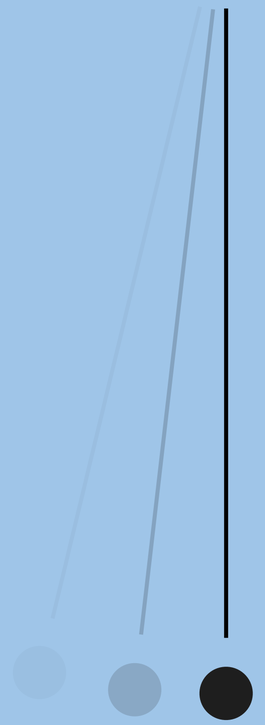
Introduction to policy search in Reinforcement Learning

Djalel Benbouzid — Data:Lab Munich, Volkswagen Group

June 27th 2018



quick outline

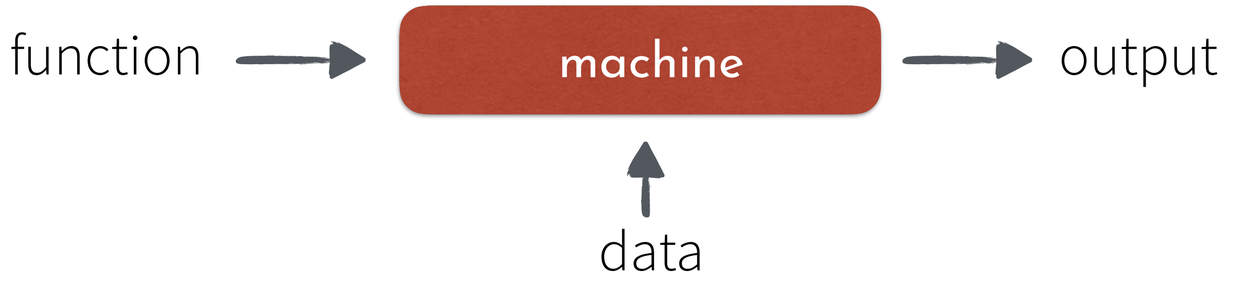
introduction and some reminders

introduce a **different way to RL**, wrt. first lecture

