Data-Driven Modeling *MSDM 5055*

Deep Learning for Modeling: Concepts, Tools, and Techniques

Week 11: Reinforcement Learning Part II: Policy-based Reinforcement Learning

Li Shuo-Hui

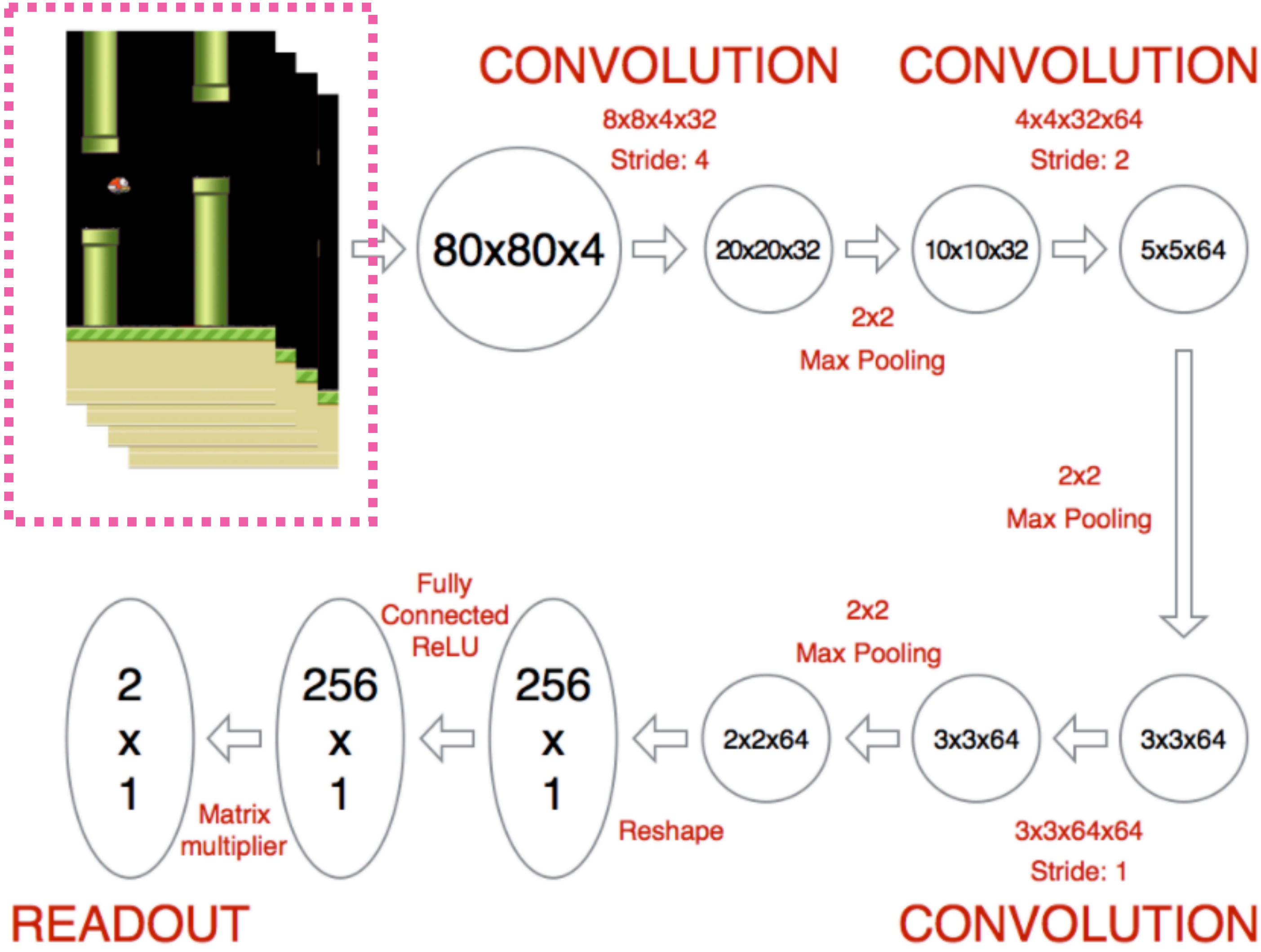
@HKUST | Spring 2023

Value-based Reinforcement Learning

