Reinforcement learning

Episode 1

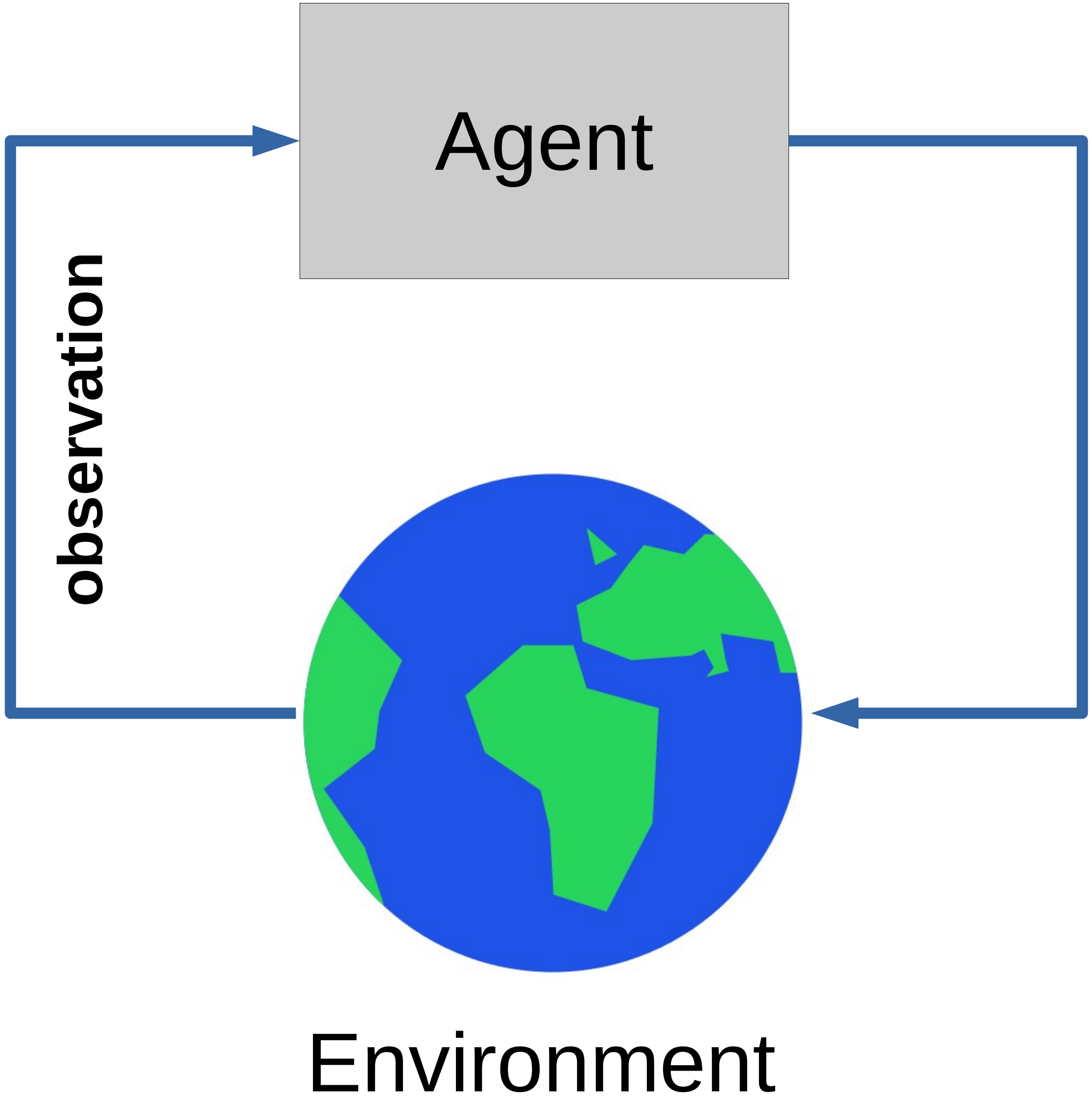
Black box optimization

Yandex
Data Factory

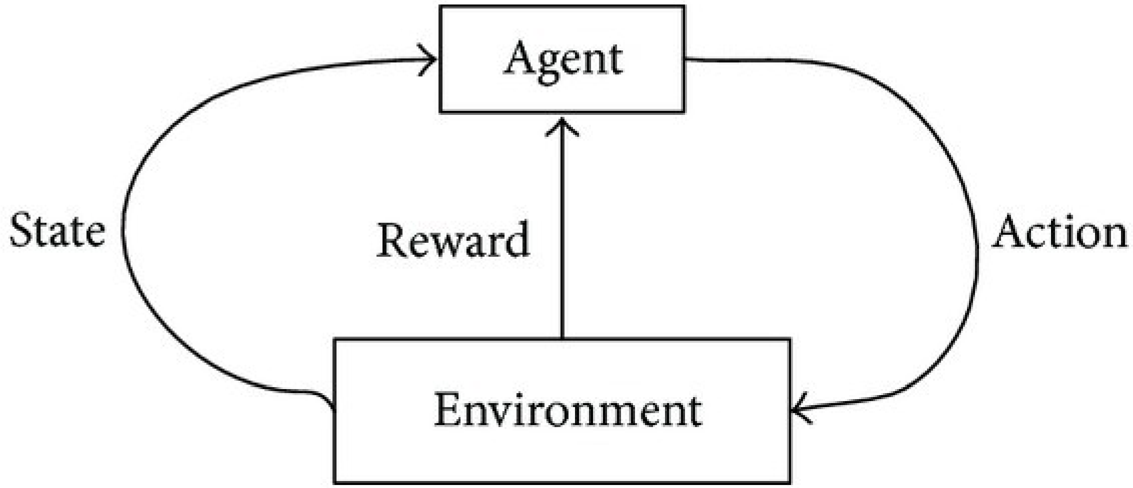
LAMBDA

British Hedgehog
Preservation Society

Recap: reinforcement learning



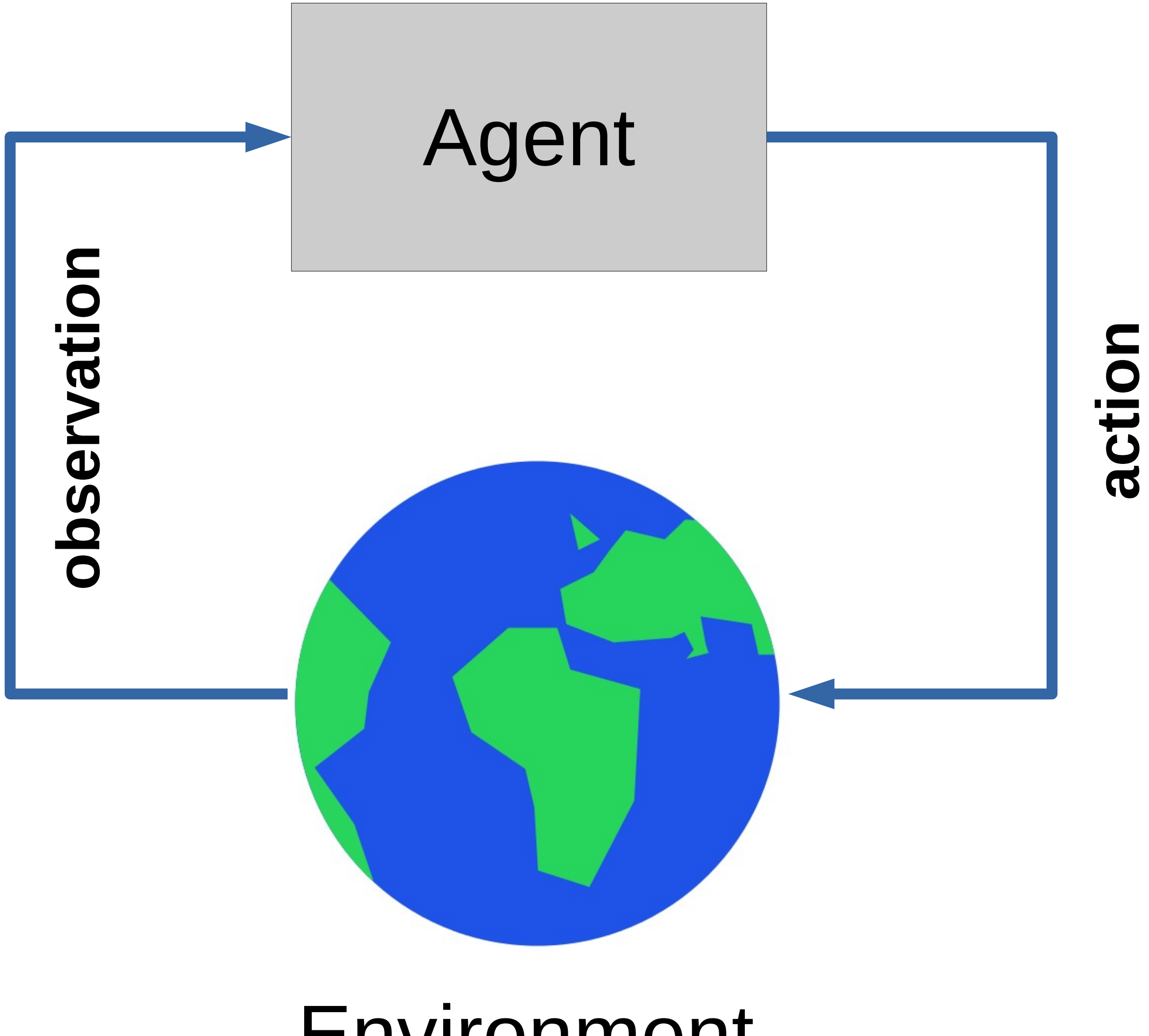
Recap: MDP



Classic MDP(Markov Decision Process)
Agent interacts with environment

- Environment states: $s \in S$
- Agent actions: $a \in A$
