Introduction to Reinforcement Learning

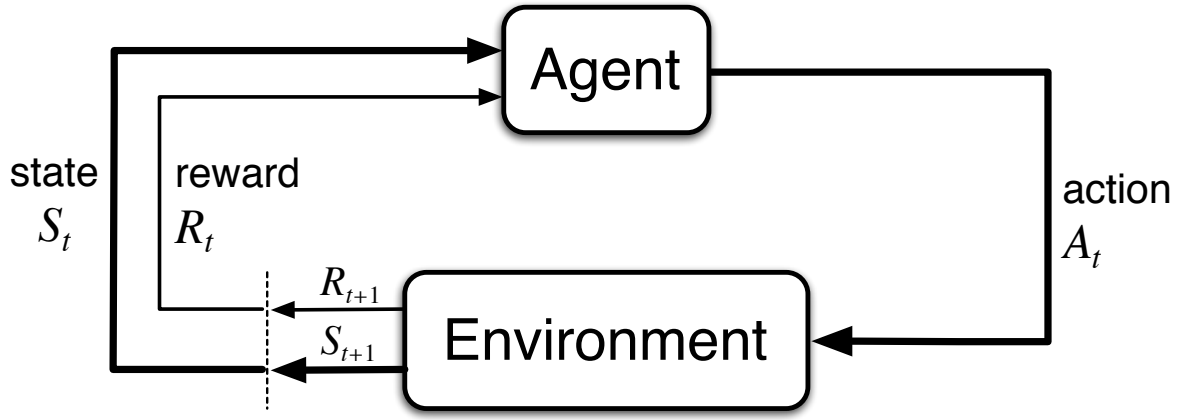
Part 2: Control (value-based and policy-based), Exploration, Knowledge representation

Doina Precup
McGill University/Mila and DeepMind Montreal
dprecup@cs.mcgill.ca

With thanks to Rich Sutton, Reasoning & Learning Lab at McGill/Mila, DeepMind

Based on Sutton & Barto, Reinforcement Learning: An Introduction (2nd ed), 2019

The Agent-Environment Interface



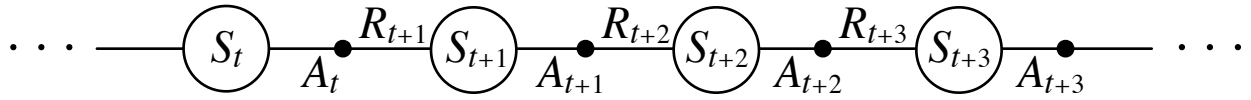
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$



Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \mathcal{S}$$

- There are always one or more policies that are better than or equal to all the others. These are the **optimal policies**. We denote them all π_*.
- Optimal policies share the same **optimal state-value function**:

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in \mathcal{S}$$

- Optimal policies also share the same **optimal action-value function**: $q_*(s,a) = \max_\pi q_\pi(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$

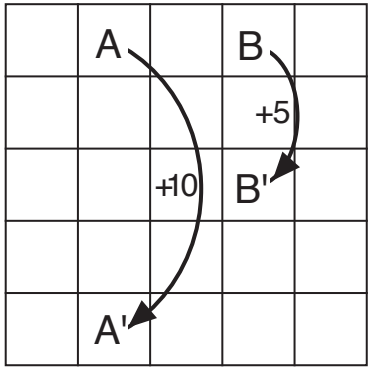
  This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to v_* is an optimal policy.

Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.

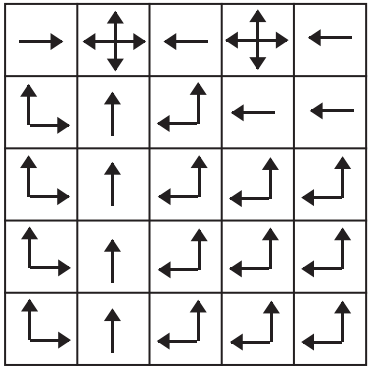
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

What About Optimal Action-Value Functions?

Given q_*, the agent does not even have to do a one-step-ahead search:

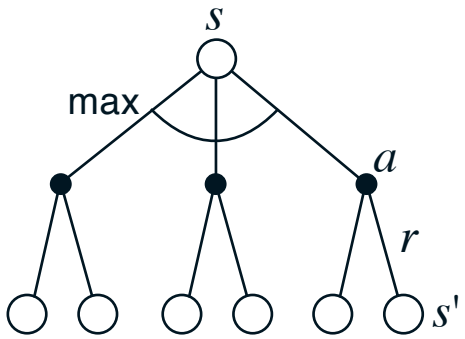
$$\pi_*(s) = \arg\max_a q_*(s,a)$$

Bellman Optimality Equation for v_*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$
\begin{aligned}
v_*(s) &= \max_a q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

The relevant backup diagram:



v_* is the unique solution of this system of nonlinear equations.

Value Iteration

Recall the **full policy-evaluation backup**:

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big] \qquad \forall s \in \mathcal{S}$$

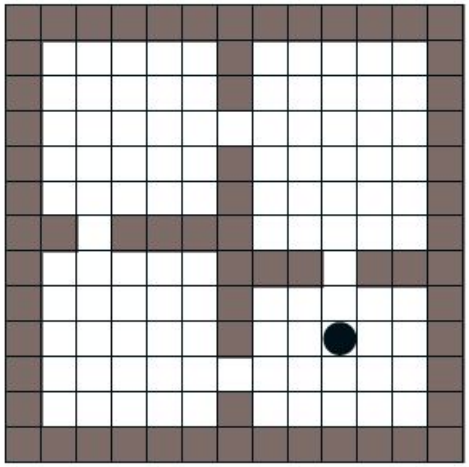
Here is the **full value-iteration backup**:

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big] \qquad \forall s \in \mathcal{S}$$

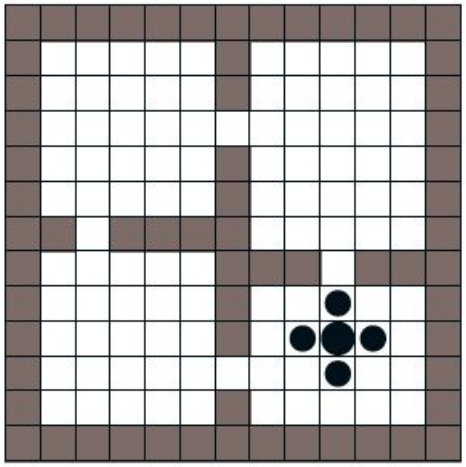
Illustration: Rooms Example

Four actions, fail 30% of the time

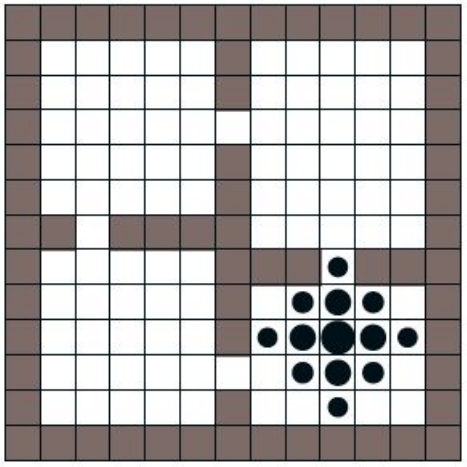
No rewards until the goal is reached, $\gamma = 0.9$.



Iteration #1



Iteration #2

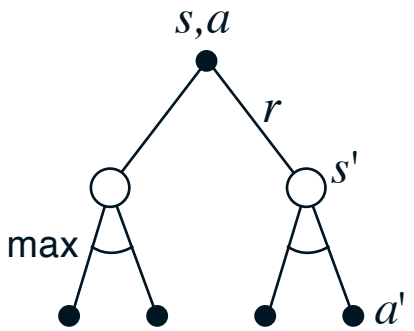


Iteration #3

Bellman Optimality Equation for q_*

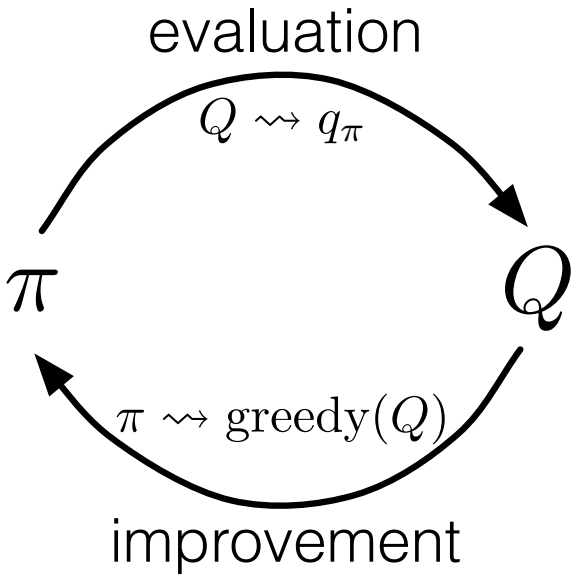
$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

The relevant backup diagram:



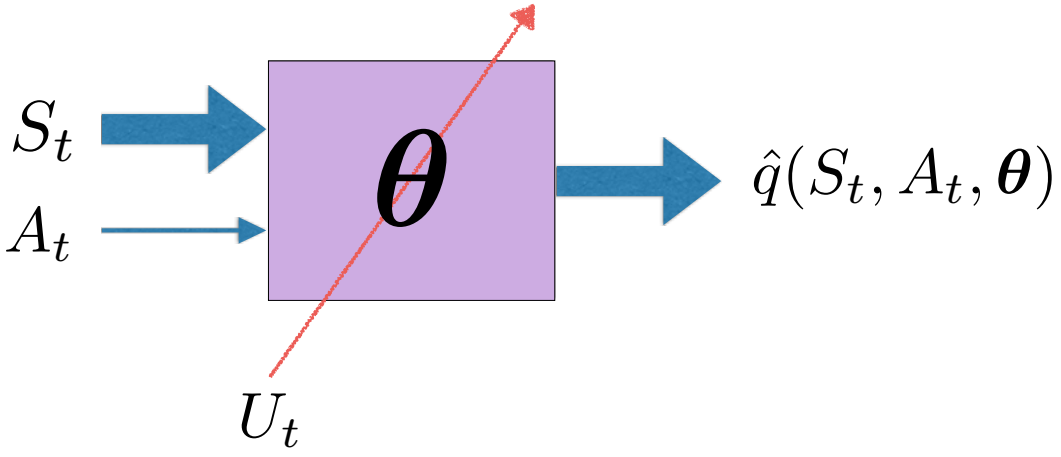
q_* is the unique solution of this system of nonlinear equations.

