CPSC 533V: Learning to Move

Reinforcement Learning Lectures

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:

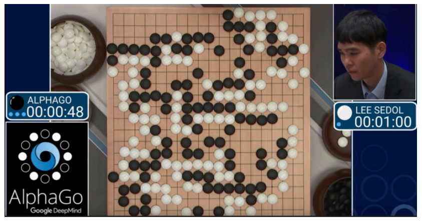
- Rich Sutton, U Alberta [Sutton] http://incompleteideas.net/book/the-book-2nd.html
- Emma Brunskill, Stanford [Brunskill] https://web.stanford.edu/class/cs234/index.html
- David Silver, UCL [Silver] http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Pieter Abbeel, UC Berkeley [Abbeel]
- Sergey 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, DDPG, gradient-free methods, policy-gradient methods
- current perspectives

What is RL?



[DeepMind]

[Mobileye]

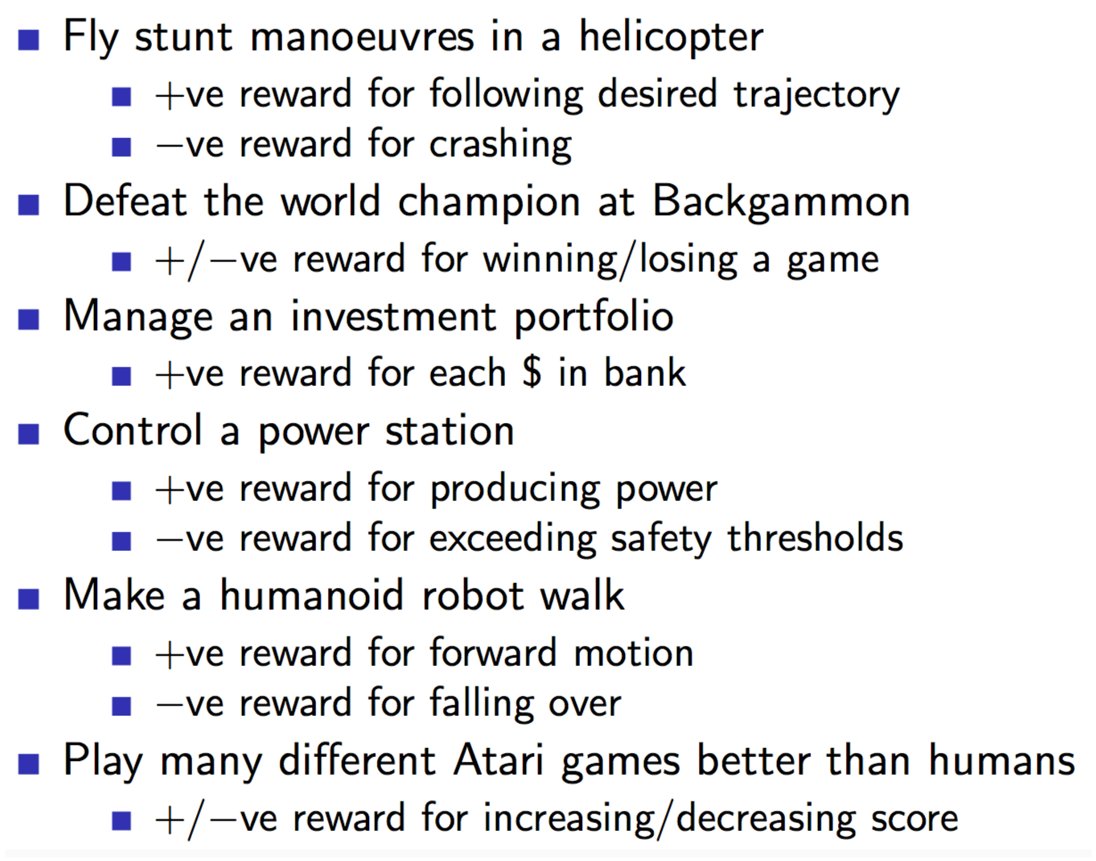
[Anybotics]

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

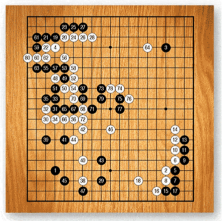
Defining good decisions: rewards



- 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]

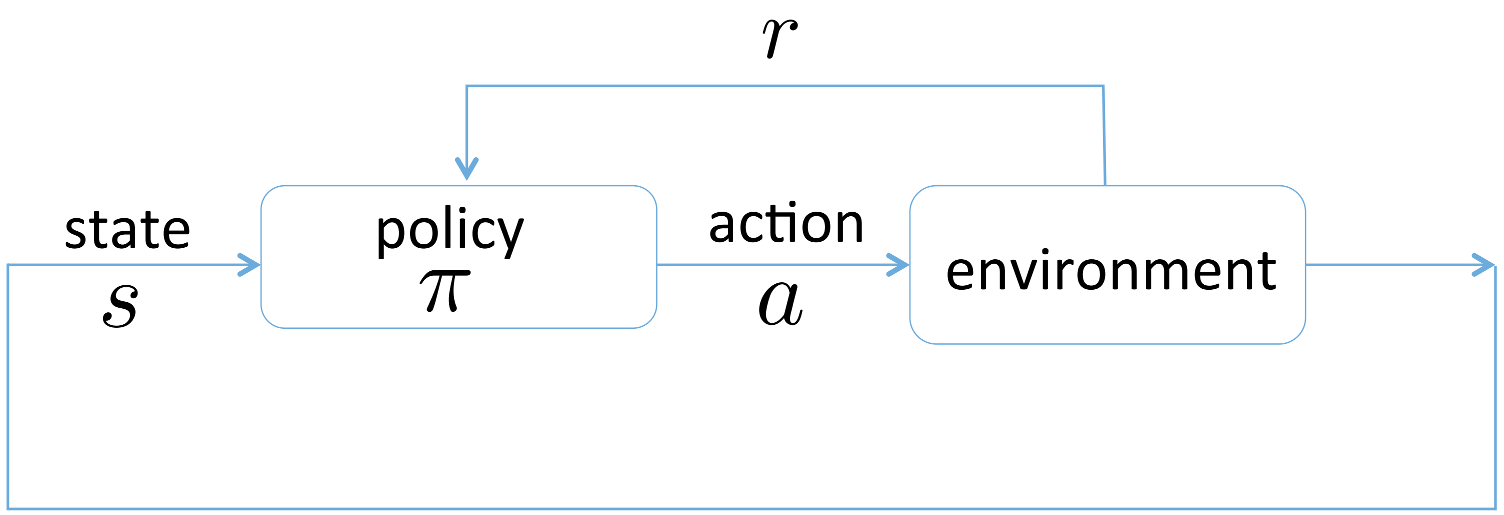
RL involves:



- 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



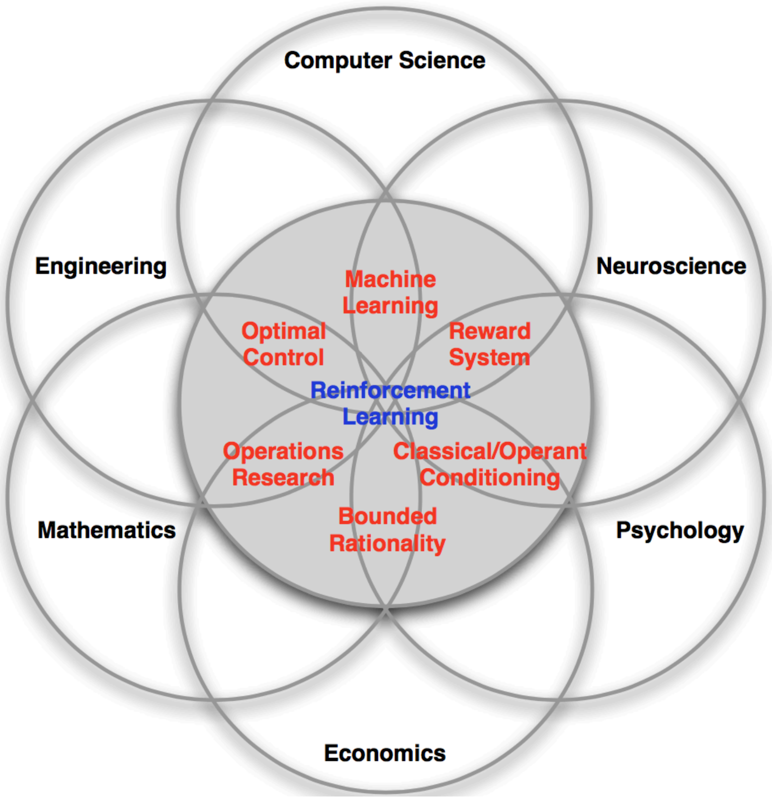
maximize sum of rewards: $G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

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

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

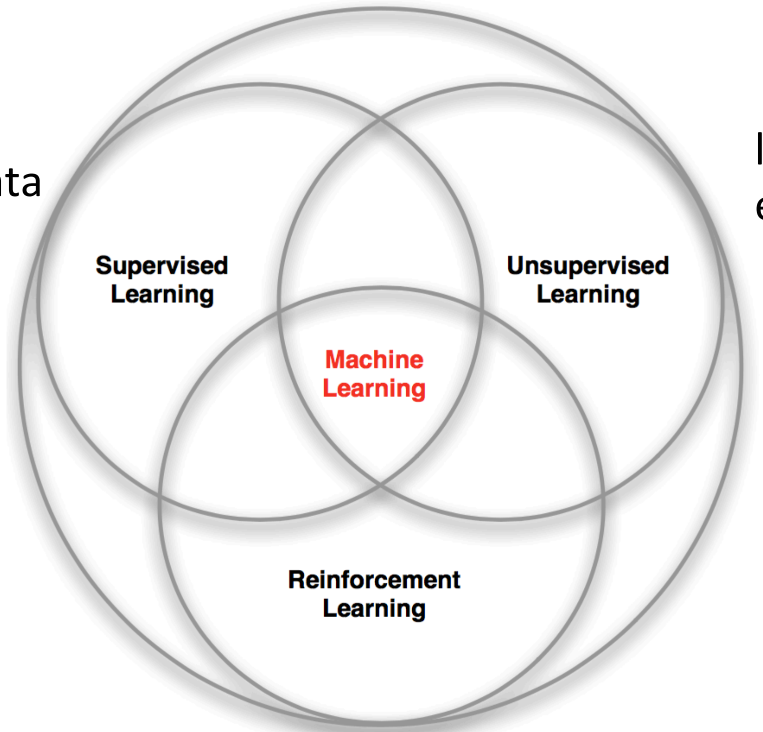


[Silver]

Branches of Machine Learning

labeled data

latent structure ,
e.g., autoencoders



scalar rewards over time

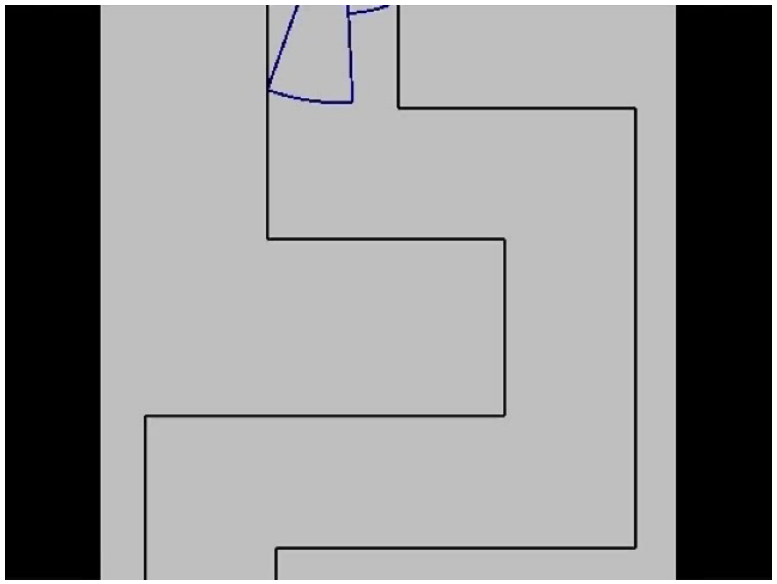
[Silver]

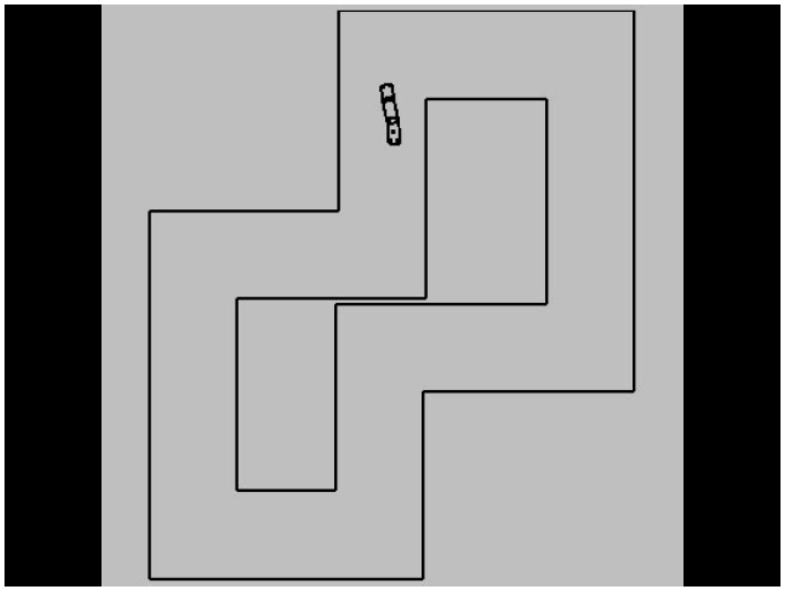
Simple Example: Backing up a Truck

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

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

backing up a double trailer

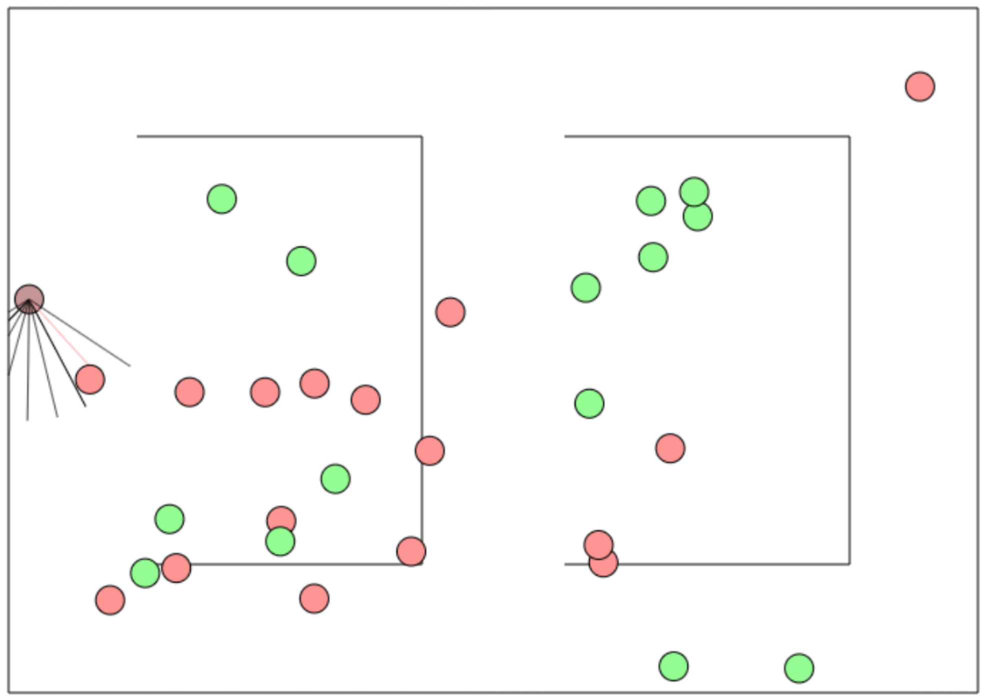




[“Learning to Steer on Winding Tracks Using Semi-Parametric Control Policies”, ICRA 2005]

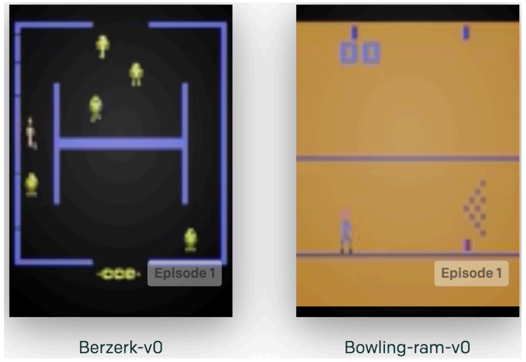
Simple Example

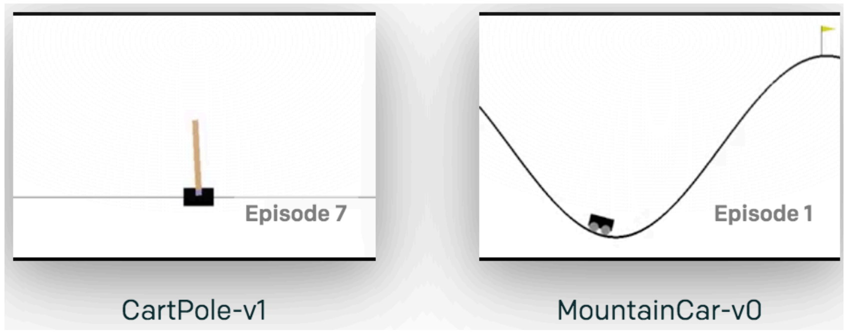
state: 9 sensors x 3 values x 2 timesteps
actions: move in 5 directions



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

OpenAI Gym [gym.openai.com]

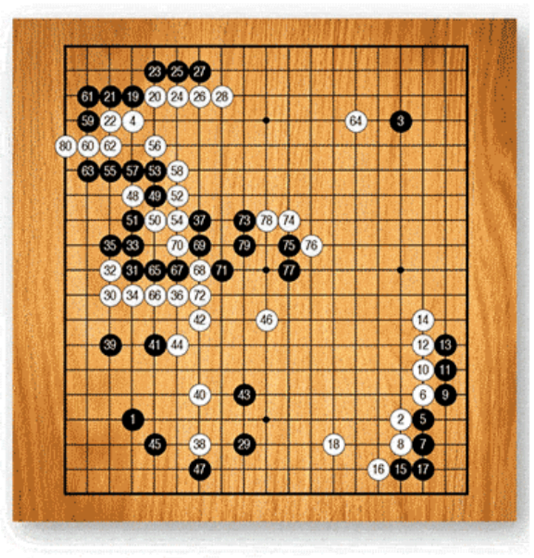




Episode 1
Episode 1
Episode 3
Ant-v2
HalfCheetah-v2
Hopper-v2

Why Deep RL?

