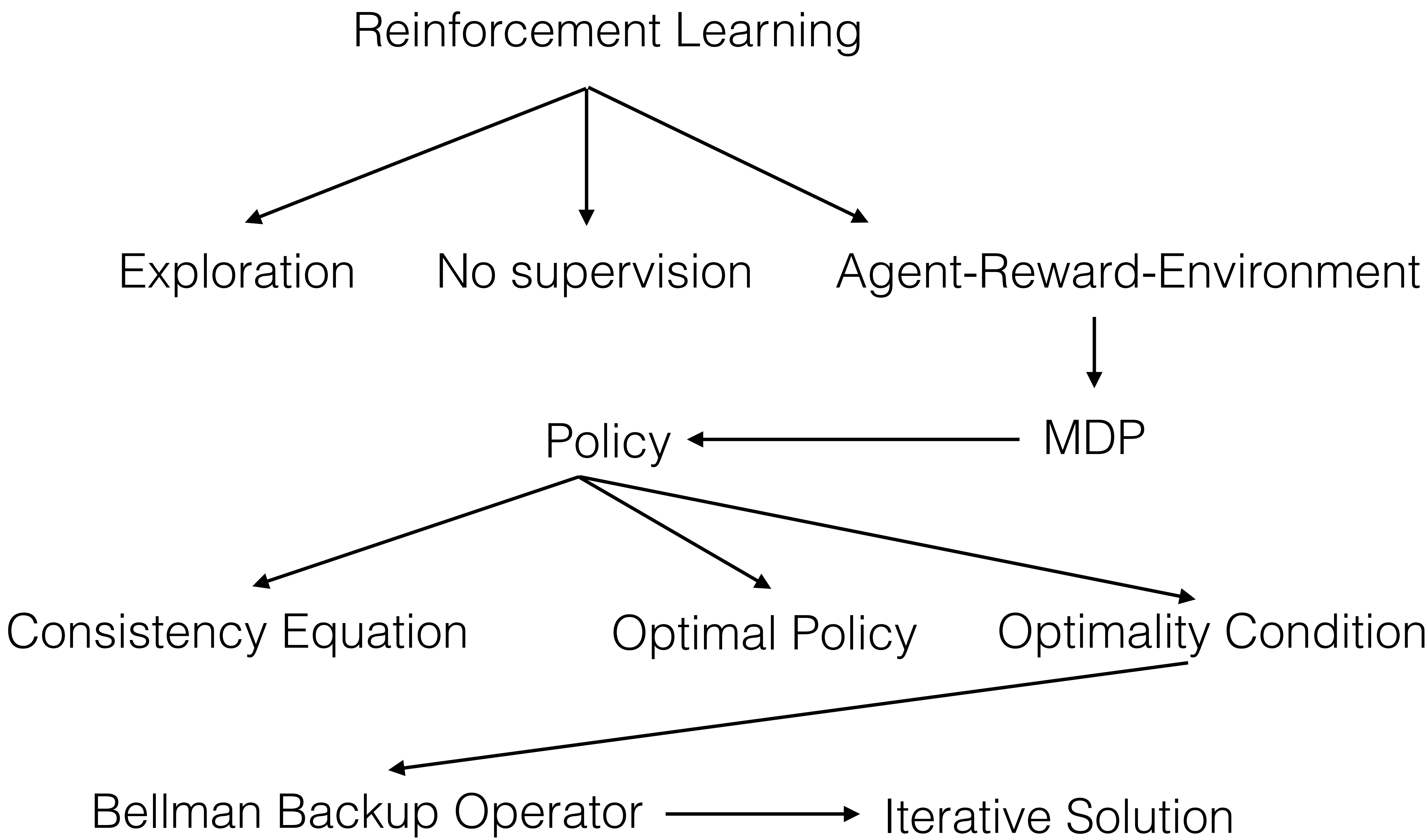
Reinforcement Learning
Part 2

Dipendra Misra
Cornell University
dkm@cs.cornell.edu

https://dipendramisra.wordpress.com/

From previous tutorial



Interaction with the environment

action

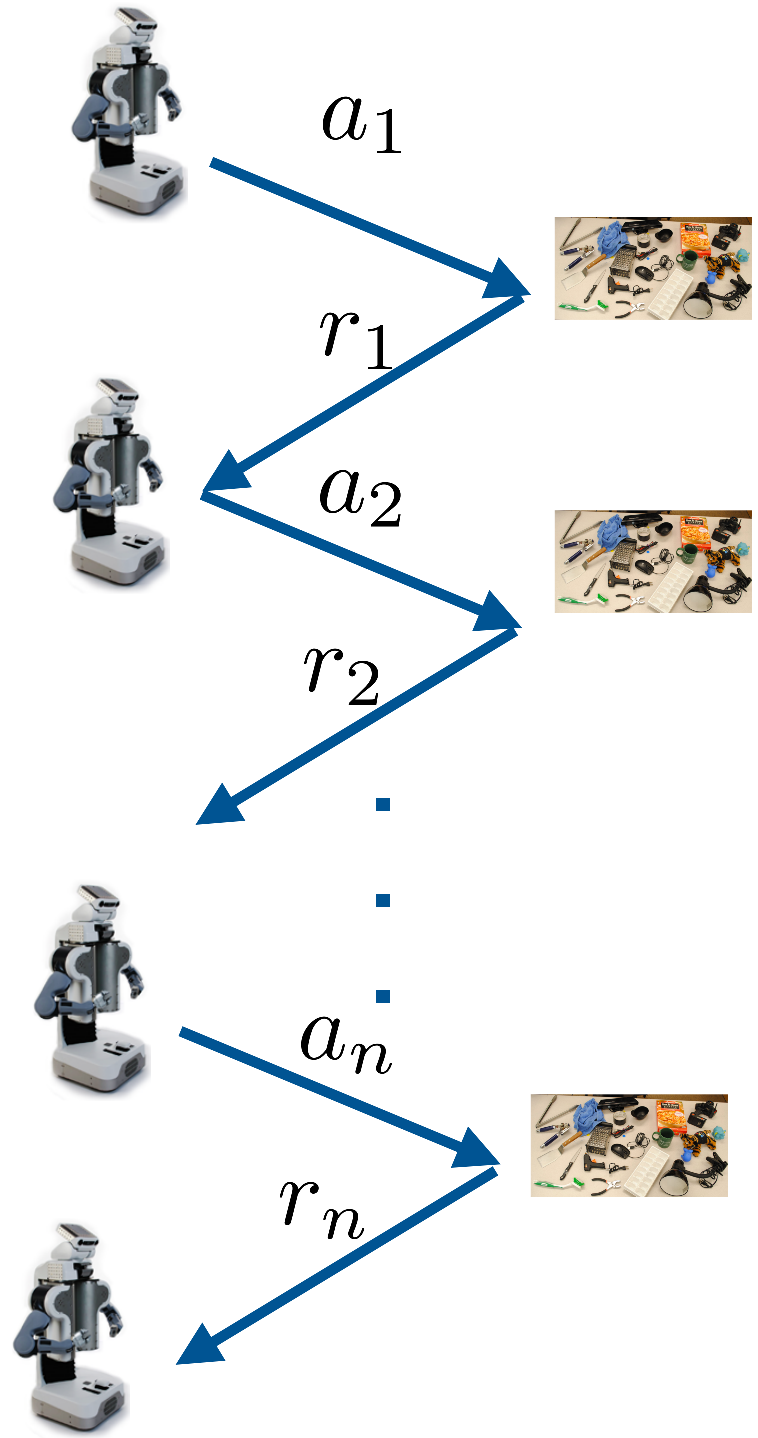
reward
+
new environment

Scalar reward

Setup from Lenz et. al. 2014

Rollout

$$\langle s_1, a_1, r_1, s_2, a_2, r_2, s_3, \cdots a_n, r_n, s_n \rangle$$



Setup from Lenz et. al. 2014

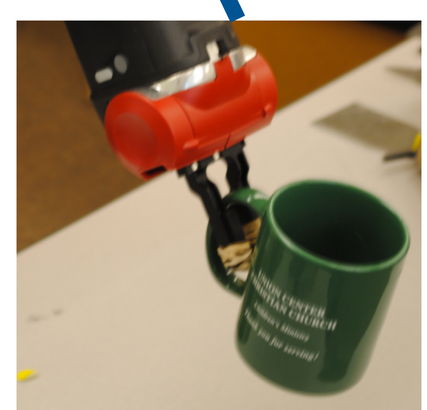
Setup

$$s_t \xrightarrow[r_t]{a_t} s_{t+1}$$

e.g.,1$

Policy

$$\pi(s, a) = 0.9$$



From previous tutorial

An optimal policy π^* exists such that:

$$V^{\pi^*}(s) \geq V^{\pi}(s) \quad \forall s \in \mathcal{S}, \pi$$

Bellman's self-consistency equation

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{s,s'} \left\{ R^{a}_{s,s'} + \gamma V^{\pi}(s') \right\}$$

Bellman's optimality condition

$$V^{*}(s) = \max_{a} \sum_{s'} P^{a}_{s,s'} \{ R^{a}_{s,s'} + \gamma V^{*}(s') \}$$