Policy iteration



- Policy evaluation using MC, TD, eligibility traces…
- Policy improvement step: greedify with respect to value (or action-value) function $A_t = \arg\max_a \hat{q}(S_t, a)$

Value function approximation (VFA) for control



The Need to Explore

- If the agent is taking the currently greedy action, we say it is exploiting its knowledge
- But the agent also needs to try actions that are currently not optimal, in order to learn about them, ie. explore:
- Exploration needs to persist if the environment might change

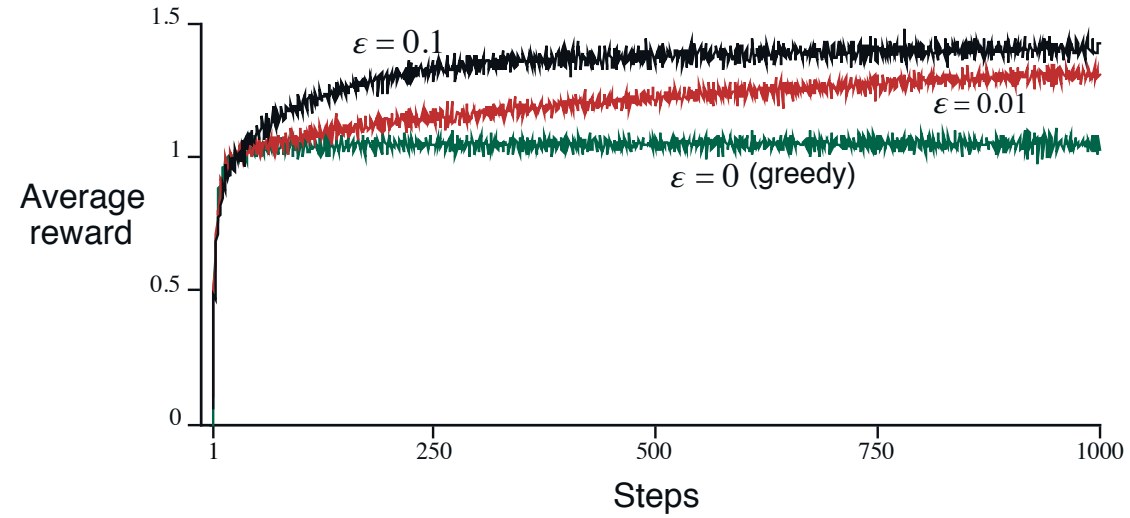
Types of exploration

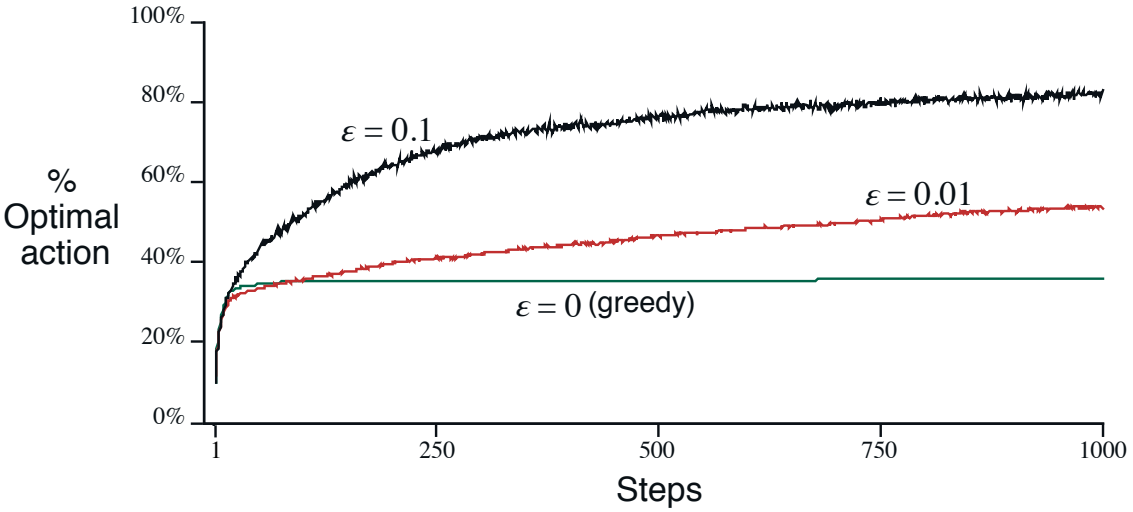
- Randomization: add noise to the greedy policy (eg epsilon-greedy)
- Optimism in the face of uncertainty: prefer actions with high uncertainty (eg UCB)
- Probability matching: select actions according to their probability of being best (eg Thompson sampling)
- Plan to explore (eg Bayes-adaptive MDP)

ε-Greedy Action Selection

- In greedy action selection, you always exploit
- In ε-greedy, you are usually greedy, but with probability ε you instead pick an action at random (possibly the greedy action again)
- This is perhaps the simplest way to balance exploration and exploitation

Illustration of ε-Greedy Methods





Sarsa

- Always learn the action-value function of the current policy
- Always act near-greedily wrt the current action-value estimates
- The learning rule is: $\theta_{t+1} = \theta_t + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \nabla Q(S_t, A_t)$

(Semi-)gradient methods for on-policy control

- Always learn the action-value function of the current policy
- Always act near-greedily wrt the current action-value estimates

- The learning rule is:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \Big[U_t - \hat{q}(S_t, A_t, \boldsymbol{\theta}_t) \Big] \nabla \hat{q}(S_t, A_t, \boldsymbol{\theta}_t)$$

update target, e.g.,

$$U_t = G_t \quad \text{(MC)}$$

$$U_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \boldsymbol{\theta}_t) \quad \text{(Sarsa)}$$

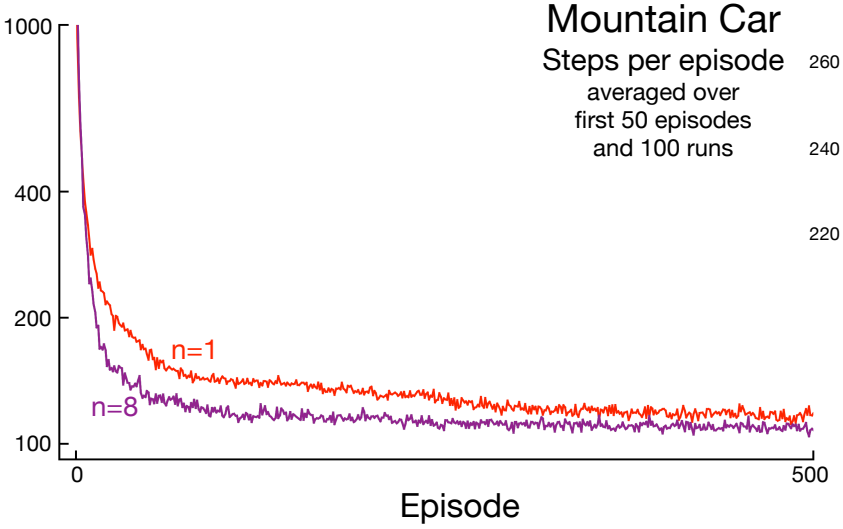
$$U_t = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}(S_{t+1}, a, \boldsymbol{\theta}_t) \quad \text{(Expected Sarsa)}$$

$$U_t = \sum_{s',r} p(s', r|S_t, A_t) \Big[r + \gamma \sum_{a'} \pi(a'|s') \hat{q}(s', a', \boldsymbol{\theta}_t) \Big] \quad \text{(DP)}$$

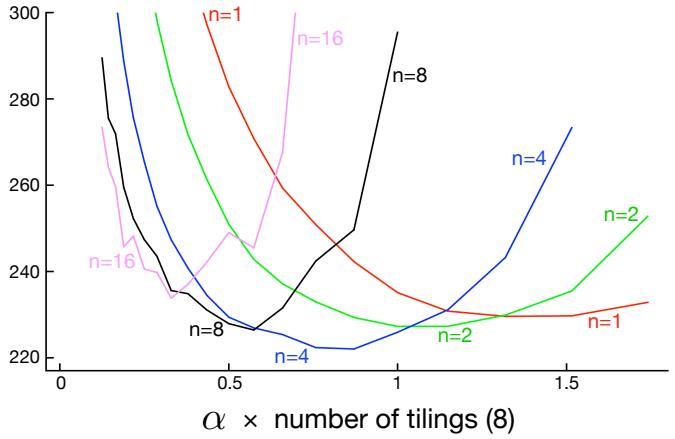
n-step semi-gradient Sarsa is better for *n*>1

$$\boldsymbol{\theta}_{t+n} \doteq \boldsymbol{\theta}_{t+n-1} + \alpha \left[G_t^{(n)} - \hat{q}(S_t, A_t, \boldsymbol{\theta}_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t, \boldsymbol{\theta}_{t+n-1}), \qquad 0 \leq t < T$$

Mountain Car
Steps per episode
log scale
averaged over 100 runs



Mountain Car
Steps per episode
averaged over
first 50 episodes
and 100 runs



Summing up on-policy control

- Control is straightforward in the on-policy case
- Formal results (bounds) exist for the linear, on-policy case (eg. Gordon, 2000, Perkins & Precup, 2003 and follow-up work)
    - we get chattering near a good solution, not convergence

Q-learning

- Value iteration for action values:

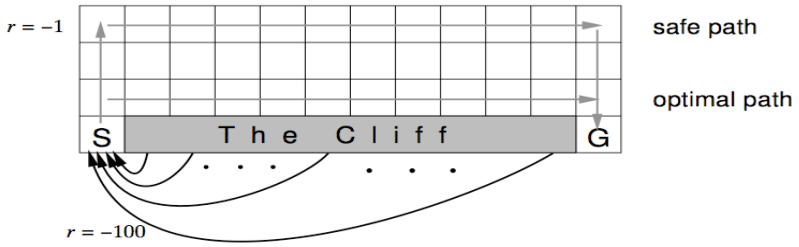
$$q_*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} q_*(s',a')$$

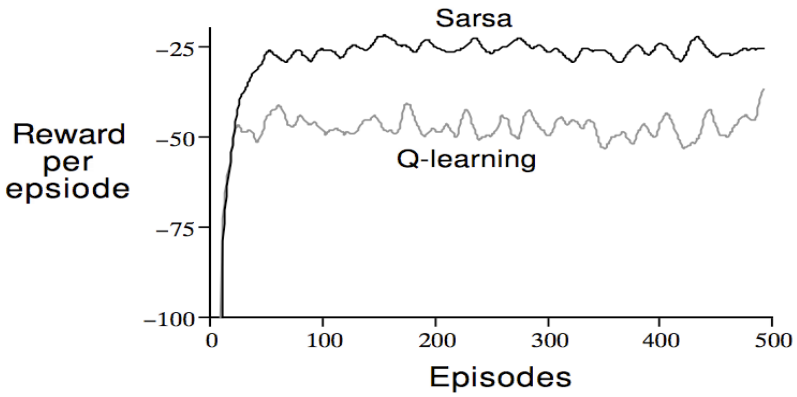
- This is *off-policy*: No matter what the policy does, we imagine the best possible course of action
- Q-learning: uses the sampled, bootstrapped corresponding target

$$U_t = R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a)$$

- In the tabular case, converges to q_* *even if the behavior policy is random*!
- Theoretical properties with function approximation are very bad but empirical results are very strong

Sarsa vs Q-learning



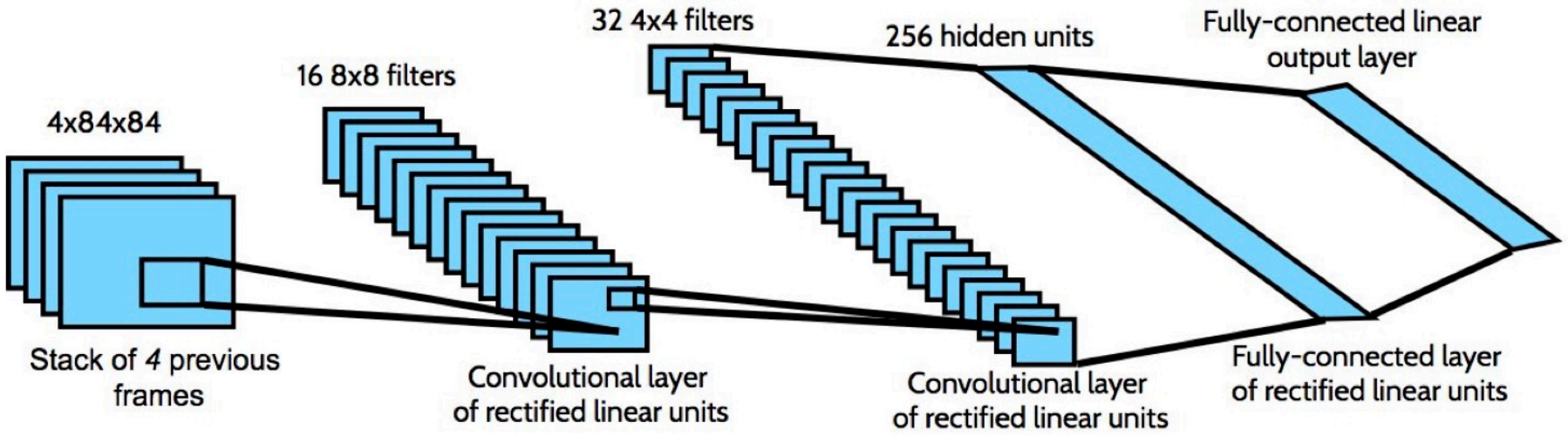


DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

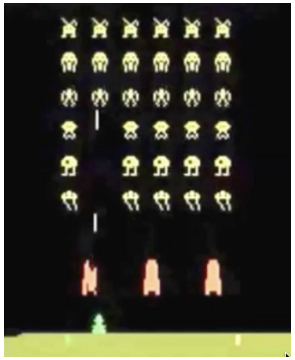
- Learns to play video games **from raw pixels**, simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$