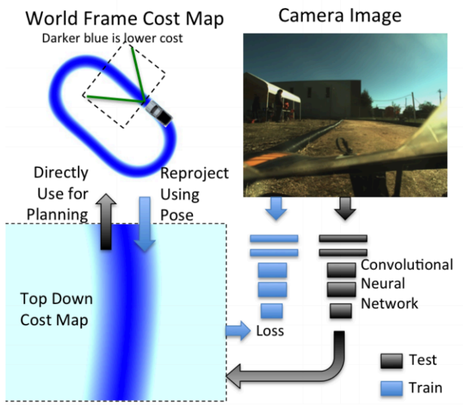
AlphaZero (DeepMind)



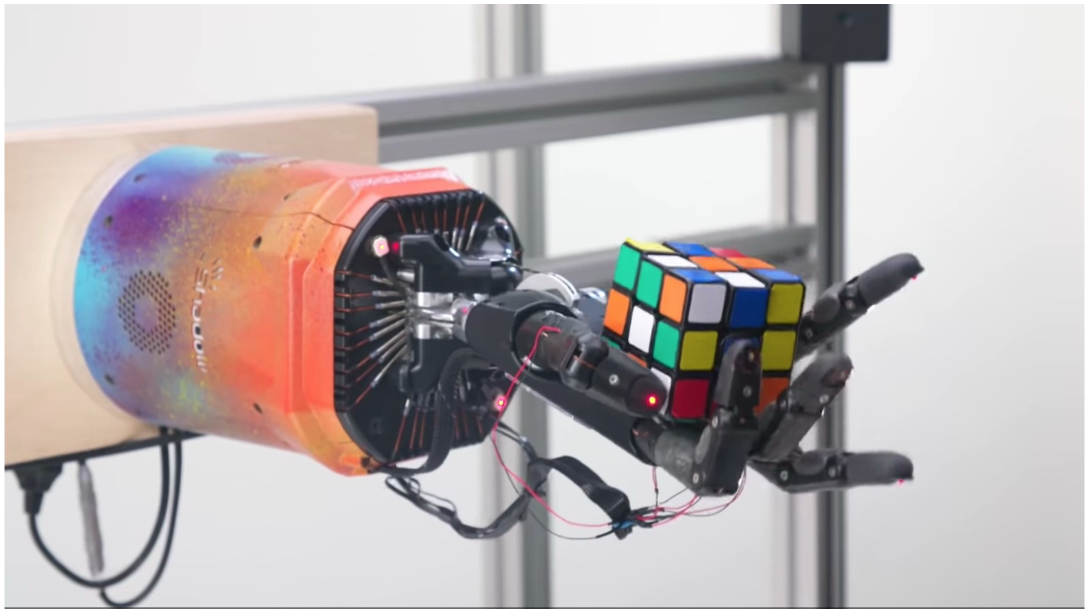
[DeepMind: Nature 2017]

Aggressive autonomous driving of a scale vehicle (Georgia Tech)

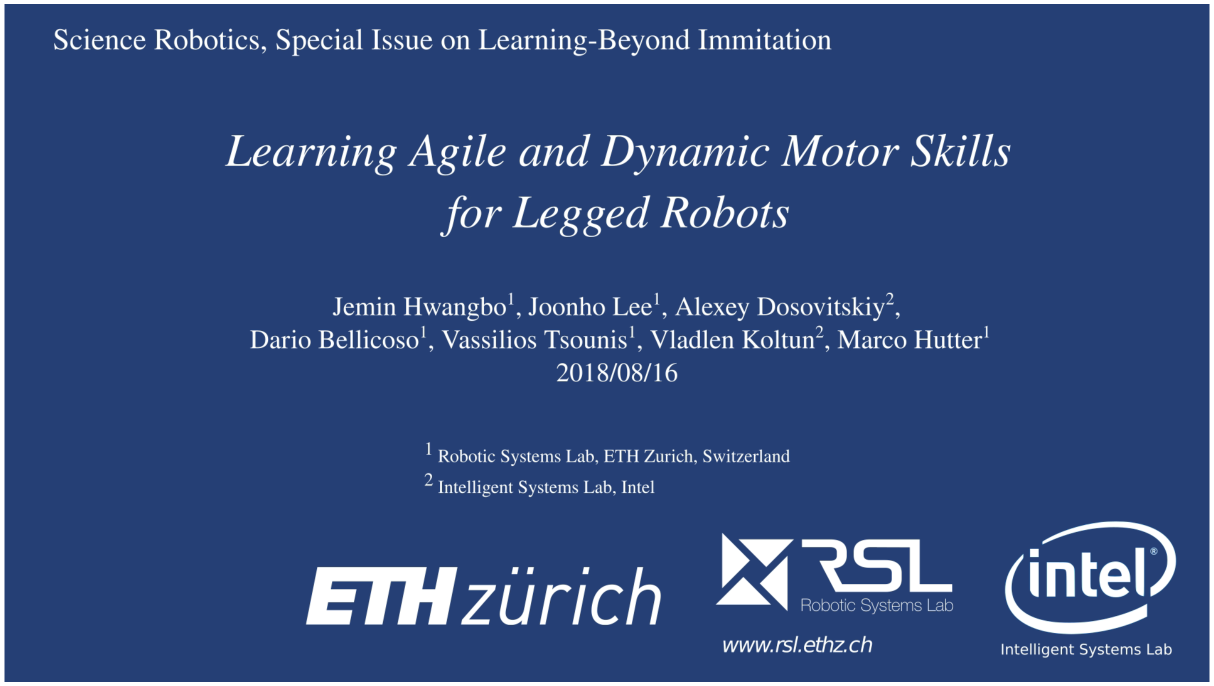
[Georgia Tech: CoRL 2017]



Rubik’s cube manipulation (OpenAI)



Anymal: Quadrupedal locomotion + getup (ETH Zurich)



Science Robotics, Special Issue on Learning-Beyond Immitation

Learning Agile and Dynamic Motor Skills for Legged Robots

Jemin Hwangbo[1], Joonho Lee[1], Alexey Dosovitskiy[2], Dario Bellicoso[1], Vassilios Tsounis[1], Vladlen Koltun[2], Marco Hutter[1]
2018/08/16

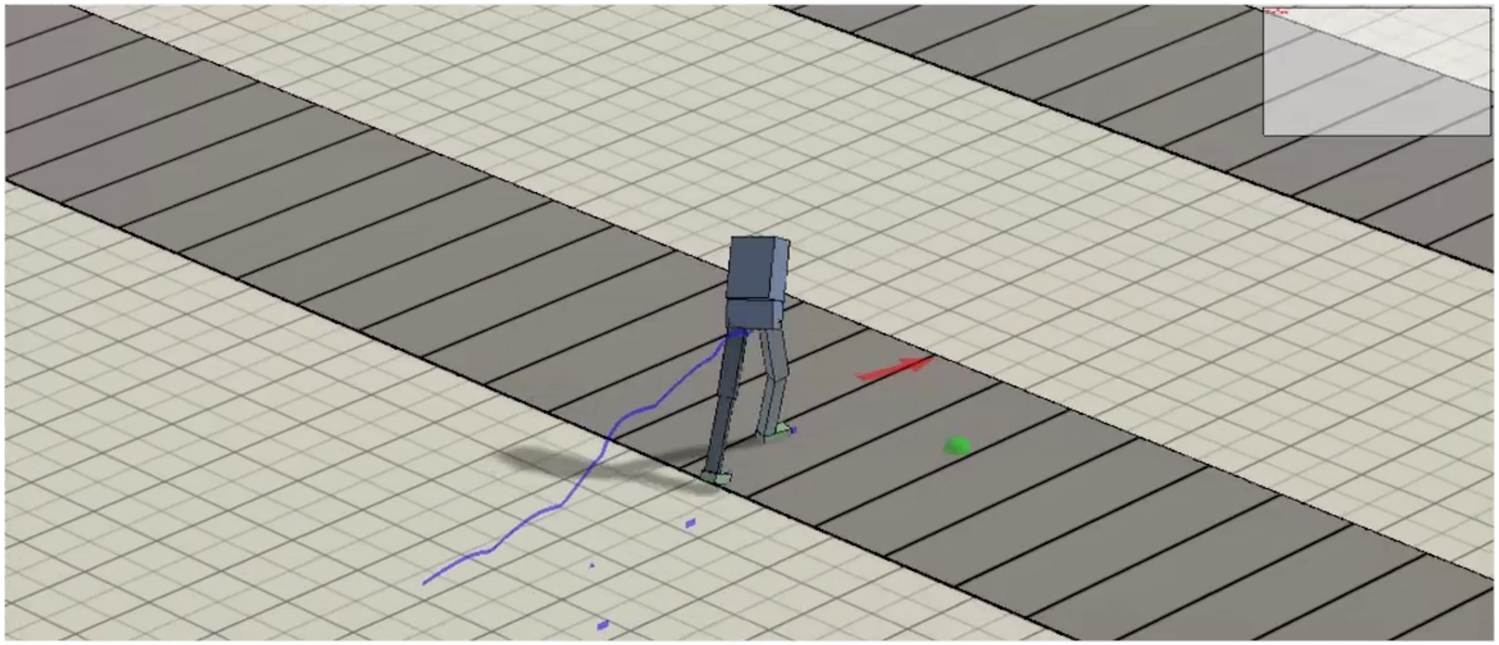
[1] Robotic Systems Lab, ETH Zurich, Switzerland
[2] Intelligent Systems Lab, Intel

ETH zürich RSL Robotic Systems Lab www.rsl.ethz.ch intel Intelligent Systems Lab

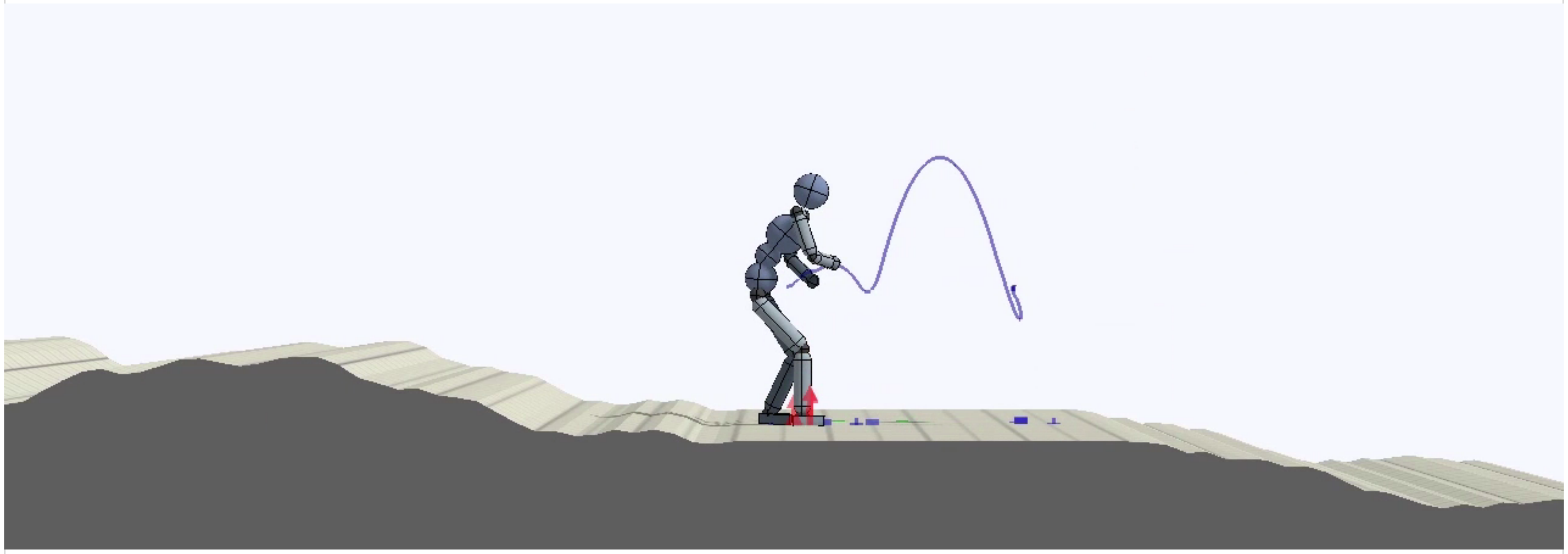
[Science Robotics, 2018]

[DeepLoco: SIGGRAPH 2017]

Walking on Conveyor Belts

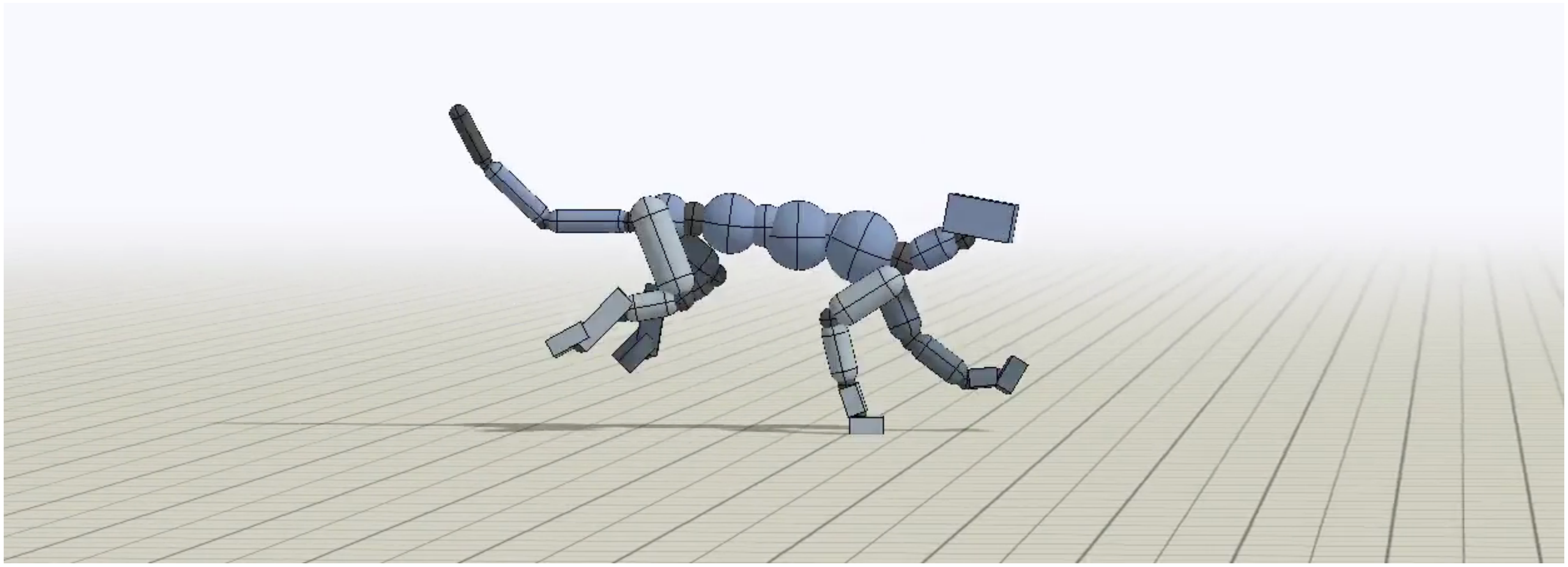


Retargeting



and retarget to different environments.

simulated lion



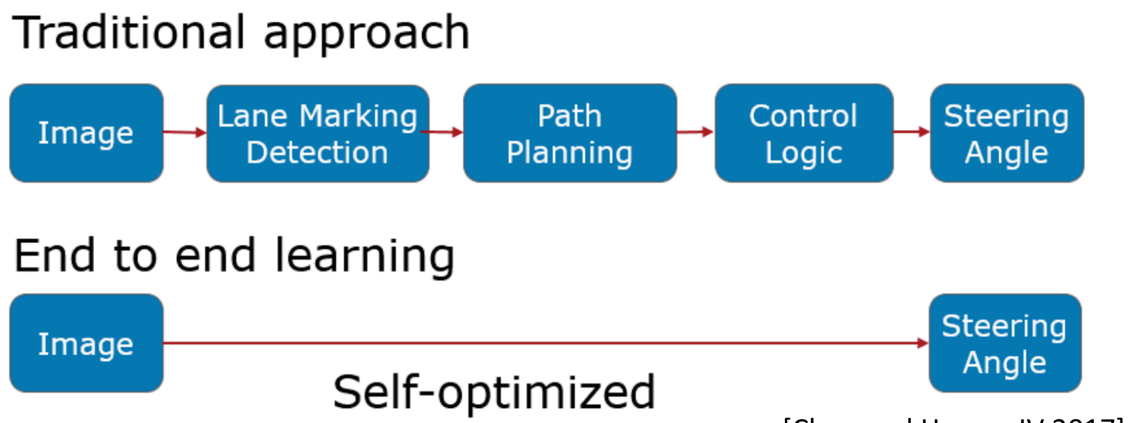
Physics-based Lion   (UC Berkeley, UBC, Ziva Dynamics)

[“DeepMimic”, SIGGRAPH 2018]

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

Why Deep RL? (continued)

End-to-end learning: performance benefits



[Chen and Huang, IV 2017]

“pixels to torques”

Good control from rich sensory streams is often the limiting factor

[https://robotik.dfki-bremen.de/en/research/robot-systems/exoskelett-aktiv-ca.html]

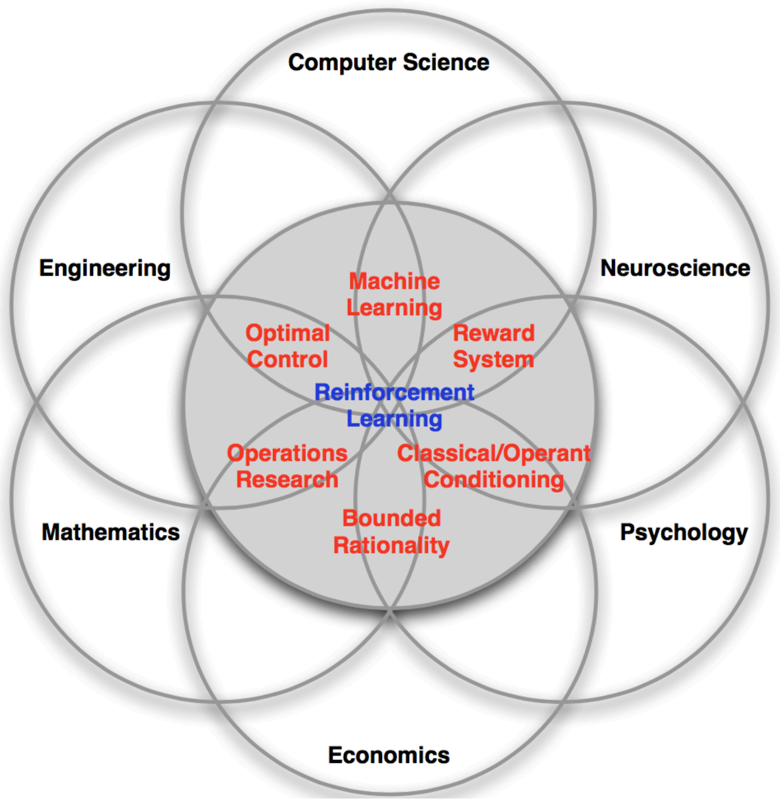
How do humans and animals learn?

