Reinforcement learning
Episode 0

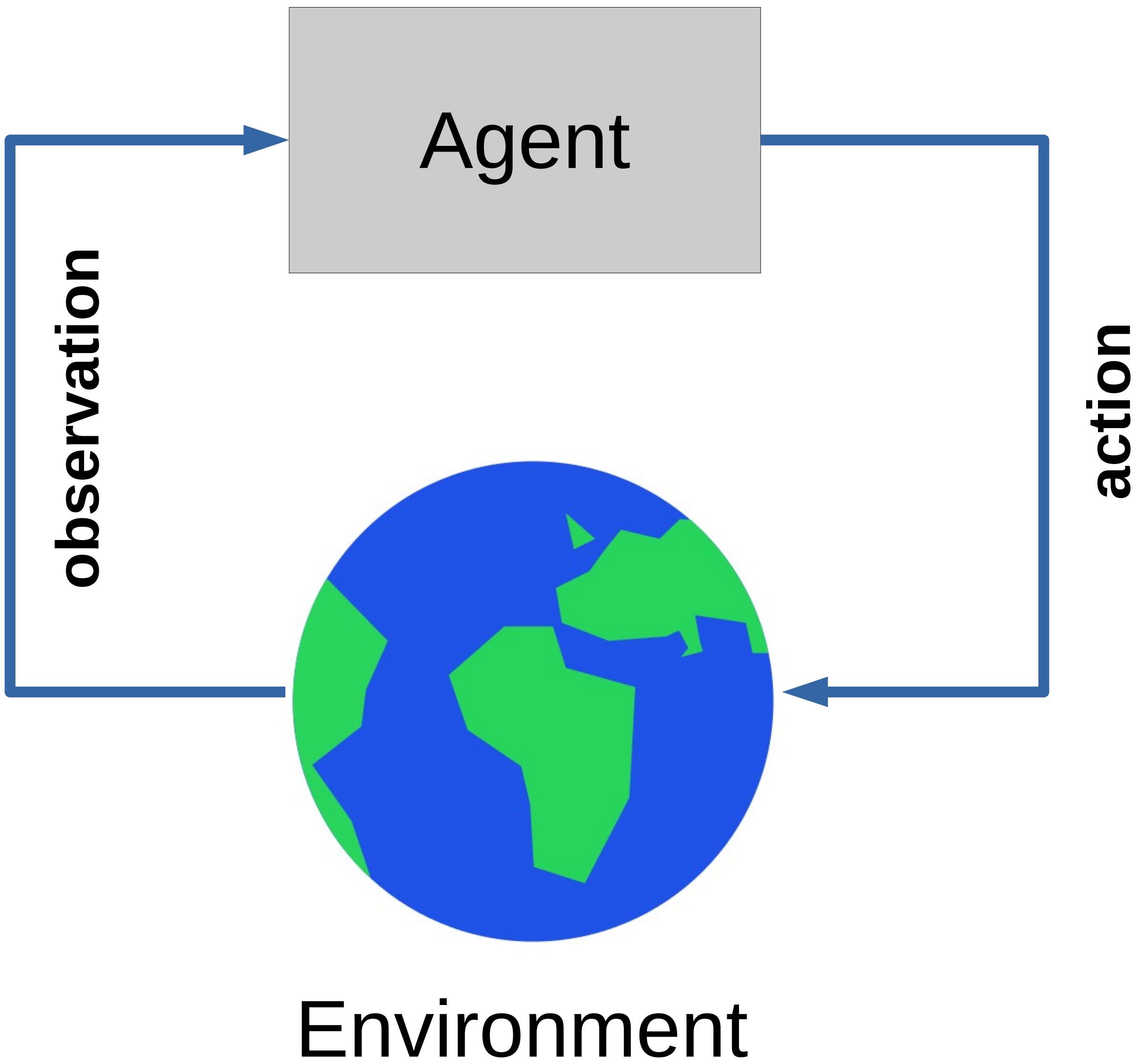
Monte-carlo methods

Yandex
Data Factory

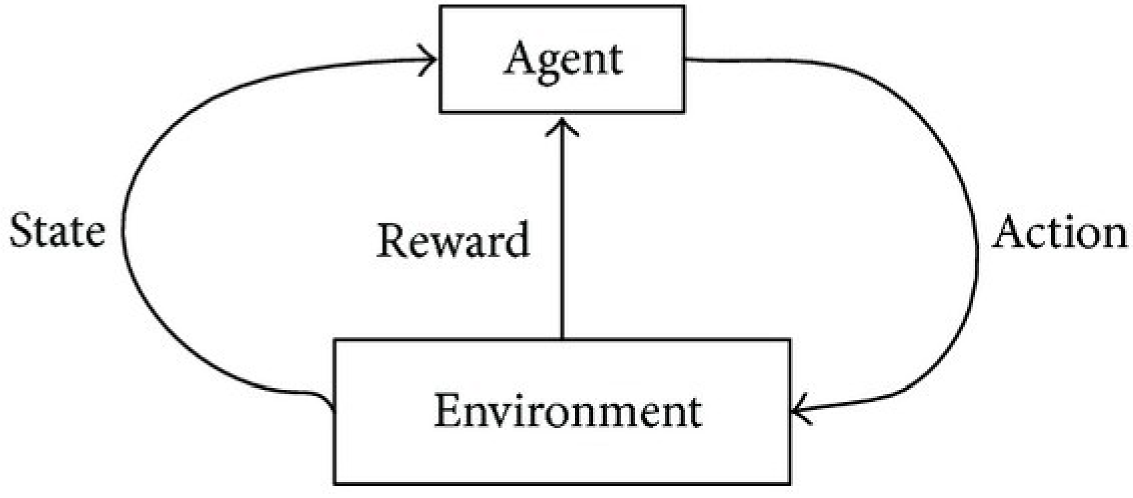
LAMBDA

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: $P(s_{t+1} | s_t, a_t)$

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

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

Combinatorial optimization

- Maximize score over policy
- No gradient
- Naive solution: iterate over all policies
- Heuristics:
  - Genetic Algorithm, differential evolution, etc.
  - Ant Colony Algorithms

Crossentropy method

- Stochastic optimization
- Not specific to RL

Crossentropy method

- Stochastic optimization
- Not specific to RL