Problems with value function

Value functions are hard to fit

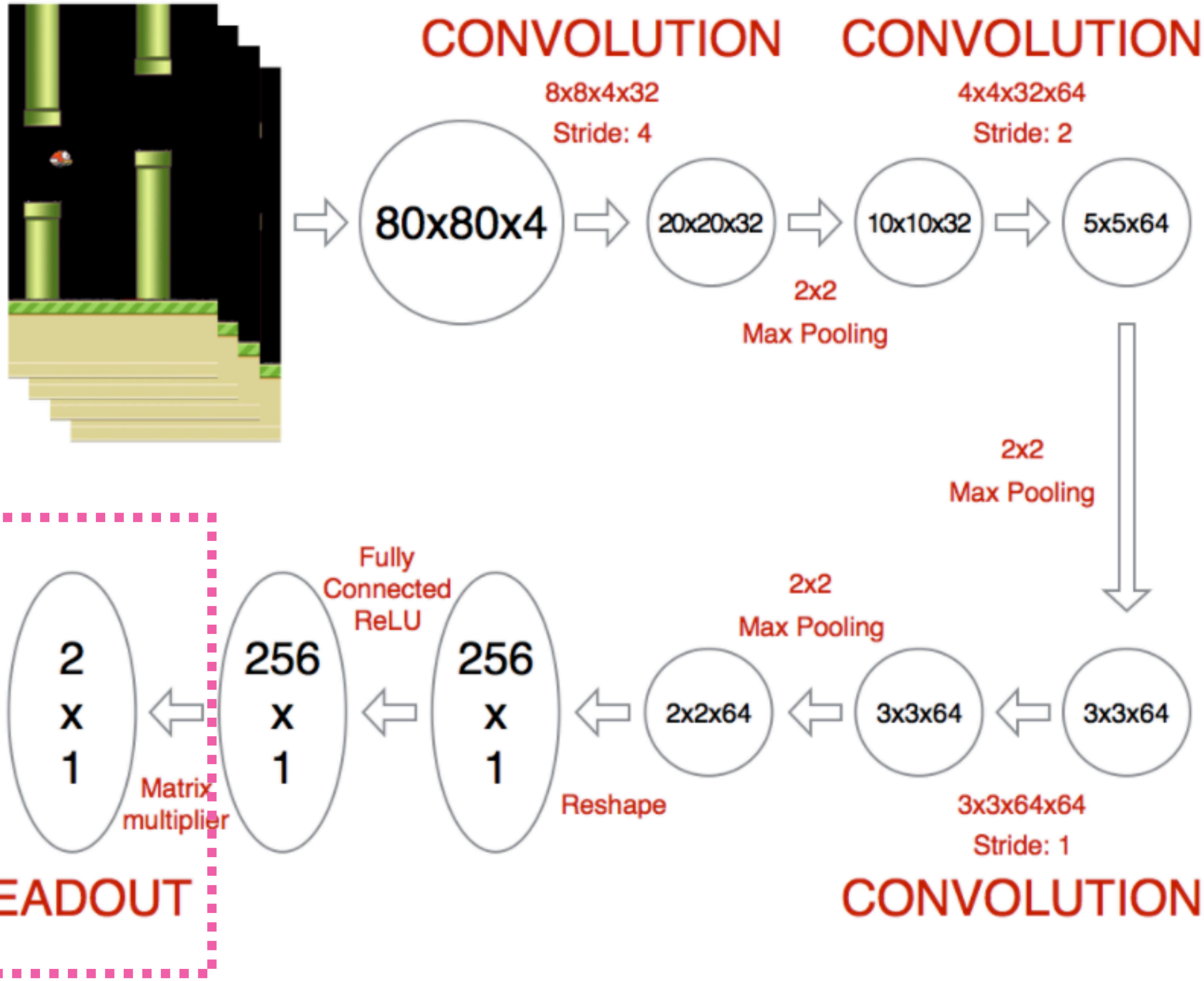
High-dimensional state



Value-based Reinforcement Learning

Problems with value function

Actions must be discrete



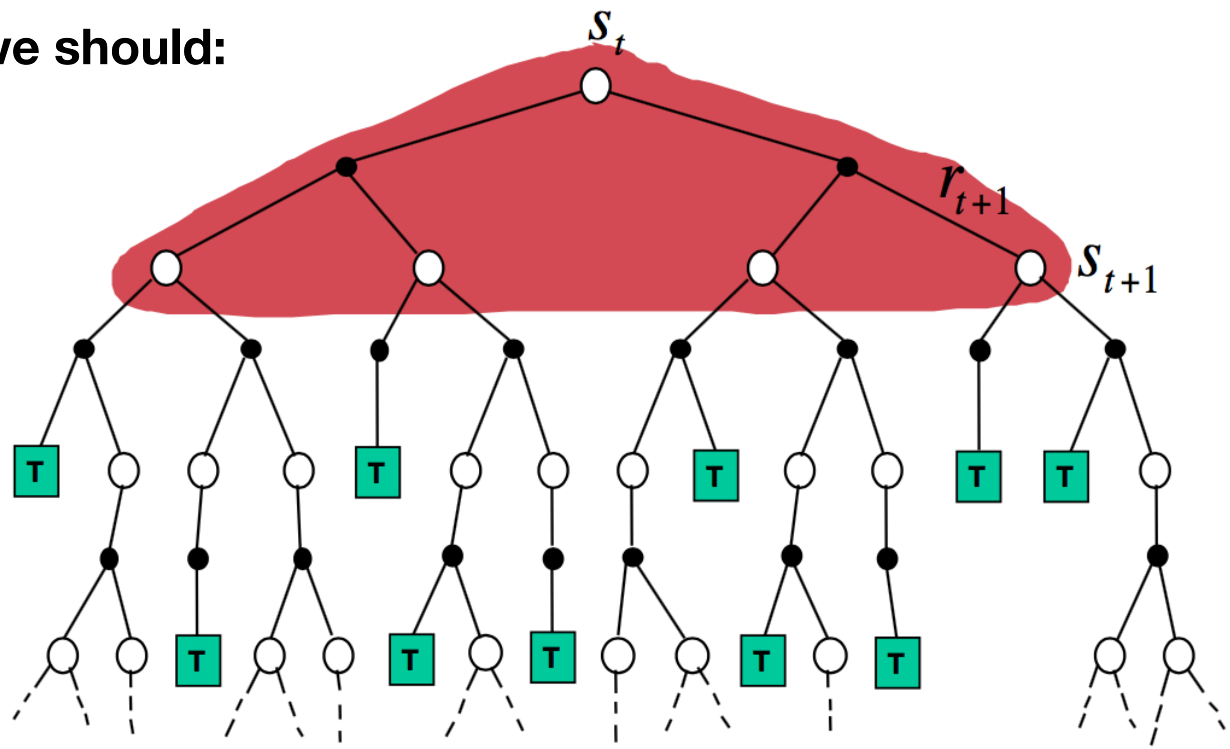
Actions are specified by indexing, thus must be discrete.

