Reinforcement Learning

Michiel van de Panne

Department of Computer Science

The University of British Columbia

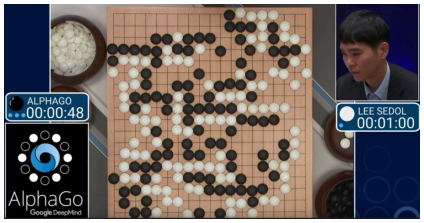
This lecture is based in part on, or inspired by, slides/blogs/demos from:

- Rich Sutton, U Alberta [Sutton] http://incompleteideas.net/book/the-book-2nd.html
- Emma Brunskill, Stanford [Brunskill] https://web.stanford.edu/class/cs234/index.html
- David Silver, UCL [Silver] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Pieter Abbeel, UC Berkeley [Abbeel]
- Sergey Levine, Chelsea Finn, John Schulman, UC Berkeley [UC Berkeley]
- Mark Schmidt, UBC [Schmidt]
- Andrej Karpathy [Karpathy] https://karpathy.github.io/2016/05/31/rl/
- Lillian Weng [Weng] https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Goals for this morning

- what is RL?
  - what can it solve? how does it differ from other ML problems?
- why Deep RL, and some caveats
- RL basics
- RL algorithms
  - Q-learning, gradient-free methods, policy-gradient methods
- current perspectives

What is RL?



[DeepMind]

[Mobileye]

[Anybotics]

Learn to make good sequences of decisions

[Brunskill]

- Drive a car
- Defeat the world champion at Go
- Manage an investment portfolio
- Sequence a series of medical tests and interventions
- Control a power station or a chemical process to maximize revenue
- Make a humanoid robot walk
- Direct attention for a computer vision task
- Understand the role of *dopamine* in the brain

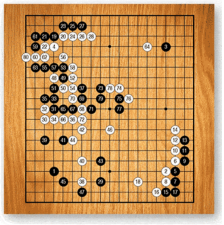
~[Silver]

Defining good decisions: rewards

- Fly stunt manoeuvres in a helicopter
  - +ve reward for following desired trajectory
  - −ve reward for crashing
- Defeat the world champion at Backgammon
  - +/−ve reward for winning/losing a game
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score

[Abbeel]

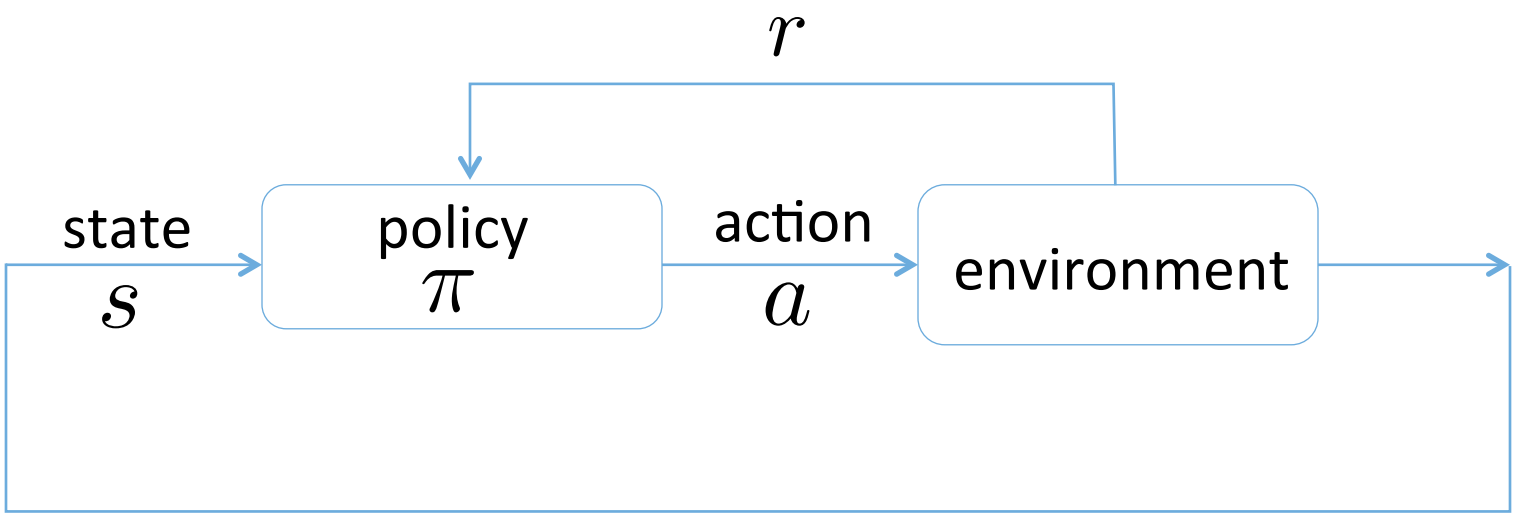
RL involves:



- optimization
  - find an optimal way to make sequential decisions
- delayed consequences
  - decisions now can impact things much later, e.g., climate change
  - challenge: temporal credit assignment is hard (what caused later high or low rewards?)
  - challenge: need to reason about long-term ramifications
- exploration
  - explore vs exploit tradeoff: try a new restaurant or go to one you already like?
  - current policy impacts future data collection: potential instabilities
- generalization
  - impossible to visit all states during learning, e.g. image input to a policy

~[Brunskill]

Reinforcement Learning



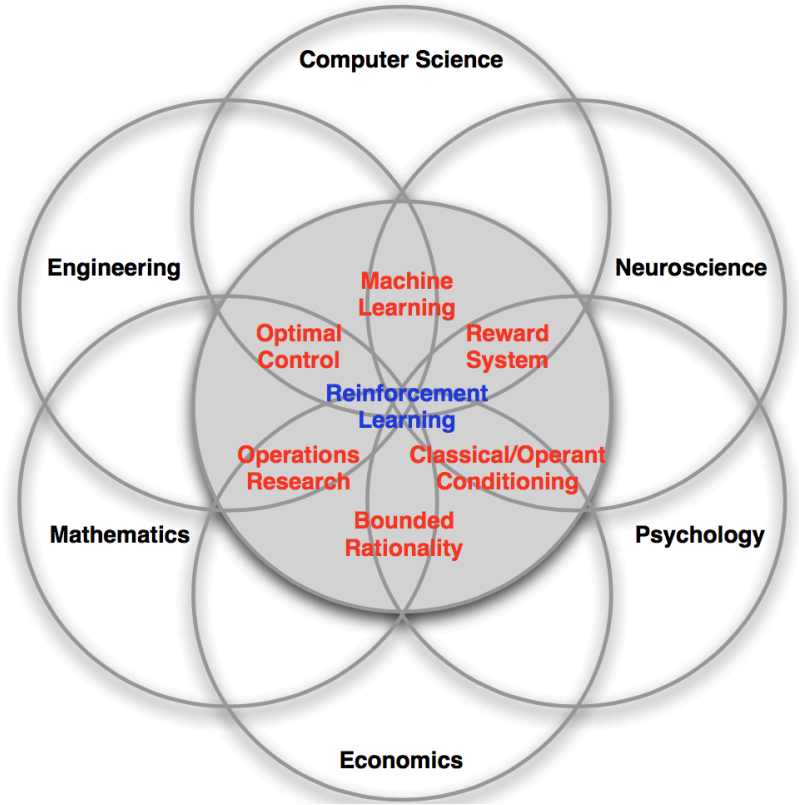
maximize sum of rewards: $G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

$$0 \leq \gamma \leq 1$$

Discrete vs Continuous

- discrete states, discrete actions
  - e.g., Go, Chess, tic-tac-toe
  - "tabular" policies for small problems
- continuous states, discrete actions
  - e.g., Atari games, DOTA, cart-and-pole
- continuous states, continuous actions
  - robotics, process control, autonomous driving

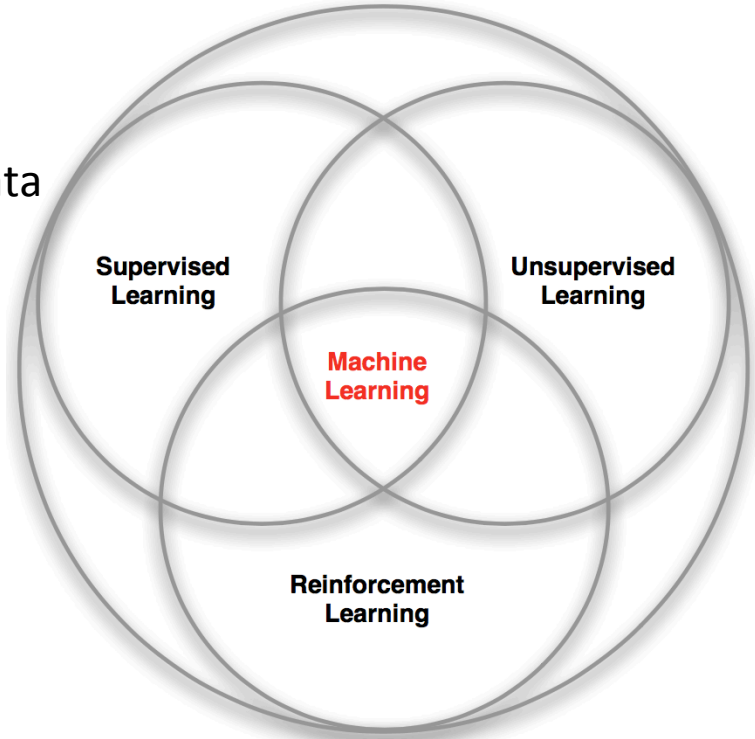
The Many Faces of Reinforcement Learning



[Silver]

Branches of Machine Learning

labeled data



latent structure ,
e.g., autoencoders

scalar rewards over time

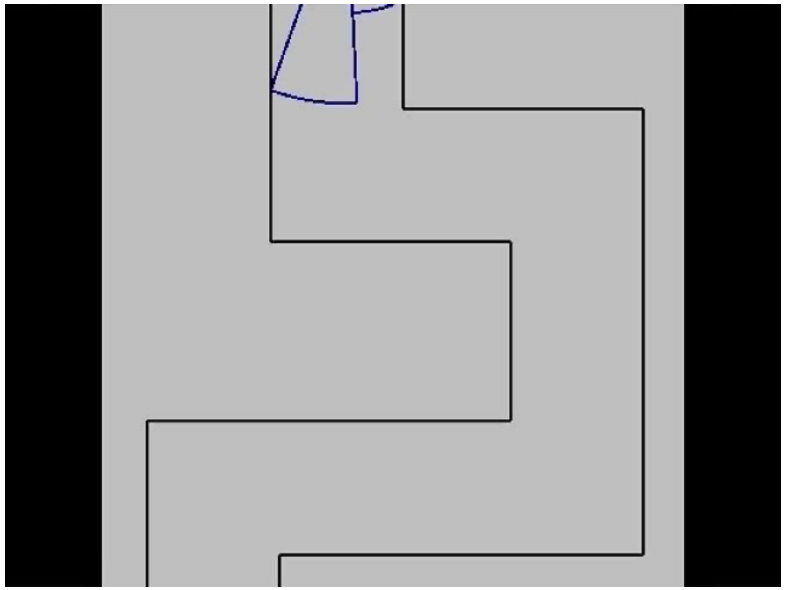
[Silver]

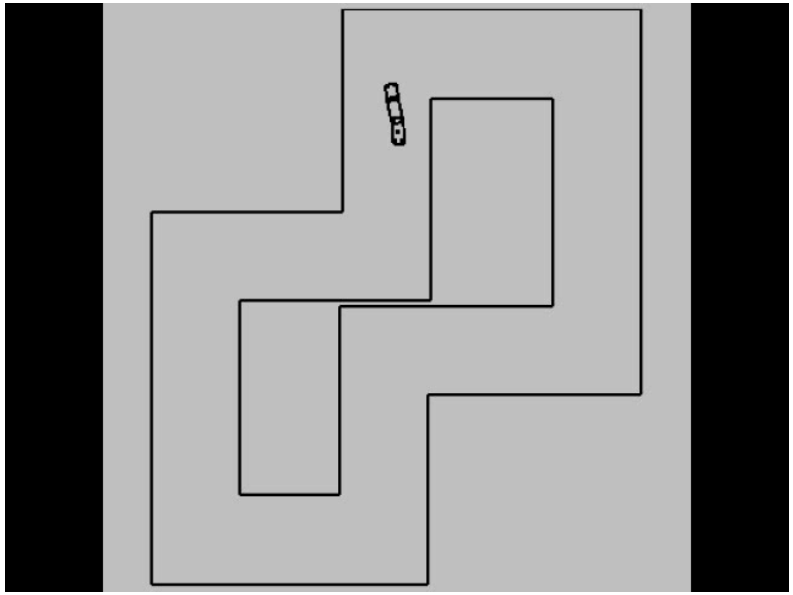
Simple Example: Backing up a Truck

$\pi : \mathbb{R}^5 \rightarrow \mathbb{R}$

input: 4 distances + cab-angle
output: steering angle

backing up a double trailer

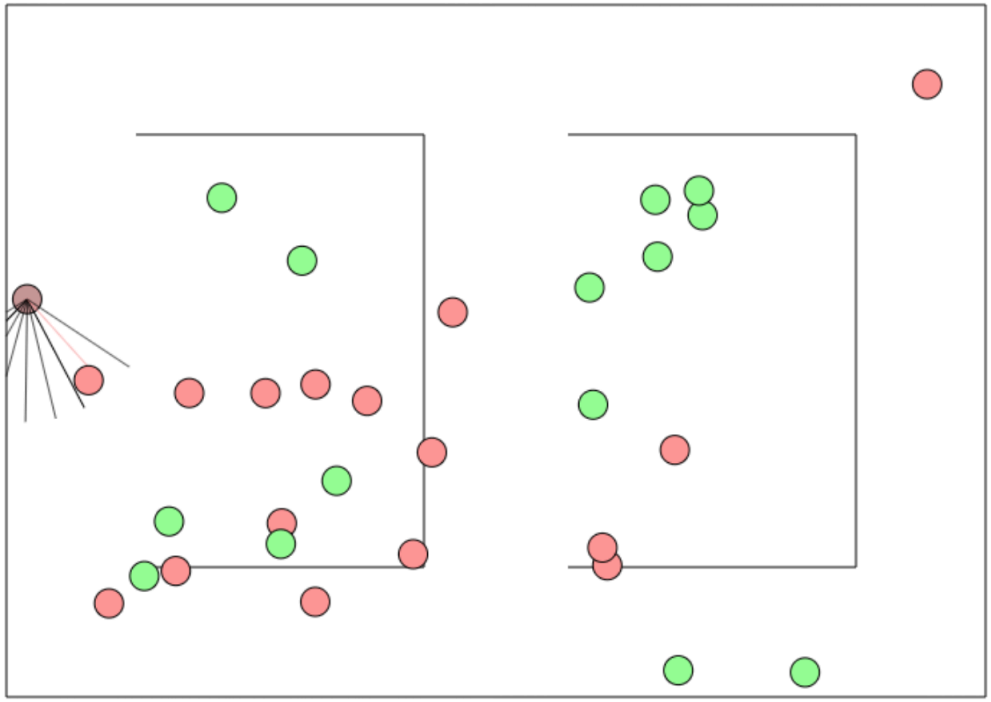




["Learning to Steer on Winding Tracks Using Semi-Parametric Control Policies", ICRA 2005]

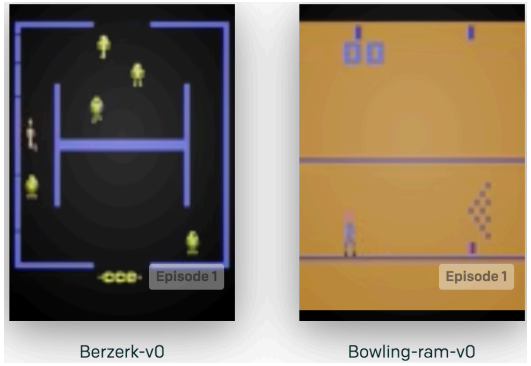
Simple Example

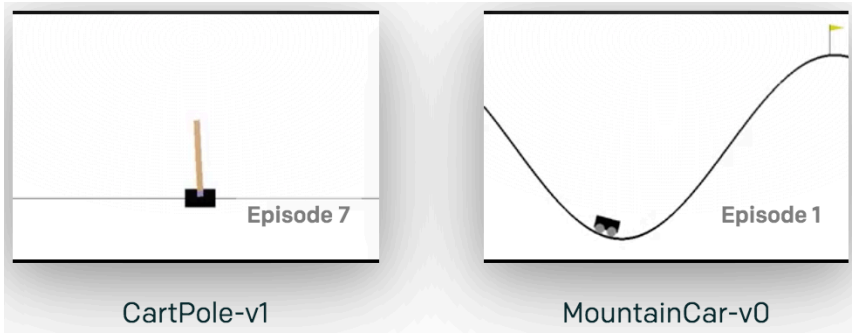
state: 9 sensors x 3 values x 2 timesteps
actions: move in 5 directions