- State transition: $P(s_{t+1} | s_t, a_t)$

Recap: reinforcement learning



Feedback (Monte-Carlo)

- Naive objective: $R(z)$

$$z=[s_0, a_0, s_1, a_1, s_2, a_2, \ldots, s_n, a_n]$$

Deterministic policy:

- Find policy with highest expected reward

$$\pi(s) \rightarrow a : E[R] \rightarrow max$$

Feedback (Monte-Carlo)

Whole session

- Naive objective: $R(z)$

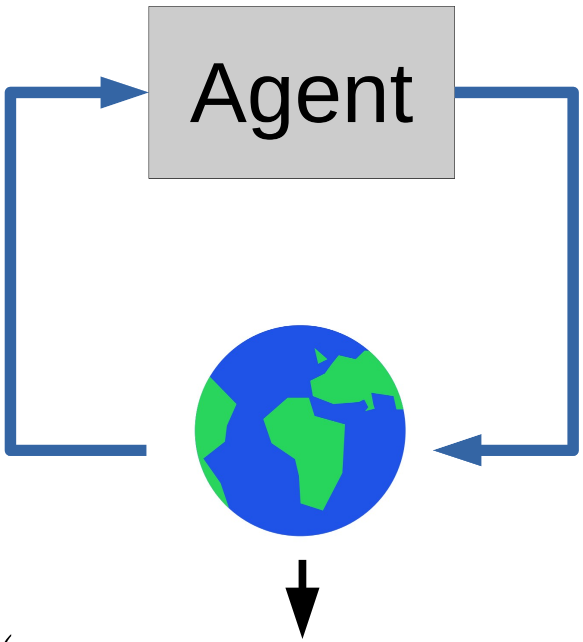
$$z = [s_0, a_0, s_1, a_1, s_2, a_2, \ldots, s_n, a_n]$$