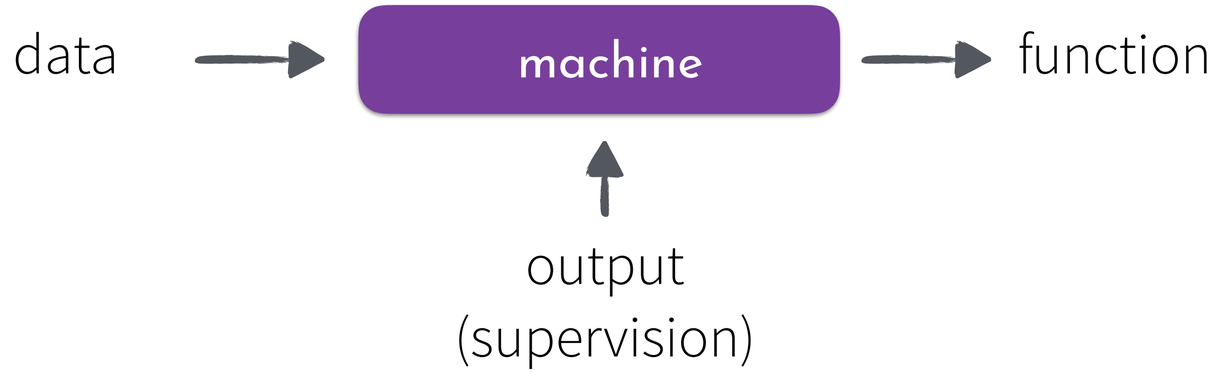
gradients-based methods

gradients-free methods

“traditional” programming

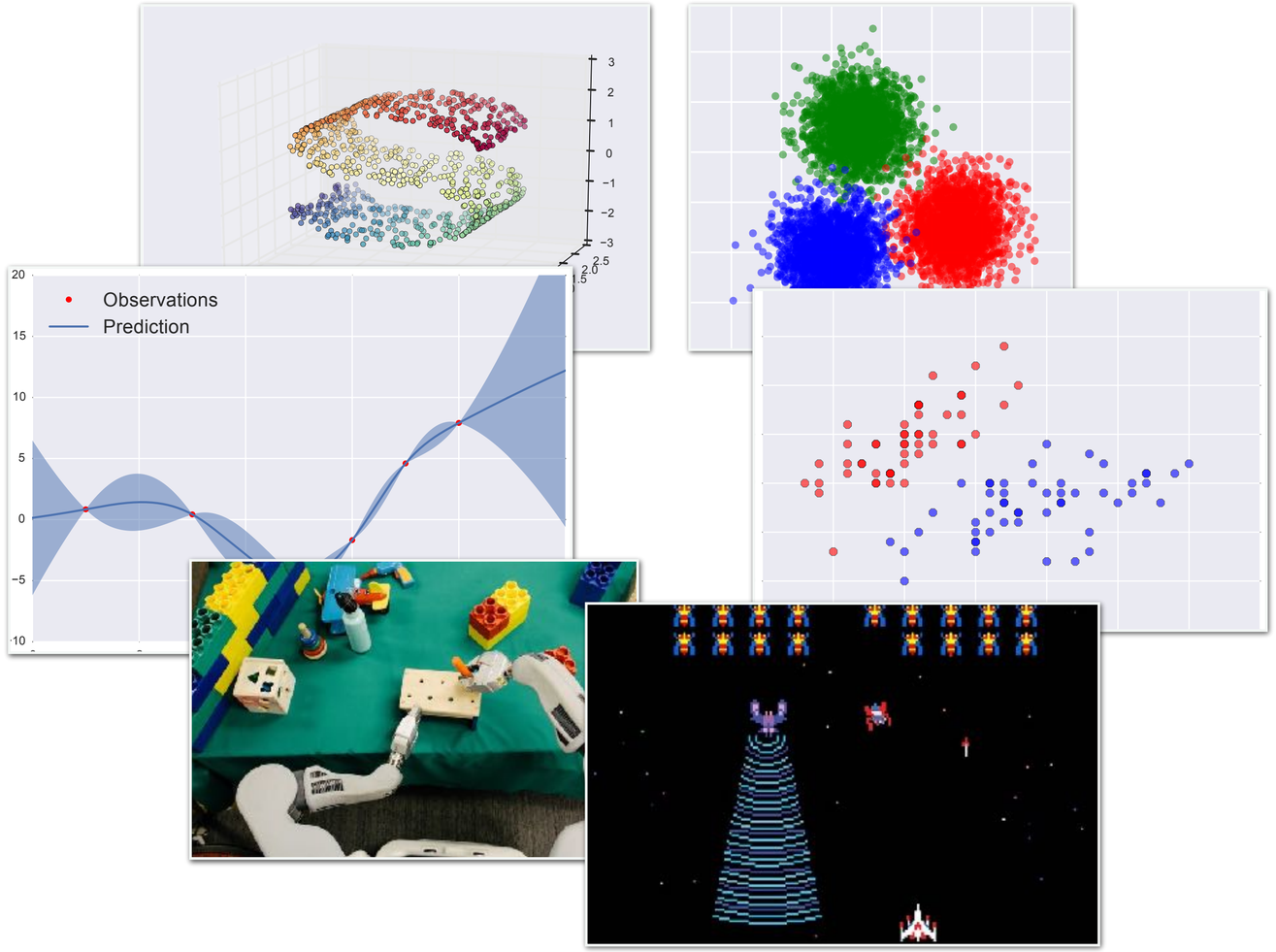


machine learning

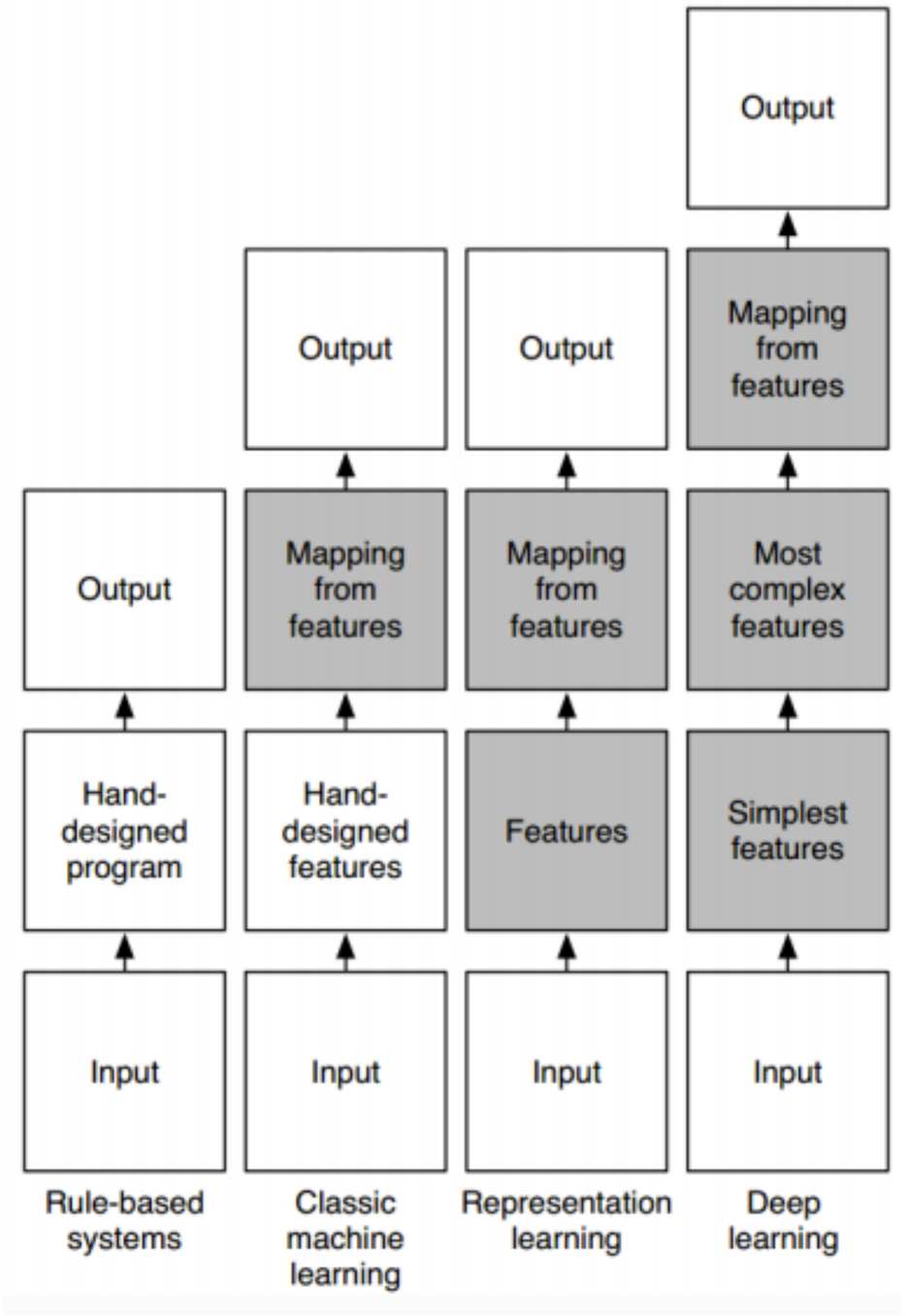


Machine Learning paradigms

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning



what about deep learning?