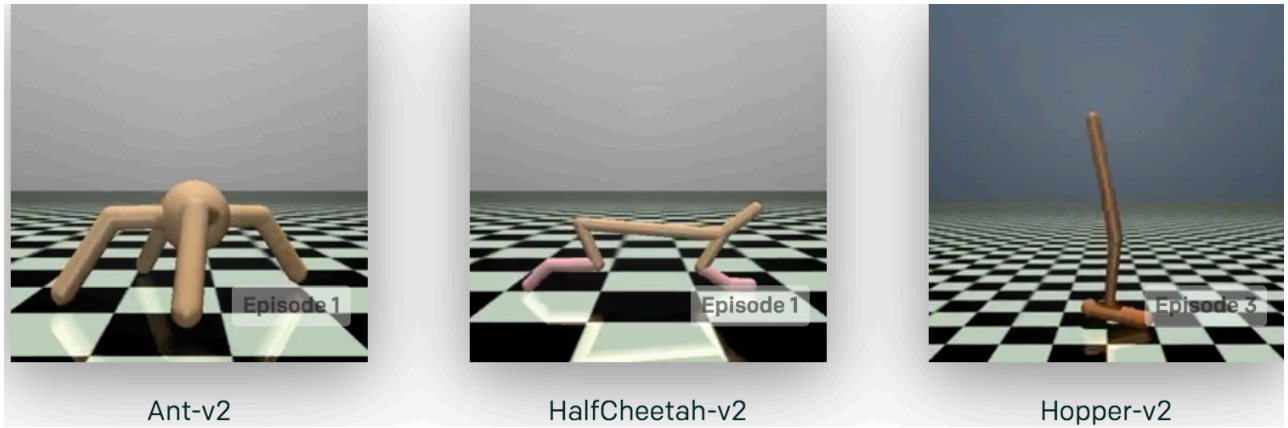


[https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html]

OpenAI Gym [gym.openai.com]

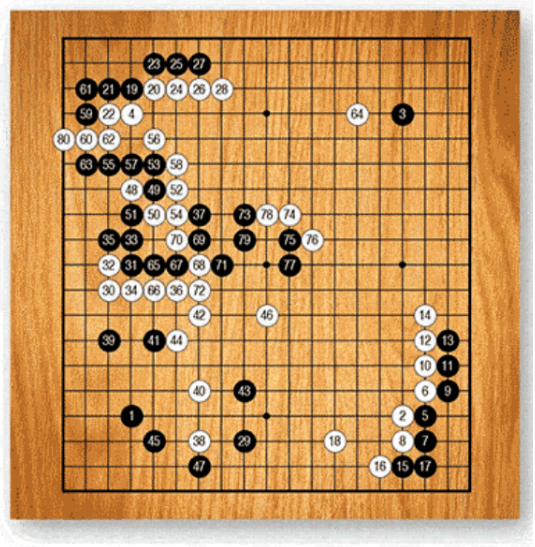


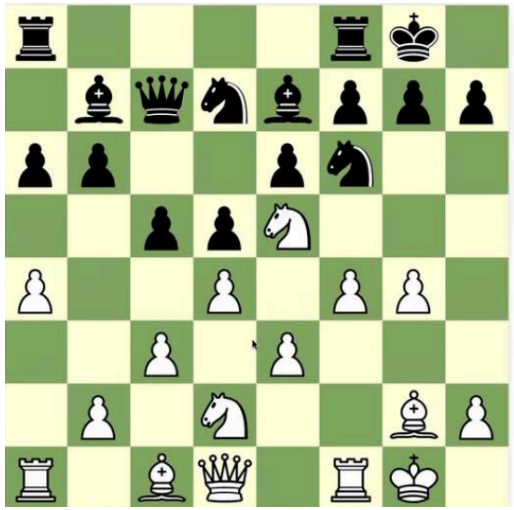




Why Deep RL?

AlphaZero (DeepMind)

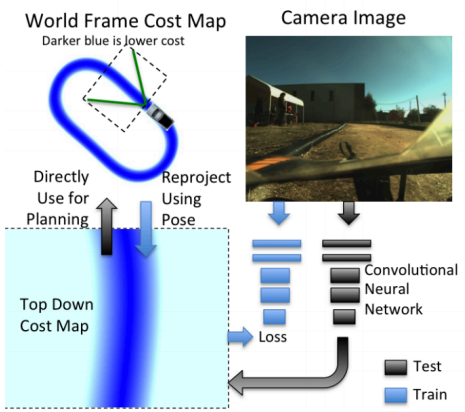




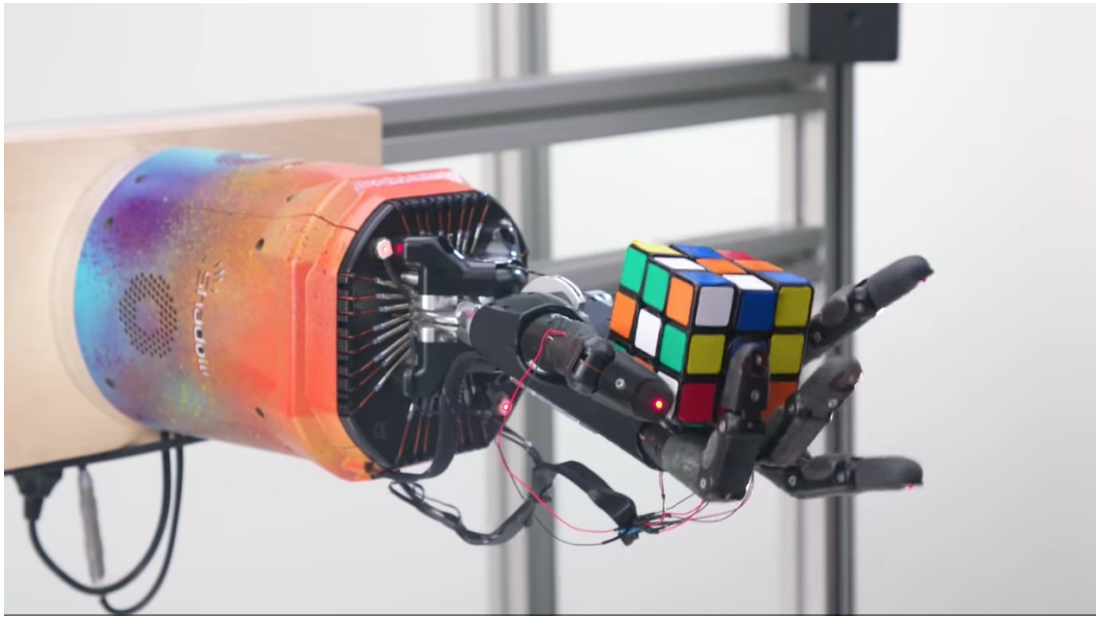
[DeepMind: Nature 2017]

Aggressive autonomous driving of a scale vehicle (Georgia Tech)

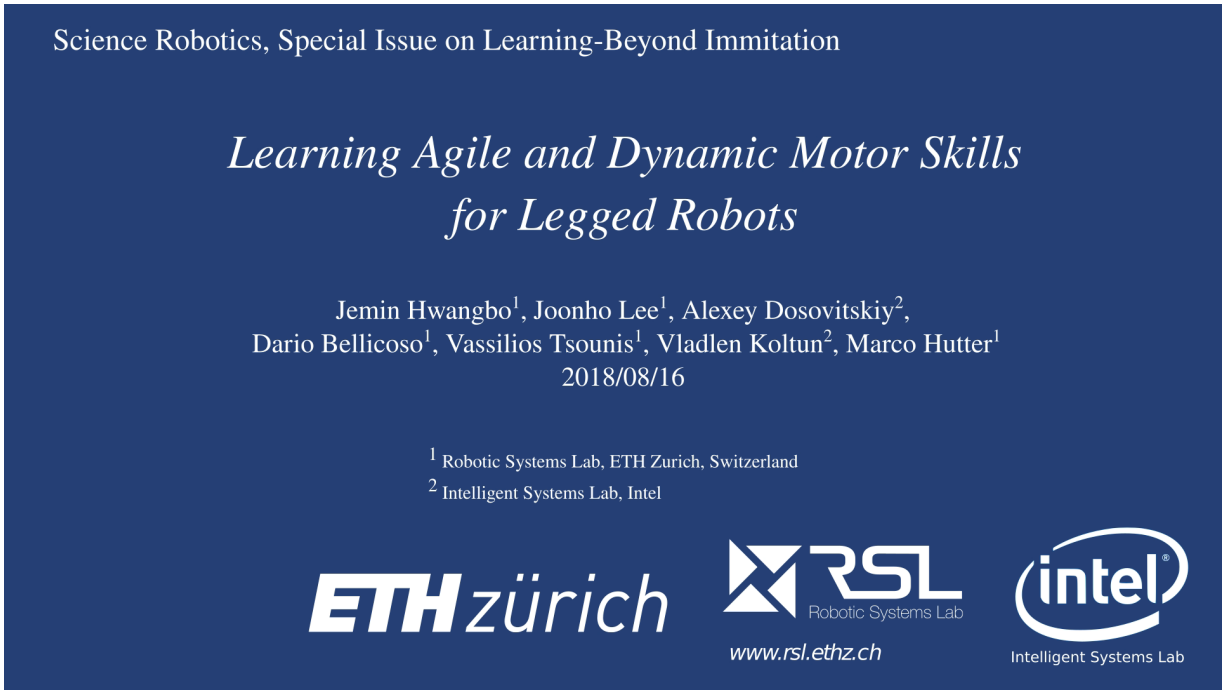
[Georgia Tech: CoRL 2017]



Rubik’s cube manipulation (OpenAI)



Anymal: Quadrupedal locomotion + getup (ETH Zurich)



[Science Robotics, 2018]

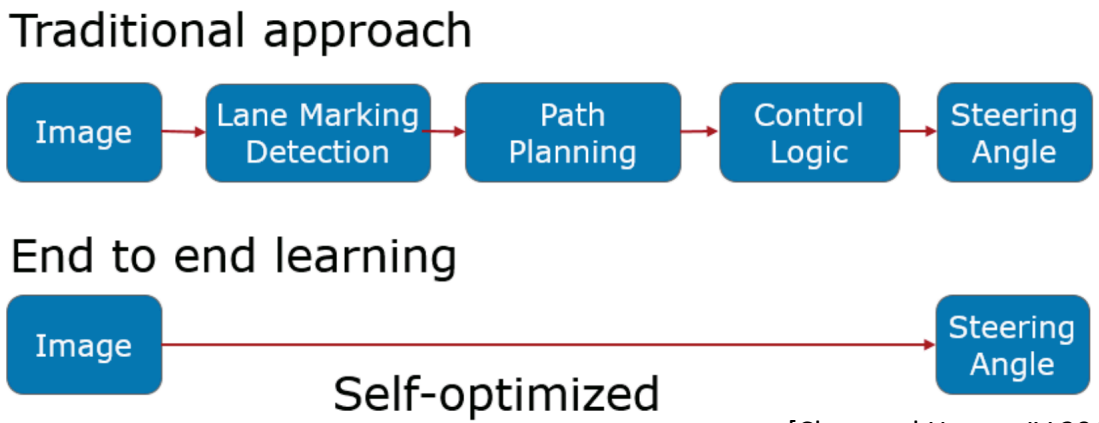
Physics-based Lion (UC Berkeley, UBC, Ziva Dynamics)

["DeepMimic", SIGGRAPH 2018]

Cassie: Bipedal Locomotion (UBC, U Oregon, Agility Robotics)

Why Deep RL? (continued)

End-to-end learning



[Chen and Huang, IV 2017]

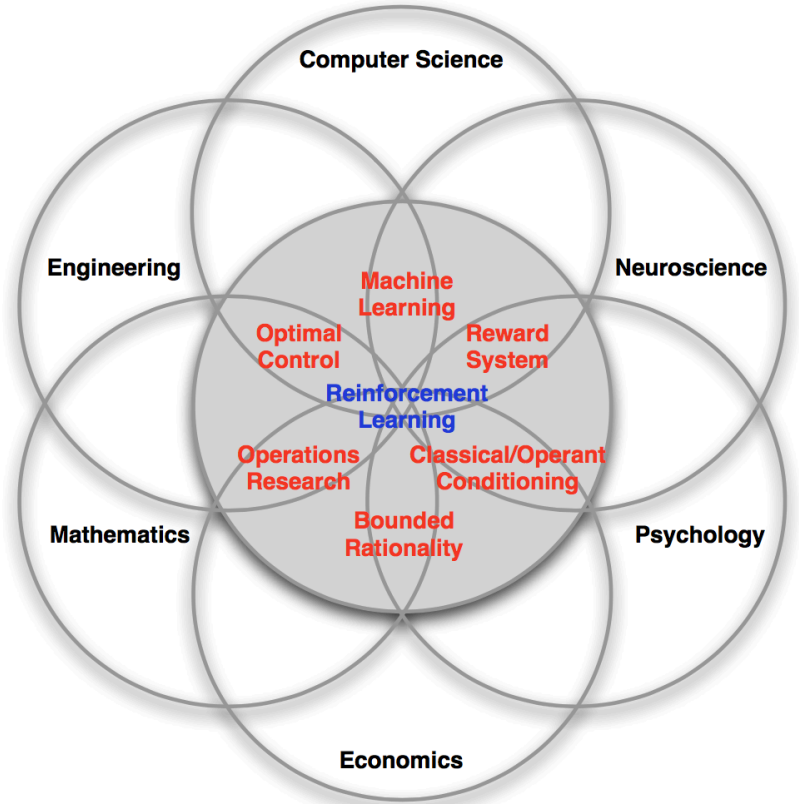
"pixels to torques"

Follow the money

- Alphabet invests ~ $2B in DeepMind
- Microsoft invests $1B in OpenAI
- autonomous driving: Mobileye, Wayve, Waymo, Tesla, …

Some Caveats

RL has a long history (by many names)



[Silver]

Generic Algorithm?
Best results often still have domain knowledge

- domain knowledge
  - defining: state, action, reward
  - hyperparameters
  - example expert data

but:

- no-free lunch: *inductive bias* needed for efficient learning
- progress is being made towards generalizable methods !
  - AlphaZero (Go, Chess), DQN (Atari games), OpenAI five (DOTA game)

RL learning often has poor "sample efficiency"

- often far too slow to learn directly in the real world
  - e.g., thousands of years in simulation (OpenAI hand)
- learning requires good simulation models, so not truly "model free"
- many newborn animals can walk within minutes or hours: giraffe, horse foal, piglets, camels, zebra and more

but:

- compute is cheap
- nature: has done much learning on an evolutionary time scale; is similar "transfer learning" possible for RL?

RL has very limited data to learn from

Y. LeCun

How Much Information is the Machine Given during Learning?

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

RL can be difficult to apply in practice

- Production systems
- Autonomous driving
- Business management
- Chemistry
- Computer Systems
- Energy
- Healthcare
- Robotics/manufacture

"Challenges of Real-World Reinforcement Learning"
[Dulac-Arnold et al.]

" … the research advances in RL are often hard to leverage in real-world systems due to a series of assumptions that are rarely satisfied in practice"

1. Training off-line from the fixed logs of an external behavior policy.
2. Learning on the real system from limited samples.
3. High-dimensional continuous state and action spaces.
4. Safety constraints that should never or at least rarely be violated.
5. Tasks that may be partially observable …
6. Reward functions that are unspecified, multi-objective, or risk-sensitive.
7. System operators who desire explainable policies and actions.
8. Inference that must happen in real-time at the control frequency of the system.
9. Large and/or unknown delays in the system actuators, sensors, or rewards.