Deterministic policy:

- Find policy with highest expected reward

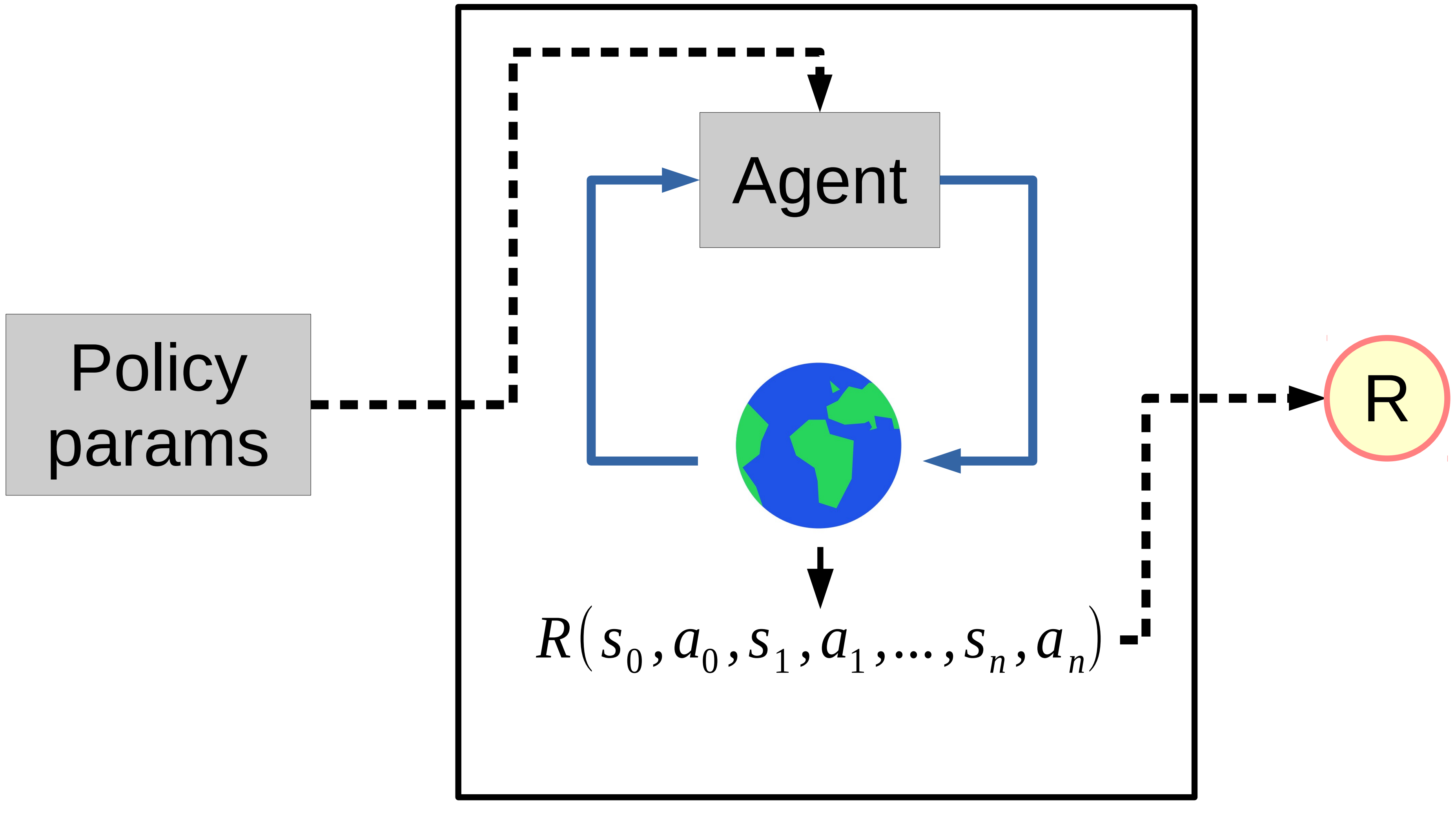
$$\pi(s) \rightarrow a : E[R] \rightarrow max$$