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)

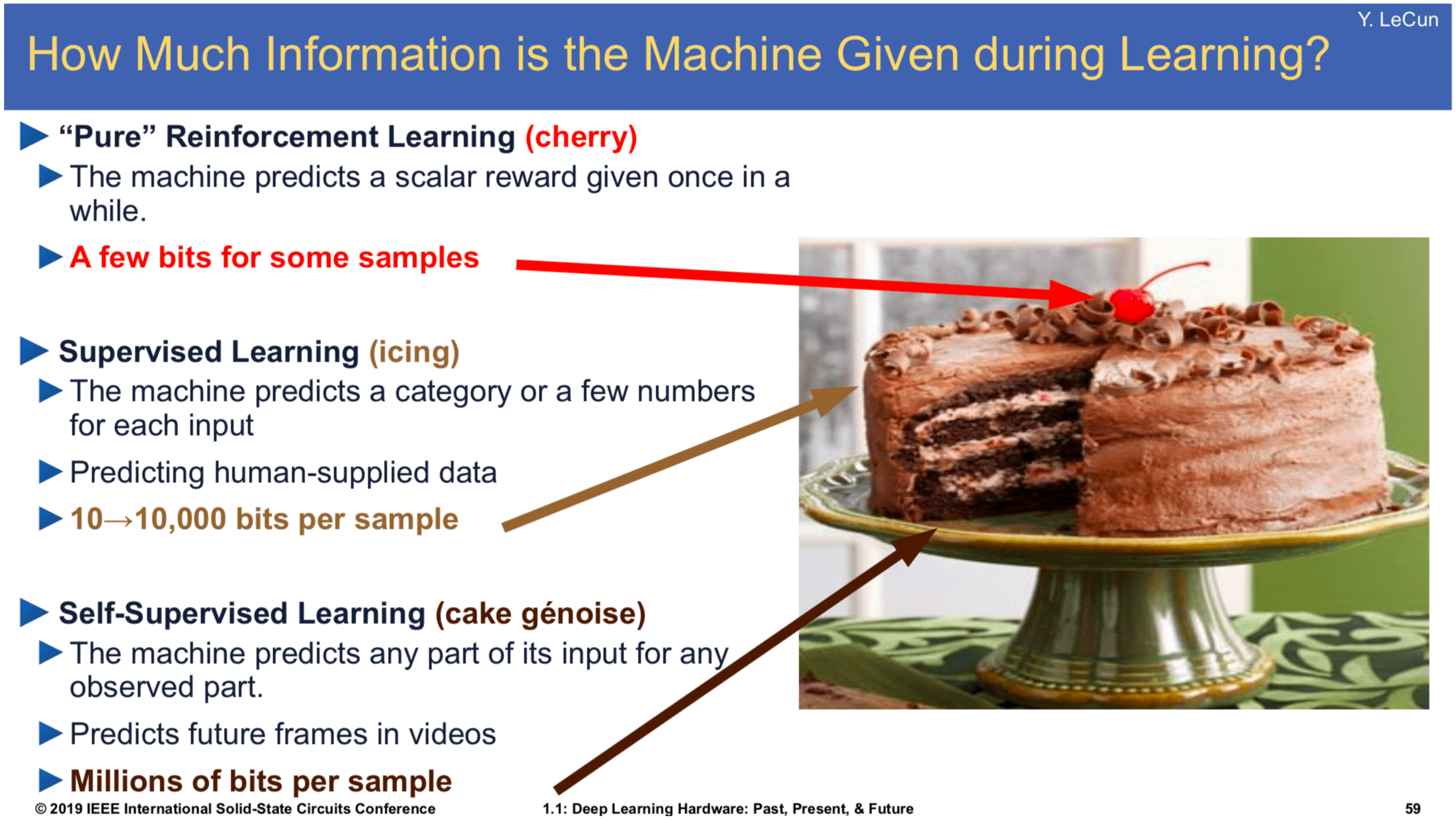
RL learning often has poor “sample efficiency”

- 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**

© 2019 IEEE International Solid-State Circuits Conference 1.1: Deep Learning Hardware: Past, Present, & Future 59

RL can be difficult to apply in practice

Reinforcement Learning for Real Life

ICML 2019 Workshop

June 14, 2019, Long Beach, CA, USA

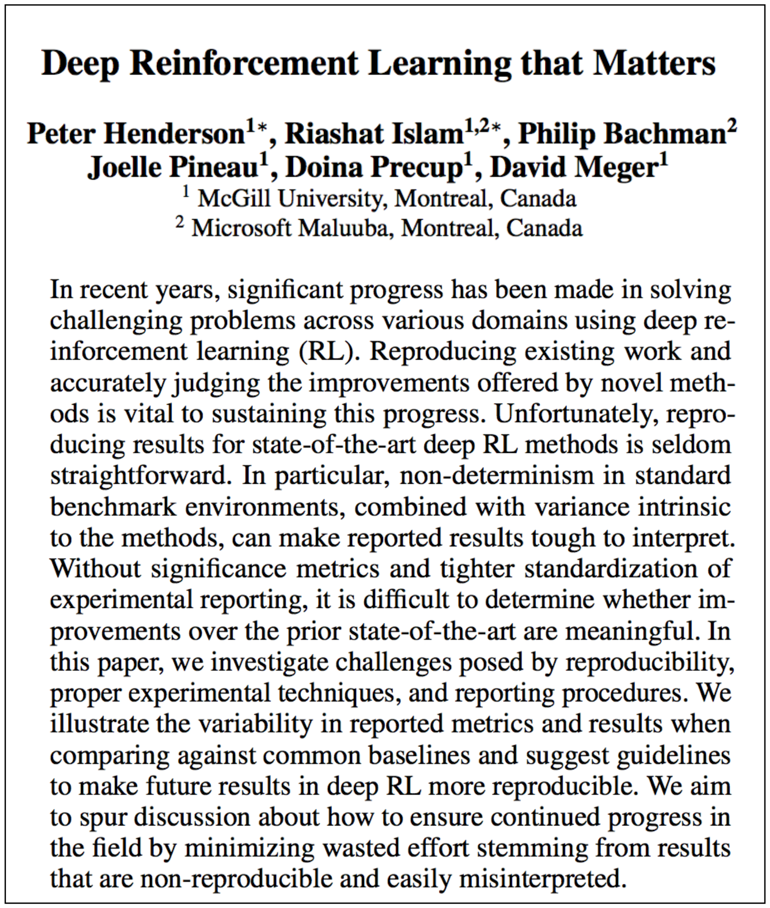
- Production systems
- Autonomous driving
- Business management
- Chemistry
- Computer Systems
- Energy
- Healthcare
- Robotics/manufacture

"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

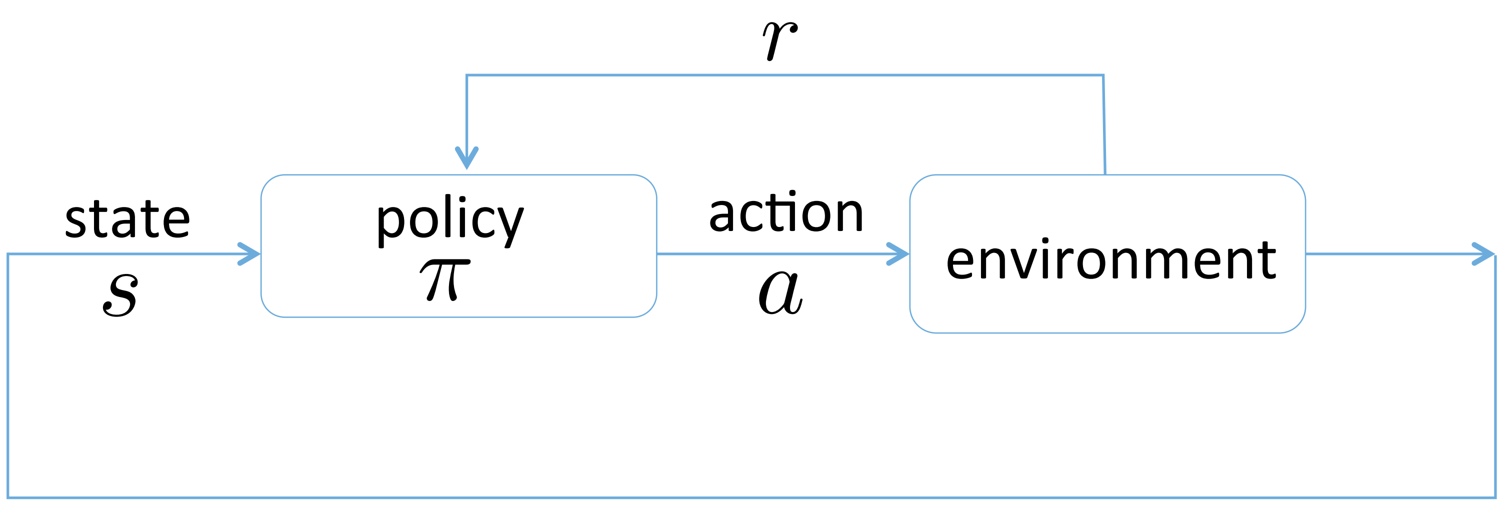
Peter Henderson[1]*, Riashat Islam[1,2]*, Philip Bachman[2]
Joelle Pineau[1], Doina Precup[1], David Meger[1]
[1] McGill University, Montreal, Canada
[2] Microsoft Maluuba, Montreal, Canada

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:

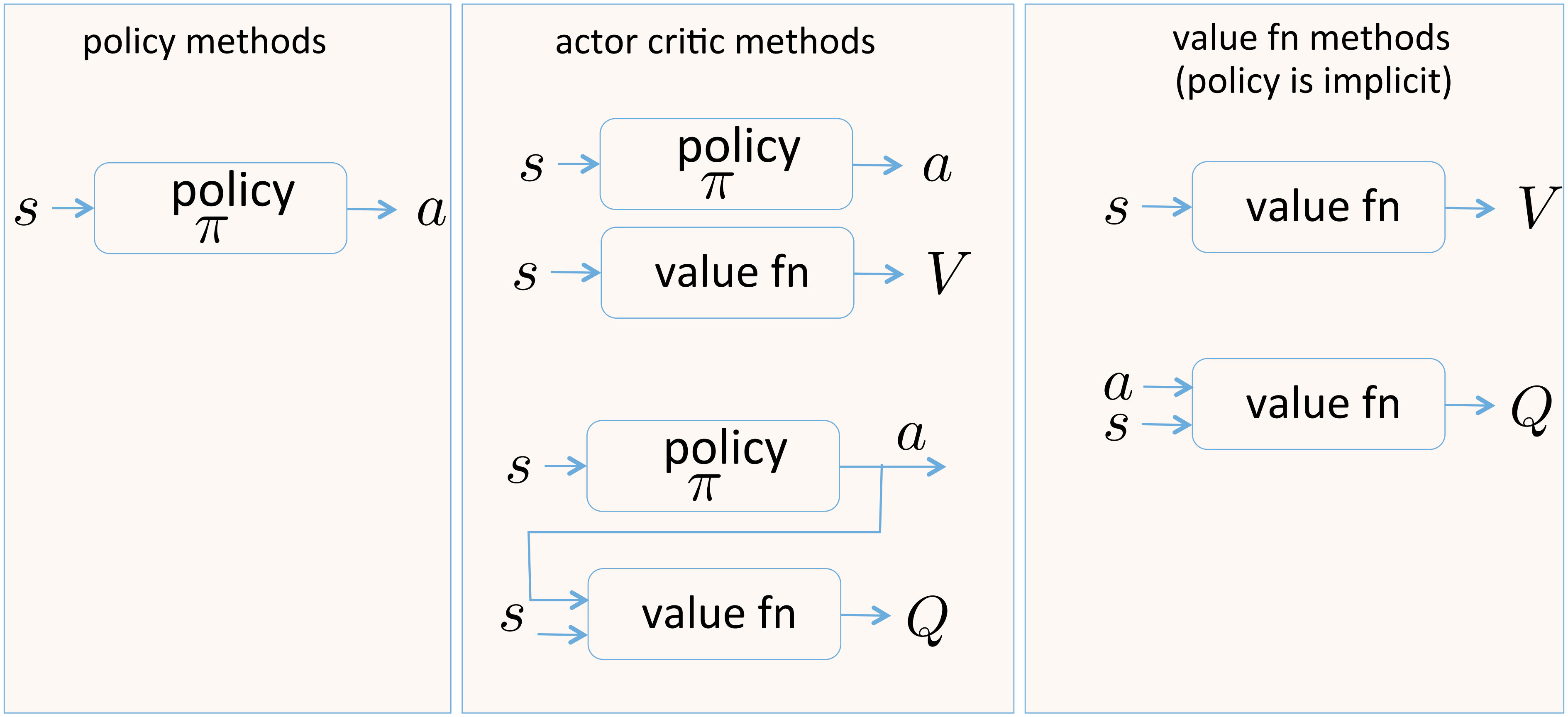
$$\{(s_i, a_i)\}$$

$$\Downarrow$$

$$a = \pi(s)$$

- requires an expert
- learned policy is unaware of the actual task (!)
- suffers from compounding errors over time
    - can be mitigated by collecting data during perturbations
- transfer between two Deep-NN policies:
  "policy distillation" "policy cloning"

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

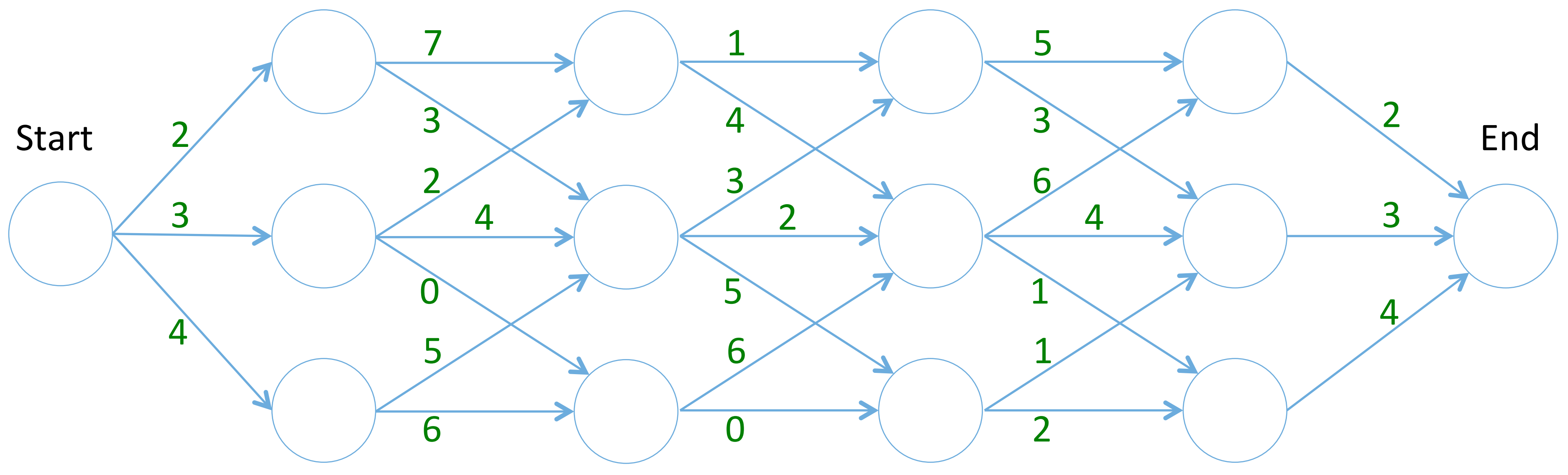
Common Assumptions

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

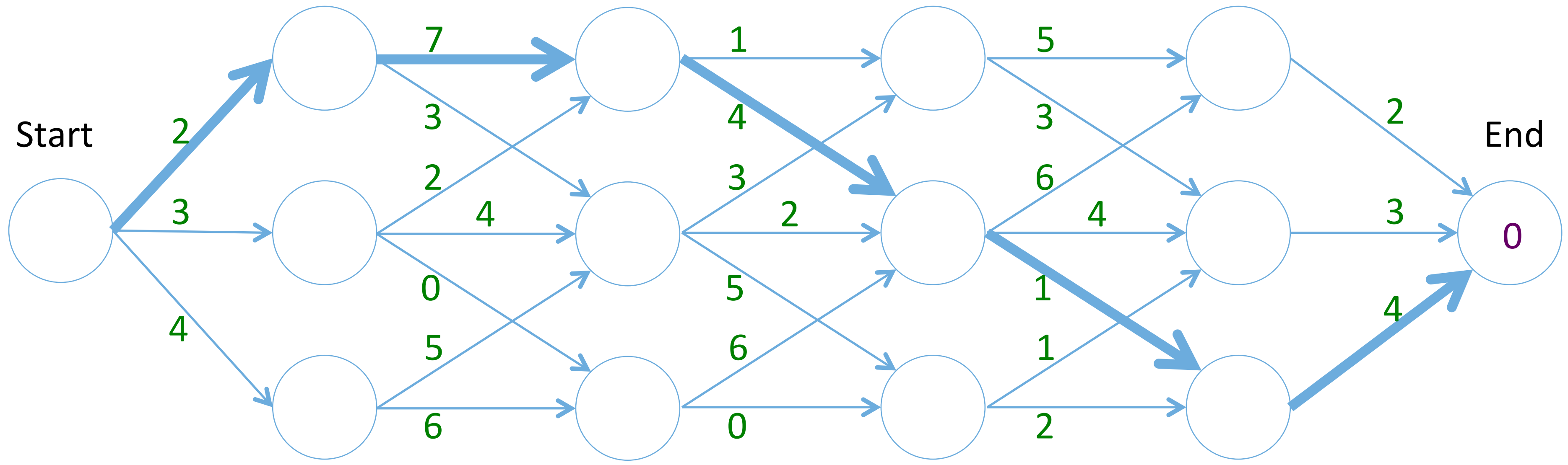
$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \ldots, S_t]$$

- Markov Decision Process (MDP) vs
- Partially Observable Markov Decision Process (POMDP)
  - state needs to be estimated; 'belief state'

Laying the Foundations: A Simple Deterministic Example



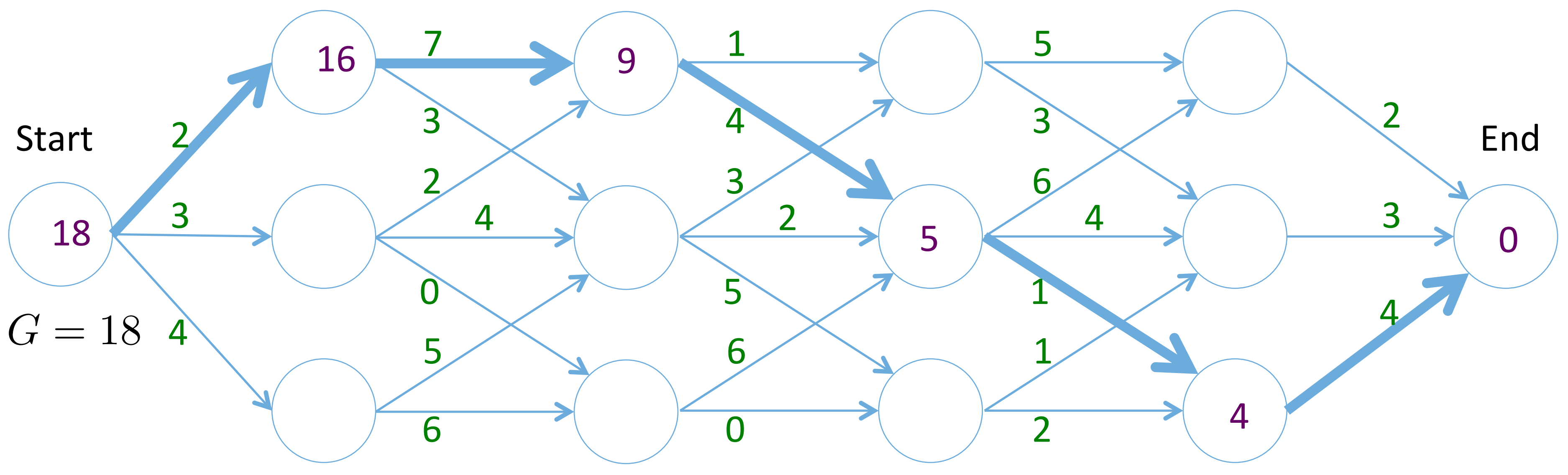
Returns for a given sequence of decisions