Value-based Reinforcement Learning

Problems with value function

Dynamic program approaches are hard

Ideally, we should:



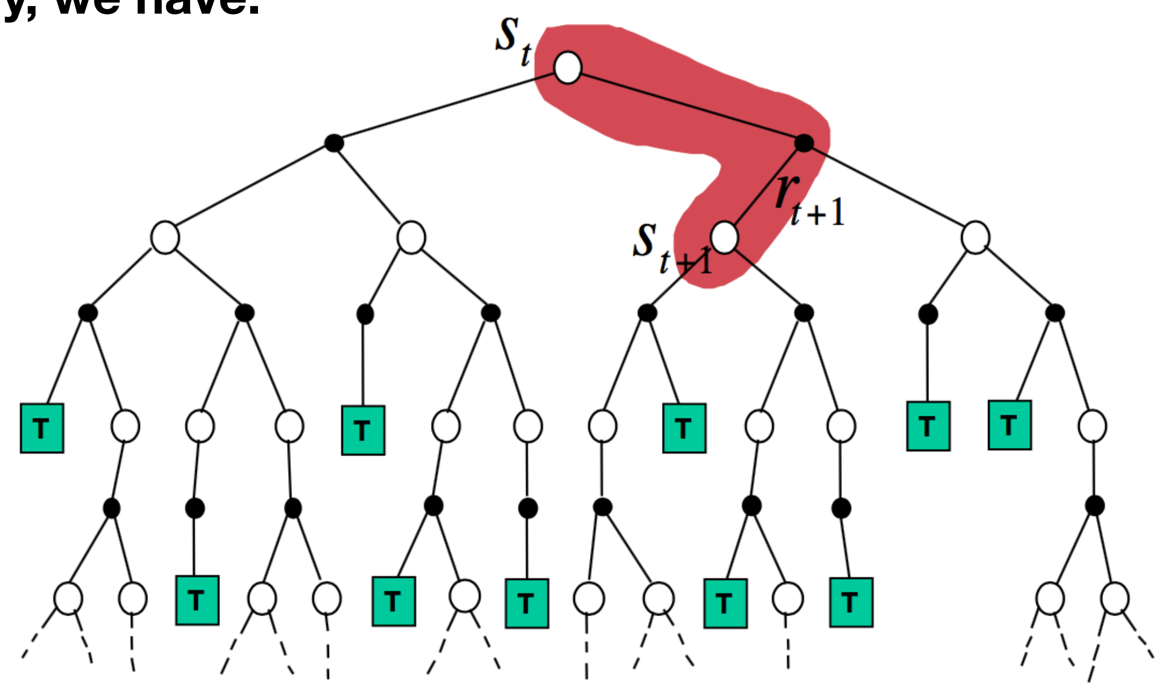
$$Q_\pi(\mathbf{s}_t, a) = \mathbb{E}\left[R_t + \gamma Q_\pi(\mathbf{s}_{t+1}, a_{t+1})\right]$$