source: Bengio et. al

it's all about inferring functions and abstracting problems away

supervised learning:
function mapping, I/O

$$f : \mathcal{X} \to \mathcal{Y} \qquad f \sim p(y \mid \mathbf{x})$$

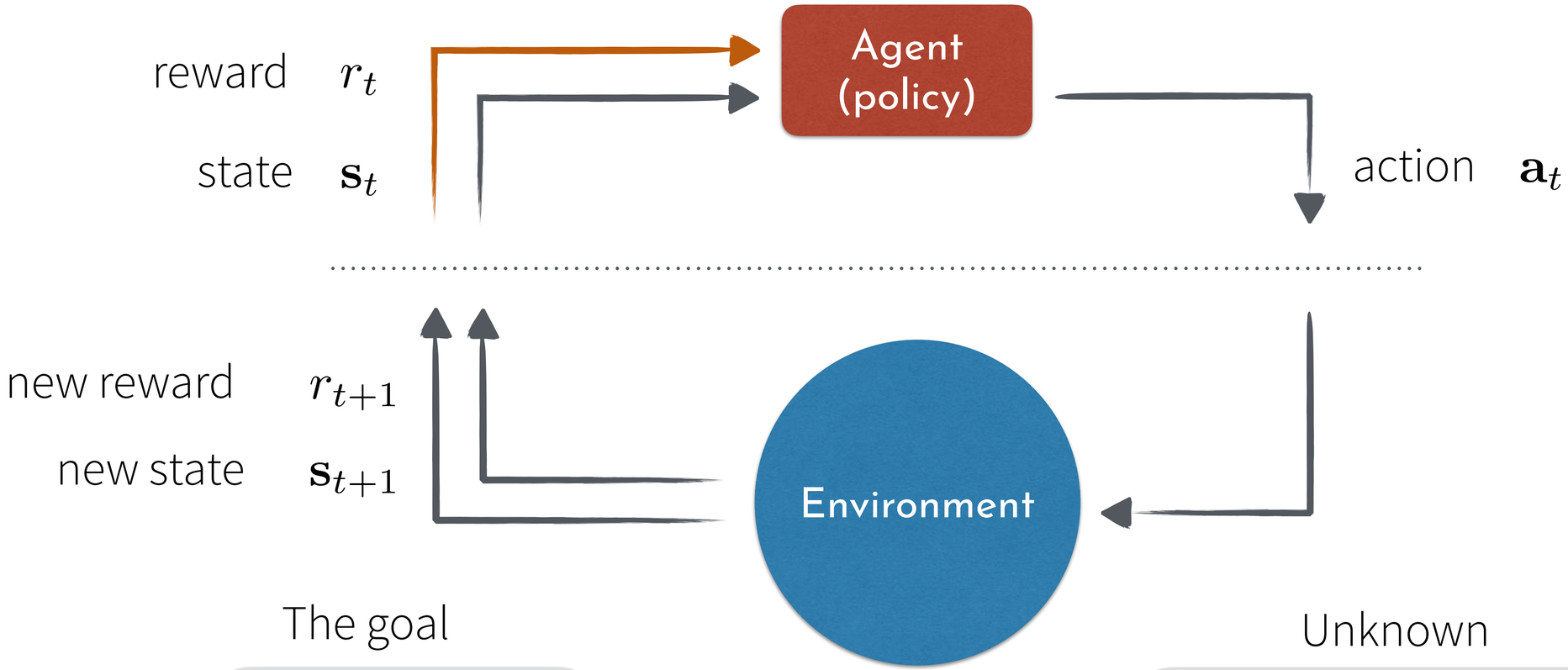
unsupervised learning:
learning densities, and/or representations