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

Above we have $\gamma = 1$

Returns for a given sequence of decisions



$G = 18$

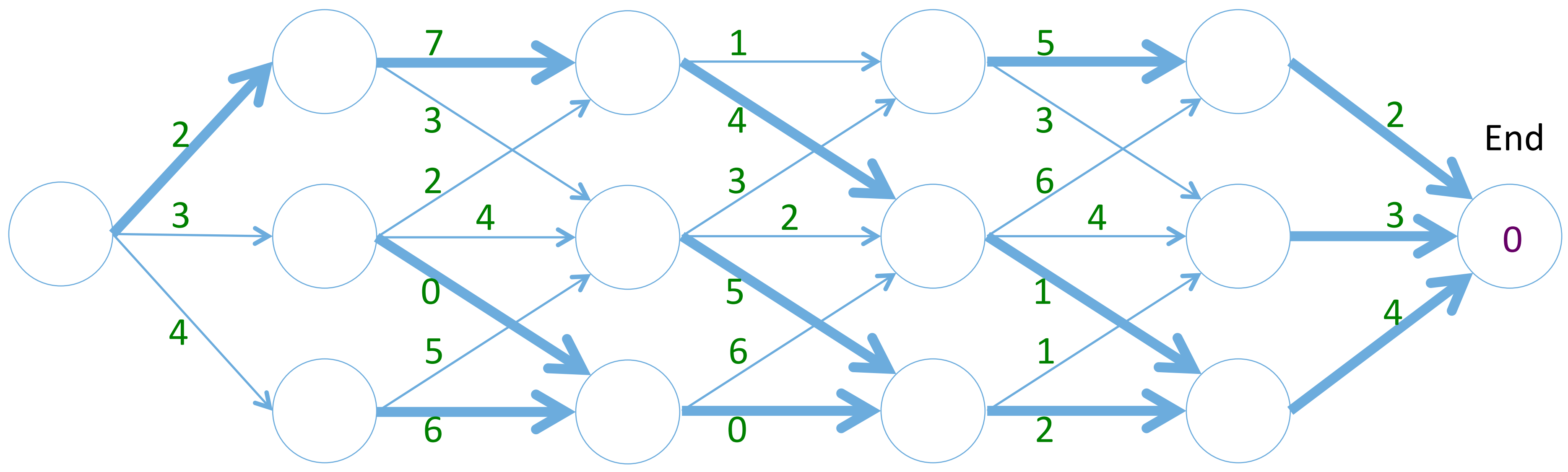
$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

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

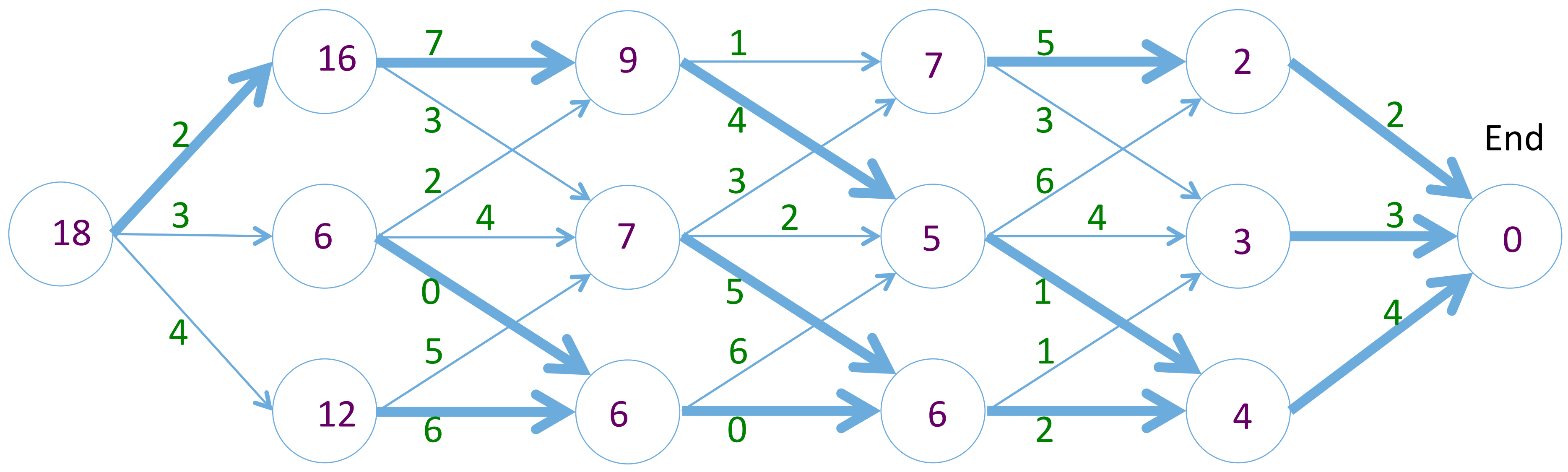
Assume that we begin in a random state, for this example, *"exploring start"*
and therefore we care about the returns from all states.



$$\gamma = 1$$

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

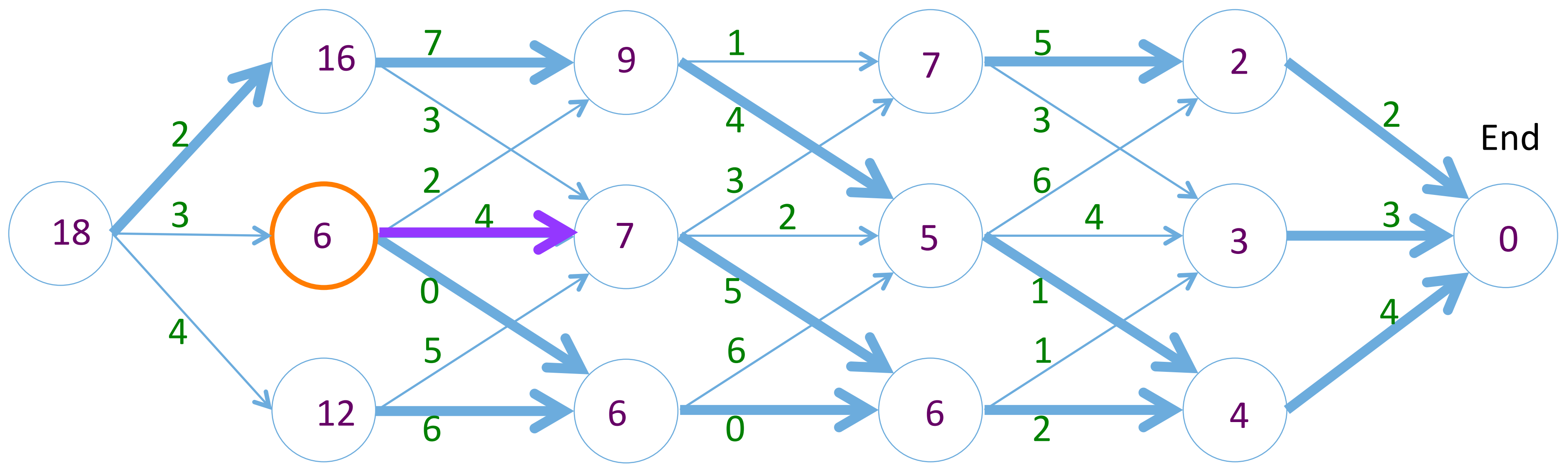
Returns for a given policy π



$$\gamma = 1$$

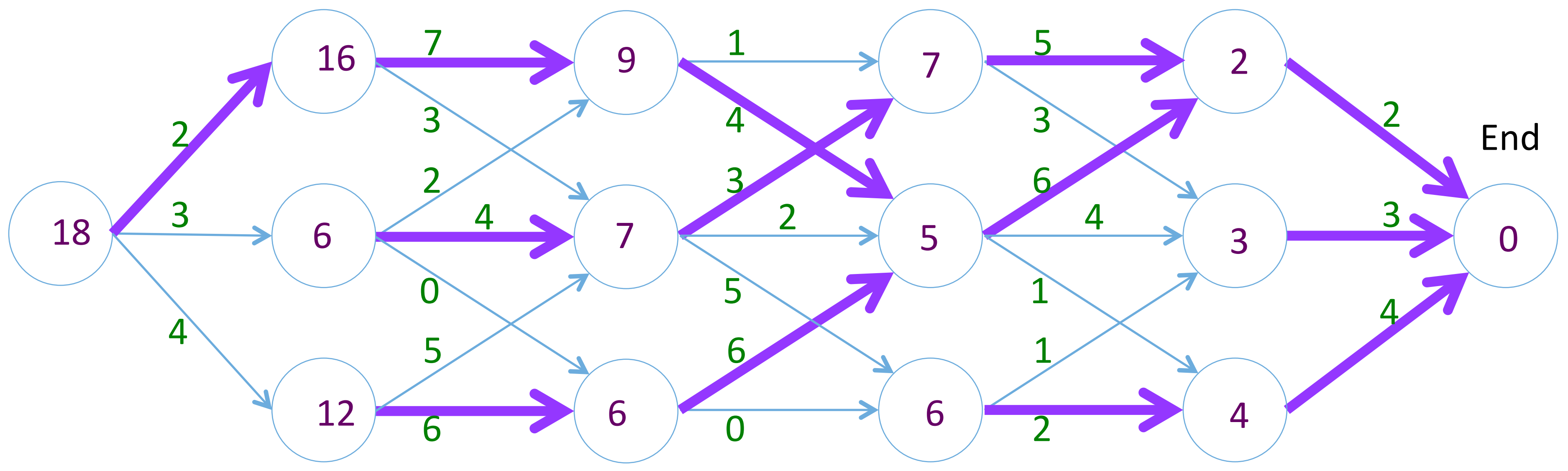
$$G_t = r_{t+1} + \gamma r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Given the known returns, now improve the policy



$$\pi(s) \leftarrow \arg\max \sum_{s',r} [r(s,a) + \gamma V(s')] \qquad \gamma = 1$$

Final improved policy

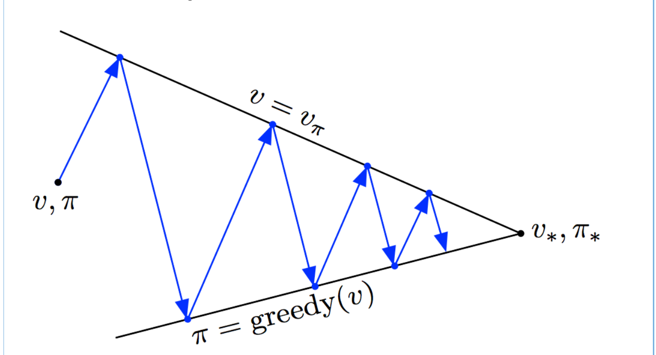


Terminology

- *Generalized Policy Iteration*

$$\begin{tikzcd} \pi \arrow[r, shift left, "\text{E: policy evaluation}"] & V_\pi \arrow[l, shift left, "\text{I: policy improvement}"] \end{tikzcd}$$

conceptual view



optimal policy,
optimal value fn

$$V_*, \pi_*$$

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

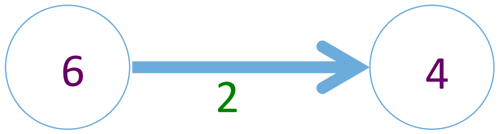
Terminology

- *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*:

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

similarly for the value functions

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

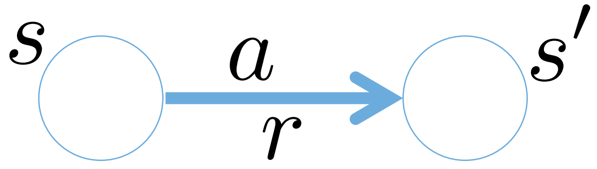


  - iterative value estimates depend on previous value estimates (!)
- *prioritized sweeping*: order updates based on size of expected change in V

Terminology

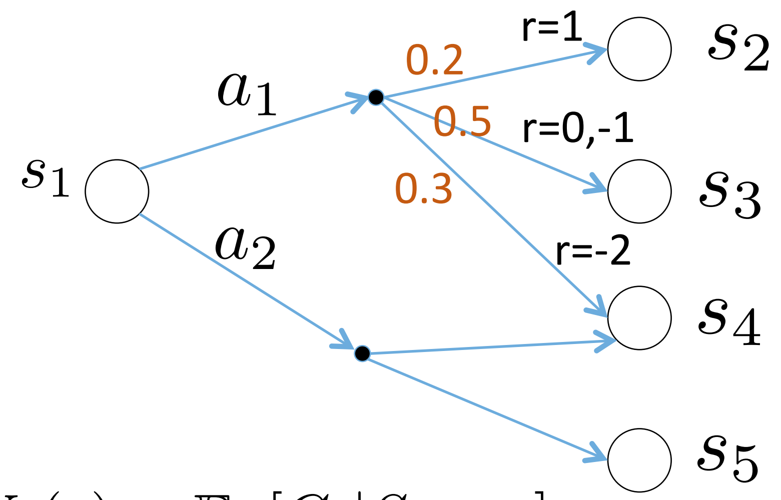
- *discount factor* $0 \leq \gamma \leq 1$
  - allows for continuing episodes, i.e., infinite length, which would otherwise produce a potentially infinite return
  - probability of episode continuing at any given time
  - preference for current rewards over future rewards, given that there may be more 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



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

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

The value function provides the **expected returns**

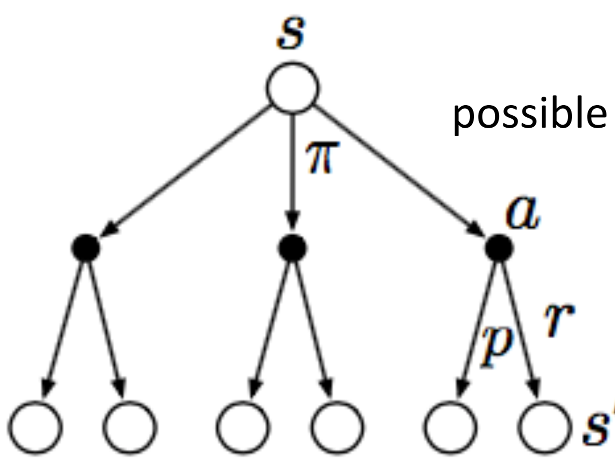
dynamics function: $p(s', r | s_1, a_1)$
"state transition model"

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

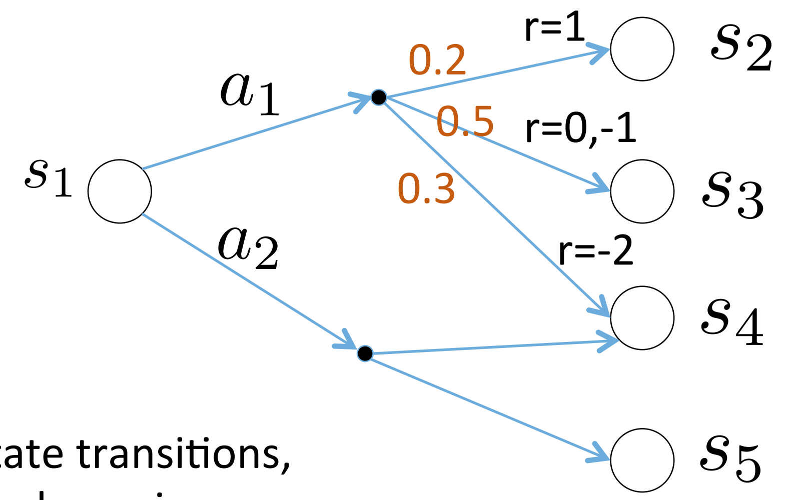
sums to 1

Common “backup diagram”



possible actions

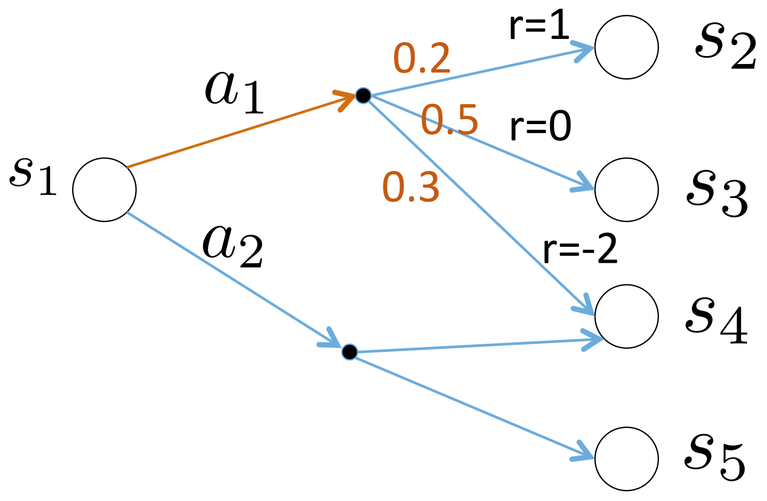
distribution of state transitions,
due to stochastic dynamics,
with unique rewards

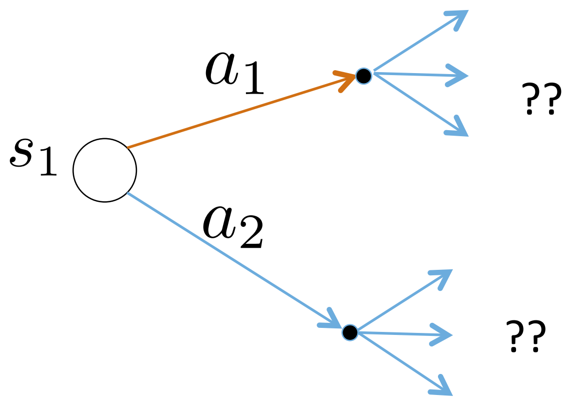


[Sutton, p81]

Model-based

Model-free





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

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

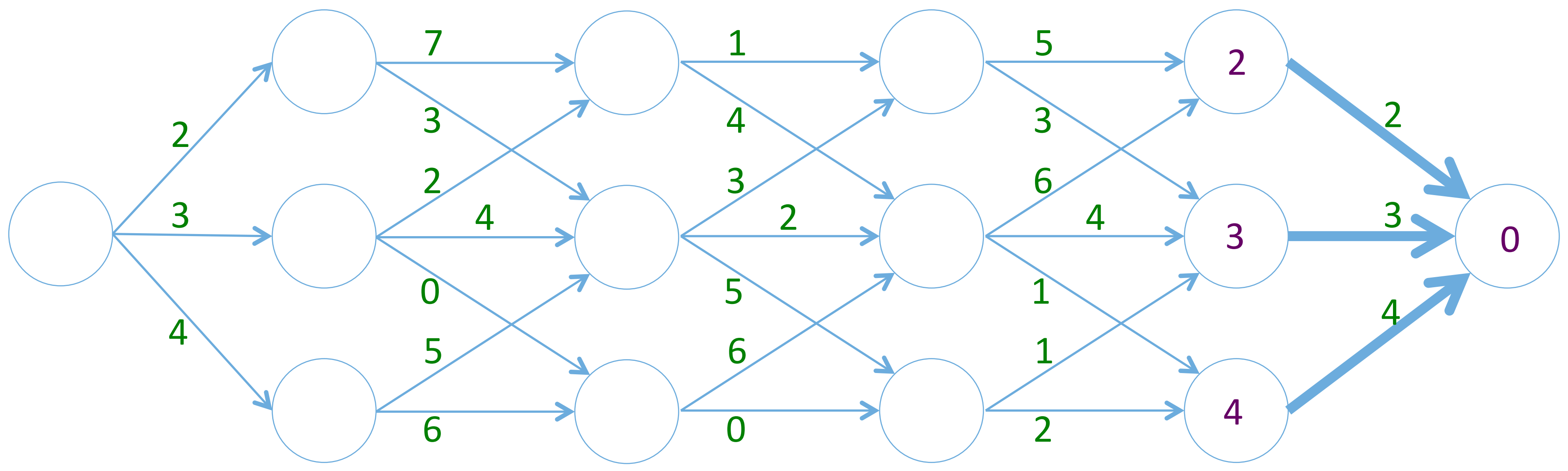
model-based: all state-transitions and rewards are known in advance, follow some known distribution