Solving MDP

To solve an MDP (or RL problem)
is to find an optimal policy

Dynamic Programming Solution

Initialize V^0 randomly

do

$$V^{t+1} = TV^t$$

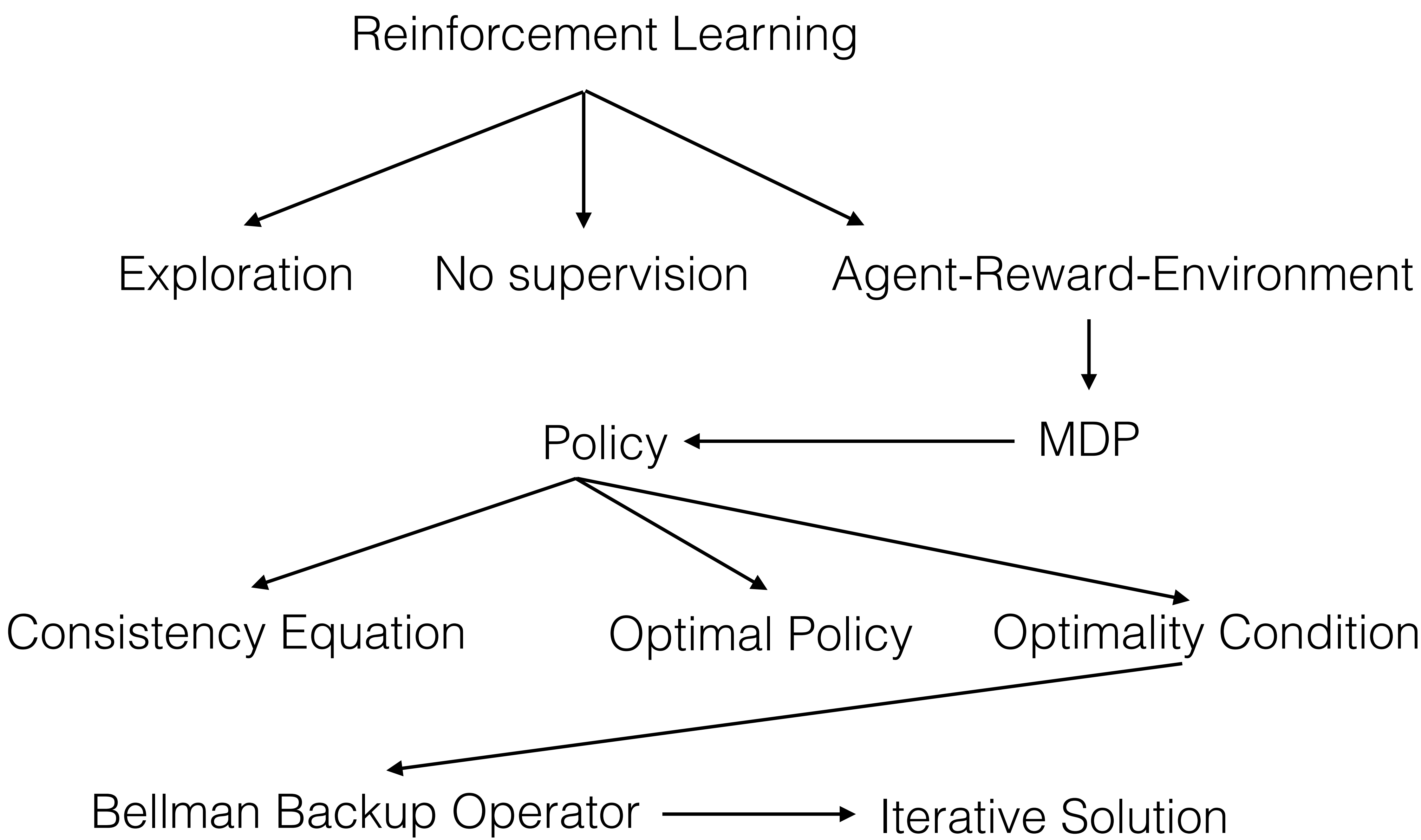
until $\|V^{t+1} - V^t\|_\infty > \epsilon$

return V^{t+1}

$$T : V \to V$$

$$(TV)(s) = \max_a \sum_{s'} P^a_{s,s'} \{R^a_{s,s'} + \gamma V(s')\}$$

From previous tutorial



Dynamic Programming Solution

Initialize V^0 randomly

do

$$V^{t+1} = TV^t$$

until $\|V^{t+1} - V^t\|_\infty > \epsilon$

Problem?

return V^{t+1}

$$T : V \rightarrow V$$

$$(TV)(s) = \max_a \sum_{s'} P^a_{s,s'} \{R^a_{s,s'} + \gamma V(s')\}$$

Learning from rollouts

Step 1: gather experience using a behaviour policy

Step2: update value functions of an estimation policy

On-Policy and Off-Policy

On policy methods

behaviour and estimation policy are same

Off policy methods

behaviour and estimation policy can be different

Advantage?

