Reinforcement Learning Workshop

Day 3

Guido Novati

CSElab
Computational Science & Engineering Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

What is (Deep) Supervised Learning (DL)

- DL assumes a dataset and/or a data-generating process.
- DL assumes a parameterized model (neural network).
- Typically the model is designed to have easily-computable analytical gradients.
- Objective is:
  - to **learn** to recognize **features of the data** (a.k.a. representation learning)
  - such that **given a datapoint** as input
  - the model's **output minimizes** a prescribed **loss function**.

Parameterized representations of the data

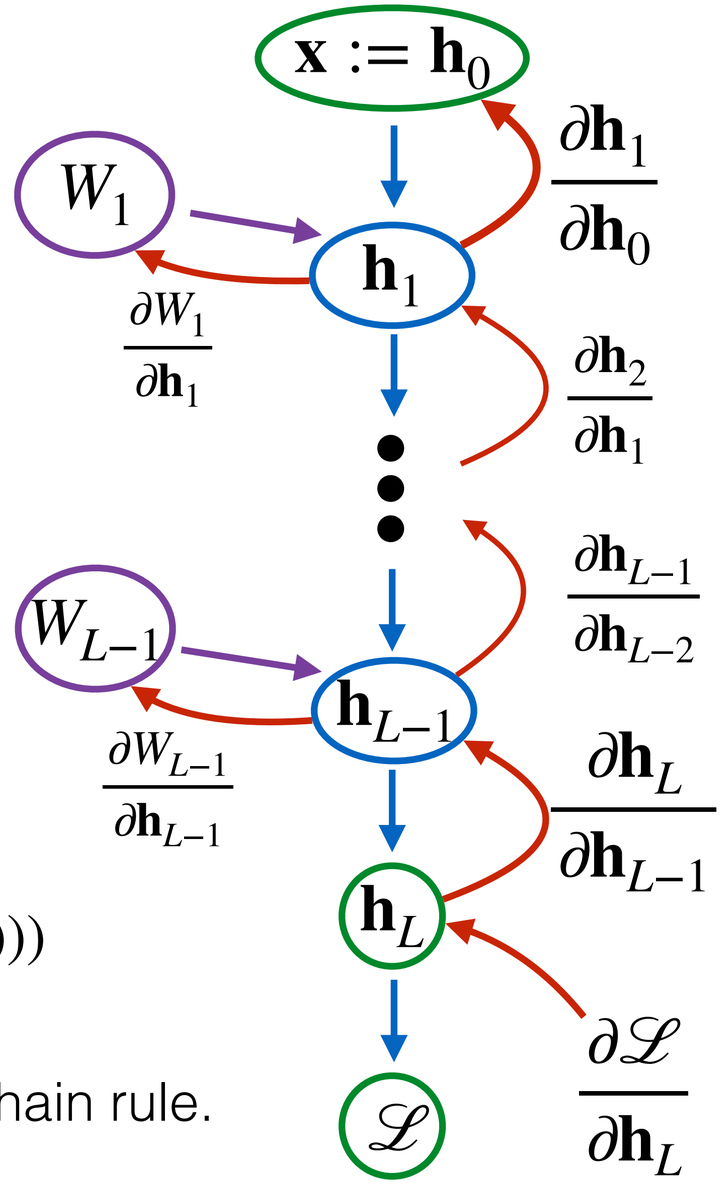
Typically composition of many functions:

1) Linear transformations: $\mathbf{h}_{l+1} = W_{l+1}\,\mathbf{h}_l + \mathbf{b}_{l+1}$

2) Non-linear activation functions: $\mathbf{h}_{l+2} = F_{l+2}\left(\mathbf{h}_{l+1}\right)$

3) Loss function on the output

- Mean squared error $\hat{\mathscr{L}}^{MSE}_{\mathtt{w}}(\mathbf{x}, \mathbf{y}) = \left[\mathbf{y} - F_{\mathtt{w}}(\mathbf{x})\right]^2$
- Log-likelihood $\hat{\mathscr{L}}^{LL}_{\mathtt{w}}(\mathbf{x}, \mathbf{y}) = -\log \mathbb{P}\left[F_{\mathtt{w}}(\mathbf{x}) \mid \mathbf{y}\right]$



Note: $\mathtt{w} = \{W_i,\ \mathbf{b}_i\ \}_{i \in [1,\, L]}$ and $F_{\mathtt{w}}(\mathbf{x}) = F_L(W_{L-1}F_{L-2}(\cdots F_2(W_1\mathbf{x})))$

back-propagation: compute gradient of loss w.r.t parameters by chain rule.

Deep Learning in practice

Objective: find a local minimum of expected loss on distribution of data:

$$\mathtt{w}^* = \arg\min_{\mathtt{w}} \mathscr{L}(\mathtt{w}) = \arg\min_{\mathtt{w}} \mathbb{E}\left[\hat{\mathscr{L}}_{\mathtt{w}}(x, y) \,\middle|\, \begin{matrix} x \sim X \\ y \sim Y \end{matrix}\right]$$

- Estimate loss function through Monte Carlo sampling: $\mathscr{L}(\mathtt{w}) \approx \frac{1}{B}\sum_{i=1}^{B} \hat{\mathscr{L}}_{\mathtt{w}}(x_i, y_i)$
- For each sample, compute gradient of the loss w.r.t. parameters $\frac{\partial \hat{\mathscr{L}}_{\mathtt{w}}(x_i, y_i)}{\partial \mathtt{w}}$
- Update the parameters with stochastic gradient descent: $\Delta\mathtt{w} = -\frac{\epsilon}{B}\sum_{i=1}^{B} \frac{\partial \hat{\mathscr{L}}_{\mathtt{w}}(x_i, y_i)}{\partial \mathtt{w}}$

Intuition: linear "Neural Network"

One linear layer from input to output: $h_{\mathtt{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{i=1} x_i w_i + b$

Mean squared error: $\mathscr{L}^{MSE}_{\mathtt{w}}(\mathbf{x}, y) = \frac{1}{2}\left[y - h_{\mathtt{w}}(\mathbf{x})\right]^2$

Back-propagation: $\frac{\partial h_{\mathtt{w}}(\mathbf{x})}{\partial w_i} = x_i$

$\frac{\partial \mathscr{L}^{MSE}_{\mathtt{w}}(\mathbf{x}, y)}{\partial h_{\mathtt{w}}} = (h_{\mathtt{w}}(\mathbf{x}) - y)$

single datapoint SGD: $\Delta \mathbf{w} = -\epsilon \frac{\partial \mathscr{L}^{MSE}_{\mathtt{w}}(\mathbf{x}, y)}{\partial \mathbf{w}} = -\epsilon \frac{\partial \mathscr{L}^{MSE}_{\mathtt{w}}(\mathbf{x}, y)}{\partial h_{\mathtt{w}}} \frac{\partial h_{\mathtt{w}}(\mathbf{x})}{\partial \mathbf{w}} = -\epsilon \left[h_{\mathtt{w}}(\mathbf{x}) - y\right] \mathbf{x}$

in general: SGD: $\mathtt{w}^{(k+1)} = \mathtt{w}^{(k)} - \epsilon \nabla_{\mathtt{w}} \mathop{\mathbb{E}}_{X,Y} \left[\hat{\mathscr{L}}^{MSE}_{\mathtt{w}}(\mathbf{x}, \mathbf{y})\right]$

Chain rule: $\mathtt{w}^{(k+1)} = \mathtt{w}^{(k)} - \epsilon \mathop{\mathbb{E}}_{X,Y} \left[\nabla_{F_{\mathtt{w}}} \hat{\mathscr{L}}^{MSE}_{\mathtt{w}}(\mathbf{x}, \mathbf{y}) \cdot \nabla_{\mathtt{w}} F_{\mathtt{w}}(\mathbf{x})\right]$

$\mathtt{w}^{(k+1)} = \mathtt{w}^{(k)} + \epsilon \mathop{\mathbb{E}}_{X,Y} \left[\left(\mathbf{y} - F_w(\mathbf{x})\right) \cdot \nabla_w F_w(\mathbf{x})\right]$

move network output towards target y

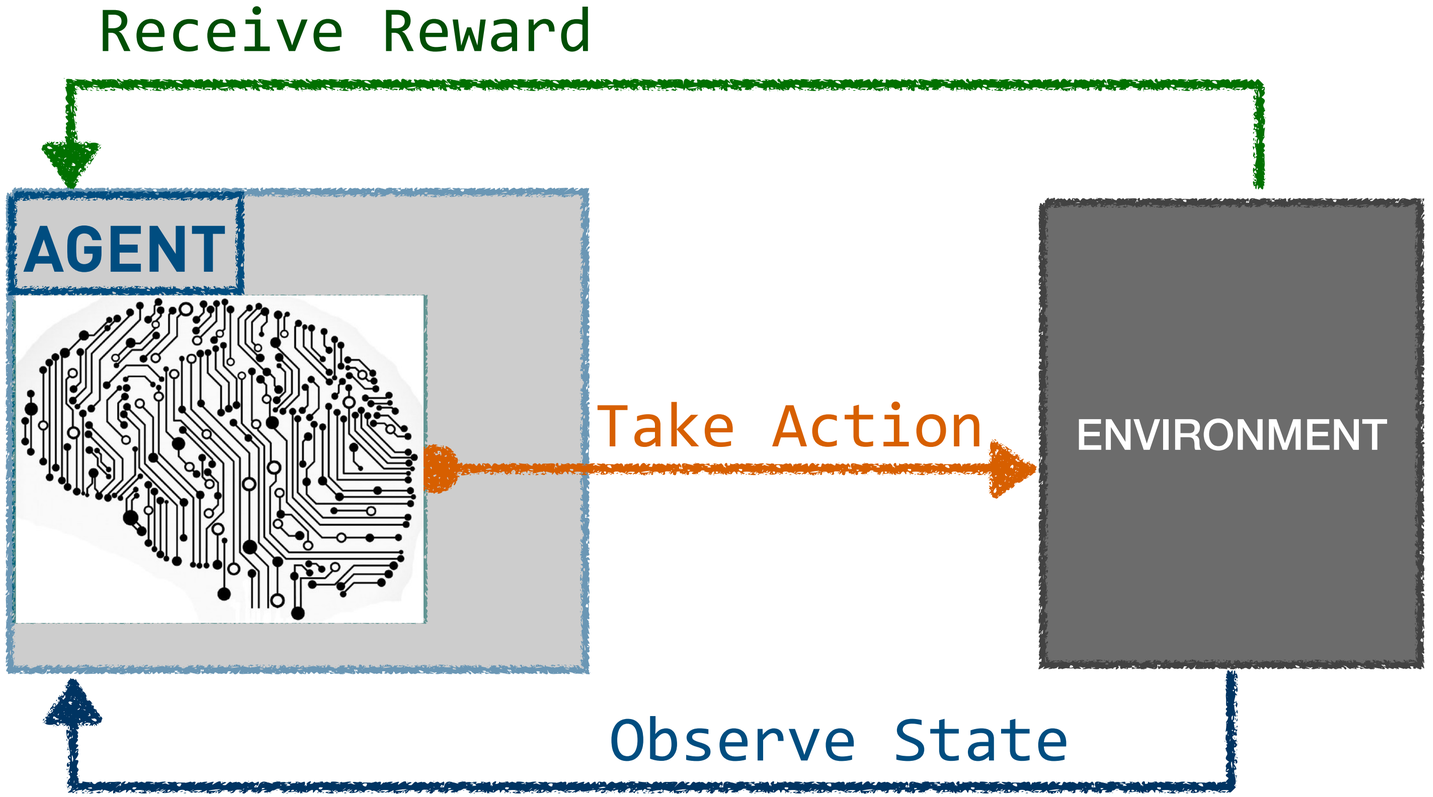
Today's Plan

- What is Deep Supervised Learning and what is Reinforcement Learning
- Introduction to fundamental RL assumptions & tradeoffs
- Core RL concepts:
    - Policy, Value functions, approximate models
- RL via approximating value functions:
    - Monte Carlo, SARSA, Q-learning
- Hands-on RL with pytorch

Reinforcement Learning's loop

Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain.

Alan Turing,
Computing Machinery and Intelligence



What is Reinforcement Learning

- Mathematical framework to **search optimal strategies** for sequential decision processes.
- RL describes one (or more) **agent** with the capacity to observe its **environment** and act.
- Action influences the state of the env and/or agent's ability to perform future actions.
- Success is measured by total sum of rewards obtained by the agent.

- Reward hypothesis: "Any goal can be described by maximization of expected rewards"

The challenge of RL

- **Exploration/exploitation dilemma**
    - Exploitation : Make the best decision given current information
    - Exploration : Gather more information
    - The best long-term strategy may involve short-term sacrifices.
    - When have we gathered enough information to make the best overall decisions?
- **Temporal credit assignment**
    - Temporal structure of the optimization problem
    - Goodness of action measured over future consequences
    - Which sequence of actions caused the high/low rewards?
- **Representation learning**
    - Deep learning challenge : state may be sensors or sight (CNN).
    - Optimal strategy may require remembering facts from the past (RNN).
    - Learn to approximate probability distributions over arbitrarily large state and action spaces.

What are states / actions / rewards?

The **state** $s_t \in \mathcal{S}$ can be either:

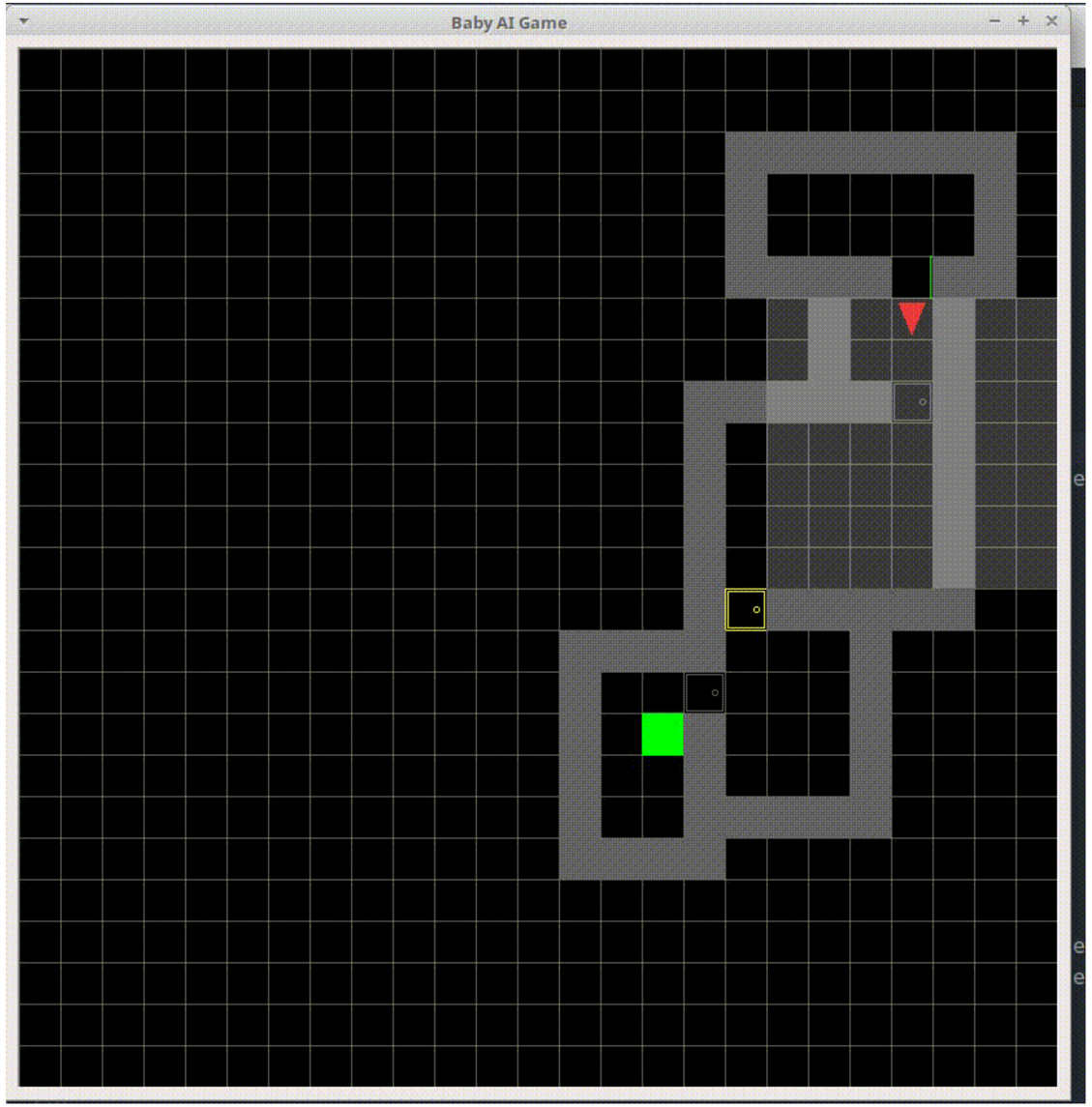
- **discrete** $\mathcal{S} = \{s^{(0)}, s^{(1)}, s^{(2)}, \cdots, s^{(N_S)}\}$ i.e. corresponds to one out of N_S alternatives
- **continuous** $\mathcal{S} = \mathbb{R}^{dim_S}$ i.e. is a vector in a dim_S space

The **action** $a_t \in \mathcal{A}$ can be either:

- **discrete** $\mathcal{A} = \{a^{(0)}, a^{(1)}, a^{(2)}, \cdots, a^{(N_A)}\}$ (policy may be a Bernoulli distribution)
- **continuous** $\mathcal{A} = \mathbb{R}^{dim_A}$ (policy may be a Gaussian distribution)

The reward $r_t \in \mathbb{R}$ is a scalar.

RL with discrete actions and states



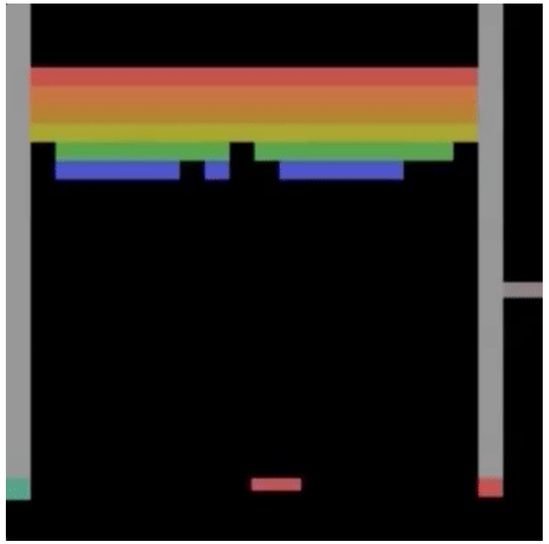
- First applications of RL: grid-world scenarios
- Often simplified by **take Action == Go to State**
- Often amenable to dynamic programming

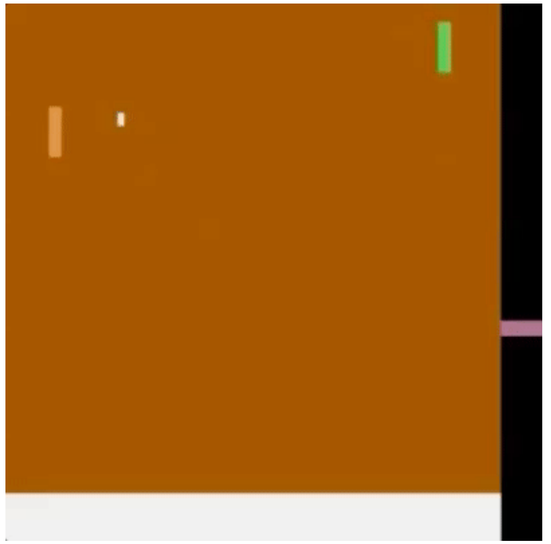
- RL + deep learning allow mitigating curse of dimensionality and recent breakthroughs in classic games with large state spaces:

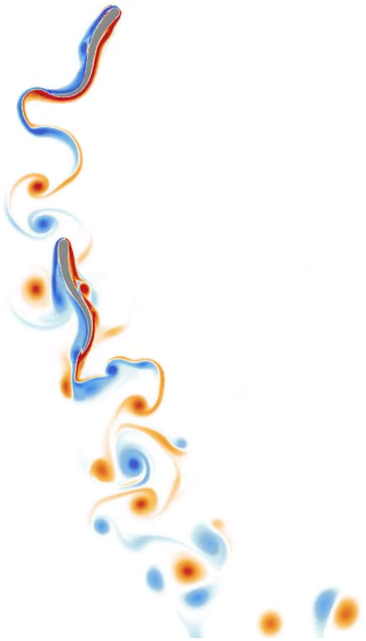
Silver et al. 2016

RL with discrete actions and continuous states

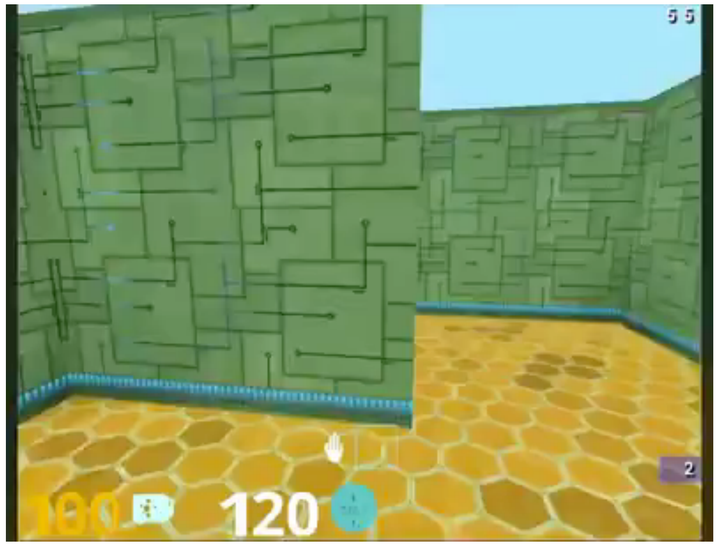
Mnih et al. 2015





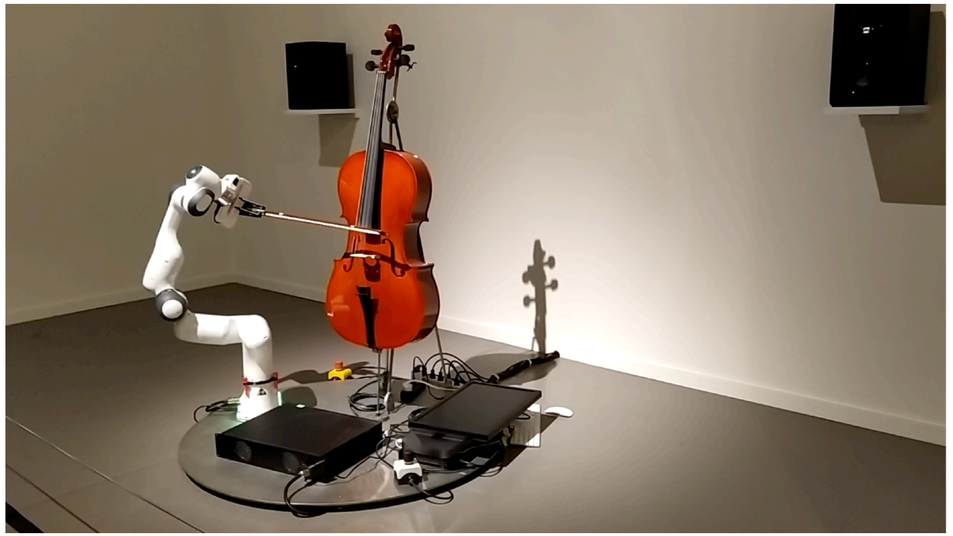


Jaderberg et al. 2016

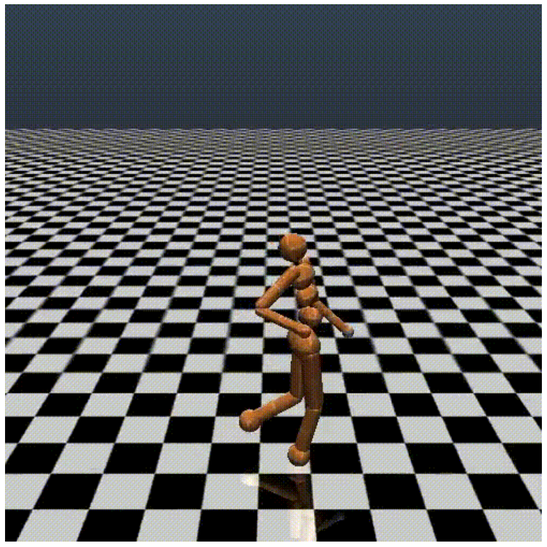


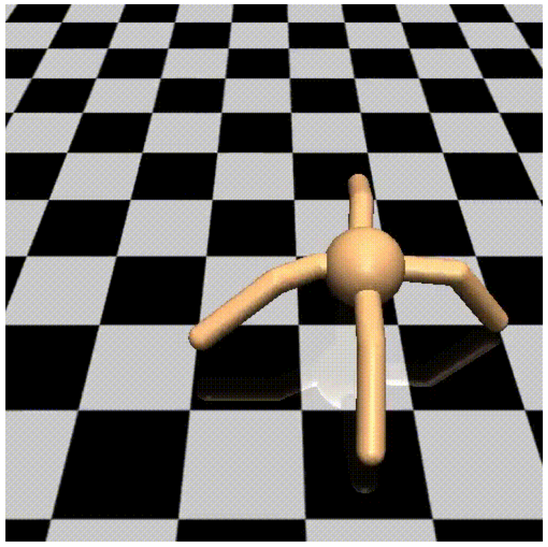
RL with continuous actions and continuous states

