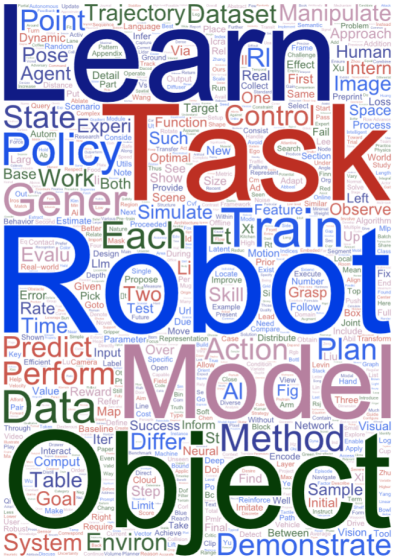
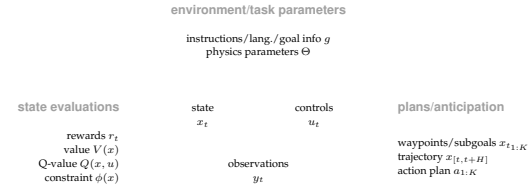


Robot Learning

Reinforcement Learning

Marc Toussaint
Technical University of Berlin
Summer 2024

I. What is learned?



- So far we discussed dynamics and imitation learning
  - The mappings we learned concerned x, y, u (including also dynamics parameters Θ and constraints $\phi(x)$)
  - Demonstration data was given, or dynamics data well-collected
  - There is no external task/cost evaluation
- In RL, we assume **rewards** r **given**, which opens a new dimension
  - We will learn state values (V-, Q-function) and a policy maximizing expected discounted rewards
  - RL is more autonomous in that it explores the world and generates its own data
  - But it relies on an externally given reward function

Outline

- First essentials towards modern Deep RL methods
- Then a discussion of challenges

Markov Decision Process

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.

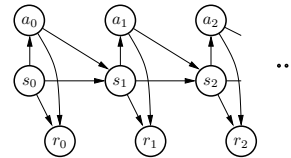
Markov Decision Process

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A parameterized policy $\pi_\theta(a_t|s_t)$.

Markov Decision Process

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A parameterized policy $\pi_\theta(a_t|s_t)$.
- Together they define the path distribution $(\xi = (s_{0:T+1}, a_{0:T}))$

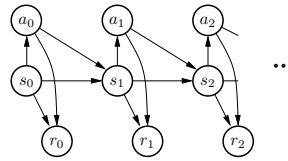
$$P_\theta(\xi) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t)\ P(s_{t+1}|s_t, a_t)$$



Markov Decision Process

- *The world:* An MDP $(\mathcal{S}, \mathcal{A}, P, R, P_0, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition probabilities $P(s_{t+1} \mid s_t, a_t)$, reward fct $r_t = R(s_t, a_t)$, initial state distribution $P_0(s_0)$, and discounting factor $\gamma \in [0, 1]$.
- *The agent:* A parameterized policy $\pi_\theta(a_t|s_t)$.
- Together they define the path distribution $(\xi = (s_{0:T+1}, a_{0:T}))$

$$P_\theta(\xi) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t)\ P(s_{t+1}|s_t, a_t)$$



and the **expected discounted return** (with *discounting factor* $\gamma \in [0, 1)$)

$$J(\theta) = \mathbb{E}_{\xi \sim P_\theta}\{\underbrace{\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)}_{R(\xi)}\} = \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

Value functions

[The following assumes a deterministic policy $a = \pi(s)$; stochastic $\pi(a|s)$ is handled with expectations over a.]

- The **value function** of a policy π_θ gives the return when started in state s:

$$V^\pi(s) = \mathbb{E}\left\{\sum_t \gamma^t r_t \mid s_0 = s\right\}$$

$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\left\{V^\pi(s')\right\} \qquad \text{(Bellman Equation)}$$

Value functions

[The following assumes a deterministic policy $a = \pi(s)$; stochastic $\pi(a|s)$ is handled with expectations over a.]