Behaviour Policy

- Encourage exploration of search space
- Epsilon-greedy policy

$$\pi_\epsilon(s,a) = \begin{cases} 1-\epsilon+\dfrac{\epsilon}{|\mathcal{A}(s)|} & a = \arg\max_{a'} Q(s,a') \\ \dfrac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

Temporal Difference Method

$$Q^\pi(s,a) = E_\pi\left[\sum_{t\geq 0}\gamma^t r_{t+1} | s_1 = s, a_1 = a\right]$$

$$= E_\pi\left[r_1 + \gamma\left(\sum_{t\geq 0}\gamma^t r_{t+2}\right) | s_1 = s, a_1 = a\right]$$

$$= E_\pi\left[r_1 + \gamma Q^\pi(s_2, a_2)\,|s_1 = s, a_1 = a\right]$$

$$Q^\pi(s,a) = (1-\alpha)Q^\pi(s,a) + \alpha(r_1 + \gamma Q^\pi(s_2,a_2))$$

combination of monte carlo and dynamic programming

SARSA

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma Q(s', a') - Q(s, a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

Converges w.p.1 to an optimal policy as long as all state-action pairs are visited infinitely many times and epsilon eventually decays to 0 i.e. policy becomes greedy.

On or off?

Q-Learning

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \underline{\max_{a'}} Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$;
    until s is terminal

Resemblance to Bellman optimality condition

$$Q^*(s,a) = \sum_{s'} P^a_{s,s'} \{R^a_{s,s'} + \gamma \max_{a'} Q^*(s',a')\}$$

For proof of convergence see:
http://users.isr.ist.utl.pt/~mtjspaan/readingGroup/ProofQlearning.pdf

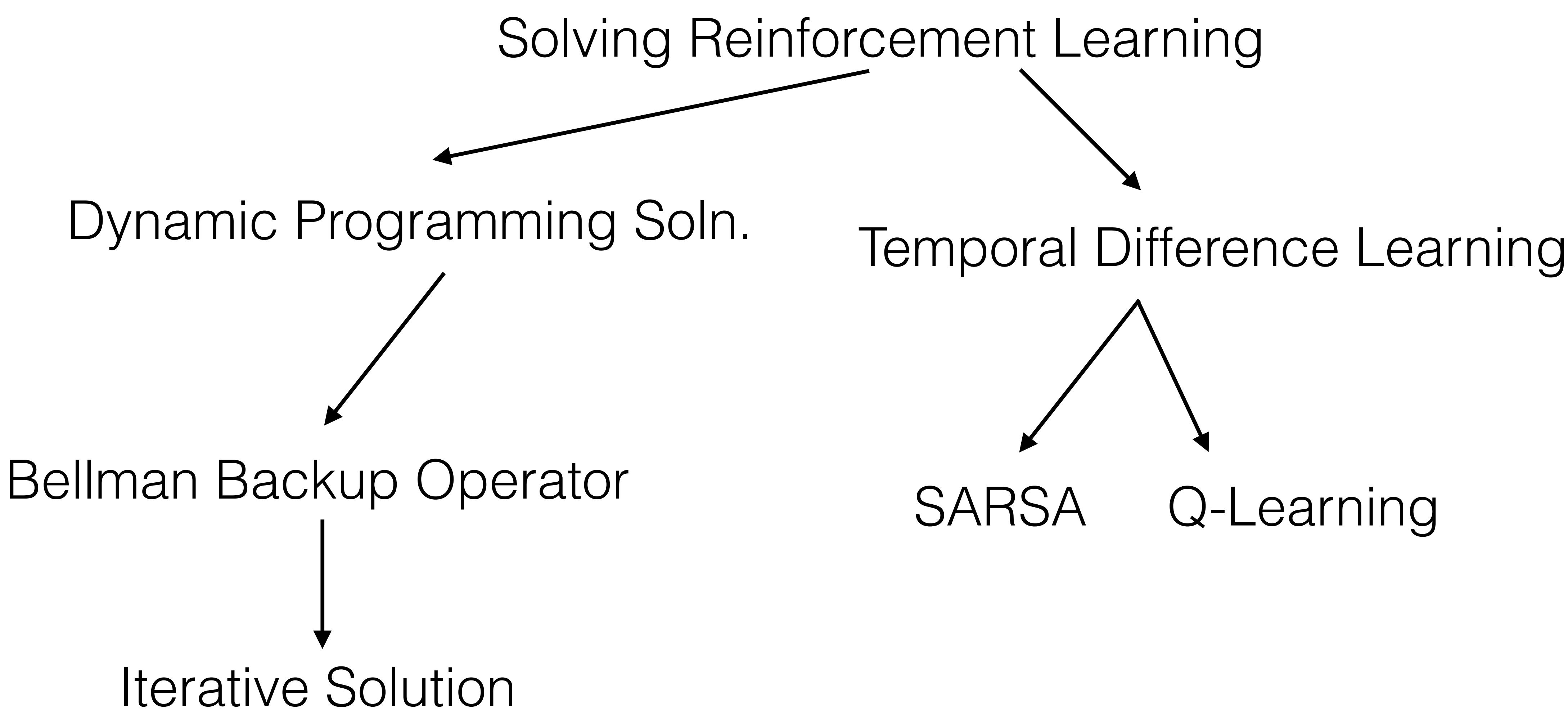
On or off?