Black box optimization setup

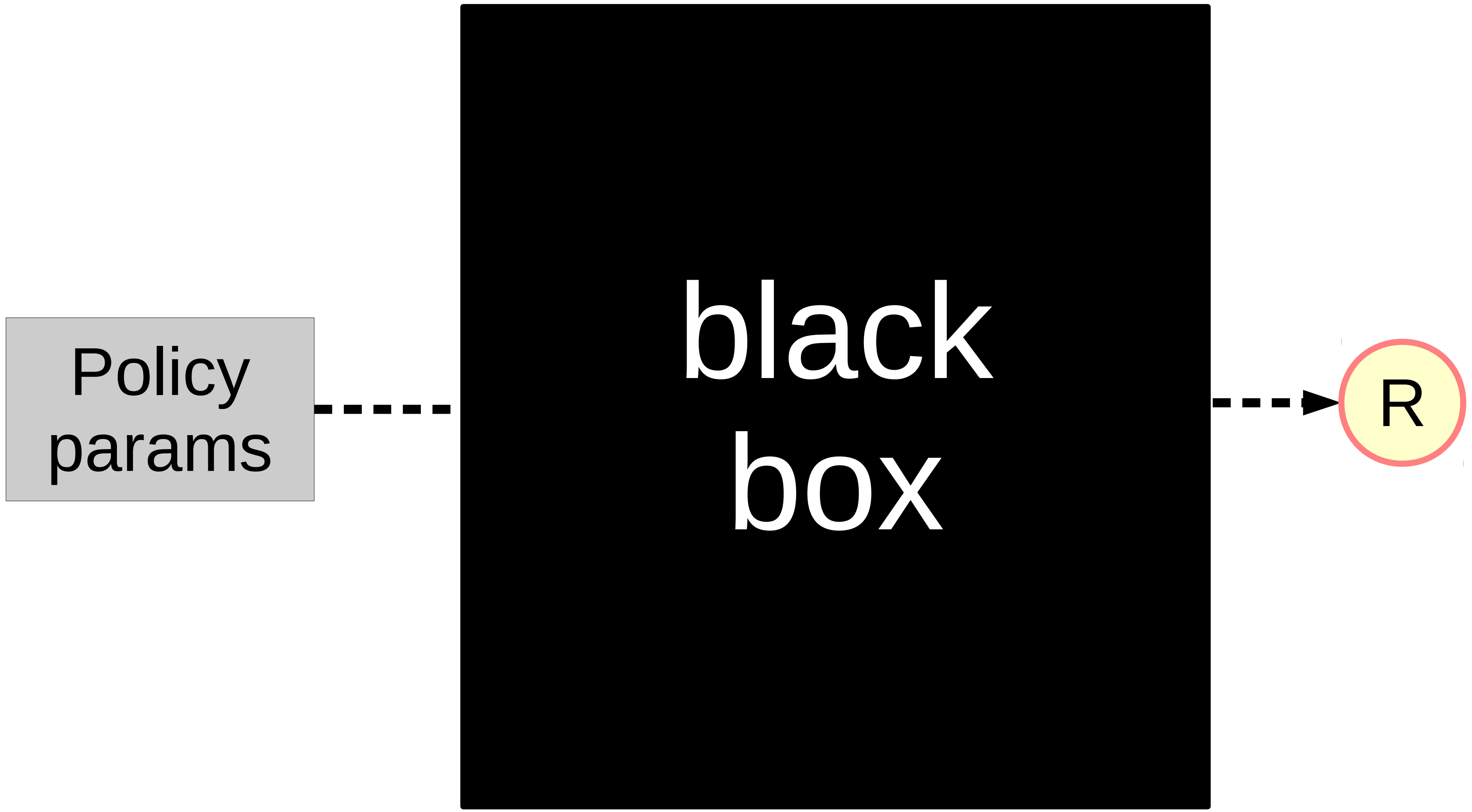


$$R(s_0, a_0, s_1, a_1, \ldots, s_n, a_n)$$

Black box optimization setup



Black box optimization setup



Today's menu

Evolution strategies

- A general black box optimization
- Easy to implement & scale

Crossentropy method

- A general method with special case for RL
- Works remarkably well in practice

Evolution Strategies

– Introduce distribution over weights

$$\theta \sim N\left(\theta | \mu, \sigma^2\right)$$

– Maximize expected reward

$$J = \underset{N\left(\theta | \mu, \sigma^2\right)}{E} R$$

Evolution Strategies

– Introduce distribution over weights

$$\theta \sim N\left(\theta|\mu,\sigma^2\right)$$

other P(θ) will work as well

– Maximize expected reward

$$J = \mathop{E}_{N\left(\theta|\mu,\sigma^2\right)} R$$

Evolution Strategies

- Introduce distribution over weights

$$\theta \sim N\left(\theta \mid \mu, \sigma^2\right)$$

- Maximize expected reward

$$J = \underset{N\left(\theta \mid \mu, \sigma^2\right)}{E} \quad \underset{s, a, s', a', \ldots}{E} \quad R\left(s, a, s', a', \ldots\right)$$

Evolution Strategies

– Expected reward (using math. Expectation)

$$J = \underset{N(\theta|\mu,\sigma^2)}{E} \quad \underset{s,a,s',a',\ldots}{E} \quad R(s,a,s',a',\ldots)$$

Q: How can we estimate J in practice?

for large/infinite state space

Evolution Strategies

– Expected reward (using math. Expectation)

$$J = \underset{N(\theta|\mu,\sigma^2)}{E} \quad \underset{s,a,s',a',\ldots}{E} R(s,a,s',a',\ldots)$$

– Approximate with sampling

$$J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in z_i} R(s,a,\ldots)$$

Sample Θ from $\theta \sim N(\theta|\mu,\sigma^2)$

Evolution Strategies

- Expected reward (using math. Expectation)

$$J = \underset{N(\theta|\mu,\sigma^2)}{E} \; \underset{s,a,s',a',\dots}{E} R(s,a,s',a',\dots)$$

- Approximate with sampling

reward for session

$$J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in z_i} R(s,a,\dots)$$

Sample Θ from $\theta \sim N(\theta|\mu,\sigma^2)$

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

- What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

Evolution Strategies

– Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

– What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

Q: Can we estimate ∇J with samples?

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

- What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(s, a, s', a', \ldots) R(s, a, s', a', \ldots)$$

Not a valid expectation
∇ N (Θ|μ,σ) is not a distribution

Logderivative trick

Simple math

$$\nabla \log f(x) = ???$$

(try chain rule)

Logderivative trick

Simple math

$$\nabla \log f(x) = \frac{1}{f(x)} \cdot \nabla f(x)$$

$$f(x) \cdot \nabla \log f(z) = \nabla f(z)$$

Logderivative trick

Analytical inference

$$\nabla J = \int \nabla \left[N(\theta|\mu,\sigma^2) \right] \cdot \int P(s,a,s',a',\ldots) R(s,a,s',a',\ldots)$$

$$\nabla N(\theta|\mu,\sigma^2) = N(\theta|\mu,\sigma^2) \cdot \nabla \log N(\theta|\mu,\sigma^2)$$

Evolution Strategies

Analytical inference

$$\nabla J = \int \left[N(\theta|\mu, \sigma^2) \cdot \nabla \log N(\theta|\mu, \sigma^2) \right] \underset{s,a,\dots}{E} R(s, a, \dots)$$

Q: How can we estimate ∇J in now?