RL + Deep Learning, applied to Classic Atari Games

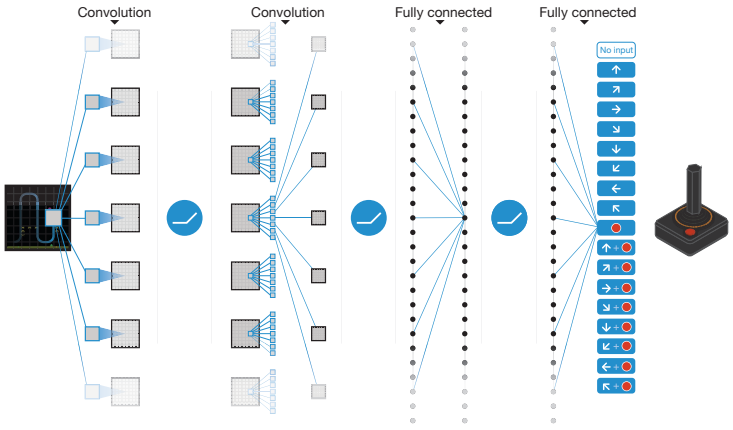
Google Deepmind 2015, Bowling et al. 2012





- Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone

mapping raw screen pixels



to predictions of final score for each of 18 joystick actions

- Learned to play better than all previous algorithms and at human level for more than half the games

Same learning algorithm applied to all 49 games!
w/o human tuning

DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Learns to play video games **from raw pixels**, simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Core components of DQN include:
  - Target networks (Mnih et al. 2015)

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}^-) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

  - Experience replay (Lin 1992): replay previous tuples (s, a, r, s')

Target Network Intuition

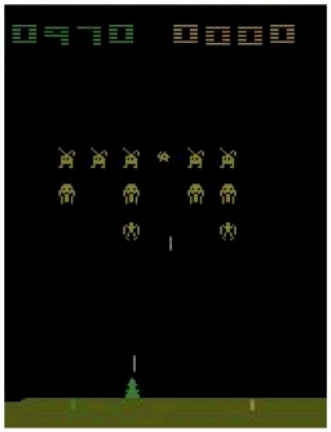
(Slide credit: Vlad Mnih)

- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

s

s'



DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

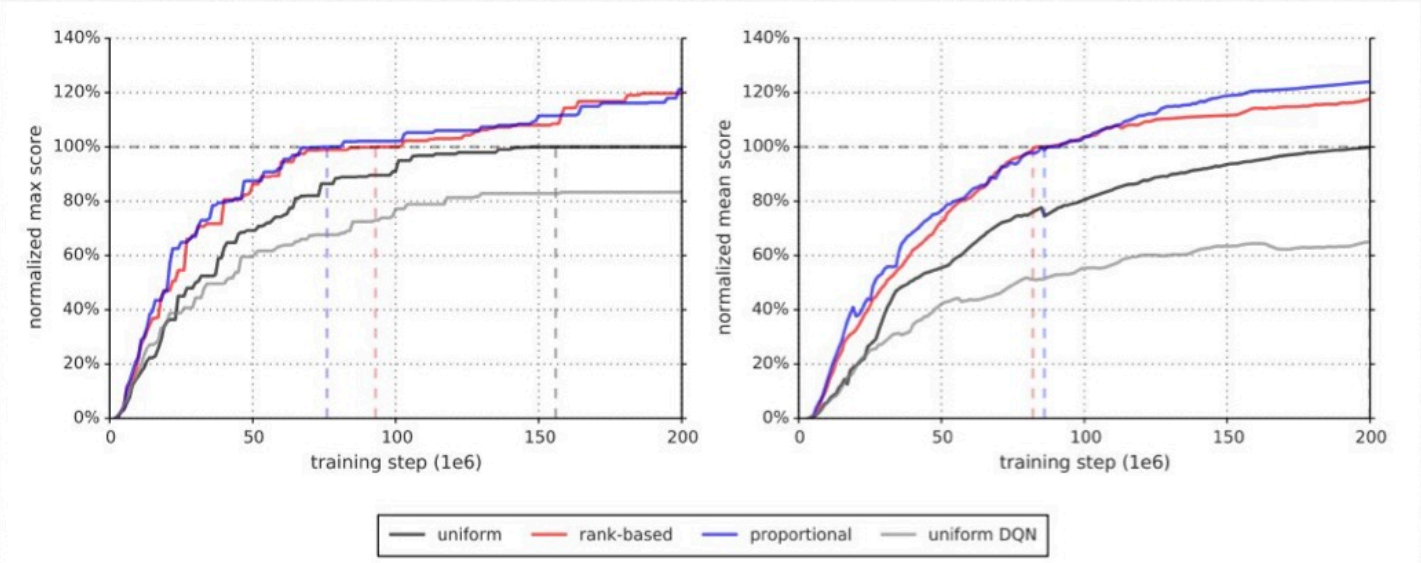
- Many later improvements to DQN
  - Double Q-learning (van Hasselt 2010, van Hasselt et al. 2015)
  - Prioritized replay (Schaul et al. 2016)
  - Dueling networks (Wang et al. 2016)
  - Asynchronous learning (Mnih et al. 2016)
  - Adaptive normalization of values (van Hasselt et al. 2016)
  - Better exploration (Bellemare et al. 2016, Ostrovski et al., 2017, Fortunato, Azar, Piot et al. 2017)
  - Distributional losses (Bellemare et al. 2017)
  - Multi-step returns (Mnih et al. 2016, Hessel et al. 2017)
  - … many more …

Prioritized Experience Replay

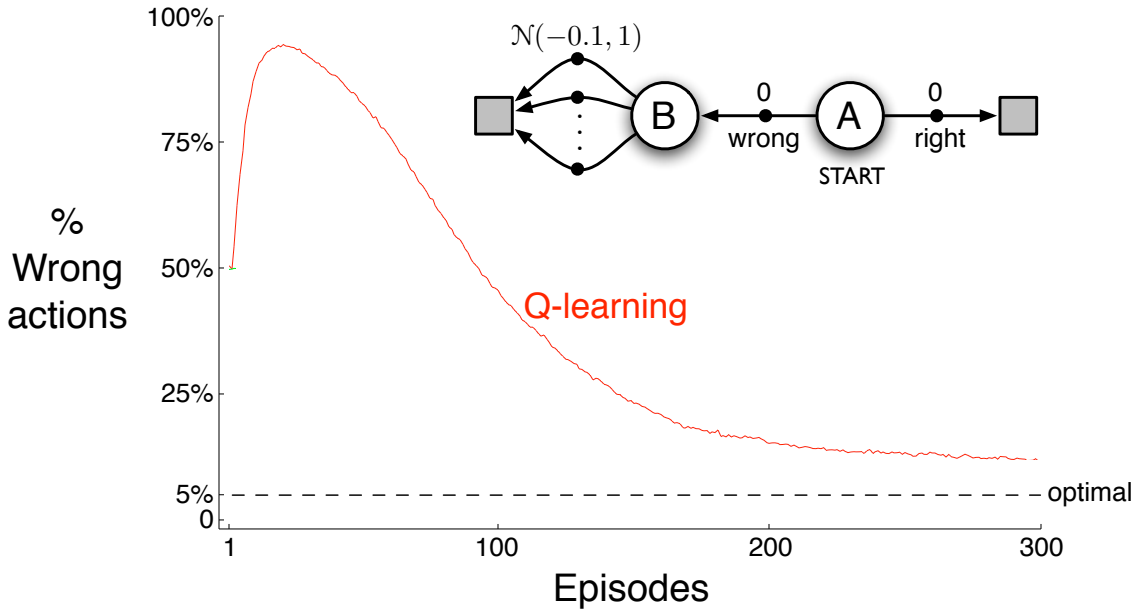
"Prioritized Experience Replay", Schaul et al. (2016)

- **Idea**: Replay transitions in proportion to TD error:

$$\left| r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right|$$



Maximization Bias Example



Tabular Q-learning: $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Double Q-Learning

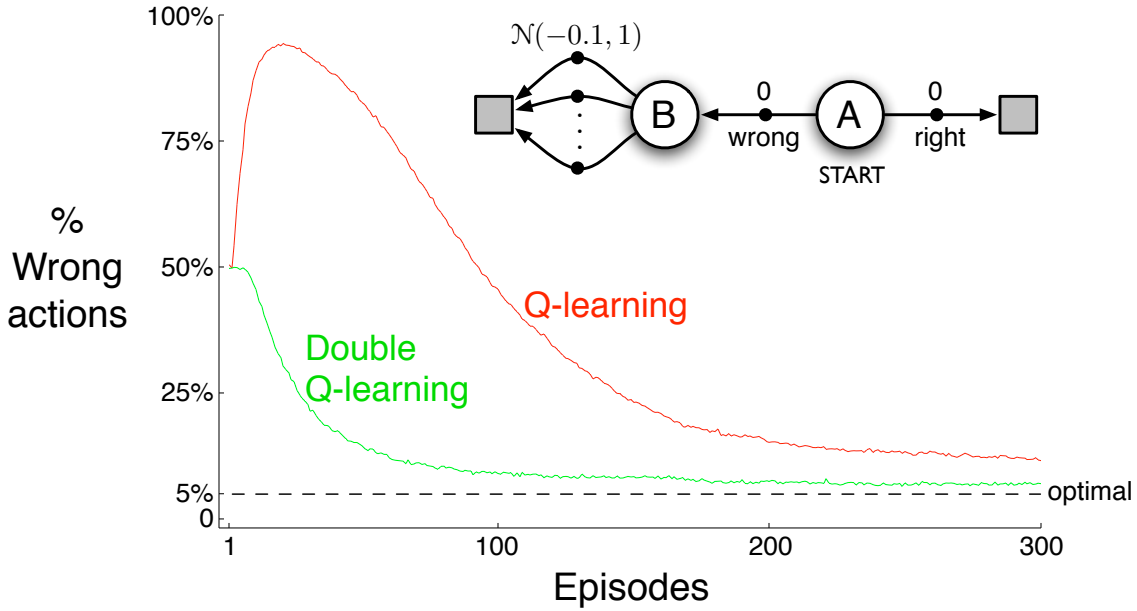
Hado van Hasselt 2010

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are indep.)
  - pick Q_1 or Q_2 at random to be updated on each step
- If updating Q_1, use Q_2 for the value of the next state and vice versa

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big(R_{t+1} + Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big)$$

- Action selections are (say) ε-greedy with respect to the sum of Q_1 and Q_2

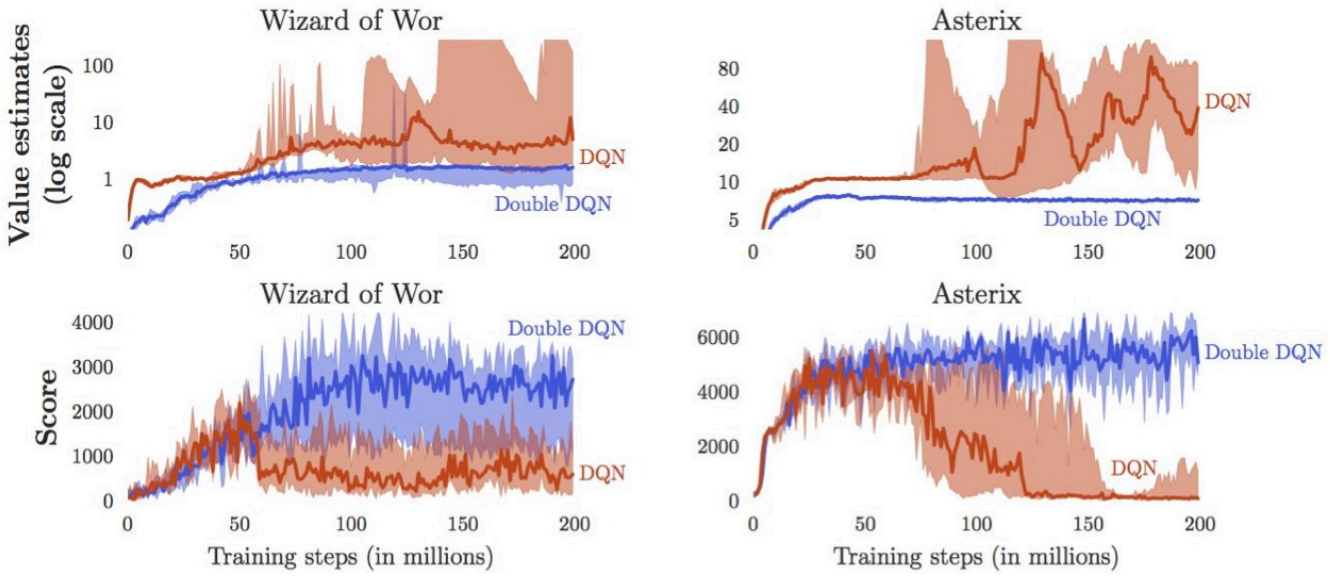
Example of Maximization Bias



Double Q-learning:

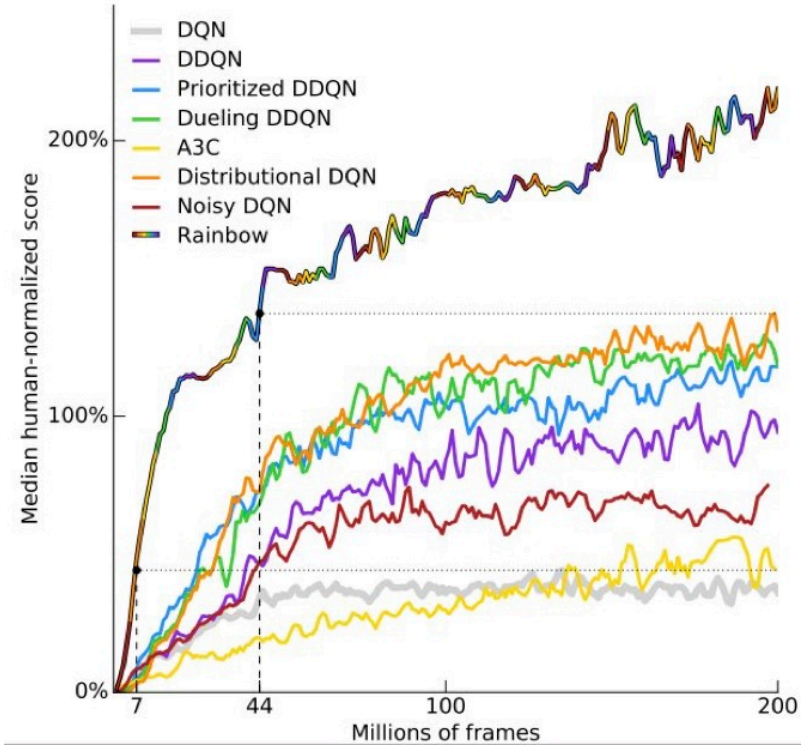
$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Double DQN



cf. van Hasselt et al, 2015)

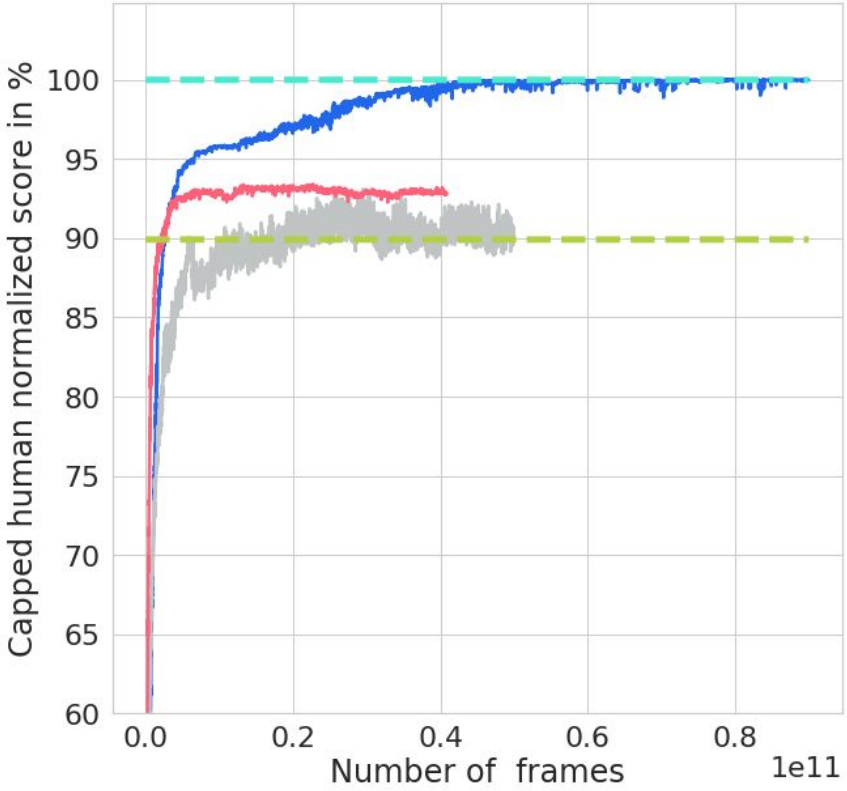
DQN improvements



Rainbow model, (Hessel et al, 2017)

Agent57

(AP Badia et al, 2020)

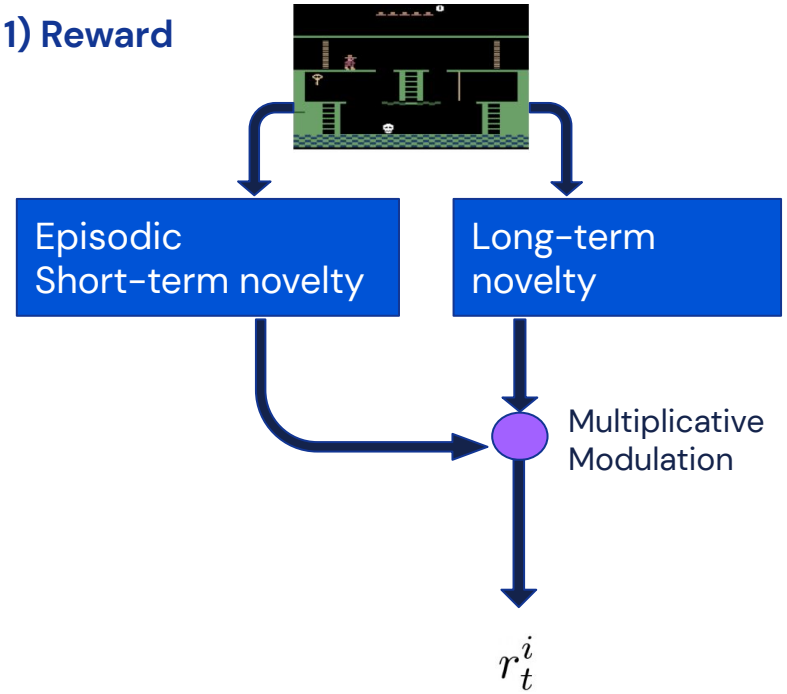


Main Improvement/Components:

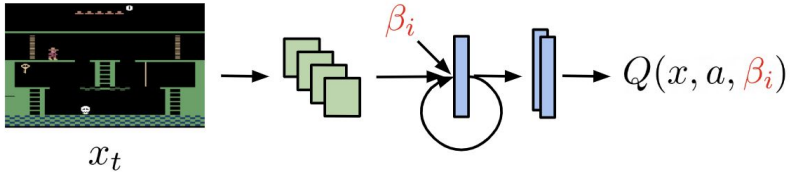
- Novelty-based intrinsic reward (NGU) (AP Badia et al, 2019)
- (Diversity) Train multiple profiles of behaviours based on trading off intrinsic reward + extrinsic reward
- (Diversity) Different discount factors
- (Adaptation) Non-stationary adaptive bandit to choose these behaviours (Schaul, Borsa, et al. 2019)

Agent57 (Never-Give-Up Reward)

1) Reward



2) Agent



$$Q(x, a, \beta_i) \leftarrow\cdots r_t^{\beta_i} = r_t^e + \beta_i r_t^i$$

- Each β_j has an associated discount γ_j
- (β_j, γ_j) sampled uniformly across actors.

Approaches to control

1. Previous approach: *Action-value methods*:
   - learn the value of each action;
   - pick the max (usually)
2. New approach: *Policy-gradient methods*:
   - learn the parameters of a stochastic policy
   - update by gradient ascent in performance
   - includes *actor-critic methods*, which learn *both* value and policy parameters

Why approximate policies rather than values?

- In many problems, the policy is simpler to approximate than the value function
- In many problems, the optimal policy is stochastic
  - e.g., bluffing, POMDPs
- To enable smoother change in policies
- To avoid a search on every step (the max)
- To better relate to biology

Gradient-bandit algorithm

- Store action preferences $H_t(a)$
  rather than action-value estimates $Q_t(a)$
- Instead of ε-greedy, pick actions by an exponential soft-max:

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- Also store the sample average of rewards as $\bar{R}_t$
- Then update:

$$H_{t+1}(a) = H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbf{1}_{a=A_t} - \pi_t(a)\big)$$

$\frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t)$

1 or 0, depending on whether the predicate (subscript) is true

Derivation of gradient-bandit algorithm

In exact *gradient ascent*:

$$H_{t+1}(a) \doteq H_t(a) + \alpha \frac{\partial\, \mathbb{E}\,[R_t]}{\partial H_t(a)}, \tag{1}$$

where:

$$\mathbb{E}[R_t] \doteq \sum_b \pi_t(b) q_*(b),$$

$$\begin{aligned}
\frac{\partial\, \mathbb{E}[R_t]}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \left[\sum_b \pi_t(b) q_*(b) \right] \\
&= \sum_b q_*(b) \frac{\partial\, \pi_t(b)}{\partial H_t(a)} \\
&= \sum_b \left(q_*(b) - X_t \right) \frac{\partial\, \pi_t(b)}{\partial H_t(a)},
\end{aligned}$$

where X_t does not depend on b, because $\sum_b \frac{\partial\, \pi_t(b)}{\partial H_t(a)} = 0$.

$$
\begin{aligned}
\frac{\partial \mathbb{E}[R_t]}{\partial H_t(a)} &= \sum_b \big(q_*(b) - X_t\big) \frac{\partial \pi_t(b)}{\partial H_t(a)} \\
&= \sum_b \pi_t(b) \big(q_*(b) - X_t\big) \frac{\partial \pi_t(b)}{\partial H_t(a)} / \pi_t(b) \\
&= \mathbb{E}\left[\big(q_*(A_t) - X_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right] \\
&= \mathbb{E}\left[\big(R_t - \bar{R}_t\big) \frac{\partial \pi_t(A_t)}{\partial H_t(a)} / \pi_t(A_t) \right],
\end{aligned}
$$

where here we have chosen $X_t = \bar{R}_t$ and substituted R_t for $q_*(A_t)$, which is permitted because $\mathbb{E}[R_t | A_t] = q_*(A_t)$.

For now assume: $\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)\big(\mathbf{1}_{a=b} - \pi_t(a)\big)$. Then:

$$
\begin{aligned}
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \pi_t(A_t) \big(\mathbf{1}_{a=A_t} - \pi_t(a)\big) / \pi_t(A_t) \big] \\
&= \mathbb{E}\big[\big(R_t - \bar{R}_t\big) \big(\mathbf{1}_{a=A_t} - \pi_t(a)\big) \big].
\end{aligned}
$$

$$
H_{t+1}(a) = H_t(a) + \alpha\big(R_t - \bar{R}_t\big)\big(\mathbf{1}_{a=A_t} - \pi_t(a)\big), \text{ (from (1), QED)}
$$

Thus it remains only to show that

$$\frac{\partial \pi_t(b)}{\partial H_t(a)} = \pi_t(b)\big(\mathbf{1}_{a=b} - \pi_t(a)\big).$$

Recall the standard quotient rule for derivatives:

$$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}.$$

Using this, we can write...