$$p(\mathbf{x}) \qquad \text{or} \qquad p(\mathbf{x}, \mathbf{z})$$

reinforcement learning:
learn a series of actions, a policy

$$\pi(\mathbf{s}) \mapsto a_1, a_2, a_3, \ldots$$

the RL paradigm



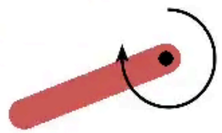
The goal

$$\max \mathbb{E}\left[\sum_t r_t\right]$$

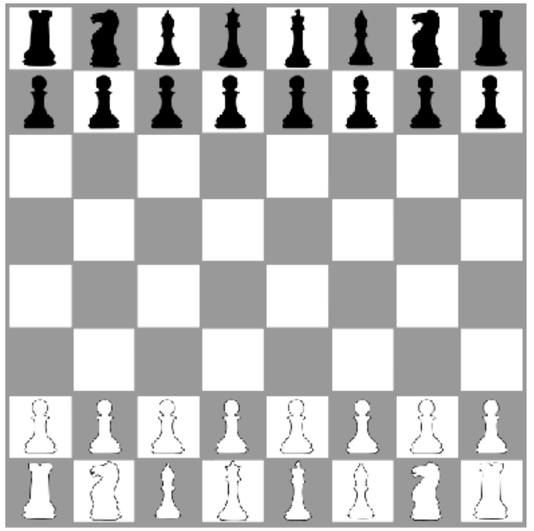
Unknown

$$\mathbf{s}_{t+1} \sim \mathbb{P}(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

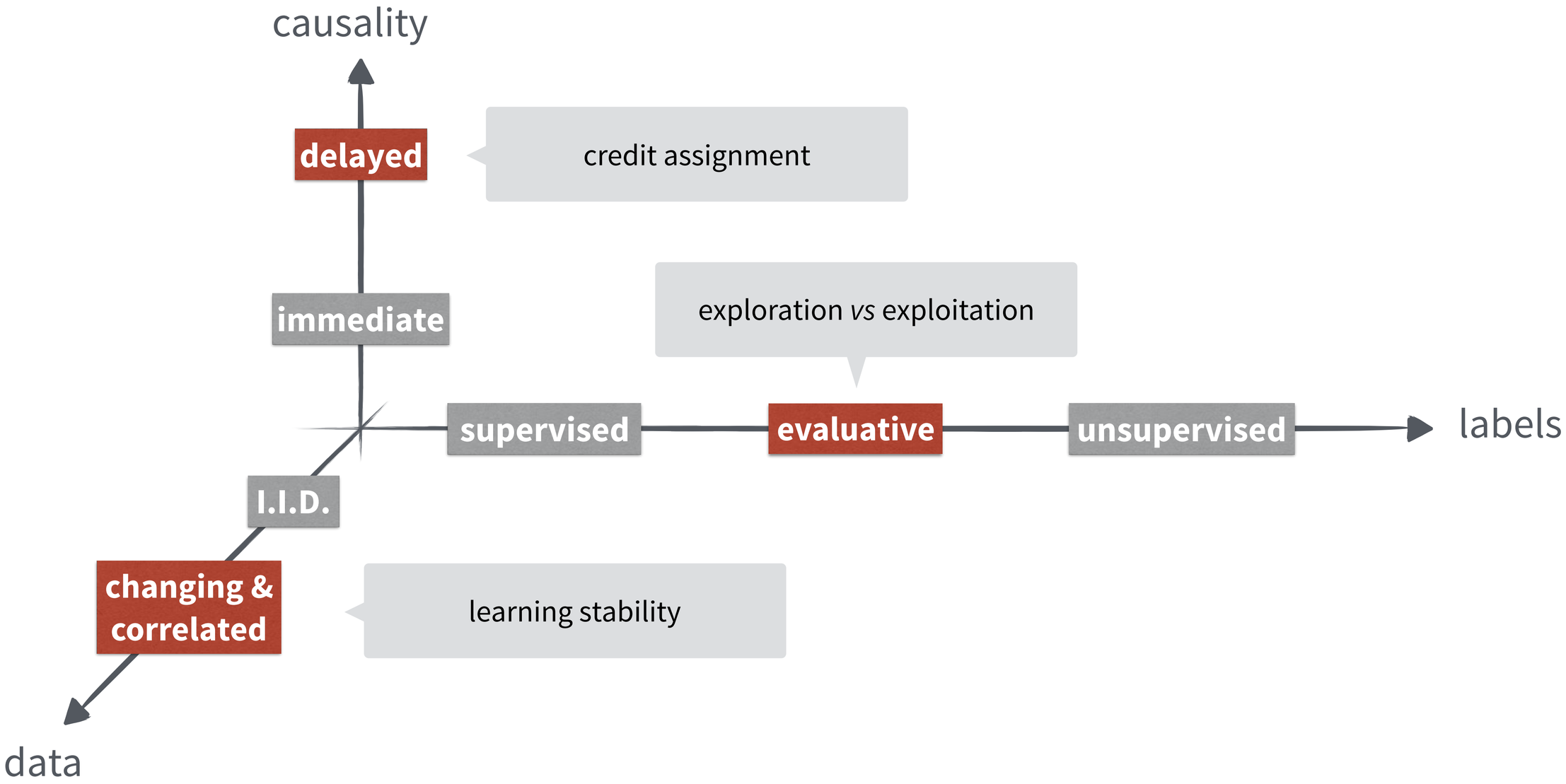
$$r_t \sim \mathbb{P}(r_t \mid \mathbf{s}_t, \mathbf{a}_t)$$