Value-based Reinforcement Learning

Problems with value function

Dynamic program approaches are hard

But, practically, we have:



Temporal difference (TD) method

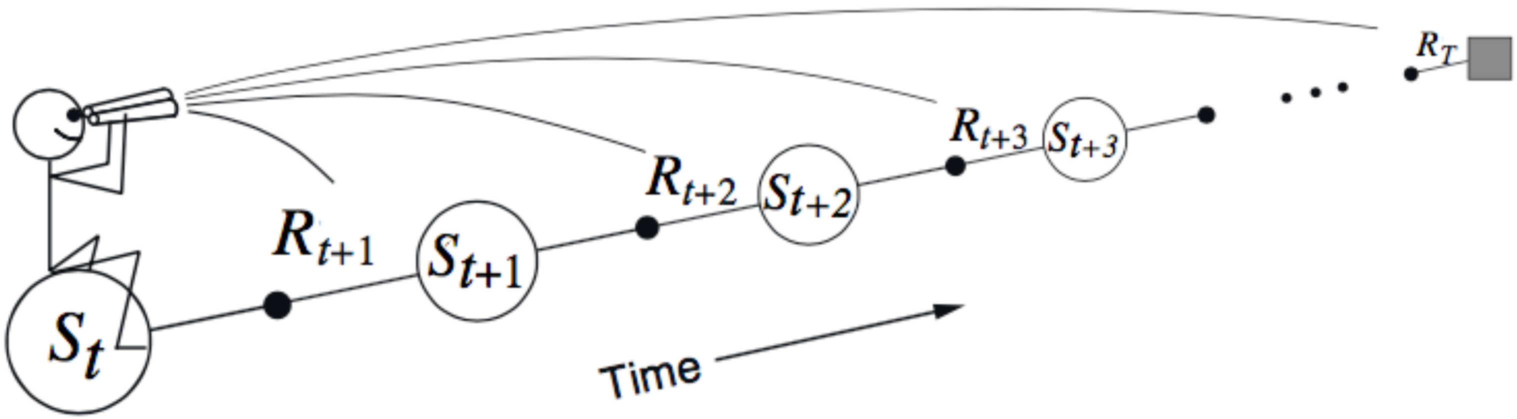
$$Q_\pi(\mathbf{s}_t, a) = \mathbb{E}\left[R_t + \gamma \max_{a' \in \mathcal{A}} Q_\pi(\mathbf{s}_{t+1}, a')\right]$$

Value-based Reinforcement Learning

Problems with value function

Dynamic program approaches are hard

How to approximate ideal DP



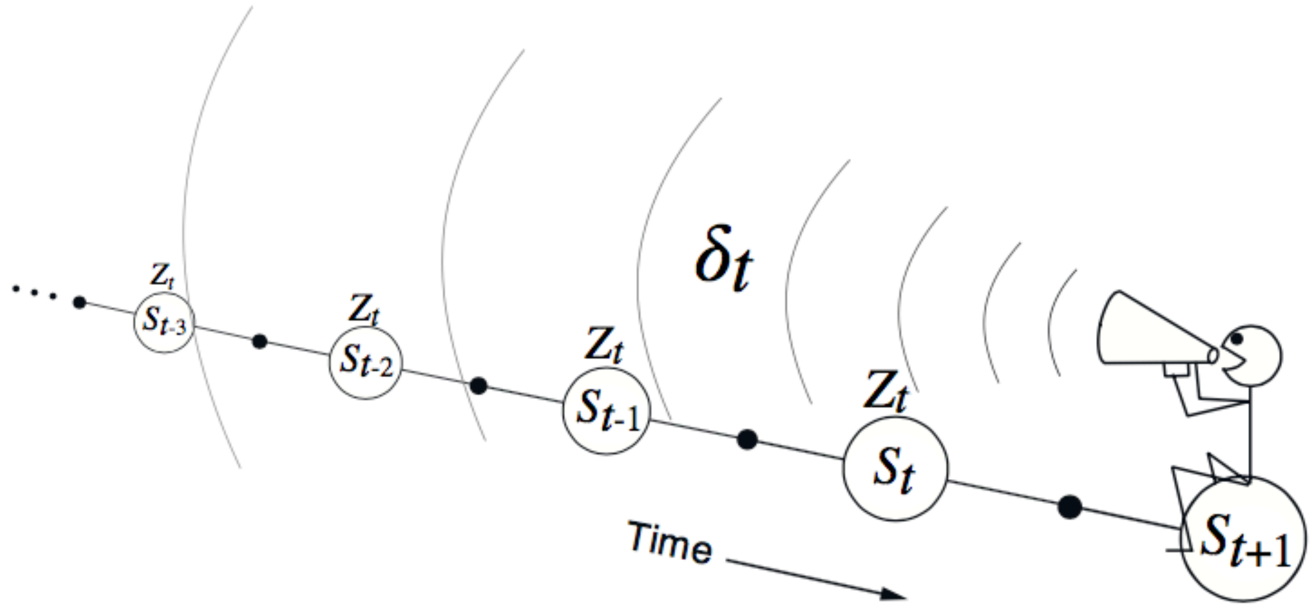
Forward-view TD(λ)