Quotient Rule: $$\frac{\partial}{\partial x}\left[\frac{f(x)}{g(x)}\right] = \frac{\frac{\partial f(x)}{\partial x} g(x) - f(x)\frac{\partial g(x)}{\partial x}}{g(x)^2}$$

$$\begin{aligned}
\frac{\partial \pi_t(b)}{\partial H_t(a)} &= \frac{\partial}{\partial H_t(a)} \pi_t(b) \\
&= \frac{\partial}{\partial H_t(a)} \left[\frac{e^{H_t(b)}}{\sum_{c=1}^k e^{H_t(c)}}\right] \\
&= \frac{\frac{\partial e^{H_t(b)}}{\partial H_t(a)} \sum_{c=1}^k e^{H_t(c)} - e^{H_t(b)} \frac{\partial \sum_{c=1}^k e^{H_t(c)}}{\partial H_t(a)}}{\left(\sum_{c=1}^k e^{H_t(c)}\right)^2} && \text{(Q.R.)} \\
&= \frac{\mathbf{1}_{a=b} e^{H_t(a)} \sum_{c=1}^k e^{H_t(c)} - e^{H_t(b)} e^{H_t(a)}}{\left(\sum_{c=1}^k e^{H_t(c)}\right)^2} && \left(\frac{\partial e^x}{\partial x} = e^x\right) \\
&= \frac{\mathbf{1}_{a=b} e^{H_t(b)}}{\sum_{c=1}^k e^{H_t(c)}} - \frac{e^{H_t(b)} e^{H_t(a)}}{\left(\sum_{c=1}^k e^{H_t(c)}\right)^2} \\
&= \mathbf{1}_{a=b} \pi_t(b) - \pi_t(b)\pi_t(a) \\
&= \pi_t(b)\left(\mathbf{1}_{a=b} - \pi_t(a)\right). && \text{(Q.E.D.)}
\end{aligned}$$

Gradient-bandit algorithms on the 10-armed testbed

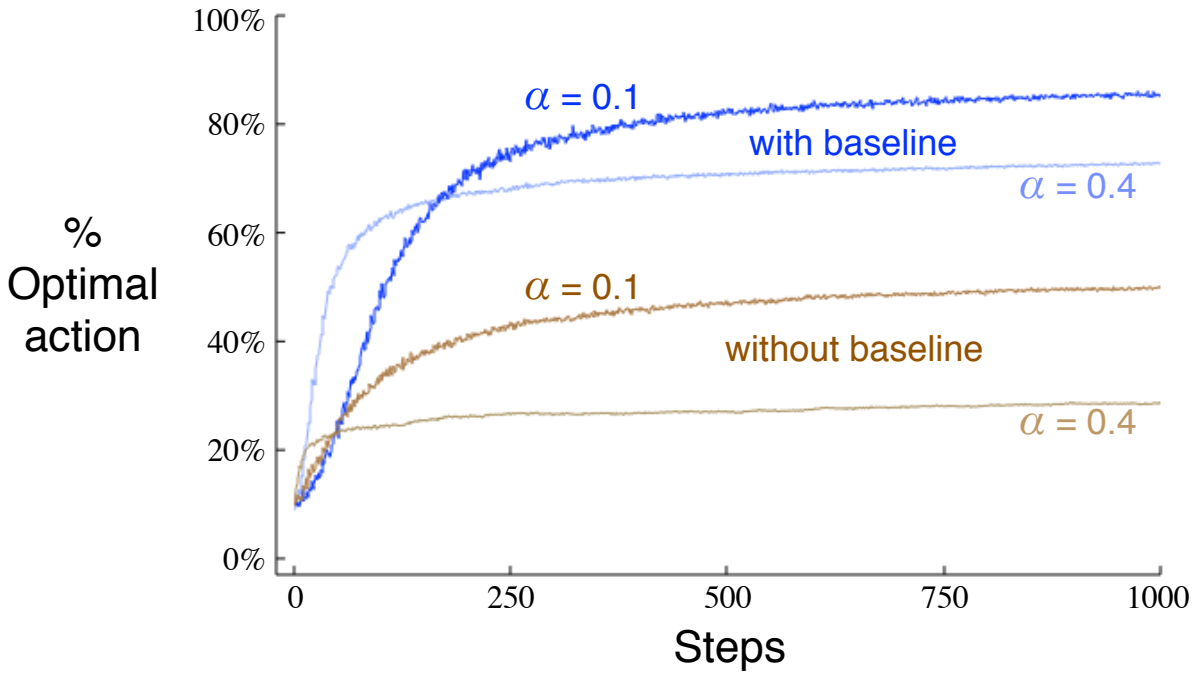


Figure 2.6: Average performance of the gradient-bandit algorithm with and without a reward baseline on the 10-armed testbed when the $q_*(a)$ are chosen to be near $+4$ rather than near zero.

Example: linear-exponential policies (discrete actions)

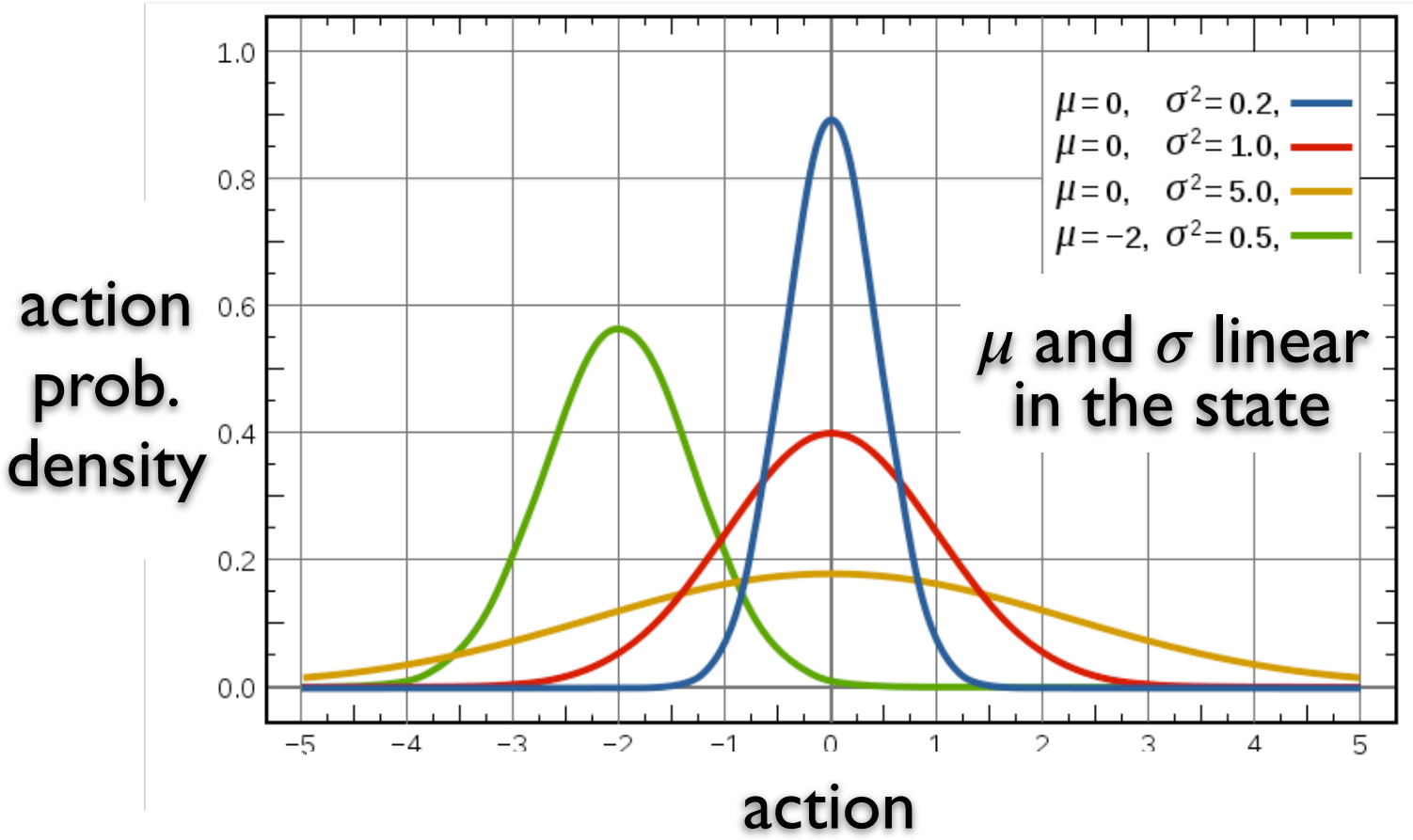
- The "preference" for action a in state s is linear in θ and a state-action feature vector $\phi(s,a)$
- The probability of action a in state s is exponential in its preference

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{\exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a))}{\sum_b \exp(\boldsymbol{\theta}^\top \boldsymbol{\phi}(s,b))}$$

- Corresponding *eligibility function*:

$$\frac{\nabla \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \boldsymbol{\phi}(s,a) - \sum_b \pi(b|s,\boldsymbol{\theta})\boldsymbol{\phi}(s,b)$$

Example: linear-gaussian policies (continuous actions)



eg, linear-gaussian policies (continuous actions)

- The mean and std. dev. for the action taken in state s are linear and linear-exponential in

$$\boldsymbol{\theta} \doteq (\boldsymbol{\theta}_\mu^\top; \boldsymbol{\theta}_\sigma^\top)^\top \qquad \mu(s) \doteq \boldsymbol{\theta}_\mu^\top \boldsymbol{\phi}(s) \qquad \sigma(s) \doteq \exp(\boldsymbol{\theta}_\sigma^\top \boldsymbol{\phi}(s)$$

- The probability density function for the action taken in state s is gaussian

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{1}{\sigma(s)\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s))^2}{2\sigma(s)^2}\right)$$

Gradient of Gaussian policy

$$\frac{\nabla_{\boldsymbol{\theta}_\mu} \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \frac{1}{\sigma(s)^2}(a - \mu(s))\boldsymbol{\phi}_\mu(s)$$

$$\frac{\nabla_{\boldsymbol{\theta}_\sigma} \pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \left(\frac{(a - \mu(s))^2}{\sigma(s)^2} - 1\right)\boldsymbol{\phi}_\sigma(s)$$

General policy-gradient setup

Given a policy parameterization:

$$\pi(a|s,\boldsymbol{\theta}) \qquad \frac{\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})}{\pi(a|s,\boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \log \pi(a|s,\boldsymbol{\theta})$$

And objective:

$$\eta(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(S_0) \quad \text{(or average reward)}$$

Approximate stochastic gradient ascent:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \widehat{\nabla \eta(\boldsymbol{\theta}_t)}$$

Typically, based on the Policy-Gradient Theorem:

$$\nabla \eta(\boldsymbol{\theta}) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

Deriving REINFORCE from the PGT

$$
\begin{aligned}
\nabla\eta(\boldsymbol{\theta}) &= \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}), \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a q_\pi(S_t,a) \nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta}) \right] \\
&= \mathbb{E}_\pi \left[\gamma^t \sum_a \pi(a|S_t,\boldsymbol{\theta}) q_\pi(S_t,a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|S_t,\boldsymbol{\theta})}{\pi(a|S_t,\boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[\gamma^t q_\pi(S_t,A_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \qquad \text{(replacing } a \text{ by the sample } A_t \sim \pi) \\
&= \mathbb{E}_\pi \left[\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})} \right] \qquad \text{(because } \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)\text{)}
\end{aligned}
$$

Thus

$$
\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \widehat{\nabla\eta(\boldsymbol{\theta}_t)} \doteq \boldsymbol{\theta}_t + \alpha \gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}
$$

