Introduction to Policy Search in Reinforcement Learning

Djalel Benbouzid — Machine Learning Research Lab, Volkswagen Group

ARGMAX.ai
VOLKSWAGEN GROUP ML RESEARCH

AAISU21

August 24th 2021

introduction

“traditional” programming

machine learning

“traditional” programming

supervised learning

reinforcement learning

Learning to act

Can we solve that with supervised learning?

Learning to play Mario with Supervised Learning?

- First, how do we collect labels (optimal controls) for our data?
- Let's assume we have access to an *oracle* that can label any part of the game for us
  - So, we collect *a whole lot* of labelled data about one stage
  - Can we guarantee anything for the next stage?

- Now, what if instead of labels (the optimal actions) we only get a scalar signal for the actions we make (something like a reward/punishment)
  - We try some actions, we get some reward…
  - Question: should we stick to these actions? Or try new ones for potentially higher rewards?

- Can associate a reward with *all* the actions? Should we?

RL deals with all of these questions (and more)…

Some definitions

Reinforcement learning is the study of how to use past data to enhance the future manipulation of **dynamical systems**. (Ben Recht)

A way of programming agents by **reward and punishment** without needing to specify **how** the task is to be achieved. (L. Kaelbling, M. Littman and A. Moore, 1996)

RL is the study of agents that learn by **trial and error**, so as to maximise the cumulative reward and punishment.

Learning to control

- what could be the observations?
- what could be the controls?
- what could be the reward?

Learning to plan

- what could be the observations?
- what could be the controls?
- what could be the reward?

Learning to plan

- what could be the observations?
- what could be the controls?
- what could be the reward?

Success of Reinforcement Learning

Ng et al, 2004

Mnih et al, 2013

Hess et al, 2017

Silver et al, 2017

Kalashnikov et al. 2018

What makes **RL** ~~different from~~ harder than **SL**?

Origins of RL

Edward L. Thorndike (1874 - 1949)

Learning by trial and error

Timeline of RL

Core concepts

The agent-environment interface

- The environment is usually assumed to be known
- The boundary expresses the limits of the agent's control, more than a "physical" separation
- Here we only consider full observability of the states

Markov Decision Process (MDP)

- An MDP consists in
  - a set of states $\mathcal{S}$
  - a set of actions $\mathcal{A}$
  - transition probabilities

    $\mathbb{P}(S_{t+1} \mid S_t, A_t)$
  - a reward function

    $r_{t+1} = \mathbb{E}[R_{t+1} \mid S_t, A_t]$
  - a discount factor $\gamma \in [0, 1]$
- A trajectory (or rollout) of horizon T

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

The goal

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_t r_t\right]$$

Unknown

$$\mathbb{P}(S_{t+1} \mid S_t, A_t)$$

$$\mathbb{P}(R_{t+1} \mid S_t, A_t)$$

The Markov assumption

- The Markov property means the present state is enough for making decisions
- The rest of the history does not need to be stored

$$\mathbb{P}(S_{t+1} \mid S_t, \ldots, S_0) = \mathbb{P}(S_{t+1} \mid S_t)$$

- The future is independent of the past given the present

$$S_{>t} \perp\!\!\!\perp S_{<t} \mid S_t$$

States

- Set of states $\mathcal{S}$
  - The state s_t describe the configuration of the environment at a particular time t
  - The state contains all necessary information for predicting the subsequent state
  - It also sufficient for making a decision
  - It can be discrete or continuous
  - eg.
    - joints angle and angular velocities of a robot arm
    - a couple of frames from a video game

Actions

- A set of actions $\mathcal{A}$
  - The actions modify the state of the environment
  - Sometimes called *controls*
  - Actions can can be discrete or continuous
  - eg.
    - torque applied to the joints of a robot arm
    - move left/right/up/down in a maze
  - The goal is to learn a policy $\pi(a \mid s)$

MDP example

source: wikipedia

Reward and returns

The reward hypothesis (paraphrased from Sutton & Barto, 2018)

Goals and purposes can be expressed as maximizing the expected value of the cumulative sum of a received scalar signal (ie., the reward)

- The reward is a single scalar signal, R_t, emitted by the environment
  - The reward is **evaluative**, not indicative
  - It can be sparse, delayed
  - It expresses *what* we want to achieve, not *how* to achieve it
- The agent maximizes the **total amount of rewards** it gets (the *return* G)
  - → Greedy maximization is often not optimal
- Examples of rewards in nature?