MARIO
005400
×22
WORLD
1-1
TIME
367



what makes **RL** harder than **SL**?



two main approaches for Reinforcement Learning

Value-based

$$V(s) = \mathbb{E}_t[R_t \mid S_t = s]$$
$$Q(s,a) = \mathbb{E}_t[R_t \mid S_t = s, A_t = a]$$

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

- dynamic programming
- temporal-difference (TD) learning
- indirect method in some sense
- not for large / continuous action space (until recently)

Policy-based

$$\pi_\theta(s,a) = \mathbb{P}(a \mid s, \theta)$$

optimization problem
Loss function $J(\theta)$
directly search in θ space

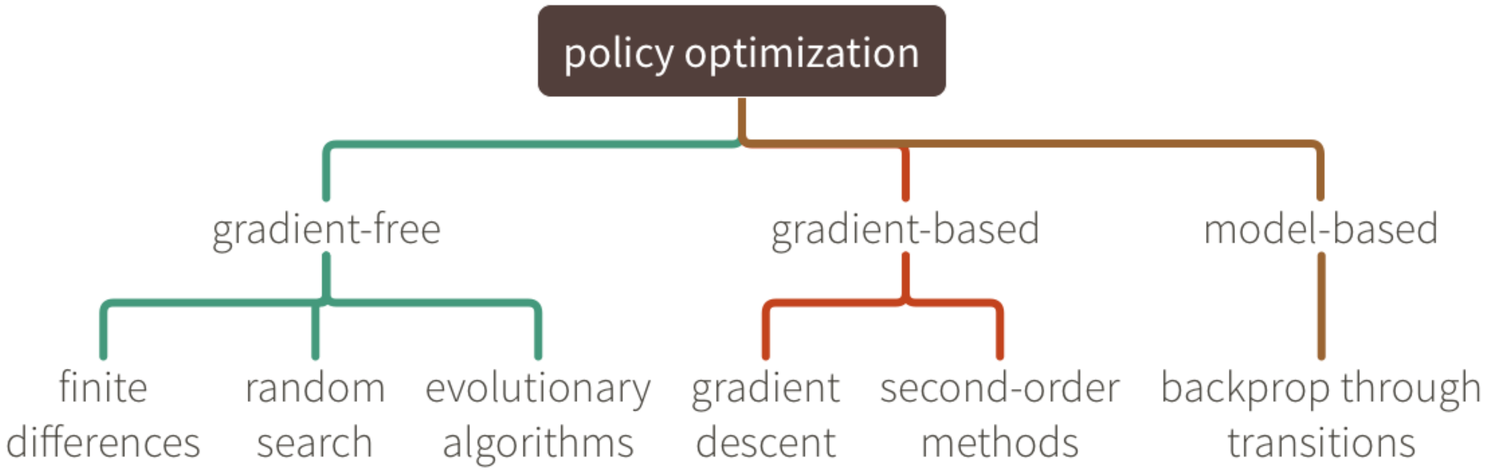
- stochastic policies
- handles continuous action space
- but… suffers from local optima and high variance

policy search (or optimization)

Parametrized policy $a \sim \pi_\theta(s) = \mathbb{P}(a \mid s, \theta)$ or $a = \pi_\theta(s) = f(a \mid s, \theta)$

Objective function $J(\theta) = \mathbb{E}_{p_\theta(\mathbf{s}_{1:t}, \mathbf{a}_{1:t})} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$

Find θ that maximizes $J(\theta)$



Policy Gradient

policy gradient: general formulation

actions $a \sim \pi_\theta(s) = \mathbb{P}(a \mid s, \theta)$

transitions $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a})$

trajectories $\tau = (\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$

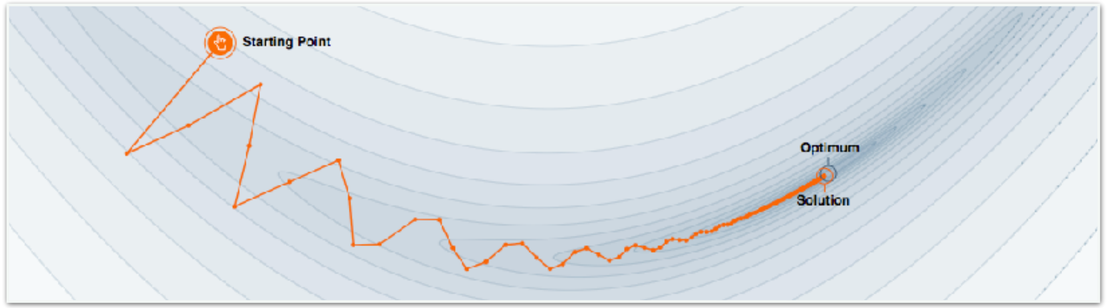
$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

cumulative reward $r(\tau) = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