REINFORCE with baseline

Policy-gradient theorem with baseline:

$$\nabla\eta(\boldsymbol{\theta}) = \sum_s d_\pi(s) \sum_a q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

$$= \sum_s d_\pi(s) \sum_a \Big(q_\pi(s,a) - b(s) \Big) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

any function of state, not action

Because

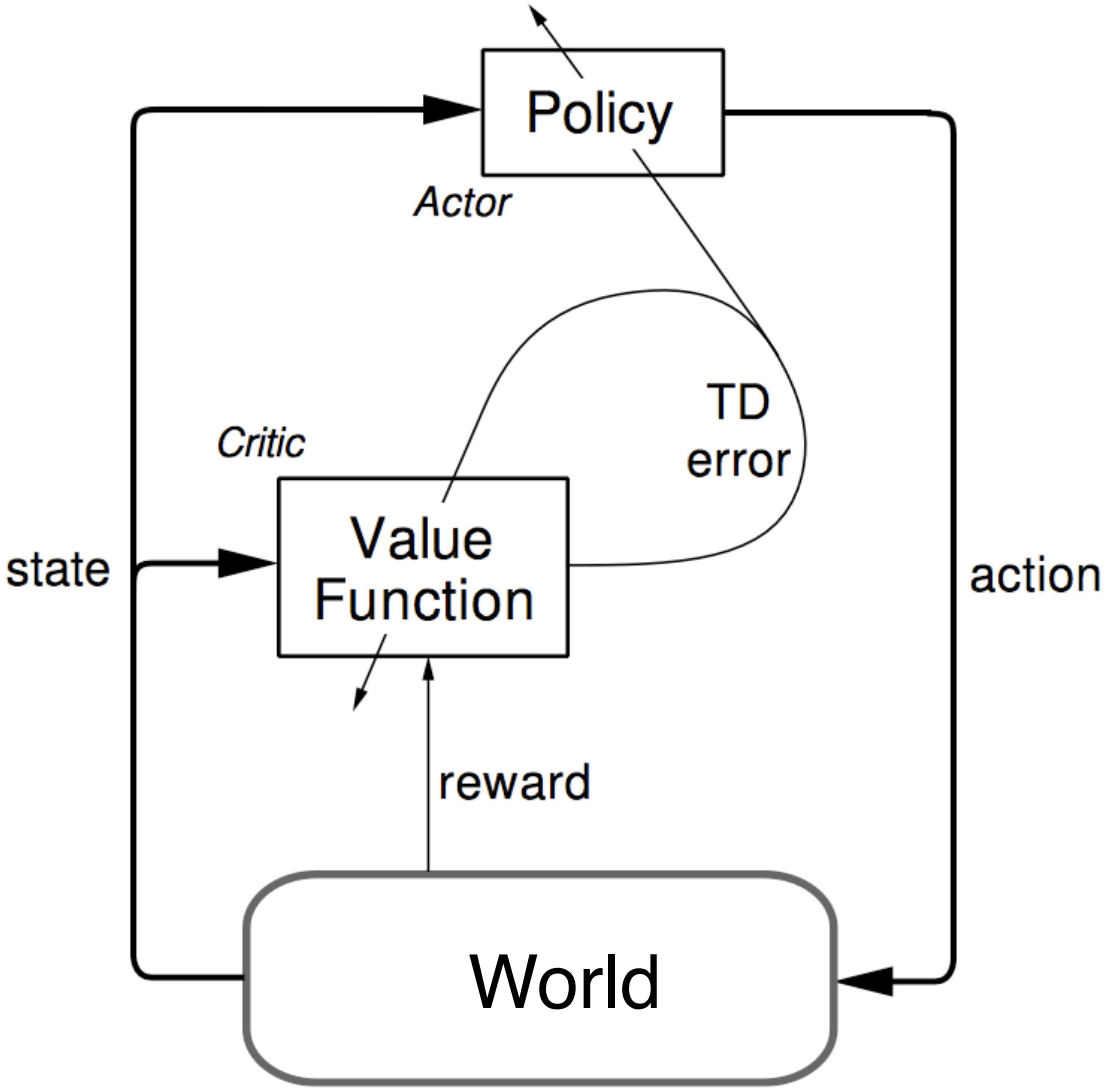
$$\sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} \sum_a \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} 1 = 0 \qquad \forall s \in \mathcal{S}$$

Thus

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \gamma^t \Big(G_t - b(S_t) \Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}$$

e.g., $b(s) = \hat{v}(s, \mathbf{w})$

Actor-critic architecture



Actor-Critic methods

REINFORCE with baseline:

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \gamma^t \Big(G_t - b(S_t) \Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}$$

Actor-Critic method:

$$\begin{aligned}
\boldsymbol{\theta}_{t+1} &\doteq \boldsymbol{\theta}_t + \alpha \gamma^t \Big(G_t^{(1)} - \hat{v}(S_t, \mathbf{w}) \Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \\
&= \boldsymbol{\theta}_t + \alpha \gamma^t \Big(R_{t+1} - \bar{R}_t + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}) \Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}
\end{aligned}$$

A2C and A3C

- Pseudocode

  for iteration$=1, 2, \ldots$ **do**

  Agent acts for T timesteps (e.g., $T = 20$),

  For each timestep t, compute

$$\hat{R}_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_t)$$
$$\hat{A}_t = \hat{R}_t - V(s_t)$$

  $\hat{R}_t$ is target value function, in regression problem

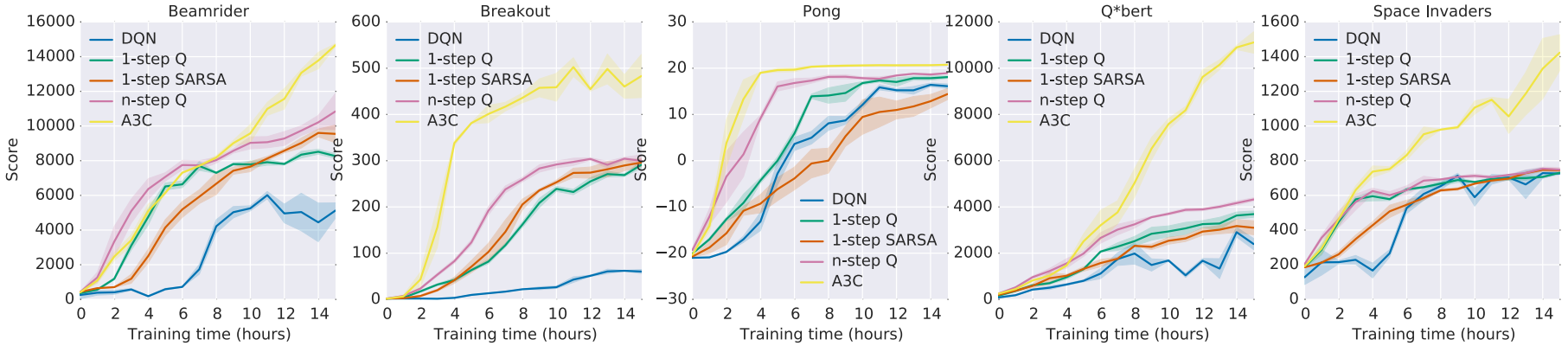
  $\hat{A}_t$ is estimated advantage function

  Compute loss gradient $g = \nabla_\theta \sum_{t=1}^{T} \left[-\log \pi_\theta(a_t \mid s_t) \hat{A}_t + c(V(s) - \hat{R}_t)^2 \right]$

  g is plugged into a stochastic gradient descent variant, e.g., Adam.

  end for

A3C results


Beamrider
Breakout
Pong
Q*bert
Space Invaders
DQN
1-step Q
1-step SARSA
n-step Q
A3C
Score
Training time (hours)

Revisiting the objective

- Let $\eta(\pi)$ denote the expected return of π
- We collect data with π_{old}. Want to optimize some objective to get a new policy π
- Define $L_{\pi_{\text{old}}}(\pi)$ to be the "surrogate objective"[1]

$$L(\pi) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi(a \mid s)}{\pi_{\text{old}}(a \mid s)} A^{\pi_{\text{old}}}(s, a)\right]$$

$$\nabla_\theta L(\pi_\theta)\big|_{\theta_{\text{old}}} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta_{\text{old}}} \quad \text{(policy gradient)}$$

- Local approximation to the performance of the policy; does not depend on parameterization of π

Trust Region Policy Optimization (TRPO)

- Constrained optimization problem

$$\max_{\pi} L(\pi), \text{ subject to } \overline{\mathrm{KL}}[\pi_{\mathrm{old}}, \pi] \leq \delta$$

$$\text{where } L(\pi) = \mathbb{E}_{\pi_{\mathrm{old}}} \left[\frac{\pi(a \mid s)}{\pi_{\mathrm{old}}(a \mid s)} A^{\pi_{\mathrm{old}}}(s, a) \right]$$

- Construct loss from empirical data

$$\hat{L}(\pi) = \sum_{n=1}^{N} \frac{\pi(a_n \mid s_n)}{\pi_{\mathrm{old}}(a_n \mid s_n)} \hat{A}_n$$

- Make quadratic approximation and solve with conjugate gradient algorithm

Proximal Policy Gradient (PPO)

- Use penalty instead of constraint

$$\underset{\theta}{\text{minimize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - \beta \overline{\text{KL}}[\pi_{\theta_{\text{old}}}, \pi_\theta]$$

- Pseudocode:

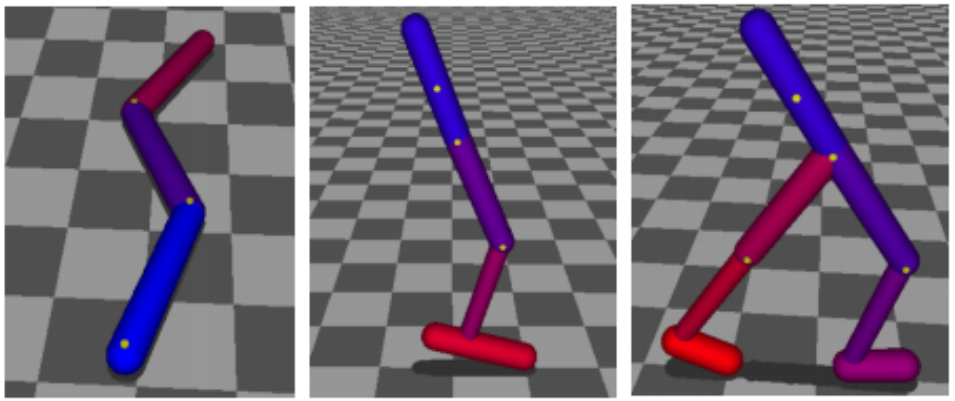
  for iteration=$1, 2, \ldots$ **do**
    Run policy for T timesteps or N trajectories
    Estimate advantage function at all timesteps
    Do SGD on above objective for some number of epochs
    If KL too high, increase β. If KL too low, decrease β.
  end for

- $\approx$ same performance as TRPO, but only first-order optimization

Results

Applied to

- Locomotion controllers in 2D



- Atari games with pixel input

Deep Deterministic Policy Gradient (DDPG)

- Incorporate replay buffer and target network ideas from DQN for increased stability
- Use lagged (Polyak-averaging) version of Q_ϕ and π_θ for fitting Q_ϕ (towards $Q^{\pi,\gamma}$) with TD(0)

$$\hat{Q}_t = r_t + \gamma Q_{\phi'}(s_{t+1}, \pi(s_{t+1}; \theta'))$$

- Pseudocode:

for iteration$=1, 2, \ldots$ **do**
  Act for several timesteps, add data to replay buffer
  Sample minibatch
  Update π_θ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
  Update Q_ϕ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$,
end for

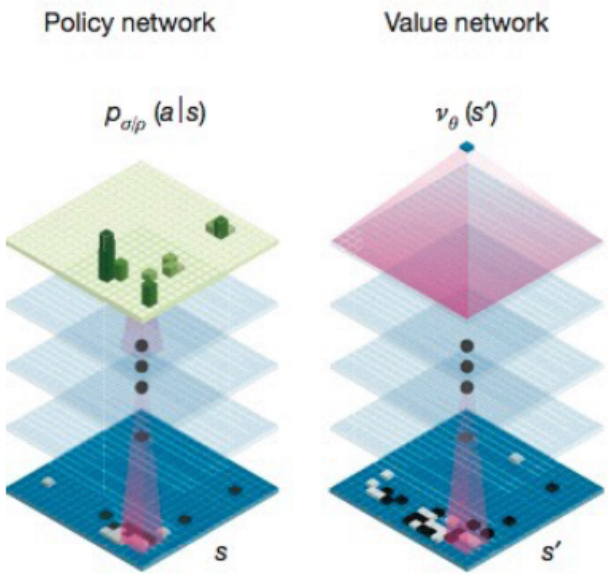
DDPG results

Applied to 2D and 3D robotics tasks and driving with pixel input

The generality of policy-gradient

- Can be applied whenever we can compute the effect of parameter changes on the action probabilities,
- E.g., has been applied to spiking neuron models
- There are many possibilities other than linear-exponential and linear-gaussian, e.g., mixture of random, argmax, and fixed-width gaussian; learn the mixing weights, drift/diffusion models
- Can be applied whenever we can compute the effect of parameter changes on the action probabilities, $\nabla \pi(A_t|S_t, \boldsymbol{\theta})$