Value-based Reinforcement Learning

Problems with value function

Dynamic program approaches are hard

How to approximate ideal DP



Backward-View TD(λ)

Value-based Reinforcement Learning

Problems with value function

Policy will not stable before value function converges

- Planning/policy is not reliable before value function converges. May pose an obstacle for online learning, e.g., autopilot car/airplane, where wrong doings are costly.

Policy Gradient

Intuition

Directly optimizing the policy to maximizing the accumulated reward.

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t$$

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t$$

$$= \sum_{s_t,a_t,r_t} \sum_t p(s_t) \boxed{p_\theta(a_t|s_t)} p(r_t|a_t,s_t) \gamma^t r_t$$

Policy network
Output the action pdf given the state

Policy Gradient

Intuition

Directly optimizing the policy to maximizing the accumulated reward.

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t$$

$$\begin{aligned} J(\pi_\theta) &= \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t \\ &= \sum_{s_t, a_t, r_t} \sum_t p(s_t) p_\theta(a_t|s_t) p(r_t|a_t, s_t) \gamma^t r_t \end{aligned}$$

Environment
We have no control over these probabilities

Policy Gradient

Gradient estimation

To optimize, one should compute the gradient of the accumulate reward function

$$J(\pi_\theta) = \underset{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>}{\mathbb{E}} \sum_t \gamma^t r_t$$

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \frac{\partial}{\partial \theta} \underset{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>}{\mathbb{E}} \sum_t \gamma^t r_t$$

$$= \frac{\partial}{\partial \theta} \sum_{s_t, a_t, r_t} \sum_t p(s_t) p_\theta(a_t | s_t) p(r_t | a_t, s_t) \gamma^t r_t$$

$$= \sum_{s_t, a_t, r_t} \sum_t p(s_t) \frac{\partial}{\partial \theta} p_\theta(a_t | s_t) p(r_t | a_t, s_t) \gamma^t r_t$$

Policy Gradient

Gradient estimation

This is similar problem where we have solved for generative models.

$$\frac{\partial}{\partial\theta}J(\pi_\theta) = \frac{\partial}{\partial\theta}\mathbb{E}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>}\sum_t \gamma^t r_t$$