Reprodicibility and Brittle Results

Deep Reinforcement Learning that Matters

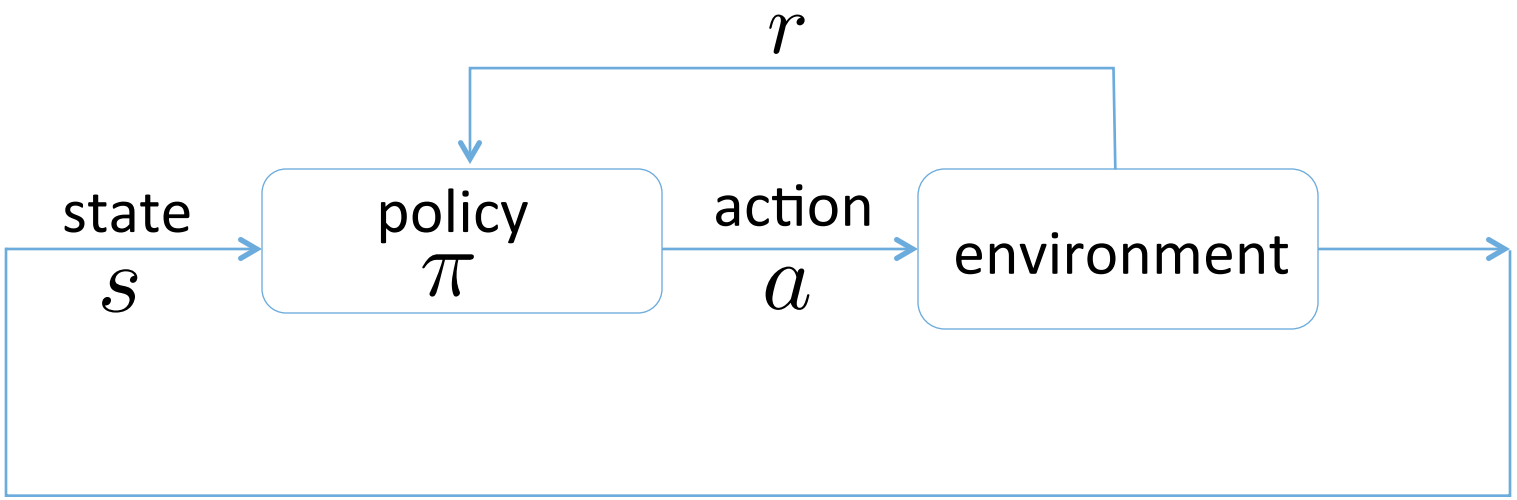
Peter Henderson[1]*, Riashat Islam[1,2]*, Philip Bachman[2]
Joelle Pineau[1], Doina Precup[1], David Meger[1]
[1] McGill University, Montreal, Canada
[2] Microsoft Maluuba, Montreal, Canada

In recent years, significant progress has been made in solving challenging problems across various domains using deep reinforcement learning (RL). Reproducing existing work and accurately judging the improvements offered by novel methods is vital to sustaining this progress. Unfortunately, reproducing results for state-of-the-art deep RL methods is seldom straightforward. In particular, non-determinism in standard benchmark environments, combined with variance intrinsic to the methods, can make reported results tough to interpret. Without significance metrics and tighter standardization of experimental reporting, it is difficult to determine whether improvements over the prior state-of-the-art are meaningful. In this paper, we investigate challenges posed by reproducibility, proper experimental techniques, and reporting procedures. We illustrate the variability in reported metrics and results when comparing against common baselines and suggest guidelines to make future results in deep RL more reproducible. We aim to spur discussion about how to ensure continued progress in the field by minimizing wasted effort stemming from results that are non-reproducible and easily misinterpreted.

"Unfortunately, reproducing results for state-of-the-art deep RL methods is seldom straightforward."

RL Basics

Reinforcement Learning – basics



maximize $G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

"return", cumulative discounted rewards

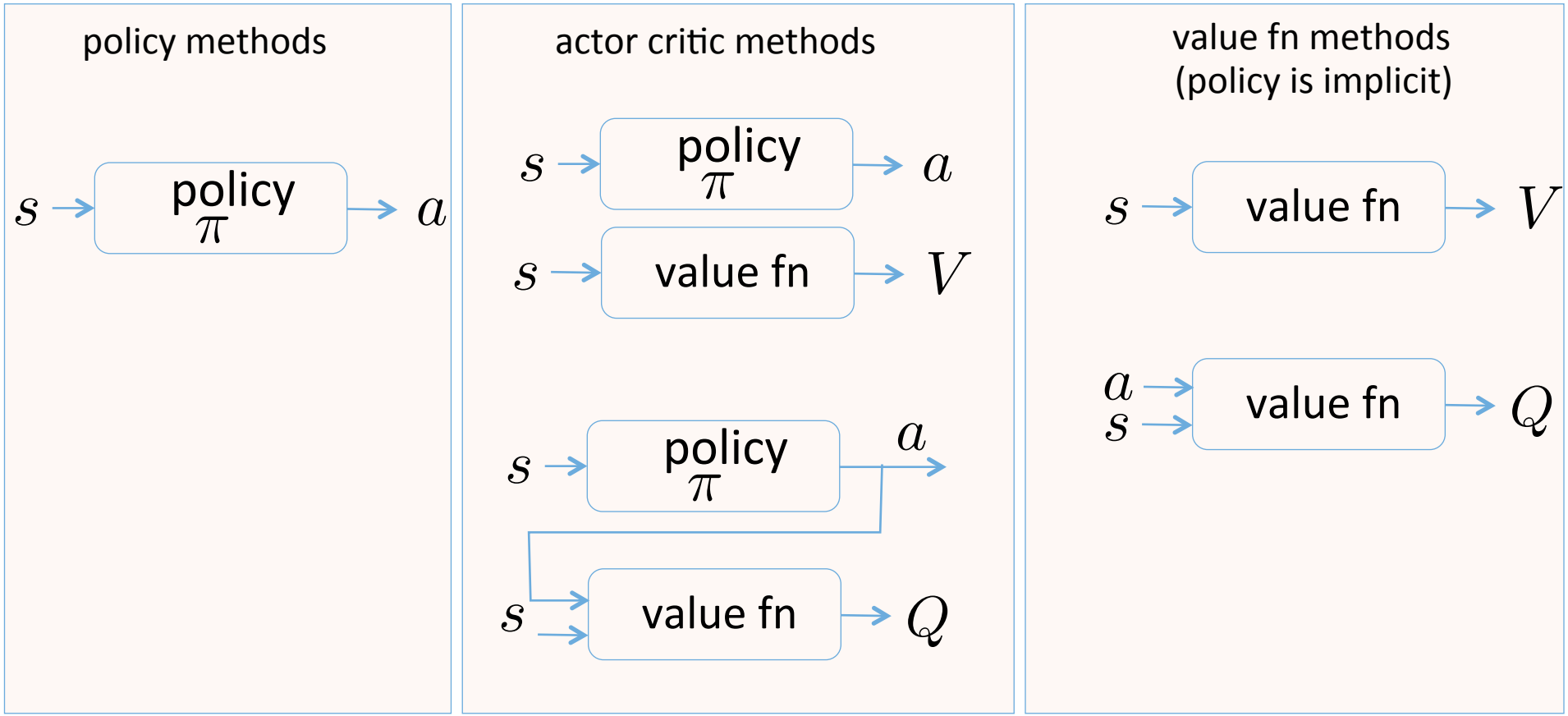
A Simple Solution to Learning a Policy: Copy an expert "Imitation Learning"

Observe what an expert does, and try to copy it:

Learn $a = \pi(s)$ from collected (s,a) tuples

- requires an expert
- unlikely to ever do better than the expert
- suffers from compounding errors over time
  - can be mitigated by collecting data during perturbations

Deep RL: key building blocks



Common Assumptions

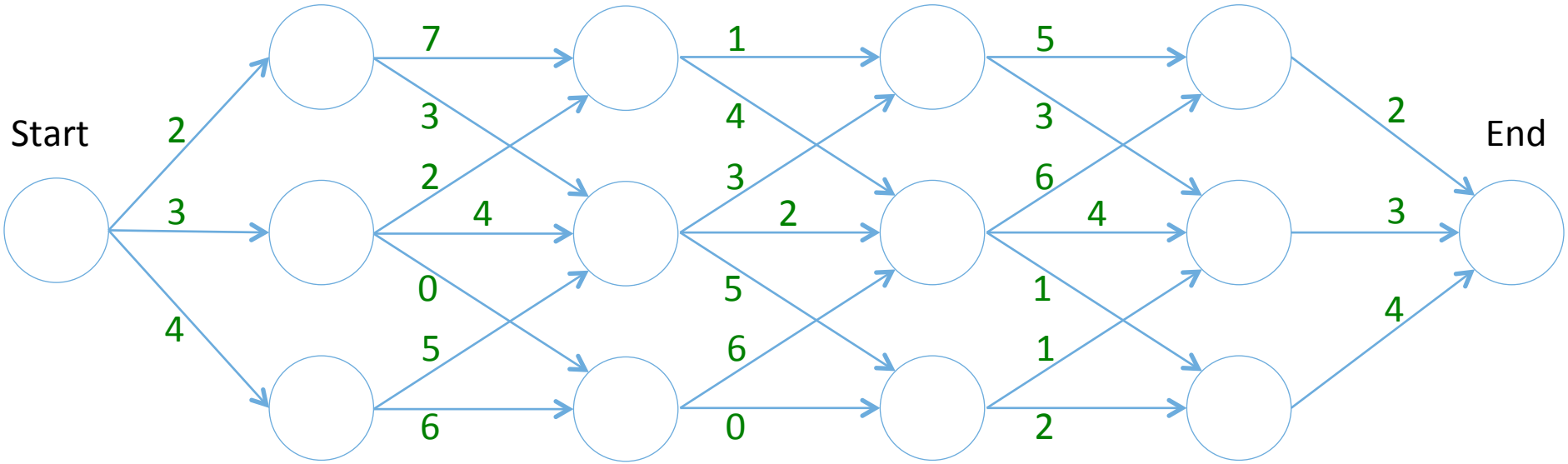
- *episodic* tasks
  - repeated interactions with the world e.g., games, trials
  - begins from a standard starting state (or distribution of states)
  - ends when reaching terminal state, or a fixed time T
- *discounting:* γ
  - avoid infinite rewards when T=∞
  - summarize uncertainty abou the future
- *stochastic policies*: action probabilities $\pi(a|s)$ vs $a = \pi(s)$
  - "smoothes"the learning, e.g., probability of playing a card, steering left, etc
  - provides exploratory actions (for some algorithms)
  - usually switch to deterministic policy once learning is complete

Common Assumptions

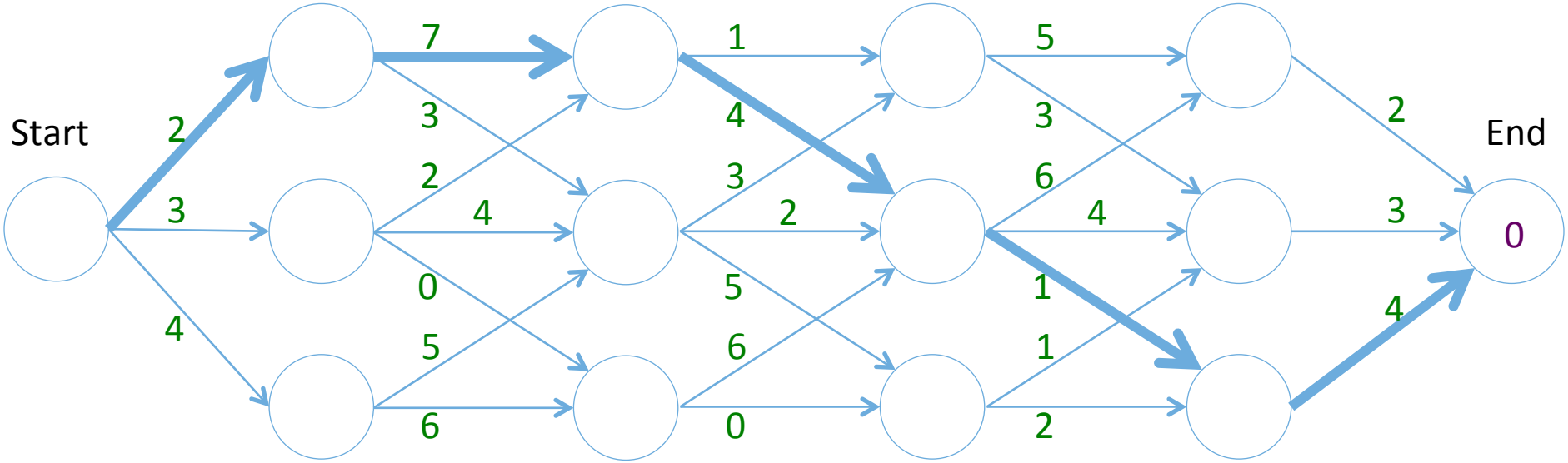
- Markov Property
  - the future only depends on the current state, not the history

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

Laying the Foundations:
A Simple Deterministic Example



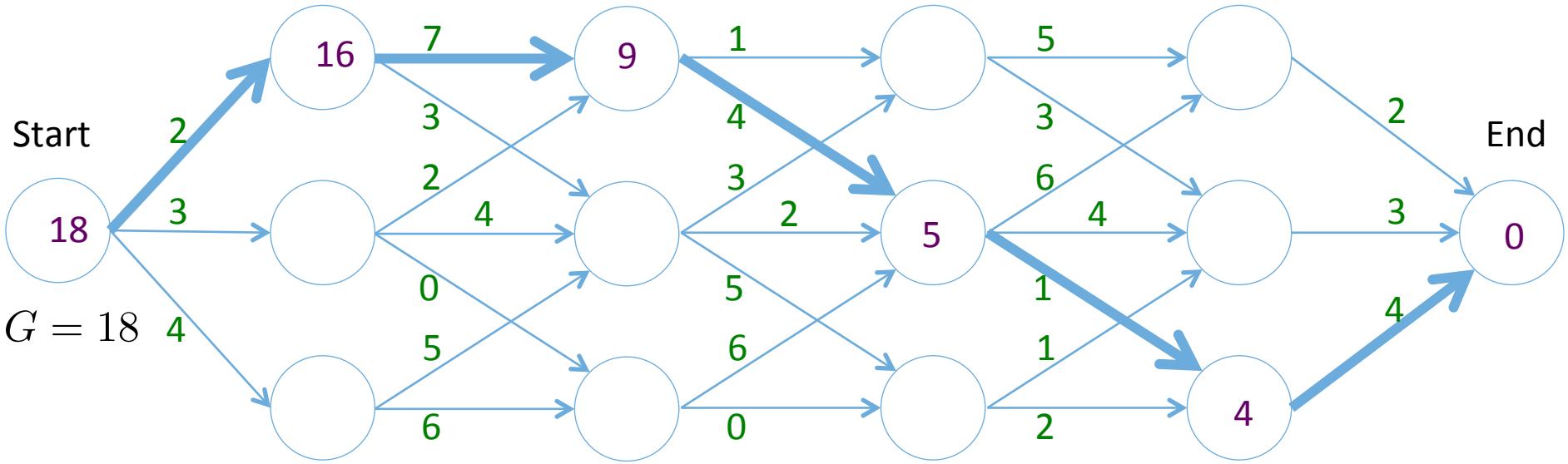
Returns for a given sequence of decisions



$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Above we have $\gamma = 1$

Returns for a given sequence of decisions



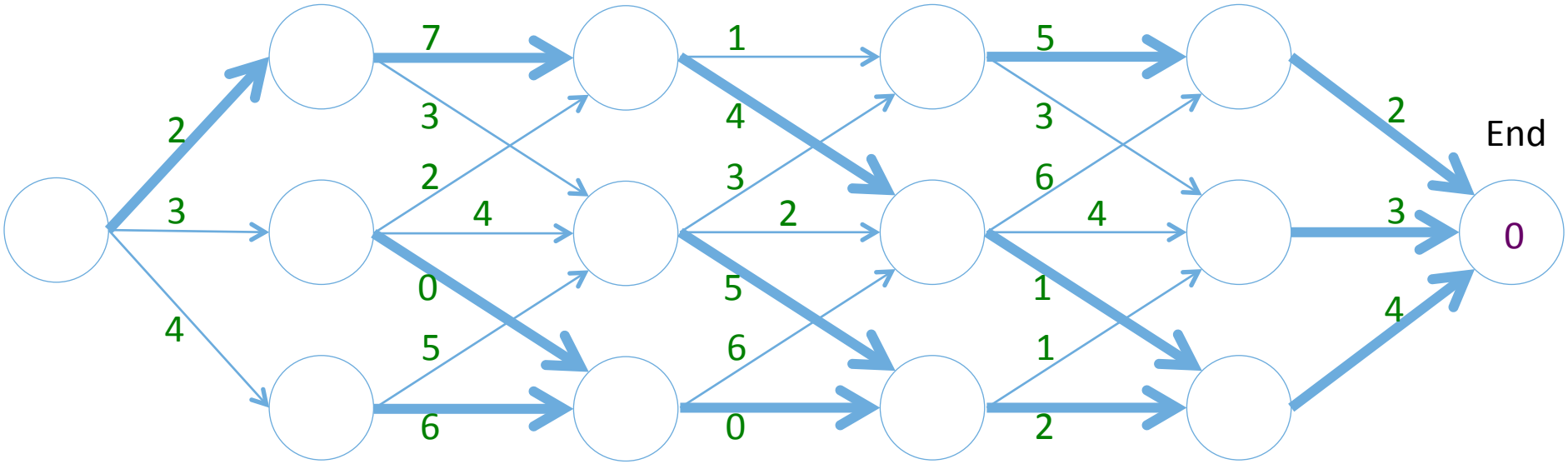
$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Above we have $\gamma = 1$

$$\gamma = 0.9: \; G(\text{start}) = 2 + (0.9)7 + (0.9)^2 4 + (0.9)^3 1 + (0.9)^4 4$$

