbf{u}}_t)$                                    (6.42)
    Run LQR backward pass on state $\delta\mathbf{x}_t$ and action $\delta\mathbf{u}_t$                  (6.43)
    Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \widehat{\mathbf{x}}_t) + \alpha\mathbf{k}_t + \widehat{\mathbf{u}}_t$     (6.44)
    Update $\widehat{\mathbf{x}}_t$ and $\widehat{\mathbf{u}}_t$ based on states and actions in forward pass     (6.45)
```

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

6.3 References

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

7 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.

7.1 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.

7.2 Model-based RL v.1.0

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

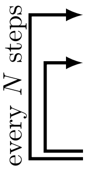
1. Run based policy $\pi_0(\mathbf{a}_t, \mathbf{s}_t)$ to collect $\mathcal{D} = \{(\mathbf{s},\ \mathbf{a},\ \mathbf{s'})_i\}$
2. Learn dynamic model $f(\mathbf{s},\mathbf{a})$ to minimize $\sum_i ||f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i||^2$
3. Plan through $f(\mathbf{s},\mathbf{a})$ to choose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

7.3 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. 6, e.g., random shooting, CEM, MCTS, LQR.

7.4 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

7.5 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].

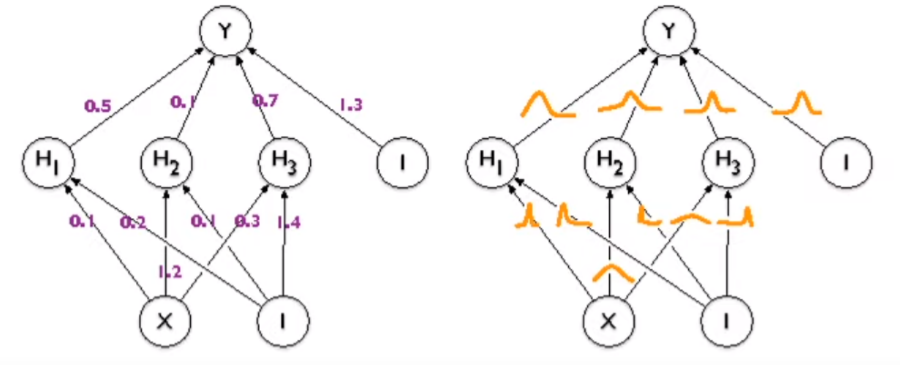


Figure 7.1: Normal neural network (left) and Bayesian neural network (right).

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D}) \qquad \text{common approximation} \tag{7.1}$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i) \qquad \text{common choice for each marginal prob.} \tag{7.2}$$

- 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 $\mathcal{D}$.

$$p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i) \qquad \text{mixture of delta func.} \tag{7.3}$$

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) \tag{7.4}$$

7.6 Planning with Uncertainty

As mentioned before, the model uncertainty is used in step 3 of the model-based RL algorithm (Subsec. 7.4). 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

7.7 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.

8 Model-based RL with Images

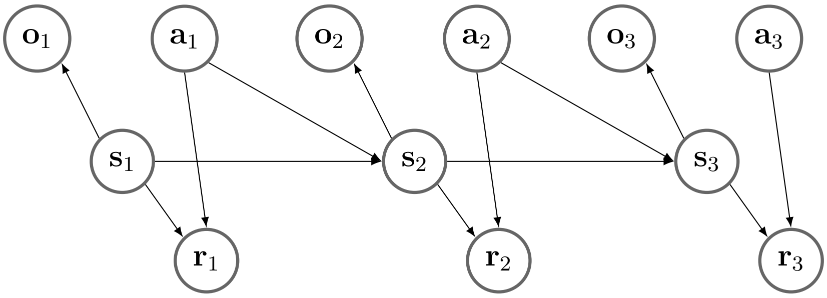
8.1 Latent Space Models

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

$$p(\mathbf{o}_t|\mathbf{s}_t) \qquad \text{- observation model: high-dimensional, but not dynamic} \tag{8.1}$$

$$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \qquad \text{- dynamics model: low-dimensional, but dynamic} \tag{8.2}$$

$$p(r_t|\mathbf{s}_t, \mathbf{a}_t) \qquad \text{- reward model} \tag{8.3}$$



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.

8.2 Deterministic Single-Step Encoder

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

$$q_\psi(\mathbf{s}_t|\mathbf{o}_t) = \delta(\mathbf{s}_t = g_\psi(\mathbf{o}_t)) \Rightarrow \mathbf{s}_t = g_\psi(\mathbf{o}_t) \qquad \text{deterministic encoder} \tag{8.4}$$

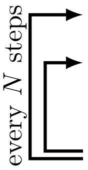
$\Rightarrow$ The goal for model-based RL with latent space is now:

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi \left[g_\psi(\mathbf{o}_{t+1,i}) | g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i} \right] + \log p_\phi \left[\mathbf{o}_{t,i} | g_\psi(\mathbf{o}_{t,i}) \right] \tag{8.5}$$

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

8.3 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 $\mathcal{D}$

8.4 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 and Levine (2017) [FL17]. Deep visual foresight for planning robot motion.