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

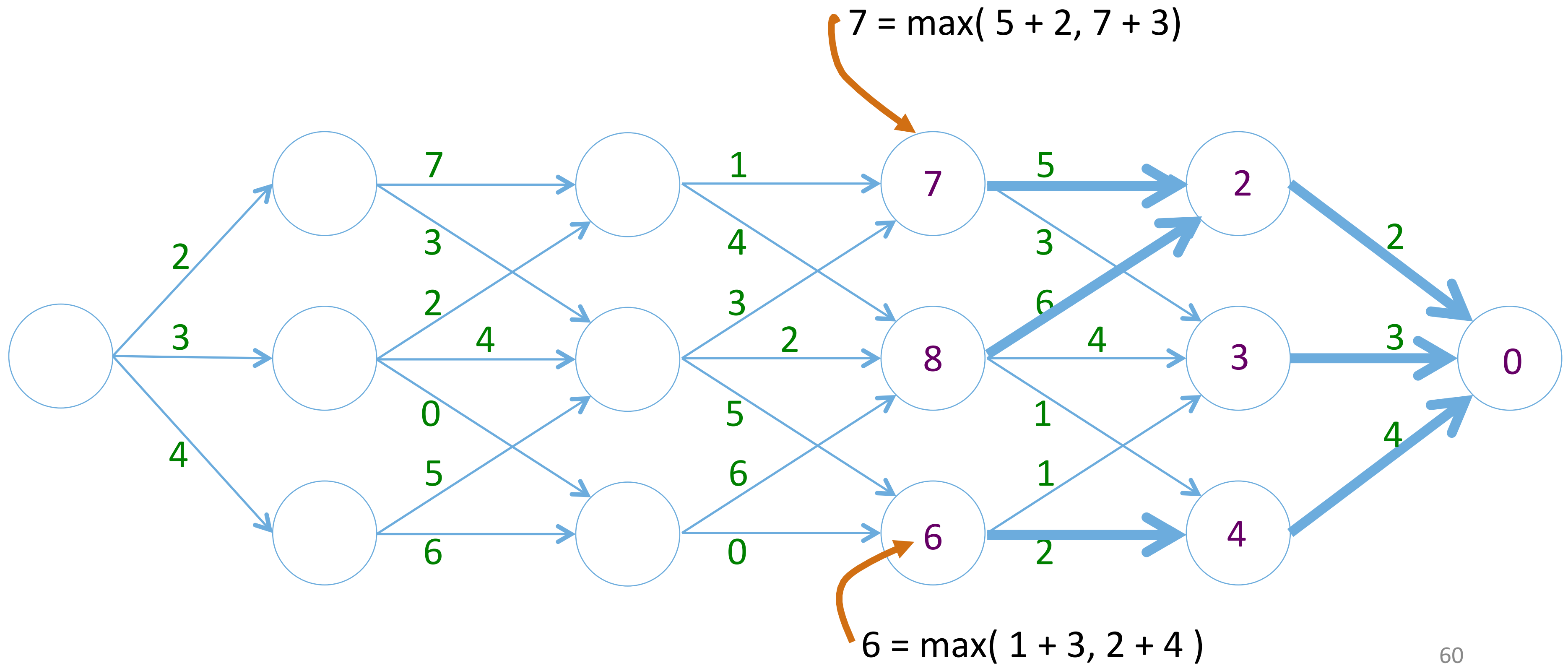
most basic RL: model-free); More advanced: model-based methods learn an approximate models for p()

Determining the best returns for every state

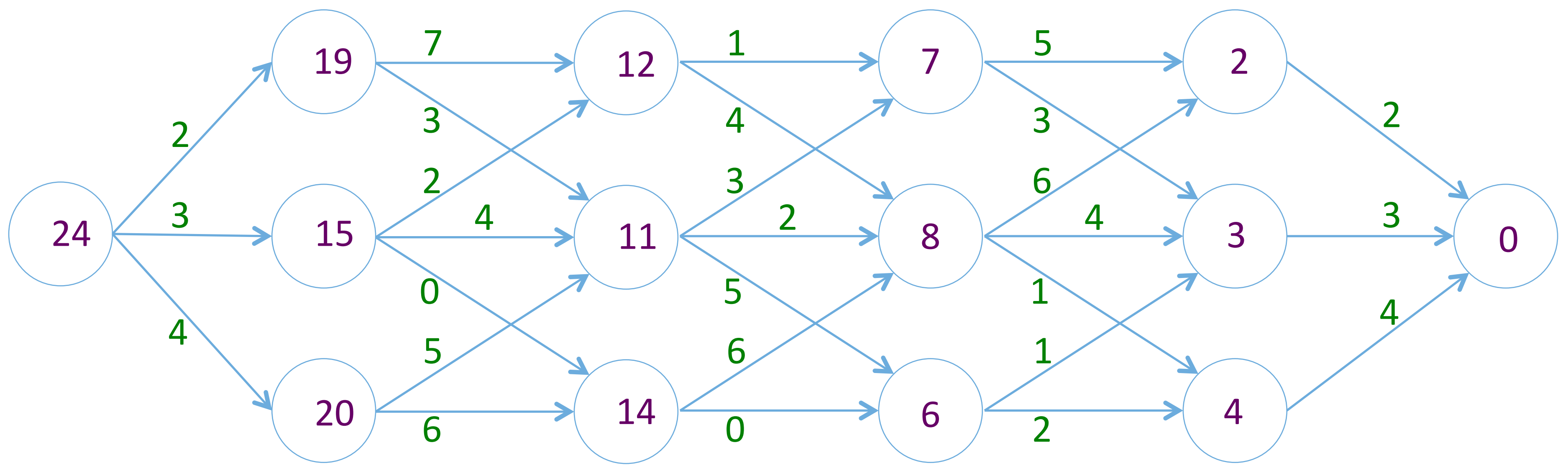
("Dynamic Programming" because the model is known)



“Bellman Backups”



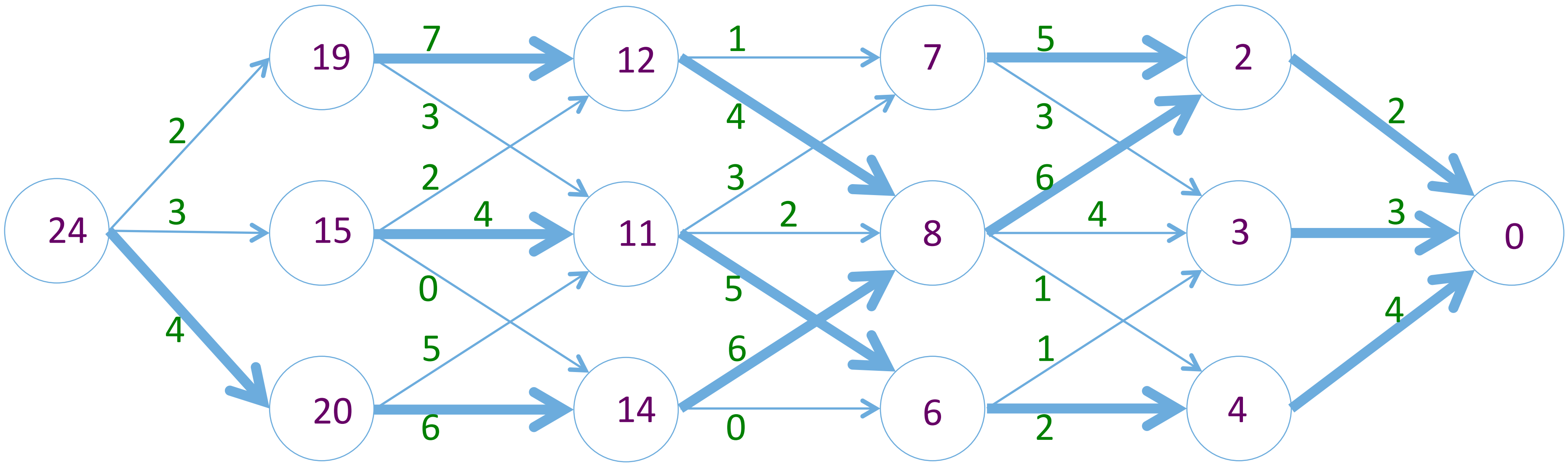
Final best returns for all states



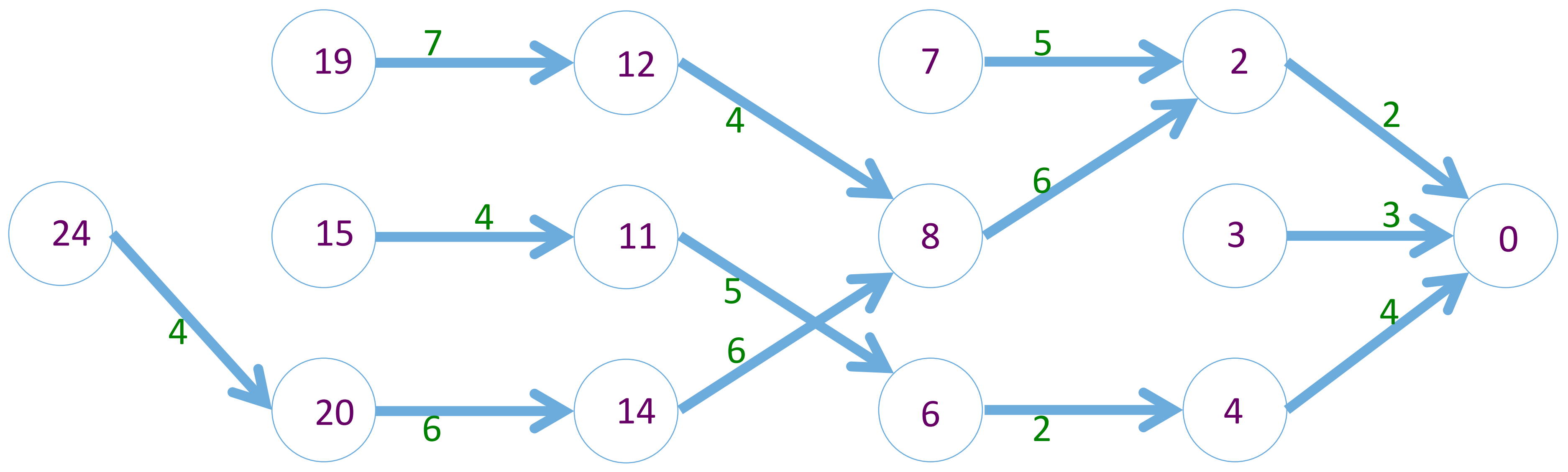
Optimal Policy (implicit in optimal value fn)

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: $V_{k+1}(s) = \max_a (\sum_{s',r} p(s', r | s, a) \left[r(s, a) + \gamma V_k(s') \right]$

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

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

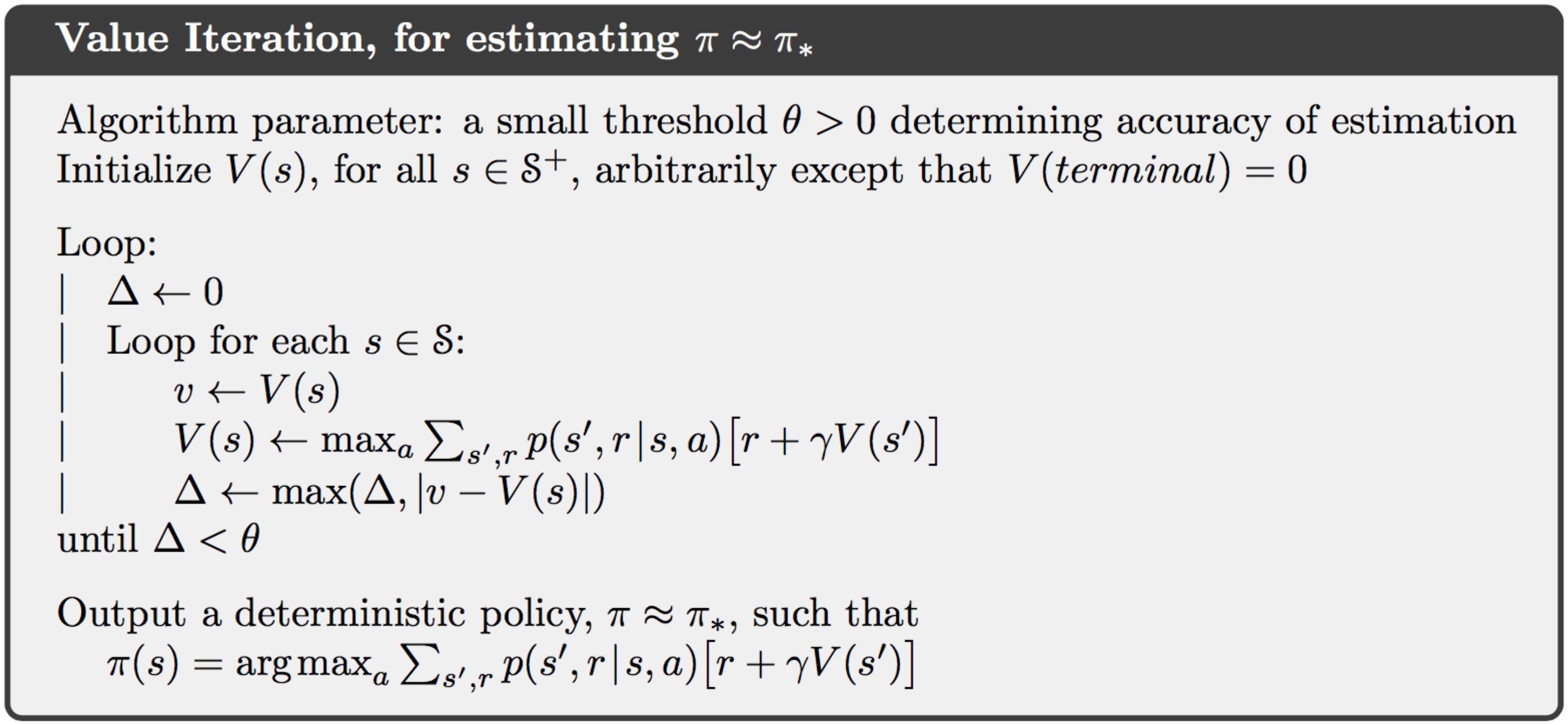
  - average reward per time-step (largely equivalent; we'll typically use this)

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

for a stochastic policy

~[Silver]

Value Iteration: iterative Bellman backups until convergence (note: requires a model!)