Summary

- SARSA and Q-Learning

- On vs Off policy. Epsilon greedy policy.

What we learned



Another Approach

- So far policy is implicitly defined using value functions

- Can’t we directly work with policies

Policy Gradient Methods

- Parameterized policy $\pi_\theta(s, a)$
- Optimization $\max_\theta J(\theta)$ where $J(\theta) = E_{\pi_\theta(s,a)}\left[\sum_t \gamma^t r_{t+1}\right]$
- Gradient descent. Smoothly evolving policy.
- Obtaining gradient estimator?

On or off?

Finite Difference Method

$$\frac{\partial J(\theta)}{\partial \theta_i} \approx \frac{J(\theta + \epsilon e_i) - J(\theta - \epsilon e_i)}{2\epsilon}$$

$$\theta_i^{t+1} \leftarrow \theta_i^t + \alpha \frac{\partial J(\theta^t)}{\partial \theta_i^t}$$

- Easy to implement and works for all policies.

Problem?

Likelihood Ratio Trick

$$J(\theta) = E_{t\sim p_\theta(t')}[R(t)] = \sum_t R(t) p_\theta(t)$$

$$\max_\theta J(\theta)$$

$$\nabla_\theta J(\theta) = \sum_t R(t) \nabla_\theta p_\theta(t)$$

$$= \sum_t R(t) p_\theta(t)\, \nabla_\theta \log p_\theta(t)$$

$$= E_{t\sim p_\theta(t')}[R(t) \nabla_\theta \log p_\theta(t)]$$

$$= E_{t\sim p_\theta(t')}[(R(t) - b) \nabla_\theta \log p_\theta(t)] \quad \forall\, b$$

Reinforce (Multi Step)

Policy gradient theorem:

$$\nabla_\theta J(\theta) = E_{\pi_\theta(s,a)}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta(s,a)}(s,a)]$$

initialize θ

for each episode $\langle s_1, a_1, r_1, s_2, a_2, r_2, s_3, \cdots a_n, r_n, s_n \rangle \sim \pi_\theta(s_1, a_1)$

for $t \in \{1, n\}$

$$v_t \sim Q_\theta^\pi(s_t, a_t)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

