CPEN 455: Deep Learning

Lecture 12: Deep Reinforcement Learning Part II

Renjie Liao

University of British Columbia

Winter, Term 2, 2024

Outline

- Reinforcement Learning (RL) I
  - Overview & Applications
  - RL Formulation & Taxonomy
  - Markov Decision Process (MDP)
  - Bellman Equations
  - Solving RL problems using the Bellman Equations
- Reinforcement Learning (RL) II
  - Model-Free RL: (Deep) Q-learning
  - Model-Free RL: Policy Gradient & Actor-Critic
  - Model-Based RL: Dyna-Q

RL Taxonomy: Recap

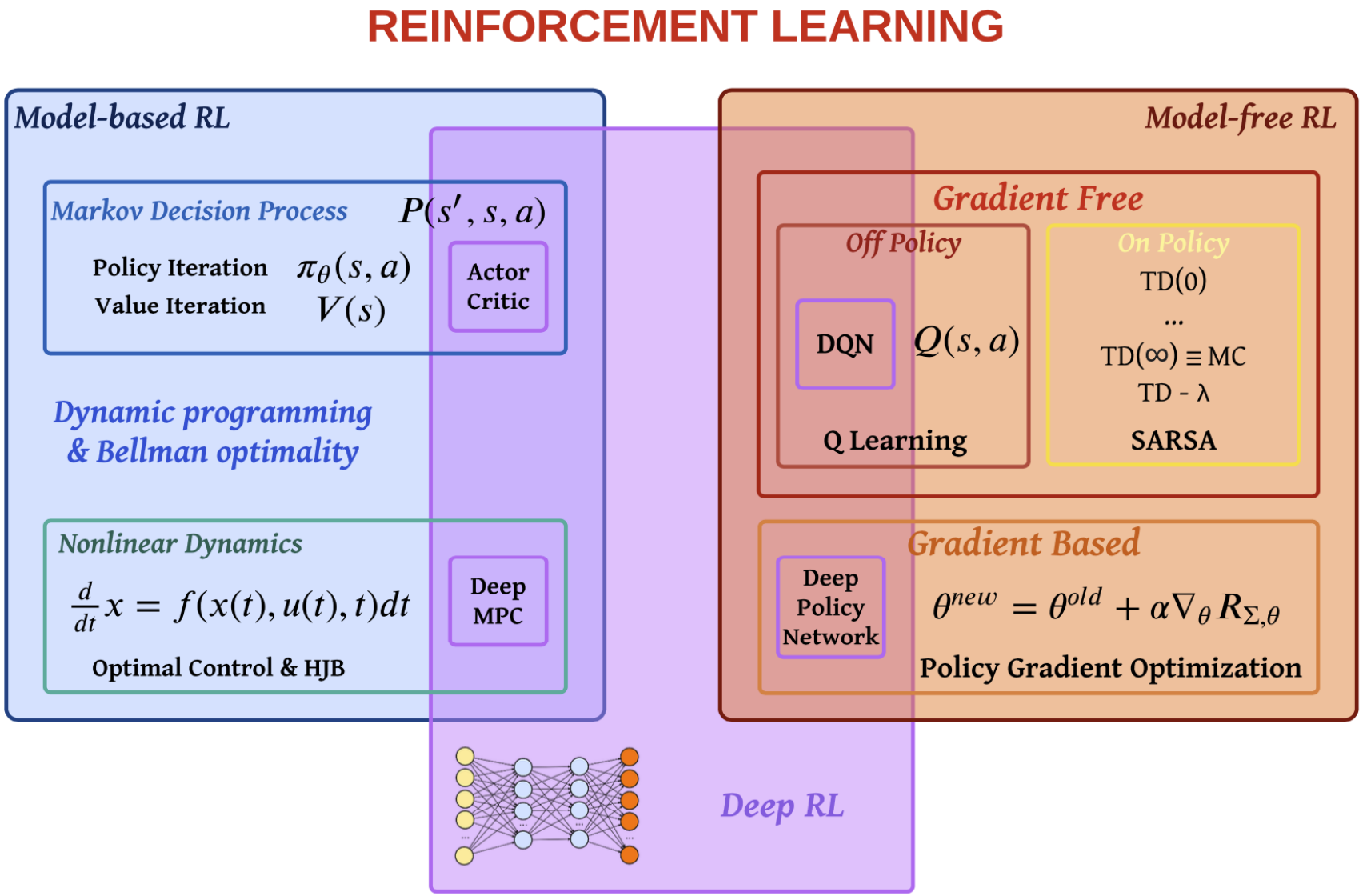


Image Credit: [1]

RL Taxonomy: Recap

Category	Subcategory	Examples	Does It Learn a Model?
Model-Free RL	**Value-Based**	Q-Learning, SARSA, DQN, TD-Learning	❌ No
	Policy-Based	REINFORCE, PPO, TRPO, SAC	❌ No
	Actor-Critic	A2C, A3C, DDPG, TD3	❌ No
Model-Based RL	**Explicit Model Learning**	Dyna-Q, AlphaZero, World Models	✅ Yes
	Planning Methods	Monte Carlo Tree Search (MCTS), Model-Predictive Control (MPC)	✅ Yes

RL Taxonomy: Recap

Category	Subcategory	Examples	Does It Learn a Model?
Model-Free RL	**Value-Based**	Q-Learning, SARSA, DQN, TD-Learning	❌ No
	Policy-Based	REINFORCE, PPO, TRPO, SAC	❌ No
	Actor-Critic	A2C, A3C, DDPG, TD3	❌ No
Model-Based RL	**Explicit Model Learning**	Dyna-Q, AlphaZero, World Models	✅ Yes
	Planning Methods	Monte Carlo Tree Search (MCTS), Model-Predictive Control (MPC)	✅ Yes

Model-Free RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy

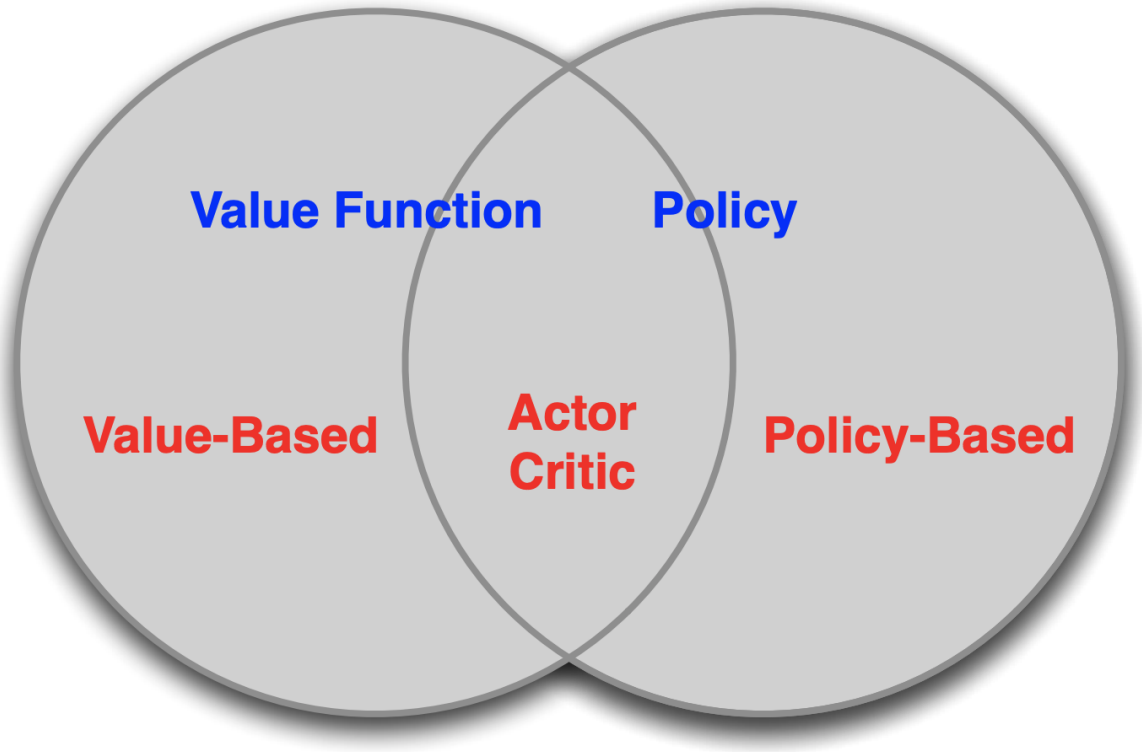
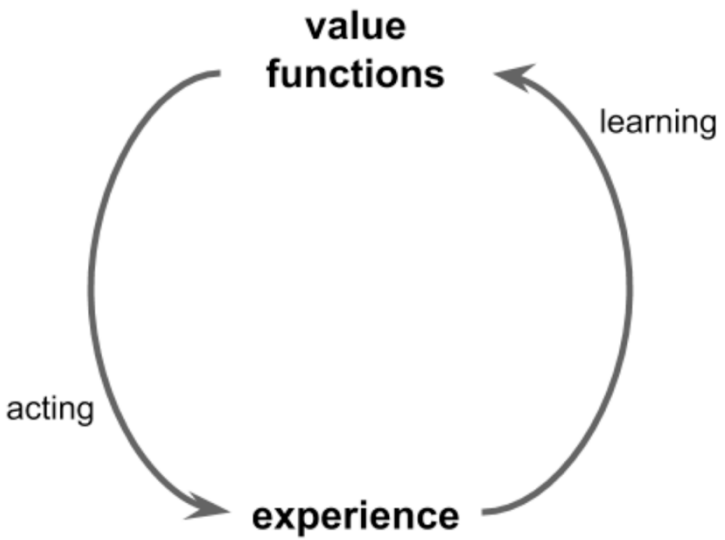