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

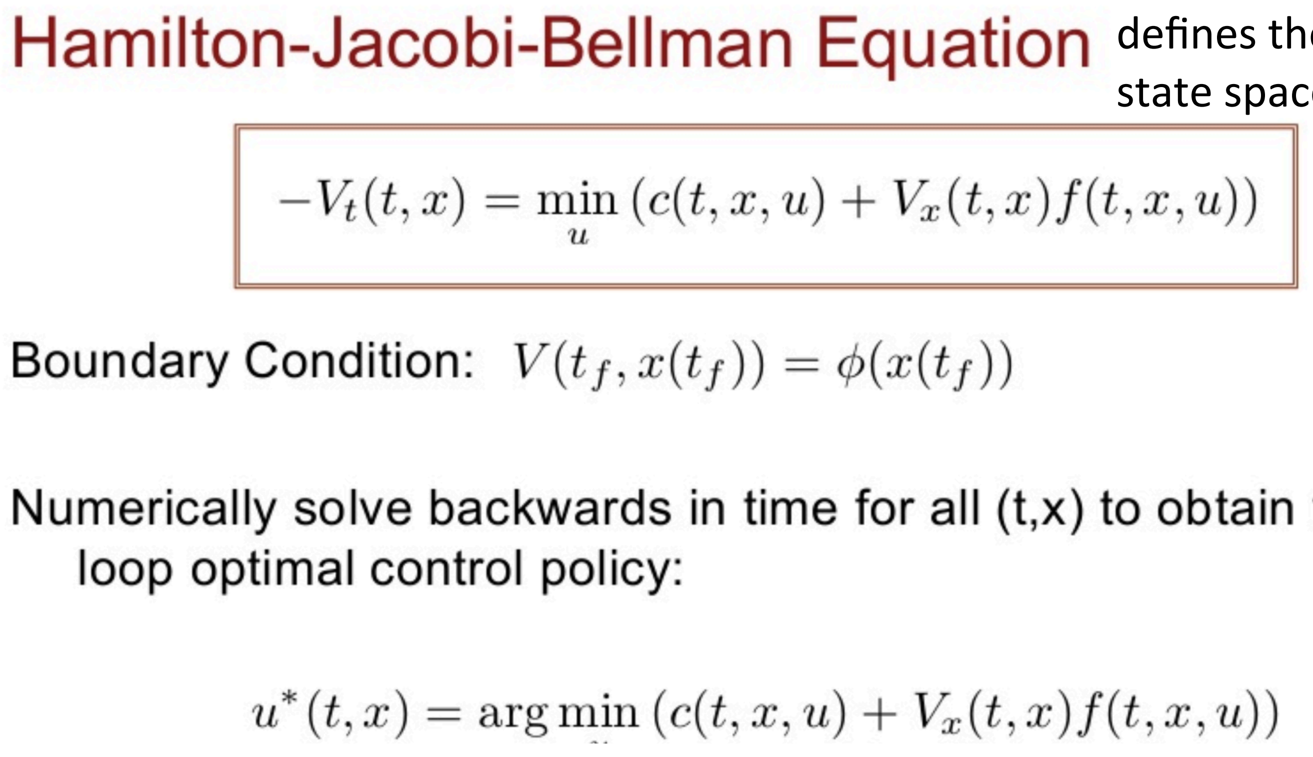
```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

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

[Sutton, p83]

Remarks

- in general, need repeated sweeps through all states (our case: one sweep)
- dynamic programming: all states, actions are visited; dynamics known
  - discrete states and actions: planning on a graph; Viterbi algorithm
  - continuous states and actions
    - linear system, quadratic cost function: Linear Quadratic Regulator (known optimal solution)
    - more arbitrary dynamics and costs: Hamilton-Jacobi solvers (for low-D systems)
- constructing the policy requires knowing the dynamics model, p():

But in general "model free" RL, we don't know the model, Solve an n-dimensional PDE that defines the value function in the state space.



Hamilton-Jacobi-Bellman Equation

$$\boxed{-V_t(t,x) = \min_u \left(c(t,x,u) + V_x(t,x)f(t,x,u)\right)}$$

Boundary Condition: $V(t_f, x(t_f)) = \phi(x(t_f))$

Numerically solve backwards in time for all (t,x) to obtain the closed-loop optimal control policy:

$$u^*(t,x) = \arg\min \left(c(t,x,u) + V_x(t,x)f(t,x,u)\right)$$

[https://www.slideshare.net/HarukiNishimura/stochastic-control-and-information-theoretic-dualities-complete-version]

Hamilton-Jacobi-Bellman equations

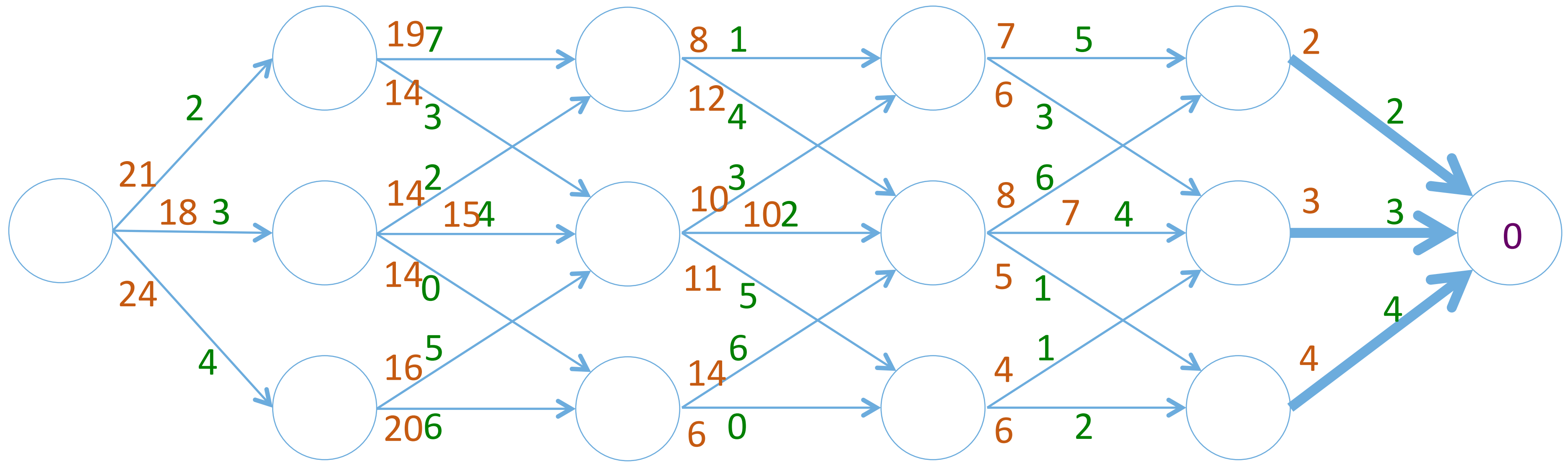
- continuous-state, continuous-action problems
- a non-linear PDE (partial differential equation) in the value function, whose solution is the value function itself
- Eikonal equation (optics!) is a particular case of HJB equations
- numerical solutions, often using level-set methods
  - solution evolves backwards in time
  - compute level sets in the value function
- feasible in low-D spaces, e.g., <= 7D
- see Ian Mitchell's work

RL algorithm attributes

- state space: discrete vs continuous
- action space: discrete vs continuous
- environment: stochastic vs deterministic
- task: episodic vs continuing
- policy: stochastic vs deterministic
- learning: model-free vs model-based
- learning: with-critic vs without-critic
- learning: on-policy vs off-policy
- learning: tabula rasa vs with demonstrations
- learning: simulation vs real-world

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

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

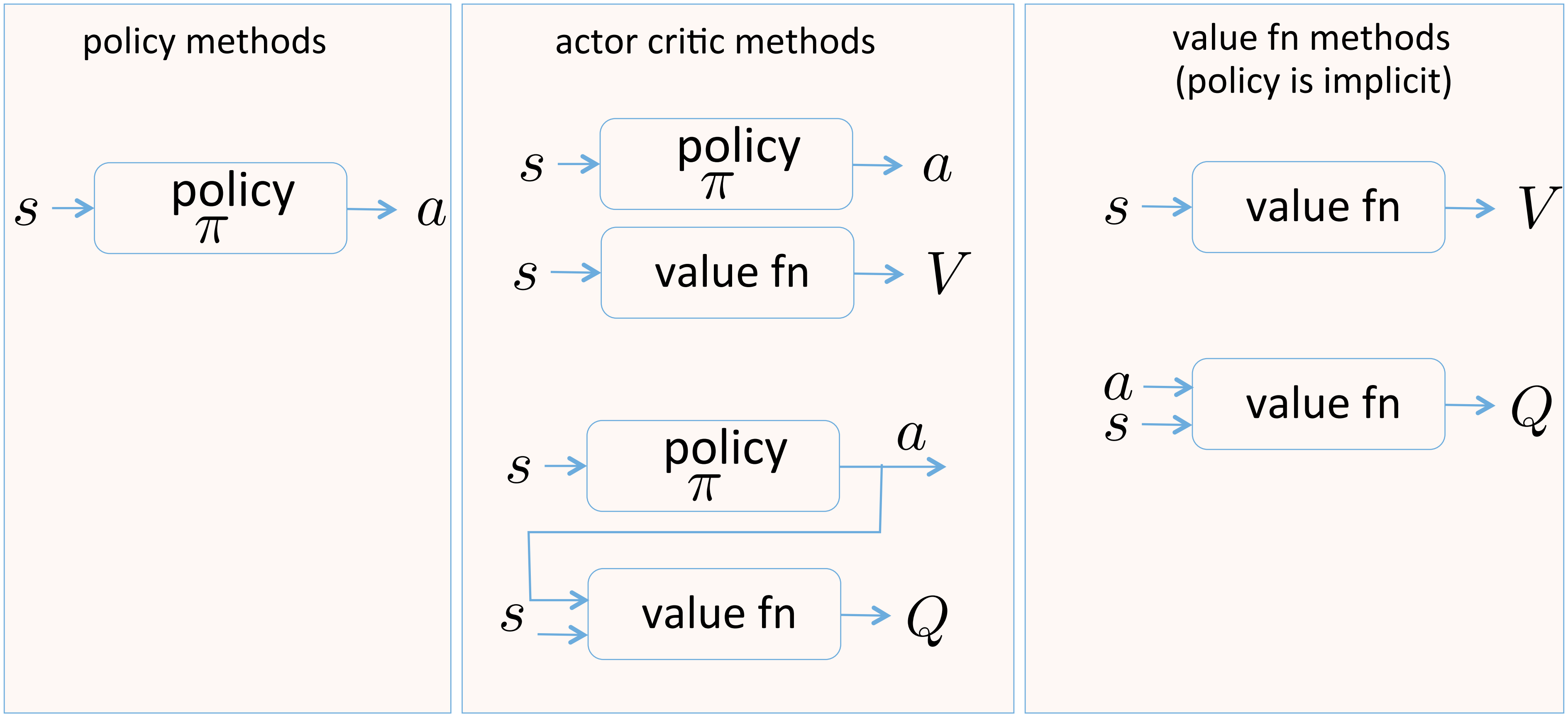
- think of having N value functions, one for the outcome of each action
- policy then becomes trivial:

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

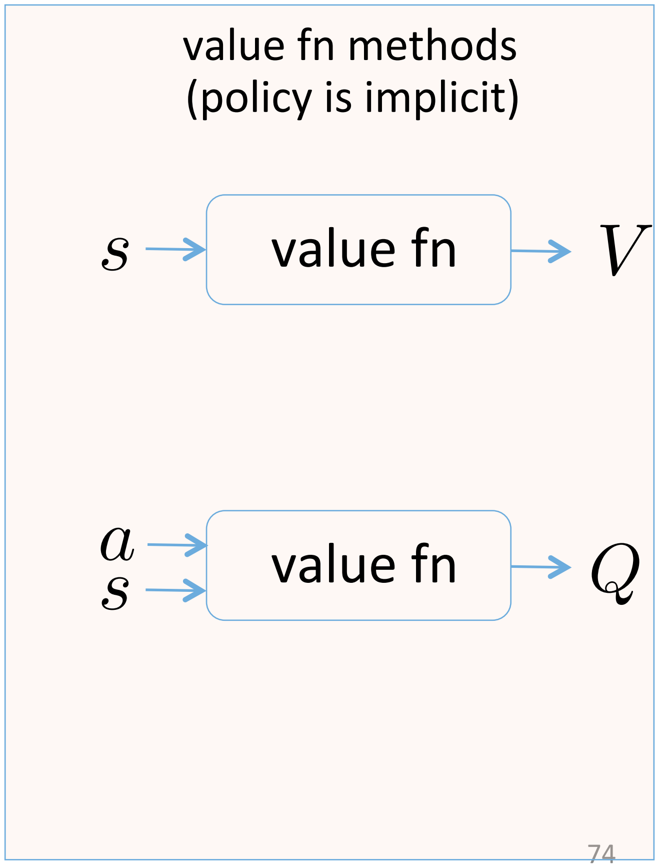
- Bellman backup
  - deterministic case:

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

Deep RL: key building blocks



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

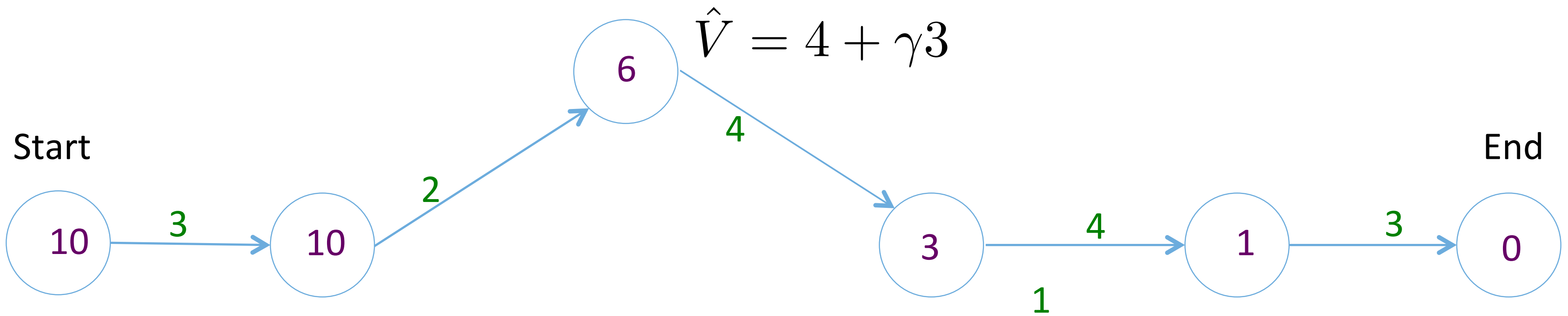
[Silver]

Temporal-difference estimation: TD(0)

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

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{\underbrace{r_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - V(S_t)}_{\delta})$$

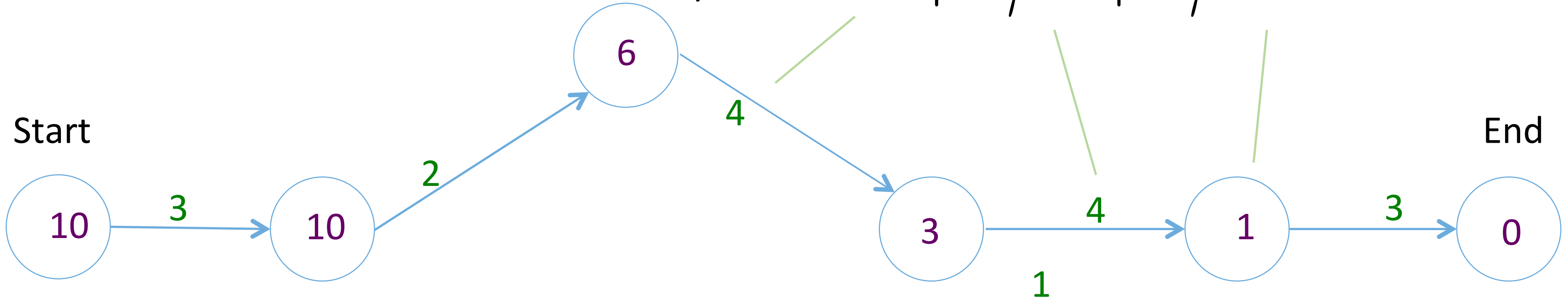
δ temporal difference



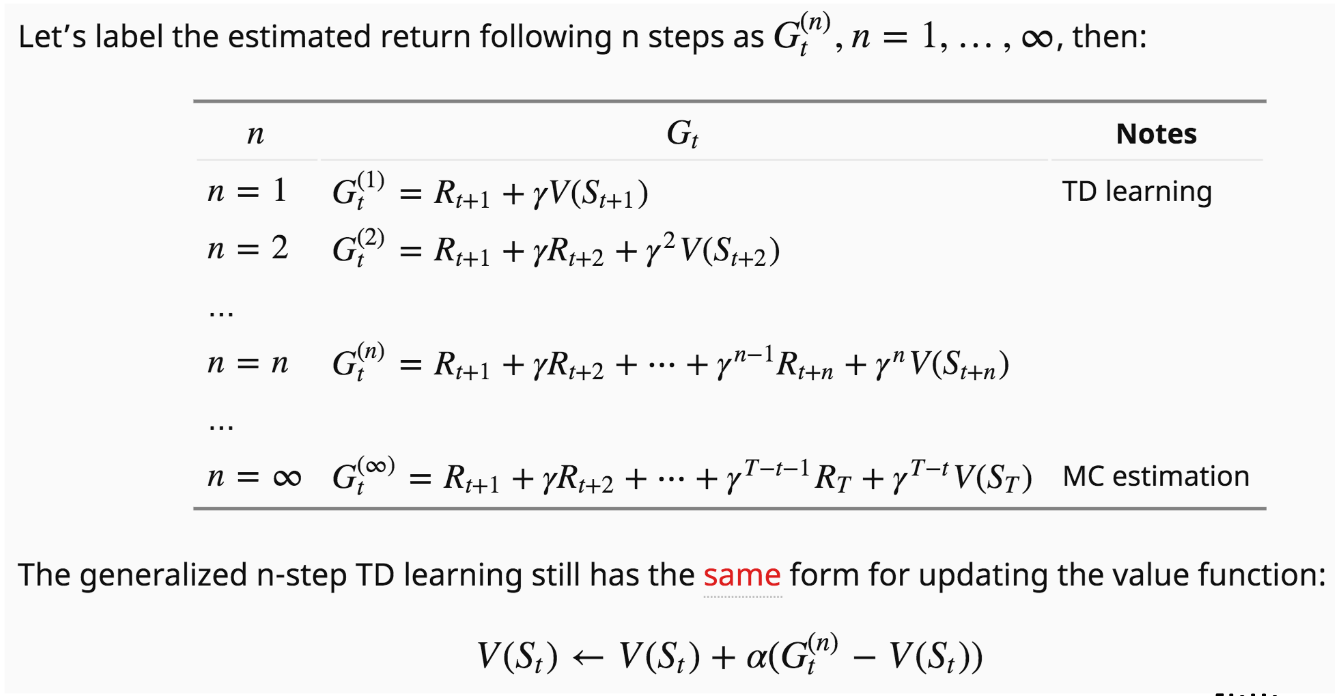
if model is known: $V(s) = \sum p(s', r|s,a)\,[r(s,a) + \gamma V(s')]$

n-step Temporal Differences

$$\hat{V} = 4 + \gamma 4 + \gamma^2 1$$



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
$n = 2$	$G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$	
...		
$n = n$	$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$	
...		
$n = \infty$	$G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T + \gamma^{T-t} V(S_T)$	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]

TD(λ)

Which value of n is best?
→ can use a weighted mix

relative weight

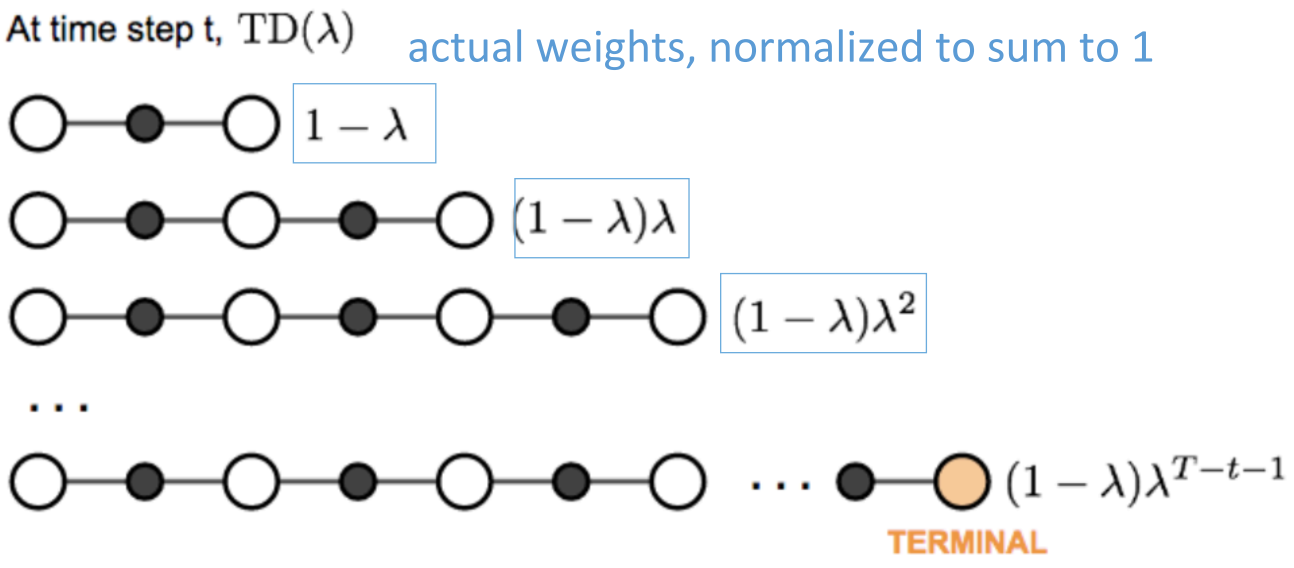
1

λ

λ^2

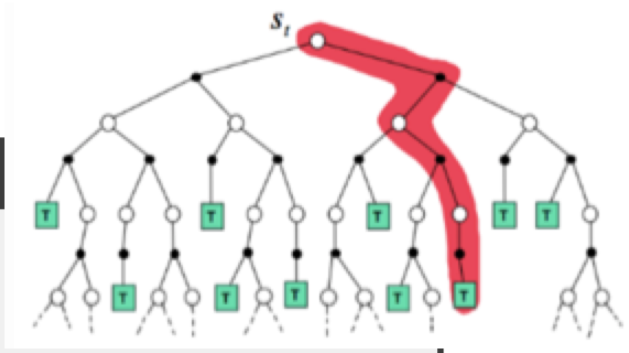
$\cdots$

$$\frac{1}{1-\lambda}$$ total



At time step t, $\mathrm{TD}(\lambda)$
actual weights, normalized to sum to 1
$1-\lambda$
$(1-\lambda)\lambda$
$(1-\lambda)\lambda^2$
$(1-\lambda)\lambda^{T-t-1}$
TERMINAL

Monte Carlo estimation of state-values



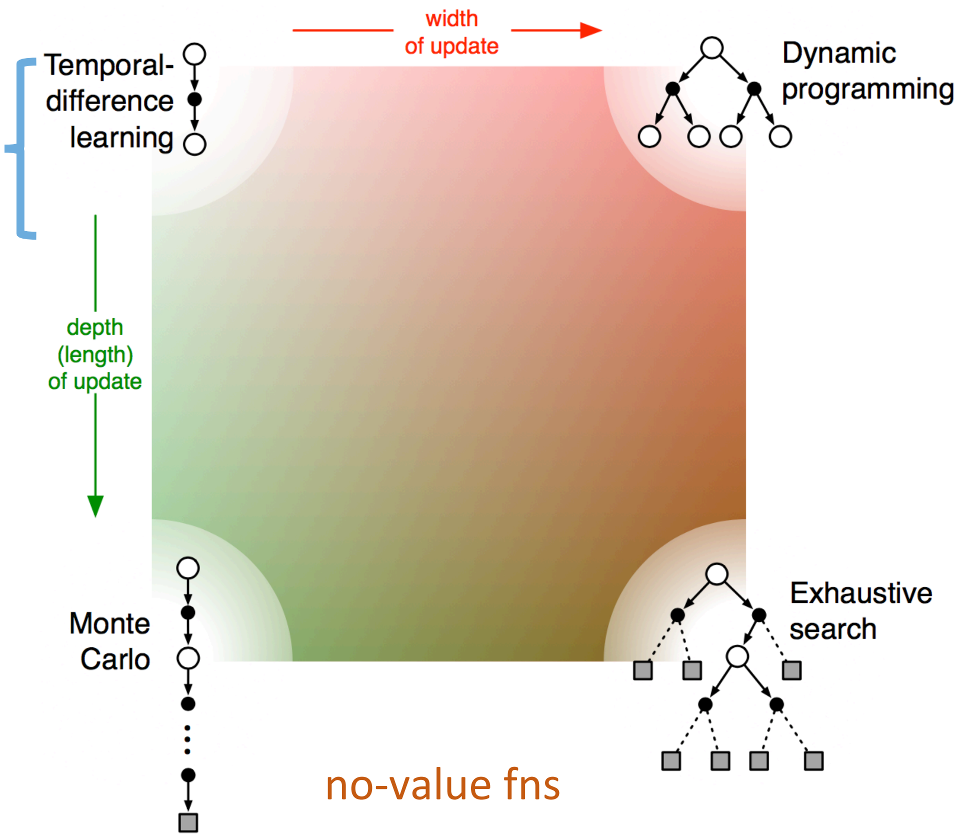
First-visit MC prediction, for estimating $V \approx v_\pi$