Evolution Strategies

Analytical inference

$$\nabla J = \int \left[N(\theta|\mu,\sigma^2) \cdot \nabla \log N(\theta|\mu,\sigma^2) \right] \underset{s,a,\dots}{E} R(s,a,\dots)$$

Sampled estimate

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log N(\theta|\mu,\sigma^2) \cdot \sum_{s,a \in z_i} R(s,a,\dots)$$

Sample Θ from $\theta \sim N(\theta|\mu,\sigma^2)$

Evolution strategies

Algorithm

1. Initialize μ_0, σ_0^2

2. Forever:

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log N(\theta | \mu, \sigma^2) \cdot \sum_{s,a \in z_i} R(s, a, \ldots)$$

$$\mu = \mu + \alpha \cdot \nabla J \qquad \sigma^2 = \sigma^2 + \alpha \cdot \nabla J$$

Evolution strategies

Features

- A general black box optimization
- Needs a lot of samples
- **Easy to implement & scale**

Evolution strategies

Features

- A general black box optimization
- **Easy to implement & scale**

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log N(\theta|\mu, \sigma^2) \cdot \sum_{s,a \in z_i} R(s, a, \ldots)$$

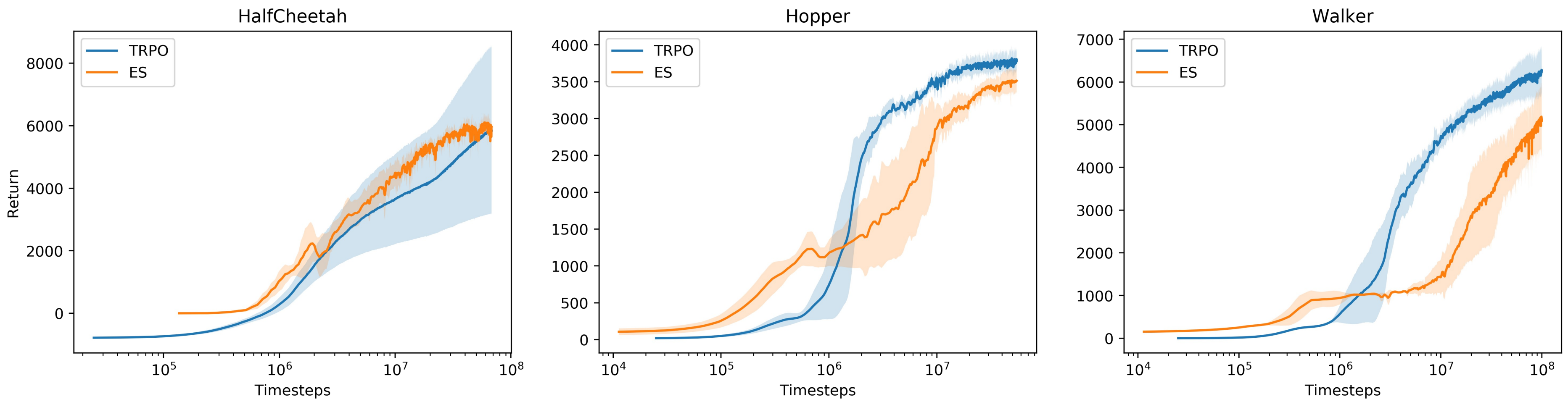
Q: You have 1000 CPUs.

Optimize this formula!

Evolution strategies

Features

- A general black box optimization
- Some results on gym



https://blog.openai.com/evolution-strategies/

Today's menu

Evolution strategies

- A general black box optimization
- Requires a lot of sampling

Crossentropy method

- A general method with special case for RL
- Works remarkably well in practice

Estimation problem

- You want to estimate

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x)\, dx$$

Estimation is not a problem

- You want to estimate

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x)\, dx$$

- So what? You just compute it!

Estimation problem

- You want to estimate

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x)\, dx$$

- So what? You just compute it!
  - **x** may be 1000-dimensional
  - **H(x)** may be costly to compute

Ideas?

Estimation in the wild

- You want to estimate

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x) dx$$

- So what? You just compute it!
  - **x** may be 1000-dimensional
  - **H(x)** may be costly to compute

$$\int_x p(x) \cdot H(x) dx \approx \frac{1}{N} \sum_{x_k \sim p(x)} H(x_k)$$

Estimation in the wild

- You want to estimate **profits!**

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x) dx$$

- **x** – user of your online game (age, gender, ...)
- **p(x)** – probability of such user
- **H(x)** – try to guess :)

Estimation in the wild

- You want to estimate **profits!**

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x) dx$$

- **x** – user of your online game (age, gender, ...)
- **p(x)** – probability of such user
- **H(x)** – money donated by such user

Estimation in the wild

- Sampling = asking users to pass survey
- Usually costs money!
- Guess **H(median russian gamer)**?

Estimation in the wild

- Sampling = asking users to pass survey
- Usually costs money!
- H(median russian gamer) ~ 0
- It's H(hard-core donators) that matters!

Estimation in the wild

- Sampling = asking users to pass survey
- Usually costs money!
- Most H(x) are small, few are **very** large

Estimation in the wild

- Sampling = asking users to pass survey
- Usually costs money!
- 99% of H(x)=0, 1% H(x)=$1000 **(whale)**
- You make a survey of N=50 people

How accurate are we?