Image Credit: [2]

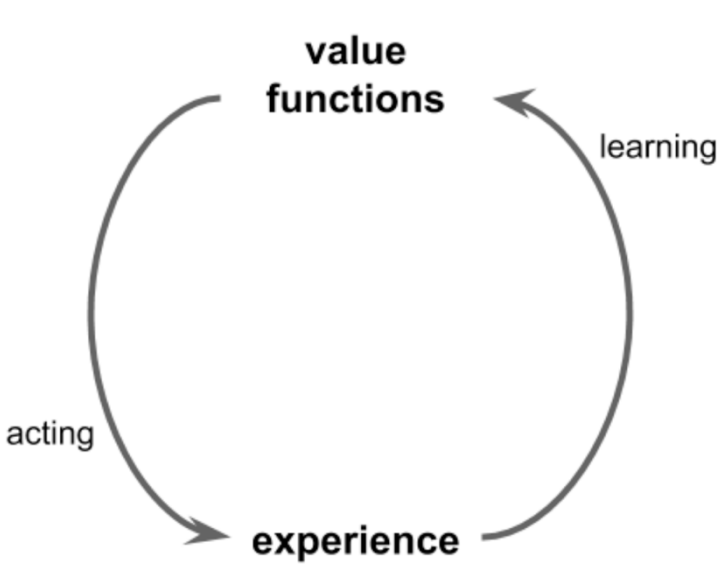
Model-Based RL



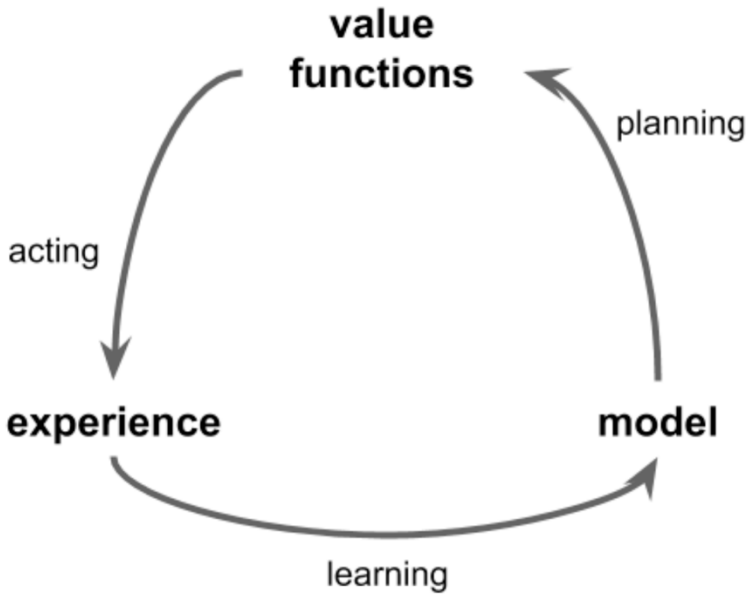
Model-Free RL

Image Credit: [3]

Model-Based RL



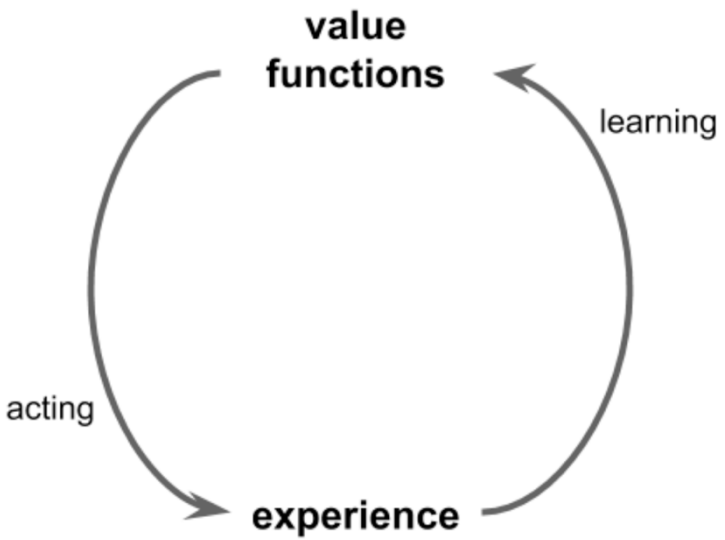
Model-Free RL



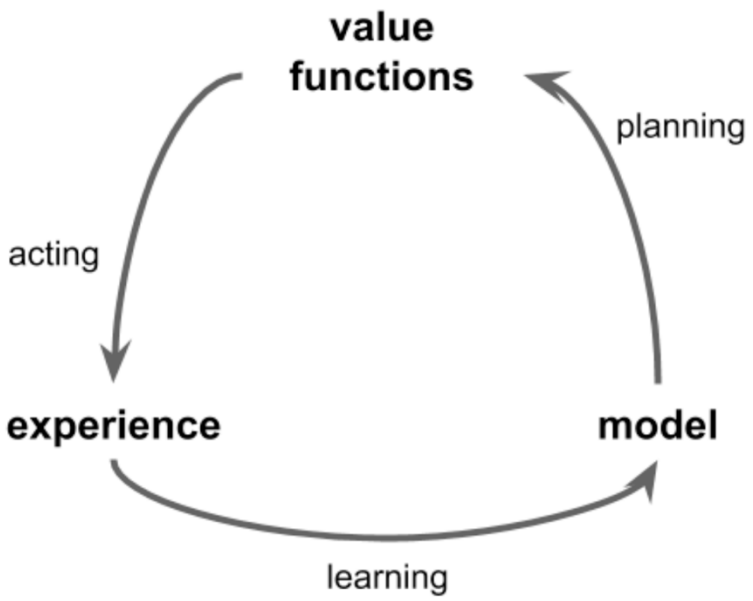
Model-Based RL

Image Credit: [3]

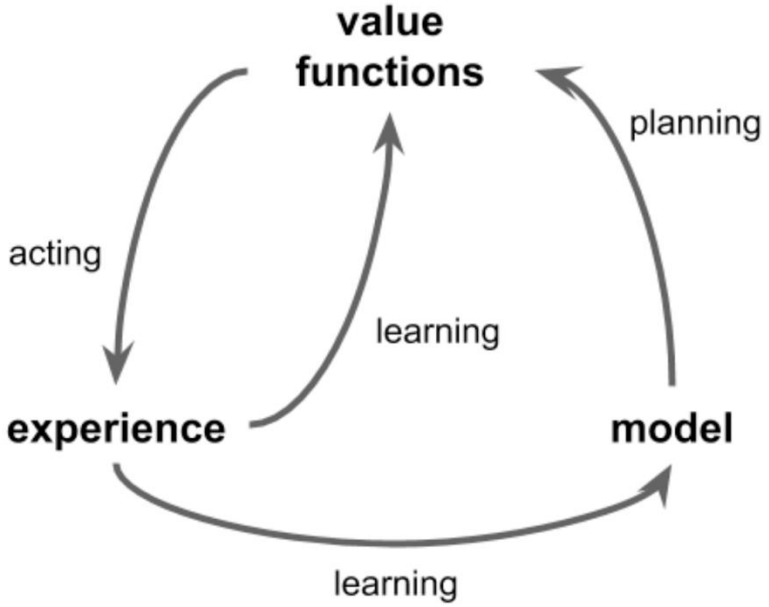
Model-Based RL



Model-Free RL



Model-Based RL



Model-Based RL

Image Credit: [3]

Outline

- Reinforcement Learning (RL) I
  - Overview & Applications
  - RL Formulation & Taxonomy
  - Markov Decision Process (MDP)
  - Bellman Equations
  - Solving RL problems using the Bellman Equations
- Reinforcement Learning (RL) II
  - **Model-Free RL: (Deep) Q-learning**
  - Model-Free RL: Policy Gradient & Actor-Critic
  - Model-Based RL: Dyna-Q

On-Policy vs. Off-Policy

- On-Policy Learning
  - “Learn on the job”
  - Learn about (target) policy π from experience sampled from (target) policy π

On-Policy vs. Off-Policy

- On-Policy Learning
  - "Learn on the job"
  - Learn about (target) policy π from experience sampled from (target) policy π
- Off-Policy Learning
  - "Look over someone's shoulder"
  - Learn about (target) policy π from experience sampled from (behavior) policy μ

Q-Learning

Consider the off-policy learning setup:

Sample episode from behavior policy $\{S_1, A_1, R_2, \ldots S_{T-1}, A_{T-1}, R_T, S_T\} \sim \mu$

We want to learn the target policy π via learning the Q function

Q-Learning

Consider the off-policy learning setup:

Sample episode from behavior policy $\{S_1, A_1, R_2, \ldots S_{T-1}, A_{T-1}, R_T, S_T\} \sim \mu$

We want to learn the target policy π via learning the Q function

Recall the *Bellman Optimality Equation*: $Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$

Q-Learning

Consider the off-policy learning setup:

Sample episode from behavior policy $\{S_1, A_1, R_2, \ldots S_{T-1}, A_{T-1}, R_T, S_T\} \sim \mu$

We want to learn the target policy π via learning the Q function

Recall the *Bellman Optimality Equation*: $Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$

We can define the *Bellman Error* (of one step) as:

$$\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

Q-Learning

Consider the off-policy learning setup:

Sample episode from behavior policy $\{S_1, A_1, R_2, \ldots S_{T-1}, A_{T-1}, R_T, S_T\} \sim \mu$

We want to learn the target policy π via learning the Q function

Recall the *Bellman Optimality Equation*: $Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$

We can define the *Bellman Error* (of one step) as:

$$\underbrace{\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a)}_{\text{Q value at } t+1 \text{ following greedy (target) policy } \pi} - \underbrace{Q(s_t, a_t)}_{\text{Q value at } t}$$

Q-Learning

Consider the off-policy learning setup:

Sample episode from behavior policy $\{S_1, A_1, R_2, \ldots S_{T-1}, A_{T-1}, R_T, S_T\} \sim \mu$

We want to learn the target policy π via learning the Q function

Recall the *Bellman Optimality Equation*: $Q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} Q_*(s', a')$

We can define the *Bellman Error* (of one step) as:

$$\underbrace{\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a)}_{\text{Q value at } t+1 \text{ following greedy (target) policy } \pi} - \underbrace{Q(s_t, a_t)}_{\text{Q value at } t}$$

Here the target policy is the greedy policy: $\pi(a|s) = \begin{cases} 1 & a = \operatorname{argmax}_a Q(s,a) \\ 0 & \text{otherwise} \end{cases}$

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Exploration-exploitation tradeoff: Q-learning only learns from the state-action pairs it visits. One often needs some strategy to improve the exploration, e.g., ϵ*-greedy policy* [5] (choose optimal action based on Q with probability 1- ϵ and choose a random action with probability ϵ).

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Exploration-exploitation tradeoff: Q-learning only learns from the state-action pairs it visits. One often needs some strategy to improve the exploration, e.g., *ϵ-greedy policy* [5] (choose optimal action based on Q with probability 1- ϵ and choose a random action with probability ϵ).

We thus choose the behavior policy to be:

$$\mu(a|s) = \begin{cases} 1 - \epsilon & a = \operatorname{argmax}_a \ Q(s, a) \\ \frac{\epsilon}{|\mathcal{A}| - 1} & \text{otherwise} \end{cases}$$

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Exploration-exploitation tradeoff: Q-learning only learns from the state-action pairs it visits. One often needs some strategy to improve the exploration, e.g., *ϵ-greedy policy* [5] (choose optimal action based on Q with probability 1- ϵ and choose a random action with probability ϵ).

We thus choose the behavior policy to be:

$$\mu(a|s) = \begin{cases} 1 - \epsilon & a = \operatorname{argmax}_a \ Q(s, a) \\ \frac{\epsilon}{|\mathcal{A}|-1} & \text{otherwise} \end{cases}$$

This ensures that we eventually visit all actions infinitely often!

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

With some mild (decaying) condition on the step size η, *Q-Learning* is guaranteed to converge:

Theorem

Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

Q-Learning

The idea of *Q-Learning* [4] is to learn a Q function that minimizes the Bellman Error. In particular, we can use the fix point iteration to update the Q function iteratively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \left[\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

With some mild (decaying) condition on the step size η, *Q-Learning* is guaranteed to converge:

Theorem

Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

For small state and action spaces, we can represent Q function as a table and learn it. However, for large or infinite spaces, we need to represent it as a parametric function, e.g., a deep neural network!

Deep Q-Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 (“deep Q-learning”) [6].

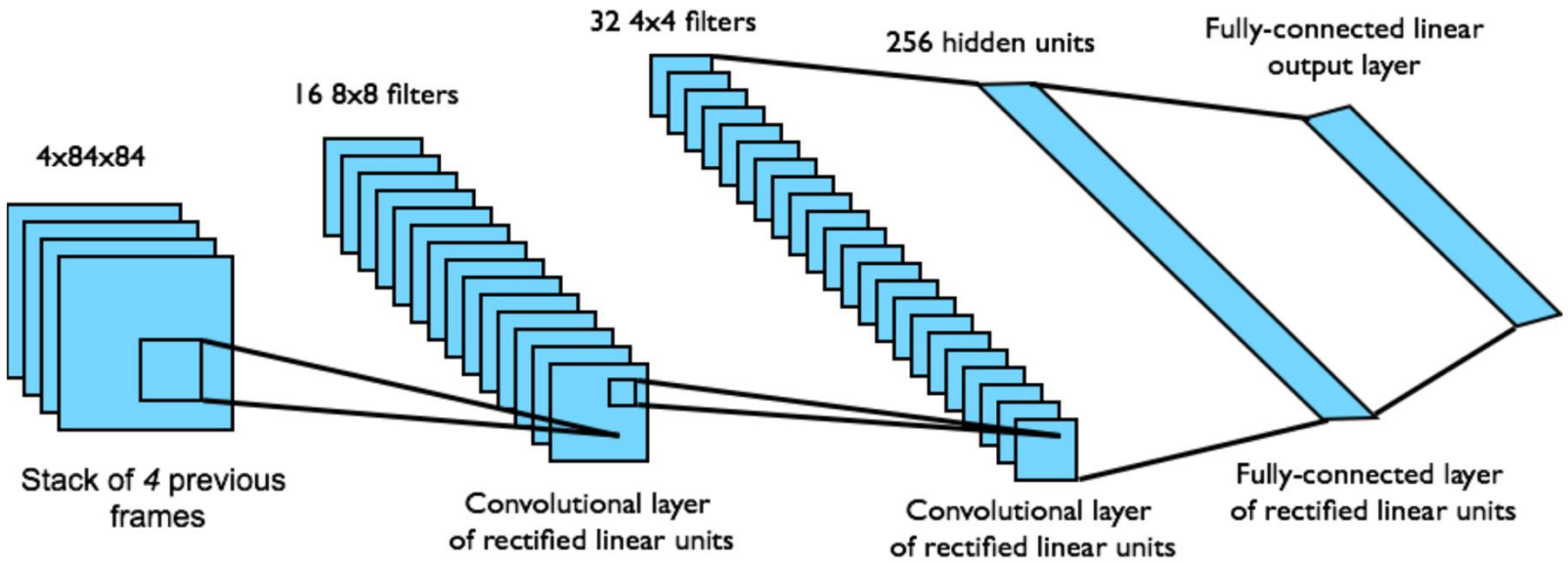


Image Credit: [7]

Deep Q-Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 (“deep Q-learning”) [6].

- Take actions following ϵ-greedy policy

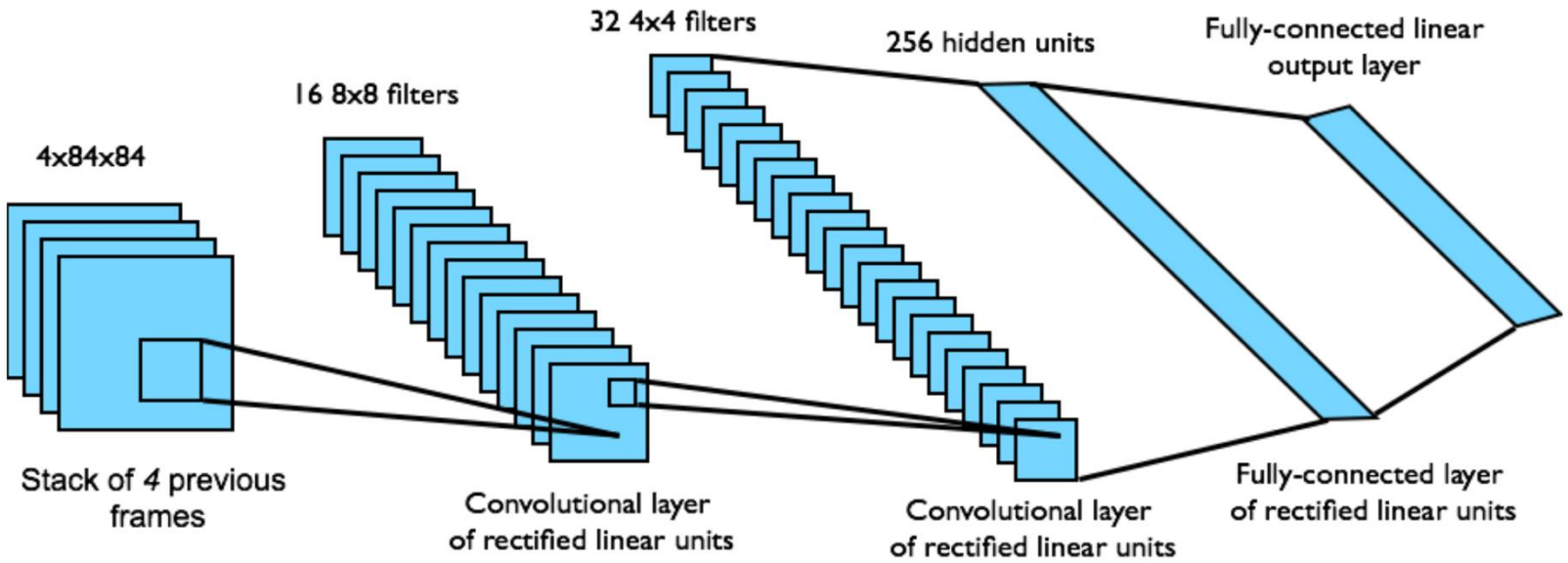


Image Credit: [7]

Deep Q-Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 ("deep Q-learning") [6].