- The **value function** of a policy π_θ gives the return when started in state s:

$$V^\pi(s) = \mathbb{E}\left\{\sum_t \gamma^t r_t \mid s_0 = s\right\}$$
$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\left\{V^\pi(s')\right\} \qquad \text{(Bellman Equation)}$$

- The **Q-function** gives the return when starting in state s and taking first action a:

$$Q^\pi(s, a) = \mathbb{E}\left\{\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a\right\}$$
$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s'|s,a}\left\{Q^\pi(s', \pi(s'))\right\} \qquad \text{(Bellman Equation)}$$

Bellman Optimality Equation

- Bellman equations ($\leftrightarrow$ *Policy Evaluation*):

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\{V^{\pi}(s')\}$$
$$Q^{\pi}(s, a) = R(s, a) + \gamma \mathbb{E}_{s'|s,a}\{Q^{\pi}(s', \pi(s'))\}$$

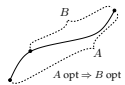
Bellman Optimality Equation

- Bellman equations ($\leftrightarrow$ *Policy Evaluation*):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \mathbb{E}_{s'|s,\pi(s)}\{V^\pi(s')\}$$
$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{Q^\pi(s', \pi(s'))\}$$

- Bellman optimality equations: ($\leftrightarrow$ Q-Iteration/Value Iteration)

$$V^*(s) = \max_a \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{V^*(s')\} \Big] = \max_a Q^*(s,a)$$
$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q^*(s', a')\}$$
$$\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$$



Richard E. Bellman (1920–1984)

[Sketch of proof: If π^* would be other than $\operatorname{argmax}_a[\cdot]$, then $\pi' = \pi$ everywhere except $\pi'(s) = \operatorname{argmax}_a[\cdot]$ would be better.]

- The core question is how to actually compute them

- Model-based: (if we know or estimated the models $P(s'|s,a), R(s,a), P(s_0)$)
  - Q-Iteration, Policy Iteration

- Data-based: (if we directly use data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^{n}$)
  - "Reinforcement Learning"
  - TD-Learning, Q-learning, Actor-Critic
  - Modern: DDPG, TC3, SAC, etc

Model-based: Q-Iteration

- Bellman Optimality equation for Q^*:

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \mid s,a} \Big\{ \underbrace{\max_{a'} Q^*(s',a')}_{V^*(s')} \Big\}$$

Model-based: Q-Iteration

- Bellman Optimality equation for Q^*:

$$Q^*(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \mid s,a} \Big\{ \underbrace{\max_{a'} Q^*(s',a')}_{V^*(s')} \Big\}$$

- **Q-Iteration:** initialize $Q_{k=0}(s,a) = 0$, then iterate:

$$\forall_s : \; V_{k+1}(s) = \max_{a'} Q_k(s,a')$$
$$\forall_{s,a} : \; Q_{k+1}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a} \big\{ V_{k+1}(s') \big\}$$

stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \leq \epsilon$

[Note: Using V_{k+1} in this iteration is like a buffer – cf. the "target network" in neural RL.]

- **Theorem:** Q-Iteration converges to the optimal state-action value function Q^*

Q-Iteration – Proof of convergence

- Let $\Delta_k = \|Q^* - Q_k\|_\infty = \max_{s,a} |Q^*(s,a) - Q_k(s,a)|$

$$
\begin{aligned}
Q_{k+1}(s,a) &= R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q_k(s',a')\} \\
&\leq R(s,a) + \gamma \mathbb{E}_{s'|s,a}\Big\{\max_{a'} \Big[Q^*(s',a') + \Delta_k\Big]\Big\} \\
&= \Big[R(s,a) + \gamma \mathbb{E}_{s'|s,a}\{\max_{a'} Q^*(s',a')\}\Big] + \gamma\Delta_k \\
&= Q^*(s,a) + \gamma\Delta_k
\end{aligned}
$$

  similarly: $Q_{k+1} \geq Q^* - \gamma\Delta_k$

- The proof translates directly also to value iteration

Model-based: Policy Iteration

- **Policy Evaluation:** Dynamic Programming for Q^π instead of Q^*: Iterate:

$$\forall_s : \; V_{k+1}(s) = Q_k(s, \pi(s))$$
$$\forall_{s,a} : \; Q_{k+1}(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\big\{V_{k+1}(s')\big\}$$

  stopping criterion: $\max_{s,a} |Q_{k+1}(s,a) - Q_k(s,a)| \leq \epsilon$

- **Policy Improvement:** Then update the policy to become better:

$$\pi(s) \leftarrow \operatorname*{argmax}_a Q(s,a)$$

- Iterating the two steps above is guaranteed to converge
- This is also called **actor-critic** (with π=actor, and Q^π=critic)