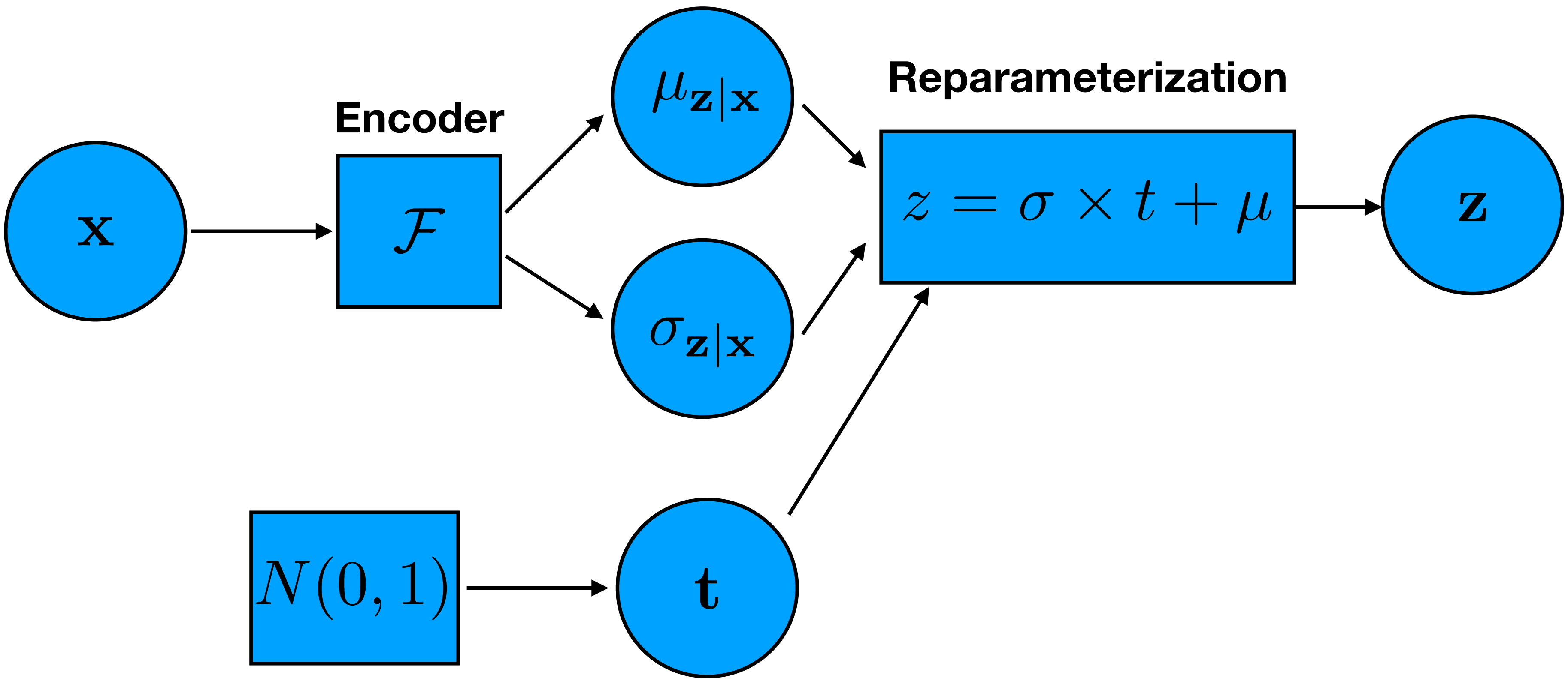
$$= \frac{\partial}{\partial\theta}\sum_{s_t,a_t,r_t}\sum_t p(s_t)p_\theta(a_t|s_t)p(r_t|a_t,s_t)\gamma^t r_t$$

$$= \sum_{s_t,a_t,r_t}\sum_t p(s_t)\boxed{\frac{\partial}{\partial\theta}p_\theta(a_t|s_t)}p(r_t|a_t,s_t)\gamma^t r_t$$

Policy Gradient

Gradient estimation

Reparameterization trick: Convert the estimation into a operation



Only works on a few distributions, e.g., Gaussian;

Policy Gradient

REINFORCE method

For policy, we need a more flexible probability, which means reparameterization will not work.

$$\sum \frac{\partial}{\partial \theta} p_\theta = \sum p_\theta \frac{1}{p_\theta} \frac{\partial p_\theta}{\partial \theta} = \sum p_\theta \frac{\partial}{\partial \theta} \log p_\theta = \mathbb{E} \frac{\partial}{\partial \theta} \log p_\theta$$

Policy Gradient

REINFORCE method

$$
\begin{aligned}
J(\pi_\theta) &= \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t \\
&= \sum_{s_t,a_t,r_t} \sum_t p(s_t) p_\theta(a_t|s_t) p(r_t|a_t,s_t) \gamma^t r_t \\
&= \mathop{\mathbb{E}}_{s_t,a_t,r_t} \sum_t \gamma^t r_t
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial\theta} J(\pi_\theta) &= \frac{\partial}{\partial\theta} \mathop{\mathbb{E}}_{<\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R}>} \sum_t \gamma^t r_t \\
&= \sum_{s_t,a_t,r_t} \sum_t p(s_t) \frac{\partial}{\partial\theta} p_\theta(a_t|s_t) p(r_t|a_t,s_t) \gamma^t r_t \\
&= \mathop{\mathbb{E}}_{s_t,a_t,r_t} \sum_t \gamma^t r_t \cdot \frac{\partial}{\partial\theta} \log p_\theta(a_t|s_t)
\end{aligned}
$$

Policy Gradient

Comparison

Reparameterization

Pros:
- Accurate;
- Convenient for most models.

Cons:
- Only works on a few distributions, e.g., Gaussian;
- Wouldn't work when we don't know the distribution type.

REINFORCE

Cons:
- Not accurate, gradient has high variations;

Pros:
- Works on all distributions;
- Works even when we don't know the distribution type.