- Take actions following ϵ-greedy policy
- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay buffer* and sample random mini-batch of tuples from the buffer

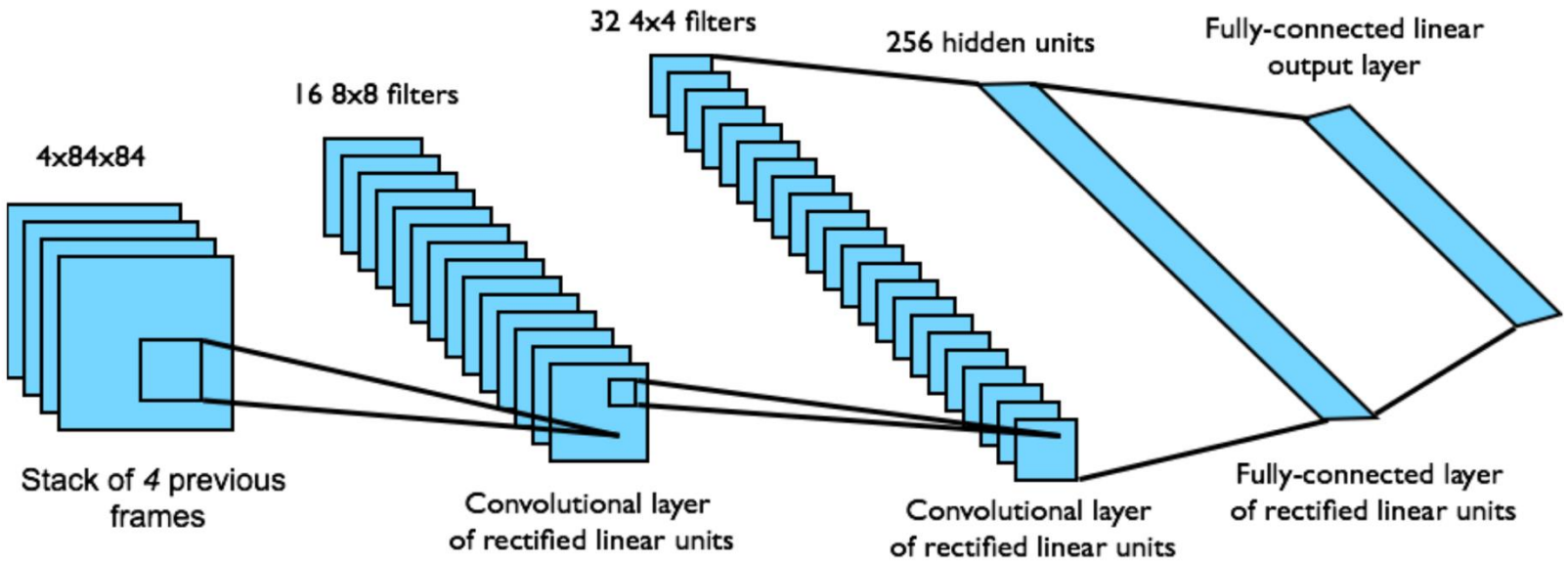


Image Credit: [7]

Deep Q-Learning

Approximating Q function with a neural net is a decades-old idea, but DeepMind got it to work really well on Atari games in 2013 ("deep Q-learning") [6].

- Take actions following ϵ-greedy policy
- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in *replay buffer* and sample random mini-batch of tuples from the buffer
- Compute Q-targets w.r.t. old and fixed parameters $\bar{\theta}$

$$\theta_{t+1} \leftarrow \theta_t + \eta \mathbb{E}_{s_t, a_t, s_{t+1}} \left[\left(\mathcal{R}_{s_t}^{a_t} + \gamma \max_a Q_{\bar{\theta}}(s_{t+1}, a) - Q_{\theta_t}(s_t, a_t) \right) \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta_t} \right]$$

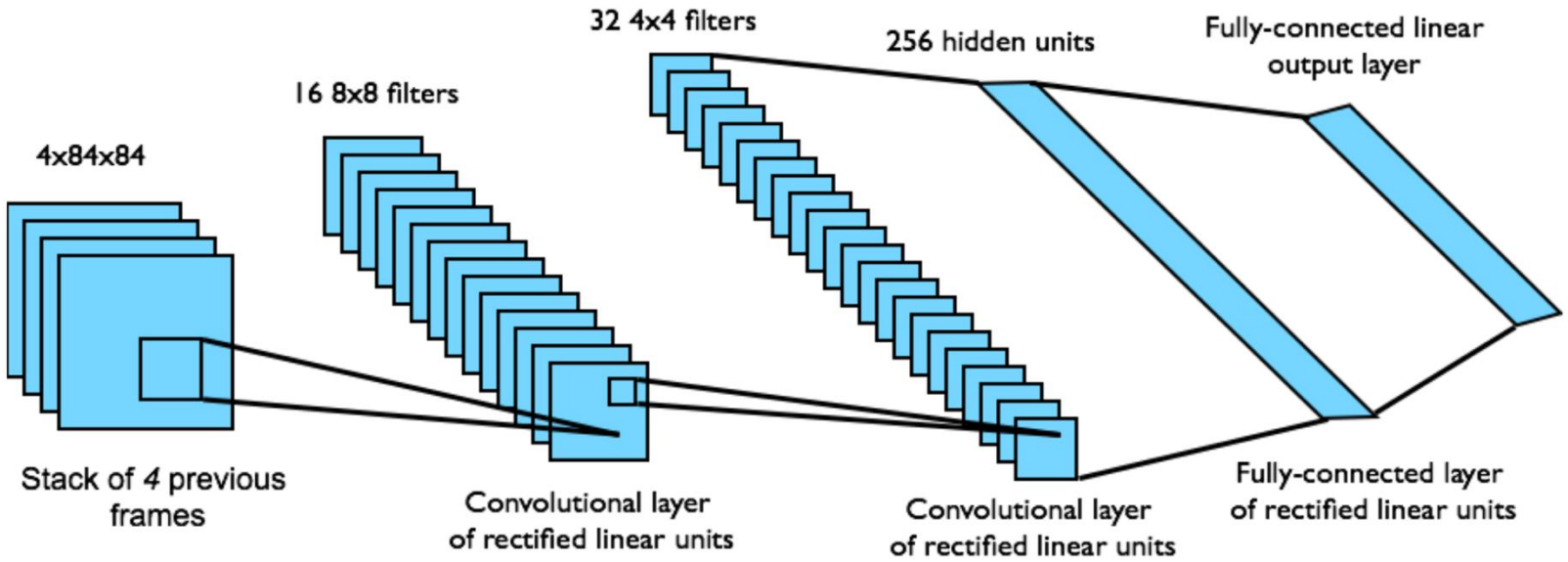


Image Credit: [7]

Outline

- Reinforcement Learning (RL) I
  - Overview & Applications
  - RL Formulation & Taxonomy
  - Markov Decision Process (MDP)
  - Bellman Equations
  - Solving RL problems using the Bellman Equations
- Reinforcement Learning (RL) II
  - Model-Free RL: (Deep) Q-learning
  - **Model-Free RL: Policy Gradient & Actor-Critic**
  - Model-Based RL: Dyna-Q

Policy Gradient Methods

In deep Q learning, we parameterize the Q function as a neural network and learn it to minimize the Bellman error. A policy is then obtained from Q function, e.g., via ϵ-greedy strategy.

Policy Gradient Methods

In deep Q learning, we parameterize the Q function as a neural network and learn it to minimize the Bellman error. A policy is then obtained from Q function, e.g., via ϵ-greedy strategy.

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy

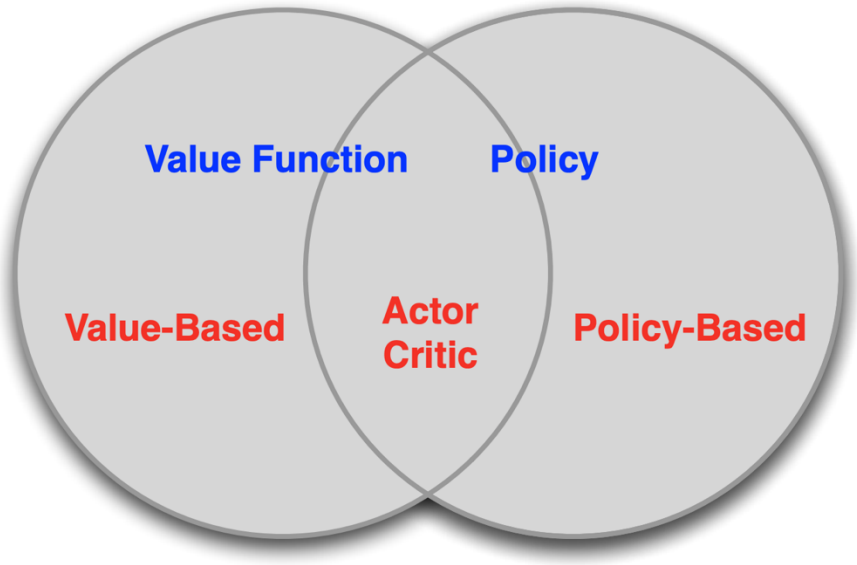


Image Credit: [2]

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Often converge to local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_{\theta}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right]$$

Policy Gradient Methods

In policy based methods, we directly parameterize the policy and learn it to maximize some reward.

Given a trajectory $\boldsymbol{\tau}$, let us consider the simple expected reward:

$$J(\theta) = \mathbb{E}_{\boldsymbol{\tau}}\left[R(\boldsymbol{\tau})\right] = \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\sum_{t=1}^{T} R_t\right] = \mathbb{E}_{\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)}\left[\sum_{t=1}^{T} R_t(A_t, S_t)\right]$$

Log derivative trick:

$$\nabla_\theta J(\theta) = \nabla_\theta \int \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \int \nabla_\theta \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \int \mathbb{P}_\theta(\boldsymbol{\tau}) \nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau}$$

$$= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right]$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right]\end{aligned}$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right]
\end{aligned}
$$

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}
$$

We use the Monte Carlo Approximation. This is known as REINFORCE algorithm [8]!