Reward and returns (2)

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\
&= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

- The return G_t is the *discounted* cumulative reward
- $\gamma \in [0, 1]$: discount factor
- Examples or rewards
  - chess: +1 when winning, 0 when losing
  - walking: distance travelled
  - solving a maze: -1 for every step, 0 when done

Example

Episodic / continuing tasks

- Episodic tasks run for a (variable) number of steps before stopping
- Continuing tasks do not have terminal states
- They can however treated as unified

- Discounting expresses preference for short-term over long-term (more uncertain) reward
- It also makes things mathematically convenient for continuing tasks

figure source: Sutton & Barto 2018

The goal: training agents

- An agent might consist in one or more items among:

  - **a policy** π : a mapping from states to actions
  - **a value function** V **or** Q: estimate the expected return of state (or state-action)
  - **a model** of its environment: learning to predict the next states and rewards

Policies

- a function that maps **states → actions**
- can be

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

- note that the policy does not depend on the history (the Markov property)
- Policies can be greedy wrt. a value function

Examples of policies

- A random policy

- Probability of 1/4 for each direction

Examples of policies

- Another stochastic policy

Examples of policies

- A deterministic policy

Direct policy search

Motivation

General Idea:

- **Parametrize** a policy
- Define a **loss** function
- Apply an **optimization** procedure on the policy parameters to minimize the loss

Policy search: general formulation

- The policy is stochastic
- The policy is parametrized by the parameter vector θ
- Eg. a linear function, a neural network
- Notice $p_\theta(\tau)$ is also parametrized

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

Policy search: objective function

- Let us define a generic objective function wrt. the probably of the trajectories

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)]$$
$$= \int r(\tau) p_\theta(\tau) \, \mathrm{d}\tau$$

- We seek to find the *optimal* set of parameters $\theta^\star$ such that

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

can we use gradient ascent? 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.

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

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

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

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

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

➠ 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

Policy parametrization

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

Contiunous actions:
Gaussian policies

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

often a neural network

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

Discrete actions:
Softmax policies

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

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

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

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

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

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

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

➠ 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

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

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

actions should only be influenced by future rewards

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

Recap

we defined an optimization procedure

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

we derived a gradient update

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

we reduced the variance of our estimator

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

we further subtracted a baseline

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

Gradient-free methods

Motivation

- Are we obliged to use gradient-based methods?
- No! zero order methods can do the job. However, it depends on
  - The size of the policy parametrization
  - The cost of sampling trajectories
- Gradients convey some information but other methods have their own advantages

Finite differences

- approximate the gradient with numerical methods
- perturb the kth component of $\boldsymbol{\theta}$ by a small step size $\boldsymbol{\nu}$
- sample $\boldsymbol{\delta}$ (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:

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

$j \leftarrow j + 1$.

end while

.

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

Can we improve upon that?

- Simple “tricks” lead to solving simulated locomotion tasks
- The policy is linear
- Normalize the rewards (scaling)
- Select the perturbations that lead to the biggest improvement

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

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**)

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

while ending condition not satisfied **do**

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

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

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

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

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

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.

Make the update step:

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

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

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]

$j \leftarrow j + 1$

end while

augmented random search

verage reward evaluated over 100 random seeds, shown by percentile

Figure 1: An evaluation of ARS over 100 random seeds on the MuJoCo locomotion tasks. The dotted lines represent median rewards and the shaded regions represent percentiles. For Swimmer-v1 we used ARS **V1**. For Hopper-v1, Walker2d-v1, and Ant-v1 we used ARS **V2-t**. For HalfCheetah-v1 and Humanoid-v1 we used ARS **V2**.

source: https://www.argmin.net/2018/03/20/mujocoloco/

Cross-entropy method

- Evolutionary algorithm (related to CMA-ES)
- Define a proposal distribution over the parameters $\boldsymbol{\theta}$

$$\mathbb{P}(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \qquad \boldsymbol{\Sigma} = \boldsymbol{\sigma}^2 \boldsymbol{I}$$

- Generate trajectories according to $\mathbb{P}(\boldsymbol{\theta})$
- Rank them according to their return
- Use the $\boldsymbol{K}$ best performing ones to refit $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$

animation: Brodie Lawson

Bottomline of gradient-free methods

- **easy** to implement, there are often good **baselines**
- 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

- **bottomline of policy search:** casting RL into an optimization problem and then apply your favorite optimization procedure!

Closing

References

- **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)*
- **Dynamic Programming and Optimal Control** Dimitri P. Bertsekas
  http://athenasc.com/dpbook.html

Questions?