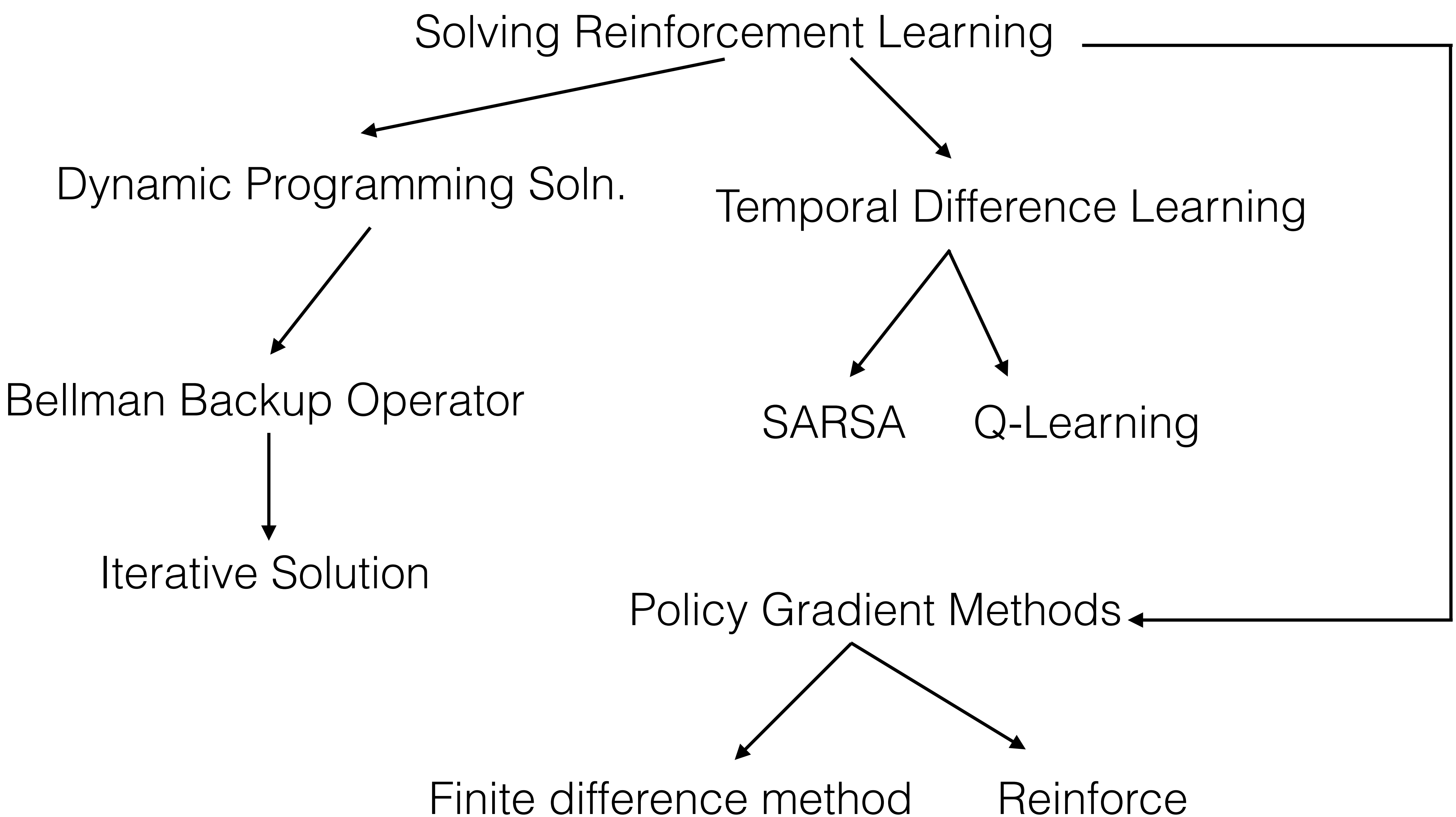
return θ

content from David Silver

Summary

- SARSA and Q-Learning
- On vs Off policy. Epsilon greedy policy.
- Policy Gradient Methods

What we learned



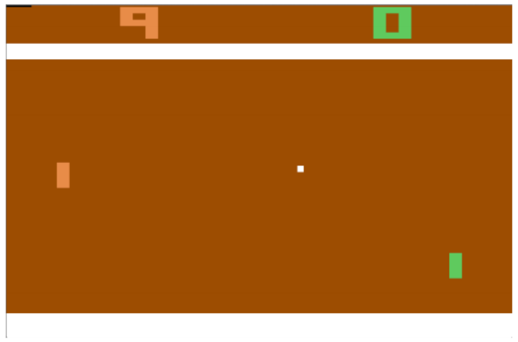
What we did not cover

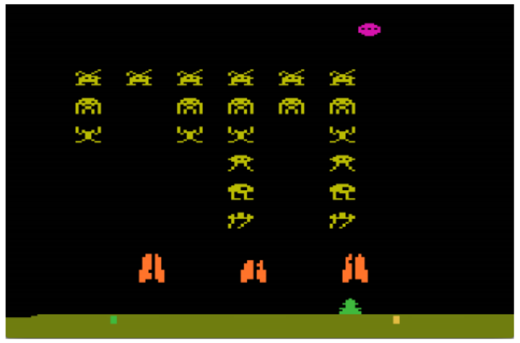
- Generalized policy iteration
- Simple monte carlo solution
- TD(λ) algorithm
- Convergence of Q-learning, SARSA
- Actor-critic method

• • •

Application

Playing Atari game with Deep RL






State is given by raw images.

Learn a good policy for a given game.