Returns for a given policy π

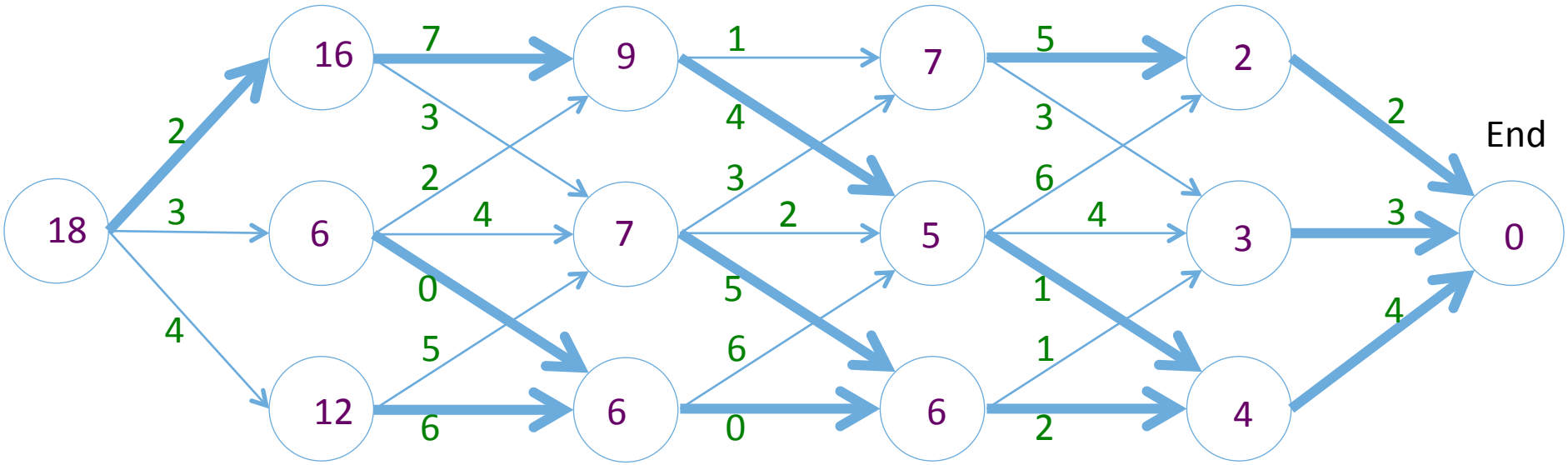
Assume that we begin in a random state, for this example, *"exploring start"* and therefore we care about the returns from all states.



$$\gamma = 1$$

$$G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

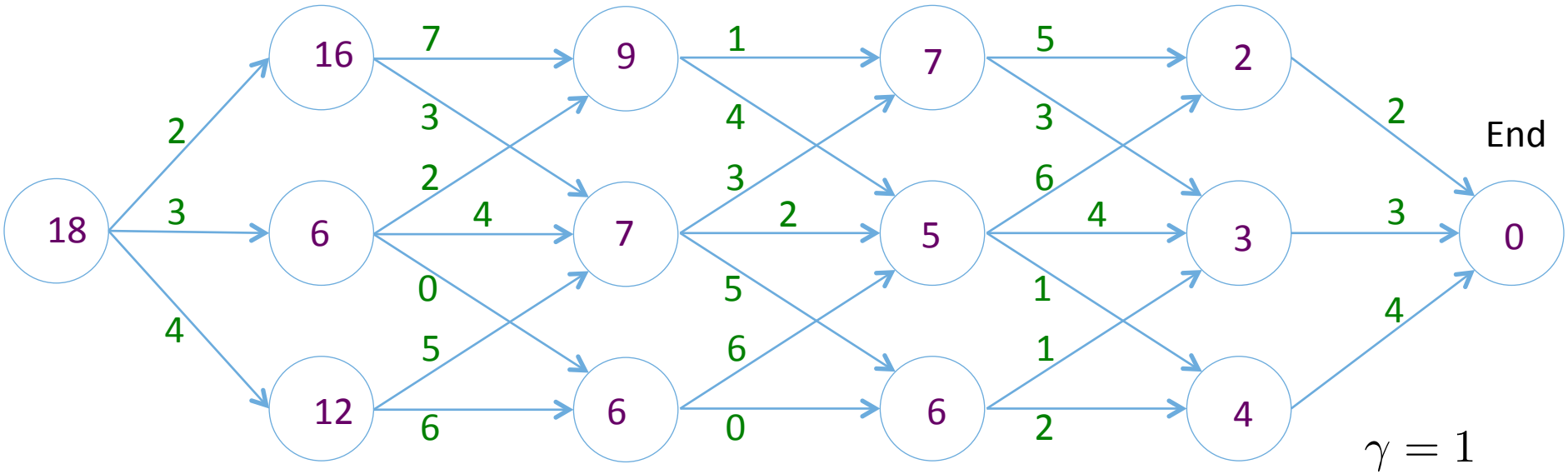
Returns for a given policy π



$$\gamma = 1$$

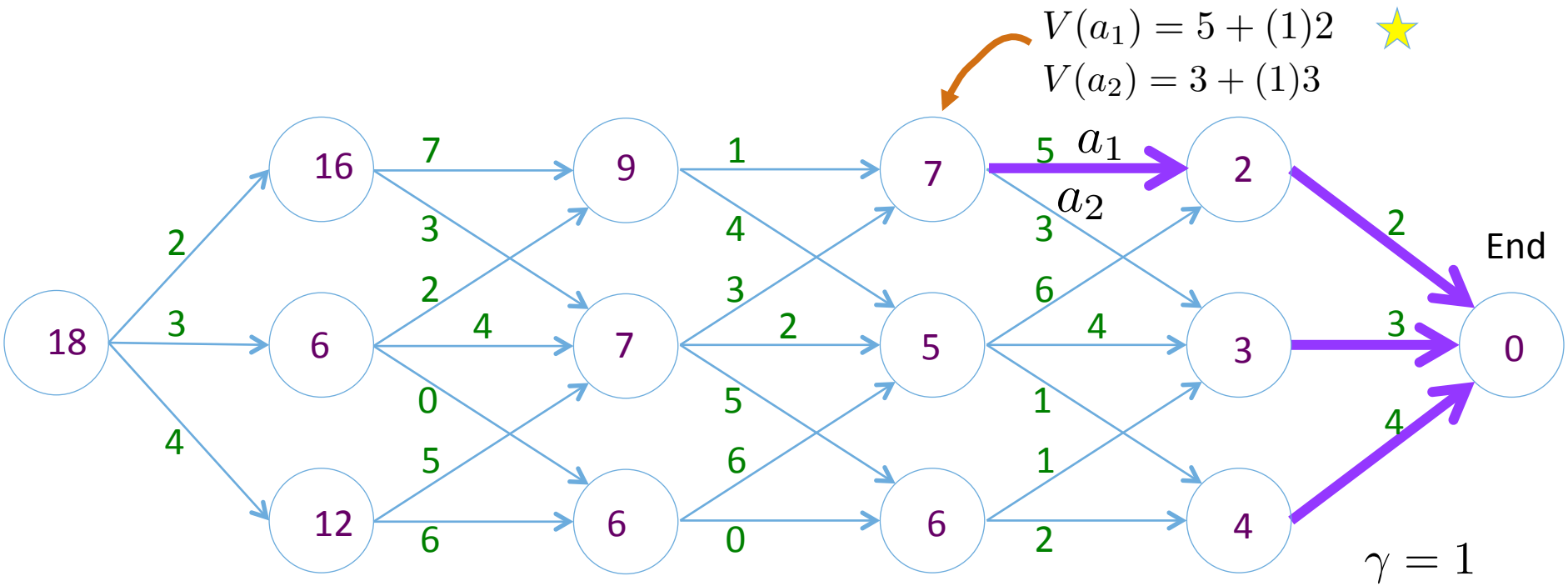
$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Given the known returns, now improve the policy



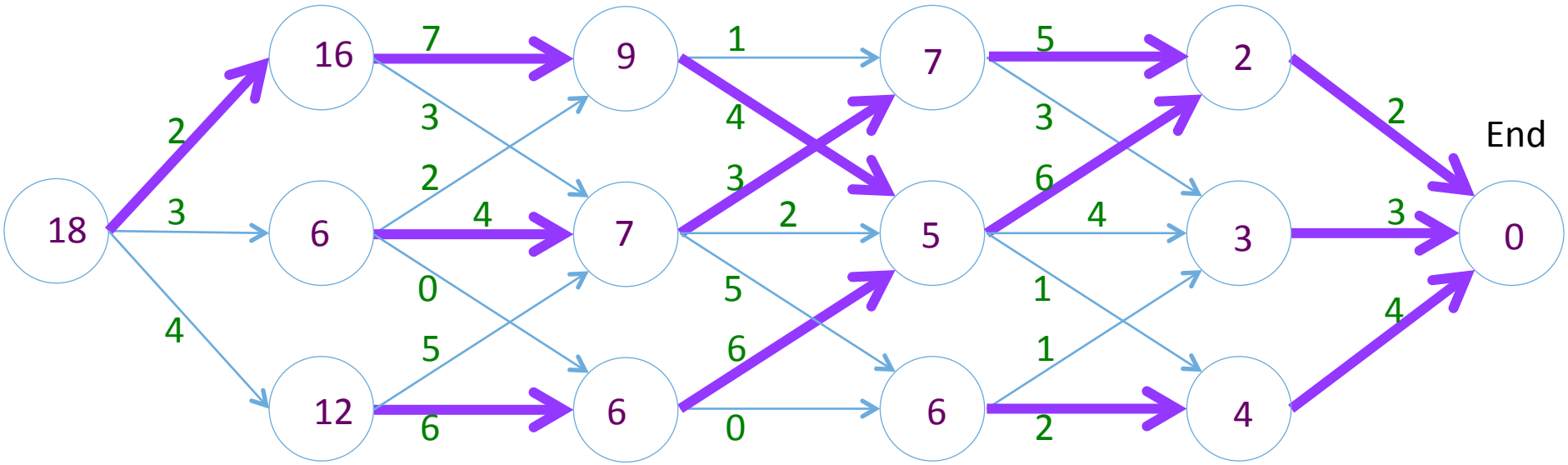
$$\pi(s) \leftarrow \arg\max \sum_{s',r} [r(s,a) + \gamma V(s')]$$

Given the known returns, now improve the policy



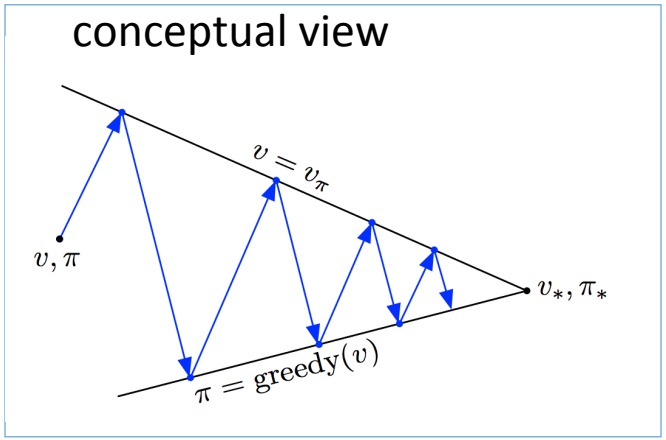
$$\pi(s) \leftarrow \arg\max \sum_{s',r} [r(s,a) + \gamma V(s')]$$

Given the known returns, now improve the policy



Terminology

- *Generalized Policy Iteration*



E: policy evaluation

$\pi \rightleftarrows V_\pi$

I: policy improvement

optimal policy,
optimal value fn

V_*, π_*

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} V_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluation}} \cdots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluation}} V_*$$

Terminology

- *state value* function V(s): $V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
  - **expected** returns for a given state, under a given policy "how good is a state?"
  - for our deterministic example, $V_\pi(s) = G_\pi(s)$
- *value iteration, with bootstrapping*:

  iterative value estimates depend on previous value estimates (!)

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots$$
$$G_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \ldots)$$
$$G_t = r_{t+1} + \gamma G_{t+1}$$

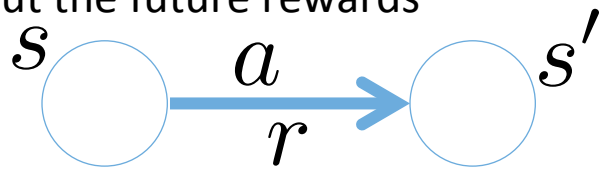
(and similarly for the value functions)

for each iteration k
  for each state s
    $V_{k+1}(s) = r + \gamma V_k(s')$

- *prioritized sweeping*: order updates based on size of expected change in V

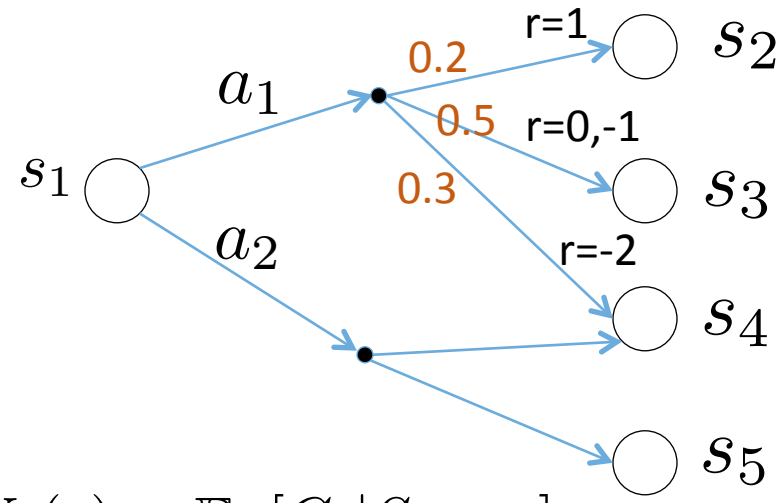
Terminology

- *discount factor* $0 \leq \gamma \leq 1$
  - allows for continuing episodes, i.e., infinite length, which would otherwise produce a potentially infinite return
  - preference for current rewards over future rewards, given that there may be more uncertainty about the future rewards
- *experience tuples* (s, a, r, s')



  - from state s, we took action a, and received reward r and ended in state s'

Stochastic Environments



$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

$$V_{\pi,k+1}(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma V_{\pi,k}(s')]$$

dynamics function: $p(s', r|s_1, a_1)$
"state transition model"

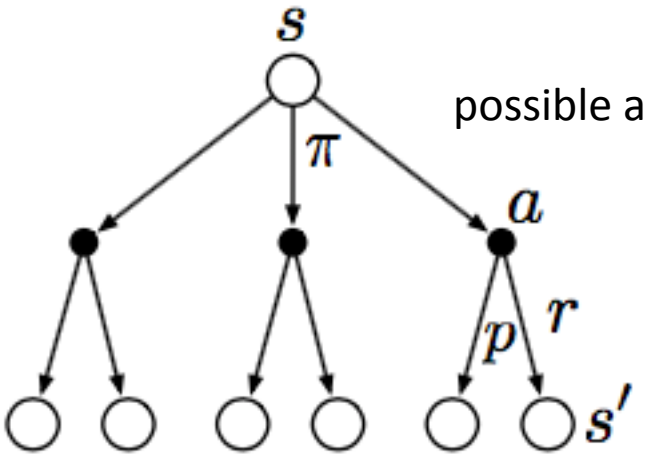
when in state s_1 and taking action a_1
what is the probability of ending up
in state s' and receiving reward r