- The two discussed methods (Q-Iteration and Policy Iteration) can compute optimal policies, but require a known (or estimated) model
- To approximately do the same from data, we follow two strategies
  - Whenever there was an expectation $\mathbb{E}\{\cdot\}$ in these equations, we replace it by sample data
  - Whenever there was a full function update (e.g. $\forall_{s,a} : Q(s,a) \leftarrow \cdots$ or policy improvement) we need to replace it by a **data-based loss functions** and do gradient steps.

- The two discussed methods (Q-Iteration and Policy Iteration) can compute optimal policies, but require a known (or estimated) model
- To approximately do the same from data, we follow two strategies
  - Whenever there was an expectation $\mathbb{E}\{\cdot\}$ in these equations, we replace it by sample data
  - Whenever there was a full function update (e.g. $\forall_{s,a} : Q(s,a) \leftarrow \cdots$ or policy improvement) we need to replace it by a **data-based loss functions** and do gradient steps.

- For simplicity, the following focusses on Policy Iteration (or *actor-critic*) approaches

  [Similar strategies can be applied for "Deep Q-Learning":

  V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski. Human-level control through deep reinforcement learning.
  nature, 518(7540):529–533, 2015.
  URL: https://www.nature.com/articles/nature14236

  But major RL methods nowadays follow actor-critic approaches]

Data-based: Bellman Loss for the Q-function

- Recall

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\big\{Q^\pi(s', \pi(s'))\big\}$$

- Given data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^T$, define the **Bellman residual**:

$$\mathcal{B}^\pi(Q_\theta, \bar{Q}) = \mathbb{E}_{(s,a,r,s')\sim D}\big\{\ [Q_\theta(s,a) - r - \gamma \bar{Q}(s', \pi(s'))]^2\ \big\}$$

Data-based: Bellman Loss for the Q-function

- Recall

$$Q^\pi(s,a) = R(s,a) + \gamma \mathbb{E}_{s'|s,a}\big\{Q^\pi(s', \pi(s'))\big\}$$

- Given data $D = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^T$, define the **Bellman residual**:

$$\mathcal{B}^\pi(Q_\theta, \bar{Q}) = \mathbb{E}_{(s,a,r,s')\sim D}\big\{\ [Q_\theta(s,a) - r - \gamma \bar{Q}(s', \pi(s'))]^2\ \big\}$$

- This defines a supervised ML problem for Q_θ! We have Q-gradients and can do standard SGD.
  - Actually we want $\bar{Q} \equiv Q_\theta$, and could compute gradients also accounting for $\gamma \bar{Q}(s', \pi(s'))$. This is called **Bellman residual minimization**, and known since the 80ies, but has challenges [12, 4]
  - So instead, during training we fix $\bar{Q}$ to some "old version" of Q_θ: We set $\bar{Q} = Q_{\bar{\theta}}$ where $\bar{\theta}$ is a low-pass filter of θ (a **delayed** version of the current parameters θ). This stabilizes training.

- So, for a given policy π, $\mathcal{B}^\pi(Q_\theta, \bar{Q})$ defines a loss for Q_θ
- How can we also define a loss function for the policy?

Data-based: Return Maximization for the Policy

- To train the policy, we choose to directly maximize expected return:

$$J(\theta) = \mathbb{E}_{\xi \sim P_\theta} \Big\{ \underbrace{\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)}_{R(\xi)} \Big\} = \int_\xi P_\theta(\xi)\ R(\xi)\ d\xi$$

  - This is not really an error, but exactly what we aim to maximize
  - All we need is the gradient $\frac{\partial}{\partial \theta} J(\theta)$

Policy Gradient $\frac{\partial}{\partial\theta}J(\theta)$

[The word "policy gradient" means gradient of $J(\theta)$ w.r.t. the policy parameters θ.]