- Ebert, Finn, Lee, and Levine (2017) [EFL+17]. Self-Supervised Visual Planning with Temporal Skip Connections.

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

9 Model-Based Policy Learning

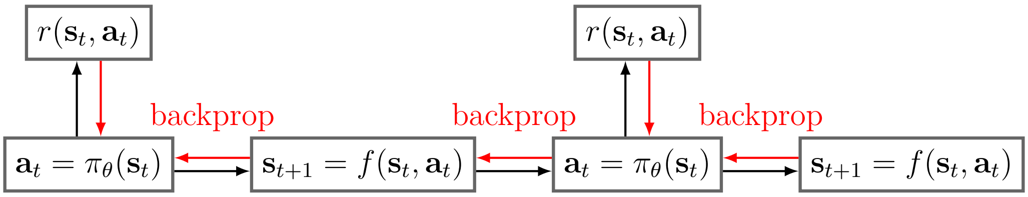
As mentioned before, model-based RL v.1.5 algorithm (Sec. 7.3) is a ***stochastic open-loop*** algorithm. The agent does see the next state, but it doesn't able to reason about the fact that more information will be available and make use out it. It simply plans the whole action sequence at each time 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: $$\begin{cases} p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T|\mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \\ \mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \mathbb{E}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{a}_1, \ldots, \mathbf{a}_T\right] \end{cases}$$
- Stochastic closed-loop case: $$\begin{cases} p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1)\prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \\ \pi = \arg\max_{\pi} \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \end{cases}$$

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

9.1 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

9.2 Model-Free Learning With a Model

This is one of the solutions for Model-based RL v.2.0 (Sec. 9.1): 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 $\pi$, 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]

9.3 Local Models

This is the second solution for model-based RL v.2.0 (Sec. 9.1): 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. 6.2), 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 $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}$.

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

If continuous system sufficiently smooth and initial state distribution quite tight ⇒ do linearization regression at every time step

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: $p(\mathbf{u}_t|\mathbf{x}_t) = \delta\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$
  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: $p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$
  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 K-L divergence small (between old and new trajectories) [LA14]

$$p(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t) \qquad \text{the controller} \tag{9.1}$$

$$p(\tau) = p(\mathbf{x}_1)\prod_{t=1}^{T} p(\mathbf{u}_t|\mathbf{x}_t)p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) \qquad \text{the resulting trajectory} \tag{9.2}$$

$$D_{KL}(p(\tau)||\bar{p}(\tau)) \leq \epsilon \qquad \text{constraint on K-L divergence} \tag{9.3}$$

9.4 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 demonstrations of expert and combine them into a global policy. The global policy can now be represented by a neural net and learned by 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. $\tilde{c}_{k,i}(\mathbf{x}_t, \mathbf{u}_t)$
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 $\tilde{c}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = c(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

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 K-L between $\pi_{LQR,i}$ and $\pi_\theta PP$

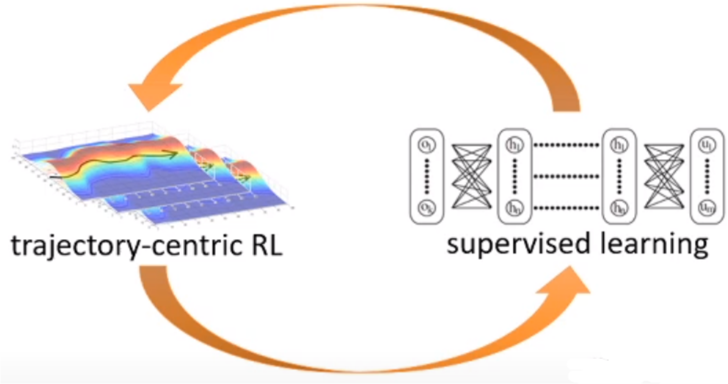


Figure 9.1: Guided policy search: algorithm sketch (src).

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 $\widetilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

9.5 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.

10 Exploration

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

10.1 Bandits Problems

10.1.1 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.

$$\mathcal{A} = \{\text{pull arm}\} \tag{10.1}$$

$$r(\text{pull arm}) = ? \tag{10.2}$$

Figure 10.1: One-armed bandit (src).