```
Input: a policy $\pi$ to be evaluated
Initialize:
    $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless $S_t$ appears in $S_0, S_1, \ldots, S_{t-1}$:
            Append $G$ to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$
```

bootstrapping, i.e., value-functions

most methods
we'll care about

model-free

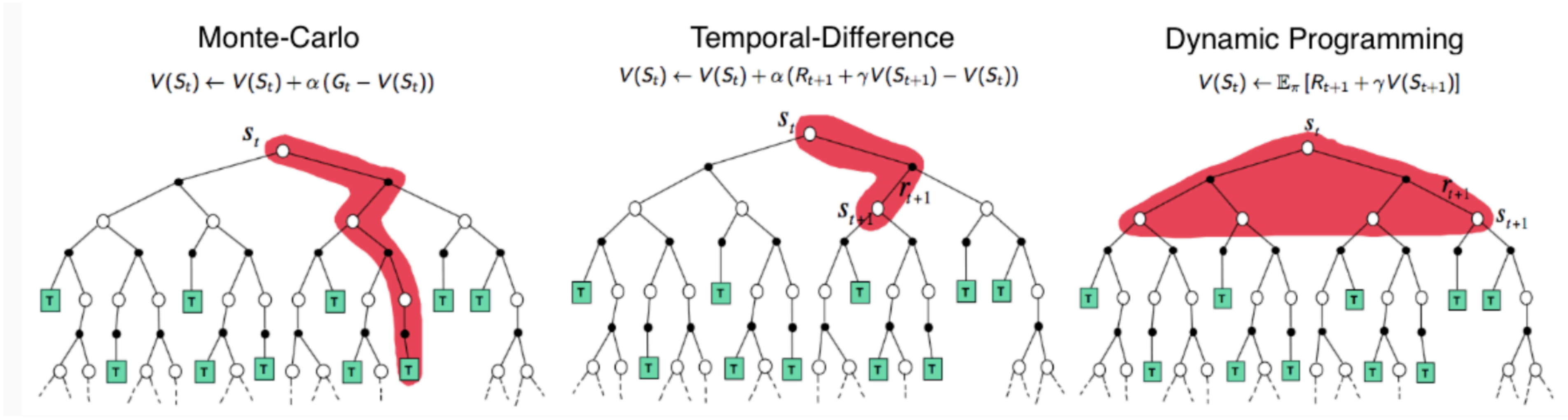


model-based
(tabular setting)

no-value fns

[Sutton, p190]

Another view of the corners of the design space



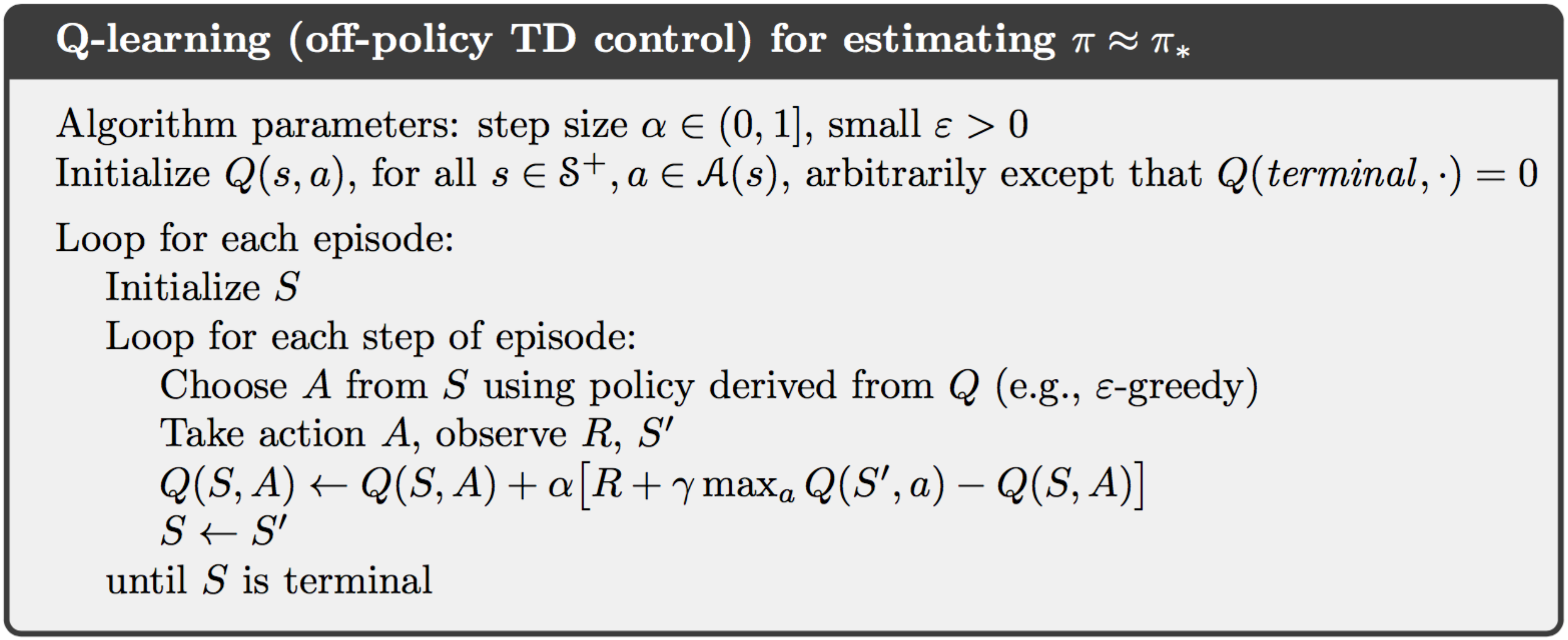
Monte-Carlo
$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
s_t
T
Temporal-Difference
$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
s_t
r_{t+1}
s_{t+1}
Dynamic Programming
$V(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1})]$
s_t
r_{t+1}
s_{t+1}

[Silver/Wang]

Temporal-difference for estimating Q(s,a)

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

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., $\varepsilon$-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]

More on Q-learning

- provably converges in the tabular setting (discrete states & actions) as long as all state-action pairs continue to be updated
- the default policy will be non-optimal because of the exploration. We need to reduce ϵ gradually to zero in order to converge to optimal policy.

Maximization Bias

$$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$$

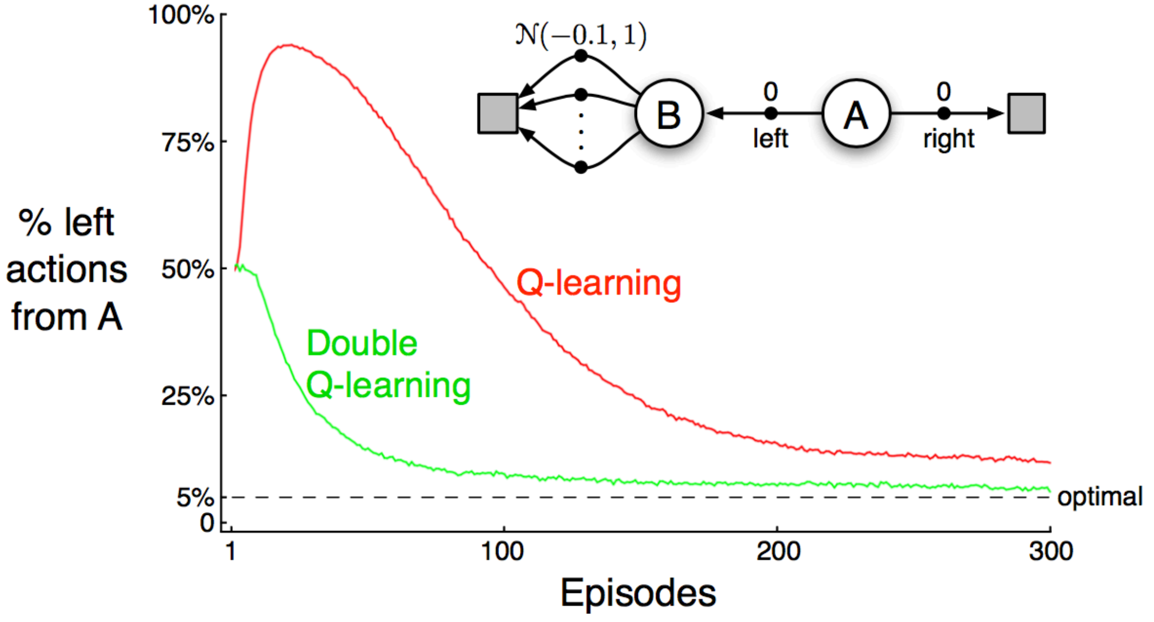
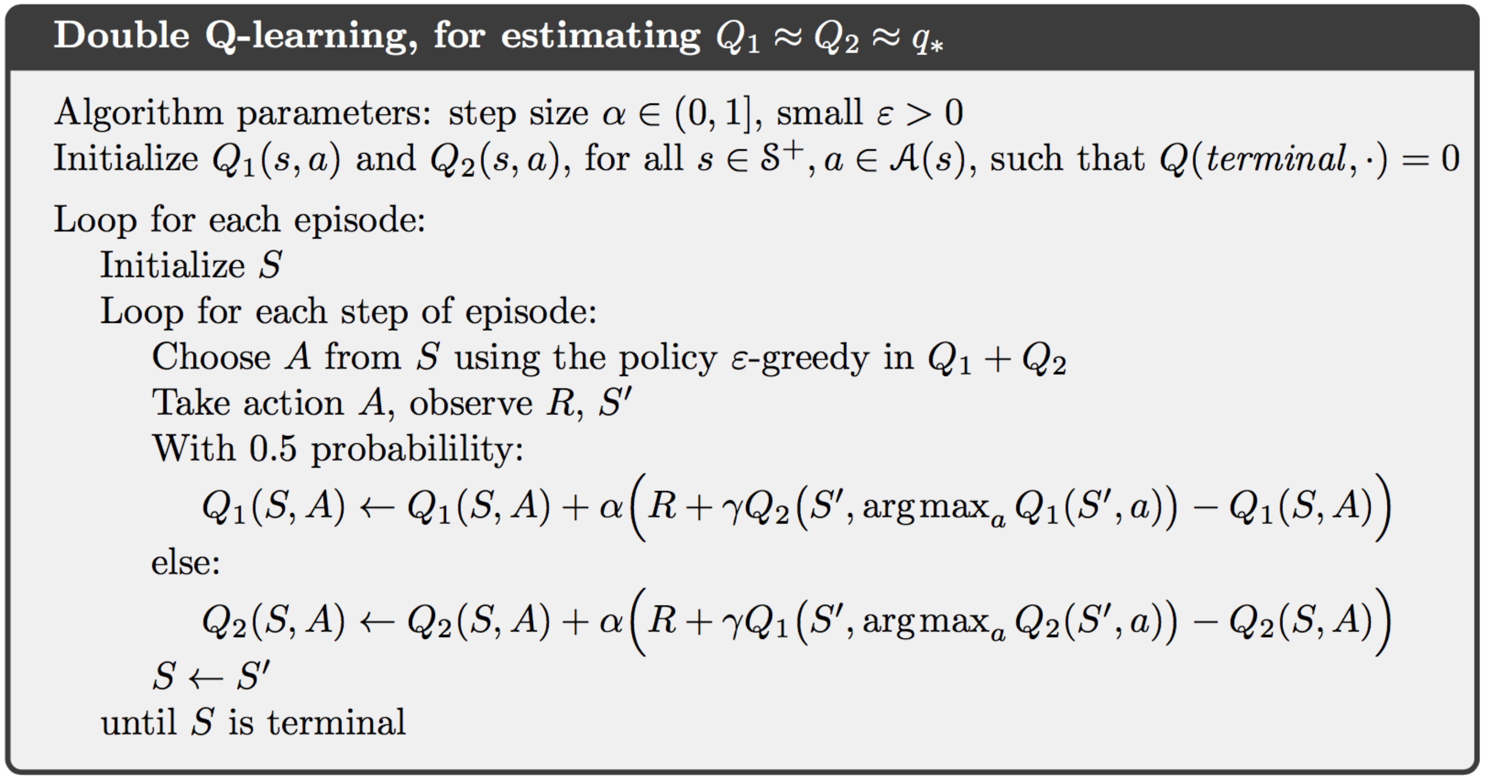


Figure 6.5: Comparison of Q-learning and Double Q-learning on a simple episodic MDP (shown inset). Q-learning initially learns to take the left action much more often than the right action, and always takes it significantly more often than the 5% minimum probability enforced by ε-greedy action selection with $\varepsilon = 0.1$. In contrast, Double Q-learning is essentially unaffected by maximization bias. These data are averaged over 10,000 runs. The initial action-value estimates were zero. Any ties in ε-greedy action selection were broken randomly.

[Sutton p134]

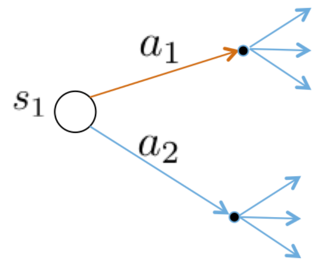
A fix: Double Q Learning



Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
   Initialize $S$
   Loop for each step of episode:
      Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
      Take action $A$, observe $R$, $S'$
      With 0.5 probabilility:
         $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
      else:
         $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
      $S \leftarrow S'$
   until $S$ is terminal
```

Additional understanding of experience-based updates of V(s) or Q(s,a)



$$V(s) = \mathbb{E}[r(s,a) + \gamma V(s')]$$

$$V(s) = \sum_{s',r} p(s',r|s,a)\left[r(s,a) + \gamma V(s')\right]$$

if model is known

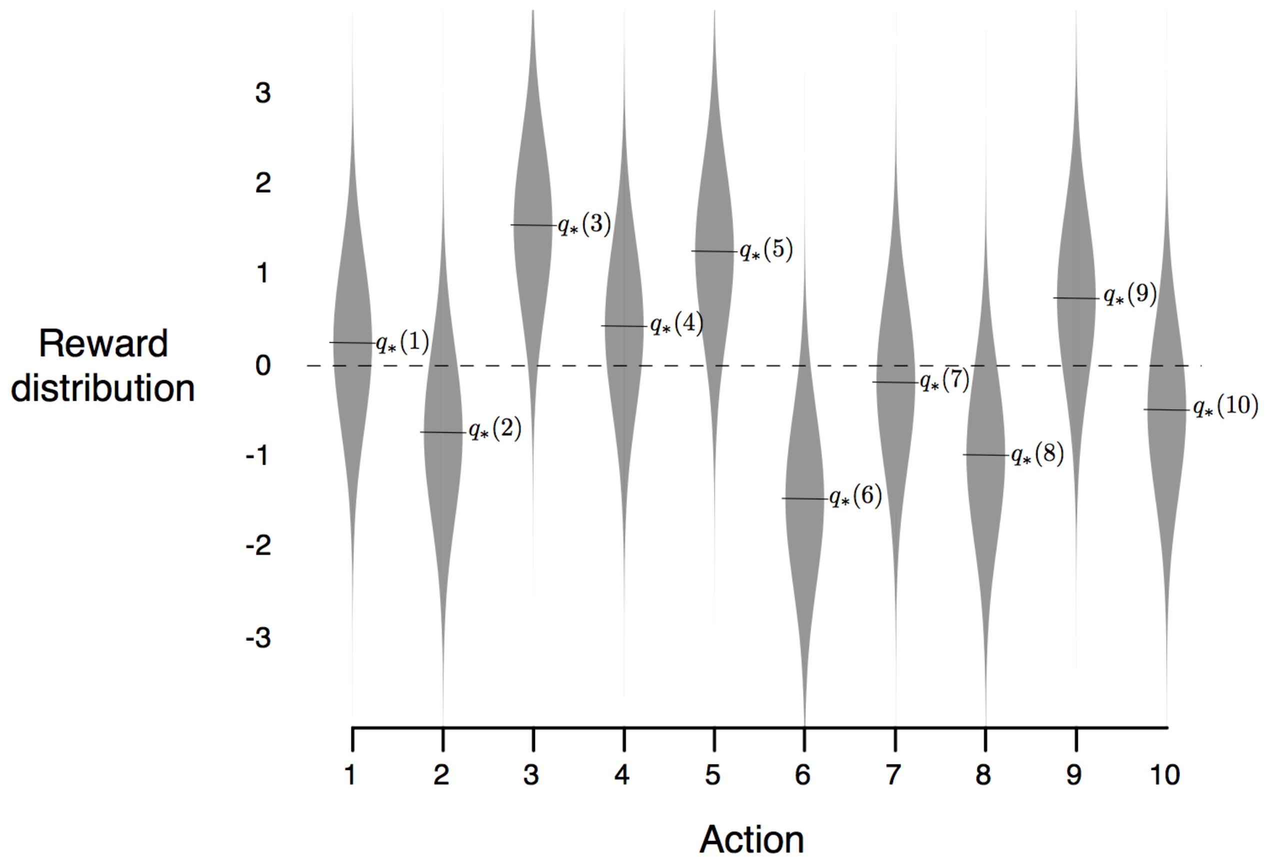
$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{\underbrace{r_{t+1} + \gamma V(S_{t+1})}_{\text{target}} - V(S_t)}_{\delta})$$

fixed fraction adjustment

$$V(S_t) \leftarrow V(S_t) + \alpha(\hat{V}(S_t) - V(S_t))$$

$$V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha\hat{V}(S_t)$$

Bandits



[Sutton, p28]

Q-functions for bandits

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

$$V(S_t) \leftarrow V(S_t) + \alpha(\hat{V}(S_t) - V(S_t))$$

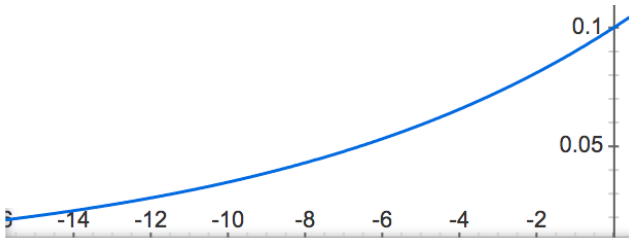
decreasing “step size” over time

constant “step size”

[Sutton, p31]

Updating using a constant step size

This gives more weight to recent rewards,
which is useful when tracking a non-stationary problem.