- For a deterministic policy $a = \pi_\theta(s) \in \mathbb{R}^d$:

$$\frac{\partial}{\partial\theta}J(\theta) = \mathbb{E}_{s\sim P_\theta}\left\{\frac{\partial}{\partial a}Q^{\pi_\theta}(s,a)\Big|_{a=\pi_\theta(s)}\frac{\partial}{\partial\theta}\pi(s)\right\}$$

[Derived here: [18], and led to the **Deep Deterministic Policy Gradient (DDPG)** method [11]. Is the foundation of many followups. This gradient is somewhat noisy, D4PG is an improvement.]

Policy Gradient $\frac{\partial}{\partial\theta}J(\theta)$

[The word "policy gradient" means gradient of $J(\theta)$ w.r.t. the policy parameters θ.]

- For a deterministic policy $a = \pi_\theta(s) \in \mathbb{R}^d$:

$$\frac{\partial}{\partial\theta}J(\theta) = \mathbb{E}_{s\sim P_\theta}\left\{\frac{\partial}{\partial a}Q^{\pi_\theta}(s,a)\Big|_{a=\pi_\theta(s)}\frac{\partial}{\partial\theta}\pi(s)\right\}$$

[Derived here: [18], and led to the **Deep Deterministic Policy Gradient (DDPG)** method [11]. Is the foundation of many followups. This gradient is somewhat noisy, D4PG is an improvement.]

- For a stochastic policy $\pi_\theta(a|s)$: (standard "Policy Gradient Theorem"):

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{\partial}{\partial\theta}\int P_\theta(\xi)\ R(\xi)\ d\xi = \int P_\theta(\xi)\frac{\partial}{\partial\theta}\log P_\theta(\xi)R(\xi)d\xi$$
$$= \mathbb{E}_{\xi\sim P_\theta}\left\{\frac{\partial}{\partial\theta}\log P_\theta(\xi)R(\xi)\right\} = \mathbb{E}_{\xi\sim P_\theta}\left\{\sum_{t=0}^{H}\gamma^t\ [\frac{\partial}{\partial\theta}\log\pi_\theta(a_t|s_t)]\ \underbrace{\sum_{t'=t}^{H}\gamma^{t'-t}r_{t'}}_{Q^{\pi_\theta}(s_t,a_t)}\right\}$$

RL: Interleaving training with data collection

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

- Actor-Critic style Deep RL:
  - $\frac{\partial}{\partial\theta}\mathcal{B}(Q_\theta, \bar{Q})$ provides gradient steps for Q_θ
  - $\frac{\partial}{\partial\theta} J(\theta)$ provides gradient steps for π_θ
  - gradually training both is interleaved with collecting more data

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods.
In *International Conference on Machine Learning*, pages 1587–1596, 2018.
URL: https://proceedings.mlr.press/v80/fujimoto18a.html

Techniques to improve methods

- Papers on techniques in state-of-the-art methods:
  - In Deep Q-Learning (DQN) approaches: [7] (Rainbow paper)
  - In Actor-Critic approaches: [3] (TD3 paper)
  - A state-of-the-art actor-critic method: [5] (SAC paper)

Techniques to improve methods

- Papers on techniques in state-of-the-art methods:
  - In Deep Q-Learning (DQN) approaches: [7] (Rainbow paper)
  - In Actor-Critic approaches: [3] (TD3 paper)
  - A state-of-the-art actor-critic method: [5] (SAC paper)
- Many ideas:
  - Replay buffers ("experience replay"): Limited buffer of experiences to train on (approximates $P_\theta(s, a, r, s')$)
  - Double Q-Learning: maintain 2 indep. Q-functions $Q_{1,2}(s, a)$ (and use min in policy update)
  - Delayed targets: low pass filter $\bar{Q}$ of Q as target
  - Smoothed policy samples: add (clipped) noise when sampling policy in Bellman loss
  - Prioritized Replay: (pick replay data where Bellman error is largest)
  - Dueling Networks: (decompose Q in value and advantage)
  - Multi-Step Learning: (n-step updates)
  - Distributional RL: (let Q-function predict return *distribution*, not mean)
  - Noisy Nets: (replace ϵ-greedy exploration by "learnt noise")

Discussion

- The previous material should enable you to read about modern Deep RL methods (TD3, D4PG, SAC)

- Rest of this lecture is discussion
  - Why do we actually learn Q and not V?
  - What if we have partial observability?
  - How is the data collected?
  - How are reward functions engineered?
  - Why not just use black-box optimization?