Abbeel et al. 2006



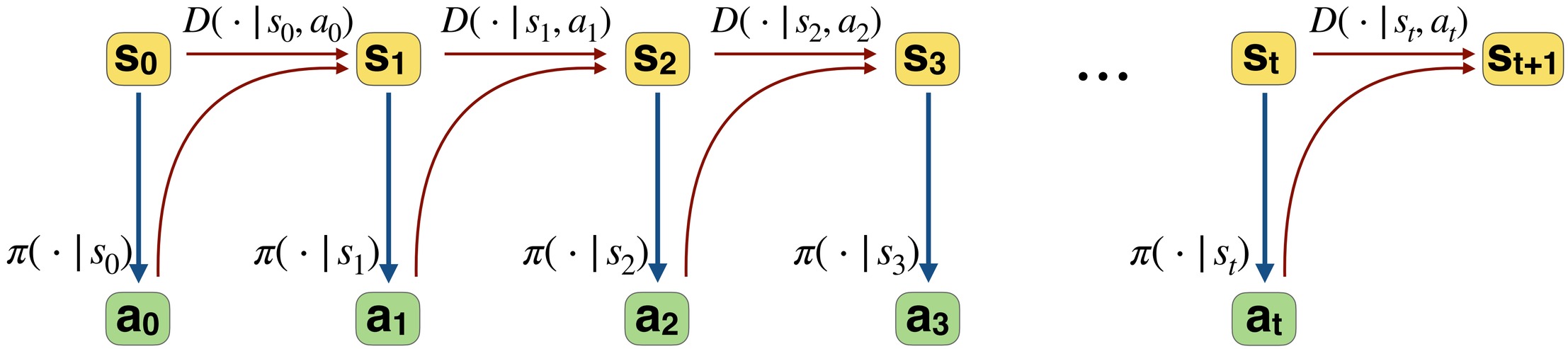
Schulman 2015





- Most PDE / ODE / robotic systems / real world applications

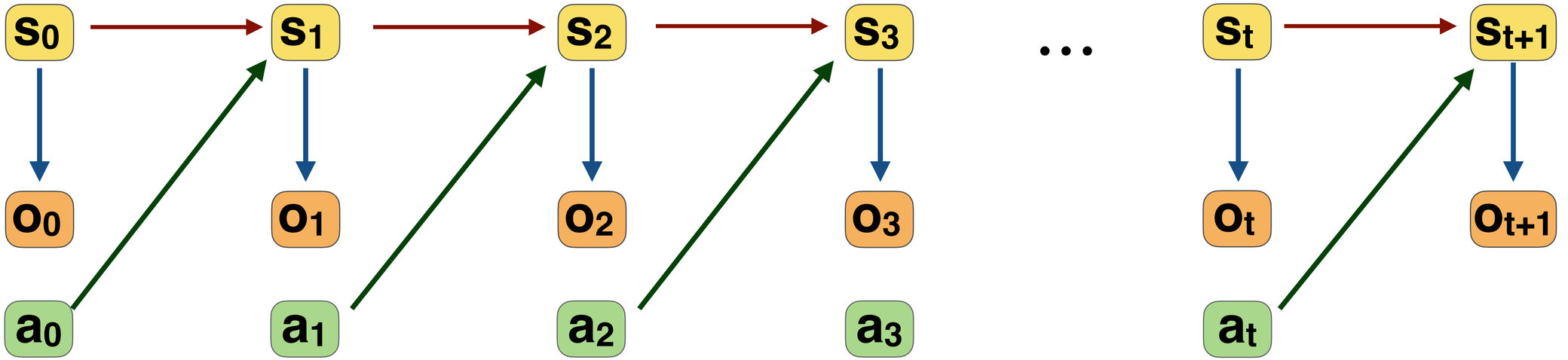
Markov decision process (MDP)



- Probability of sampling an action a_t depends only on state s_t (may be deterministic)
- Probability of transitioning to state s_{t+1} depends on both s_t and a_t (may be deterministic)
- Reward r_{t+1} is in general function of $(s_t,\ a_t,\ s_{t+1})$

- "The future is independent of the past given the present" : $\mathbb{P}(s_{t+1} \mid s_t,\ s_{t-1},\ \cdots,\ s_0) = \mathbb{P}(s_{t+1} \mid s_t)$
- In order to pick optimal action, agent only needs to know current state.
- This is said to be a *fully observable* problem: state fully describes environment.

Partially observable Markov decision process (POMDP)



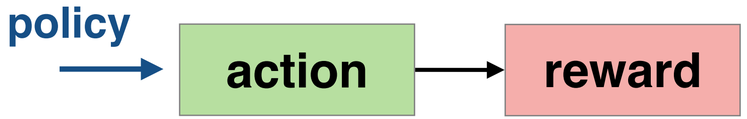
- Agent receives only partial information about the state of the environment : **observation**
- Probability of current state can only be inferred by looking at all "sensimotor history":

$$\mathbb{P}(s_t) = \mathbb{F}(o_0,\, a_0,\, o_1,\, a_1,\, \cdots,\, o_t)$$

- Policy $\pi(\,\cdot\,|\,o_t)$ is probably sub-optimal : does not include enough information to estimate the state
- Optimal policy is function of entire history : $a_t \sim \pi(\,\cdot\,|\,s_t) = \pi(\,\cdot\,|\,o_0,\, a_0,\, o_1,\, a_1,\, \cdots,\, o_t)$
- Recurrent Neural Networks (RNN) may be used to encode time history in the policy

Variants of the RL problem

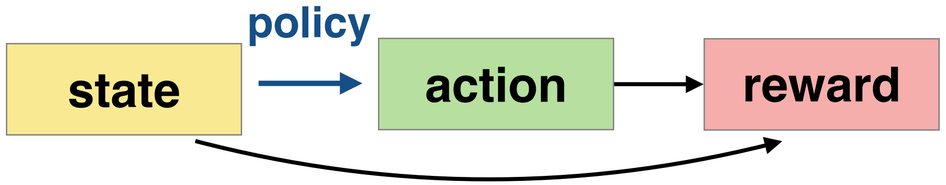
- Multi-armed bandit problem



Imagine a gambler at a row of slot machines, who has to decide which machines to play, how many times to play each machine and in which order to play them.

- RL without neither states nor dynamics.
- Purely exploration/exploitation dilemma.

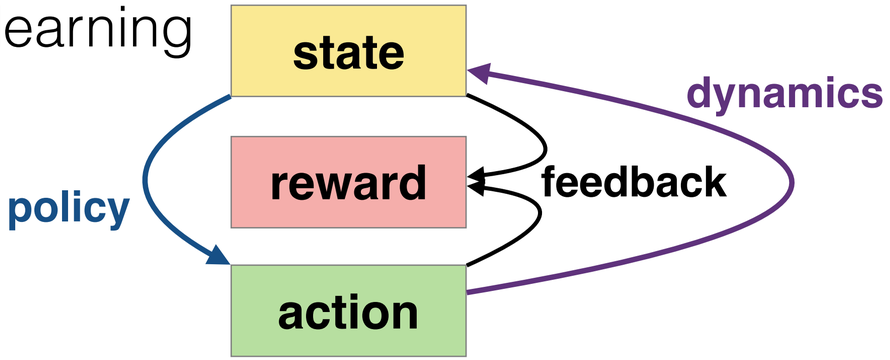
- Contextual bandits problem



Imagine that slot machines have clues (e.g. brand).

- RL without dynamics.
- Exploration/exploitation dilemma
- Feature learning challenge

- Reinforcement learning

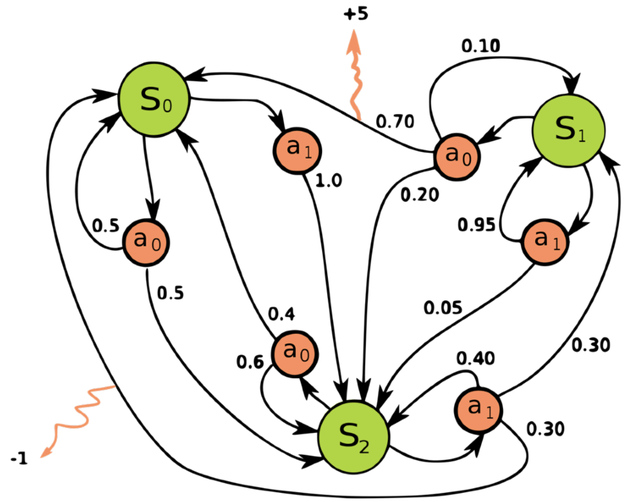


- Environment dynamics.
- Exploration/exploitation dilemma
- Feature learning challenge
- Temporal credit assignment problem

Differences from neighboring fields

Dynamic programming (DP)

- Finite-dimensional problems (i.e. discrete)
- Available model of the environment:
  - $D(s_{t+1} \mid s_t, a_t)$ is a known transition matrix.
  - "*for each state do …*"
- e.g. shortest path / resource allocation



Optimal control (OC)

- Sets of ODEs describing the state of the system and a cost functional to minimize.
- May solve Hamilton–Jacobi–Bellman (HJB) equations
- e.g. linear quadratic controller (linear ODE with quadratic actuation cost)

$$\begin{cases} \dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \\ J = \mathbf{x}^T(T)F\mathbf{x}(T) + \int_0^T \left[\mathbf{x}^T(t)Q\mathbf{x}(t) + \mathbf{u}^T(t)R\mathbf{u}(t)\right] \mathrm{d}t \end{cases}$$

optimal actuation has form:
$\mathbf{u}(t) = K\mathbf{x}(t)$

RL is characterized by **trial-and-error interaction** with an unknown (**black-box**) environment.

RL: Markov decision process with parameterized policy

RL learns a policy

- Function that computes actions given states.
- **Improvable** i.e. it has parameters `w` which can be optimized.
- Some stochasticity is required to explore dynamics and possible improvements.

$$s_0 \sim D_0(\cdot)$$

Environment at random initial condition

$$a_0 \sim \pi_{\mathtt{w}}(\cdot \mid s_0)$$

Agent computes policy given state

$$s_1, r_1 \sim D(\cdot \mid s_0, a_0)$$

Environment advances in time and agent receives reward

$$\dots$$

$$a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_t)$$

$$s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t)$$

$$\dots$$

$$J(w) = \mathbb{E}\left[\sum_t^{\infty} r_t \;\middle|\; \begin{array}{c} a_t \sim \pi_w(\cdot \mid s_t) \\ s_{t+1} \sim D(\cdot \mid a_t, s_t) \end{array} \right]$$

optimal parameters

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} J(\mathbf{w})$$

Value functions: estimate of future rewards given a policy

- Q-function "What are expected reward by starting in (s,a) and following policy?"

$$Q^{\pi_w}(s,a) = \mathbb{E}\left[r_1 + r_2 + r_3 + \cdots \,\middle|\, \begin{matrix} s_0 = s & a_t \sim \pi_w(\cdot \mid s_t), \\ a_0 = a & s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{matrix} \right]$$

- This estimator is **unattainable!**
- In RL we may never "set s_0 to s and set a_0 to a" and follow the policy.
- In fact we don't know if we can ever reproduce/see again a given s.
- We can only keep interacting with the environment.

Value functions: estimate of future rewards given a policy

$$Q^{\pi_w}(s,a) = \mathbb{E}\left[r_1 + r_2 + r_3 + \cdots \,\middle|\, \begin{matrix} s_0 = s & a_t \sim \pi_w(\cdot \mid s_t), \\ a_0 = a & s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{matrix} \right]$$

- May include discount $\gamma \in [0,\ 1]$ to bound expectation:

$$Q^{\pi_w}(s,a) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots \,\middle|\, \begin{matrix} s_0 = s & a_t \sim \pi_w(\cdot \mid s_t), \\ a_0 = a & s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{matrix} \right]$$

- γ affects optimal policy!
  - γ close to 0: optimization is **shortsighted**, but **easier** and more **robust** to uncertainty.
  - γ close to 1: **exact**, non-relaxed, evaluation, but may be unbounded ($Q \to \infty$).

- Decomposes into Bellman Equation:

$$Q^{\pi_w}(s,a) = \mathbb{E}\left[r + \gamma Q^{\pi_w}(s',a') \,\middle|\, \begin{matrix} s',\ r \sim D(\cdot \mid s,a) \\ a' \sim \pi_w(\cdot \mid s') \end{matrix} \right]$$

Value functions: estimate of future rewards given a policy

- (γ-)Optimal policy maximizes value function everywhere:

$$\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a)$$

- Similarly, optimal Q-function is maximal for each (s,a) pair:

$$Q^*(s, a) = \max_\pi Q^\pi(s, a) = Q^{\pi^*}(s, a) \quad \text{for each (s,a)}$$

- Bellman Equation for optimal Q-function:

$$Q^{\pi^*}(s, a) = \mathbb{E}\left[r + \gamma Q^{\pi^*}(s', a') \,\middle|\, \begin{matrix} s',\ r \sim D(\cdot \mid s, a) \\ a' \sim \pi^*(\cdot \mid s') \end{matrix} \right]$$

$$Q^{\pi^*}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^{\pi^*}(s', a') \,\middle|\, s',\ r \sim D(\cdot \mid s, a) \right]$$

Three value functions

- **Q-function**: "What are expected rewards by starting in (s,a) and following policy?"

$$Q^{\pi_w}(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \;\middle|\; \begin{matrix} s_0 = s, & a_t \sim \pi_w(\cdot \mid s_t), \\ a_0 = a, & s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{matrix}\right]$$

- **State value**: "What are expected rewards by starting in s and following policy?"

$$V^{\pi_w}(s) = \mathbb{E}\left[Q^{\pi_w}(s,a) \;\middle|\; a \sim \pi_w(\cdot \mid s)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \;\middle|\; \begin{matrix} s_0 = s, & a_t \sim \pi_w(\cdot \mid s_t), \\ a_0 = \pi_w(\cdot \mid s_0), & s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{matrix}\right]$$

- **Action advantage**: "How much better is it to do action **a** compared to expected rewards from π?"

$$A^{\pi_w}(s,a) = Q^{\pi_w}(s,a) - V^{\pi_w}(s)$$

- Therefore:

$$\mathbb{E}\left[A^{\pi_w}(s,a) \;\middle|\; a = \pi_w(\cdot \mid s)\right] = 0$$

Episodes: an extension to the classic RL loop

Instead of infinite interaction loop (continuing tasks) much of RL analysis assumes that loop is split into episodes (episodic tasks).

classic RL loop

$$s_0 \sim D_0(\cdot)$$

$$a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_0)$$

$$s_1,\, r_1 \sim D(\cdot \mid s_0,\, a_0)$$

$$\dots$$

$$a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_t)$$

$$s_{t+1},\, r_{t+1} \sim D(\cdot \mid s_t,\, a_t)$$

$$\dots$$

$$\infty$$

episode-based loop

$$\infty$$

$$\dots$$

$$a_{T-1}^{(k)} \sim \pi_{\mathtt{w}}(\cdot \mid s_{T-1}^{(k)})$$

$$s_T^{(k)},\, r_T^{(k)} \sim D(\cdot \mid s_{T-1}^{(k)},\, a_{T-1}^{(k)})$$

$$s_0^{(k+1)} \sim D_0(\cdot)$$

$$a_0^{(k+1)} \sim \pi_{\mathtt{w}}(\cdot \mid s_0^{(k+1)})$$

$$\dots$$

$$\infty$$

- Episode may last $T^{(k)}$ or T step. (i.e. number of steps per episode may vary)
- After s_T no actions, sample initial conditions.
- Agent may reach **terminal state** (& terminal reward) due to to success/failure in task or to timeout.
- Or reach a **cutoff state**. This is a convent and arbitrary time horizon ("test policy for at most T steps").
- Episodic tasks allow $\gamma = 1$!

State visitation frequency

Consider exploring the MDP with given π_w:

for $k = 1, 2, \cdots, \infty$:

$$
\begin{aligned}
s_0^{(k)} &\sim D_0(\,\cdot\,) \\
a_0^{(k)} &\sim \pi_w(\,\cdot \mid s_0^{(k)}) \\
s_1^{(k)},\, r_1^{(k)} &\sim D(\,\cdot \mid s_0^{(k)},\, a_0^{(k)}) \\
&\cdots \\
a_{T-1}^{(k)} &\sim \pi_w(\,\cdot \mid s_{T-1}^{(k)}) \\
s_T^{(k)},\, r_T^{(k)} &\sim D(\,\cdot \mid s_{T-1}^{(k)},\, a_{T-1}^{(k)})
\end{aligned}
$$

What is the probability of a given state s?

$$
s \;\sim\; \eta^{\pi_w}(\,\cdot\,) \;\propto\; \sum_{k=0}^{\infty} \mathbb{P}\left[s = s_k \;\middle|\; \begin{array}{c} s_0 \sim D_0(\,\cdot\,) \\ a_k \sim \pi_w(\,\cdot \mid s_k) \\ s_{k+1} \sim D(\,\cdot \mid s_k, a_k) \end{array} \right]
$$

- Even if D and/or π_w are deterministic
- As long as at least one between D_0 or π_w are stochastic
- MDP defines a random process that outputs states

Therefore we could write, for example:

$$
J(w) = \mathbb{E}\left[Q^{\pi_w}(s,a) \;\middle|\; \begin{array}{c} s \sim \eta^{\pi_w}(\,\cdot\,) \\ a \sim \pi_w(\,\cdot \mid s) \end{array} \right]
$$

prob of a state given the policy

prob of action given the state

"maximize expected cumulative rewards for any (s,a) reached by the policy"

Components of a RL algorithm

- Common building blocks of RL:

deep learning difficulty

- **Policy**: a function to compute actions given a state → Learn by directly optimizing **policy** parameters (e.g. Schulman et al., 2015; Silver et al. 2016).
- **Value function**: a function to estimate rewards obtained by a policy → Learn the **value** function and act to maximize estimated rewards (e.g. Tesauro, 1992; Mnih et al. 2015).
- **Environment model**: a function to estimate environment dynamics. → Learn a **model** and use it to **plan** optimal actions (e.g. Ha & Schmidhuber, 2018; Schrittwieser et al. 2019).

Why so many RL algorithms?!

- Tradeoffs:
  - **Sample efficiency** v.s. **simplicity/ease of use**
  - **Stability** v.s. **local optima**
  - **Stochastic** v.s. **deterministic**
- Assumptions:
  - **Continuous** v.s. **discrete**
  - **Low dimensional** v.s. **high dimensional**
  - **Fully-observable** v.s. **partially observable**

E.g. **off-policy** vs **on-policy** methods
on-policy: after each update old data is useless
off-policy: re-use data over many iterations

E.g. learn a **policy** vs learn the **value** function
policy: stable, but maybe stuck in local optima
value: with NN is numerically unstable

deterministic (policy/update) algorithms must introduce randomness somewhere to explore
stochastic (policy/update) can naturally learn to explore, but may be less sample-efficient

E.g. **tabular** v.s. **NN** approximators
tabular: more convergence guarantees
NN: reduce curse of dimensionality, generality

Approximating value functions

- **Goal:** learn $Q^{\pi}(s,a) = \mathbb{E}\left[\hat{q}_t^{\pi} \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\cdot) \\ a \sim \pi_w(\cdot \mid s) \end{matrix}\right] \quad \begin{matrix} \forall s \in \mathcal{S} \\ \forall a \in \mathcal{A} \end{matrix}$
- Where $\hat{q}_t^{\pi} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1}$
- With function approximator : $Q_{\mathtt{w}}(s,a) \approx Q^{\pi}(s,a) \quad \begin{matrix} \forall s \in \mathcal{S} \\ \forall a \in \mathcal{A} \end{matrix}$
- First assume discrete action space

  But **Q-learning is unstable if applied to NN**:
  - Updates from temporally correlated data ($s_t,\ s_{t+1},\ s_{t+2}, \cdots$),
  - $\max$ operation leads $Q_{\mathtt{w}}$ to explode.