Objective function $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]$

$$= \int r(\tau) p_\theta(\tau) \, \mathrm{d}\tau$$

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$



non-differentiable objective function!

➠ likelihood–ratio trick

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \int r(\tau) p_\theta(\tau)\,\mathrm{d}\tau \\
&= \int r(\tau) \nabla_\theta p_\theta(\tau)\,\mathrm{d}\tau \\
&= \int r(\tau) \nabla_\theta p_\theta(\tau) \left(\frac{p_\theta(\tau)}{p_\theta(\tau)}\right) \mathrm{d}\tau \\
&= \int r(\tau) p_\theta(\tau) \left(\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}\right) \mathrm{d}\tau \\
&= \int r(\tau) p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)\,\mathrm{d}\tau \\
&= \mathbb{E}_{\tau \sim p_\theta(\tau)} [r(\tau) \nabla_\theta \log p_\theta(\tau)]
\end{aligned}
$$

scale by the trajectory evaluation and follow the gradient of the trajectories logprop
(in expectation)

many names in the literature: score function estimator, likelihood–ratio trick, etc.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau) \nabla_\theta \log p_\theta(\tau)]$$

$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta \log p_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) + p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} r(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_t^{(i)} \,\middle|\, \mathbf{s}_t^{(i)} \right) \right)$$

➠ the REINFORCE algorithm

unbiased estimator

1 - interact with the environment (generate N trajectories)

2 - compute their cumulative reward

3 - compute an estimate of loss gradient $g \approx \nabla_\theta J(\theta)$

4 - update the policy parameters $\theta \leftarrow \theta + \alpha \times g$

Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Reinforcement Learning. Springer, Boston, MA, 1992.

Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems. 2000.

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Initialize policy parameter $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G \leftarrow$ return from step t
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$

source: Sutton & Barto

Gaussian policies

$$\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \sim \mathcal{N}(\mu(\mathbf{s}_t), \Sigma)$$

often a neural network

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = \frac{1}{2}\Sigma^{-1}(\mathbf{a}_t - \mu(\mathbf{s}_t))\nabla_\theta \mu(\mathbf{s}_t)$$

Softmax policies

$$\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = \frac{e^{f(\mathbf{s}_t, \mathbf{a}_t)}}{\sum_{\mathbf{a}'_t} e^{f(\mathbf{s}_t, \mathbf{a}'_t)}}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = f(\mathbf{s}_t, \mathbf{a}_t) - \mathbb{E}_{\pi_\theta}[f(\mathbf{s}_t, \cdot)]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau) \nabla_\theta \log p_\theta(\tau)]$$

$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta \log p_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) + p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \right]$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} r(\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_t^{(i)} \,\middle|\, \mathbf{s}_t^{(i)} \right) \right)$$

➠ the REINFORCE algorithm

unbiased estimator

high variance!

1 - interact with the environment (generate N trajectories)

2 - compute their cumulative reward

3 - compute an estimate of loss gradient $g \approx \nabla_\theta J(\theta)$

4 - update the policy parameters $\theta \leftarrow \theta + \alpha \times g$

Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Reinforcement Learning. Springer, Boston, MA, 1992.

Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems. 2000.

reducing the variance with the temporal structure

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right]$$

actions should only be influenced by future rewards

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \; Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

the return R_t is a sample of the Q value function $R_t = \sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \sim Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$

reducing the variance with a baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log p_\theta(\tau) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) - b(\mathbf{s}_t) \right) \right]$$

$$\begin{aligned}
\mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) B] &= \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) B \ \mathrm{d}\tau \\
&= \int p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} B \ \mathrm{d}\tau \\
&= \int \nabla_\theta p_\theta(\tau) B \ \mathrm{d}\tau \\
&= \nabla_\theta \int p_\theta(\tau) B \ \mathrm{d}\tau \\
&= B \ \nabla_\theta \int p_\theta(\tau) \ \mathrm{d}\tau \\
&= B \times 0
\end{aligned}$$

➠ still unbiased estimator

which baseline?

constant, or moving average $b(\mathbf{s}_t) = b \approx \frac{1}{N} \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

estimate the value function $b(\mathbf{s}_t) = \hat{V}_\nu(\mathbf{s}_t) = \mathbb{E}_t[R_t \mid S_t = s]$

intuition: only increase probability of actions that improve over the average

More substance in

Greensmith, Evan, Peter L. Bartlett, and Jonathan Baxter. "**Variance reduction techniques for gradient estimates in reinforcement learning.**" Journal of Machine Learning Research, 2004

let's recap

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \; Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \; Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi_\theta}(\mathbf{s}_t) \right]$$

actor-critic algorithms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \; Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

déjà-vu? Q-learning!