- That's enough bullet-points!

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 100-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 100-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!**

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

- **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:

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

Importance sampling

- TL;DR:

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

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

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?

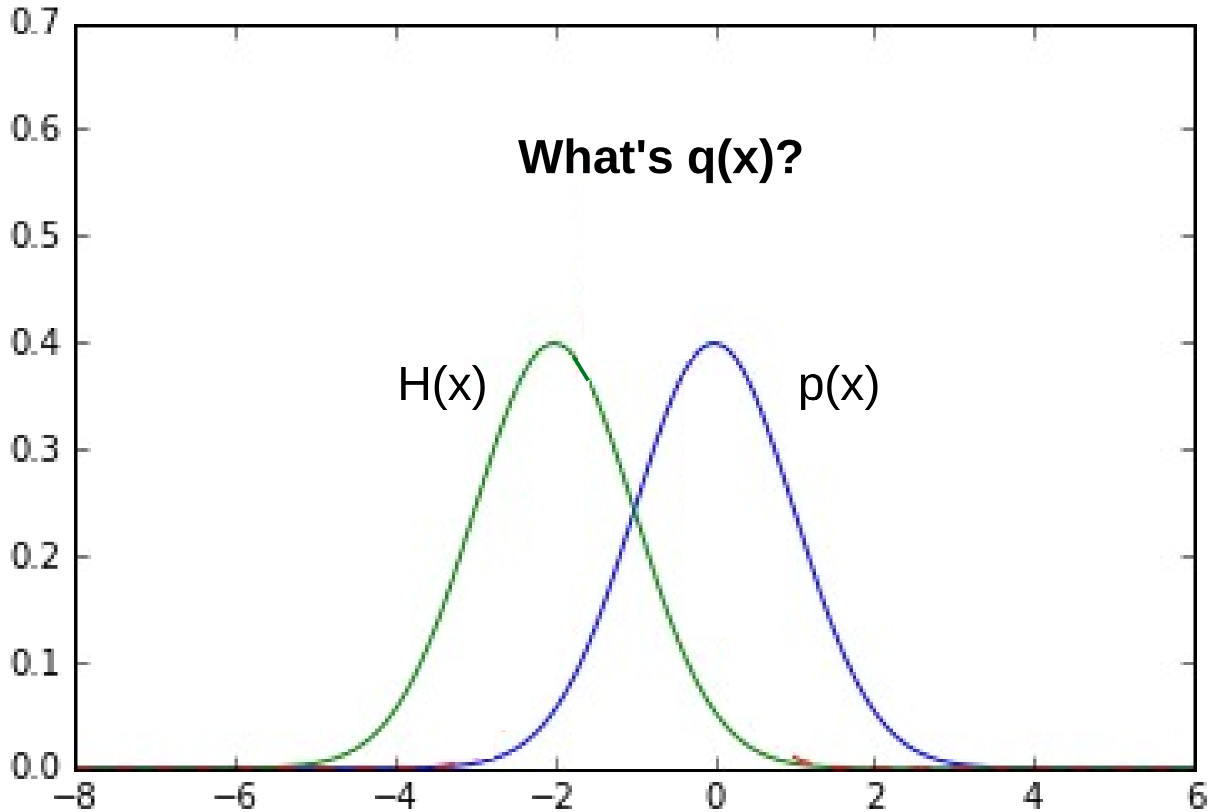
Importance sampling

- 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?