Policy Gradient

Put things together

1. **Define a policy network:** output/sample action pdf when input a state;

$$\pi_\theta(a|s) = p_\theta(a|s)$$

2. **Estimate the accumulate reward**

$$J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \gamma^t r_t$$

3. **Estimate the gradient of accumulate reward**

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \gamma^t r_t \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t|s_t)$$

3. **Gradient descent**

$$\theta \leftarrow \theta + \lambda \frac{\partial}{\partial \theta} J(\pi_\theta)$$

Vanilla Policy Gradient Reinforcement Learning

The score function problem

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \boxed{\gamma^t r_t} \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t | s_t)$$

Advantage function:
"Score" of the action at time t
Higher score means a good action;
Lower score means a bad action.

Different advantage function gives different accumulate reward J function

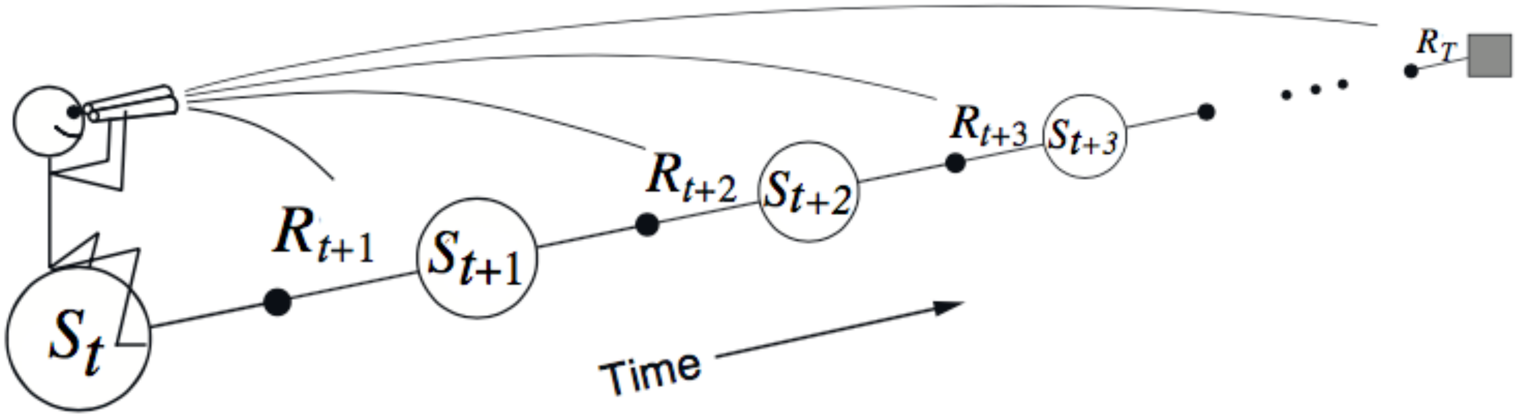
It turns out the discounted reward is not a good advantage function.

Vanilla Policy Gradient Reinforcement Learning

The score function problem

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \boxed{A_t} \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t | s_t)$$

Advantage function:
"Score" of the action at time t
Higher score means a good action;
Lower score means a bad action.



$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$$

Vanilla Policy Gradient Reinforcement Learning

The baseline problem

$$\frac{\partial}{\partial\theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \cdot \frac{\partial}{\partial\theta} \log p_\theta(a_t | s_t)$$

Consider a “good” state s, all action from this s will defaultly carry a high accumulate reward

Compare to the “base” reward of the state s, the difference of choosing actions will not be obvious. Thus, this pose an obstacle for finding the best policy.

We need a way to evaluate and substrate this baseline of the states

This also explain why it’s called an advantage function

Vanilla Policy Gradient Reinforcement Learning

The baseline problem

$$\frac{\partial}{\partial\theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \cdot \frac{\partial}{\partial\theta} \log p_\theta(a_t|s_t)$$

$$\frac{\partial}{\partial\theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \cdot \frac{\partial}{\partial\theta} \log p_\theta(a_t|s_t)$$

One simple baseline: the mean value of rewards received at this state

Vanilla Policy Gradient Reinforcement Learning

Put things together

1. **Define a policy network:** output/sample action pdf when input a state;

$$\pi_\theta(a|s) = p_\theta(a|s)$$

2. **Sample MDP trajectories**

3. **Estimate the gradient**

$$\frac{\partial}{\partial\theta}J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t,a_t,r_t}\sum_t\left(\sum_{t'\geq t}\gamma^{t'-t}r_{t'} - b(s_t)\right)\cdot\frac{\partial}{\partial\theta}\log p_\theta(a_t|s_t)$$

4. **Gradient descent and update the mean value**

$$\theta \leftarrow \theta + \lambda\frac{\partial}{\partial\theta}J(\pi_\theta)$$

Actor-critic Reinforcement Learning

Revisit: the value functions

$$V_\pi(\mathbf{s}_t) = \mathbb{E}\left[R_t + \gamma V_\pi(\mathbf{s}_{t+1})\right]$$

$$Q_\pi(\mathbf{s}_t, a) = \mathbb{E}\left[R_t + \gamma Q_\pi(\mathbf{s}_{t+1}, a_{t+1})\right]$$

V gives estimation of how “good” a state is;
Q gives estimation of how “good” an action is at state s.