Policy Gradient Methods

Let us substitute $\mathbb{P}_\theta(\boldsymbol{\tau}) = \mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)$ $\quad R(\boldsymbol{\tau}) = \sum_{t=1}^{T} R_t(A_t,S_t)$

We have

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \mathbb{P}_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log\left(\mathbb{P}_0(S)\prod_{t=1}^{T}\pi_\theta(A_t|S_t)\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta\left(\log \mathbb{P}_0(S) + \sum_{t=1}^{T}\log\pi_\theta(A_t|S_t) + \sum_{t=1}^{T}\log\mathbb{P}(S_{t+1}|S_t,A_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&= \mathbb{E}_{\boldsymbol{\tau}}\left[\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(A_t|S_t)\right)\left(\sum_{t=1}^{T} R_t(A_t,S_t)\right)\right] \\
&\approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=1}^{T} r_t^{(i)}\right)
\end{aligned}$$

Full Reward

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Reward-to-go: *my current policy can not change past rewards (causality)*!

Policy Gradient Methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right) \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

Policy Gradient Methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right) \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

Policy Gradient Methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right) \right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t \right) \right)$

Policy Gradient Methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t\right)\right)$$

This works since

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \pi_\theta(A_t|S_t) b_t\right] &= \iint_{A_t,S_t} p(S_t)\pi_\theta(A_t|S_t)\nabla_\theta \log \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\int_{A_t} \nabla_\theta \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\nabla_\theta \int_{A_t} \pi_\theta(A_t|S_t)\mathrm{d}A_t b_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)(\nabla_\theta 1) b_t \mathrm{d}S_t = 0
\end{aligned}$$

Policy Gradient Methods

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Reward-to-go: *my current action can not change past rewards (causality)*!

Policy gradient typically has a high variance due to the Monte Carlo approximation in high dimension.

To reduce the variance, a common approach is the **control variate** method, *a.k.a.*, **baseline**.

We can compute the average reward as a baseline: $b_t = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t+1}^{T} r_{t'}^{(i)}$

We then subtract the baseline: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} - b_t\right)\right)$$

This works since

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\tau}}\left[\nabla_\theta \log \pi_\theta(A_t|S_t) b_t\right] &= \iint_{A_t,S_t} p(S_t)\pi_\theta(A_t|S_t)\nabla_\theta \log \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\int_{A_t} \nabla_\theta \pi_\theta(A_t|S_t) b_t \mathrm{d}A_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)\nabla_\theta \int_{A_t} \pi_\theta(A_t|S_t)\mathrm{d}A_t b_t \mathrm{d}S_t \\
&= \int_{S_t} p(S_t)(\nabla_\theta 1) b_t \mathrm{d}S_t = 0
\end{aligned}$$

Since the baseline does not depend on the action, subtracting the baseline still leads to an unbiased estimator. By carefully choosing the baseline, we can reduce the variance!

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \operatorname{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \operatorname{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

- Natural Policy Gradient [9]: Preconditioning with Fisher-Information Matrix

  $\mathbf{F} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right]$ $\qquad$ $D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^\top \mathbf{F} (\theta' - \theta)$

  $$\theta' \leftarrow \operatorname{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \leq \epsilon$$

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

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \operatorname{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

- Natural Policy Gradient [9]: Preconditioning with Fisher-Information Matrix

  $\mathbf{F} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right]$ $\qquad$ $D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^\top \mathbf{F} (\theta' - \theta)$

  $$\theta' \leftarrow \operatorname{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \leq \epsilon$$

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

- Trust Region Policy Optimization (TRPO) [10]: select ϵ and solve for the optimal α while solving $\mathbf{F}^{-1} \nabla_\theta J(\theta)$ using conjugate gradient

Advanced Policy Gradient Methods

What is wrong with the Policy Gradient besides high variance?

Similar to other optimization methods, step sizes matter for convergence!

$$\theta' \leftarrow \text{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } \|\theta' - \theta\|^2 \leq \epsilon$$

- Natural Policy Gradient [9]: Preconditioning with Fisher-Information Matrix

  $\mathbf{F} = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^\top\right]$ $\qquad$ $D_{\text{KL}}(\pi_{\theta'} \| \pi_\theta) \approx (\theta' - \theta)^\top \mathbf{F} (\theta' - \theta)$

  $$\theta' \leftarrow \text{argmax}(\theta' - \theta)^\top \nabla_\theta J(\theta) \quad \text{s.t. } D_{\text{KL}}(\pi_{\theta'} \| \pi_\theta) \leq \epsilon$$

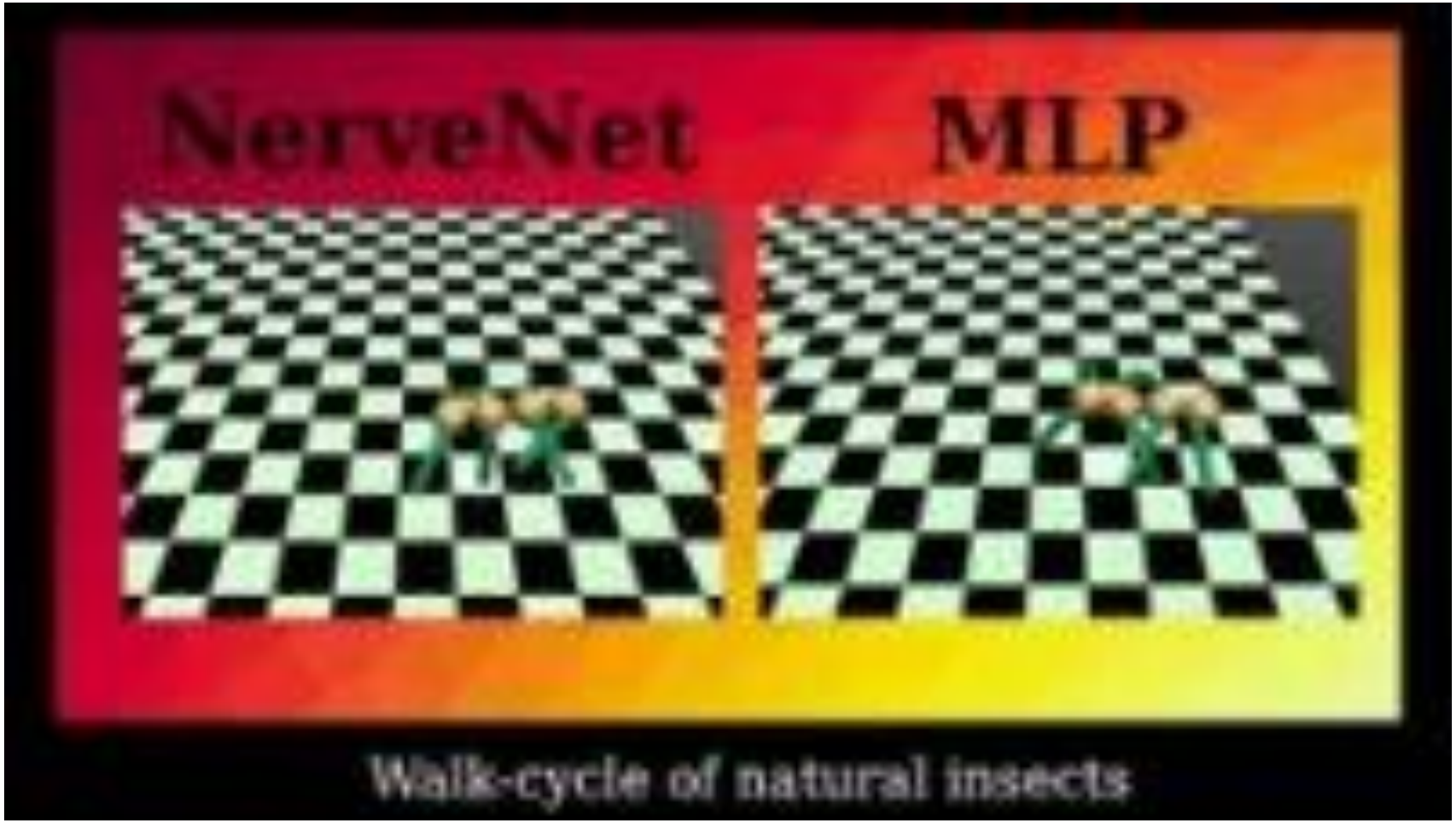
  $$\theta' \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$$

- Trust Region Policy Optimization (TRPO) [10]: select ϵ and solve for the optimal α while solving $\mathbf{F}^{-1} \nabla_\theta J(\theta)$ using conjugate gradient

- Proximal Policy Optimization (PPO) [11]: turn above constrained optimization into an unconstrained one (with clipped loss) and use modern 1st-order optimizers like Adam.

Demo

Simulated Continuous Control in Mujoco using PPO [11] and graph neural networks [12]:



Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)}) \left(\sum_{t'=t+1}^{T} r_{t'}^{(i)} \right) \right)$$

Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

Actor-Critic Methods

Recall policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!