Knowledge in AlphaGo



- *Policy*: what to do (probability of action given current "state") - ie *procedural knowledge*
- *Value function*: estimation of expected long-term return - ie *predictive knowledge*

Building Knowledge with Reinforcement Learning

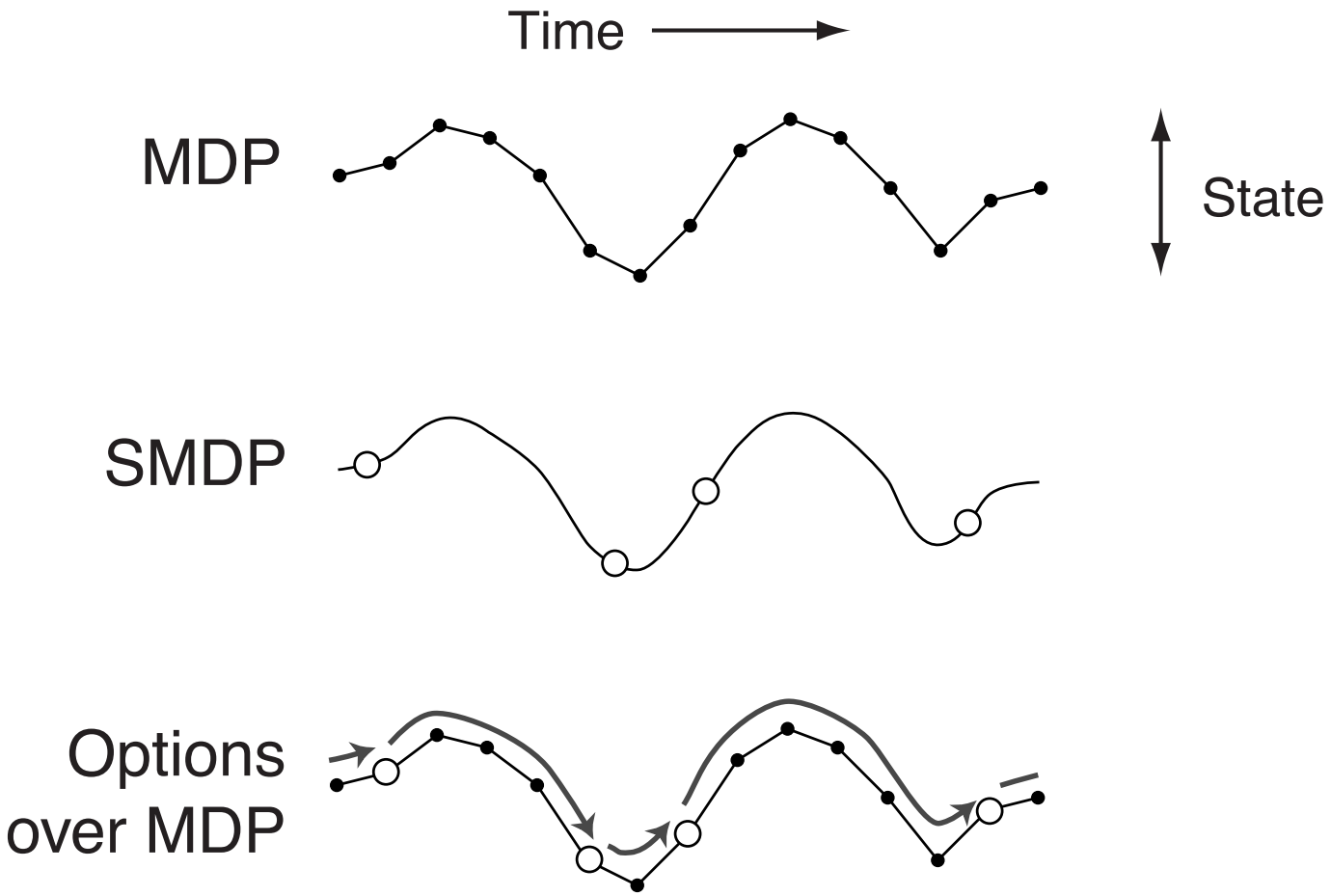
- Focusing on two types of knowledge:
  - *Procedural knowledge*: skills, goal-driven behavior
  - *Predictive, empirical knowledge*: Analogous to the laws of physics, predicting effects of actions
- The knowledge must be:
  - *Expressive*: able to represent many things, including abstractions like objects, space, people, and extended actions
  - *Learnable*: from data without labels or supervision (for scalability)
  - *Composable*: suitable for supporting planning / reasoning by assembling existing pieces

Procedural Knowledge: Options

- An *option* ω consists of 3 components
  - An *initiation set* $I_\omega \subseteq \mathcal{S}$ (aka precondition)
  - A *policy* $\pi_\omega : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$
    $\pi_\omega(a|s)$ is the probability of taking a in s when following option ω
  - A termination condition $\beta_\omega : \mathcal{S} \rightarrow [0, 1]$:
    $\beta_\omega(s)$ is the probability of terminating the option ω upon entering s
- Eg., robot navigation: if there is no obstacle in front (I_ω), go forward (π_ω) until you get too close to another object (β_ω)
- Inspired from macro-actions / behaviors in robotics / hybrid planning and control

Cf. Sutton, Precup & Singh, 1999; Precup, 2000

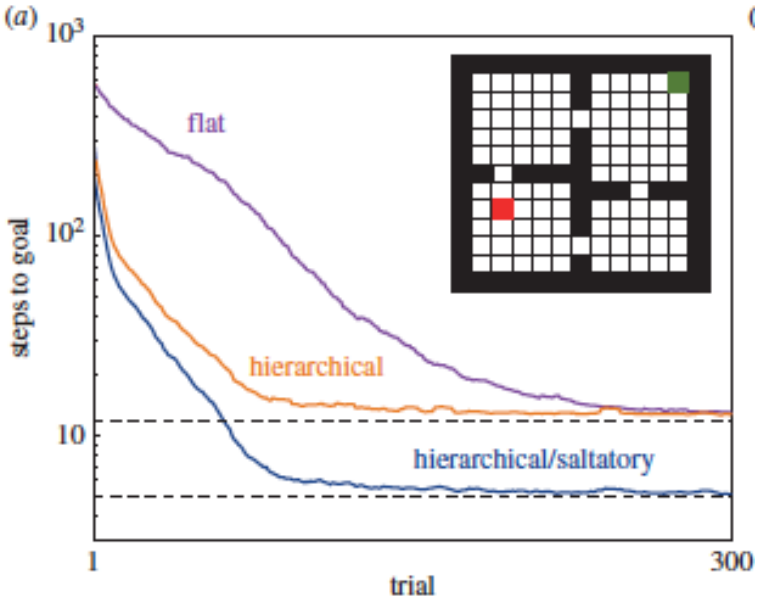
Decision-Making with Options



Learning and planning algorithms are the same at all levels of abstraction!

Option Models Provide Semantics

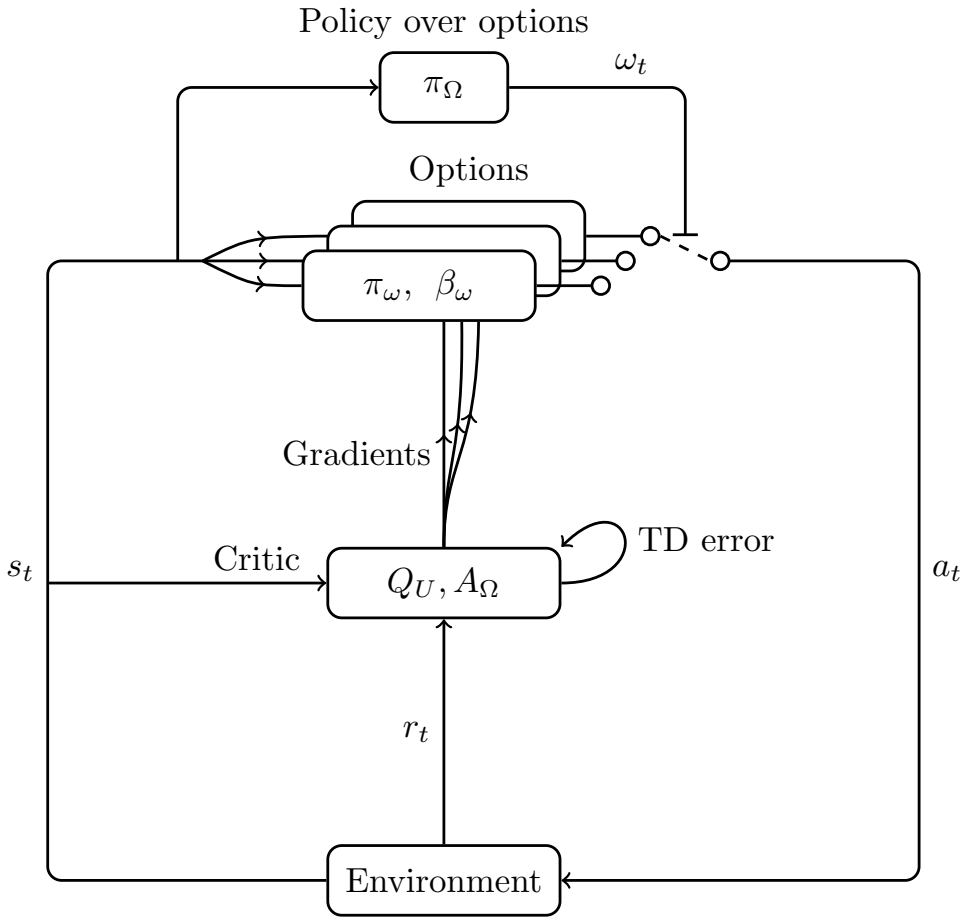
- Models of actions consist of immediate reward and transition probability to next state
- Models of options consist of reward until termination and (discounted) transition to termination state
- Models are *predictions about the future* and provide more *benefits beyond hierarchical behavior* (cf Botvinick & Weinsteein, 2014)



Option-Critic: Learn Options that Optimize Return

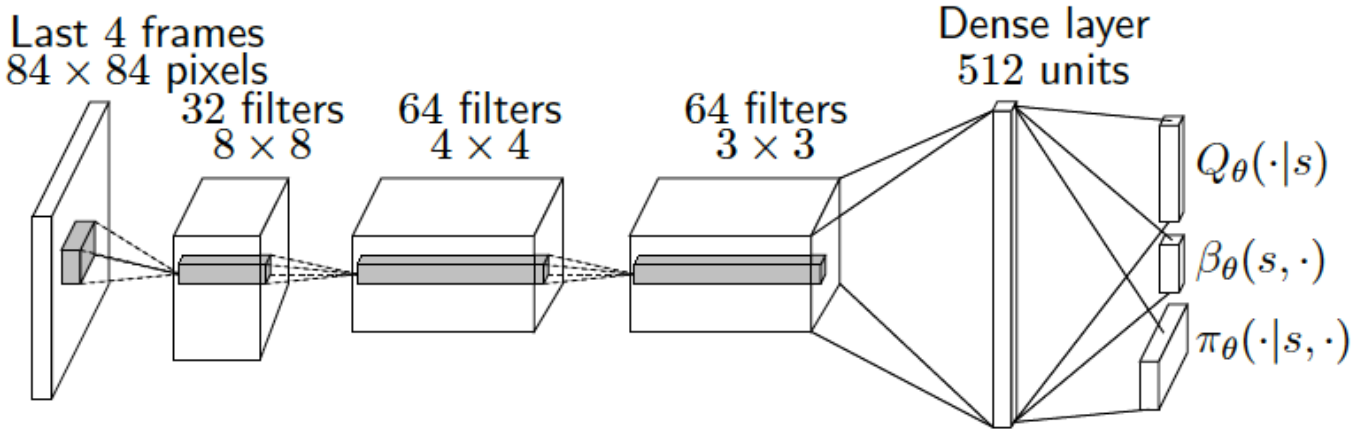
- Explicitly state an *optimization objective* and then solve it to find a set of options
- Handle both *discrete and continuous* set of state and actions
- Learning options should be *continual* (avoid combinatorially-flavored computations)
- Options should provide *improvement within one task* (or at least not cause slow-down...)