$p(s', r\|s_1, a_1)$	next state	reward
0.2	s_2	1
0.4	s_3	0
0.1	s_3	-1
0.3	s_4	-2

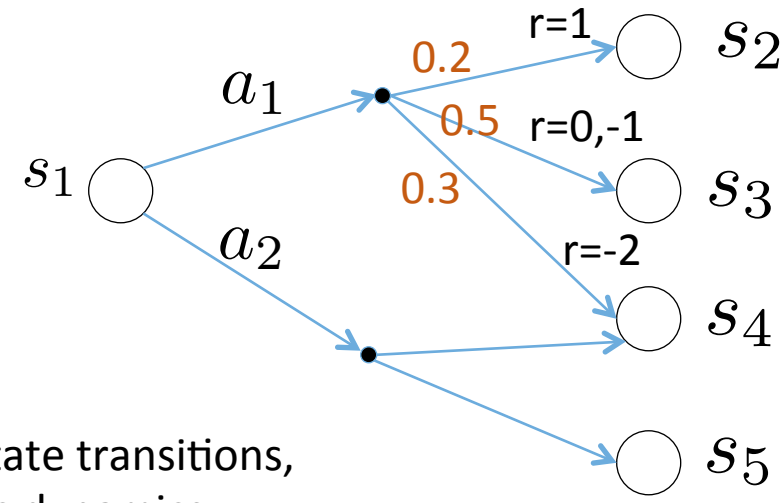
sums to 1

The value function provides the **expected returns**

Common “backup diagram”



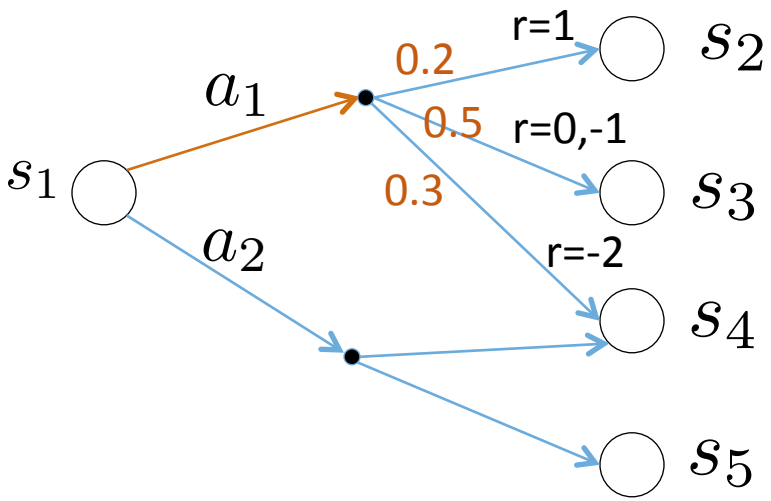
possible actions



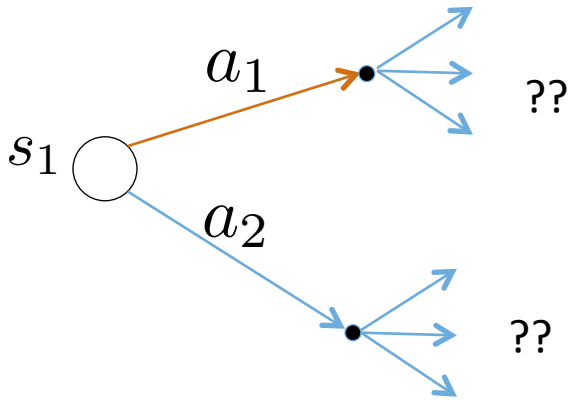
distribution of state transitions,
due to stochastic dynamics,
with unique rewards

[Sutton, p81]

Model-based



Model-free



$$V_{\pi,k+1}(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma V_{\pi,k}(s')]$$

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

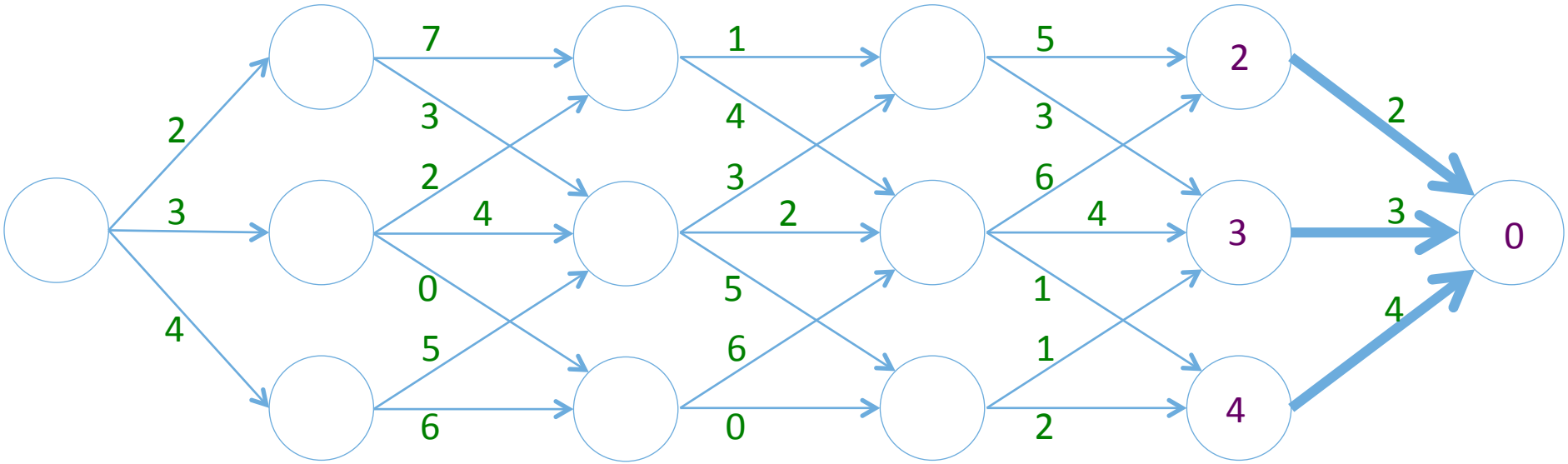
model-based: all state-transitions and rewards are known in advance, follow some known distribution

model-free: state-transitions and rewards are not known [therefore need to discover these thru experience!]

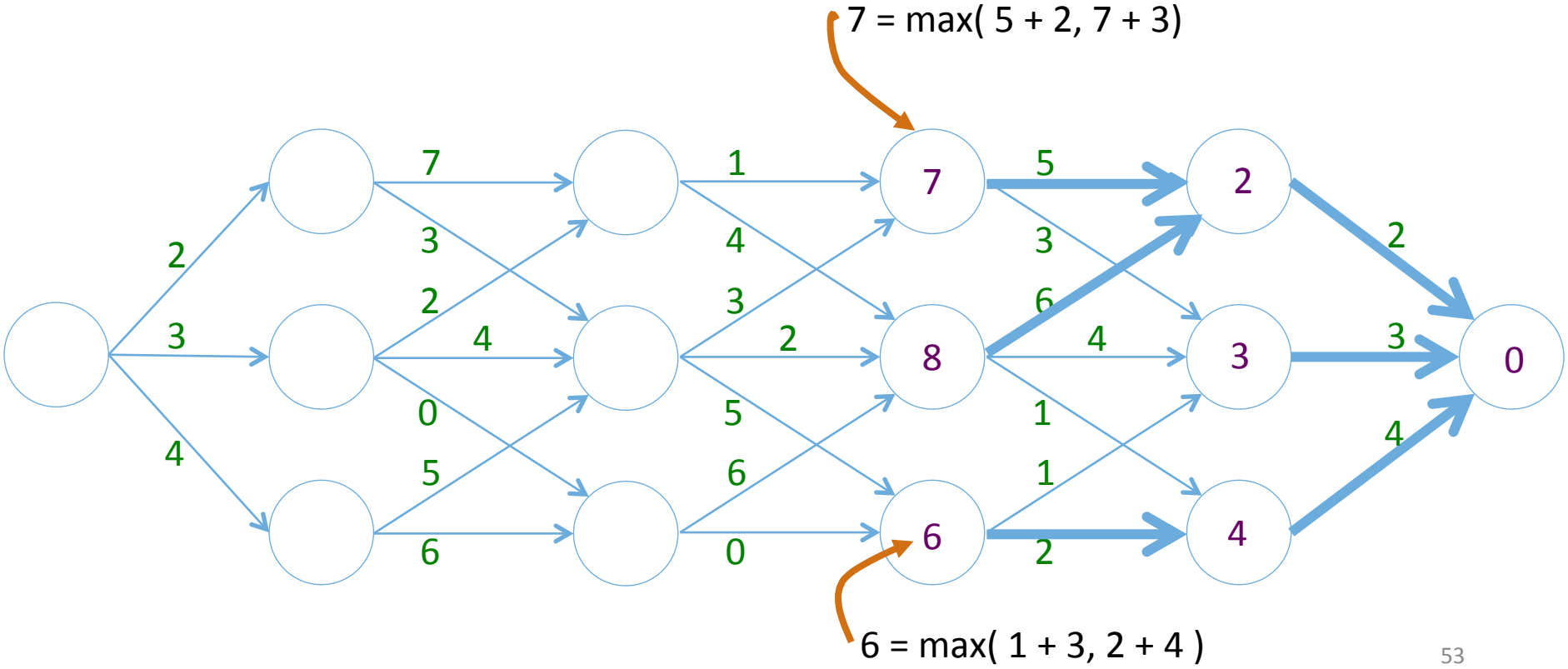
THIS TALK (and most basic RL: model-free); More advanced: learn approximate models for p()

Determining the best returns for every state

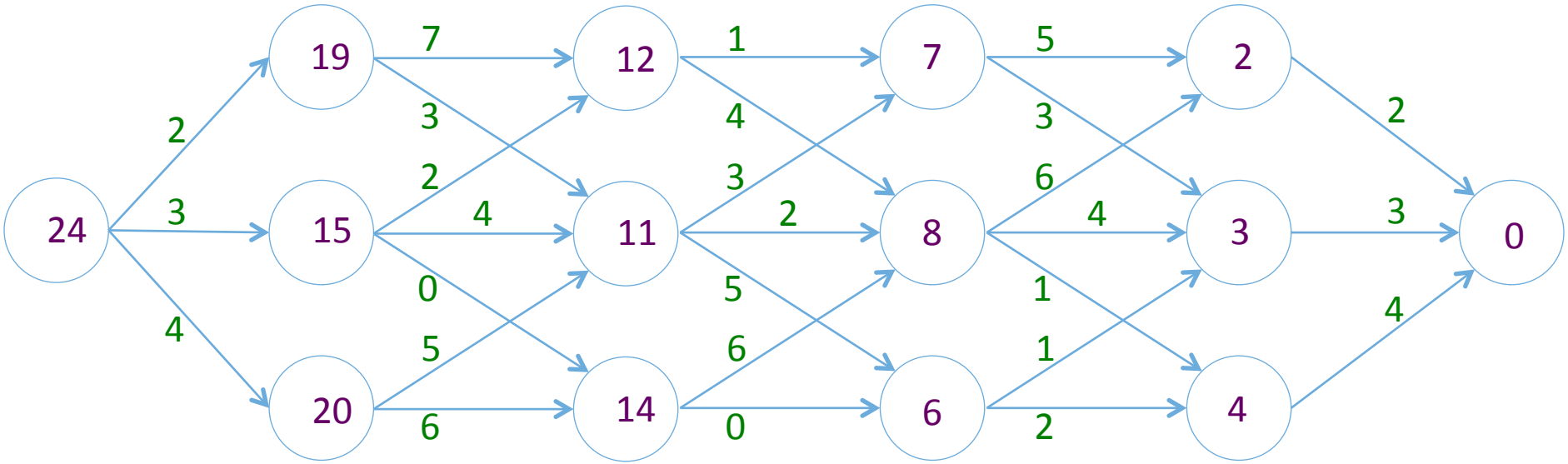
(“Dynamic Programming” because the model is known)



“Bellman Backups”



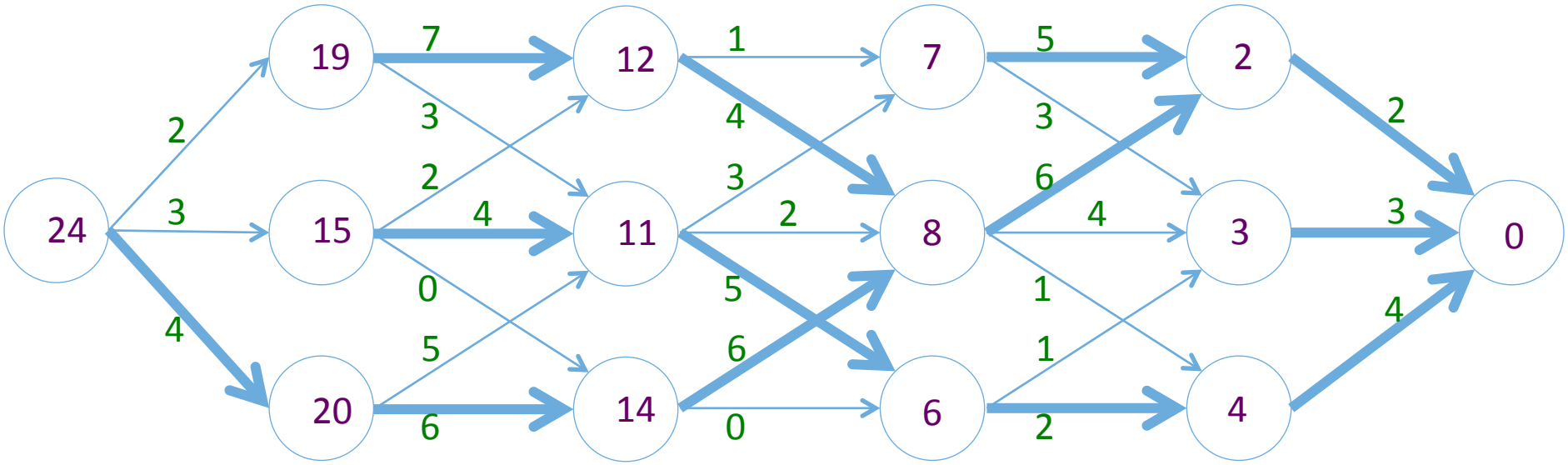
Final best returns for all states



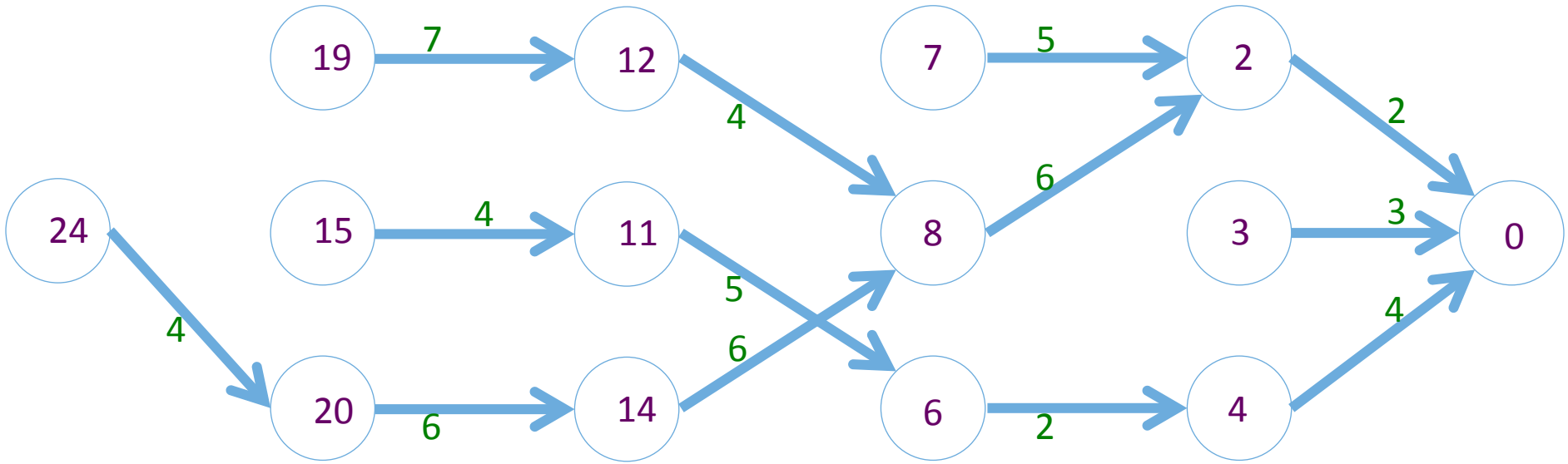
Optimal Policy (implicit in optimal value fn)

value fn update: $V_{k+1}(s) = \max_a (r + \gamma V_k(s'))$

related implicit policy: $\pi_{k+1}(s) = \arg\max_a (r + \gamma V_k(s'))$



Optimal Policy



More Definitions

- *Bellman backup*: a local iterative improvement of the value function
  - deterministic case: $V_{k+1}(s) = \max_a(r + \gamma V_k(s'))$
  - stochastic case: $V_{k+1}(s) = \max_a(\sum_{s',r} p(s', r|s, a)\,[r(s, a) + \gamma V_k(s')]$

Policy Objective Functions

(what is the overall thing that we wish to optimize?)