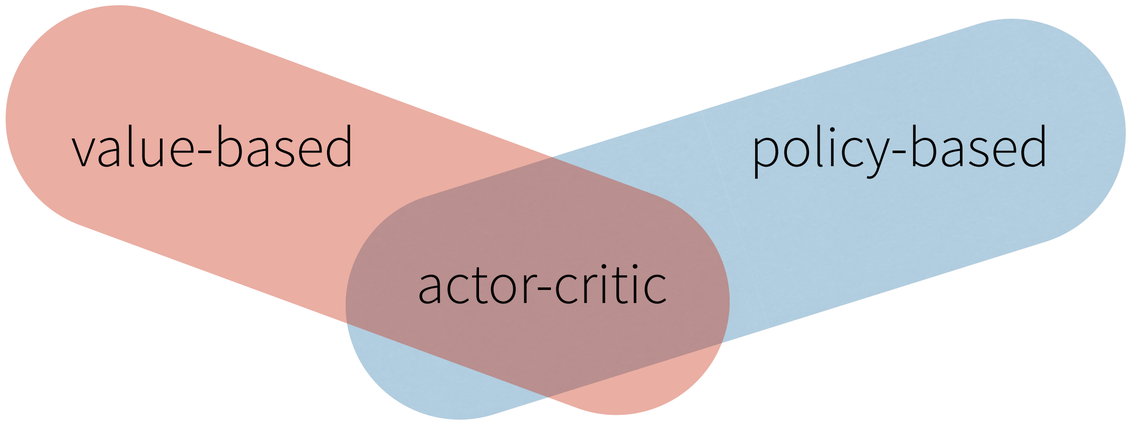
$$\approx \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \; \hat{Q}_\omega(\mathbf{s}_t, \mathbf{a}_t) \right]$$

instead of sampling $Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$

one can approximate it $\hat{Q}_\omega(\mathbf{s}_t, \mathbf{a}_t)$

we now have two parameters to fit

- θ for the policy **(the actor)**
- ω for the value function **(the critic)**



One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^d$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I \delta \nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

source: Sutton & Barto

actor-critic algorithms with a baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \underbrace{Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi_\theta}(\mathbf{s}_t)}_{A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)} \right]$$

advantage function

TD error $\delta_t^{\pi_\theta} = r_t + \gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t,)$

is an unbiased sample of the Advantage function $\delta_t^{\pi_\theta} \sim A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$

family of policy gradient algorithms

$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\ R_t$	vanilla policy gradient
$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\ (R_t - \hat{V}_\nu)$	REINFORCE
$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\ \hat{Q}_\omega(\mathbf{s}_t, \mathbf{a}_t)$	actor-critic
$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\ \hat{A}_\omega(\mathbf{s}_t, \mathbf{a}_t)$	advantage actor-critic
$\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\ \delta_t$	TD actor-critic