Option-Critic Architecture



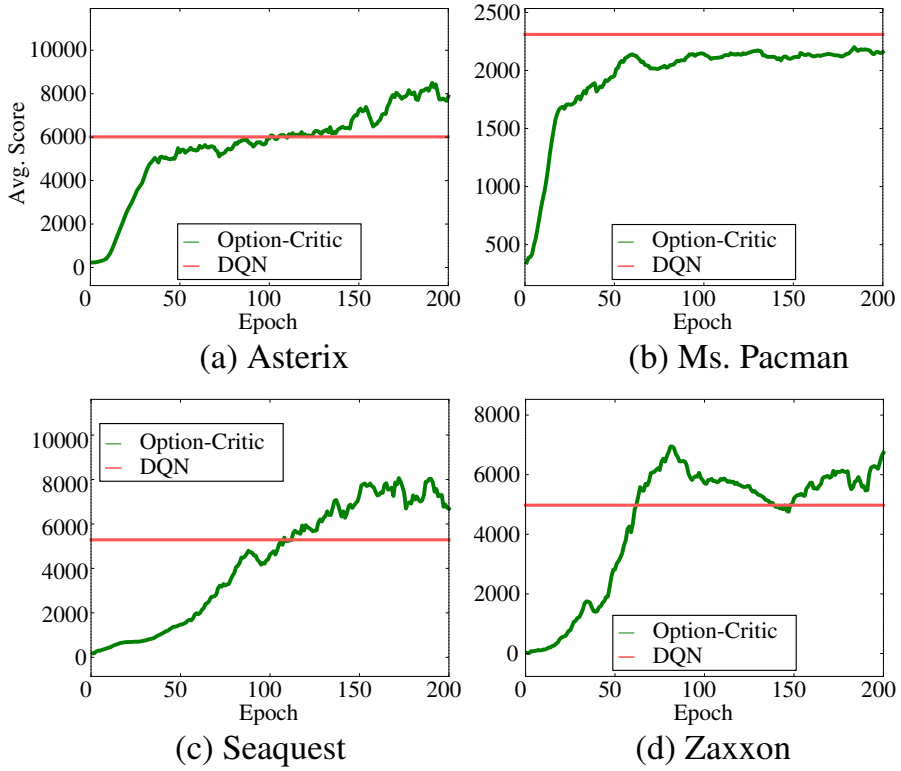
Cf. Bacon, Harb & Precup, 2017; Bacon, 2018

Option-Critic with Deep Learning



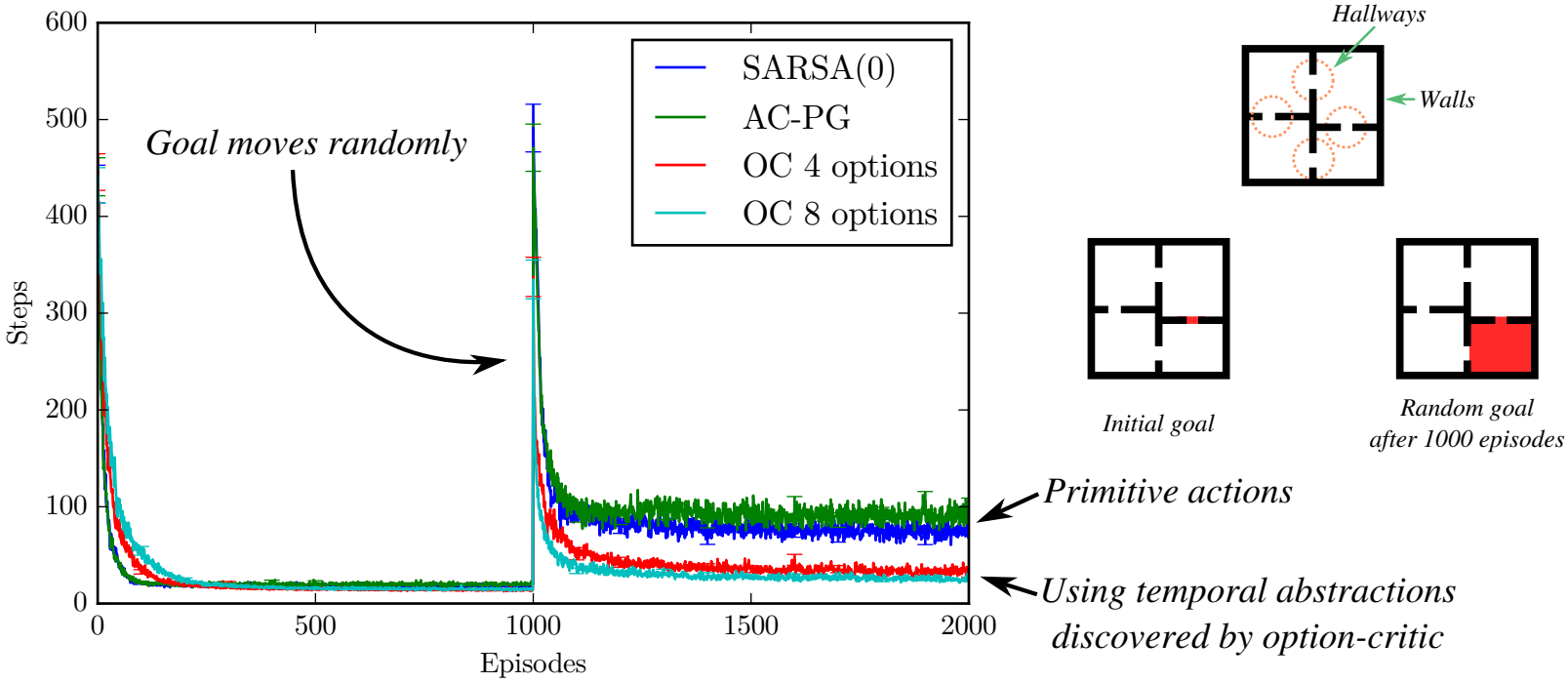
- Intuitive updates:
  - Internal policies change to take better actions
  - Termination becomes more likely if there is a more advantageous option to pick
- Ongoing: same approach for initiation (w K. Khetarpal)

Results: Atari



- Results match or exceed those obtained by state-of-art primitive actions approach, within a single task

Transfer Learning with Option-Critic



Predictive knowledge: Value Function

- Given a policy π, a discount factor γ and a reward function r, the value function of the policy is given by:

$$\begin{aligned} v_\pi(s) &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k)\gamma^{k-t} | S_t = s, A_{t:\infty} \sim \pi] \\ &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k) \prod_{i=t+1}^{k} \gamma | S_t = s, A_{t:\infty} \sim \pi] \end{aligned}$$

- r is the *signal of interest* for the prediction
- γ defines the time scale over which we want to make the prediction (in a very crude way)
- Optimal value function: given a discount factor γ and a reward function r, compute v_{π^*} and π^*, the optimal policy wrt γ, r
-

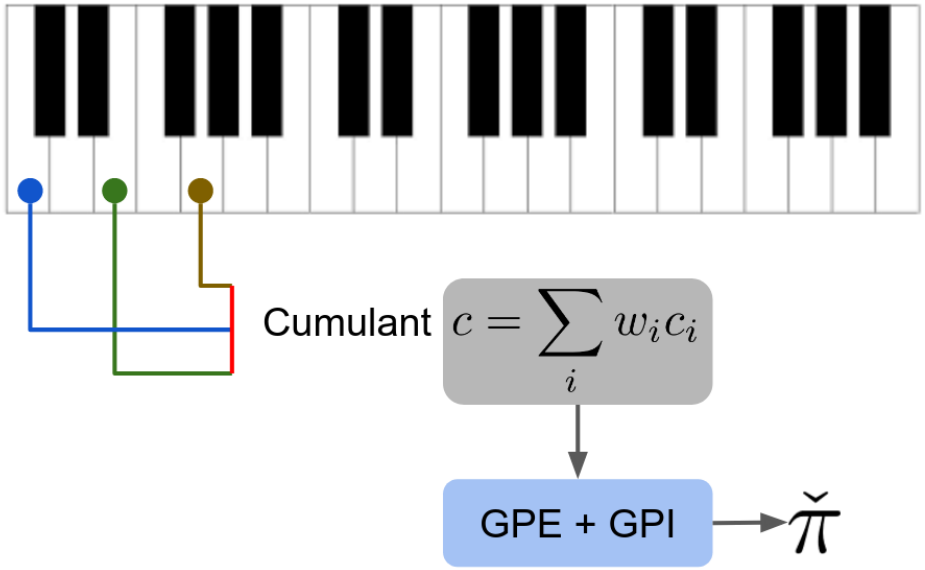
Option models are GVFs

- The reward model for an option ω is defined as:

$$r_\omega(s) = \mathbb{E}_\omega[r(S_t, A_t) + \gamma(1 - \beta_\omega(S_{t+1}))r_\omega(S_{t+1})|S_t = s]$$

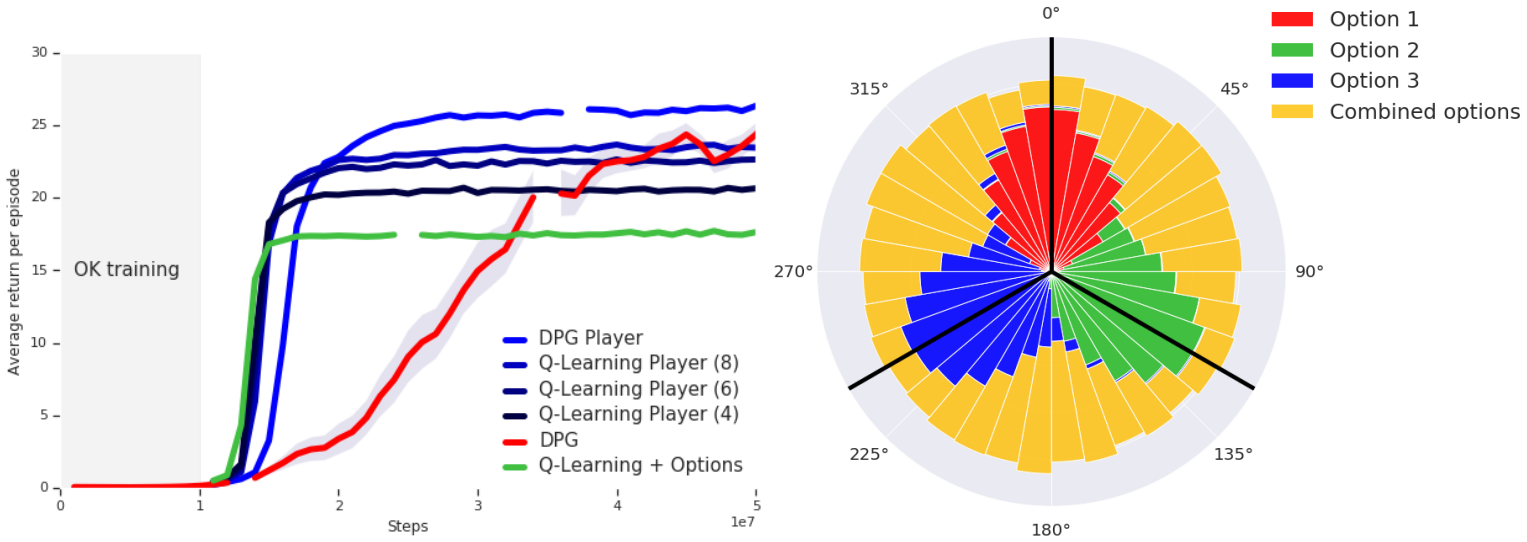
- This means the option reward model is a GVF:
  - policy is π_ω
  - *cumulant* is the environment reward r
  - *continuation function* is $\gamma(1 - \beta_\omega)$
- Option transition model can be similarly written as a GVF

GVFs for synthesizing new behaviors



Option-keyboard - Barreto et al, 2019, based on ideas of Rich Sutton

Option-Keyboard for Moving Target Arena



General way to synthesize quickly new behavior for combinations of reward functions!

Open questions

- Huge gap between theory and practice!
- Is there a natural way to exploit more stable function approximators? Eg kernels, averages…
- Policy or value-based? Depends on application
- Improve stability of deep RL
- Exploration, exploration, exploration….