- episodic environments: use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- continuing environments, e.g., learning how to walk
  - average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

  - average reward per time-step (largely equivalent; we'll typically use this)

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

for a stochastic policy

~[Silver]

Value Iteration
(iterative Bellman backups until convergence)

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

[Sutton, p83]

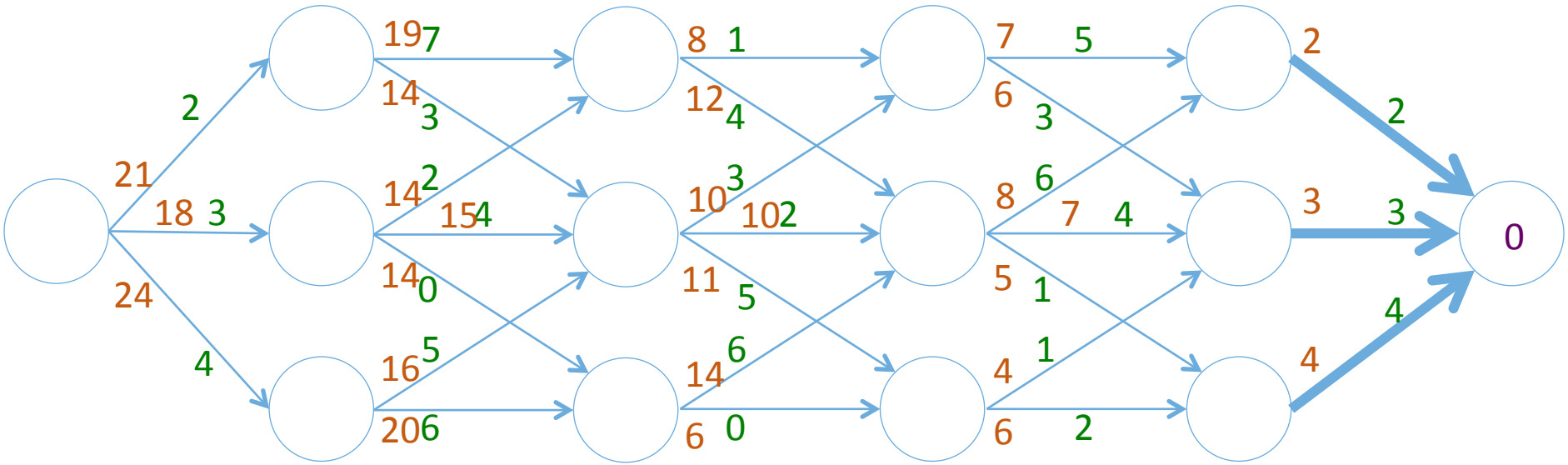
Remarks

- in general, need repeated sweeps through all states (our case: one sweep)
- dynamic programming: all states, actions are visited; dynamics known
- constructing the policy requires knowing the dynamics model, p():

But in general “model free” RL, we don’t know the model,

Learning with State-Action Values: Q(s,a)

(store cost-to-go on the edges)



$$Q_{k+1}(s,a) = r + \gamma \max_{a'}(Q_k(s',a'))$$

State-Action Value function: Q(s,a) (equivalently called "Action Value" function)

- caches the expected results of each possible action

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, a_t = a]$$

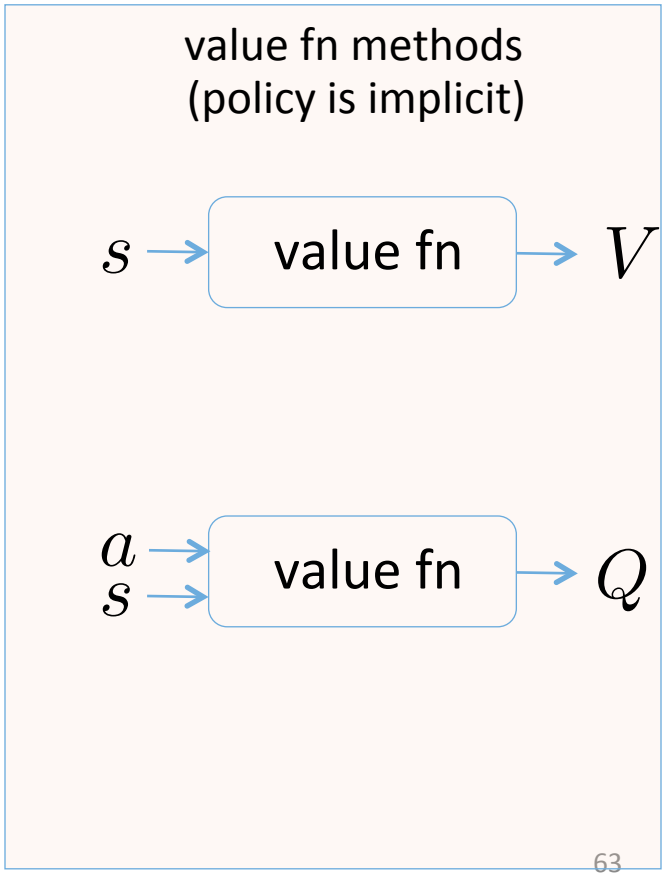
- think of having N value functions, one for the outcome of each action
- policy then becomes trivial:

$$\pi(s) = \arg\max_a(Q(s,a))$$

- Bellman backup
  - deterministic case:

$$Q_{k+1}(s,a) = r + \gamma \max_{a'}(Q_k(s',a'))$$

Q-learning
(off-policy TD version)



Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is model-free: no knowledge of MDP transitions / rewards
- TD learns from incomplete episodes, by bootstrapping
- TD updates a guess towards a guess

[Silver]

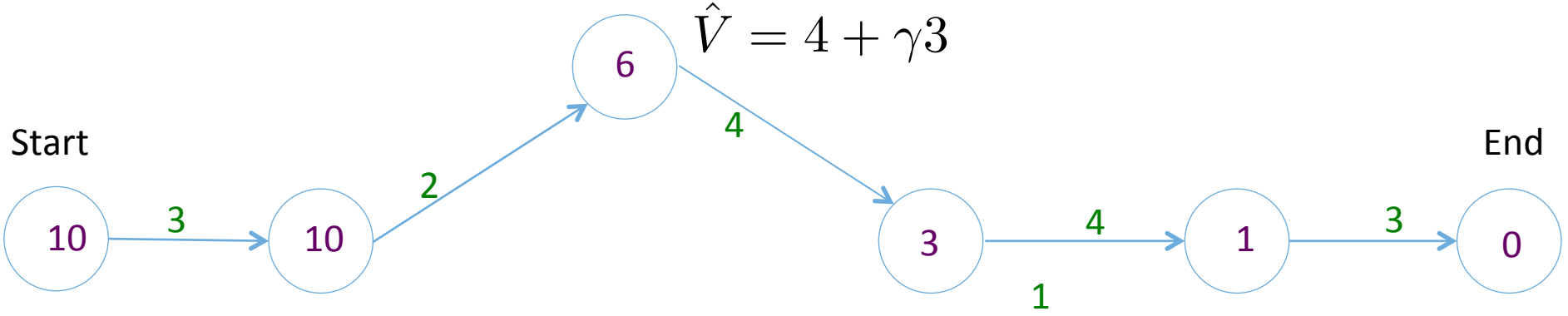
Temporal-difference estimation: TD(0)

- Given a policy, estimate its value function from episodic data

$$V(S_t) \leftarrow V(S_t) + \alpha(r_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

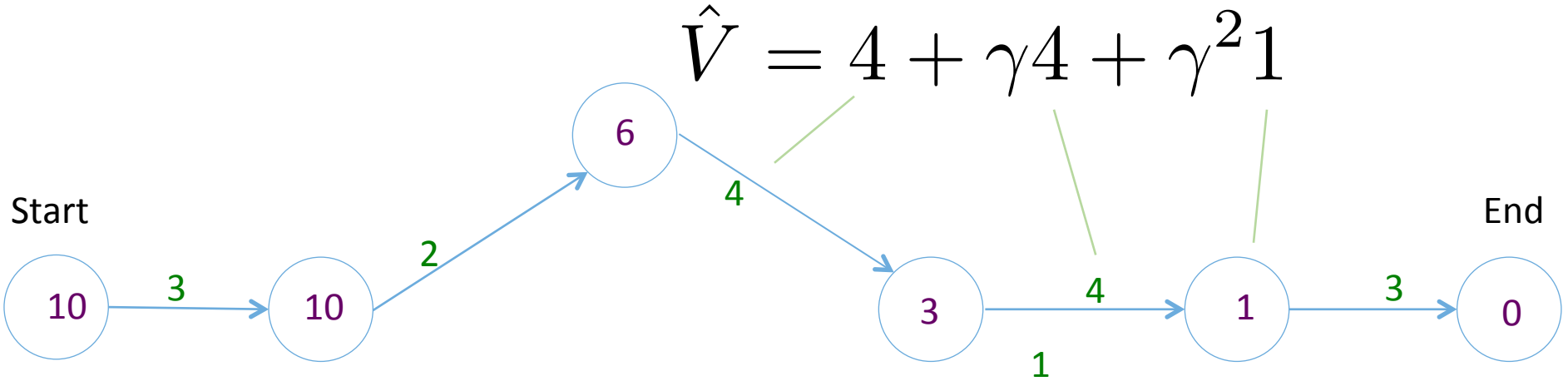
target

δ temporal difference



if model is known: $V(s) = \sum p(s', r|s, a)\,[r(s, a) + \gamma V(s')]$

n-step Temporal Differences



n-step Temporal Differences, all the way to Monte Carlo estimation

Let's label the estimated return following n steps as $G_t^{(n)}, n = 1, \dots, \infty$, then:

n	G_t	**Notes**
$n = 1$	$G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$	TD learning
$n = 2$	$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$	
...		
$n = n$	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$	
...		
$n = \infty$	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T + \gamma^{T-t} V(S_T)$	MC estimation

The generalized n-step TD learning still has the same form for updating the value function:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$

[lillianweng]

Temporal-difference for estimating Q(s,a)

$$Q(s_t, a_t) \leftarrow Q(s, a) + \alpha[\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)}_{\text{target}} - Q(s_t, a_t)]$$

δ temporal difference

Q Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

[Sutton, p131]

Off-policy vs On-policy learning

- On-policy: Use the deterministic outcomes or samples from the target policy to train the algorithm.
- Off-policy: Training on a distribution of transitions or episodes produced by a different behavior policy rather than that produced by the target policy. For example, the actions could come from another agent.

- Q-learning allows for off-policy learning

ϵ -Greedy Exploration (for discrete actions)

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability 1 − ε choose the greedy action
- With probability ε choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

[Silver]

Scalability

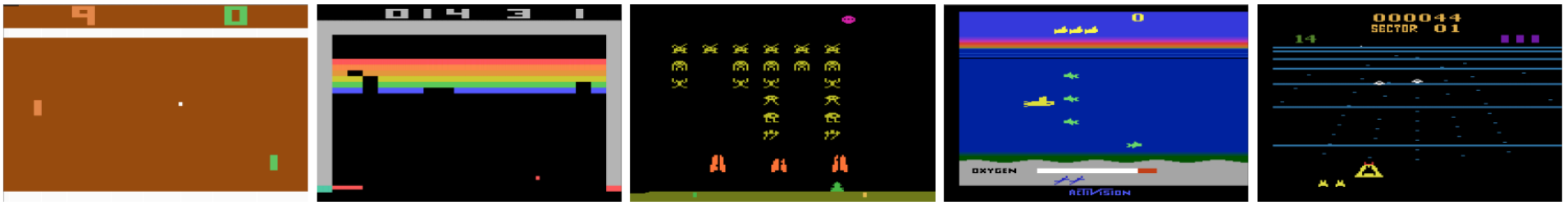
- simple problems
  - discrete states, discrete actions: store Q(s,a) in tabular form
- more interesting problems
  - Backgammon: 10^20 states
  - Go: 10^170 states
- continuous states and continuous actions ?
  - "curse of dimensionality"
- therefore need to approximate:
  - value functions and/or policy

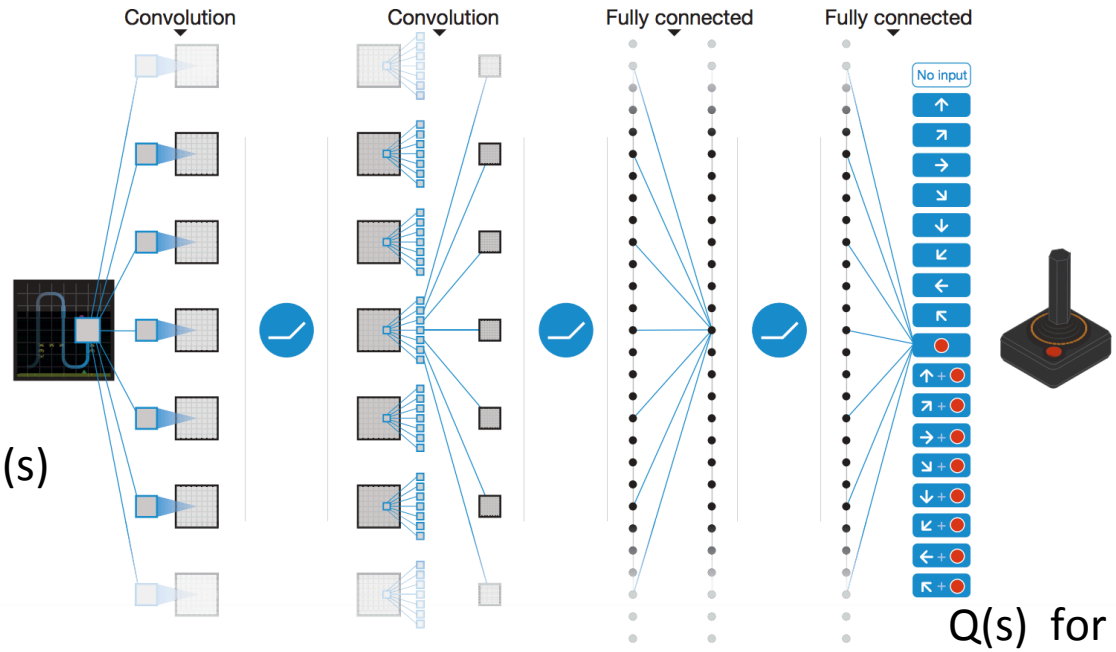
Deep Q-Networks (DQN)

Q-learning for discrete actions, off policy

Playing Atari with Deep Reinforcement Learning [NeurIPS 2013]

Human-level control through deep reinforcement learning [Nature 2015]





game image(s)

Q(s) for each action

[awjuliani]

DQN Algorithm

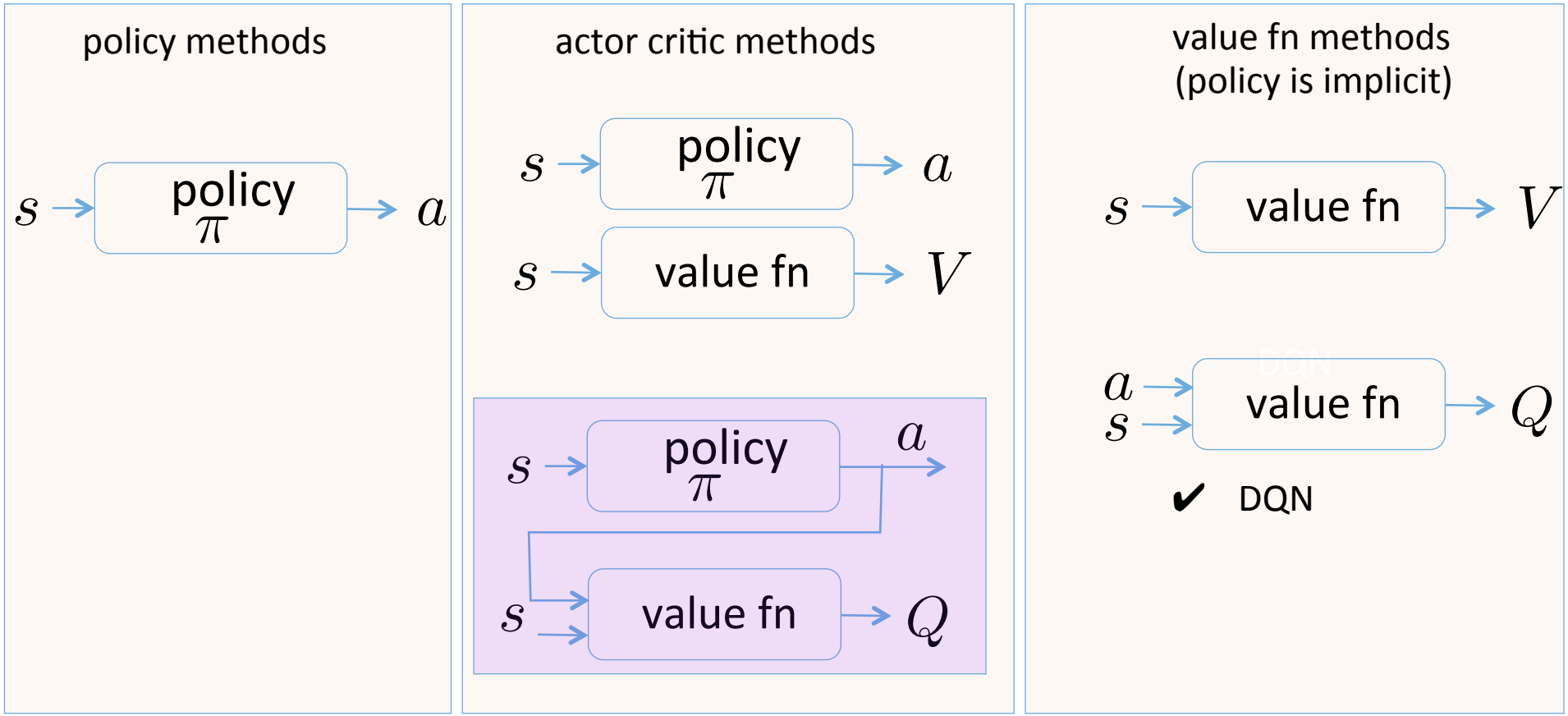
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

sample tuples from an "experience replay" buffer

maintain old and new Q networks; then periodically swap these

[Mnih et al., 2015]