10.1.2 Multi-Armed Bandit

Multi armed bandit is a bank of multiple one-armed bandit slot machines. Different machines have different reward distribution. This problem is a 1-step stateless MDP

$$\mathcal{A} = \{\text{pull}_1, \text{pull}_2, \ldots, \text{pull}_n\} \tag{10.3}$$

$$r(a_n) = ? \tag{10.4}$$

$$\text{assume } r(a_n) \sim p(r|a_n) \tag{10.5}$$

10.1.3 Contextual Bandits

the reward distribution depends on some external measurable variable.

10.1.4 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!

10.1.5 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

10.2 Regret

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

$$Reg(T) = T.\mathbb{E}[r(a^*)] - \sum_{t=1}^{T} r(a_t) \qquad \text{regret}$$

$$\mathbb{E}[r(a^*)] \qquad \text{expected reward of best action}$$

$$\sum_{t=1}^{T} r(a_t) \qquad \text{the actual sum of reward}$$

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. Thus would not be used to calculate the regret. The actual one is:

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

- ϵ-greedy: Assume $\epsilon = 10\%$, the professor spend 10% of the days (30 days) to explore the distribution, and the rest to exploit the current belief.

  $$\rho \approx 100$$

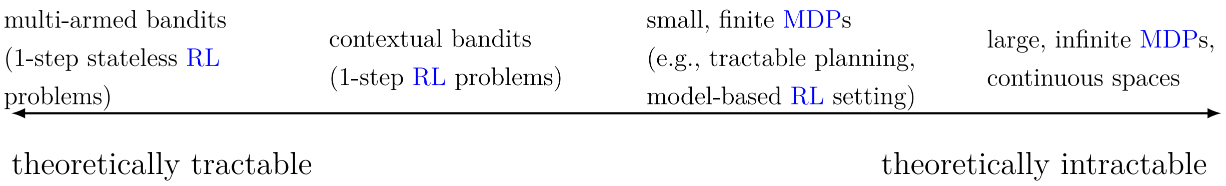
- Zero regret strategy: As time goes on $T \to \infty$, the regret will approach to 0 $\rho \to 0$

10.3 Optimality

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

10.4 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

10.5 ϵ-first

10.6 ϵ-greedy

10.7 UCB

Upper Confidence Bounce (UCB) would weigh actions based on their previous rewards and how many times they have been tested.

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

in which $Q_t(a)$ is the current belief about the reward, $N_t(a)$ is the number of times the action a was chosen, t is just the number of current time step. The second term measures how uncertain we currently are about the actions.

UCB-1 use Chernoff - Hoeffding Inequality:

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

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

UCB is more difficult than ϵ-greedy to extend beyond bandits to more general RL problems (nonstationary problems, large state spaces)

10.7.1 Algorithm

1. Pull each arm once
2. Update the reward belief
3. Choose the arm with the highest upper confidence bound.

10.8 Gradient Bandits

[TODO:]

11 Inverse Reinforcement Learning

Prior to this, we have been manually design the reward function, which defines the task. In other 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.

12 Challenges and Open Problems

Bibliography

[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.* PMLR. 2015, pp. 1613–1622.

[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* 31 (2018).

[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.

[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* 31 (2018).

[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.* Citeseer. 2011, pp. 465–472.

[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.

[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.

[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.* 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.* 2018.

[GHK17] Y. Gal, J. Hron, and A. Kendall. “Concrete dropout”. In: *Proc. of the Conf. on Neural Information Processing Systems* 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.* PMLR. 2016, pp. 2829–2838.

[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).

[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* 32 (2019).

[JM70] D. H. Jacobson and D. Q. Mayne. *Differential dynamic programming.* 24. Elsevier Publishing Company, 1970.

[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).

[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* 27 (2014).

[LFD+16] S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". In: *The 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.* PMLR. 2013, pp. 1–9.

[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.

[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.* PMLR. 2016, pp. 1928–1937.

[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* 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.

[PBS15] E. Parisotto, J. L. Ba, and R. Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342* (2015).

[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*. PMLR. 2018, pp. 4065–4074.

[PS08] J. Peters and S. Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural Networks* 21.4 (2008), pp. 682–697.

[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*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[Rie05] M. Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*. Springer. 2005, pp. 317–328.

[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*. PMLR. 2015, pp. 1889–1897.

[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* 12 (1999).

[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*. 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).

[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* 30 (2017).

[TET12] Y. Tassa, T. Erez, and E. Todorov. “Synthesis and stabilization of complex behaviors through online trajectory optimization”. In: *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 4906–4913.

[Tho14] P. Thomas. “Bias in natural actor-critic algorithms”. In: *Proc. of the Int. Conf. on Machine Learning*. PMLR. 2014, pp. 441–448.

[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.

[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*. PMLR. 2016, pp. 1995–2003.