1 sample estimate of Q:

$$\hat{q}_t^{\pi} = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1} \,\middle|\, \begin{matrix} a_t \sim \pi, \\ s_{t+1}, r_{t+1} \sim D \end{matrix}$$

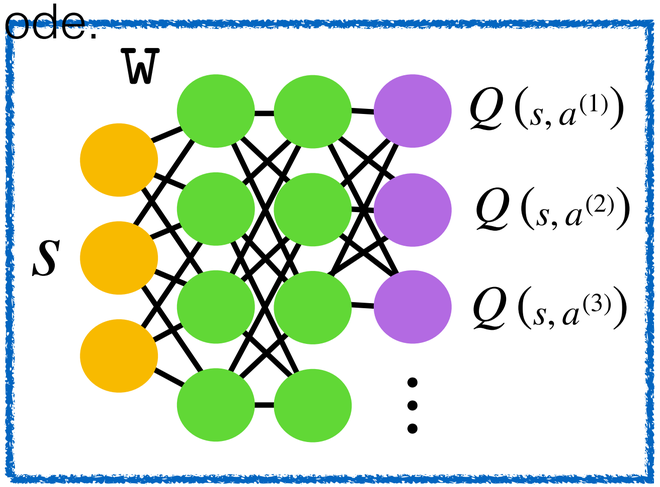
analytical estimate of Q:

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\pi}\left[\hat{q}^{\pi}(s,a)\right]$$

parametric estimate of Q:

$$Q_{\mathtt{w}}(s,a) \approx Q^{\pi}(s,a)$$

	$\mathbf{a^{(1)}}$	$\mathbf{a^{(2)}}$	...	$\mathbf{a^{(N_A)}}$
$\mathbf{s^{(1)}}$	$Q\left(s^{(1)}, a^{(1)}\right)$	$Q\left(s^{(1)}, a^{(2)}\right)$	...	$Q\left(s^{(1)}, a^{(N_A)}\right)$
$\mathbf{s^{(2)}}$	$Q\left(s^{(2)}, a^{(1)}\right)$	$Q\left(s^{(2)}, a^{(2)}\right)$		$Q\left(s^{(2)}, a^{(N_A)}\right)$
...	...	...	...	...
$\mathbf{s^{(N_S)}}$	$Q\left(s^{(N_S)}, a^{(1)}\right)$	$Q\left(s^{(N_S)}, a^{(2)}\right)$	...	$Q\left(s^{(N_S)}, a^{(N_A)}\right)$



Estimating value functions

- **Goal:** learn approximate $Q^{\pi}_{\mathtt{w}}(s,a) \approx Q^{\pi}(s,a) \quad \begin{matrix} \forall s \in \mathcal{S} \\ \forall a \in \mathcal{A} \end{matrix}$
- **But:** we cannot **ever** compute $Q^{\pi}(s,a)$ because we do not have a model that we can reset to (s,a).
- How can we estimate $Q^{\pi}(s,a)$ with finite data?

- **Monte Carlo Learning**
  - Learn from full episodes/sequences of actions
  - Update the value towards **empirical mean returns**

- **Temporal Difference Learning**
  - Update the value towards **consistency with next estimated value**
  - Estimates based in part on other estimate: bootstrapping

Monte Carlo Learning

Given a policy π

For i = 0, 1, 2 … perform "**roll-outs**":

$$
\begin{aligned}
s_0^{(i)} &\sim D_0(\,\cdot\,) \\
a_0^{(i)} &\sim \pi(\,\cdot \mid s_0^{(i)}) \\
s_1^{(i)},\, r_1^{(i)} &\sim D(\,\cdot \mid s_0^{(i)},\, a_0^{(i)}) \\
&\cdots \\
a_{T-1}^{(i)} &\sim \pi(\,\cdot \mid s_{T-1}^{(i)}) \\
s_T^{(i)},\, r_T^{(i)} &\sim D(\,\cdot \mid s_{T-1}^{(i)},\, a_{T-1}^{(i)})
\end{aligned}
$$

Example: discrete states & actions

- Each pair (s, a) corresponds to one entry in a multi-dimensional grid
- We can update $Q_{\mathtt{w}}(s_t^{(i)}, a_t^{(i)})$ independently from other entries
- $\dfrac{\mathrm{d}\mathscr{L}(\mathtt{w})_t^{(i)}}{\mathrm{d}\mathtt{w}} = Q_{\mathtt{w}}(s_t^{(i)}, a_t^{(i)}) - \hat{q}_t^{\pi\,(i)}$
- Update with SGD:
  $Q_{\mathtt{w}}(s_t^{(i)}, a_t^{(i)}) \leftarrow Q_{\mathtt{w}}(s_t^{(i)}, a_t^{(i)}) + \epsilon \left[\hat{q}_t^{\pi\,(i)} - Q_{\mathtt{w}}(s_t^{(i)}, a_t^{(i)})\right]$

For each step, compute: $\hat{q}_t^{\pi\,(i)} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1}^{(i)}$ **1 sample estimate of** $Q^{\pi}(s_t^{(i)}, a_t^{(i)})$

Minimize approximation error:

$$\mathscr{L}(\mathtt{w})_t^{(i)} = \frac{1}{2}\left[\hat{q}_t^{\pi\,(i)} - Q_{\mathtt{w}}^{\pi}(s_t^{(i)}, a_t^{(i)})\right]^2$$

Only works for episodic RL!

Monte Carlo Learning

Given a policy π

For i = 0, 1, 2 … perform "**roll-outs**":

$$
\begin{aligned}
s_0^{(i)} &\sim D_0(\,\cdot\,) \\
a_0^{(i)} &\sim \pi(\,\cdot \mid s_0^{(i)}) \\
s_1^{(i)},\, r_1^{(i)} &\sim D(\,\cdot \mid s_0^{(i)},\, a_0^{(i)}) \\
&\cdots \\
a_{T-1}^{(i)} &\sim \pi(\,\cdot \mid s_{T-1}^{(i)}) \\
s_T^{(i)},\, r_T^{(i)} &\sim D(\,\cdot \mid s_{T-1}^{(i)},\, a_{T-1}^{(i)})
\end{aligned}
$$

Example: continuous states & actions

- Neural network (maybe linear) with input (s, a)
- Update of $Q_{\mathtt{w}}^{\pi}(s_t^{(i)}, a_t^{(i)})$ affects NN output for other states and actions
- Update with SGD by back-propagating $\dfrac{\mathrm{d}\mathscr{L}(\mathtt{w})_t^{(i)}}{\mathrm{d}\mathtt{w}}$
- $\Delta\mathtt{w} = \epsilon \;\left[\hat{q}_t^{\pi\,(i)} - Q_{\mathtt{w}}^{\pi}(s_t^{(i)}, a_t^{(i)})\right] \nabla_{\mathtt{w}} Q_{\mathtt{w}}^{\pi}(s_t^{(i)}, a_t^{(i)})$

For each step, compute: $\hat{q}_t^{\pi\,(i)} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1}^{(i)}$ **1 sample estimate of** $Q^{\pi_{\mathtt{w}}}(s_t^{(i)}, a_t^{(i)})$

Minimize approximation error:

$$\mathscr{L}(\mathtt{w})_t^{(i)} = \frac{1}{2}\left[\hat{q}_t^{\pi\,(i)} - Q_{\mathtt{w}}^{\pi}(s_t^{(i)}, a_t^{(i)})\right]^2$$

Only works for episodic RL!

Temporal Difference Learning

Given a policy π

For i = 0, 1, 2 … perform "**roll-outs**":

$$
\begin{aligned}
s_0^{(i)} &\sim D_0(\,\cdot\,) \\
a_0^{(i)} &\sim \pi(\,\cdot \mid s_0^{(i)}) \\
s_1^{(i)},\, r_1^{(i)} &\sim D(\,\cdot \mid s_0^{(i)},\, a_0^{(i)}) \\
&\cdots \\
a_{T-1}^{(i)} &\sim \pi(\,\cdot \mid s_{T-1}^{(i)}) \\
s_T^{(i)},\, r_T^{(i)} &\sim D(\,\cdot \mid s_{T-1}^{(i)},\, a_{T-1}^{(i)})
\end{aligned}
$$

- Each step, update $Q_{\mathtt{w}}$ to be consistent with next estimate:

$$\mathscr{L}(\mathtt{w})_t^{(i)} = \frac{1}{2}\left[r_{t+1}^{(i)} + \gamma Q_{\mathtt{w}}\left(s_{t+1}^{(i)}, a_{t+1}^{(i)}\right) - Q_{\mathtt{w}}\left(s_t^{(i)}, a_t^{(i)}\right)\right]^2$$

- TD target: $r_{t+1} + \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right)$
- Can learn before knowing final outcome, with incomplete data or in infinite state/action loops.
- Forces $Q_{\mathtt{w}}$ to be *internally consistent* in its predictions.

One link between MC and TD

- The TD error is: $\delta_t = r_{t+1} + \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right) - Q_{\mathtt{w}}\left(s_t, a_t\right)$
- The MC error is: $\hat{q}_t^{\pi} - Q_{\mathtt{w}}\left(s_t, a_t\right) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1} - Q_{\mathtt{w}}\left(s_t, a_t\right)$

- Unroll the MC error: $\hat{q}_t^{\pi} - Q_{\mathtt{w}}\left(s_t, a_t\right) = r_{t+1} + \gamma \hat{q}_{t+1}^{\pi} - Q_{\mathtt{w}}\left(s_t, a_t\right)$

$$
\begin{aligned}
&= r_{t+1} + \gamma \hat{q}_{t+1}^{\pi} - Q_{\mathtt{w}}\left(s_t, a_t\right) + \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right) - \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right) \\
&= \delta_t + \gamma \left(\hat{q}_{t+1}^{\pi} - Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right) \right) \\
&= \delta_t + \gamma \delta_{t+1} + \gamma^2 \left(\hat{q}_{t+2}^{\pi} - Q_{\mathtt{w}}\left(s_{t+2}, a_{t+2}\right) \right) \\
&= \sum_{t'=t}^{T-1} \gamma^{t'-t} \delta_{t'}
\end{aligned}
$$

- Therefore: $\hat{q}_t^{\pi} = Q_{\mathtt{w}}\left(s_t, a_t\right) + \sum_{t'=t}^{T-1} \gamma^{t'-t} \delta_{t'}$

Monte Carlo target is a sum of TD errors!

MC vs TD

Bias/Variance trade-off

- Monte Carlo target $\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1}$ is:
  - **unbiased** (exact in the limit of ∞ data)
  - **high variance** (requires **many** samples for a good estimate)
  - Why? Because it depends on future $T - t$ states/actions

- **Good convergence properties**
- **Not sensitive to initial guess**
- **Simple to use**

- Temporal Difference target $r_{t+1} + \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right)$ is:
  - **low variance** (depends only on next state/action)
  - **biased** (if next guess is inaccurate, update will be wrong)

- **More data-efficient than MC**
- **More unstable with NN**
- **More sensitive to initial guess**

Multi-step targets

- Train approximate Q towards a target value: $\mathscr{L}(\mathtt{w})_t = \frac{1}{2}\left[\hat{q}_t^\pi - Q_\mathtt{w}(s_t, a_t)\right]^2$
- Options for estimation of target value:

  - $\hat{q}_t^\pi = r_{t+1} + \gamma Q_\mathtt{w}(s_{t+1}, a_{t+1})$
  - $\hat{q}_t^\pi = r_{t+1} + \gamma r_{t+2} + \gamma^2 Q_\mathtt{w}(s_{t+2}, a_{t+2})$
  - $\hat{q}_t^\pi = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 Q_\mathtt{w}(s_{t+3}, a_{t+3})$
  - $\ldots$
  - $\hat{q}_t^\pi = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'+1}$

Temporal Difference learning

Monte Carlo learning

- N-steps estimate aim to balance bias and variance

RL through value learning

- **Policy Evaluation:** learn approximate $Q_{\mathtt{w}}^{\pi}(s,a)$ by executing policy rollouts
- **Policy Optimization**: optimize $\pi(a \mid s)$ as to improve $Q^{\pi}(s,a)$
- Value learning-based RL:
  - Train Q-approximator towards TD/MC target:

$$\mathscr{L}(\mathtt{w}) = \mathbb{E}\left[\frac{1}{2}\left[\hat{q}^{\pi_{\mathtt{w}}}(s,a) - Q_{\mathtt{w}}(s,a)\right]^2 \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\,\cdot\,), \\ a \sim \pi_{\mathtt{w}}(\,\cdot \mid s), \end{matrix}\right]$$

  - Sample actions corresponding to greater expected rewards:
    - maximum entropy: $\pi_{\mathtt{w}}(a \mid s) \propto \exp\left[Q_{\mathtt{w}}(s,a)\right]$
    - greedy plus noise: $\pi_{\mathtt{w}}(a \mid s) = \arg\max_a Q_{\mathtt{w}}(s,a) + \xi(a)$
- Learn to estimate returns (**policy evaluation**) and use the estimate to get higher returns (**policy optimization**)

ϵ-greedy policy:

- $\epsilon_t \in (0,1)$ probability of random action
- else:

$$\pi_{\mathtt{w}}(s_t) = \arg\max_a Q_{\mathtt{w}}(s_t,a)$$

max entropy policy:

- discrete actions:

$$\pi_{\mathtt{w}}(a \mid s) = \frac{\exp Q_{\mathtt{w}}(s,a)}{\sum_{ai=1}^{d_A} \exp Q_{\mathtt{w}}(s,ai)}$$

SARSA : on-policy value learning

SARSA

Initialize $\mathtt{w}^{(0)}$

for iteration i in 1, 2, ...

- Collect E episodes by executing $\pi_{\mathtt{w}^{(i)}}$
  - max entropy: $\pi_{\mathtt{w}^{(i)}}(a \mid s) \propto \exp\left[Q_{\mathtt{w}^{(i)}}(s,a)\right]$
  - ϵ-greedy
- # of collected steps (i.e. actions) may be $N = TE$
- For each step t, compute TD target:

$$\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} = r_{t+1} + \gamma Q_{\mathtt{w}^{(i)}}(s_{t+1}, a_{t+1})$$

- Update $\mathtt{w}^{(i)}$ by SGD by minimizing loss:

$$\mathscr{L}^{MSE}(\mathtt{w}^{(i)}) = \frac{1}{N}\sum_{t=0}^{N}\left[\frac{1}{2}\left[\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} - Q_{\mathtt{w}^{(i)}}(s_t, a_t)\right]^2\right]$$

- Here we describe a batch version of SARSA where multiple steps are collected to perform each update
- May use maximum entropy policy or greedy policy with a small chance of random actions.
- Because we use on-policy actions (a_{t+1}) in Bellman update, this is an **on-policy** method.
- Can generalize to **n-step** SARSA or use **Monte Carlo** rewards (e.g. see Mnih 2016)

SARSA : Rummery & Niranjan 1994

Q-learning : off-policy value learning

- Greedy policy is to pick action with max $Q_{\mathtt{w}}$, but we **need** to explore (add exploratory noise).
- **Bellman optimality equation**: optimal value function is computed by following greedy behavior.
  - **Random action** a_t to explore
  - **Deterministic action** $a' = \arg\max Q_{\mathtt{w}}$ to compute **Bellman update**

Q-learning

Initialize $\mathtt{w}^{(0)}$

for iteration i in 1, 2, ..., N

- Collect D episodes by executing $\pi_{\mathtt{w}^{(i)}}$ (max-entropy or ϵ-greedy)
- For each step, compute target with Bellman Eq:

$$\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} = r_{t+1} + \gamma \max_{a'} Q_{\mathtt{w}^{(i)}}(s_{t+1}, a')$$

- Update $\mathtt{w}^{(i)}$ by SGD by minimizing loss:

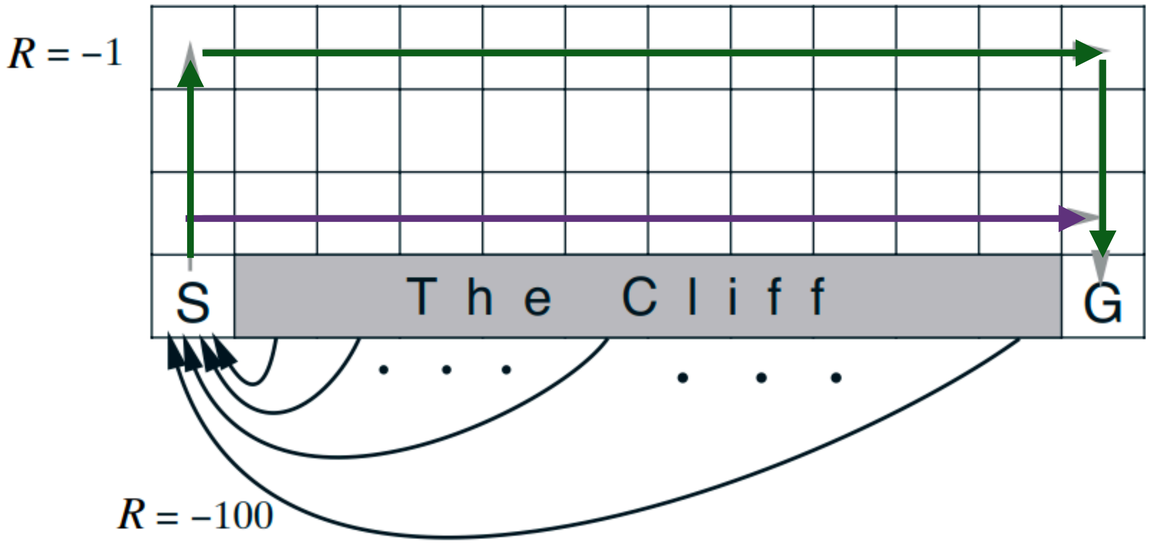
$$\mathscr{L}^{MSE}(\mathtt{w}^{(i)}) = \frac{1}{N} \sum_{t=0}^{N} \left[\frac{1}{2} \left[\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} - Q_{\mathtt{w}^{(i)}}(s_t, a_t) \right]^2 \right]$$

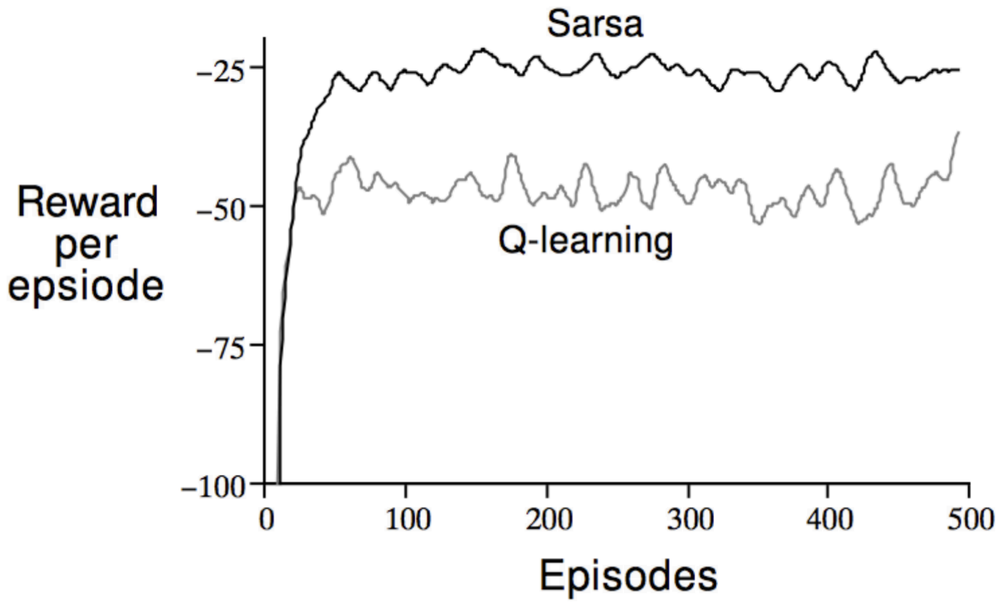
- **Off-policy** method:
  - learn $Q_{\mathtt{w}}$ for a deterministic optimal **policy**
  - while acquiring data with an exploratory **behavior**
- $\mathop{\mathbb{E}}_{X_1, X_2} \left[\max \{X_1, X_2\} \right] \geq \max \left\{ \mathop{\mathbb{E}}_{X_1} [X_1], \mathop{\mathbb{E}}_{X_2} [X_2] \right\}$

  Because $Q_{\mathtt{w}}$ are noisy, Q-learning pathologically over-estimates returns

Q-learning : Watkins & Dayan 1992

Q-learning vs SARSA





Cliff walking example by Sutton

Suppose we attempt to solve this problem with ϵ -greedy policy with both SARSA and Q-learning

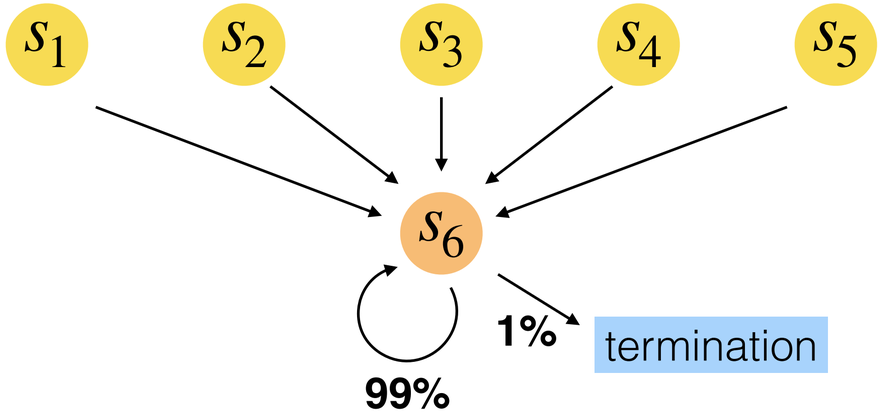
Which method will perform best?

Q-learning approximates Q_w assuming optimal actions: cannot account for ϵ-chance of random actions

SARSA is on-policy: accounts for ϵ-chance of random actions and takes safe path.