Why do we actually learn Q and not V?

Why do we actually learn Q and not V?

- $Q(s, a)$ tells us what is the best action $a = \text{argmax}_a Q$

Why do we actually learn Q and not V?

- $Q(s, a)$ tells us what is the best action $a = \text{argmax}_a Q$
- In control, value functions are also estimated, but never Q (I think). Why?

  [E.g. the Hamilton-Jacobi-Bellman Eq: $-\frac{\partial}{\partial t} V(x,t) = \min_u \left[c(x,u) + \frac{\partial V}{\partial x} f(x,u) \right]$.]

Why do we actually learn Q and not V?

- $Q(s, a)$ tells us what is the best action $a = \text{argmax}_a Q$
- In control, value functions are also estimated, but never Q (I think). Why?
  [E.g. the Hamilton-Jacobi-Bellman Eq: $-\frac{\partial}{\partial t} V(x,t) = \min_u \left[c(x,u) + \frac{\partial V}{\partial x} f(x,u) \right]$.]

- Without Q-function, we'd somehow have to learn how to walk up-hill on V:
  - Learn an *inverse model* $(s, \Delta s) \mapsto a$
  - Learn a "flow" policy $\pi : s \mapsto \Delta s \approx \frac{\partial}{\partial s} V(s)$

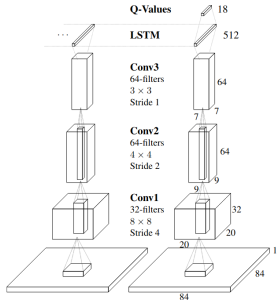
What if we have partial observability?

- Policy has only access to observations $y_{0:t}$

What if we have partial observability?

- Policy has only access to observations $y_{0:t}$

$\rightarrow$ Make the Q function a recursive NN



M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *2015 Aaai Fall Symposium Series*, 2015.
URL: https://cdn.aaai.org/ocs/11673/11673-51288-1-PB.pdf

How is the data collected?

How is the data collected?

- A core challenge in modern RL!
- Many modern methods *require* that the data is collected from the current π_θ!
  - So that $\mathbb{E}\{\cdot\}$ can be replaced by the data in the Bellman equations
  - This is called **on-policy** – we'll discuss off-policy next time
  - But π is so uninformed! So non-exploring! So iid. in each step ($\sim$ Brownian noise)
  - Check pseudo codes of mentioned methods (SAC, DDPG, TD3, etc)

How is the data collected?

- A core challenge in modern RL!
- Many modern methods *require* that the data is collected from the current π_θ!
  - So that $\mathbb{E}\{\cdot\}$ can be replaced by the data in the Bellman equations
  - This is called **on-policy** – we'll discuss off-policy next time
  - But π is so uninformed! So non-exploring! So iid. in each step ($\sim$ Brownian noise)
  - Check pseudo codes of mentioned methods (SAC, DDPG, TD3, etc)

- In old RL (discrete state-action spaces), things were much better!
  - **Explicit Exploit or Explore** [8] – a *must read!*
  - R-MAX [1], Optimistic value initialization, Bayesian RL
  - These methods design policies to systematically explore, typically by **systematically rewarding exploration**
  - Optimism in the face of uncertainty: Rewarding decisions with uncertain outcomes!

How is the data collected?

- In Deep RL: Structured noise instead of Brownian:

  O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning.
  In *The Eleventh International Conference on Learning Representations*, 2022.
  URL: https://openreview.net/forum?id=hQ9V5QN27eS

- Parameter-space noise: (add noise to θ instead of a)

  M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter Space Noise for Exploration, 2018-01-31.
  URL: http://arxiv.org/abs/1706.01905, arXiv:1706.01905

How is the data collected?

- In Deep RL: Structured noise instead of Brownian:

  O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius. Pink noise is all you need: Colored noise exploration in deep reinforcement learning.
  In *The Eleventh International Conference on Learning Representations*, 2022.
  URL: https://openreview.net/forum?id=hQ9V5QN27eS

- Parameter-space noise: (add noise to θ instead of a)

  M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter Space Noise for Exploration, 2018-01-31.
  URL: http://arxiv.org/abs/1706.01905, arXiv:1706.01905

- Guided Policy Search

  S. Levine and V. Koltun. Guided policy search.
  In *International Conference on Machine Learning*, pages 1–9, 2013.
  URL: https://proceedings.mlr.press/v28/levine13.html

  – Use model-based trajectory optimization to generate data

- Demonstration Guided [15]
- Or just give up:
  – Offline Reinforcement Learning: Assume the data was generated somehow externally
  – Imitation Learning & Inverse RL: Learn from demonstrations

How are reward functions engineered?