$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned}
$$

[Sutton, p32]

Loss-function view of a tabular value update

$$\min_V f(V) = (\hat{V} - V)^2$$

error squared

$$\frac{\partial f}{\partial V} = -2(\hat{V} - V)$$

$$V \leftarrow V - \frac{\alpha}{2}\frac{\partial f}{\partial V}$$

take a step size of 0.5 alpha, downhill

$$V \leftarrow V + \alpha(\hat{V} - V)$$

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

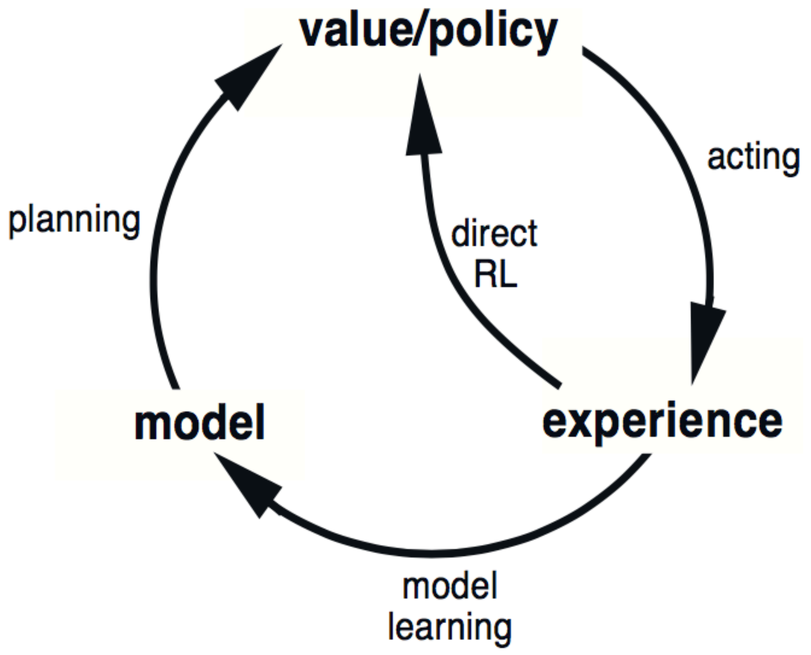
ϵ -Greedy Exploration (for discrete actions)

- 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

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

[Silver]

Basic Model-based RL



[Sutton, p162]

Basic Model-based RL

Tabular Dyna-Q

```
Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
    (a) $S \leftarrow$ current (nonterminal) state
    (b) $A \leftarrow \varepsilon$-greedy$(S, Q)$
    (c) Take action $A$; observe resultant reward, $R$, and state, $S'$
    (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
    (f) Loop repeat $n$ times:
            $S \leftarrow$ random previously observed state
            $A \leftarrow$ random action previously taken in $S$
            $R, S' \leftarrow Model(S,A)$
            $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
```

[Sutton, p164]

Scalability

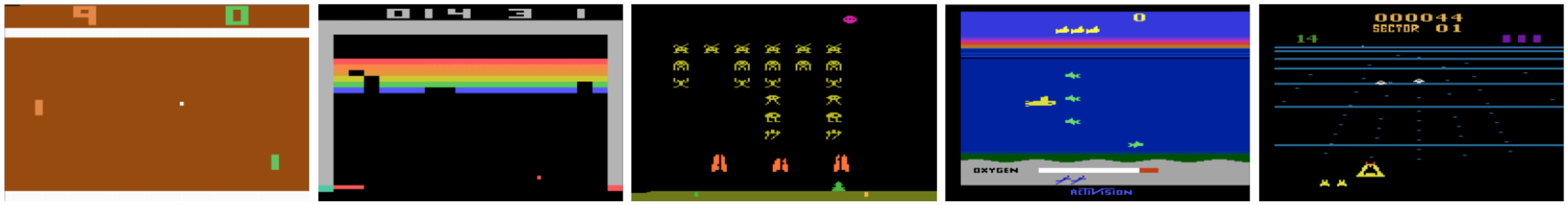
- simple problems
  - discrete states, discrete actions: store Q(s,a) in tabular form
- more interesting problems
  - Backgammon: 10^20 states
  - Go: 10^170 states
- continuous states and continuous actions ?
  - "curse of dimensionality"
- therefore need to approximate:
  - value functions and/or policy

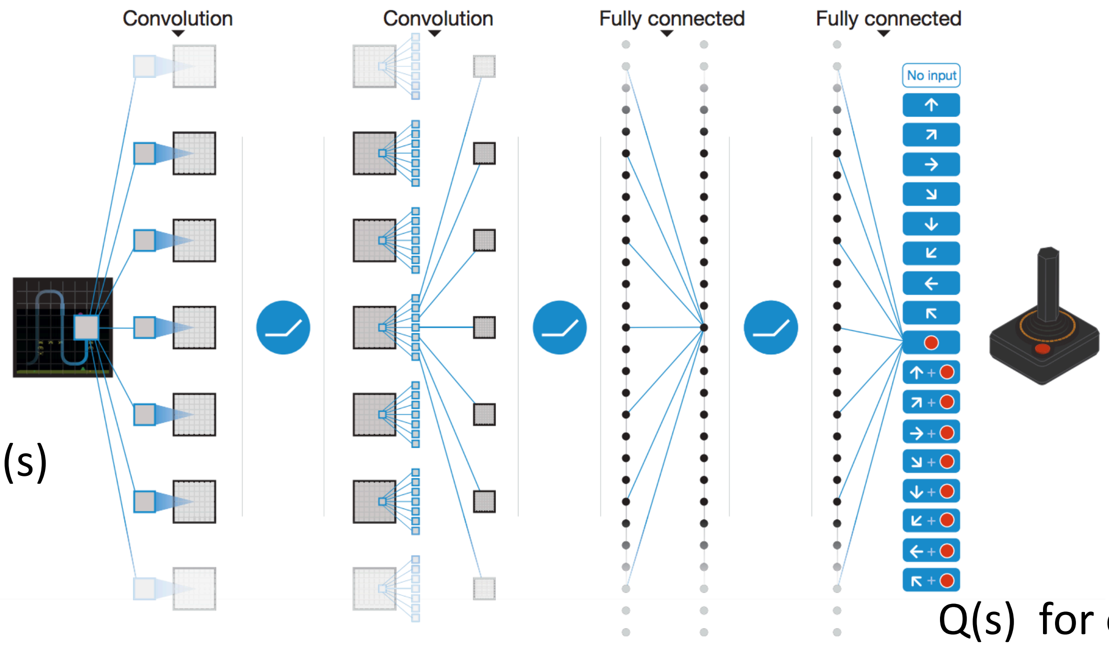
Deep Q-Networks (DQN)

Q-learning for discrete actions, off policy

Playing Atari with Deep Reinforcement Learning [NeurIPS 2013]

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



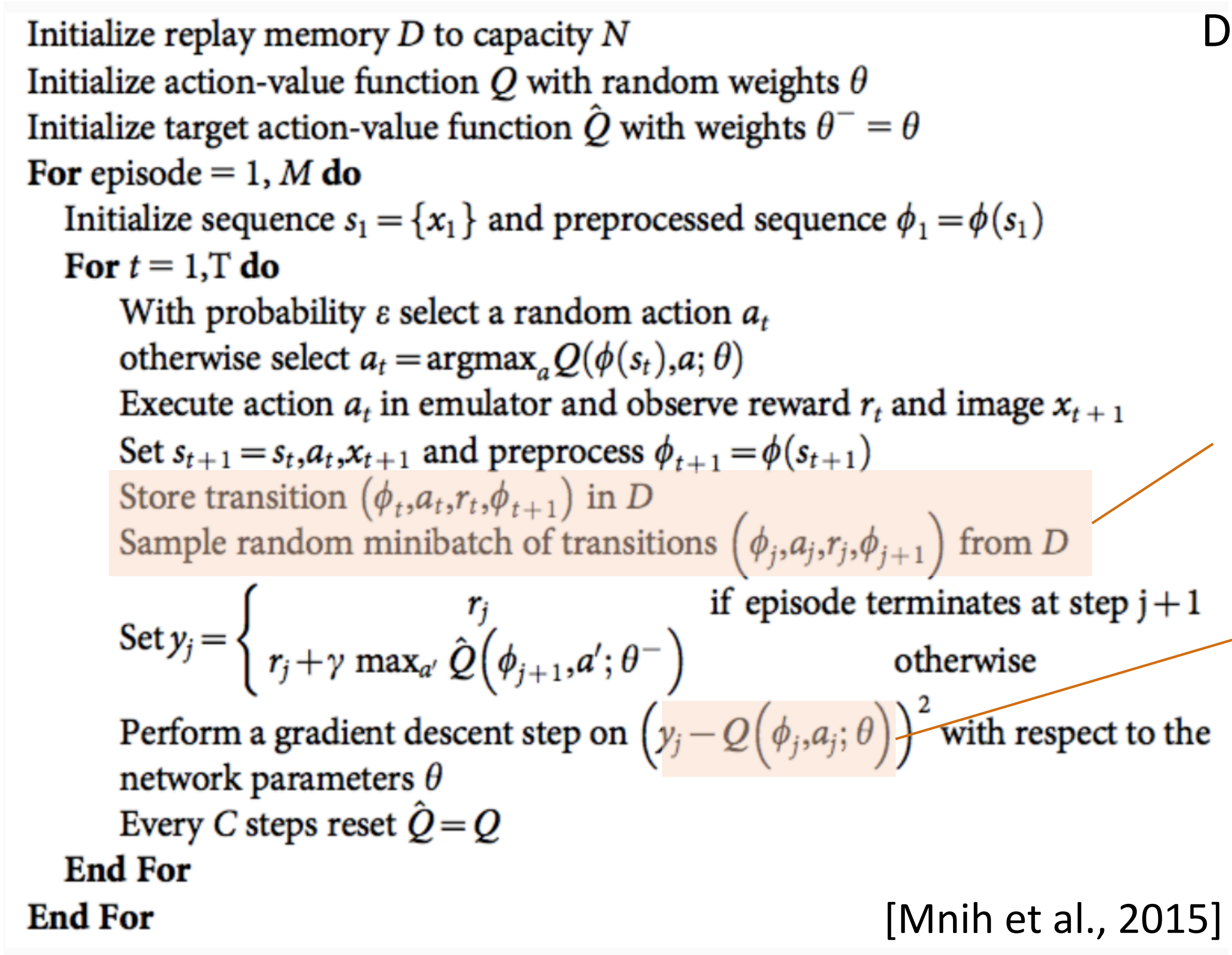


game image(s)

Q(s) for each action

[awjuliani]

DQN Algorithm



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the
        network parameters $\theta$
        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]

11.3 The Deadly Triad [Sutton, p264]

Our discussion so far can be summarized by saying that the danger of instability and divergence arises whenever we combine all of the following three elements, making up what we call *the deadly triad*:

Function approximation A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., linear function approximation or ANNs).

Bootstrapping Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods).

Off-policy training Training on a distribution of transitions other than that produced by the target policy. Sweeping through the state space and updating all states uniformly, as in dynamic programming, does not respect the target policy and is an example of off-policy training.

[Allows learning of multiple policies from a single stream of data]

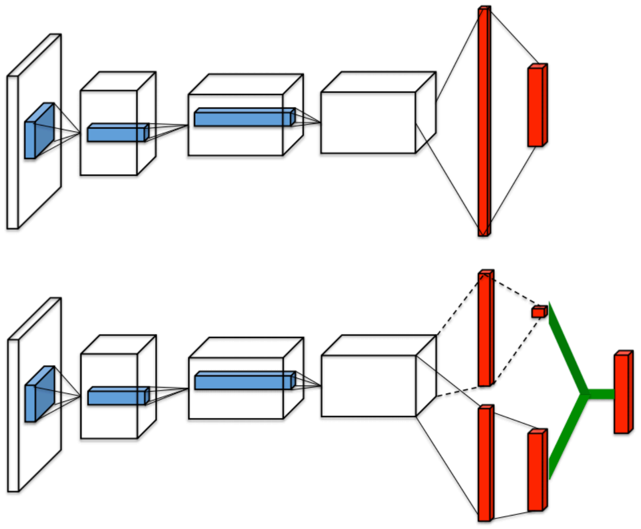
Experience Replay Buffer

- supervised learning for DNN assumes i.i.d. samples
  - independent and identically distributed samples
- a DNN learning update uses a mini-batch to compute the SGD update (SGD = stochastic gradient descent)
- mini-batch of the *n* most recent experience tuples is **not iid**
- solution
  - store all tuples in an experience replay buffer, then sample from that
- *prioritized experience replay*
  - train more on the data that is surprising,
    i.e., where the loss function is high

Dueling DQN

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \qquad \mathbb{E}_{a\sim\pi(s)}\left[A^{\pi}(s,a)\right] = 0.$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

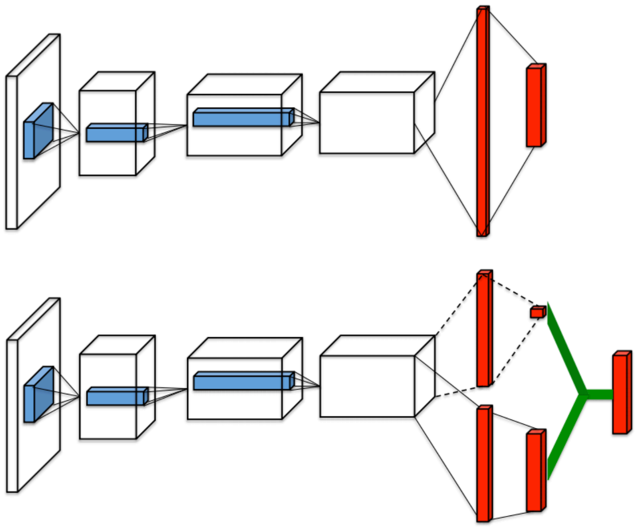


$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a'\in|\mathcal{A}|} A(s,a';\theta,\alpha)\right)$$

or

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

[Wang et al. 2015]



Distributional RL

$$Q(x,a) = \mathbb{E}\, R(x,a) + \gamma\, \mathbb{E}\, Q(X', A')$$

$$\Downarrow$$

$$Z(x,a) \overset{D}{=} R(x,a) + \gamma Z(X', A')$$

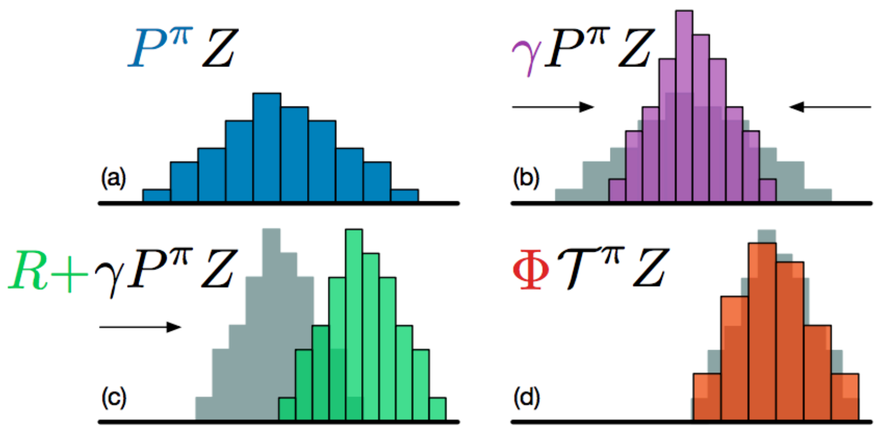
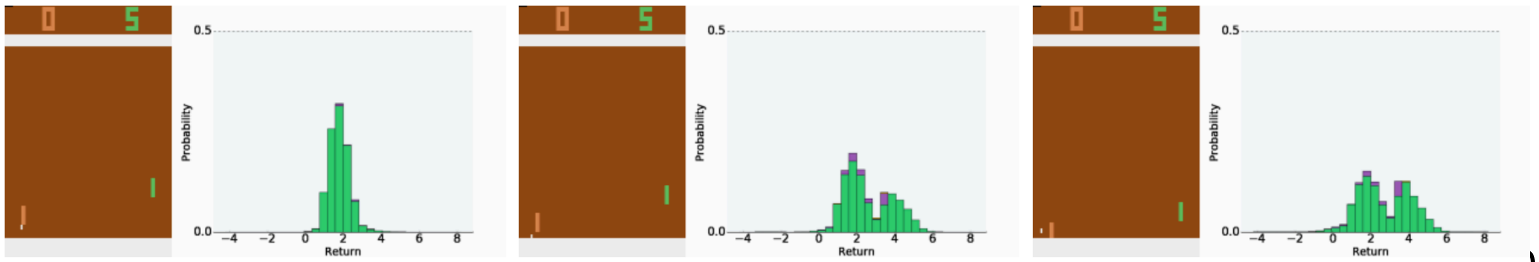
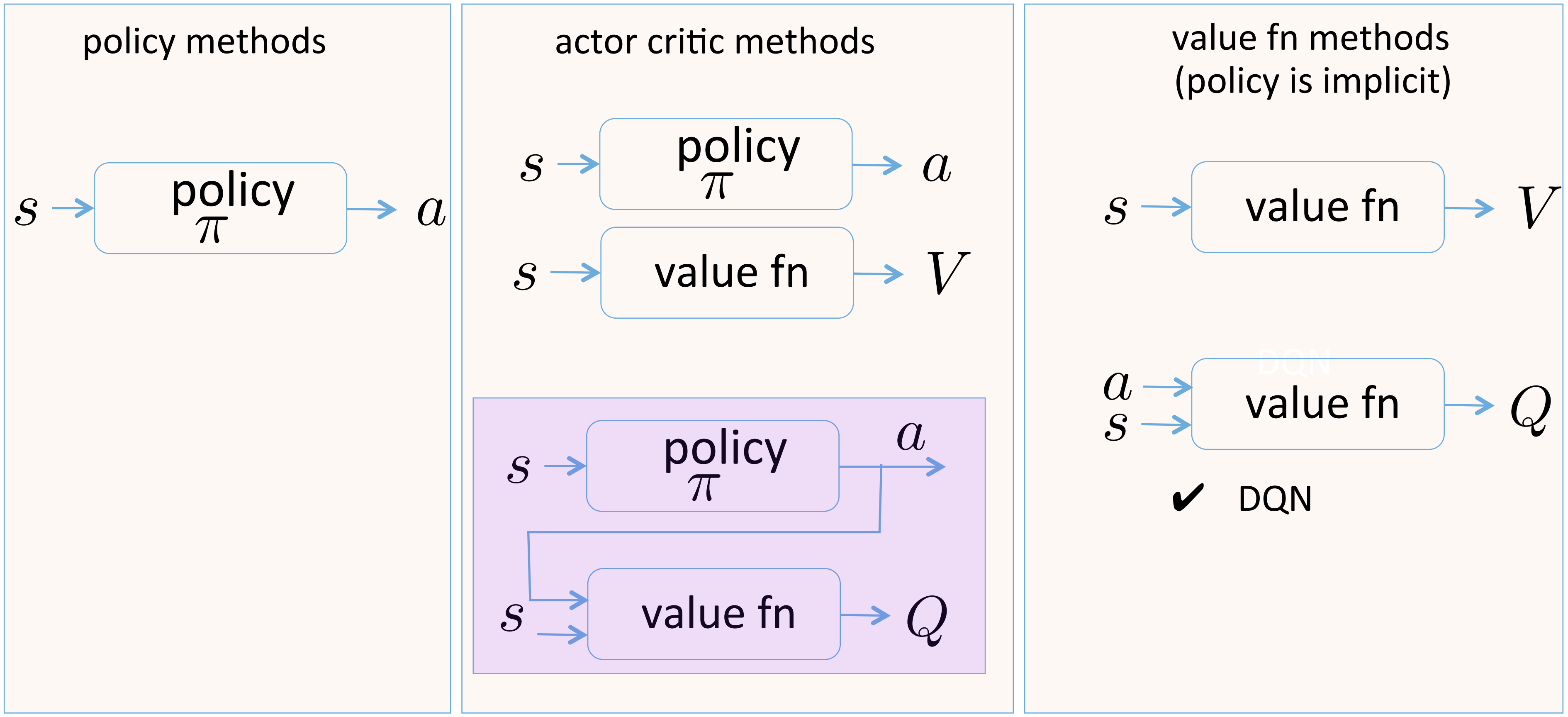


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).



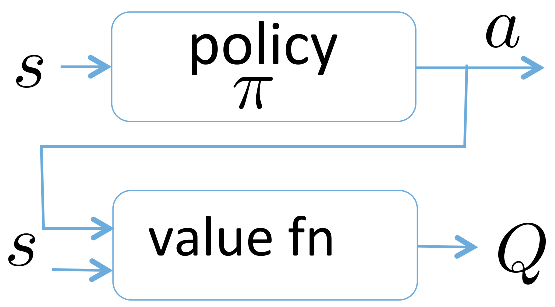
[Bellmare et al, 2017]

Deep RL: key building blocks



DDPG: Deep Deterministic Policy Gradients (off policy algorithm)

In continuous action spaces, greedy policy improvement becomes problematic, requiring a global maximisation at every step. Instead, a simple and computationally attractive alternative is to move the policy in the direction of the gradient of Q, rather than globally maximising Q.

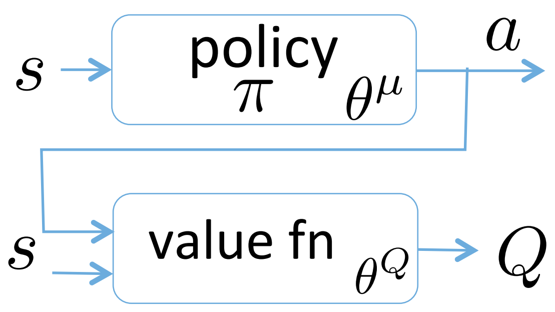


$$
\begin{aligned}
\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\
w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\
\theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \; \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}
\end{aligned}
$$

[DPG paper, Silver, ICML 2014]

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:

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

Ornstein-Uhlenbeck Process is used to add noise to the action output.

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