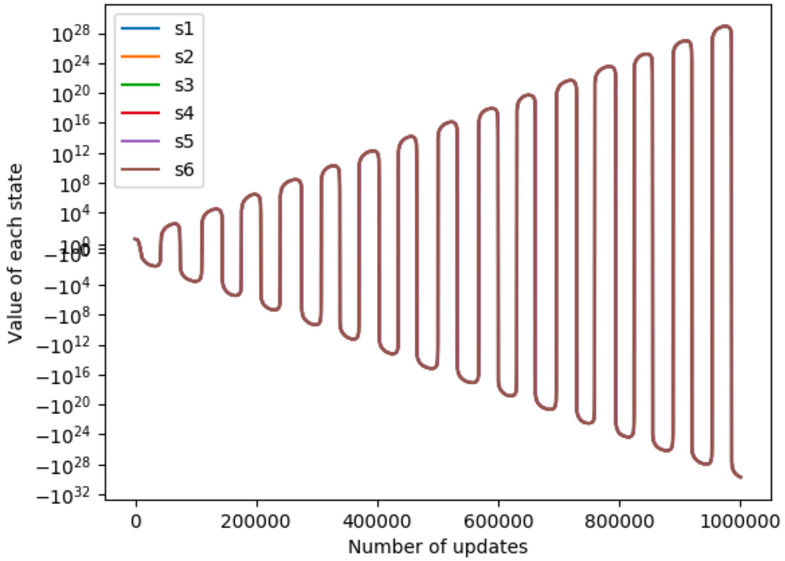
Stability of value-learning based control

Baird's counterexample



- Begin in one of the 5 first states at random
- Always transition to s_6
- Terminate with 1% probability else return to s_6
- All rewards are 0 (including termination)
- Let $V_w(s)$ initialized as:
  - $V_w(s_1) = 2w_1 + w_7$
  - $V_w(s_2) = 2w_2 + w_7$
  - $V_w(s_3) = 2w_3 + w_7$
  - $V_w(s_4) = 2w_4 + w_7$
  - $V_w(s_5) = 2w_5 + w_7$
  - $V_w(s_6) = w_6 + 2w_7$

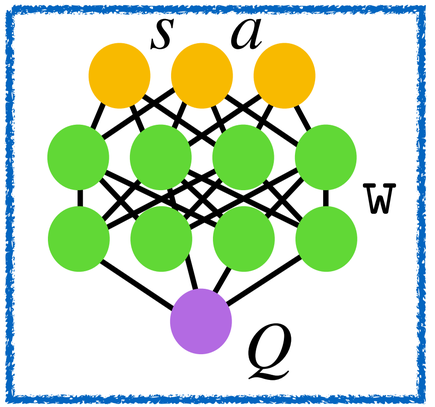
Iteratively update with TD learning:



RL control algorithm:	Tabular	Linear	Non-linear
MC	stable	oscillates around optimum	unstable
SARSA	stable	oscillates around optimum	unstable
Q-learning	stable	unstable	unstable

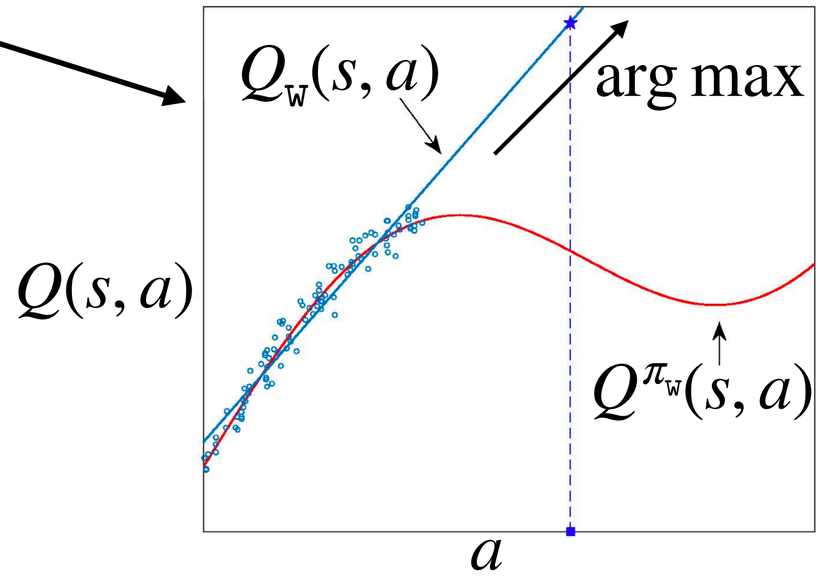
Q-learning for continuous actions

- Continuous-valued action vectors prevalent in robotics / PDEs / ODEs (even modern video-games!)
- Both s and a are vectors: natural choice for Q-network seems



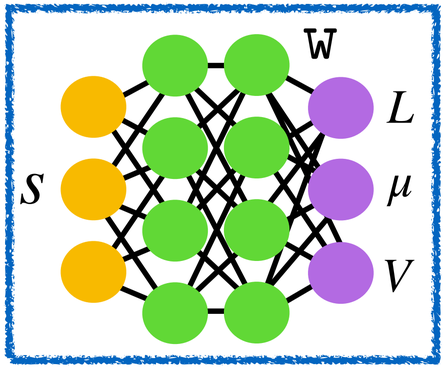
- **But:**
  - Expensive to find maxima of a NN (policy)
  - How to ensure Q_w is good approximation in all action space?
  - If linear approximation: $\arg\max_a Q_w(s,a) \to \infty$

How can we, for a given s, output a well-behaved approximation of Q_w over the entire action space? (like DQN)



Continuous Q-learning: Normalized Advantage Functions

- NAF network architecture:



NAF

Initialize $\mathtt{w}^{(0)}$

for iteration t in 1, 2, ..., N

- Collect D episodes with $\beta(\cdot \mid s) = \mathcal{N}\left(\mu_{\mathtt{w}^{(i)}}(s),\, \sigma^2\right)$
- For each step, compute target with Bellman Eq:

$$\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} = r_{t+1} + \gamma V_{\mathtt{w}^{(i)}}(s_{t+1})$$

- Update $\mathtt{w}^{(t)}$ with SGD by minimizing loss:

$$\mathcal{L}^{MSE}(\mathtt{w}^{(i)}) = \frac{1}{N}\sum_{t=0}^{N}\left[\frac{1}{2}\left[\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} - Q_{\mathtt{w}^{(i)}}(s_t, a_t)\right]^2\right]$$

- Output $V_{\mathtt{w}}(s)$ approximate state value
- $\mu_{\mathtt{w}}(s)$ vector of size $dim(A)$
- $L_{\mathtt{w}}(s)$ lower triangular matrix*:

$$L_{\mathtt{w}}(s) = \begin{bmatrix} l_{\mathtt{w}}^{(1,1)} & 0 & 0 & \cdots & 0 \\ l_{\mathtt{w}}^{(2,1)} & l_{\mathtt{w}}^{(2,2)} & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{\mathtt{w}}^{(d_A,1)} & l_{\mathtt{w}}^{(d_A,2)} & l_{\mathtt{w}}^{(d_A,3)} & \cdots & l_{\mathtt{w}}^{(d_A,d_A)} \end{bmatrix}$$

- $L_{\mathtt{w}}(s)L_{\mathtt{w}}^T(s)$ is positive definite!
- $Q_{\mathtt{w}}(s,a) = V_{\mathtt{w}} - \frac{1}{2}(a-\mu_{\mathtt{w}})^T L_{\mathtt{w}} L_{\mathtt{w}}^T (a-\mu_{\mathtt{w}})$
- $\mu_{\mathtt{w}}(s)$ is action with optimal value $V_{\mathtt{w}}(s)$!

* To ease training (bijection) elements on the diagonal should be pos. def. $l_{\mathtt{w}}^{(i,i)} \leftarrow \log\left[1 + \exp\left(l_{\mathtt{w}}^{(i,i)}\right)\right]$

NAF : Gu et al. 2016

Reinforcement Learning Workshop

Day 4
Policy Optimization

Guido Novati

CSElab
Computational Science & Engineering Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RL: Markov decision process with parameterized policy

RL learns a policy

- Function that computes actions given states.
- **Improvable** i.e. it has parameters `w` which can be optimized.
- Some stochasticity is required to explore dynamics and possible improvements.

$$s_0 \sim D_0(\cdot)$$

Environment at random initial condition

$$a_0 \sim \pi_{\mathtt{w}}(\cdot \mid s_0)$$

Agent computes policy given state

$$s_1, r_1 \sim D(\cdot \mid s_0, a_0)$$

Environment advances in time and agent receives reward

$$\dots$$

$$a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_t)$$

$$s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t)$$

$$\dots$$

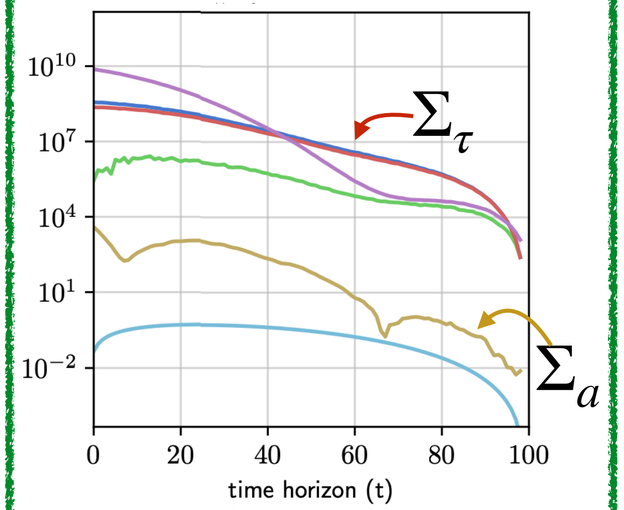
$$J(w) = \mathbb{E}\left[\sum_t^{\infty} r_t \;\middle|\; \begin{array}{c} a_t \sim \pi_w(\cdot \mid s_t) \\ s_{t+1} \sim D(\cdot \mid a_t, s_t) \end{array} \right]$$

optimal parameters

$$\mathbf{w}^\star = \arg\max_{\mathbf{w}} J(\mathbf{w})$$

Why policy optimization

- Generally **policies are simpler** functions **than Q-functions**.
  - Compare *learning entire landscape of rewards* versus *identifying actions correlated with higher rewards*.
  - Q(s,**a**) value affected by **a** and many actions into the future:
    - Need many trials to average out effect of future possible steps (noise) to get at value of individual actions (signal).
    - For most problems, single actions have little effect (can be corrected later) and therefore Q is very flat over **a**.



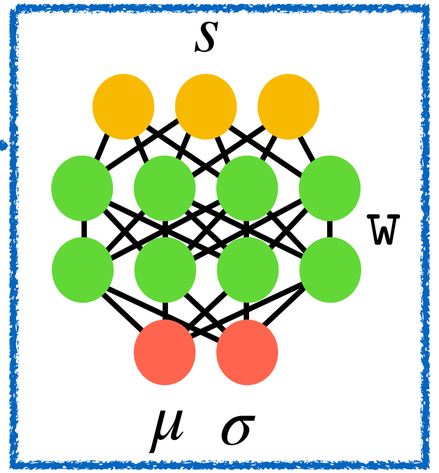
Variance of rewards estimate: due to observing a single trajectory (Σ_τ), due to sampling a single action (Σ_a). (Tucker et al., 2018)

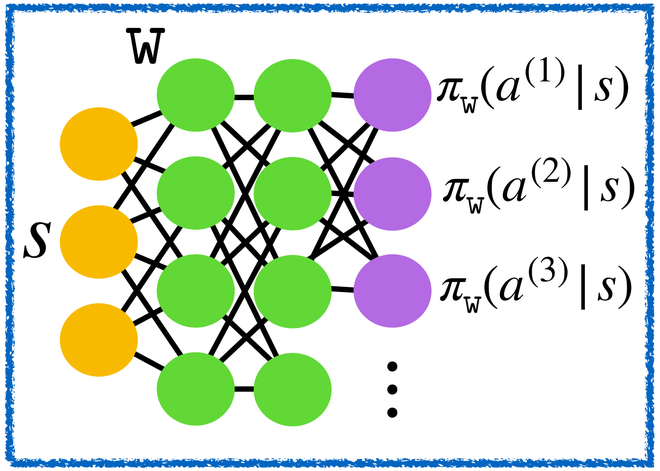
- Even if we know $Q_w(s,a)$, it may be difficult to:
  - sample $\pi_w(a \mid s) \propto \exp\left[Q_w(s,a)\right]$ (rejection sampling)
  - optimize $\pi_w(s) = \arg\max_a Q_w(s,a)$

Not an issue for discrete action problems!

Parametric policies: examples

- Directly parameterize the policy, optimize the weights:
  - Gaussian Policy $a \sim \mathcal{N}\left[\mu_{\mathtt{w}}(s),\ \sigma_{\mathtt{w}}{}^2(s)\right]$
    find Gaussian where optimal actions are most likely
  - (Generalized) Bernoulli Policy
    one NN output for each action, must sum to 1,
    (softmax output layer)
  - Deterministic policy $a = \pi_{\mathtt{w}}(s)$
    output is directly optimal action (e.g. vector)
    exploration: must add noise to the action or the weights





Cross-entropy method (CEM) for policy optimization

Initial guess on policy parameter vector $\bar{\mathtt{w}}^{(0)}$
for iteration i in 1, 2, ..., N
    for population member p = 1, 2, ..., P

- Sample $\xi_p \sim \mathcal{N}(0\ ,\ \mathbf{I})$
- Set $\mathtt{w}_p^{(i)} = \bar{\mathtt{w}}^{(i)} + \sigma\xi_p$
- Execute E episodes with policy $\pi_{\mathtt{w}_p^{(i)}}$
- Measure average utility $J(\mathtt{w}_p^{(i)})$

- Sort population according to average utility.
- Select B best candidates with $B < P$

$$\bar{\mathtt{w}}^{(i+1)} = \frac{1}{B}\sum_{p=1}^{B} \mathtt{w}_p^{(i)}$$

- Optionally, update sampling noise:

$$\sigma^{2,\,(i+1)} = \frac{1}{B}\sum_{p=1}^{B}\left(\mathtt{w}_p^{(i)} - \bar{\mathtt{w}}^{(i+1)}\right)^T\left(\mathtt{w}_p^{(i)} - \bar{\mathtt{w}}^{(i+1)}\right)$$

$$\max_{\mathtt{w}} J(\mathtt{w}) = \max_{\mathtt{w}} \mathbb{E}\left[\sum_t^T r_t \,\middle|\, \begin{array}{c} a_t \sim \pi_w(\cdot\,|\,s_t) \\ s_{t+1} \sim D(\cdot\,|\,a_t, s_t) \end{array}\right]$$

- Evolutionary algorithm.
- Treat J(w) as black box.
- Ignores all other information (states, actions, intermediate rewards).
- Because noise is intrinsic in evolutionary process, policy can be deterministic.

Evolution Strategies (ES) algorithm for policy optimization

Initial guess on policy parameter vector $\bar{\mathtt{w}}^{(0)}$
for iteration i in 1, 2, ..., N
for population member p = 1, 2, ..., P

- Sample $\xi_p \sim \mathcal{N}(0\ ,\ \mathbf{I})$
- Set $\mathtt{w}_p^{(i)} = \bar{\mathtt{w}}^{(i)} + \sigma\xi_p$
- Execute E episodes with policy $\pi_{\mathtt{w}_p^{(i)}}$
- Measure average utility $J(\mathtt{w}_p^{(i)})$

- Update mean:

$$\bar{\mathtt{w}}^{(i+1)} = \bar{\mathtt{w}}^{(i)} + \frac{\epsilon}{\sigma P}\sum_{p=1}^{P} J(\mathtt{w}_p^{(i)})\xi_p$$

- Optionally, update sampling noise:

$$\sigma \leftarrow \sigma + \frac{\epsilon}{\sigma P}\sum_{p=1}^{P} J(\mathtt{w}_p^{(i)})\frac{\|\xi_p\|^2 - 1}{dim(\mathtt{w})}$$

$$\max_{\mathtt{w}} J(\mathtt{w}) = \max_{\mathtt{w}} \mathbb{E}\left[\sum_{t}^{T} r_t \,\middle|\, \begin{matrix} a_t \sim \pi_w(\cdot\,|\,s_t) \\ s_{t+1} \sim D(\cdot\,|\,a_t, s_t) \end{matrix}\right]$$

- Evolutionary algorithm.
- Treat J(w) as black box.
- Ignores all other information (states, actions, intermediate rewards).
- Because noise is intrinsic in evolutionary process, policy can be deterministic.

Evolution Strategies : Salimans et al. 2017

Alternative optimization methods

Closely Related Approaches

CEM:

for iter i = 1, 2, ...

for population member e = 1, 2, ...

sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$

execute roll-outs under $\pi_{\theta^{(e)}}$

store $(\theta^{(e)}, U(e))$

endfor

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

where $\bar{e}$ indexes over top p %

endfor

From Pieter Abbeel's workshop at ICML'16

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

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

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

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

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

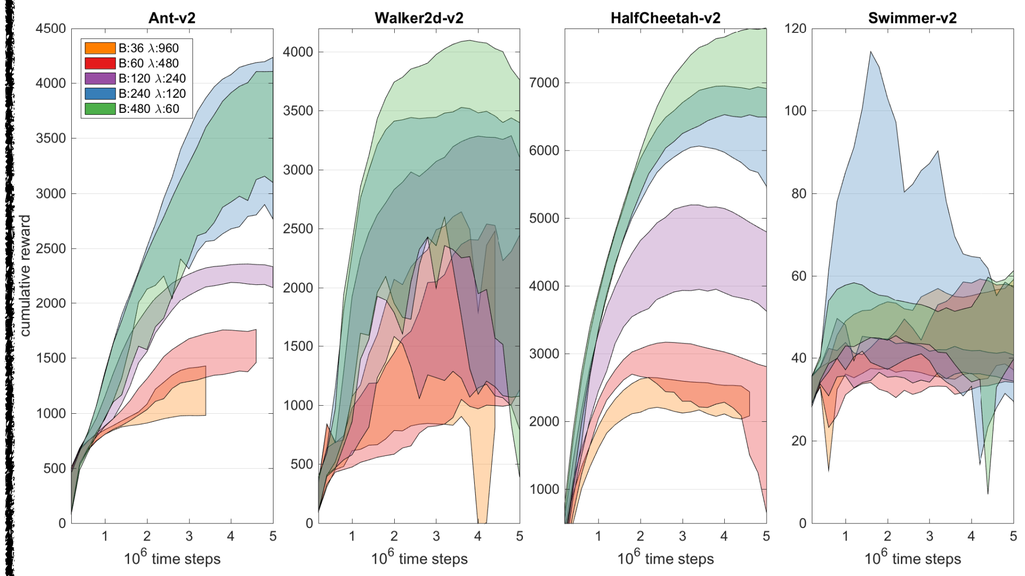
$$(\mu^{(i+1)}, \Sigma^{(i+1)}) = \arg\max_{\mu,\Sigma} \sum_{\bar{e}} w(U(\bar{e})) \log \mathcal{N}(\theta^{(\bar{e})}; \mu, \Sigma)$$

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

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

Pros&Cons of evolutionary policy optimization methods

- Simple implementation and robust.
- Fundamental limits:
  - **Probability** of finding **good policy update** by random sampling **goes to 0 as** $1/\sqrt{dim\,(\mathtt{w})}$ (most update directions are orthogonal to the ideal one).
  - There is **no temporal decomposition of the RL task**. For each $\mathtt{w}$ we only look at average returns. E.g. we do not exploit $Q^{\pi_{\mathtt{w}}}(s,a)$, we only use $\mathbb{E}\left[Q^{\pi_{\mathtt{w}}}(s,a)\right]$.



- **Example** from previous group meeting (not exactly the same, here I optimized off-policy objective)
- Kept **constant** the **workload**:
  [pop. size]*[N of evaluations per sampled $\mathtt{w}$]
- Increasing N evals is better than increasing pop. size.
- Variance of RL objective higher than variance of update estimate
- Temporal decomposition of RL is key for data-efficiency!

Likelihood ratio policy gradient (1/2)

expand notation for conveniency:

- Consider space $\mathscr{T}$ of all possible episodes $\tau = \{s_0, a_0, s_1, \cdots, s_{T-1}, a_{t-1}, s_T\}$, $\tau \in \mathscr{T}$.
- Associated return $R(\tau) = \Sigma_{i=1}^{T} r_t$
- Maximize rewards over episodes $J(\mathtt{w}) = \mathbb{E}\left[R(\tau) \middle| \tau \sim \mathbb{P}_{\mathtt{w}}(\,\cdot\,)\right]$
- Probability of episode depends on policy parameters $\mathtt{w}$.
- Equivalently write optimization: $\max_{\mathtt{w}} J(\mathtt{w}) = \max_{\mathtt{w}} \int_{\mathscr{T}} R(\tau)\mathbb{P}_{\mathtt{w}}(\tau)\mathrm{d}\tau$

Likelihood ratio policy gradient (2/2)

Take the gradient of $J(\mathtt{w}) = \int_{\mathcal{T}} R(\tau)\mathbb{P}_{\mathtt{w}}(\tau)\mathrm{d}\tau$

$$\begin{aligned}
\nabla_{\mathtt{w}} J(\mathtt{w}) &= \nabla_{\mathtt{w}} \int_{\mathcal{T}} R(\tau)\mathbb{P}_{\mathtt{w}}(\tau)\mathrm{d}\tau \\
&= \int_{\mathcal{T}} R(\tau)\, \nabla_{\mathtt{w}} \mathbb{P}_{\mathtt{w}}(\tau)\mathrm{d}\tau \\
&= \int_{\mathcal{T}} R(\tau) \frac{\mathbb{P}_{\mathtt{w}}(\tau)}{\mathbb{P}_{\mathtt{w}}(\tau)} \nabla_{\mathtt{w}} \mathbb{P}_{\mathtt{w}}(\tau)\mathrm{d}\tau \\
&= \int_{\mathcal{T}} R(\tau)\ \mathbb{P}_{\mathtt{w}}(\tau)\, \nabla_{\mathtt{w}} \log\left[\mathbb{P}_{\mathtt{w}}(\tau)\right] \mathrm{d}\tau \\
&= \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}}(\cdot)} \left[R(\tau)\, \nabla_{\mathtt{w}} \log\left[\mathbb{P}_{\mathtt{w}}(\tau)\right] \right]
\end{aligned}$$

Alternative perspective: sample episodes with current $\mathtt{w}_{old}$ and estimate J for optimal $\mathtt{w}$:

$$J(\mathtt{w}) = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}}(\cdot)} \left[R(\tau)\right] = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}_{old}}(\cdot)} \left[R(\tau) \frac{\mathbb{P}_{\mathtt{w}}(\tau)}{\mathbb{P}_{\mathtt{w}_{old}}(\tau)} \right]$$

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}_{old}}(\cdot)} \left[R(\tau) \frac{\nabla_{\mathtt{w}} \mathbb{P}_{\mathtt{w}}(\tau)}{\mathbb{P}_{\mathtt{w}_{old}}(\tau)} \right]$$

$$\nabla_{\mathtt{w}} J(\mathtt{w})\Big|_{\mathtt{w}=\mathtt{w}_{old}} = \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}_{old}}(\cdot)} \left[R(\tau) \frac{\nabla_{\mathtt{w}} \mathbb{P}_{\mathtt{w}}(\tau)\Big|_{\mathtt{w}=\mathtt{w}_{old}}}{\mathbb{P}_{\mathtt{w}_{old}}(\tau)} \right]$$

$$= \mathop{\mathbb{E}}_{\tau\sim\mathbb{P}_{\mathtt{w}_{old}}(\cdot)} \left[R(\tau)\, \nabla_{\mathtt{w}} \log \mathbb{P}_{\mathtt{w}}(\tau)\Big|_{\mathtt{w}=\mathtt{w}_{old}} \right]$$

Intuition: evaluate policy and change parameters to make good trajectory more likely.

However, what is P?