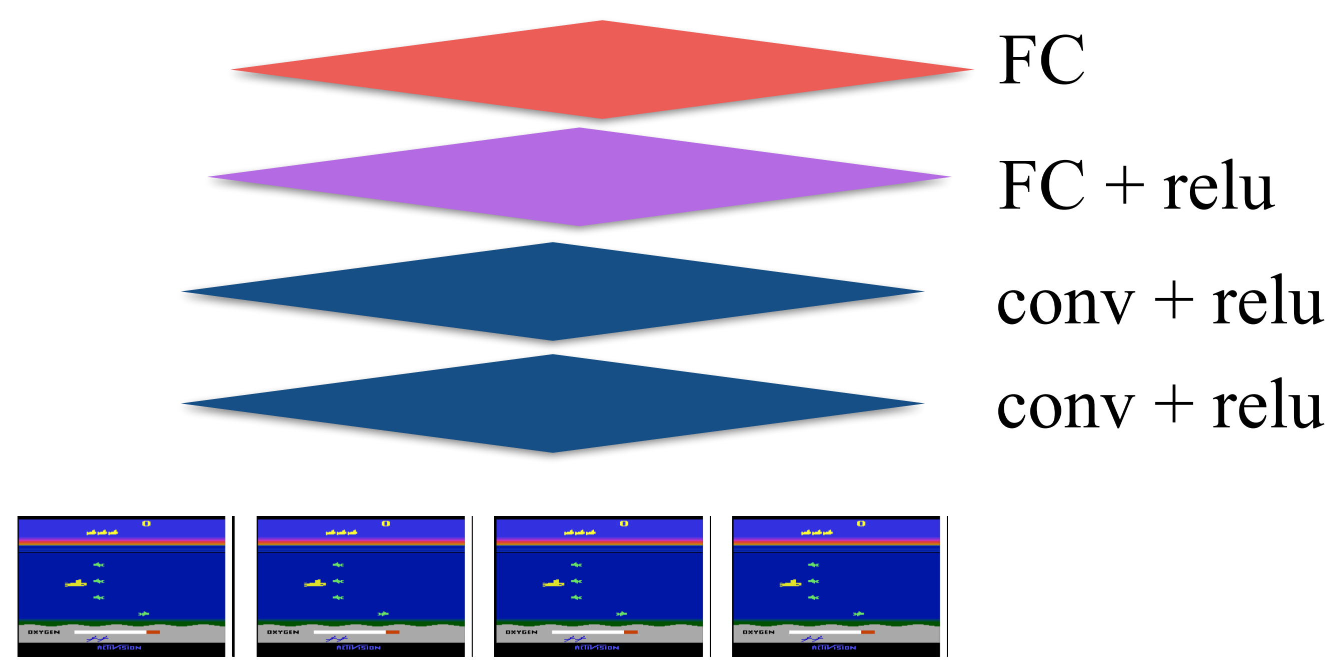
Playing Atari game with Deep RL

$$Q(s, a, \theta) \approx Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} P^a_{s,s'} \{R^a_{s,s'} + \gamma \max_{a'} Q^*(s', a')\}$$

$$= R^a_{s,s'} + \gamma \max_{a'} Q^*(s', a')$$

$$Q(s, a, \theta)$$



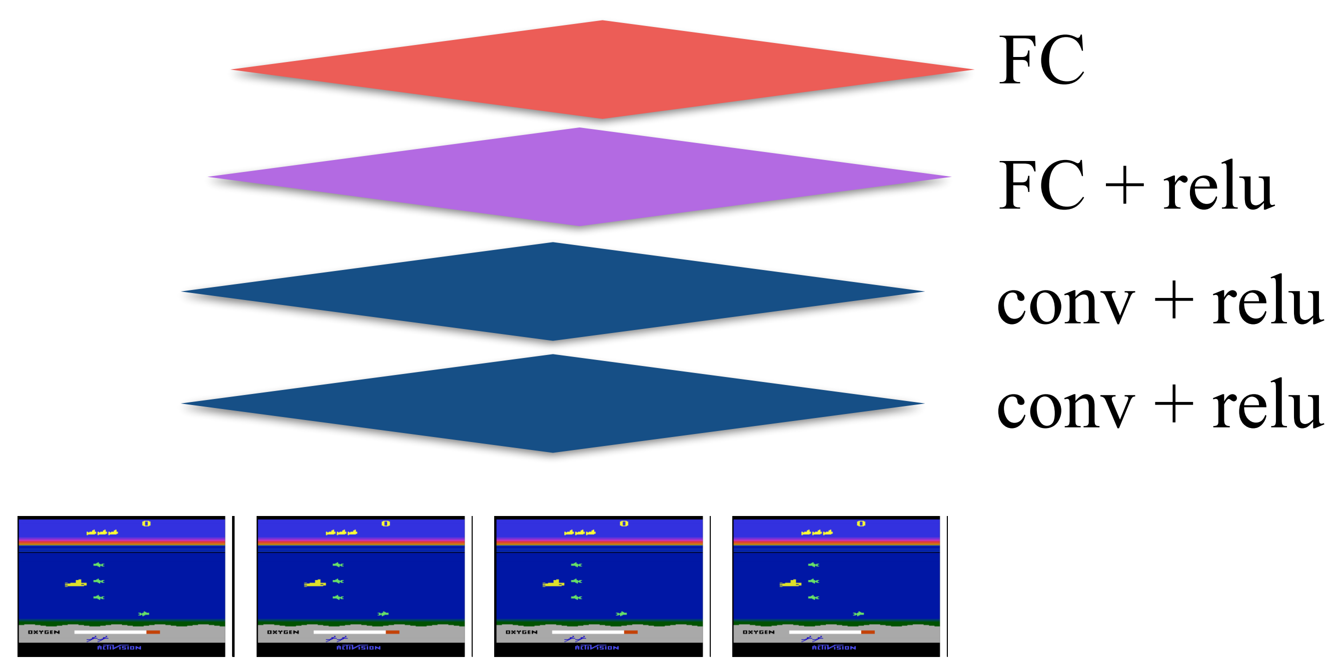
Playing Atari game with Deep RL

$$Q^*(s,a) = R^a_{s,s'} + \gamma \max_{a'} Q^*(s',a')$$

$$Q(s,a,\theta) \to R^a_{s,s'} + \gamma \max_{a'} Q(s',a',\theta)$$

$$\min(Q(s,a,\theta^t) - R^a_{s,s'} - \gamma \max_{a'} Q(s',a',\theta^{t-1}))^2$$

$$Q(s,a,\theta)$$



nothing deep about their RL

Playing Atari game with Deep RL

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Playing Atari game with Deep RL