Estimation in the wild

- Sampling = asking users to pass survey
- Usually costs money!
- 99% of H(x)=0, 1% H(x)=$1000 **(whale)**
- You make a survey of N=50 people

$$\int_x p(x) \cdot H(x)\, dx \approx \frac{1}{N} \sum_{x_k \sim p(x)} H(x_k)$$

0 whales: **H**=0, 1 whale: **H**= 5x true

Importance sampling

- Idea: we know that most whales are
  - 30-40 year old
  - single
  - wage >100k
- Sample 50% in that group, 50% rest
- Adjust for difference in distributions

Importance sampling

- Math:

$$\underset{x\sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x)\, dx = \int_x p(x) \cdot \frac{q(x)}{q(x)} \cdot H(x)\, dx$$

Importance sampling

- Math:

$$\underset{x\sim p(x)}{E} H(x) = \int_x p(x)\cdot H(x)\,dx = \int_x p(x)\cdot\frac{q(x)}{q(x)}\cdot H(x)\,dx =$$

$$= \int_x q(x)\cdot\frac{p(x)}{q(x)}\cdot H(x)\,dx = \underset{x\sim q(x)}{E} ???$$

Importance sampling

- Math:

$$\underset{x \sim p(x)}{E} H(x) = \int_x p(x) \cdot H(x)\, dx = \int_x p(x) \cdot \frac{q(x)}{q(x)} \cdot H(x)\, dx =$$

$$= \int_x q(x) \cdot \frac{p(x)}{q(x)} \cdot H(x)\, dx = \underset{x \sim q(x)}{E} \frac{p(x)}{q(x)} \cdot H(x)$$

Importance sampling

- TL;DR:

$$\underset{x \sim p(x)}{E} H(x) = \underset{x \sim q(x)}{E} \frac{p(x)}{q(x)} \cdot H(x)$$

Importance sampling

- TL;DR:

$$\underset{x \sim p(x)}{E} H(x) = \underset{x \sim q(x)}{E} \frac{p(x)}{q(x)} \cdot H(x)$$

$$\frac{1}{N} \sum_{x_k \sim p(x)} H(x_k) \approx \frac{1}{N} \sum_{x_k \sim q(x)} \frac{p(x)}{q(x)} \cdot H(x)$$

Importance sampling

- TL;DR:

$$\underset{x \sim p(x)}{E} H(x) = \underset{x \sim q(x)}{E} \frac{p(x)}{q(x)} \cdot H(x)$$

If p(x)>0, then q(x)>0

$$\underset{x \sim p(x)}{E} H(x) \approx \frac{1}{N} \sum_{x_k \sim p(x)} H(x_k) \approx \frac{1}{N} \sum_{x_k \sim q(x)} \frac{p(x)}{q(x)} \cdot H(x)$$

original distribution **other distribution**

Importance sampling

- Idea: we may know that all whales are
  - 30-40 year old
  - single
  - wage >100k
- Sample **q(x):** 50% that group, 50% rest
- **Adjust** for difference in distributions

$$\frac{1}{N}\sum_{x_k \sim p(x)} H(x_k) \approx \frac{1}{N}\sum_{x_k \sim q(x)} \frac{p(x)}{q(x)} \cdot H(x)$$

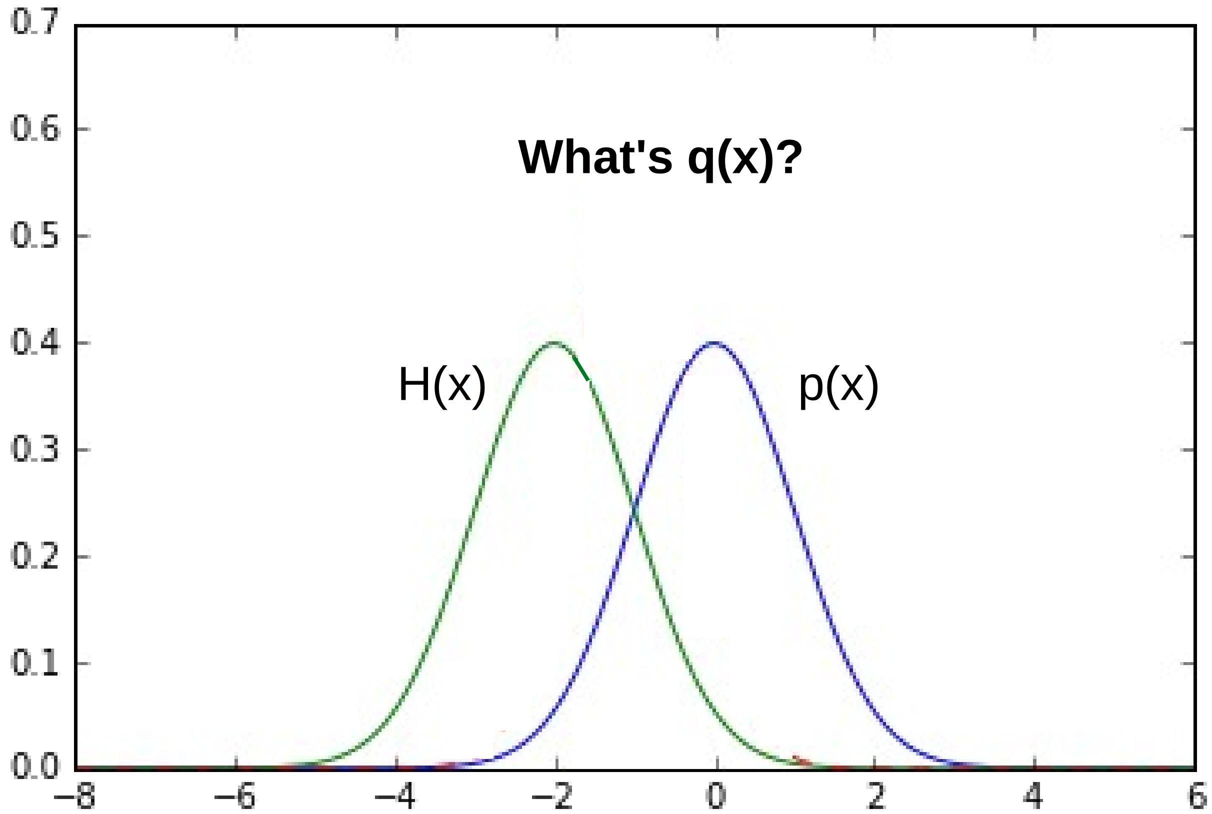
Importance sampling

- Idea: we may know that all whales are
  - 30-40 year old
  - single
  - wage >100k
- Sample from different **q(x)**
- **Adjust** for difference in distributions

Which **q(x)** is best?