Policy gradient: exploit the sequential structure

Decompose prob. of trajectory in prob of states (given by dynamics D) and prob of actions (policy $\pi_{\mathtt{w}}$):

notation

$a_t \sim \pi(\cdot \mid s_t)$	: policy prob. distribution
$s_{t+1} \sim D(\cdot \mid s_t, a_t)$	: dynamics prob. distribution
$\pi(a_t \mid s_t)$	: numerical value, prob. of a_t
$D(s_{t+1} \mid s_t, a_t)$	: numerical value, prob. of s_{t+1}

dynamics **policy**

$$\nabla_{\mathtt{w}} \log \mathbb{P}_{\mathtt{w}}(\tau) = \nabla_{\mathtt{w}} \log \left[\prod_{t=1}^{T} D(s_t \mid s_{t-1}, a_{t-1}) \cdot \pi_{\mathtt{w}}(a_{t-1} \mid s_{t-1}) \right]$$

$$= \nabla_{\mathtt{w}} \left[\sum_{t=1}^{T} \log D(s_t \mid s_{t-1}, a_{t-1}) + \sum_{t=0}^{T-1} \log \pi_{\mathtt{w}}(a_t \mid s_t) \right]$$

$$= \sum_{t=0}^{T-1} \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a_t \mid s_t)$$

analytically computable!!! (backpropagation)

remove the trajectory-based notation:

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathop{\mathbb{E}}_{\tau \sim \mathbb{P}_{\mathtt{w}}} \left[R(\tau) \nabla_{\mathtt{w}} \log \mathbb{P}_{\mathtt{w}}(\tau) \right] = \mathbb{E} \left[\left(\sum_{t'=1}^{T} r_{t'} \right) \left(\sum_{t=0}^{T-1} \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a_t \mid s_t) \right) \,\middle|\, \begin{array}{c} a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_t), \\ s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{array} \right]$$

Policy Gradient : Barto et al. 1983

Policy gradient: analytical properties (1/2)

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\left(\sum_{t'=1}^{T} r_{t'}\right)\left(\sum_{t=0}^{T-1} \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a_t \mid s_t)\right) \,\middle|\, \begin{array}{c} a_t \sim \pi_{\mathtt{w}}(\cdot \mid s_t), \\ s_{t+1}, r_{t+1} \sim D(\cdot \mid s_t, a_t) \end{array}\right]$$

- Formula computed performing many simulations with $\pi_{\mathtt{w}}$.
- Write more compactly with the state visitation frequency:

$$\eta^{\pi_w}(s) \propto \lim_{T\to\infty} \sum_{t=0}^{T} P\left(s = s_t \;\middle|\; s_0 = s,\; a_t \sim \pi_w(\cdot \mid s_t)\right)$$

- Probability of visiting a state by sampling environment ad infinitum with policy

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\left(\sum_{t'=t+1}^{T} r_{t'} \,\middle|\, \begin{array}{c} s_t = s \\ a_t = a \end{array}\right) \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s) \,\middle|\, \begin{array}{c} s \sim \eta^{\pi_w}(\cdot), \\ a \sim \pi_{\mathtt{w}}(\cdot \mid s) \end{array}\right]$$

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\hat{q}^{\pi_{\mathtt{w}}}(s,a) \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s) \,\middle|\, \begin{array}{c} s \sim \eta^{\pi_w}(\cdot), \\ a \sim \pi_{\mathtt{w}}(\cdot \mid s) \end{array}\right]$$

Policy gradient theorem: valid for any differentiable stochastic policy

Policy gradient: analytical properties (2/2)

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\hat{q}^{\pi_{\mathtt{w}}}(s,a)\,\nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\,\cdot\,). \\ a \sim \pi_{\mathtt{w}}(\,\cdot \mid s) \end{matrix}\right]$$

Consider a baseline function $b(?)$, we can write expectation over actions for a given s:

$$\int_{\mathcal{A}} \left(\hat{q}^{\pi_{\mathtt{w}}}(s,a) - b(?)\right) \pi_{\mathtt{w}}(a \mid s)\,\nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s)\mathrm{d}a$$

What can be the arguments of $b(?)$ such that result is not affected (unbiased operation)?

$$0 = \int_{\mathcal{A}} b(?)\ \pi_{\mathtt{w}}(a \mid s)\,\nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s)\mathrm{d}a = \int_{\mathcal{A}} b(?)\ \nabla_{\mathtt{w}} \pi_{\mathtt{w}}(a \mid s)\mathrm{d}a$$

$$= b(?) \int_{\mathcal{A}} \nabla_{\mathtt{w}} \pi_{\mathtt{w}}(a \mid s)\mathrm{d}a$$

$$= b(?)\,\nabla_{\mathtt{w}} \int_{\mathcal{A}} \pi_{\mathtt{w}}(a \mid s)\mathrm{d}a$$

$$= b(?)\,\nabla_{\mathtt{w}} 1 = 0$$

As long as $b(?)$ is not a function of a then the result is 0. Including:

- Any reward obtained before performing a
- State-dependent or constant baseline average rewards

E.g.:

$$\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\left(\hat{q}^{\pi_{\mathtt{w}}}(s,a) - \bar{R}\right) \nabla_{\mathtt{w}} \log \pi_{\mathtt{w}}(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\,\cdot\,) \\ a \sim \pi_{\mathtt{w}}(\,\cdot \mid s) \end{matrix}\right]$$

Average cumulative reward:

$$\bar{R} = \frac{1}{K} \sum_{k=1}^{K} \sum_{t=1}^{T} r_t$$

Policy gradient: recap & intuition

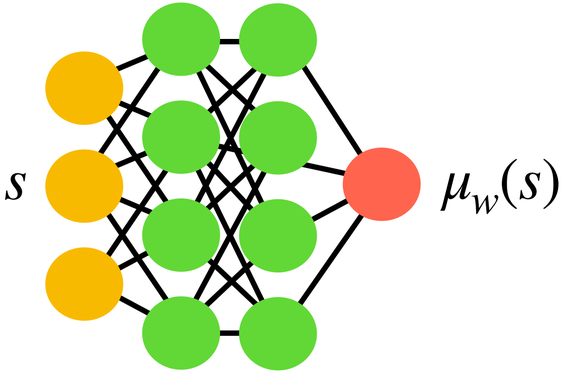
- **Policy-learning methods**: directly find the policy that maximizes the rewards.
- Gaussian policy: $\pi_{\mathtt{w}}(a \mid s) \propto \exp\left[-\frac{1}{2\sigma^2}\left(a - \mu_{\mathtt{w}}(s)\right)^2\right]$
- Update by SGD: $w_{k+1} = w_k + \epsilon\, \mathbb{E}\left[\hat{A}^{\pi_w}(s,a)\ \nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\cdot) \\ a \sim \pi_w(\cdot \mid s) \end{matrix}\right]$

  prob of a state given the policy

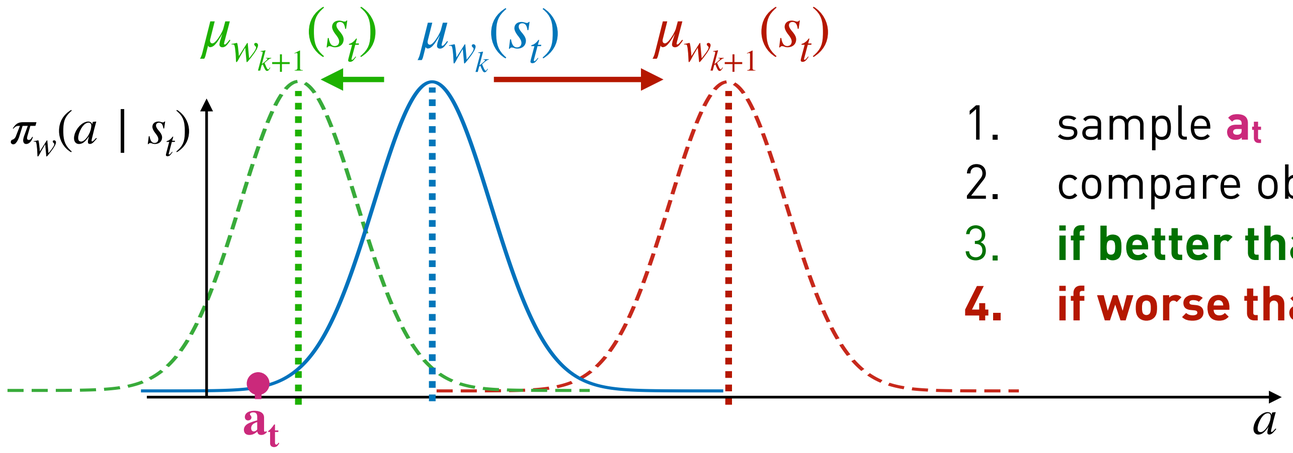
  advantage estimated from rewards

  prob of action given the state
- Chain rule: $w_{k+1} = w_k + \epsilon\, \mathbb{E}\left[\hat{A}^{\pi_w}(s,a)\ \frac{a - \mu_w(s)}{\sigma^2} \cdot \nabla_w \mu_w(s)\right]$

Assume for simplicity fixed standard deviation of the policy



$$\hat{A}_t^{\pi_w} = \sum_{t'=t}^{T-1} \gamma^{t'-t}\, r_{t'+1} - \bar{R}$$



1. sample a_t
2. compare obtained rewards relative to expectation
3. **if better than expected ($A_t > 0$) : move μ_w towards a_t**
4. **if worse than expected ($A_t < 0$) : move μ_w away from a_t**

Pure formalization of "trial and error"!

Vanilla policy gradients: REINFORCE

Initialize $\mathtt{w}^{(0)}$ and baseline $b^{(0)} = 0$
for iteration i in 1, 2, ..., N

- Collect E episodes by executing $\pi_{\mathtt{w}^{(i-1)}}$
- For time step t in each episode compute:

$$\hat{q}_t^{\pi_{\mathtt{w}}} = \Sigma_{t'=t}^{T-1} \gamma^{t'-t} r_{t'+1}$$

- Update: $b^{(i)} = \frac{1}{E \cdot T} \sum_{t=1}^{E \cdot T} \hat{q}_t^{\pi_{\mathtt{w}}}$
- Update $\mathtt{w}^{(i)}$ by SGD by minimizing loss:

$$\mathscr{L}^{PG}\left(\mathtt{w}^{(i-1)}\right) = -\frac{1}{E \cdot T} \sum_{t=1}^{E \cdot T} \left[\hat{q}_t^{\pi_{\mathtt{w}}} - b^{(i-1)}\right] \log \pi_{\mathtt{w}^{(i-1)}}(a_t \mid s_t)$$

$$\mathtt{w}^{(i)} = \mathtt{w}^{(i-1)} - \epsilon \nabla_{\mathtt{w}^{(i-1)}} \mathscr{L}^{PG}\left(\mathtt{w}^{(i-1)}\right)$$

- As formulated: simple, unbiased, but brittle: policy can diverge, requires small learning rates.
- Baseline can be replaced with NN approximation $V_{\mathtt{w}'}(s_t)$ to improve results.
- Algorithm assumes each lasts T steps. Simple extension if each episode has different length.
- Use automatic differentiation library (pytorch/ tensorflow) to optimize policy according to $\mathscr{L}^{PG}$.
- Note: $\left[\hat{q}_t^{\pi_{\mathtt{w}}} - b^{(i-1)}\right]$ is a constant in the calculation of the gradients.

REINFORCE : Williams 1992

Policy Gradient: remarks

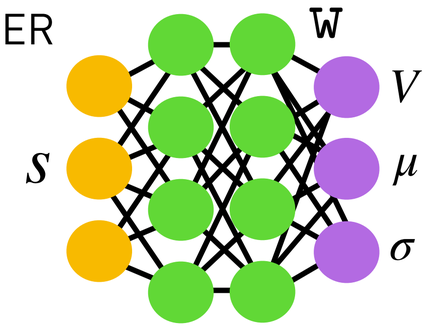
- In principle, given enough data, PG guarantees policy improvements.
- 1-sample estimate advantage: $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots - b(s_t)$
- Baseline reduces variance. Optimally if $b(s_t) \approx V^{\pi_w}(s_t)$.
- But value affected by $a_{t+1}, a_{t+2}, a_{t+3}, \cdots$
- For PG, effect of a_t is signal, future actions are noise.
- Discount factor $\gamma \in [0,1]$ reduces importance of long term dependencies.
- Similar to using time horizon (cutoff) of $1/(1-\gamma)$ time steps.
- Yields a lower variance, but biased (affects optimal policy), gradient.

more variance reduction: TD(λ) (Sutton & Barto, 1998),
Finite Horizon Methods (A2C, Mnih et al., 2016),
Generalized Advantage Function (GAE, Schulman et al., 2016)

Actor-critic : reducing the variance of PG

- Actor-critic methods update two approximators:
  - Actor (policy) perform policy optimization
  - Critic (value function) perform policy evaluation

e.g. V-RACER



One NN is both actor and critic: 2 optimization objectives collaborate to learning robust hidden layer structure

With an approximate $Q_{w'}$ or $V_{w'}$ we can write:

$$\nabla_w J(w) = \mathbb{E}\left[\Psi^{\pi_w}(s,a)\,\nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\cdot) \\ a \sim \pi_w(\cdot \mid s) \end{matrix}\right]$$

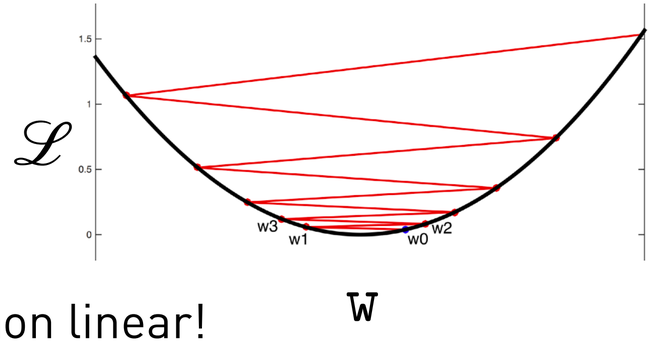
E.g.: $\Psi^{\pi_w}(s,a) = \hat{q}^{\pi_w}(s,a) - \bar{R}$ $\qquad$ $\Psi^{\pi_w}(s,a) = r' + \gamma V_{w'}(s') - V_{w'}(s)$

$\Psi^{\pi_w}(s,a) = Q_{w'}(s,a)$ $\qquad$ $\Psi^{\pi_w}(s,a) = \hat{q}^{\pi_w}(s,a) - V_{w'}(s) = \hat{A}^{\pi_w}(s,a)$

Usual tradeoff: actor-critic methods reduce the variance of the policy gradient at the cost of risking biased estimates (if the critic is wrong)

Policy step size

- Large steps in supervised learning:
- RL steps may completely change policy.
- Even small parameter changes (small learning rates): NN are non linear!
- On-policy data will be unrelated to past experiences.
- May destroy all previous learning progress (catastrophic forgetting)
- Constrained optimization:



$$\text{maximize: } J(\mathtt{w}) = \mathop{\mathbb{E}}_{\pi_{\mathtt{w}(i)}} \left[\frac{\pi_{\mathtt{w}}(a \mid s)}{\pi_{\mathtt{w}^{(i)}}(a \mid s)} A^{\pi_{\mathtt{w}(i)}}(s, a) \right]$$

$$\text{such that: } D_{KL} \left[\pi_{\mathtt{w}}(\cdot \mid s) \parallel \pi_{\mathtt{w}^{(i)}}(\cdot \mid s) \right] < \delta$$

D_{KL}: distance between distributions

$$D_{KL} \left[P(\cdot) \parallel Q(\cdot) \right] = \int P(x) \log \frac{P(x)}{Q(x)} \mathrm{d}x$$

Natural policy gradient (1/2)

$$D_{KL}\left[\pi_{\mathtt{w}}\|\pi_{\mathtt{w}^{(i)}}\right] \approx \cancel{D_{KL}\left[\pi_{\mathtt{w}^{(i)}}\|\pi_{\mathtt{w}^{(i)}}\right]} + \left(\mathtt{w}-\mathtt{w}^{(i)}\right)\nabla_{\mathtt{w}} D_{KL}\left[\pi_{\mathtt{w}}\|\pi_{\mathtt{w}^{(i)}}\right]_{\mathtt{w}=\mathtt{w}^{(i)}} + \frac{1}{2}\left(\mathtt{w}-\mathtt{w}^{(i)}\right)^T F\left(\mathtt{w}-\mathtt{w}^{(i)}\right)$$

= 0 = 0

$$J(\mathtt{w}) \approx J(\mathtt{w}^{(i)}) + (\mathtt{w}-\mathtt{w}^{(i)})\nabla_{\mathtt{w}} J(\mathtt{w})\,|_{\mathtt{w}=\mathtt{w}^{(i)}}$$

policy gradient

denote $\hat{g} = \nabla_{\mathtt{w}} J(\mathtt{w})\,|_{\mathtt{w}=\mathtt{w}^{(i)}}$

Fisher Information matrix

$$F := \nabla^2_{\mathtt{w}} D_{KL}\left[\pi_{\mathtt{w}}\|\pi_{\mathtt{w}^{(i)}}\right]_{\mathtt{w}=\mathtt{w}^{(i)}}$$

Sensitivity of policy to parameter changes (metric tensor).

Approximate, linearized problem: $\mathtt{w}^{(i+1)} = \arg\max_{\mathtt{w}} \hat{g}\cdot(\mathtt{w}-\mathtt{w}^{(i)})$

$$\text{s.t. } \frac{1}{2}\left(\mathtt{w}-\mathtt{w}^{(i)}\right)^T F\left(\mathtt{w}-\mathtt{w}^{(i)}\right) < \delta$$

Analytical solution: $\mathtt{w}^{(i+1)} = \mathtt{w}^{(i)} + \sqrt{\frac{2\delta}{\hat{g}^T F^{-1}\hat{g}}} F^{-1}\hat{g}$

natural policy gradient

Natural Policy Gradient : Kakade, 2002

Natural policy gradient (2/2)

$$\mathtt{w}^{(i+1)} = \mathtt{w}^{(i)} + \sqrt{\frac{2\delta}{\hat{g}^T F^{-1} \hat{g}}} F^{-1} \hat{g}$$

policy gradient

natural policy gradient

- Fisher information matrix F : sensitivity of policy to parameter changes.
- Even if policy is non-linear NN: **natural policy gradient produces the same policy change regardless of the model parameterization**.
- **Issue**: quadratic computational complexity (it is a Hessian).
- How to pick δ to guarantee improvements: `Trust Region Policy Optimization, Schulman et al. 2015`
- Linear complexity approximation (most popular): `Proximal Policy Optimization, Schulman et al. 2017`

Reinforcement Learning Workshop

Day 5
Experience Replay

Guido Novati

CSElab
Computational Science & Engineering Laboratory

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Why Experience Replay

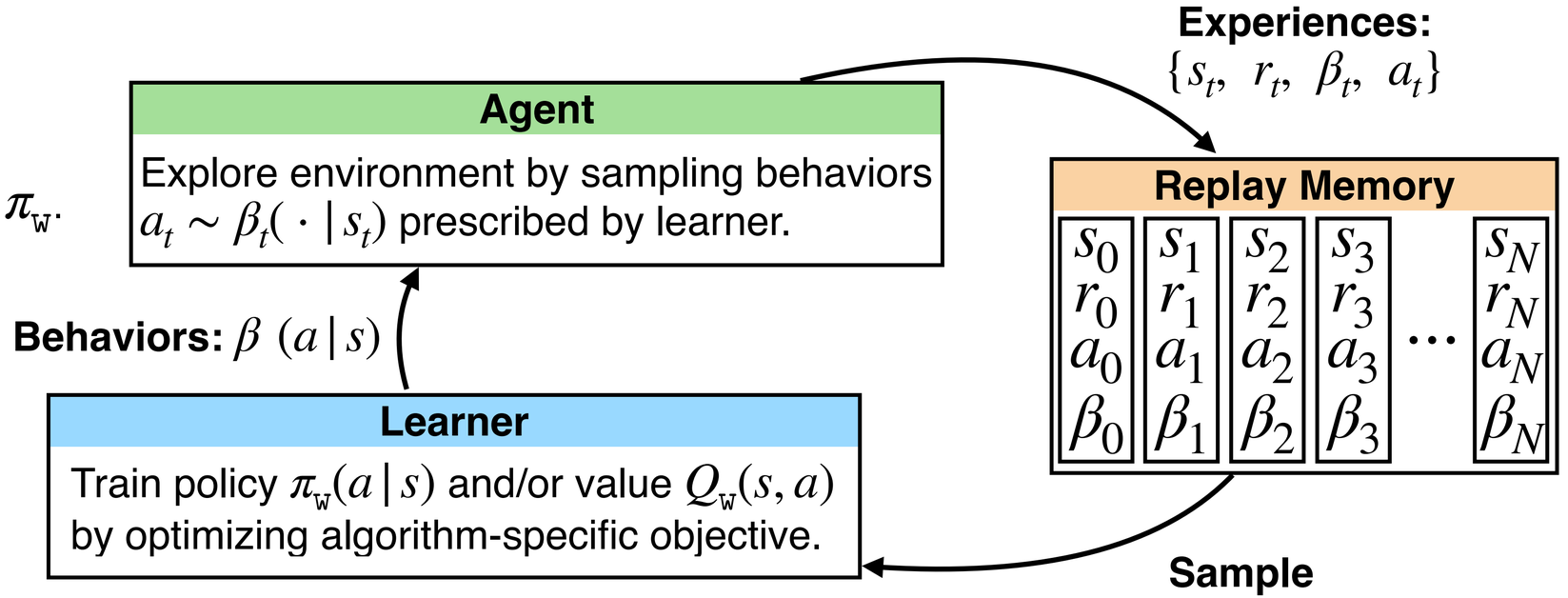
- Algorithms seen so far alternate between evaluating π_w and updating π_w:
  - **Simple** implementation.
  - **Sound** theoretical analysis.
  - After update, old experiences are discarded: extremely **data-inefficient.**
  - Experiences gathered sequentially: **temporal correlations** in training data.
  - Temporal correlations: **reduced effective Monte Carlo sample size** (high update variance).
  - High update variance: **more data** is needed **to compute** good **update** estimates.

- **Solution: re-use previously collected experiences** over multiple learning iterations.
  - Allows **high data-efficiency**
  - Allows **sampling uncorrelated data** from a training set
  - Takes RL closer to Supervised Learning (allows employing more SL techniques)
  - More cumbersome implementation and **requires compatible algorithms**.

`Experience Replay : Lin 1992`

Experience replay

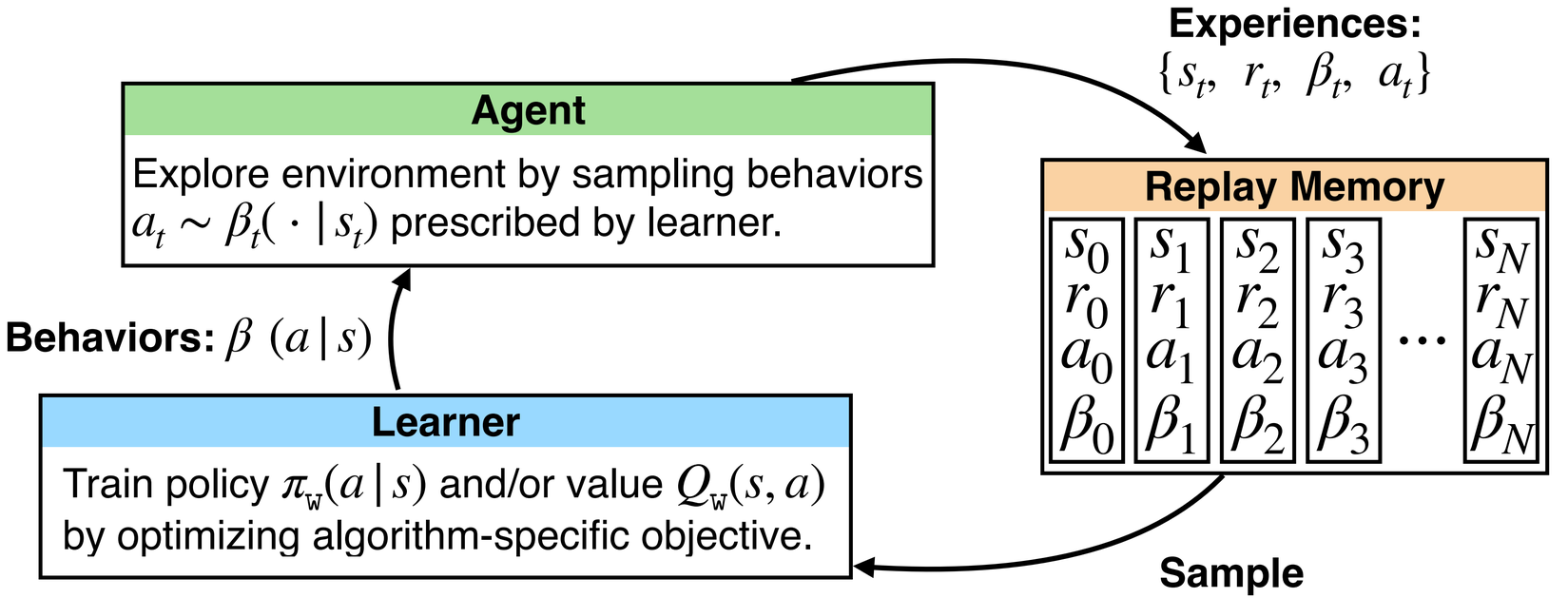
- Off-policy **training data** for policy π_w.
- **Collected by behaviors** β



For example:

- Re-use experiences collected by previous iterations of $\pi_{w^{(i)}}$
- Learn about optimal policy while collecting data with exploration strategies (e.g Q-learning).
- Learn by observe expert behaviors (expert demonstrations by humans/other agents).
- Learn from dataset of experiences without having access to environment (e.g. safety critical).
- Learn multiple policies at once, each optimal w.r.t. a specific reward function.