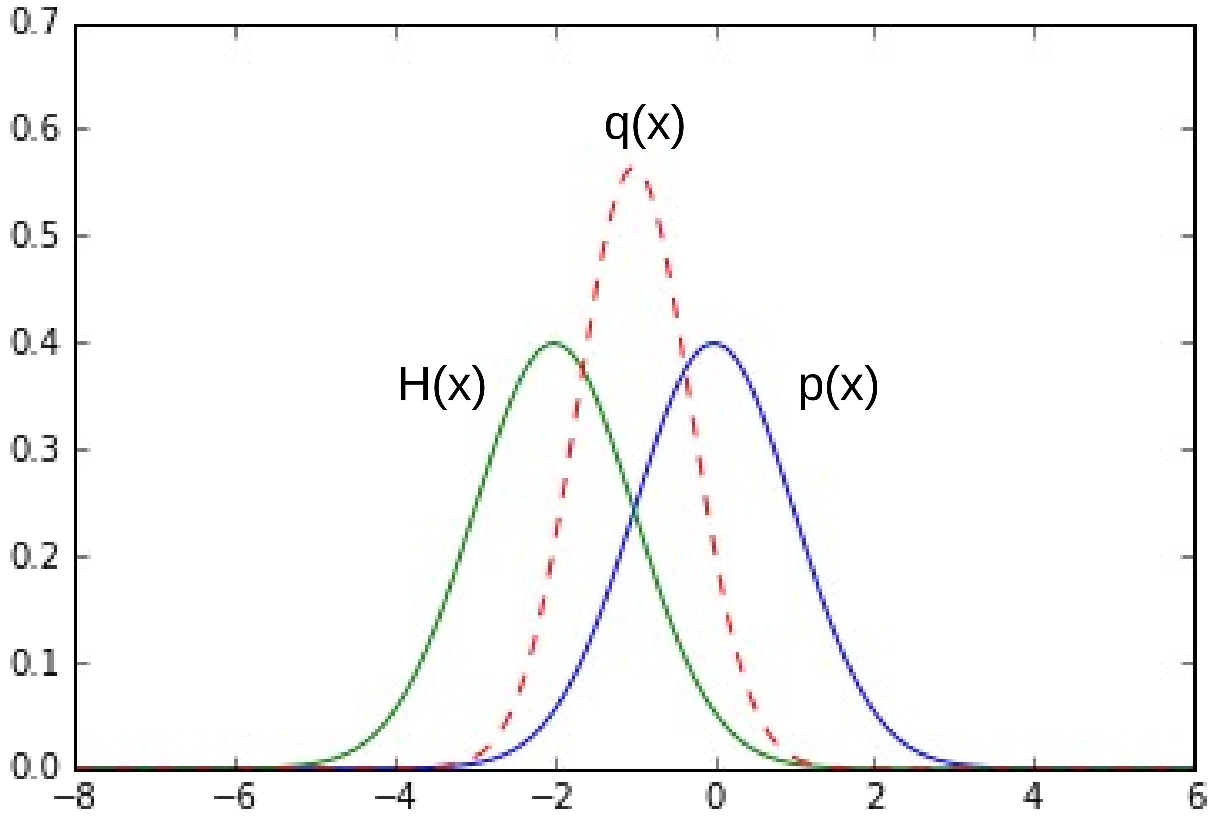
Crossentropy method

- 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 $p(x)H(x)$

- Any ideas on **how to measure difference?**

Crossentropy method

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

- Kullback-Leibler divergence

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

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

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

const(p2(x))

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

entropy **crossentropy**

Crossentropy method

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

- Minimize Kullback-Leibler divergence

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

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

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

Finally, reinforcement learning

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

$$\pi_{i+1} = \underset{\pi_{i+1}}{argmin} - \underset{x \sim \pi_i}{E} \frac{1}{\pi_i} [R_\pi \geq \psi_i] \log \pi_{i+1}$$

$$\psi_i = M\,'th\ percentile\ of\ R_{\pi_i}$$

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
- 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

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

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

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 crossentropy method

- Initialize NN weights $W_0 \leftarrow random$
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i | s_i)]$

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 $\pi_W(a|s) = N(\mu(s), \sigma^2)$ **MLPRegressor**
  - 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
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i|s_i)]$

MLPRegressor.fit(s,a)

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

Seminar