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

why replay memory?

break correlation between consecutive datapoints

Playing Atari game with Deep RL

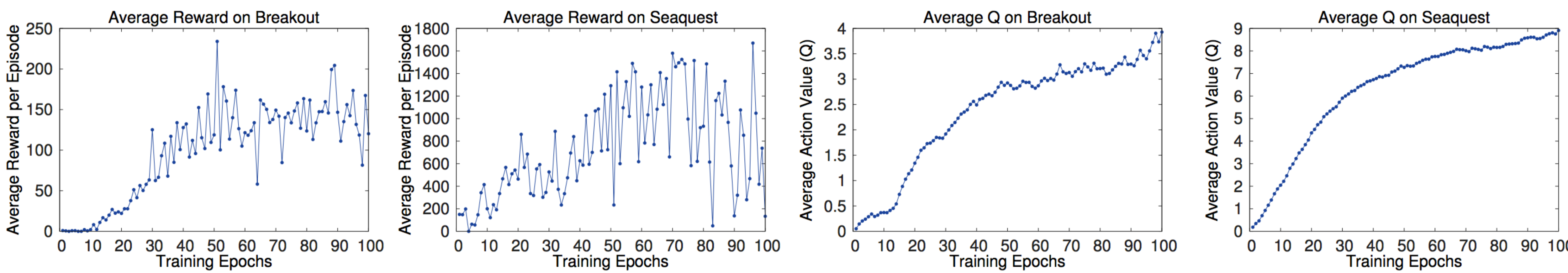


Figure 2: The two plots on the left show average reward per episode on Breakout and Seaquest respectively during training. The statistics were computed by running an ϵ-greedy policy with $\epsilon = 0.05$ for 10000 steps. The two plots on the right show the average maximum predicted action-value of a held out set of states on Breakout and Seaquest respectively. One epoch corresponds to 50000 minibatch weight updates or roughly 30 minutes of training time.

Why Deep RL is hard

$$Q^*(s,a) = \sum_{s'} P^a_{s,s'} \{R^a_{s,s'} + \gamma \max_{a'} Q^*(s',a')\}$$

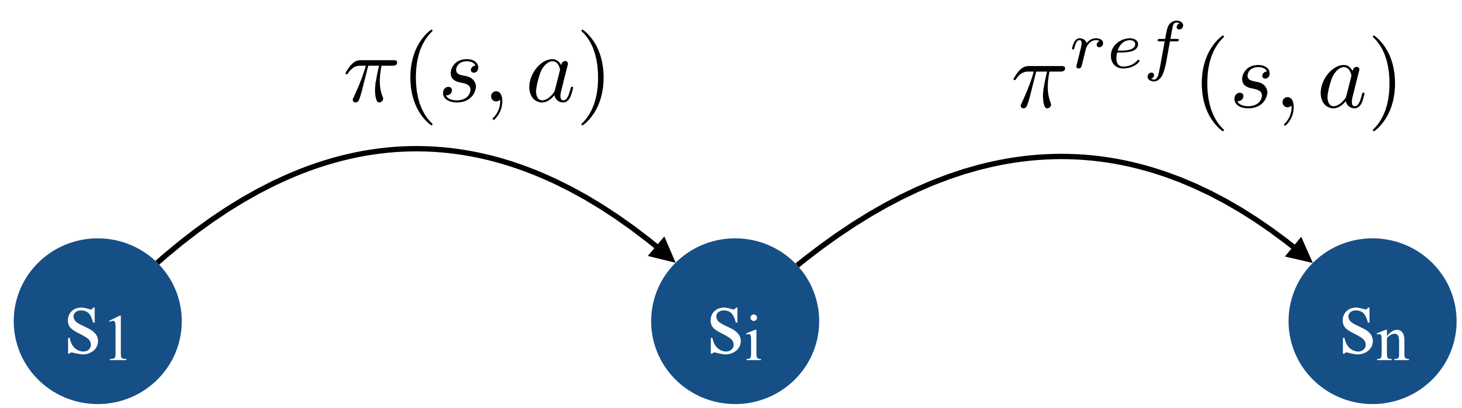
- Recursive equation blows as difference between s, s' is small

- Too many iterations required for convergence.
  10 million frames for Atari game.

- It may take too long to see a high reward action.

Learning to Search

- It may take too long to see a high reward.
- Ease the learning using a reference policy
- Exploiting a reference policy to search space better



Summary

- SARSA and Q-Learning
- On vs Off policy. Epsilon greedy policy.
- Policy Gradient Methods
- Playing Atari game using deep reinforcement learning
- Why deep RL is hard. Learning to search.