Experience replay



- Typically, RM contains N steps with first-in first-out elimination.
- Experiences may be organized in episodes, or individual steps
- Time step counter t tracks total number of actions performed in environment.
- Counter t uniquely identifies the tuple $\{s_t,\ r_t,\ \beta_t,\ a_t\}$
- Let gradient counter k track number of policy updates.
- Many algorithms: “Add F actions to RM per policy update” (typically $F = 1$)

Q-learning for NN: Deep Q-Networks (DQN)

DQN

Initialize $\mathtt{w}^{(0)}$ and target weights $\bar{\mathtt{w}}$
for iteration t in 1, 2, ..., N

- Advance one environment step with ϵ-greedy policy β_t
- Store experience $\{s_t,\ r_t,\ \beta_t,\ a_t\}$ in RM
- Sample a mini-batch of B experiences:

$$\{s_i,\ r_i,\ \beta_i,\ a_i\}_{i=1:B}$$

- With successor states and reward: $\{s'_i,\ r'_i\}_{i=1:B}$
- Set Q-learning target for each i:

$$\hat{q}_i = r'_i + \gamma \max_{a'} Q_{\bar{\mathtt{w}}}(s'_i, a')$$

- Update $\mathtt{w}^{(t)}$ with SGD by minimizing loss:

$$\hat{\mathscr{L}}^{MSE}(\mathtt{w}^{(i)}) = \frac{1}{B}\sum_{i=1}^{B}\left[\frac{1}{2}\left[\hat{q}_i - Q_{\mathtt{w}^{(i)}}(s_i, a_i)\right]^2\right]$$

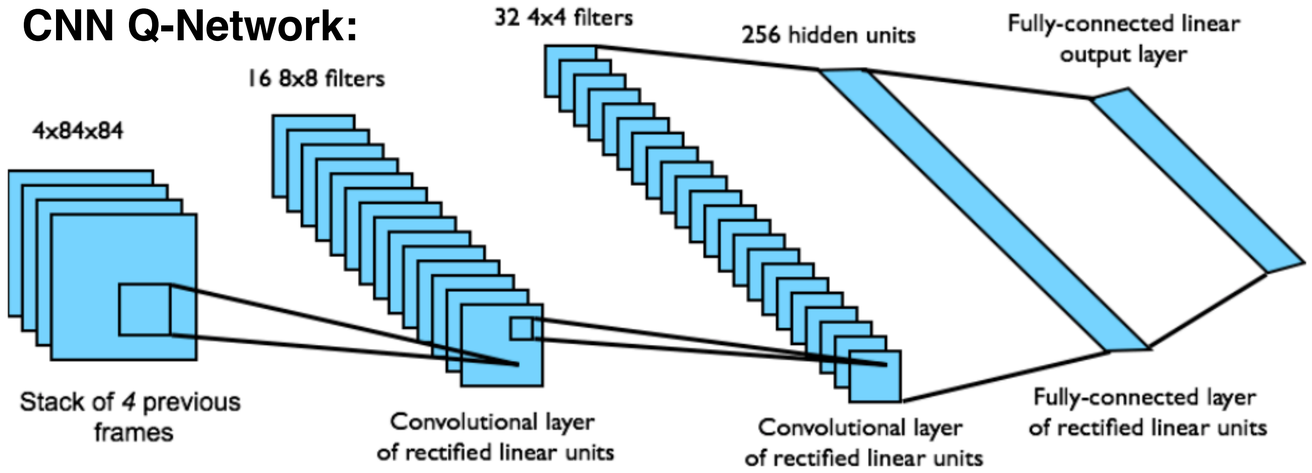
- Every D steps update target weights: $\bar{\mathtt{w}} \leftarrow \mathtt{w}^{(t)}$

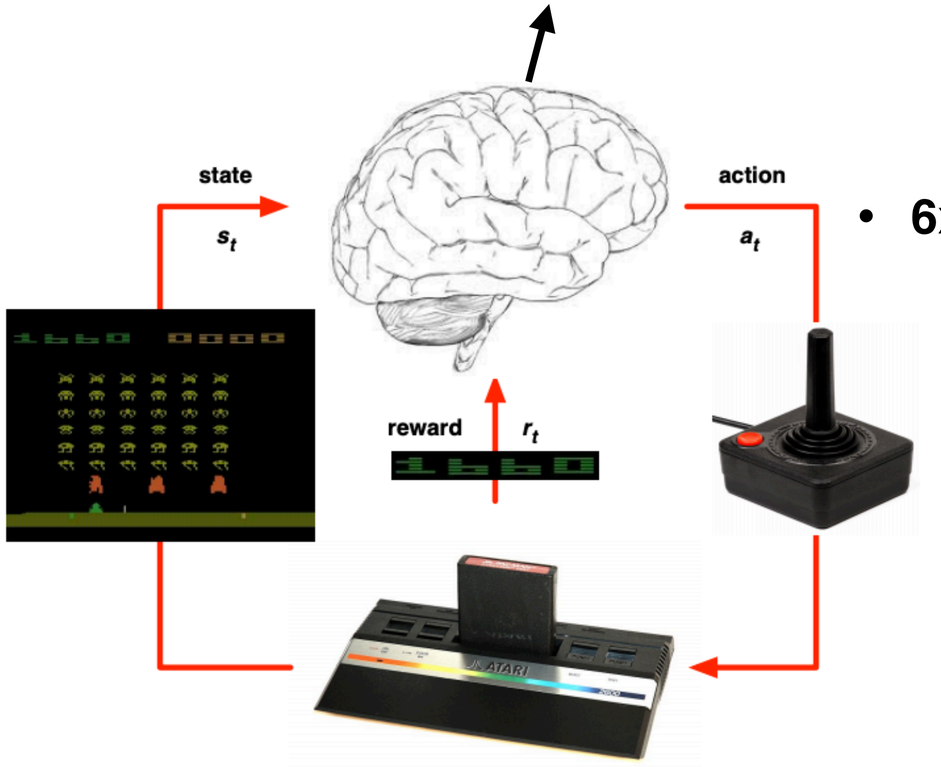
- Fixed-point iteration convergence of Q-learning unstable with deep learning.
- Target-weights are a workaround to stabilize Q-learning. Often unsuccessfully*.
- Target-weights $\bar{\mathtt{w}}$ make DQN update similar to loss minimization of supervised learning.
- Behavior β_t is ϵ_t-greedy policy
- Exploration rate ϵ_t annealed during training (from 1 to 0.05).

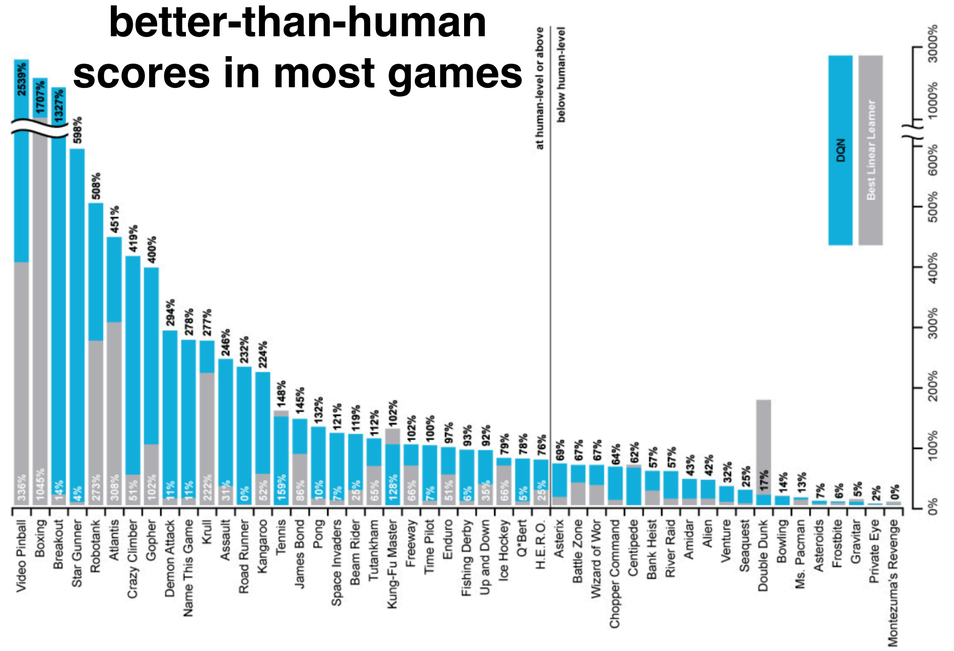
DQN : Mnih et al. 2013

*Double DQN : H van Hasselt et al. 2015

DQN for Atari



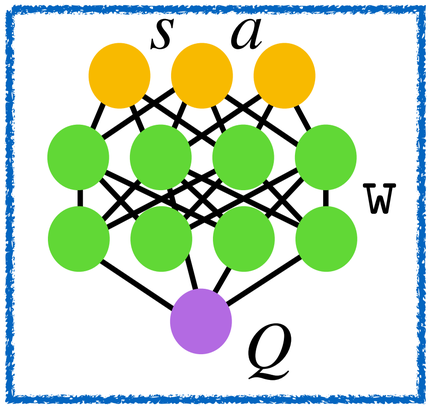




- **6x** performance improvements over the years:
  - Multi-step Q-target (Sutton & Barto, 1998)
  - Prioritized sampling for experience replay (Schaul et al, 2015)
  - Double Q-learning (van Hasselt et al, 2015)
  - Architecture: Dueling (Wang et al., 2016), Noisy (Fortunato et al. 2017)
  - Policy gradients instead of Q-learning (ACER, Wang et al., 2017)

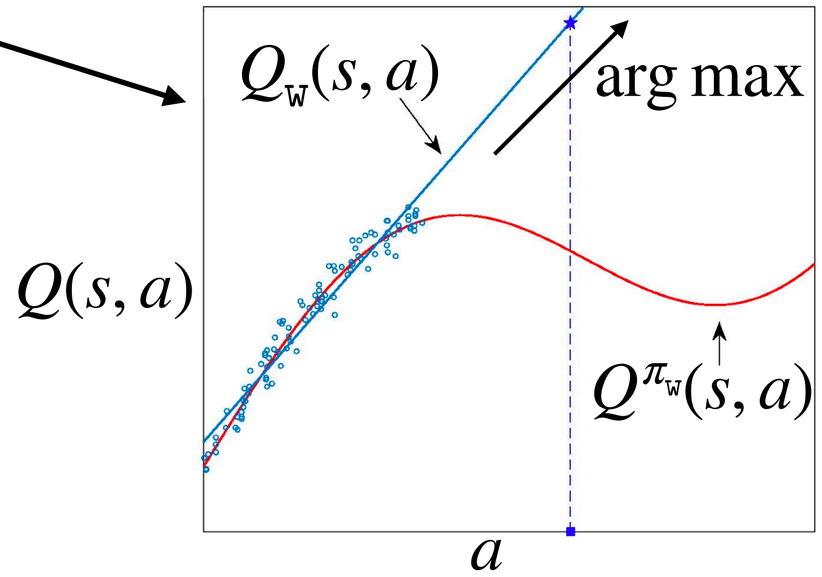
Q-learning for continuous actions

- Continuous-valued action vectors prevalent in robotics / PDEs / ODEs (even modern video-games!)
- Both s and a are vectors: natural choice for Q-network seems



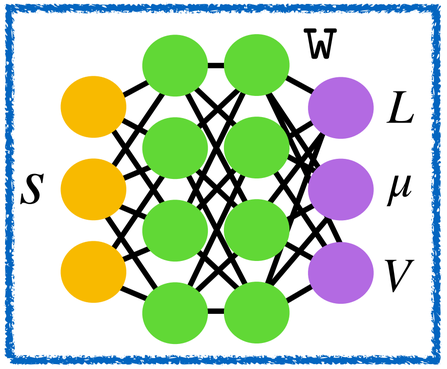
- **But:**
  - Expensive to find maxima of a NN (policy)
  - How to ensure Q_w is good approximation in all action space?
  - If linear approximation: $\arg\max_a Q_w(s,a) \to \infty$

How can we, for a given s, output a well-behaved approximation of Q_w over the entire action space? (like DQN)



Continuous Q-learning: Normalized Advantage Functions

- NAF network architecture:



NAF

Initialize $\mathtt{w}^{(0)}$

for iteration t in 1, 2, ..., N

- Collect D episodes with $\beta(\cdot \mid s) = \mathcal{N}\left(\mu_{\mathtt{w}^{(i)}}(s),\, \sigma^2\right)$
- For each step, compute target with Bellman Eq:

$$\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} = r_{t+1} + \gamma V_{\mathtt{w}^{(i)}}(s_{t+1})$$

- Update $\mathtt{w}^{(t)}$ with SGD by minimizing loss:

$$\mathcal{L}^{MSE}(\mathtt{w}^{(i)}) = \frac{1}{N}\sum_{t=0}^{N}\left[\frac{1}{2}\left[\hat{q}_t^{\pi_{\mathtt{w}^{(i)}}} - Q_{\mathtt{w}^{(i)}}(s_t, a_t)\right]^2\right]$$

- Output $V_{\mathtt{w}}(s)$ approximate state value
- $\mu_{\mathtt{w}}(s)$ vector of size $dim(A)$
- $L_{\mathtt{w}}(s)$ lower triangular matrix*:

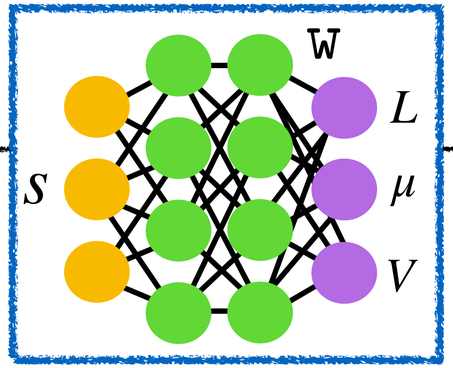
$$L_{\mathtt{w}}(s) = \begin{bmatrix} l_{\mathtt{w}}^{(1,1)} & 0 & 0 & \cdots & 0 \\ l_{\mathtt{w}}^{(2,1)} & l_{\mathtt{w}}^{(2,2)} & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{\mathtt{w}}^{(d_A,1)} & l_{\mathtt{w}}^{(d_A,2)} & l_{\mathtt{w}}^{(d_A,3)} & \cdots & l_{\mathtt{w}}^{(d_A,d_A)} \end{bmatrix}$$

- $L_{\mathtt{w}}(s)L_{\mathtt{w}}^T(s)$ is positive definite!
- $Q_{\mathtt{w}}(s,a) = V_{\mathtt{w}} - \frac{1}{2}(a-\mu_{\mathtt{w}})^T L_{\mathtt{w}} L_{\mathtt{w}}^T (a-\mu_{\mathtt{w}})$
- $\mu_{\mathtt{w}}(s)$ is action with optimal value $V_{\mathtt{w}}(s)$!

* To ease training (bijection) elements on the diagonal should be pos. def. $l_{\mathtt{w}}^{(i,i)} \leftarrow \log\left[1 + \exp\left(l_{\mathtt{w}}^{(i,i)}\right)\right]$

NAF : Gu et al. 2016

Continuous Q-Learning: Normalized Advantage Functions



NAF

Initialize $\mathbb{w}^{(0)}$ and target weights $\bar{\mathbb{w}}$
for iteration t in 1, 2, ..., N

- Advance one step with $\beta_t(a \mid s_t) = \mathcal{N}\left(\mu_{\mathbb{w}^{(i)}}(s_t),\ \sigma_t^2\right)$
- Store experience $\{s_t,\ r_t,\ \beta_t,\ a_t\}$ in RM
- Sample a mini-batch of B experiences:

$$\{s_i,\ r_i,\ \beta_i,\ a_i\}_{i=1:B}$$

- With following states and reward: $\{s'_i,\ r'_i\}_{i=1:B}$
- Set Q-learning target for each i:

$$\hat{q}_i = r'_i + \gamma V_{\bar{\mathbb{w}}}(s'_i)$$

- Update $\mathbb{w}^{(t)}$ with SGD by minimizing loss:

$$\hat{\mathscr{L}}^{MSE}(\mathbb{w}^{(i)}) = \frac{1}{B}\sum_{i=1}^{B}\left[\frac{1}{2}\left[\hat{q}_i - Q_{\mathbb{w}^{(i)}}(s_i, a_i)\right]^2\right]$$

- Every D steps update target weights: $\bar{\mathbb{w}} \leftarrow \mathbb{w}^{(t)}$

- NAF contribution:

- Output $V_{\mathbb{w}}(s)$ approximate state value
- $\mu_{\mathbb{w}}(s)$ vector of size $dim(A)$
- $L_{\mathbb{w}}(s)$ lower triangular matrix*:

$$L_{\mathbb{w}}(s) = \begin{bmatrix} l_{\mathbb{w}}^{(1,1)} & 0 & 0 & \cdots & 0 \\ l_{\mathbb{w}}^{(2,1)} & l_{\mathbb{w}}^{(2,2)} & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ l_{\mathbb{w}}^{(d_A,1)} & l_{\mathbb{w}}^{(d_A,2)} & l_{\mathbb{w}}^{(d_A,3)} & \cdots & l_{\mathbb{w}}^{(d_A,d_A)} \end{bmatrix}$$

- $L_{\mathbb{w}}(s)L_{\mathbb{w}}^T(s)$ is positive definite!
- $Q_{\mathbb{w}}(s,a) = V_{\mathbb{w}} - \frac{1}{2}(a-\mu_{\mathbb{w}})^T L_{\mathbb{w}} L_{\mathbb{w}}^T (a-\mu_{\mathbb{w}})$
- $\mu_{\mathbb{w}}(s)$ is action with optimal value $V_{\mathbb{w}}(s)$!

NAF : Gu et al. 2016

Importance sampling for off-policy rewards

Perform "roll-outs" with behavior β:

$$
\begin{aligned}
s_0^{(i)} &\sim D_0(\cdot) \\
a_0^{(i)} &\sim \beta(\cdot \mid s_0^{(i)}) \\
s_1^{(i)},\, r_1^{(i)} &\sim D(\cdot \mid s_0^{(i)},\, a_0^{(i)}) \\
&\cdots \\
a_{T-1}^{(i)} &\sim \beta(\cdot \mid s_{T-1}^{(i)}) \\
s_T^{(i)},\, r_T^{(i)} &\sim D(\cdot \mid s_{T-1}^{(i)},\, a_{T-1}^{(i)})
\end{aligned}
$$

Off-policy Episode: $\tau^{\beta} = \left\{ s_t, \beta_t, a_t, r_t \right\}_{t=0,\cdots,T}$

Off-policy MC returns: $\hat{q}_t^{\beta} = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1}^{(i)}$

TD error: $\delta_t^{\beta} = r_{t+1} + \gamma Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right) - Q_{\mathtt{w}}\left(s_t, a_t\right)$

MC target: $\hat{q}_t^{\pi} = Q_{\mathtt{w}}\left(s_t, a_t\right) + \sum_{t'=t}^{T-1} \gamma^{t'-t} \delta_{t'}^{\pi}$

Importance sampling for off-policy SARSA:

$$\hat{q}_t^{\pi} = r_{t+1} + \gamma \frac{\pi(a_{t+1} \mid s_{t+1})}{\beta(a_{t+1} \mid s_{t+1})} Q_{\mathtt{w}}(s_{t+1}, a_{t+1})$$

- Relatively low variance but biased
- Unbounded importance weights may cause numerical instability

Importance sampling for off-policy MC returns:

$$\hat{q}_t^{\pi} = \frac{\pi(a_{t+1} \mid s_{t+1})}{\beta(a_{t+1} \mid s_{t+1})} \cdot \cdots \cdot \frac{\pi(a_{T-1} \mid s_{T-1})}{\beta(a_{T-1} \mid s_{T-1})} \cdot \hat{q}_t^{\beta}$$

- Dramatic increase in variance
- Unstable due to exploding product
- Is there a lower variance expression?

Importance sampling for off-policy TD errors

- Given off-policy episode: $\tau^{\beta} = \{s_t, \beta_t, a_t, r_t\}_{t=0,\cdots,T}$
- Introduce:

$$\delta_t^{\beta} = r_{t+1} + \gamma Q_{\mathtt{w}}(s_{t+1}, a_{t+1}) - Q_{\mathtt{w}}(s_t, a_t)$$

$$\delta_t^{\pi} = r_{t+1} + \gamma \mathop{\mathbb{E}}_{a' \sim \pi} Q_{\mathtt{w}}(s_{t+1}, a') - Q_{\mathtt{w}}(s_t, a_t)$$

- E.g. $\delta_t^{\pi} = r_{t+1} + \gamma \max_{a' \sim \pi} Q_{\mathtt{w}}(s_{t+1}, a') - Q_{\mathtt{w}}(s_t, a_t)$
- E.g. $\delta_t^{\pi} = r_{t+1} + \gamma V_{\mathtt{w}}(s_{t+1}) - Q_{\mathtt{w}}(s_t, a_t)$

General operator to estimate returns from off-policy data:

$$\hat{q}_t^{\pi} = Q_{\mathtt{w}}(s_t, a_t) + \sum_{t'=t}^{T-1} \gamma^{t'-t} \delta_{t'}^{\pi} \left(\prod_{s=t+1}^{t'} c_s \right)$$

- Where we define $\prod_{s=t+1}^{t} c_s = 1$ (i.e. the term for $t' = t$) regardless of c_s
- If $c_s = 0$: we recover Q-learning
- If $c_s = \pi(a_s \mid s_s)/\beta(a_s \mid s_s)$: we recover importance sampling MC
- If $c_s = 1$: off-policy N-step methods (i.e. pretend it is not off-policy)
  No base in theory, but sometimes performs well and therefore used!
- If $c_s = \min\{1, \pi(a_s \mid s_s)/\beta(a_s \mid s_s)\}$ then we use **`Retrace`** (Munos *et al.* 2016)

Retrace

$$\hat{q}_t^{retrace} = Q_{\mathtt{w}}\left(s_t, a_t\right) + \sum_{t'=t}^{T-1} \gamma^{t'-t} \left(\prod_{s=t+1}^{t'} c_s\right) \left(r_{t+1} + \gamma V_{\mathtt{w}}\left(s_{t+1}\right) - Q_{\mathtt{w}}\left(s_t, a_t\right)\right)$$

- With $c_s = \min\left\{1,\ \pi(a_s \mid s_s)/\beta(a_s \mid s_s)\right\}$
- And $\prod_{s=t+1}^{t} c_s = 1$

Can we written recursively as:

$$\hat{q}_t^{retrace} = r_{t+1} + \gamma V_{\mathtt{w}}\left(s_{t+1}\right) + \gamma \min\left[1, \frac{\pi(a_{t+1} \mid s_{t+1})}{\beta(a_{t+1} \mid s_{t+1})}\right] \left[\hat{q}_{t+1}^{retrace} - Q_{\mathtt{w}}\left(s_{t+1}, a_{t+1}\right)\right]$$

- Clipping prevents variance explosion of Importance Sampling
- Safely uses all relevant off-policy rewards in the episode
- Provably converges to correct Q^{π} for **any** behavior policy β (tabular setting)
- Many state-of-the-art results support its efficacy (e.g. Wang 2016, Espeholt 2018)
- If β and π are similar, Retrace converges faster!