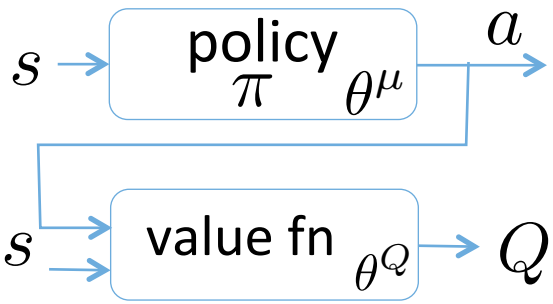
Deep RL: key building blocks



DDPG: Deep Deterministic Policy Gradients

DDPG

Q-learning for continuous actions, off-policy



key to updating the policy – chain rule:

$$\frac{\partial Q}{\partial \theta^\mu} = \frac{\partial Q}{\partial a}\frac{a}{\partial \theta^\mu}$$

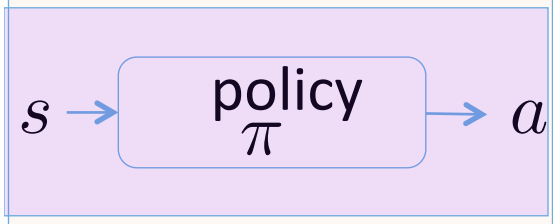
The DPG algorithm maintains a parameterized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action. The critic $Q(s, a)$ is learned using the Bellman equation as in Q-learning. The actor is updated by following the applying the chain rule to the expected return from the start distribution J with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned} \tag{6}$$

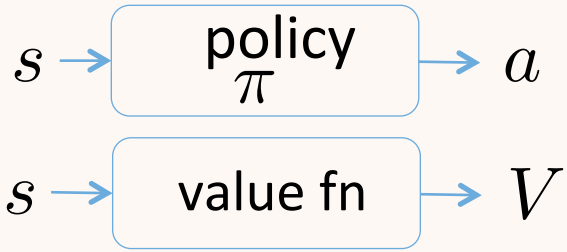
Be aware that there are other important details to get right. [Lillicrap et al. 2016]

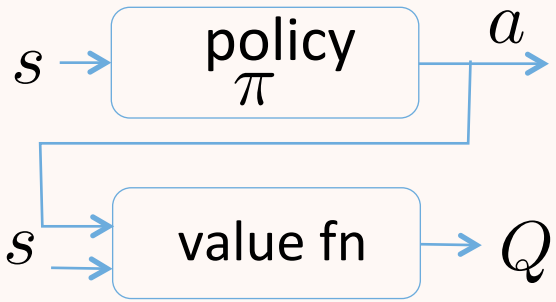
Gradient-Free Methods (Policy Search)

policy methods



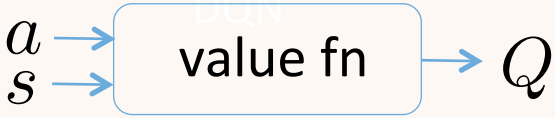
actor critic methods





✔ DDPG

value fn methods
(policy is implicit)



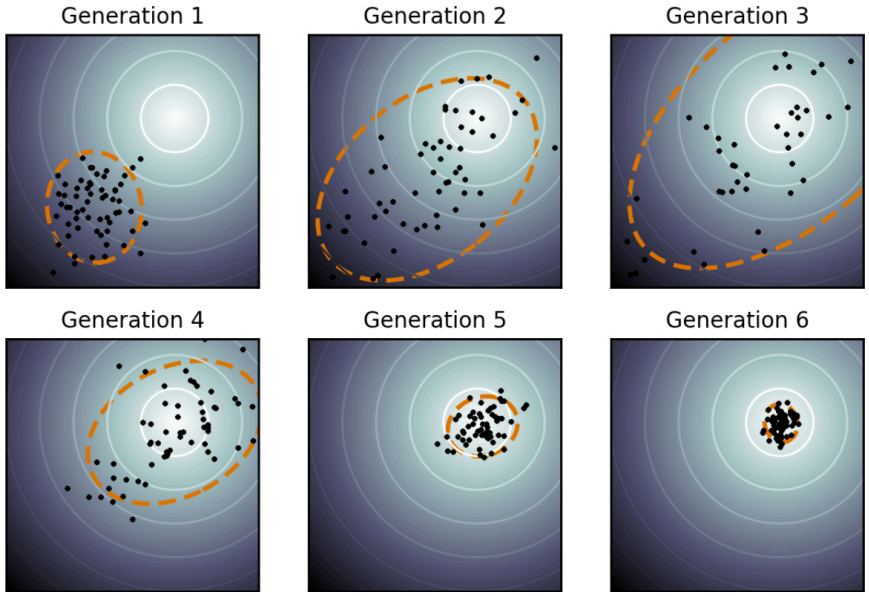
✔ DQN

Gradient Free Methods:

CEM: Cross Entropy Method
CMA: Covariance Matrix Adaptation

- Treat the policy parameters directly as defining a high-dimensional search space.
- Model the space of candidate solutions for a given generation using a Normal (or multivariate Gaussian) distribution in this space.
- Evaluate a sample population for their fitness, using 1+ episodes per candidate.
- Keep a set of the best ("elite") samples.
- Move the distribution to fit those elite samples, i.e., maximum likelihood
- repeat for the next generation

CMA
[Wikipedia]



Cross-Entropy Method

$$\max_{\theta} U(\theta) = \max_{\theta} \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

- Views *U* as a black box
- Ignores all other information other than *U* collected during episode

= evolutionary algorithm

population: $P_{\mu^{(i)}}(\theta)$

CEM:
for iter i = 1, 2, …
  for population member e = 1, 2, …
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\pi_{\theta^{(e)}}$
    store $(\theta^{(e)}, U(e))$
  endfor
  $\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})$
    where $\bar{e}$ indexes over top p %
endfor

[Abbeel]

Cross-Entropy Method

- Can work embarrassingly well

Method	Mean Score	Reference
Nonreinforcement learning		
Hand-coded	631,167	Dellacherie (Fahey, 2003)
Genetic algorithm	586,103	(Böhm et al., 2004)
Reinforcement learning		
Relational reinforcement learning+kernel-based regression	≈50	Ramon and Driessens (2004)
Policy iteration	3183	Bertsekas and Tsitsiklis (1996)
Least squares policy iteration	<3000	Lagoudakis, Parr, and Littman (2002)
Linear programming + Bootstrap	4274	Farias and van Roy (2006)
Natural policy gradient	≈6800	Kakade (2001)
CE+RL	21,252	
CE+RL, constant noise	72,705	
CE+RL, decreasing noise	348,895	

István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941

Approximate Dynamic Programming Finally Performs Well in the Game of Tetris

Victor Gabillon
INRIA Lille - Nord Europe, Team SequeL, FRANCE
victor.gabillon@inria.fr

Mohammad Ghavamzadeh*
INRIA Lille - Team SequeL & Adobe Research
mohammad.ghavamzadeh@inria.fr

Bruno Scherrer
INRIA Nancy - Grand Est, Team Maia, FRANCE
bruno.scherrer@inria.fr

[NIPS 2013]

Closely Related Approaches

CEM:
for iter i = 1, 2, …
for population member e = 1, 2, …
sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
execute roll-outs under $\pi_{\theta^{(e)}}$
store $(\theta^{(e)}, U(e))$
endfor
$\mu^{(i+1)} = \arg\max_{\mu} \sum_{\bar{e}} \log P_{\mu}(\theta^{(\bar{e})})$
where $\bar{e}$ indexes over top p %
endfor

[Abbeel]

- Reward Weighted Regression (RWR)
  - Dayan & Hinton, NC 1997; Peters & Schaal, ICML 2007

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} q(U(e), P_{\mu}(\theta^{(e)})) \log P_{\mu}(\theta^{(e)})$$

- Policy Improvement with Path Integrals (PI^2)
  - PI2: Theodorou, Buchli, Schaal JMLR2010; Kappen, 2007; (PI2-CMA: Stulp & Sigaud ICML2012)

$$\mu^{(i+1)} = \arg\max_{\mu} \sum_{e} \exp(\lambda U(e)) \log P_{\mu}(\theta^{(e)})$$

- Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)
  - CMA: Hansen & Ostermeier 1996; (CMA-ES: Hansen, Muller, Koumoutsakos 2003)

$$(\mu^{(i+1)}, \Sigma^{(i+1)}) = \arg\max_{\mu,\Sigma} \sum_{\bar{e}} w(U(\bar{e})) \log \mathcal{N}(\theta^{(\bar{e})}; \mu, \Sigma)$$

- PoWER
  - Kober & Peters, NIPS 2007 (also applies importance sampling for sample re-use)

$$\mu^{(i+1)} = \mu^{(i)} + \left(\sum_{e} (\theta^{(e)} - \mu^{(i)}) U(e) \right) / \left(\sum_{e} U(e) \right)$$

Cross-Entropy / Evolutionary Methods

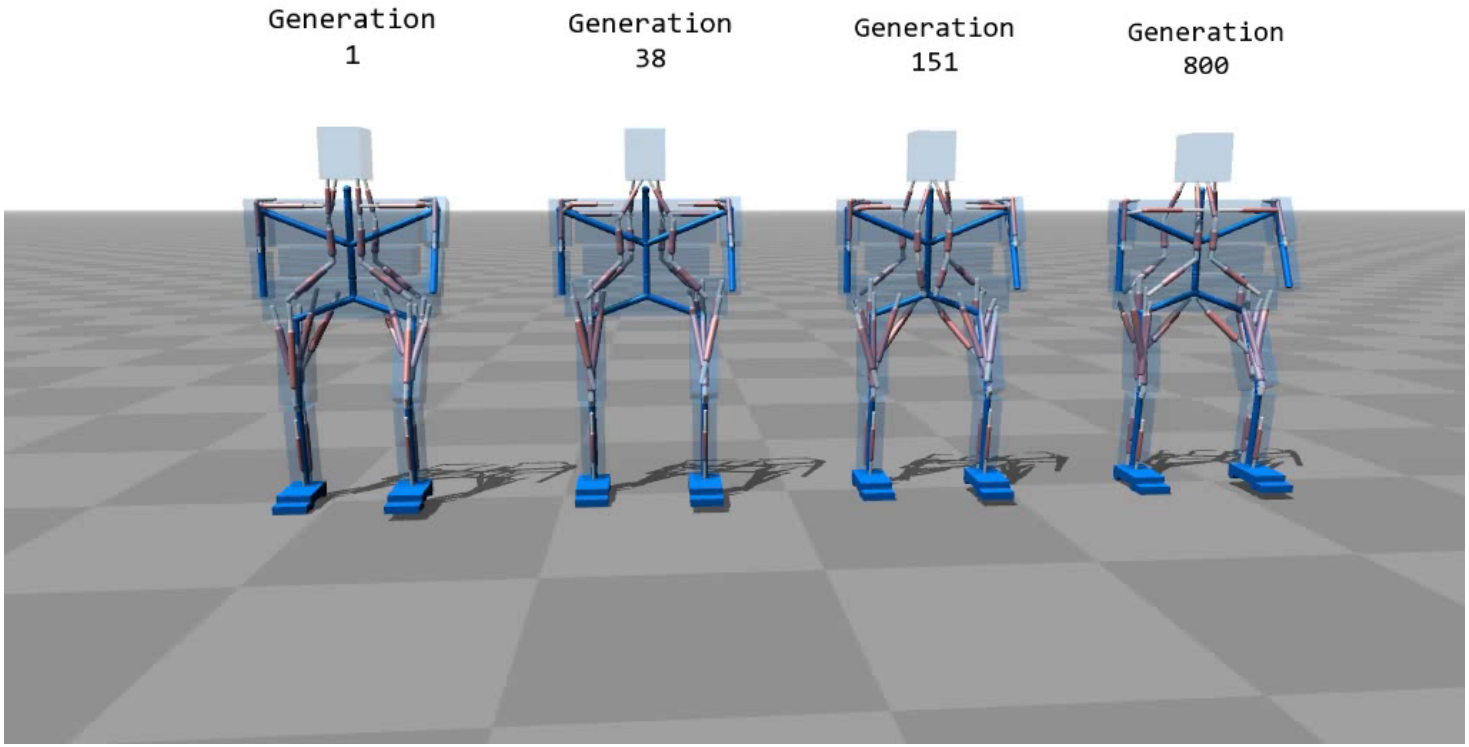
- Full episode evaluation, parameter perturbation
- Simple
- Main caveat: best when intrinsic dimensionality not too high
  - i.e., number of population members comparable to or larger than number of (effective) parameters

→ in practice OK if low-dimensional θ and willing to do do many runs

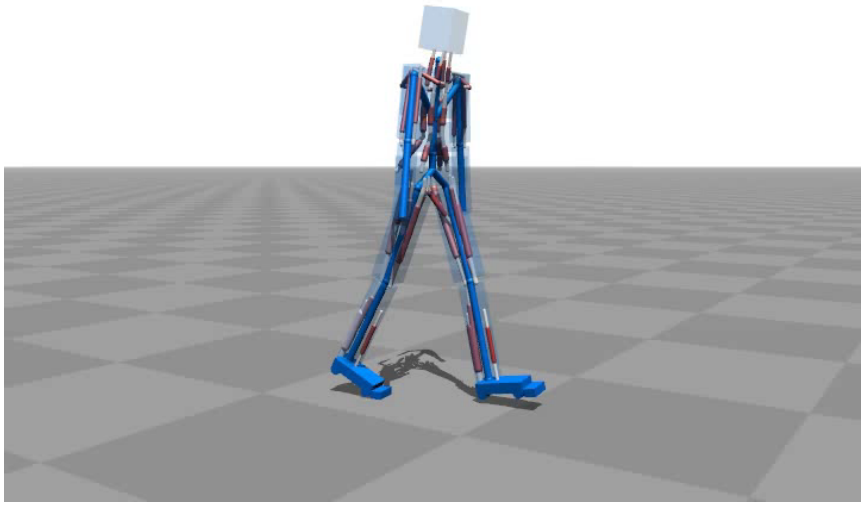
→ Easy-to-implement baseline, great for comparisons!