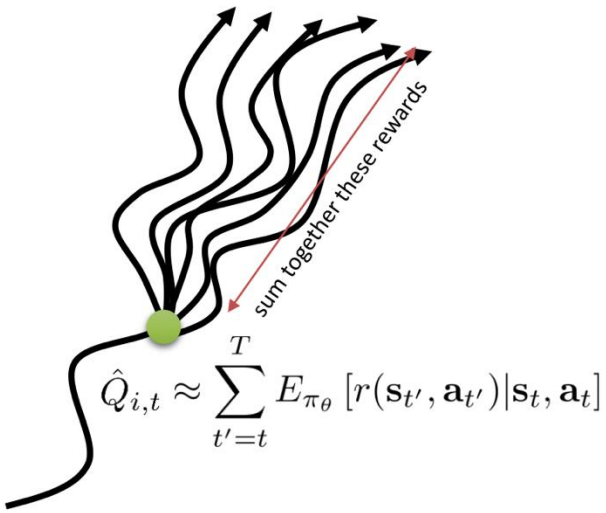


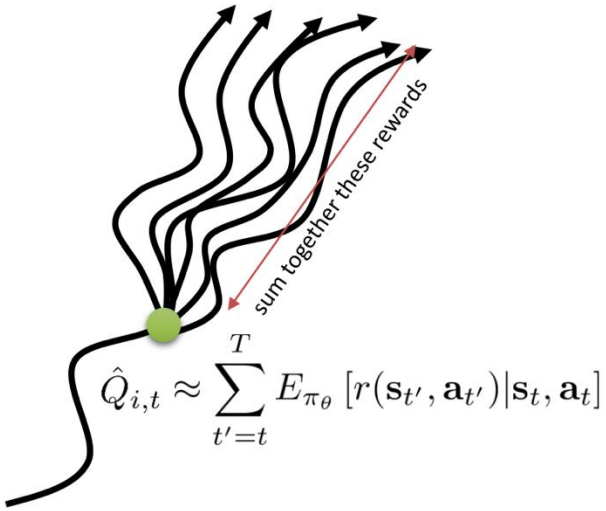
Image Credit: [13]

Actor-Critic Methods

Recall policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!



Following the idea of reducing variance via baselines, we introduce the **advantage**:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

where the value does not depend on the action, thus serving as the baseline.

We can estimate the value based on: $V_\pi(s) = \sum_a Q_\pi(s,a)\pi(a|s)$

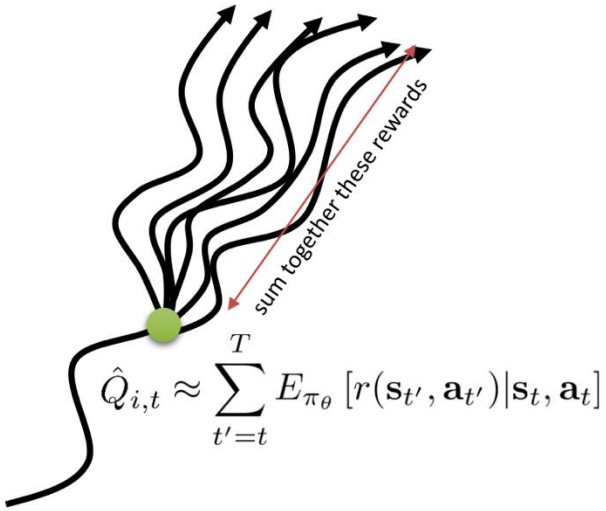
Image Credit: [13]

Actor-Critic Methods

Recall policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{t'=t+1}^{T} r_{t'}^{(i)}\right)\right)$

Since $Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t|S_t = s, A_t = a\right]$, the reward-to-go is a single-sample estimation of Q.

Q is the true expected reward-to-go and we can use more samples to estimate it!



Following the idea of reducing variance via baselines, we introduce the **advantage**:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$$

where the value does not depend on the action, thus serving as the baseline.

We can estimate the value based on: $V_\pi(s) = \sum_a Q_\pi(s,a)\pi(a|s)$

Since subtracting a baseline still gives us an unbiased estimator, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)A_{\pi_\theta}(s_t,a_t)\right]$$

Image Credit: [13]

Actor-Critic Methods

How to estimate the advantage function?

$$A_\pi(s_t, a_t) = Q_\pi(s_t, a_t) - V_\pi(s_t)$$

Actor-Critic Methods

How to estimate the advantage function?

$$
\begin{aligned}
A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\
&= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\
&\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)}
\end{aligned}
$$

Temporal Difference (TD) error!

Based on the TD error, we can reduce it to estimating the value function!

Actor-Critic Methods

How to estimate the advantage function?

$$
\begin{aligned}
A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\
&= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\
&\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)}
\end{aligned}
$$

Temporal Difference (TD) error!

Based on the TD error, we can reduce it to estimating the value function!

How to estimate the value? We can learn a value network to fit the (Monte Carlo) Policy Evaluation!

Actor-Critic Methods

How to estimate the advantage function?

$$
\begin{aligned}
A_\pi(s_t, a_t) &= Q_\pi(s_t, a_t) - V_\pi(s_t) \\
&= \mathbb{E}_{\pi_\theta}[\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) | s_t, a_t] - V_\pi(s_t) \\
&\approx \boxed{\mathcal{R}_{s_t}^{a_t} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)}
\end{aligned}
$$

Temporal Difference (TD) error!

Based on the TD error, we can reduce it to estimating the value function!

How to estimate the value? We can learn a value network to fit the (Monte Carlo) Policy Evaluation!

The target can be estimated from rollout:

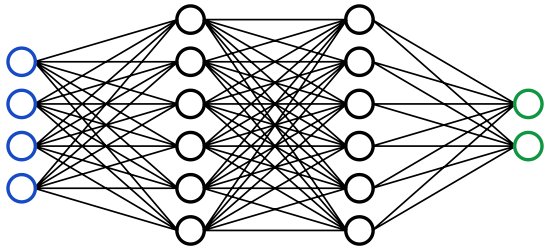
- Vanilla $\bar{V}(s_t^{(i)}) \approx \sum_{t'=t}^{T} r_{t'}^{(i)}$
- Bootstrap $\bar{V}(s_t^{(i)}) \approx r_t^{(i)} + \gamma V_\phi(s_{t+1}^{(i)})$

Actor-Critic Methods

Given the target, we can learn a value network (critic) by minimizing:

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| V_\phi(s_t^{(i)}) - \bar{V}(s_t^{(i)}) \right\|^2$$

Critic = Value $V_\phi(s_t)$

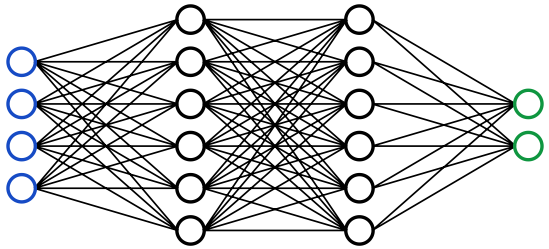


Actor-Critic Methods

Given the target, we can learn a value network (critic) by minimizing:

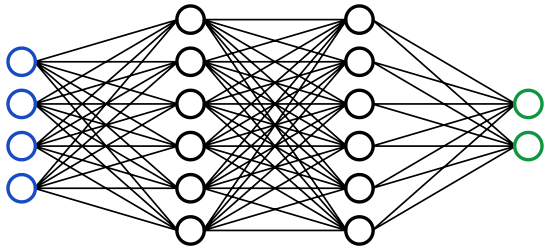
$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| V_\phi(s_t^{(i)}) - \bar{V}(s_t^{(i)}) \right\|^2$$

Critic = Value $V_\phi(s_t)$



On the other hand, we learn a policy network (actor) by following policy gradient!

Actor = Policy $\pi_\theta(a_t|s_t)$

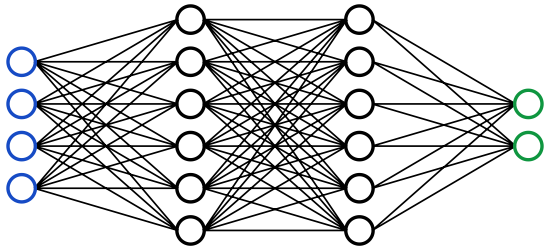


Actor-Critic Methods

Given the target, we can learn a value network (critic) by minimizing:

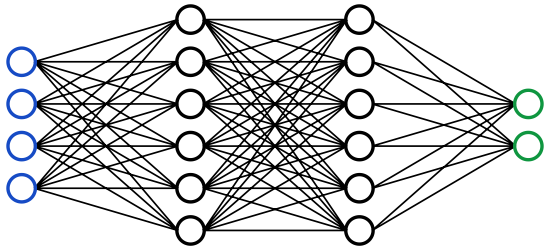
$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\| V_\phi(s_t^{(i)}) - \bar{V}(s_t^{(i)}) \right\|^2$$

Critic = Value $V_\phi(s_t)$



On the other hand, we learn a policy network (actor) by following policy gradient!