The on-policy objectives from off-policy data (1)

On-policy algorithms optimize objectives in the form: $\mathscr{L}(w) = \mathbb{E}\left[L_w(s,a) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\cdot) \\ a \sim \pi_w(\cdot \mid s) \end{matrix} \right]$

$a \sim \pi_w(\cdot \mid s)$

Probability of an action given the current policy

$s \sim \eta^{\pi_w}(\cdot)$

Probability of visiting a state with π_w

Off-policy algorithms compute L(w) from experiences gathered by behaviors β:

$a \sim \beta(\cdot \mid s)$

Actions sampled from arbitrary behaviors β.

$s \sim \eta^{\beta}(\cdot) \propto \sum_{k=0} P\left(s = s_k \mid s_0, a_k \sim \beta(a \mid s_k)\right)$

Probability of observing a state by acting with behaviors β.

The on-policy objectives from off-policy data (2)

On-policy algorithms optimize objectives in the form: $\mathscr{L}(w) = \mathbb{E}\left[L_w(s,a) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\,\cdot\,) \\ a \sim \pi_w(\,\cdot\,|\,s) \end{matrix} \right]$

Off-policy algorithms compute J(w) from expectation over past experiences:

$$a \sim \beta(\,\cdot\,|\,s) \qquad\qquad s \sim \eta^{\beta}(\,\cdot\,) := \sum_{k=0} P\left(s = s_k \,|\, s_0, a_k \sim \beta(a\,|\,s_k)\right)$$

example: first-order approximation of on-policy objectives from off-policy data:

approximation: assume that distribution of states almost constant

$$\mathscr{L}(w) = \int_S \int_A L_w(s,a)\ \pi_w(a\,|\,s)\,\mathrm{d}a\ \eta^{\pi_w}(s)\,\mathrm{d}s \approx \int_S \int_A L_w(s,a)\ \pi_w(a\,|\,s)\,\mathrm{d}a\ \eta^{\beta}(s)\,\mathrm{d}s = \tilde{\mathscr{L}}(w)$$

$$\tilde{\mathscr{L}}(w) = \int_S \int_A L_w(s,a)\ \pi_w(a\,|\,s)\frac{\beta(a\,|\,s)}{\beta(a\,|\,s)}\,\mathrm{d}a\ \eta^{\beta}(s)\,\mathrm{d}s = \mathbb{E}\left[L_w(s,a)\frac{\pi_w(a\,|\,s)}{\beta(a\,|\,s)} \,\middle|\, \begin{matrix} s \sim \eta^{\beta}(\,\cdot\,) \\ a \sim \beta(\,\cdot\,|\,s) \end{matrix} \right]$$

change of distribution:
expectation over old policies

importance sampling

The on-policy objectives from off-policy data (3)

Recall: the on-policy Policy Gradient: $\nabla_{\mathtt{w}} J(\mathtt{w}) = \mathbb{E}\left[\hat{q}^{\pi_w}(s,a)\ \nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\pi_w}(\,\cdot\,) \\ a \sim \pi_w(\,\cdot \mid s) \end{matrix}\right]$

Off-policy approximation of the Policy Gradient:

$$\nabla_{\mathtt{w}} J(\mathtt{w}) \approx \mathbb{E}\left[\hat{q}^{\pi_w}(s,a)\ \nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\beta}(\,\cdot\,) \\ a \sim \pi_w(\,\cdot \mid s) \end{matrix}\right]$$

$$= \mathbb{E}\left[\hat{q}^{\pi_w}(s,a)\ \frac{\beta(a \mid s)}{\beta(a \mid s)} \nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\beta}(\,\cdot\,) \\ a \sim \pi_w(\,\cdot \mid s) \end{matrix}\right]$$

$$= \mathbb{E}\left[\hat{q}^{\pi_w}(s,a)\ \frac{\pi_w(a \mid s)}{\beta(a \mid s)} \nabla_w \log \pi_w(a \mid s) \,\middle|\, \begin{matrix} s \sim \eta^{\beta}(\,\cdot\,) \\ a \sim \beta(\,\cdot \mid s) \end{matrix}\right]$$

Degris et al. 2012

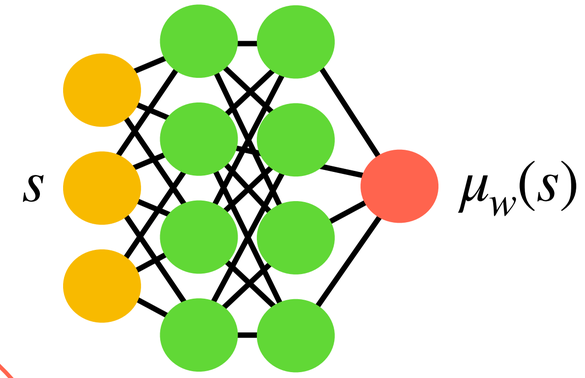
The off-policy Policy Gradient (e.g. V-RACER)

- Approximate the PG by sampling a Replay Memory and iteratively update by SGD:

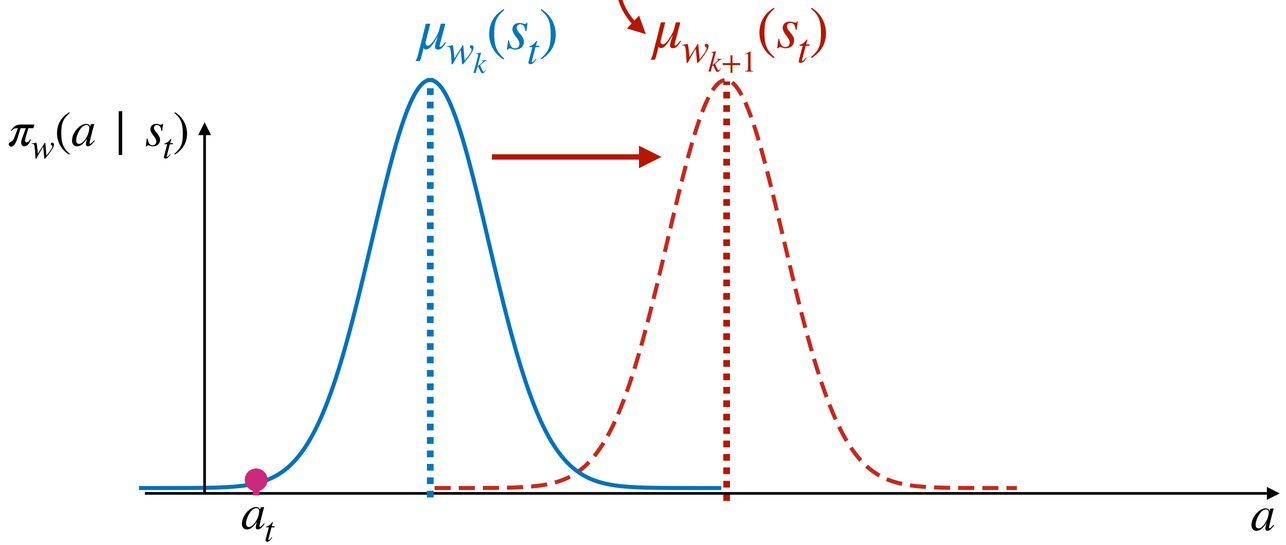
$$w_{k+1} = w_k + \epsilon\, \mathbb{E}\left[q^{\pi_w}(s,a) \cdot \frac{\pi_w(a\,|\,s)}{\beta(a\,|\,s)} \cdot \frac{a - \mu_w(s)}{\sigma^2} \nabla_w \mu_w(s) \right]$$

Monte-Carlo estimate from past experiences:

- Sample a past experience $\{s_t, \mu_t, \sigma_t, a_t, r_t\}$
- Approximate the returns $q_t^{\pi_w}$ (e.g. with network $Q_w(s_t, a_t)$ or with **Retrace** algorithm)
- **if $q^\pi > 0$: move μ_w (current weights) towards a_t**
- **if $q^\pi < 0$: move μ_w (current weights) away from a_t**



$$\beta(\,\cdot\,|\,s_t) := \mathcal{N}(\mu_t, \sigma_t^2)$$



- Policy Gradient does not say **how much μ_w should be moved away from a_t.**
- Off-policy updates can easily cause **policy to diverge from training behaviors**.

Issues of off-policy RL

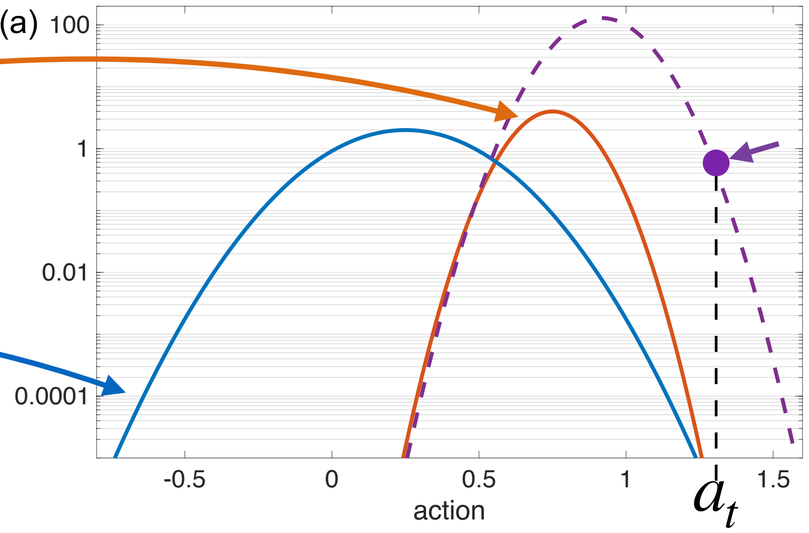
ISSUE #1

$\pi_w(a \mid s_t)$

Current policy that we aim to update

$\beta(a \mid s_t)$

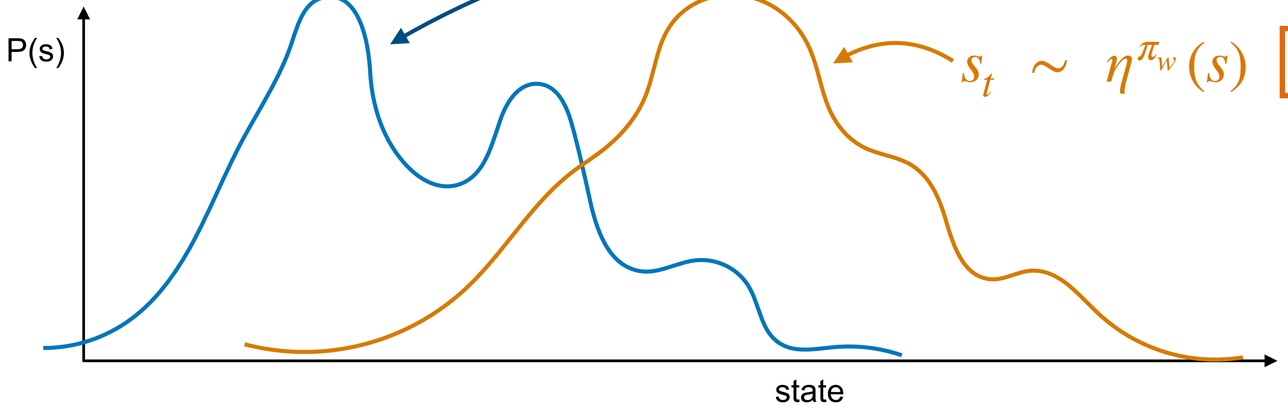
old policy followed by the agent (training data)



$$\frac{\pi_w(a_t \mid s_t)}{\beta(a_t \mid s_t)}$$

Unbounded (0 to ∞) Importance weight, increases the variance and **lowers the accuracy of the cost function** $\tilde{J}(w)$.

ISSUE #2



$s_t \sim \eta^{\beta}(s)$

Prob of encountering state with old policies (training data)

$s_t \sim \eta^{\pi_w}(s)$

Prob of encountering state with current policy

If distribution of training data is too **dissimilar from on-policy** outcomes, data may be **irrelevant to updating the policy**

Proposed solution: constrain policy changes to past policies

Remember and Forget Experience Replay

ReF-ER works with most RL methods that learn a policy by Experience Replay (e.g. Q-learning, off-policy PG, ...)

$$\tilde{\mathscr{L}}(w) = \mathbb{E}\left[\tilde{L}_w(s,a) \,\middle|\, \begin{matrix} s \sim \eta^{\beta}(\cdot) \\ a \sim \beta(\cdot\,|\,s) \end{matrix} \right]$$

algorithm-specific

Modify objective of the RL algorithm:

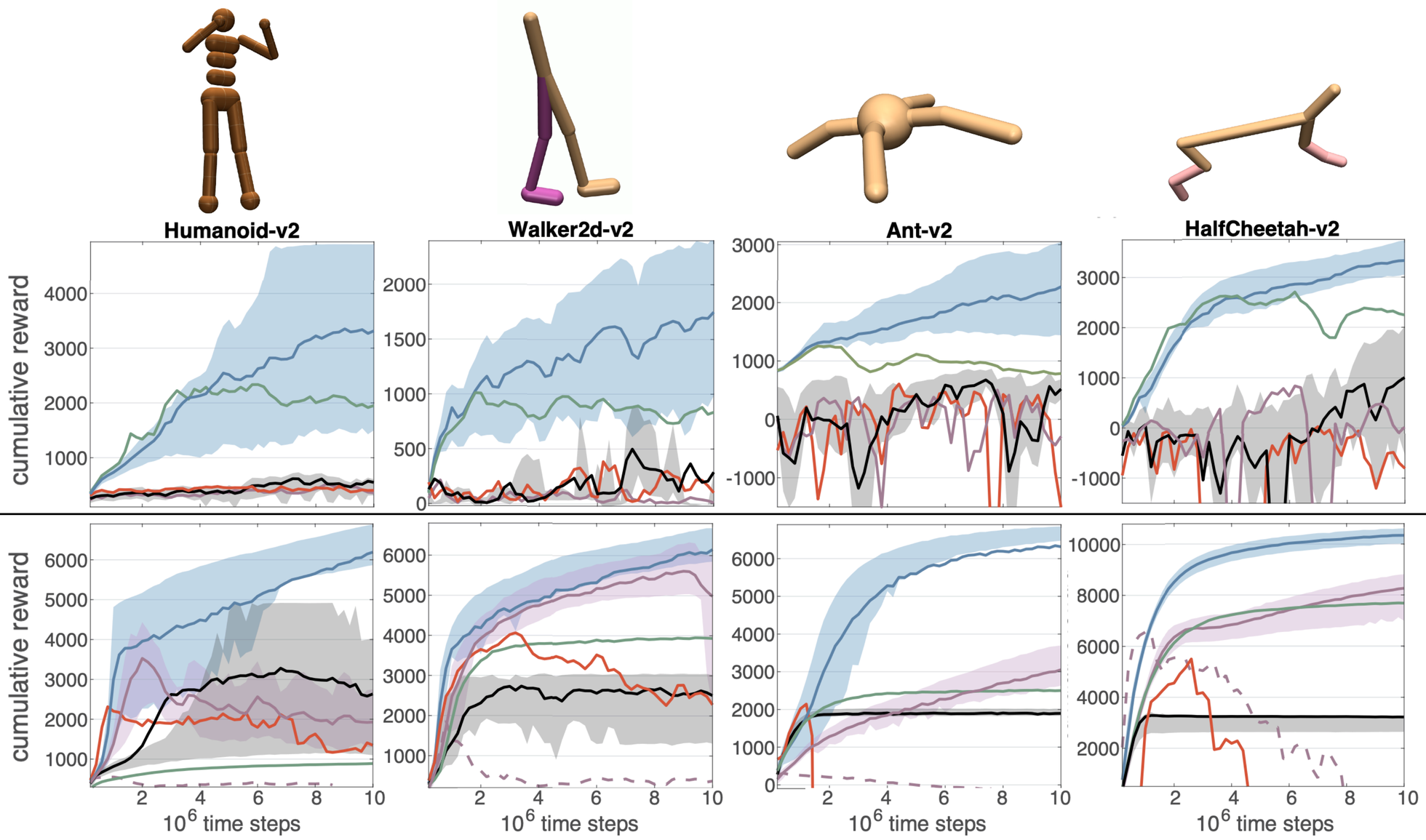
$$\nabla_{\mathtt{w}} \tilde{L}_w^{ReF-ER}(s,a) = \begin{cases} \lambda\ \nabla_{\mathtt{w}} \tilde{L}_w(s,a) \ - \ (1-\lambda)\nabla_{\mathtt{w}} D_{\mathrm{KL}}\left[\pi_w(\,|\,s) \| \beta(\,|\,s)\right] & \text{if } \frac{1}{c} < \frac{\pi_w(a\,|\,s)}{\beta(a\,|\,s)} < c \\ 0 \ - \ (1-\lambda)\nabla_{\mathtt{w}} D_{\mathrm{KL}}\left[\pi_w(\,|\,s) \| \beta(\,|\,s)\right] & \text{otherwise} \end{cases}$$

1) **Reject samples from objective estimate if importance weight π_w/β lies outside of a trust region.**

2) **Add penalization cost to attract policy back towards training behaviors.**
(Coefficient λ is iteratively updated to keep a fixed fraction of the Experience Replay data within the trust region.)

(Novati & Koumoutsakos, ICML '19 - Remember and Forget for Experience Replay)

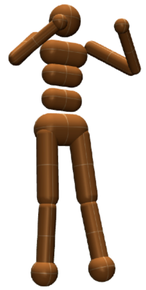
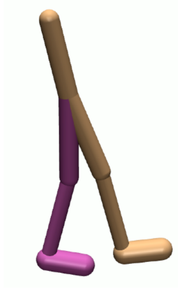
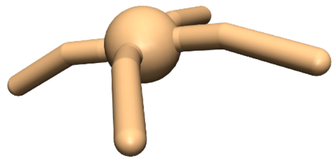
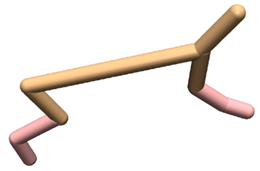
Results on OpenAI gym MuJoCo tasks (1)

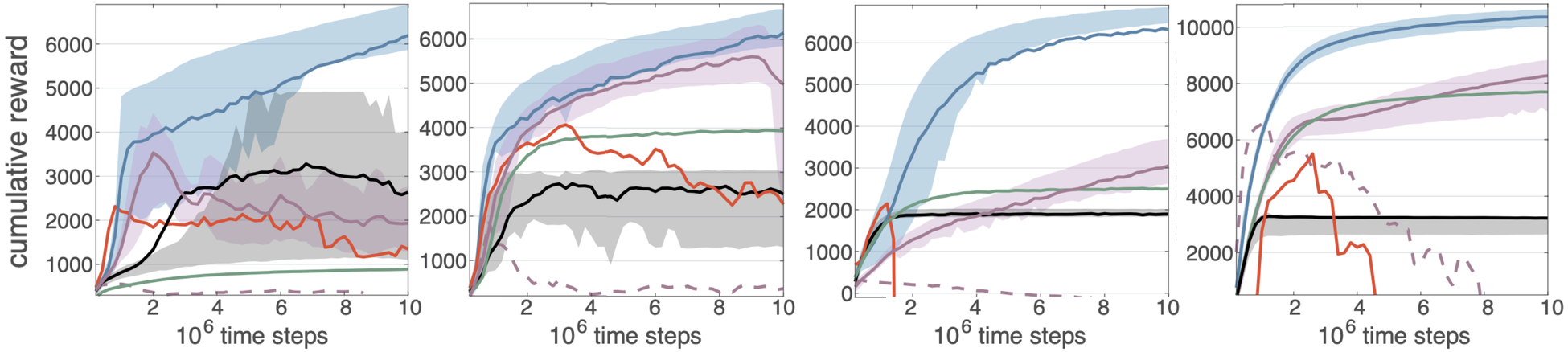


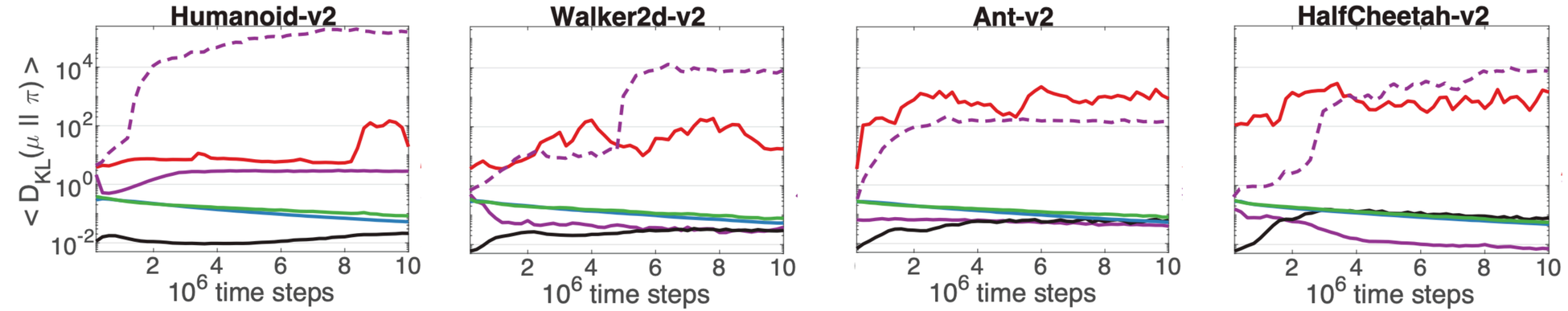
NAF + ReF-ER
NAF + ReF-ER Rule 1 only
NAF + ReF-ER Rule 2 only
NAF + Experience Replay
(Gu *et al.* 2016)
NAF + Prioritized ER
(Schaul *et al.*, 2016)

RACER + ReF-ER
RACER + ReF-ER Rule 1 only
RACER + ReF-ER Rule 2 only
PPO (Constrained On-policy PG)
(Schulman *et al.* 2017)
ACER (Off-policy PG + remedies against instability)
(Wang *et al.*, 2016)

Results on OpenAI gym MuJoCo tasks (2)





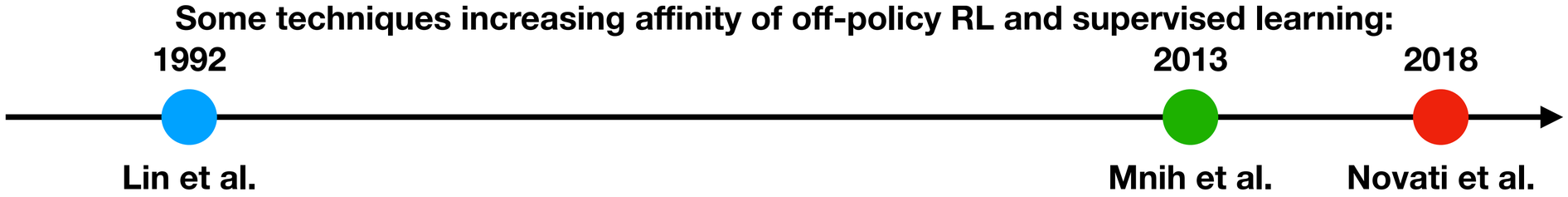
RACER + ReF-ER
RACER + ReF-ER Rule 1 only
RACER + ReF-ER Rule 2 only
PPO (Constrained On-policy PG)
(Schulman *et al.* 2017)
ACER (Off-policy PG + remedies against instability)
(Wang *et al.*, 2016)

ACER relies on target networks (like DQN) and hyper-parameter optimization.
Increasing learning rate breaks stability measures (**dashed line**).