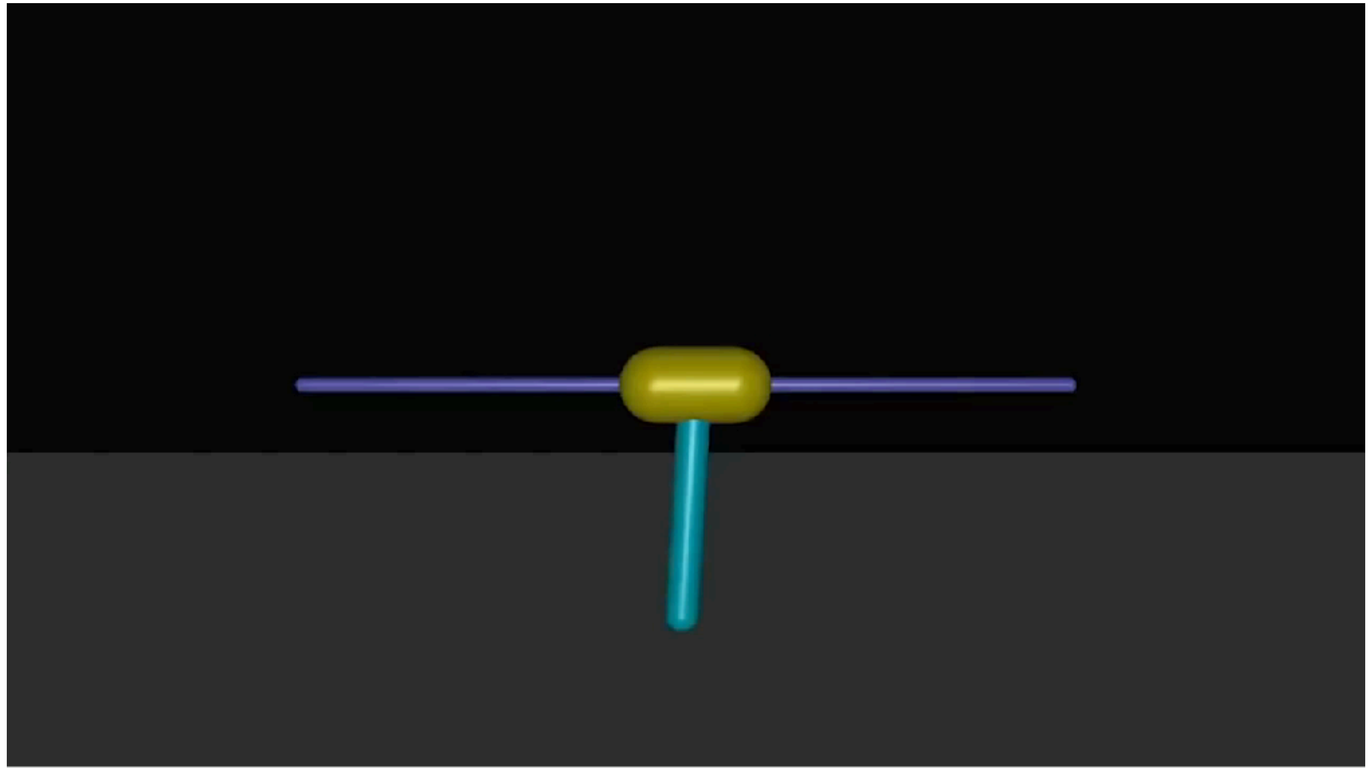
from David Silver's online course

example: Asynchronous Advantage Actor-Critic (A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.



using the MuJoCo library

gradient-free methods

finite differences

- approximate the gradient with numerical methods
- perturb the k^{th} component of θ by a small step size ν
- sample δ (eg. from a standard Gaussian distribution)
- $$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \nu\delta) - J(\theta - \nu\delta)}{2\nu}$$

Algorithm 1 Basic Random Search (BRS)

Hyperparameters: step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν

Initialize: $\theta_0 = \mathbf{0}$, and $j = 0$.

while ending condition not satisfied **do**

Sample $\delta_1, \delta_2, \ldots, \delta_N$ of the same size as θ_j, with i.i.d. standard normal entries.

Collect $2N$ rollouts of horizon H and their corresponding rewards using the policies

$$\pi_{j,k,+}(x) = \pi_{\theta_j + \nu\delta_k}(x) \quad \text{and} \quad \pi_{j,k,-}(x) = \pi_{\theta_j - \nu\delta_k}(x),$$

with $k \in \{1, 2, \ldots, N\}$.

Make the update step:

$$\theta_{j+1} = \theta_j + \frac{\alpha}{N} \sum_{k=1}^{N} \left[r(\pi_{j,k,+}) - r(\pi_{j,k,-}) \right] \delta_k \, .$$

$j \leftarrow j + 1$.

end while

.

1803.07055-Simple random search provides a competitive approach to reinforcement learning

augmented random search

augmented random search

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\textbf{V1:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\textbf{V2:} \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**

1803.07055-Simple random search provides a competitive approach to reinforcement learning

cross-entropy method

- generate random trajectories according to $\mathbb{P}(\cdot \mid \theta)$
- rank them according to their performance (eg. sum of rewards)
- use the best performing ones to improve the sampling process

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

source: Pieter Abbeel's slides

bottomline of gradient-free methods

- **easy** to implement
- good to have aside, as a toolbox for **comparison**
- not always suitable for **huge** parameter vectors
- don’t **bootstrap**, so they need to know the full trajectory
- no particular assumptions, ie., **blackbox** functions, except fast evaluation of the rewards
- common sense states: when you have a **gradient**, exploit it! (do you agree?)

closing

(some) research questions

model-based reinforcement learning

efficient reinforcement learning

exploration and intrinsic motivation

hierarchical reinforcement learning

going further

- **Reinforcement Learning: An Introduction**. Richard S. Sutton & Andrew G. Barto
  http://incompleteideas.net/book/the-book-2nd.html *(PDF freely available)*
- **Algorithms for Reinforcement Learning**. Csaba Szepesvári
  https://sites.ualberta.ca/~szepesva/RLBook.html *(PDF freely available)*
- **Approximate Dynamic Programming: Solving the curses of dimensionality** . Warren B. Powell
  http://adp.princeton.edu/

- David Silver's excellent course (on video)
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Pieter Abbeel & John Schulman's tutorial @NIPS2016 https://nips.cc/Conferences/2016/Schedule?showEvent=6198

libraries & frameworks

- **Showed during the talk**
  - javascript library https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

- **OpenAI Gym** https://gym.openai.com/
- **Keras-RL** https://github.com/matthiasplappert/keras-rl
- **TensorForce** https://reinforce.io/blog/introduction-to-tensorforce/
- **OpenAI Baselines** https://github.com/openai/baselines
- **Coach** http://coach.nervanasys.com/

questions?