- Reward shaping theory: You can add potentials without changing optimal policy

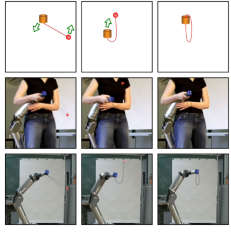
A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping.
In *Icml*, volume 99, pages 278–287, 1999.
URL: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/NgHaradaRussell-shaping-ICML1999.pdf

How are reward functions engineered?

- Reward shaping theory: You can add potentials without changing optimal policy

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping.
In *Icml*, volume 99, pages 278–287, 1999.
URL: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/NgHaradaRussell-shaping-ICML1999.pdf

- Reward engineering:



employ the same joint final reward. At the time t_c where the ball passes the rim of the cup with a downward direction, we compute the reward as $r(t_c) = \exp(-\alpha(x_c - x_b)^2 - \alpha(y_c - y_b)^2)$ while we have $r(t) = 0$ for all $t = t_c$. Here, the cup position is denoted by $[x_c, y_c, z_c] \in \mathbb{R}^3$, the ball position $[x_b, y_b, z_b] \in \mathbb{R}^3$ and we have a scaling parameter $\alpha = 100$. The directional information is necessary as the algorithm could otherwise learn to hit the bottom of the cup with the ball. The

J. Kober and J. Peters. Learning motor primitives for robotics.
In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118, 2009.
URL: https://ieeexplore.ieee.org/abstract/document/5152577/

https://www.youtube.com/watch?v=qtqubguikMk

Why not just use black-box optimization?

- Eventually, $\max_\theta J(\theta)$ is an optimization problem
  - Instead of deriving gradients (via Bellman, and Q-functions), why not treat as black-box or **derivative-free optimization** problem?

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans **Jonathan Ho** **Xi Chen** **Szymon Sidor** **Ilya Sutskever**
OpenAI

Abstract

We explore the use of Evolution Strategies (ES), a class of black box optimization algorithms, as an alternative to popular MDP-based RL techniques such as Q-learning and Policy Gradients. Experiments on MuJoCo and Atari show that ES is a viable solution strategy that scales extremely well with the number of CPUs available: By using a novel communication strategy based on common random numbers, our ES implementation only needs to communicate scalars, making it possible to scale to over a thousand parallel workers. This allows us to solve 3D humanoid walking in 10 minutes and obtain competitive results on most Atari games after one hour of training. In addition, we highlight several advantages of ES as a black box optimization technique: it is invariant to action frequency and delayed rewards, tolerant of extremely long horizons, and does not need temporal discounting or value function approximation.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning, 2017-09-07.
URL: `http://arxiv.org/abs/1703.03864`, `arXiv:1703.03864`

- Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps:

Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88

Deep Neuroevolution: Genetic Algorithms are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning

Felipe Petroski Such Vashisht Madhavan Edoardo Conti Joel Lehman Kenneth O. Stanley Jeff Clune

Uber AI Labs

{felipe.such, jeffclune}@uber.com

F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, 2018-04-20.
URL: http://arxiv.org/abs/1712.06567, arXiv:1712.06567

- Roughly: “Do you spend your time training nets, or simulating?”

	DQN	ES	A3C	RS 1B	GA 1B	GA 6B
DQN		6	6	3	6	7
ES	7		7	3	6	8
A3C	7	6		6	6	7
RS 1B	10	10	7		13	13
GA 1B	7	7	7	0		13
GA 6B	6	5	6	0	0	

Table 4. **Head-to-head comparison between algorithms on the 13 Atari games.** Each value represents how many games for which the algorithm listed at the top of a column produces a higher score than the algorithm listed to the left of that row (e.g. GA 6B beats DQN on 7 games).

