

RL @ PicsArt

Episode 1

Black box methods



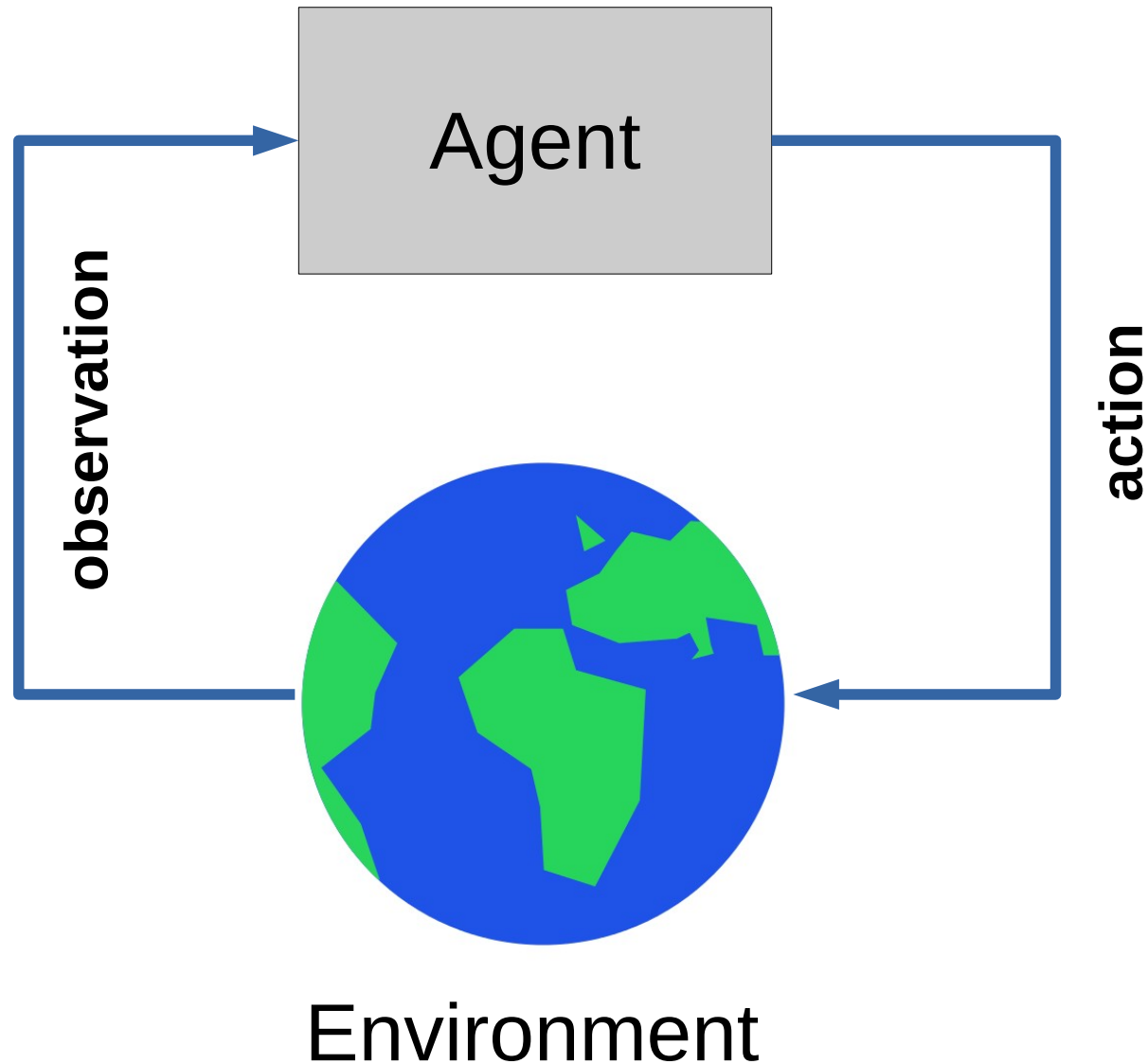
Yandex
Data Factory

LAMBDA 

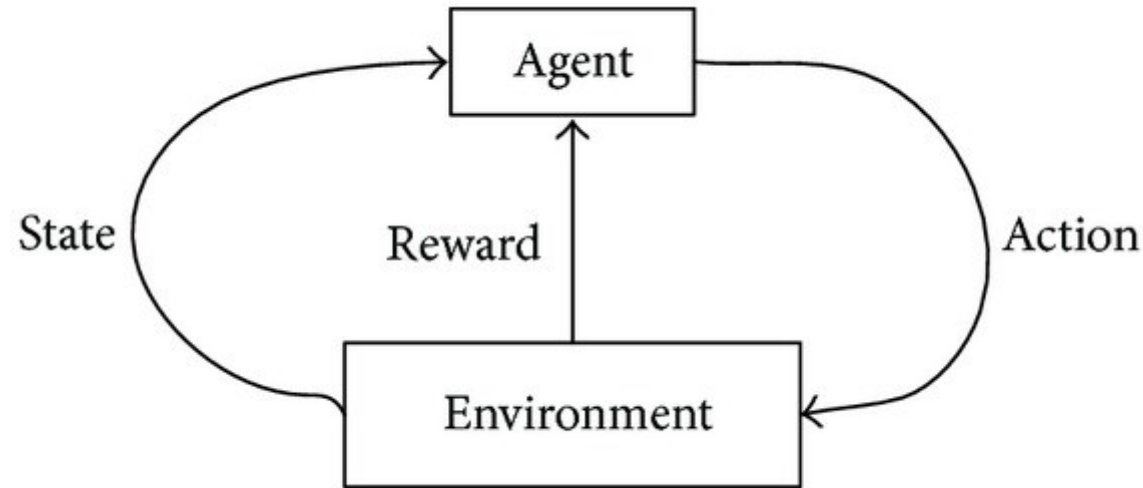


**British Hedgehog
Preservation Society**

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: $R(z)$

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

Deterministic policy:

- Find policy with highest expected reward

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

Feedback (Monte-Carlo)



Whole session

- Naive objective: $R(z)$

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

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

Today's menu

Crossentropy method

- Stochastic optimization (not specific to RL)
- Works remarkably well in practice

Monte-carlo policy gradient

RL-specific (almost)

Hopefully...

Estimation problem

- You want to estimate

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

Estimation is not a problem

- You want to estimate

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

- So what? You just compute it!

Estimation problem

- You want to estimate

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

- So what? You just compute it!
 - \mathbf{x} may be 100-dimensional
 - $\mathbf{H}(\mathbf{x})$ may be costly to compute

Ideas?

Estimation in the wild

- You want to estimate

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

- So what? You just compute it!
 - \mathbf{x} may be 100-dimensional
 - $\mathbf{H}(\mathbf{x})$ 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!**

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

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

Estimation in the wild

- You want to estimate **profits!**

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

- **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!
- $H(\text{median russian gamer}) \sim 0$
- It's $H(\text{hard-core donators})$ 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: $\mathbf{H}=0$, 1 whale: $\mathbf{H}= 5x$ 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:

$$E_{x \sim p(x)} H(x) = \int p(x) \cdot H(x) dx = \int p(x) \cdot \frac{q(x)}{q(x)} \cdot H(x) dx$$

Importance sampling

- Math:

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

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

Importance sampling

- Math:

$$E_{x \sim p(x)} H(x) = \int p(x) \cdot H(x) dx = \int p(x) \cdot \frac{q(x)}{q(x)} \cdot H(x) dx =$$

$$= \int q(x) \cdot \frac{p(x)}{q(x)} \cdot H(x) dx = E_{x \sim q(x)} \frac{p(x)}{q(x)} \cdot H(x)$$

Importance sampling

- TL;DR:

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

Importance sampling

- TL;DR:

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

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

Importance sampling

- TL;DR:

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

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

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

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

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

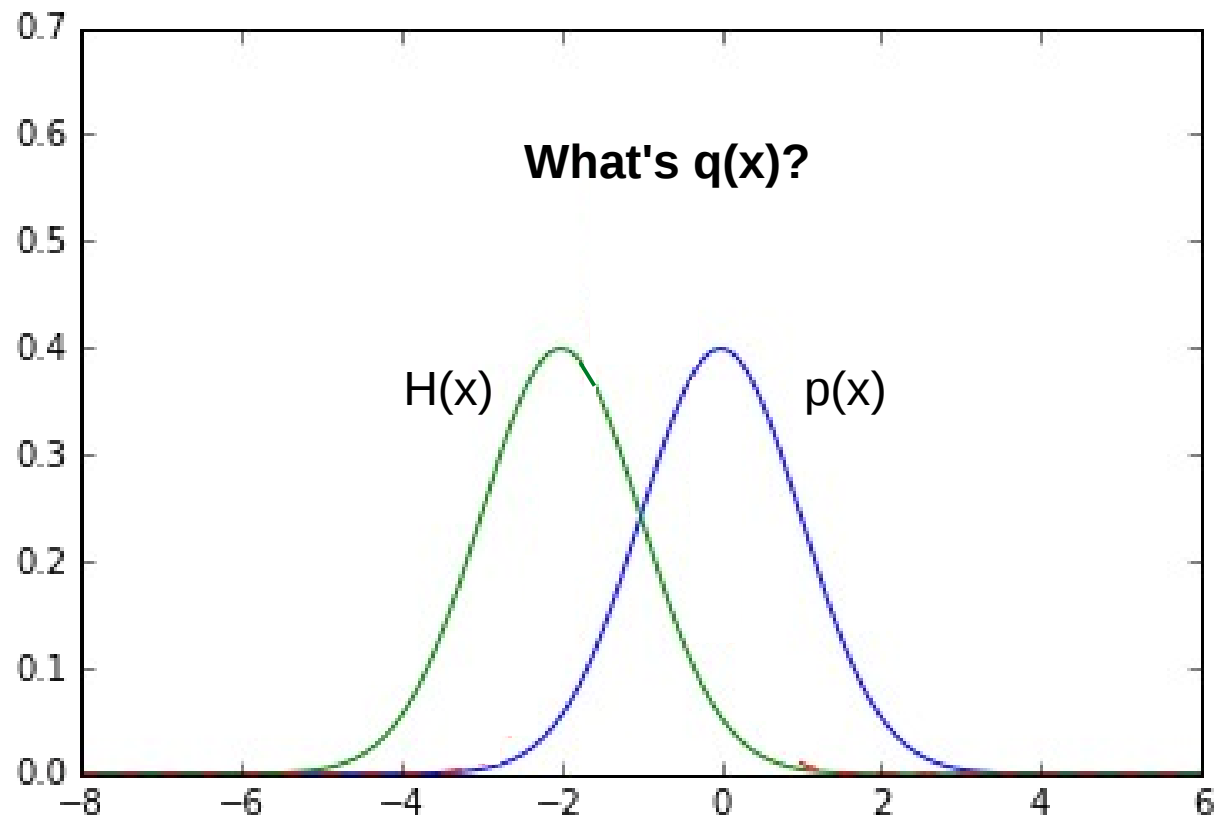
Importance sampling

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

Which $q(\mathbf{x})$ is best?