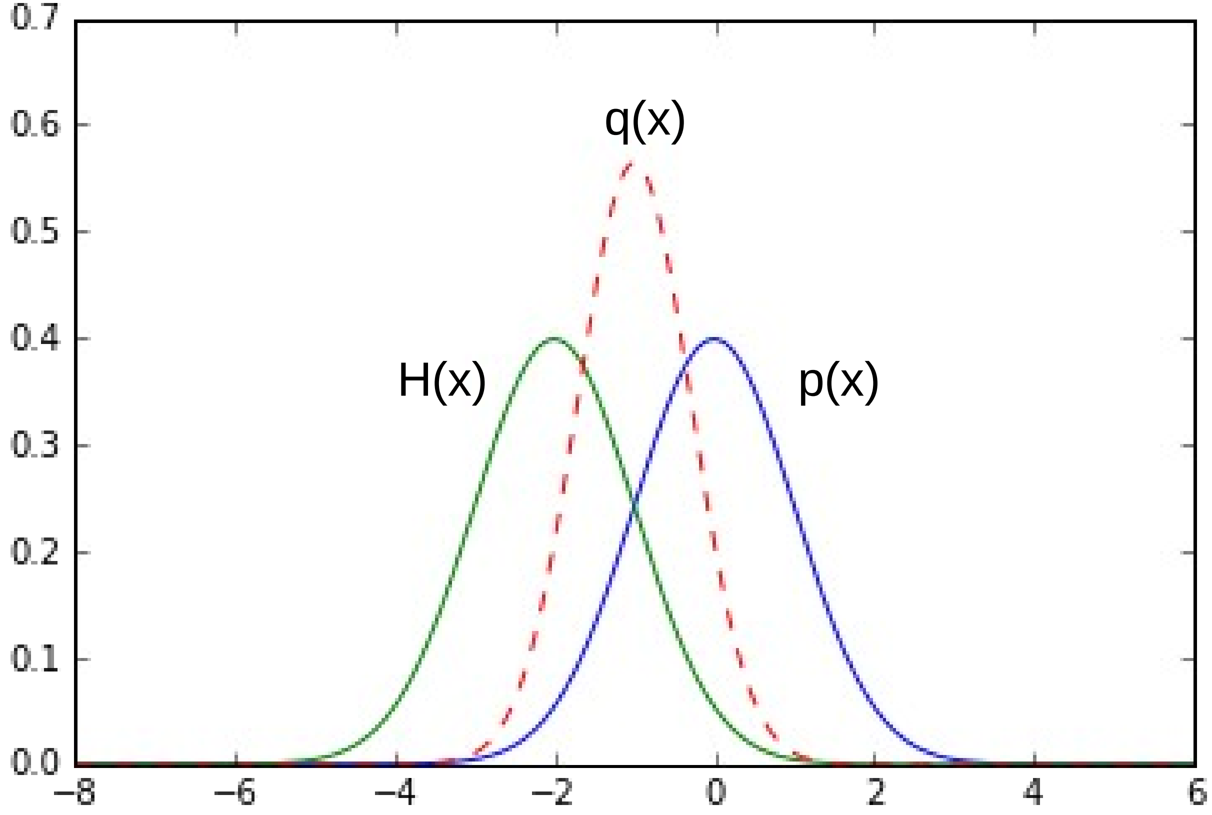
Crossentropy method

- Pick $q(x) \sim p(x) \cdot H(x)$



Crossentropy method

- Pick $q(x) \sim p(x) \cdot H(x)$



Crossentropy method

- Minimize difference between $q(x)$ and $p(x)H(x)$

- How to measure that difference? **Ideas?**

Crossentropy method

- Minimize difference between $q(x)$ and $p(x)H(x)$
- Kullback-Leibler divergence

$$KL(p_1(x) \| p_2(x)) = \mathop{E}_{x \sim p_1(x)} \log \frac{p_1(x)}{p_2(x)}$$

Crossentropy method

- Minimize difference between $q(x)$ and $p(x)H(x)$

- Kullback-Leibler divergence

$$KL(p_1(x)\|p_2(x)) = \underset{x\sim p_1(x)}{E} \log \frac{p_1(x)}{p_2(x)} =$$

$$= \underset{x\sim p_1(x)}{E} \log p_1(x) - \underset{x\sim p_1(x)}{E} \log p_2(x)$$

what? **what?**

Crossentropy method

- Minimize difference between $q(x)$ and $p(x)H(x)$

- Kullback-Leibler divergence

$$KL(p_1(x)\|p_2(x)) = \mathop{E}_{x\sim p_1(x)} \log \frac{p_1(x)}{p_2(x)} =$$

const(p2(x))

$$= \mathop{E}_{x\sim p_1(x)} \log p_1(x) - \mathop{E}_{x\sim p_1(x)} \log p_2(x)$$

entropy **crossentropy**

Crossentropy method

- Minimize difference between $q(x)$ and $p(x)H(x)$

- Minimize Kullback-Leibler divergence

$$\underset{q(x)}{argmin}[const - \underset{x \sim p(x)}{E} H(x)\log q(x)]$$

entropy **crossentropy**

Crossentropy method

- Pick q(x) to minimize **crossentropy**

$$q(x) = \underset{q(x)}{argmin}\left[- \underset{x \sim p(x)}{E} H(x) \log q(x)\right]$$

- Exact solution in many cases (e.g. gaussian)
- Otherwise use numeric optimization
  - e.g. when q(x) is a neural network

Iterative approach

- Pick q(x) to minimize **crossentropy**

$$q(x) = \underset{q(x)}{argmin}\left[-\underset{x\sim p(x)}{E} H(x)\log q(x)\right]$$

- Start with $q_0(x) = p(x)$
- Iteration

$$q_{i+1}(x) = \underset{q_{i+1}(x)}{argmin} - \underset{x\sim q_i(x)}{E} \frac{p(x)}{q_i(x)} H(x) \log q_{i+1}(x)$$