ReF-ER effectively **constrained D_{KL}**, increased stability & performance.

Off-Pol RL as Supervised Learning

- Years of research in ML gave us reliable techniques to train NN on supervised learning tasks



Experience Replay: past experiences as training data.

- Allows RL training with **mini-batches of uncorrelated samples** (closer to i.i.d. assumption)
- Distribution of **training data is almost constant** during training

DQN (Atari-paper) introduces Target Networks for Q-learning.

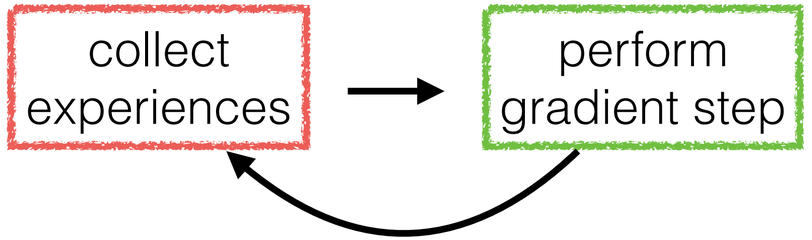
- **Target of Q-learning update** is no longer fixed point iteration, but **almost constant**.

ReF-ER controls RL updates and training data.

- Constrains similarity between **distribution** of training data and on-policy data.
- In other words: force **test set and training set to be from almost the same distribution**

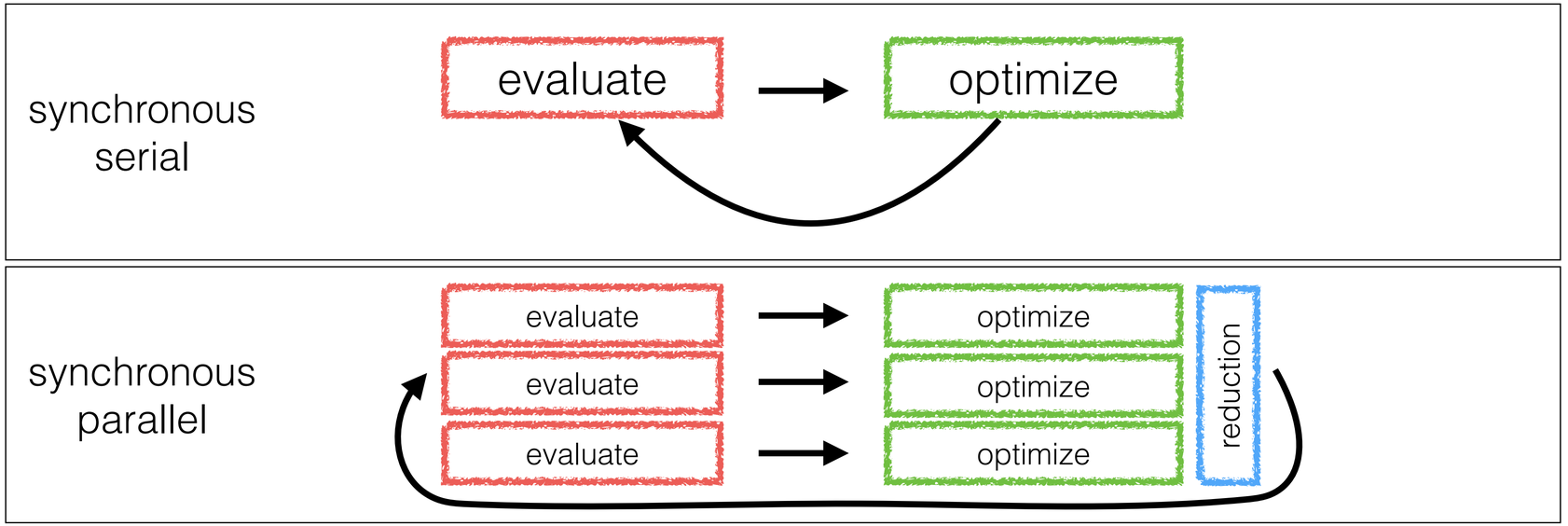
RL software architecture

- Most academic RL software is written in python (using torch or tensorflow for NN)
- Often assumes toy control problems: environment is cheap to simulate



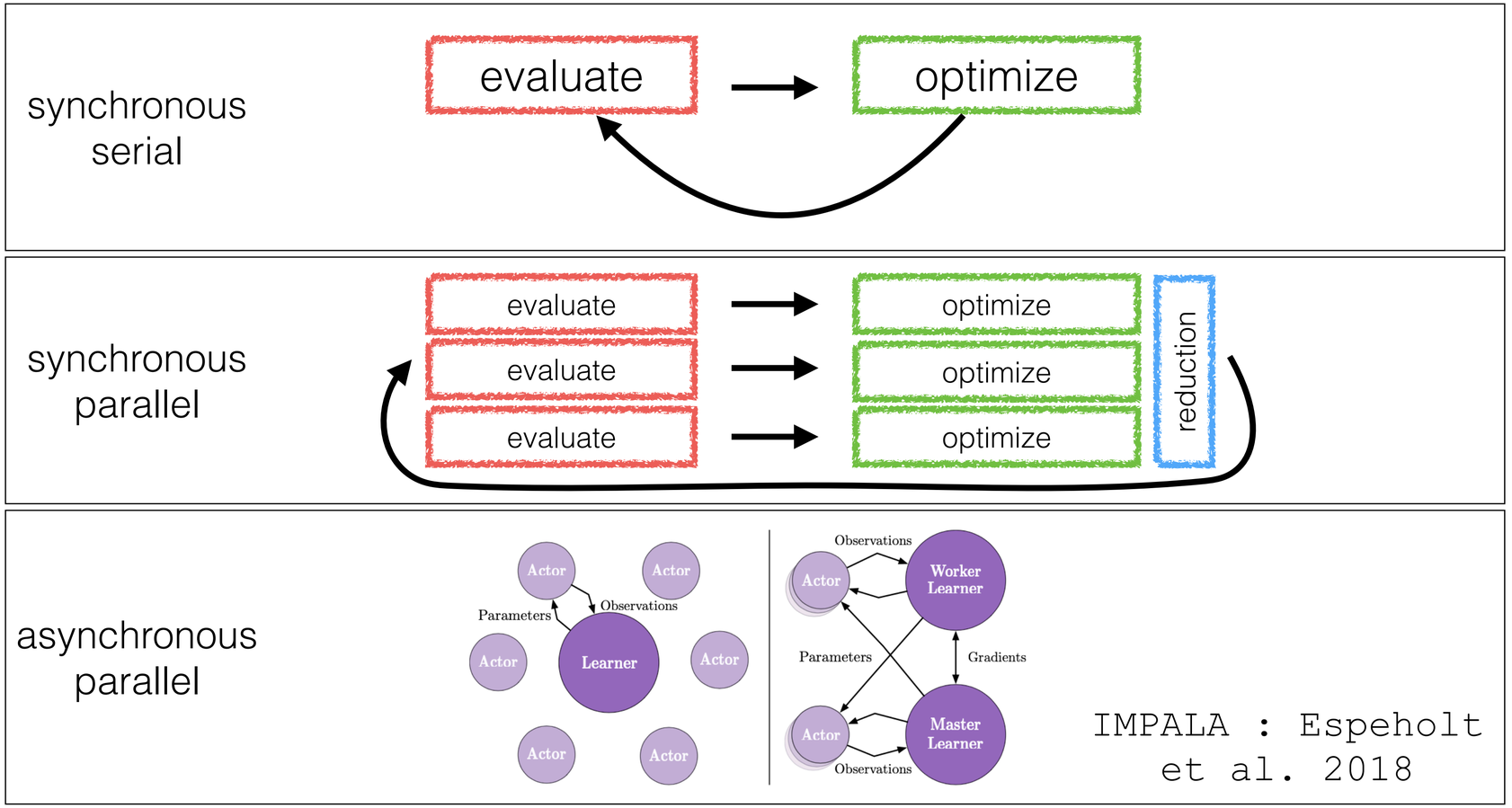
- E.g. Mnih et al. 2015 (Atari Nature paper) followed this approach
- Also most *towardsdatascience.com* tutorials
- Synchronous software design
- NN on GPU, alternates evaluation and optimization
- Toy environments are typically single threaded and CPU bound
- Environment often ends up being the bottleneck (Amdahl's law)

RL software architecture



- Second round (2016/17) of deep RL papers employed CPU-bound networks
- One “worker” per CPU core, each core runs its own environment
- Number of CPU cores became an hyper parameter in RL papers
- Performance can be bad:
    - Large load imbalances (unevenly sized episodes)
    - Many serial operations are performed

RL software architecture

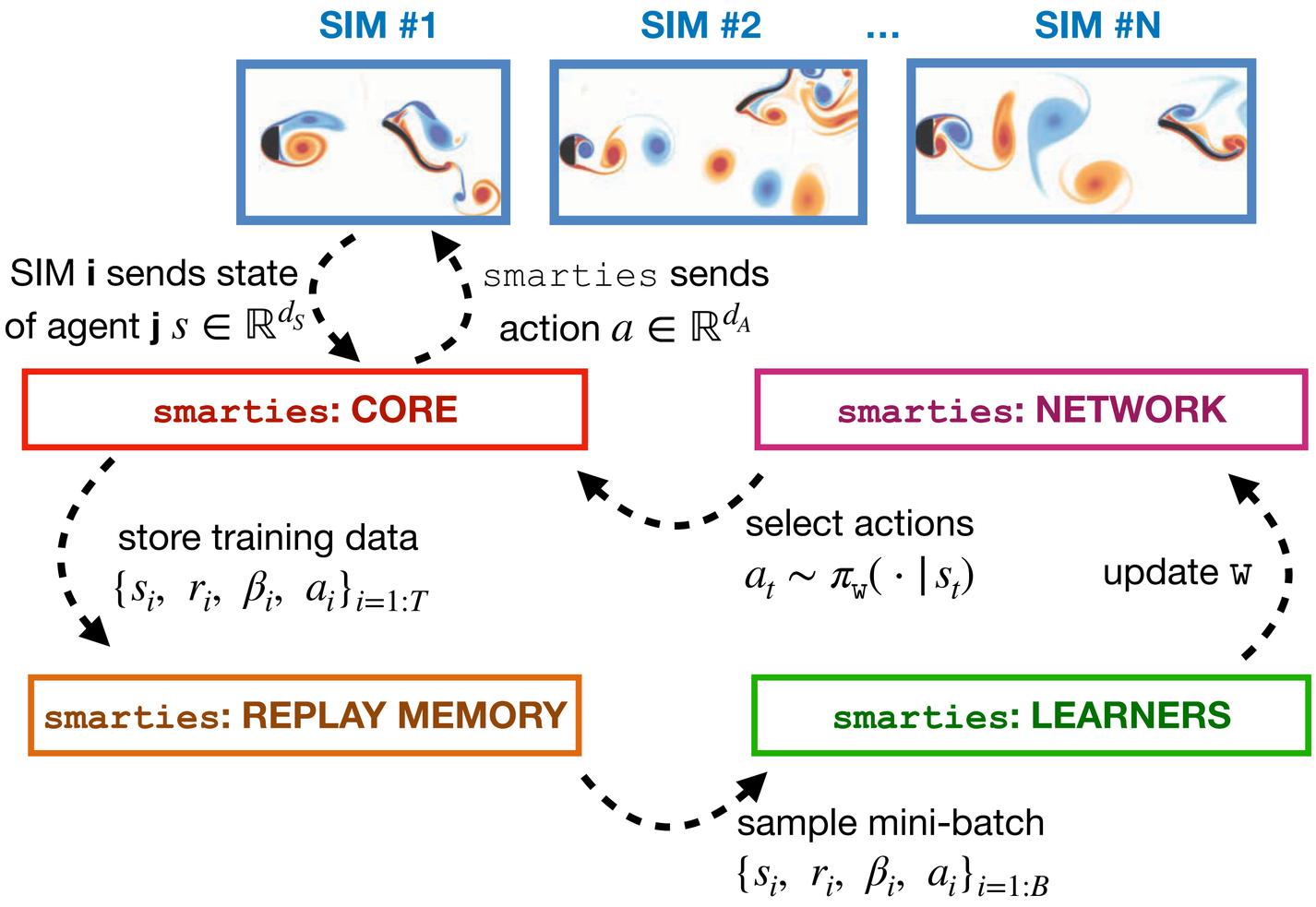


- Third round (2018/19) GPUs are back, price is much more complex software
- ‘Worker’ processes and ‘Learner’ processes, parameter servers…
- Computational resources are hyper parameters (N of workers affects performance)

smarties

- Objectives:
  - Minimally intrusive integration with existing simulation software: python, C++, fortran.
    - OpenAI gym is designed to be used for RL
    - But other environments may not be designed around RL
  - C++, asynchronous and thread-parallel:
    - Latency due to many cheap function calls is reduced
    - Algorithms can simply "*update NN by sampling RM and perform N actions*" without ruining performance: done in parallel
    - Data processing is easily parallelized.
  - Supports multiple learners, multiple workers, simulations that run on multiple CPUs…
  - Supports multiple agents with separate policies, different state/action sizes…
  - Uses RNN for partially observable problems without changing the algorithm
  - Environment communication API is thread-safe
  - One real dependency: MPI (portability?).

smarties : high level view



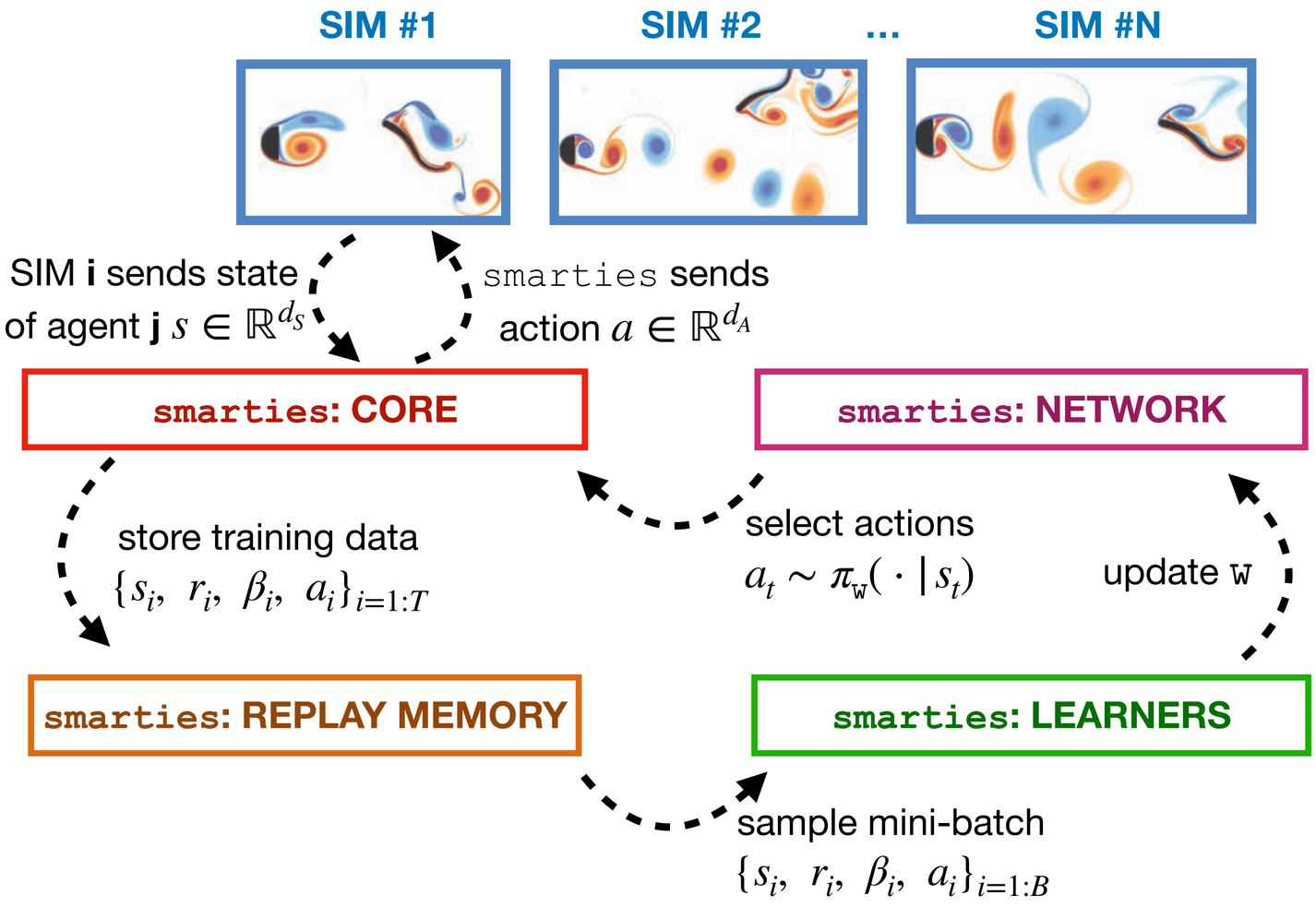

```python
import smarties as rl

def app_main(comm):
  comm.set_state_action_dims(state_dim, action_dim)
  env = Environment()
  while 1: #train loop
    env.reset()
    comm.sendInitState(env.getState())
    while 1: #simulation loop
      action = comm.recvAction()
      isTerminal = env.advance(action)
      if isTerminal:
        comm.sendTermState(env.getState(), env.getReward())
        break
      else: # normal state
        comm.sendState(env.getState(), env.getReward())

if __name__ == '__main__':
  e = rl.Engine(sys.argv)
  if( e.parse() ): exit()
  e.run( app_main )
```

https://github.com/cselab/smarties

`smarties` : the sub-components

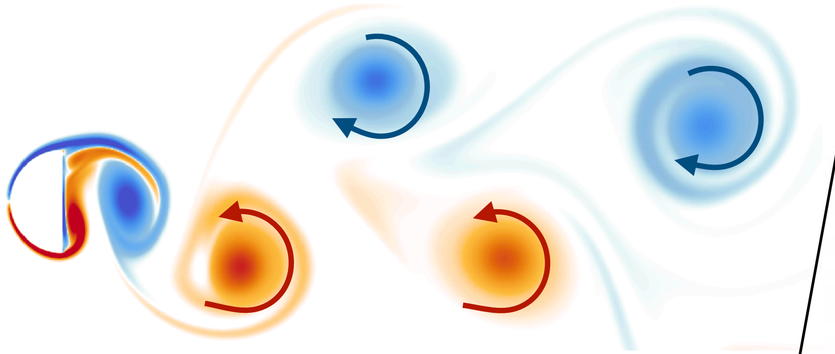


- **CORE**:
  - Common definitions: MDP, agent, environment
  - Schedules communication between workers and learners
  - Learners that send actions and recv states
  - Learners that send parameters and recv episodes
- **REPLAY MEMORY**:
  - Store, share, sample, post-process experiences
- **LEARNERS**:
  - RACER, DQN, NAF, PPO, CMA, DPG, ACER…
  - Each algorithm prescribes how to select actions and how to update network(s)
- **MATH**:
  - e.g. Gaussian/Bernoulli/Dirichlet policies and related operations
- **NETWORK**:
  - Defines various layer types (linear, LSTM, GRU, Conv2D) and non-linearities
  - Optimizers, memory management, …

`https://github.com/cselab/smarties`

ReF-ER and agents in unsteady flows

- Unsteady flows are ubiquitous in nature
- Many animal species learn to capture energy from vortices



Kármán vortex street

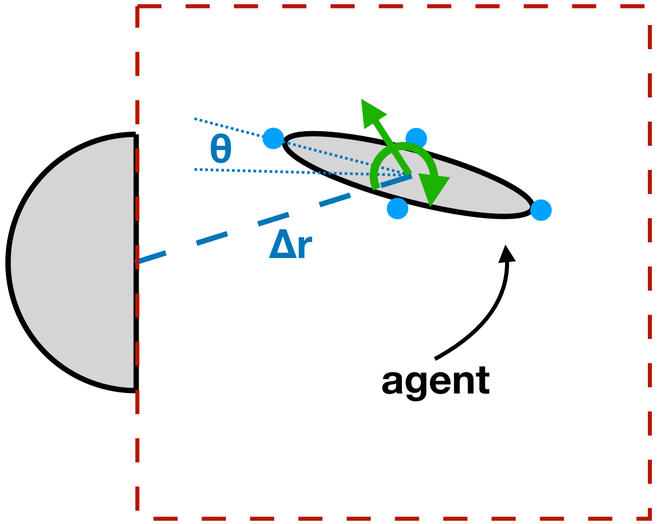
- E.g. drafting in flows behind stationary objects

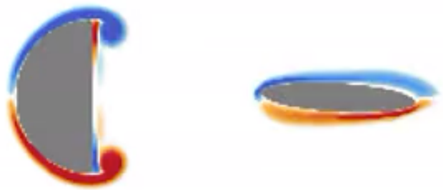
Webb, J. Exp. Bio. 1998
Hinch, Rand, J. Fish. Aquat. Sci. 2000
Liao, Beal, Lauder, Triantafyllou, J. Exp. Bio. 2003

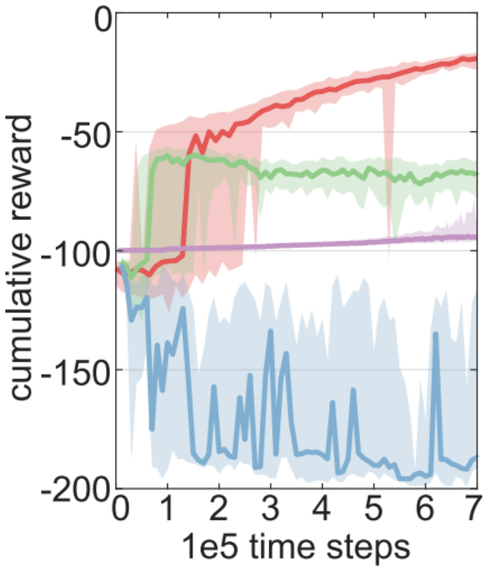
State:
- Relative position **Δr**, angle **θ**, velocities
- Flow velocity sensors (at vertices): ●

Action:
- Applied force and torque : $\vec{a} := \{f_X, f_Y, \tau\}$

Reward:
- Actuation cost: $r = -\|a\|^2$
- Terminate if reaches border: $r_T = -100$







- Uses RNNs to estimate environment from sensor data
- Legend: **RACER + ReF-ER**, **ACER**, **DPG**, **DPG + ReF-ER**
- RACER at Re = UD/v = 400, force to propel ellipse 60% lower than in quiescent flow