[Abbeel]

Biomechanical Locomotion of Imaginary Creatures



speeds



Considerations

- Pros:
  - Work with arbitrary parameterization, even non-differentiable
  - Embarrassingly easy to parallelize
- Cons:
  - Not very sample efficient since ignores temporal structure

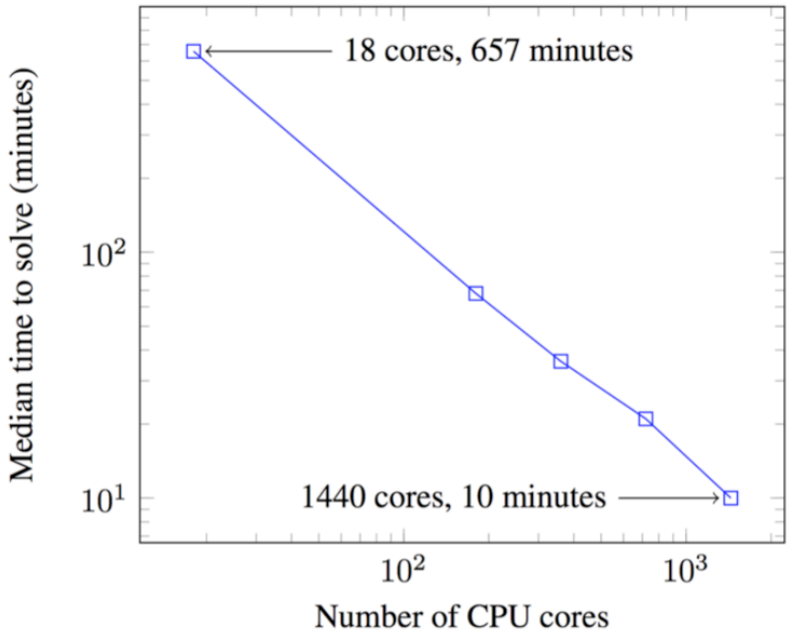
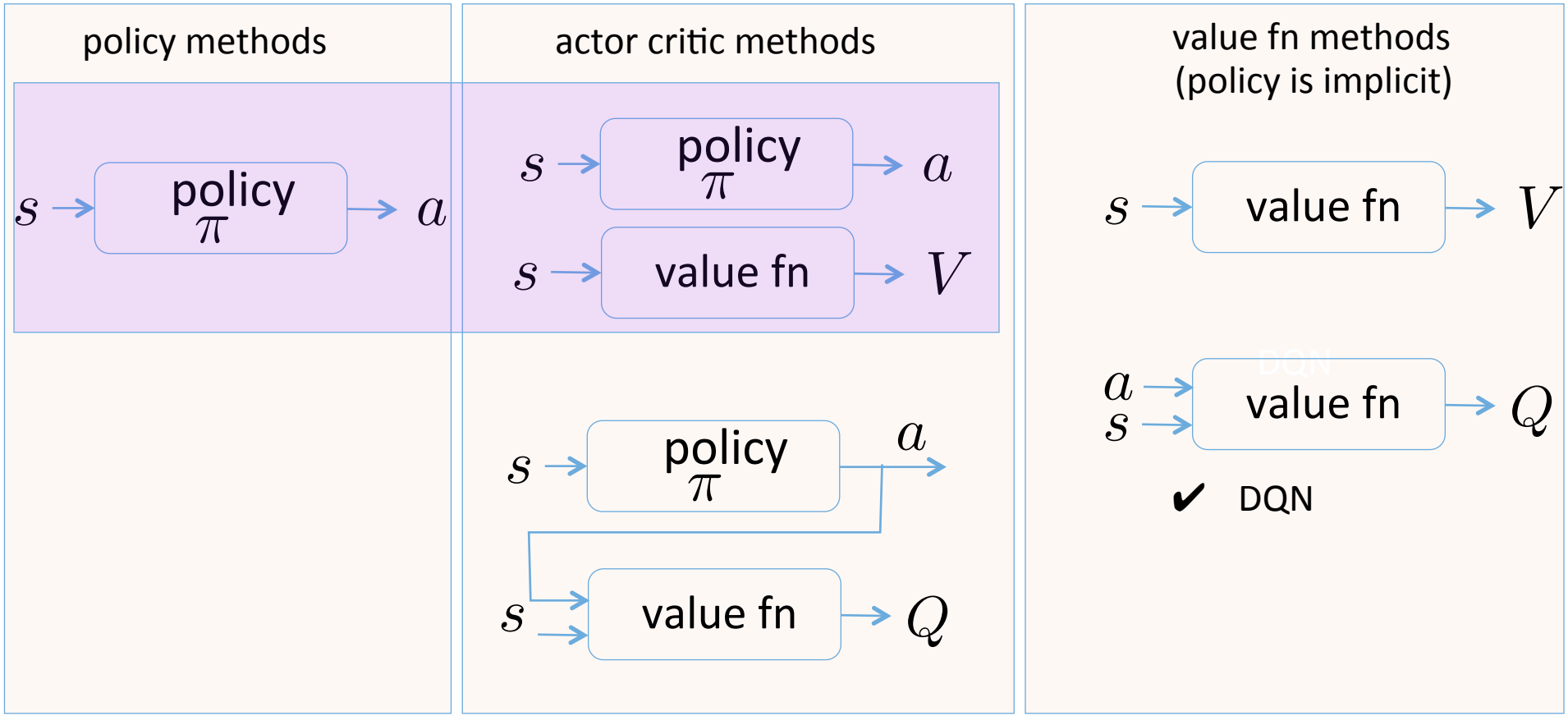


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

[Salimans, Ho, Chen, Sutskever, 2017]

Policy Gradient Methods



Policy Gradient Algorithms

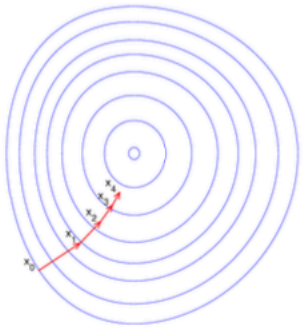
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a *local* maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

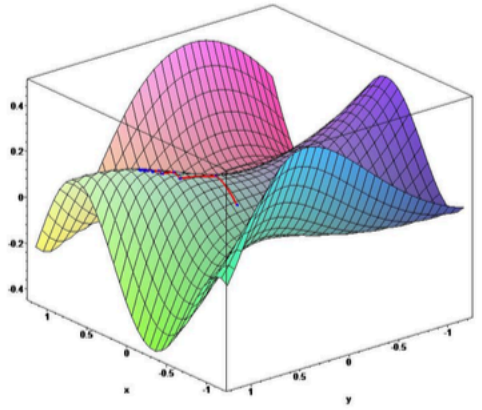
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- Where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter





[Abbeel]

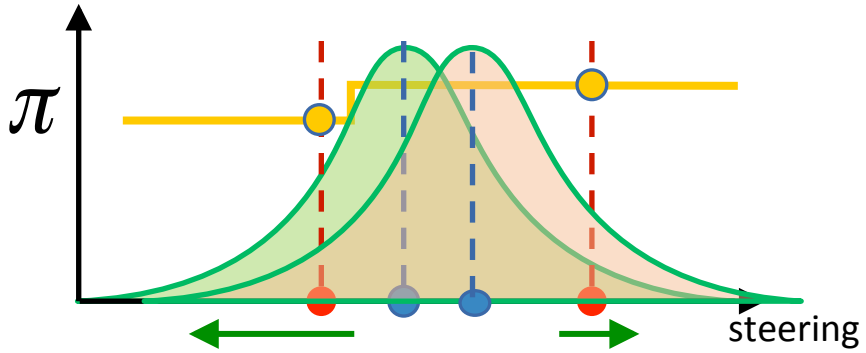
Computing the Policy Gradient via finite-differences

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- very slow
  - one gradient computation requires n+1 policy evaluations for n policy parameters
- better idea: get a (noisy) policy gradient for each policy time step!

The intuition:
Consider a one-step reward problem

"make the sampled rewards more likely"

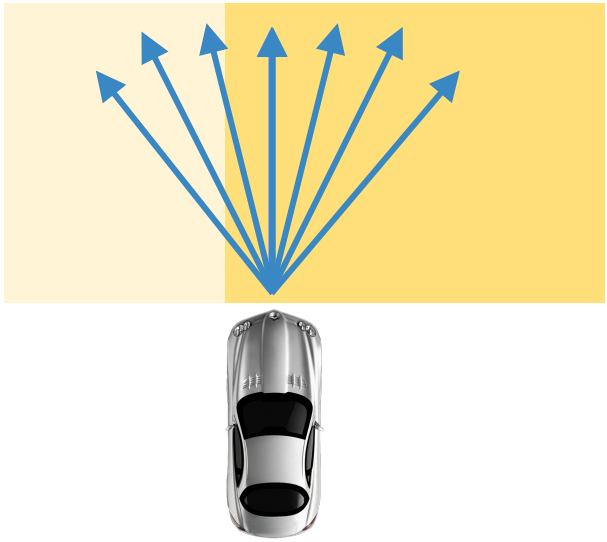


$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]$$

gradient vectors (score function)

rewards

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

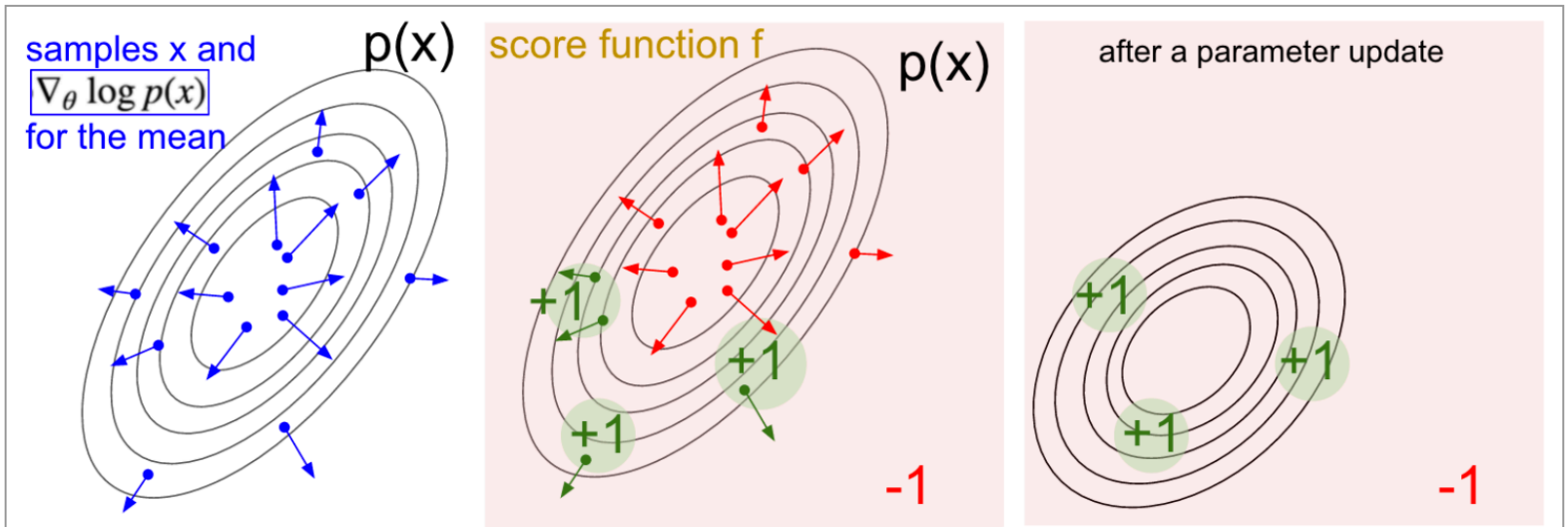


Intuition in 2D

- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\nabla_\theta \mu_\theta(s)}{\sigma^2}$$

for a symmetric Gaussian, the blue arrows are just radial vectors from the mean action to the sampled action



The math: Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \nabla_\theta \left[\sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \pi_\theta(a|s) \right]$$

$$\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

this step uses Likelihood ratios – see below

$$\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a)\, \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} Q^\pi(s,a) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$

- Likelihood ratios exploit the following identity

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$
$$= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$$

$$\frac{d}{dx}\left(\ln(x)\right) = \frac{1}{x}$$

$$\frac{d}{dx}\left(\ln f(x)\right) = \frac{f'(x)}{f(x)}$$

Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

Many variations of policy gradients !!

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}. \tag{3}$$

[Schulman 2016]

Actor-Critic Policy Gradient Algorithm

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

What's in a step-size?

- Terrible step sizes, always an issue, but how about just not so great ones?

- Supervised learning
  - Step too far → next update will correct for it
- Reinforcement learning
  - Step too far → terrible policy
  - Next mini-batch: collected under this terrible policy!
  - Not clear how to recover short of going back and shrinking the step size

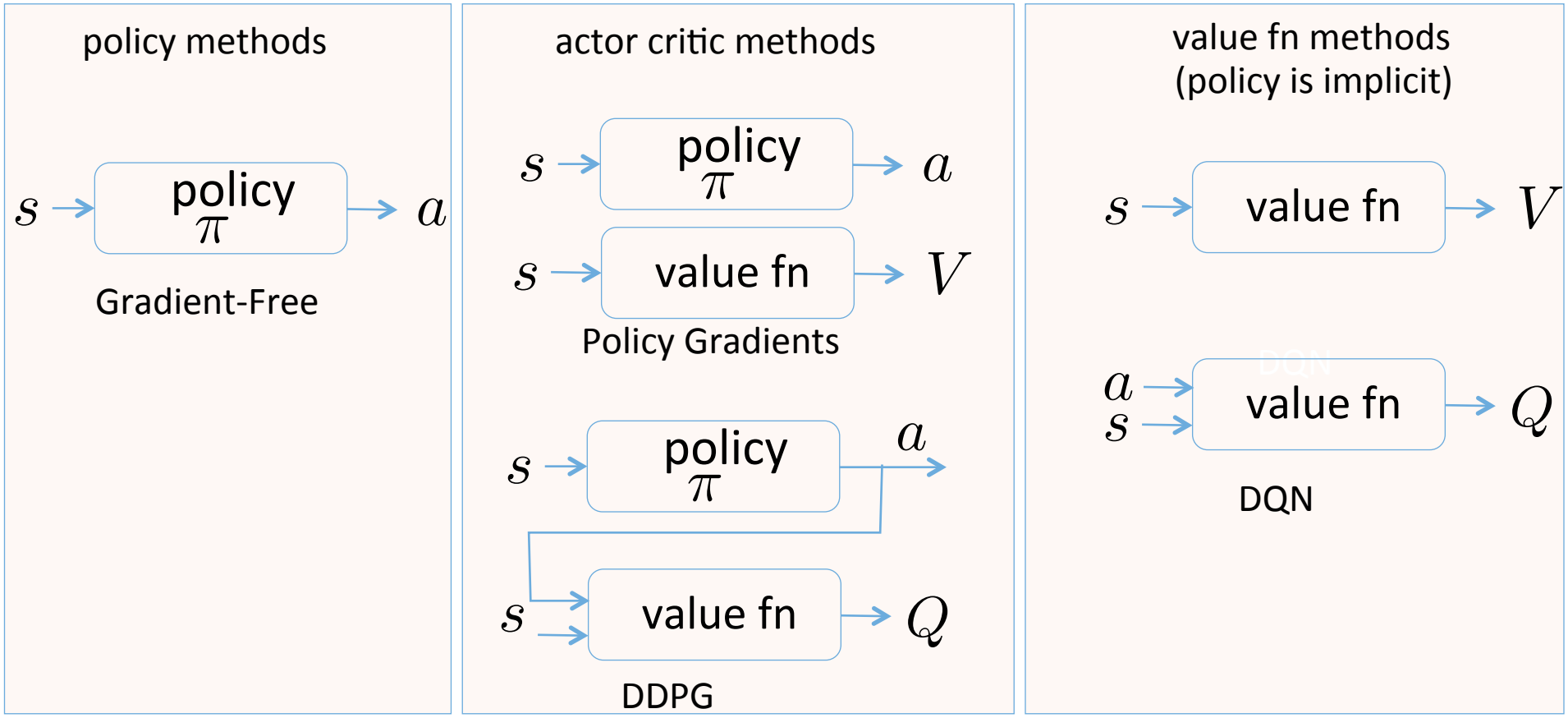
[Abbeel]

The following methods propose ways of avoiding unsafe step sizes:

TRPO: Trust-Region Policy Optimization

PPO: Proximal Policy Optimization

SUMMARY


policy methods
s
policy π
a
Gradient-Free
actor critic methods
s
policy π
a
s
value fn
V
Policy Gradients
s
policy π
a
s
value fn
Q
DDPG
value fn methods
(policy is implicit)
s
value fn
V
a
s
value fn
Q
DQN

Practical Advice for working with RL

- Techniques from supervised learning don't necessarily work in RL:
  - batch norm, dropout, big networks
- use small test problems; experiment quickly
- interpret and visualize the learning process: state visitation, value fn
- new task? make it easier until there are signs of life
- env design: visualize random policy
- explore sensitivity to parameters
- use multiple random # seeds
- automate experiments; consider using cloud computing
- much more at the link below

~[Schulman https://drive.google.com/file/d/0BxXI_RttTZAhc2ZsblNvUHhGZDA/view]

Open Directions in RL

- sim-to-real
- sparse rewards: curiousity-driven learning
- safe RL
- deep RL from human preferences; inverse RL
- learning efficiency
  - meta learning
  - ubiquitous transfer learning: never learn from scratch
  - learning from demonstrations
  - hierarchical RL
  - combining with model-predictive control
  - inductive biases e.g., convolutional networks
  - see, e.g.,: “Reinforcement Learning, Fast and Slow”

Questions?