Finally, reinforcement learning

- Objective: H(x) = [R > threshold]
- p(x) = uniform
- Threshold = M'th (e.g. 50th) percentile of R

$$\pi_{i+1}(a|s) = \underset{\pi_{i+1}}{argmin} - \underset{z\sim\pi_i(a|s)}{E}[R(z) \geq \psi_i] \log \pi_{i+1}(a|s)$$

$$\psi_i = M'th\ percentile\ of\ R(z \sim \pi_i)$$

Finally, reinforcement learning

- Objective: H(x) = [R > threshold]
- p(x) = uniform
- Threshold = M'th (e.g. 50th) percentile of R

$$\pi_{i+1}(a|s) = \underset{\pi_{i+1}}{argmin} - \underset{z \sim \pi_i(a|s)}{E} [R(z) \geq \psi_i] \log \pi_{i+1}(a|s)$$

$$\psi_i = M\text{'}th\ percentile\ of\ R(z \sim \pi_i)$$

Something wrong with the formula!

Finally, reinforcement learning

- Objective: H(x) = [R > threshold]
- p(x) = uniform
- Threshold = M'th (e.g. 50th) percentile of R

$$\pi_{i+1}(a|s) = \underset{\pi_{i+1}}{argmin} - \underset{z \sim \pi_i(a|s)}{E} [R(z) \geq \psi_i] \log \pi_{i+1}(a|s)$$

No p(x)/q(x) term as it's okay to expect over pi(a|s)

$$\psi_i = M\text{'}th\ percentile\ of\ R(z \sim \pi_i)$$

TL;DR, simplified

- Sample N=100 sessions
- Take M=25 best
- Fit policy to behave as in M best sessions
- Repeat until satisfied

Policy will gradually get better.

Tabular crossentropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Get M best games (highest reward)
- Contatenate, K state-action pairs total

$$Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$$

Tabular crossentropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Take M best (highest reward)
- Aggregate by states

$$\pi(a|s) = \frac{\sum_{s_t, a_t \in Elite} [s_t = s][a_t = a]}{\sum_{s_t, a_t \in Elite} [s_t = s]}$$

Tabular crossentropy method

- Policy is a matrix

$$\pi(a|s) = A_{s,a}$$

- Sample N games with that policy
- Take M best (highest reward)
- Aggregate by states

$$\pi(a|s) = \frac{took\, a\, at\, s}{was\, at\, s}$$

In M best games

Smoothing

- If you were in some state only once, you only take this action now.
- Apply smoothing

$$\pi(a|s) = \frac{[took\ a\ at\ s] + \lambda}{[was\ at\ s] + \lambda \cdot N_{actions}}$$

In M best games

Alternative idea: smooth updates

$$\pi_{i+1}(a|s) = \alpha \cdot \pi_{opt} + (1-\alpha)\pi_i(a|s)$$

Stochastic MDPs

- If there's randomness in environment, algorithm will prefer “lucky” sessions.
- Training on lucky sessions is no good

- Solution: sample action for each state and run several simulations with these state-action pairs. Average the results.

Approximate (deep) version

- Policy is approximated
  - Neural network predicts $\pi_W(a|s)$ given s
  - Linear model / Random Forest / ...

Can't set $\pi(a|s)$ explicitly

All state-action pairs from M best sessions

$$Elite=[(s_0,a_0),(s_1,a_1),(s_2,a_2),...,(s_k,a_k)]$$

Approximate (deep) version

Neural network predicts $\pi_W(a|s)$ given s

All state-action pairs from M best sessions

$$Elite=[(s_0,a_0),(s_1,a_1),(s_2,a_2),\ldots,(s_k,a_k)]$$

Maximize likelihood of actions in “best” games

$$\pi=\underset{\pi}{argmax}\sum_{s_i,a_i\in Elite}\log\pi(a_i|s_i)$$

Approximate (deep) version

Neural network predicts $\pi_W(a|s)$ given s

All state-action pairs from M best sessions

$$best=[(s_0,a_0),(s_1,a_1),(s_2,a_2),...,(s_K,a_K)]$$

Maximize likelihood of actions in “best” games

conveniently,

nn.fit(elite_states,elite_actions)

Approximate (deep) version

Approximate (deep) version

- Initialize NN weights $W_0 \leftarrow random$

- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $W_{i+1} = W_i + \alpha \nabla \left[\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i | s_i) \right]$

Approximate (deep) version

- Initialize NN weights $W_0 \leftarrow random$

model = MLPClassifier()

- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $$W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i | s_i)]$$

model.fit(elite_states,elite_actions)

Continuous action spaces

- Continuous state space
- Model $\pi_W(a|s) = N(\mu(s), \sigma^2)$
  - Mu(s) is neural network output
  - Sigma is a parameter or yet another network output
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i|s_i)]$

What changed?

Continuous action spaces

- Continuous state space **model = MLPRegressor()**
- Model $\pi_W(a|s) = N(\mu(s), \sigma^2)$
  - Mu(s) is neural network output
  - Sigma is a parameter or yet another network output
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i|s_i)]$

**model.fit(elite_states,
elite_actions)**

Nothing!

Tricks

- Remember sessions from 3-5 past iterations
  - Threshold and use all of them when training
  - May converge slower if env is easy to solve.

- Regularize with entropy
  - to prevent premature convergence.

- Parallelize sampling
- Use RNNs if partially-observable

Monte-carlo: upsides

- Great for short episodic problems
- Very modest assumptions
  - Easy to extend to continuous actions, partial observations and more

Monte-carlo: downsides

- Need full session to start learning
- Require a lot of interaction
  - A lot of crashed robots / simulations

Gonna fix that next lecture!

- Need full s o start learning
- Require a eraction
  - A lot of crashed robots / simulations

Seminar