Actor = Policy $\pi_\theta(a_t|s_t)$



Algorithm 1 (Batch) Actor-Critic Algorithm

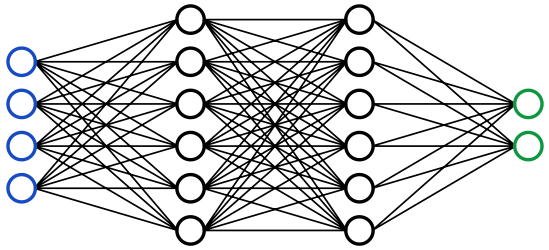
Input: Initialize θ and ϕ
Repeat
  Sample $\{\mathbf{a}^{(i)}, \mathbf{s}^{(i)}\} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$
  Update ϕ by minimizing $\frac{1}{2}\sum_i \left\| V_\phi(\mathbf{s}^{(i)}) - \bar{V}(\mathbf{s}^{(i)}) \right\|^2$
  $\hat{A}_{\pi_\theta}(\mathbf{s}^{(i)}, \mathbf{a}^{(i)}) = \mathcal{R}_{\mathbf{s}^{(i)}}^{\mathbf{a}^{(i)}} + \gamma V_\pi(\mathbf{s}^{(i)\prime}) - V_\pi(\mathbf{s}^{(i)})$
  $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^{(i)}|\mathbf{s}^{(i)}) \hat{A}_{\pi_\theta}(\mathbf{s}^{(i)}, \mathbf{a}^{(i)})$
  $\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$
Until J converges
Return Final θ

Actor-Critic Methods

Given the target, we can learn a value network (critic) by minimizing:

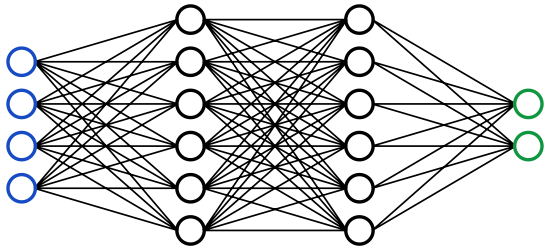
$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\| V_\phi(s_t^{(i)}) - \bar{V}(s_t^{(i)}) \right\|^2$$

Critic = Value $V_\phi(s_t)$



On the other hand, we learn a policy network (actor) by following policy gradient!

Actor = Policy $\pi_\theta(a_t|s_t)$



Algorithm 1 (Batch) Actor-Critic Algorithm

1: **Input**: Initialize θ and ϕ
2: **Repeat**
3: Sample $\{\mathbf{a}^{(i)}, \mathbf{s}^{(i)}\} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$
4: Update ϕ by minimizing $\frac{1}{2}\sum_i \left\| V_\phi(\mathbf{s}^{(i)}) - \bar{V}(\mathbf{s}^{(i)}) \right\|^2$
5: $\hat{A}_{\pi_\theta}(\mathbf{s}^{(i)}, \mathbf{a}^{(i)}) = \mathcal{R}_{\mathbf{s}^{(i)}}^{\mathbf{a}^{(i)}} + \gamma V_\pi(\mathbf{s}^{(i)\prime}) - V_\pi(\mathbf{s}^{(i)})$
6: $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}^{(i)}|\mathbf{s}^{(i)}) \hat{A}_{\pi_\theta}(\mathbf{s}^{(i)}, \mathbf{a}^{(i)})$
7: $\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla_\theta J(\theta)$
8: **Until** J converges
9: **Return** Final θ

Episodic trajectories

Outline

- Reinforcement Learning (RL) I
  - Overview & Applications
  - RL Formulation & Taxonomy
  - Markov Decision Process (MDP)
  - Bellman Equations
  - Solving RL problems using the Bellman Equations
- Reinforcement Learning (RL) II
  - Model-Free RL: (Deep) Q-learning
  - Model-Free RL: Policy Gradient & Actor-Critic
  - **Model-Based RL: Dyna-Q**

Model-Based RL

Advantages:

- Can efficiently learn models via supervised learning
- Can reason about model uncertainty

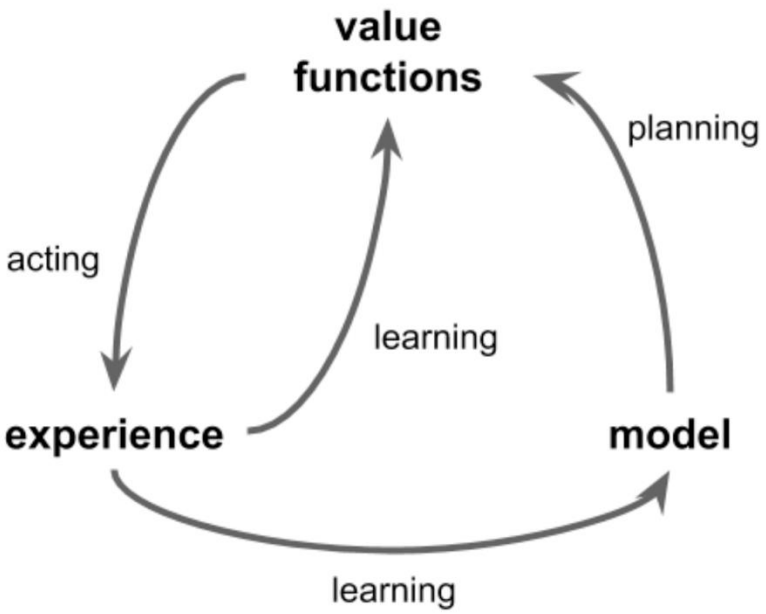


Image Credit: [3]

Model-Based RL

Advantages:

- Can efficiently learn models via supervised learning
- Can reason about model uncertainty

Disadvantages:

- First learn a model and then do planning
  → Two sources of approximation error
- Sometimes learning a model is harder than solving the tasks at hand

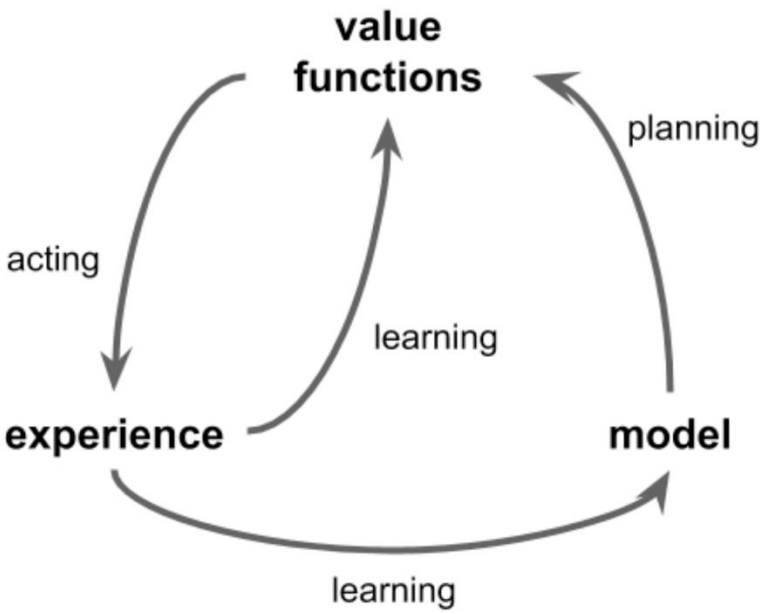


Image Credit: [3]

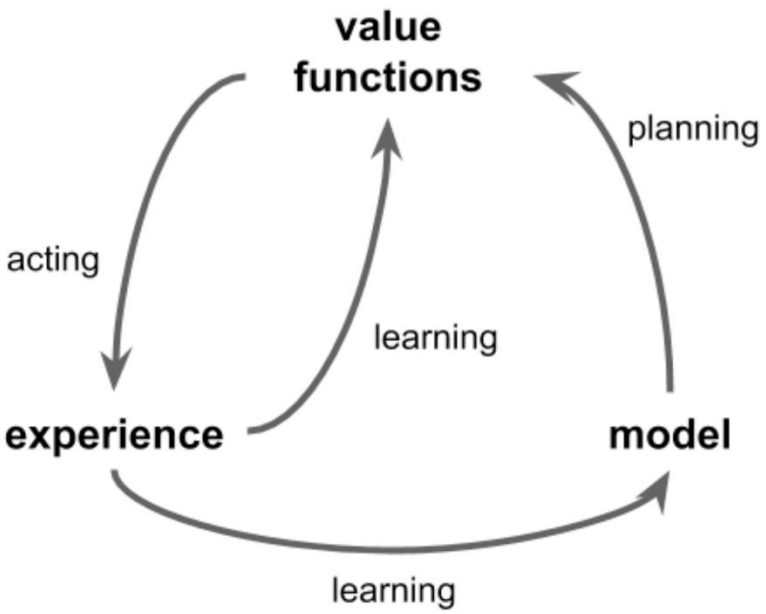
Model-Based RL

Advantages:

- Can efficiently learn models via supervised learning
- Can reason about model uncertainty

Disadvantages:

- First learn a model and then do planning
  → Two sources of approximation error
- Sometimes learning a model is harder than solving the tasks at hand



Recall what a model is, e.g., an MDP that represents how state transits and what the next reward should be!