They gives the advantage function.

$$A_t = Q_\pi(\mathbf{s}_t, a_t) - V_\pi(\mathbf{s}_t)$$

Actor-critic Reinforcement Learning

Q Actor Critic

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t Q_\pi(\mathbf{s}_t, a_t) \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t | s_t)$$

w. r. t. $\pi_\theta(a|s) = p_\theta(a|s)$

Actor network:

The policy network, when given a state s, it output the pdf of the actions.

It's optimized by policy gradient with the advantage function defined by the critic network (Q learning)

Critic network:

The Q value network, when given a state and an action, it output the estimate score of the pair.

It' optimized by the Q-learning algorithm (via a ground truth given by Bellman equation)

Actor-critic Reinforcement Learning

Put things together: Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
Then sample the next action $a' \sim \pi_\theta(a'|s')$
Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
and use it to update the parameters of Q function:
$w \leftarrow w + \alpha_w \frac{\partial}{\partial w} \delta_t$
Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Actor-critic Reinforcement Learning

Put things together: Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \frac{\partial}{\partial w}\delta_t$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Initialization

Actor-critic Reinforcement Learning

Put things together: Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \frac{\partial}{\partial w} \delta_t$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Sample MDP trajectories

Actor-critic Reinforcement Learning

Put things together: Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \frac{\partial}{\partial w} \delta_t$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Update the actor or the policy network

Actor-critic Reinforcement Learning

Put things together: Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.

for $t = 1 \ldots T$: **do**

Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$

Then sample the next action $a' \sim \pi_\theta(a'|s')$

Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:

$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \frac{\partial}{\partial w} \delta_t$$

Move to a $\leftarrow a'$ and s $\leftarrow s'$

end for

Update the critic or the Q value network (Q-learning)

Actor-critic Reinforcement Learning

Advantage Actor Critic

From Bellman equation, we have

$$A_t = Q_\pi(\mathbf{s}_t, a_t) - V_\pi(\mathbf{s}_t)$$

$$Q_\pi(\mathbf{s}, q) = R + \gamma \sum_{\mathbf{s}' \in \mathcal{S}} P(\mathbf{s}'|\mathbf{s}, a) V_\pi(\mathbf{s}')$$

$$A_t = r_t + \gamma V_\pi(\mathbf{s}_{t+1}) - V_\pi(\mathbf{s}_t)$$

Actor-critic Reinforcement Learning

Advantage Actor Critic

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \left(Q_\pi(\mathbf{s}_t, a_t) - V_\pi(\mathbf{s}_t) \right) \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t | s_t)$$

w. r. t. $\pi_\theta(a|s) = p_\theta(a|s)$

$$\frac{\partial}{\partial \theta} J(\pi_\theta) = \mathop{\mathbb{E}}_{s_t, a_t, r_t} \sum_t \left(r_t + \gamma V_\pi(\mathbf{s}_{t+1}) - V_\pi(\mathbf{s}_t) \right) \cdot \frac{\partial}{\partial \theta} \log p_\theta(a_t | s_t)$$

w. r. t. $\pi_\theta(a|s) = p_\theta(a|s)$

Actor-critic Reinforcement Learning

Put things together: Advantage Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta \left(r_t + \gamma V_w(s') - V_w(s)\right) \partial \log \pi_\theta(a|s)$
  the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma\, V_w(s') - V_w(s)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \frac{\partial}{\partial w} \delta_t$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Summary

Value-based

Pros:

- When it works, it performs good;
- When it works, sampling and training is efficient.

Cons:

- May not work for your case, e.g., continuous action space;
- No guarantee of find a good enough policy.

Policy gradient

Pros:

- Generally works for most cases;
- Will converge to a local minima, i.e., find a good enough policy.

Cons:

- High variance on gradient;
- Require lots of samples to train.

Actor-critic

- A joint method of policy-based and value-based;
- Usually, state-of-the-art results (SOTA) are achieved using variant of this kind of algorithm.