- Conclusion: It varies from problem to problem what is better.
  And it is suprising that “naive” black-box ES can beat elaborate RL-methods

[1] R. I. Brafman and M. Tennenholtz.
R-max-a general polynomial time algorithm for near-optimal reinforcement learning.
Journal of Machine Learning Research, 3:213–231, 2002.
URL: https://www.jmlr.org/papers/volume3/brafman02a/brafman02a.pdf?ref=https://githubhelp.com.

[2] O. Eberhard, J. Hollenstein, C. Pinneri, and G. Martius.
Pink noise is all you need: Colored noise exploration in deep reinforcement learning.
In *The Eleventh International Conference on Learning Representations*, 2022.
URL: https://openreview.net/forum?id=hQ9V5QN27eS.

[3] S. Fujimoto, H. Hoof, and D. Meger.
Addressing function approximation error in actor-critic methods.
In *International Conference on Machine Learning*, pages 1587–1596, 2018.
URL: https://proceedings.mlr.press/v80/fujimoto18a.html.

[4] M. Geist, B. Piot, and O. Pietquin.
Is the Bellman residual a bad proxy?
Advances in Neural Information Processing Systems, 30, 2017.
URL: https://proceedings.neurips.cc/paper/2017/hash/e0ab531ec312161511493b002f9be2ee-Abstract.html.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine.
Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
In *International Conference on Machine Learning*, pages 1861–1870, 2018.
URL: https://proceedings.mlr.press/v80/haarnoja18b.

[6] M. Hausknecht and P. Stone.
Deep recurrent q-learning for partially observable mdps.
In *2015 Aaai Fall Symposium Series*, 2015.
URL: https://cdn.aaai.org/ocs/11673/11673-51288-1-PB.pdf.

[7] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver.
Rainbow: Combining improvements in deep reinforcement learning.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
URL: https://ojs.aaai.org/index.php/AAAI/article/view/11796.

[8] M. Kearns and S. Singh.
Near-optimal reinforcement learning in polynomial time.
Machine Learning, 49(2/3):209–232, 2002.
URL: http://link.springer.com/10.1023/A:1017984413808.

[9] J. Kober and J. Peters.
Learning motor primitives for robotics.
In *2009 IEEE International Conference on Robotics and Automation*, pages 2112–2118, 2009.
URL: https://ieeexplore.ieee.org/abstract/document/5152577/.

[10] S. Levine and V. Koltun.
Guided policy search.
In *International Conference on Machine Learning*, pages 1–9, 2013.
URL: https://proceedings.mlr.press/v28/levine13.html.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra.
Continuous control with deep reinforcement learning, 2019-07-05.
URL: http://arxiv.org/abs/1509.02971, arXiv:1509.02971.

[12] O.-A. Maillard, R. Munos, A. Lazaric, and M. Ghavamzadeh.
Finite-sample analysis of Bellman residual minimization.
In *Proceedings of 2nd Asian Conference on Machine Learning*, pages 299–314, 2010.
URL: http://proceedings.mlr.press/v13/maillard10a.html.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski.
Human-level control through deep reinforcement learning.
nature, 518(7540):529–533, 2015.
URL: https://www.nature.com/articles/nature14236.

[14] A. Y. Ng, D. Harada, and S. Russell.
Policy invariance under reward transformations: Theory and application to reward shaping.
In *Icml*, volume 99, pages 278–287, 1999.
URL: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/NgHaradaRussell-shaping-ICML1999.pdf.

[15] K. Pertsch, Y. Lee, Y. Wu, and J. J. Lim.
Demonstration-Guided reinforcement learning with learned skills.
2021.
arXiv:2107.10253.

[16] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz.
Parameter Space Noise for Exploration, 2018-01-31.
URL: http://arxiv.org/abs/1706.01905, arXiv:1706.01905.

[17] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever.
Evolution Strategies as a Scalable Alternative to Reinforcement Learning, 2017-09-07.
URL: http://arxiv.org/abs/1703.03864, arXiv:1703.03864.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller.
Deterministic policy gradient algorithms.
In *International Conference on Machine Learning*, pages 387–395, 2014.
URL: http://proceedings.mlr.press/v32/silver14.html.

[19] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune.

Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, 2018-04-20.
URL: http://arxiv.org/abs/1712.06567, arXiv:1712.06567.