Image Credit: [3]

Dyna-Q

Dyna-Q [14] combines model-free and model-based RL:

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience

Dyna-Q

Dyna-Q [14] combines model-free and model-based RL:

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience

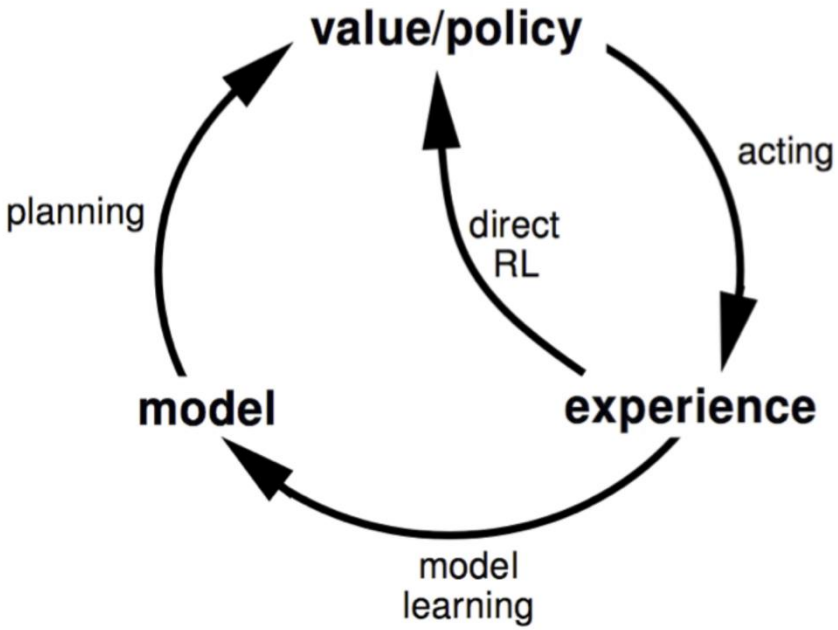
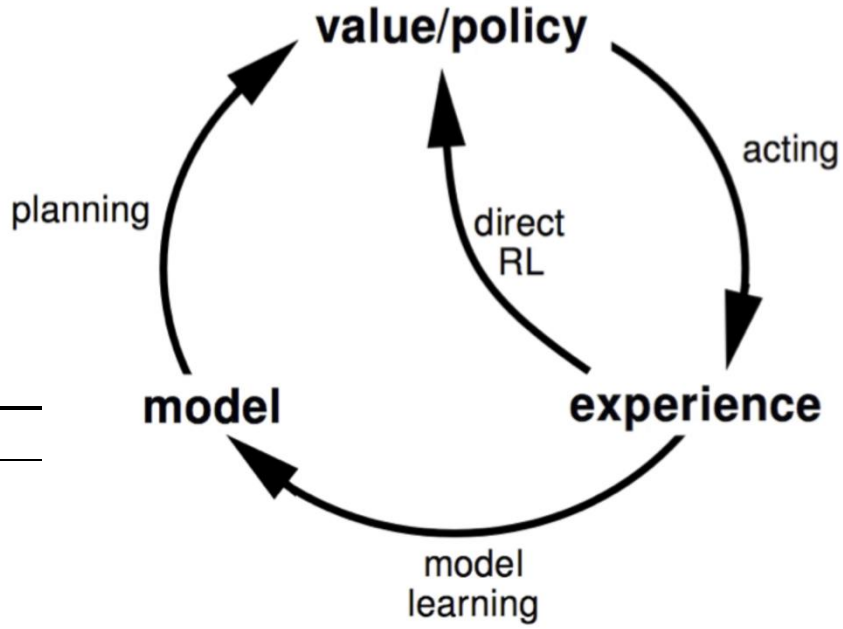


Image Credit: [15]

Dyna-Q

Dyna-Q [14] combines model-free and model-based RL:

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience



Algorithm 2 Dyna-Q Algorithm

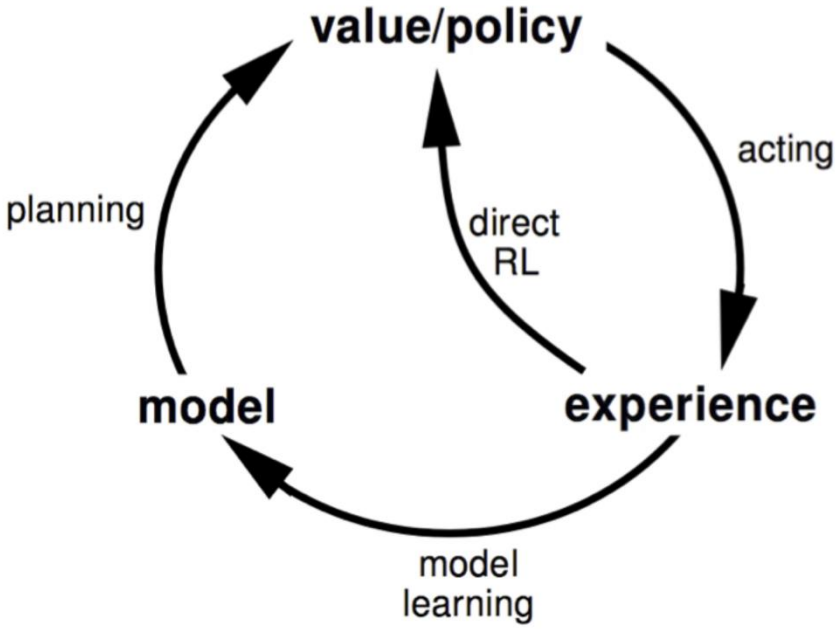
Input: Initialize $Q(s,a)$ and Model(s,a) for all s and a
Do forever
  $s \leftarrow$ current (nonterminal) state
  $a \leftarrow \epsilon$-greedy(s,Q)
  Execute action a, observe reward r and state s'
  $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
  Model$(s,a) \leftarrow r, s'$
  Repeat n times
    $s \leftarrow$ random previously observed state
    $a \leftarrow$ random action previously taken in state s
    $r, s' \leftarrow$ Model(s,a)
    $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

Image Credit: [15]

Dyna-Q

Dyna-Q [14] combines model-free and model-based RL:

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience



Algorithm 2 Dyna-Q Algorithm

Input: Initialize $Q(s,a)$ and Model(s,a) for all s and a
Do forever
  $s \leftarrow$ current (nonterminal) state
  $a \leftarrow \epsilon$-greedy(s,Q)
  Execute action a, observe reward r and state s'
  $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
  Model$(s,a) \leftarrow r, s'$
  Repeat n times
    $s \leftarrow$ random previously observed state
    $a \leftarrow$ random action previously taken in state s
    $r, s' \leftarrow$ Model(s,a)
    $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

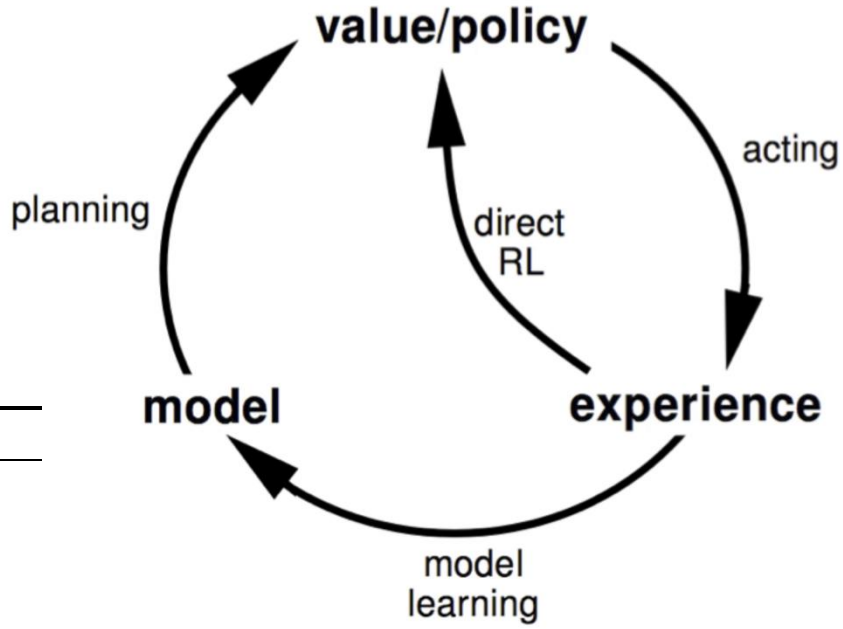
Supervised learning from real experience!

Image Credit: [15]

Dyna-Q

Dyna-Q [14] combines model-free and model-based RL:

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience



Algorithm 2 Dyna-Q Algorithm

Input: Initialize $Q(s,a)$ and Model(s,a) for all s and a
Do forever
  $s \leftarrow$ current (nonterminal) state
  $a \leftarrow \epsilon$-greedy(s,Q)
  Execute action a, observe reward r and state s'
  $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
  Model$(s,a) \leftarrow r, s'$
  Repeat n times
    $s \leftarrow$ random previously observed state
    $a \leftarrow$ random action previously taken in state s
    $r, s' \leftarrow$ Model(s,a)
    $Q(s,a) \leftarrow Q(s,a) + \eta\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

Supervised learning from real experience!

Learning from simulated experience (imagination)!

Image Credit: [15]

References

[1] Murphy, K.P., 2023. Probabilistic machine learning: Advanced topics. MIT Press.

[2] https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

[3] https://github.com/yjavaherian/deepmind-x-ucl-rl/blob/main/slides/All%20Lectures.pdf

[4] Watkins, C.J. and Dayan, P., 1992. Q-learning. Machine learning, 8, pp.279-292.

[5] Sutton, R.S. and Barto, A.G., 1998. Reinforcement Learning: An Introduction.

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

[7] https://www.davidsilver.uk/wp-content/uploads/2020/03/FA.pdf

[8] Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement learning, pp.5-32.

[9] Peters, J. and Schaal, S., 2008. Reinforcement learning of motor skills with policy gradients. Neural networks, 21(4), pp.682-697.

[10] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

[11] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

References

[12] Wang, T., Liao, R., Ba, J. and Fidler, S., 2018. Nervenet: Learning structured policy with graph neural networks. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada (Vol. 30).

[13] https://rail.eecs.berkeley.edu/deeprlcourse/deeprlcourse/static/slides/lec-6.pdf

[14] Sutton, R.S., 1990, June. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming. In Proceedings of the seventh international conference (1990) on Machine learning (pp. 216-224).

[15] https://www.davidsilver.uk/wp-content/uploads/2020/03/dyna.pdf

Questions?