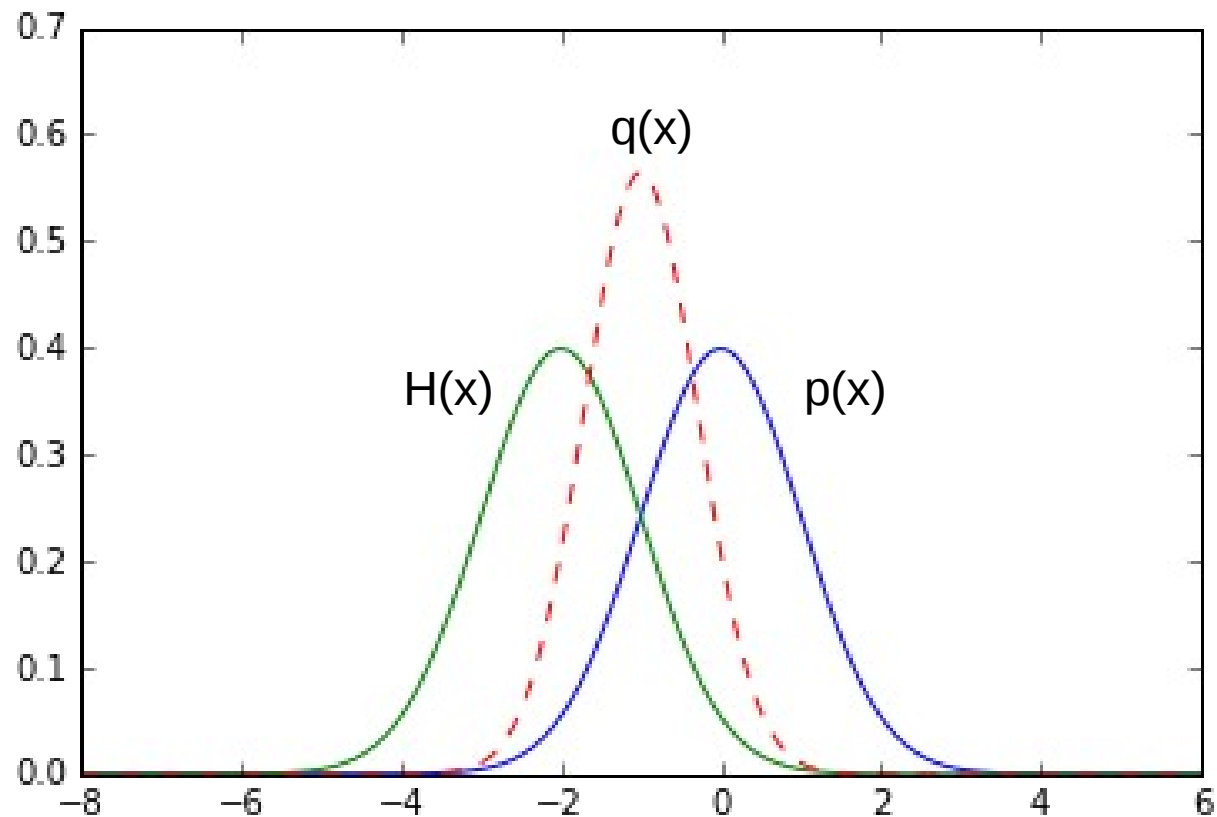
Crossentropy method

- 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

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

Crossentropy method

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

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

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

what?

what?

Crossentropy method

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

$$\begin{aligned}
 KL(p_1(x) \parallel p_2(x)) &= E_{x \sim p_1(x)} \log \frac{p_1(x)}{p_2(x)} = \\
 &= \text{const}(p_2(x)) + E_{x \sim p_1(x)} \log p_1(x) - E_{x \sim p_1(x)} \log p_2(x) \\
 &\quad \uparrow \qquad \qquad \qquad \uparrow \\
 &\quad \text{entropy} \qquad \qquad \text{crossentropy}
 \end{aligned}$$

Crossentropy method

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

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

↑
entropy

↑
crossentropy

Crossentropy method

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

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

- 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**

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

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

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

Finally, reinforcement learning

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

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

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

Finally, reinforcement learning

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

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

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

Something wrong with the formula!

Finally, reinforcement learning

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

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

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

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

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

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

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

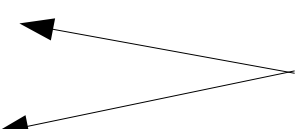
$$\pi(a|s) = \frac{\sum_{s_t, a_t \in Elite} [s_t = s][a_t = a]}{\sum_{s_t, a_t \in Elite} [s_t = s]}$$

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

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


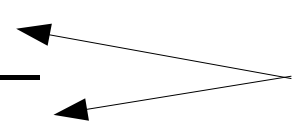
In M best games

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

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

Approximate (deep) version

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

All state-action pairs from M best sessions

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

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

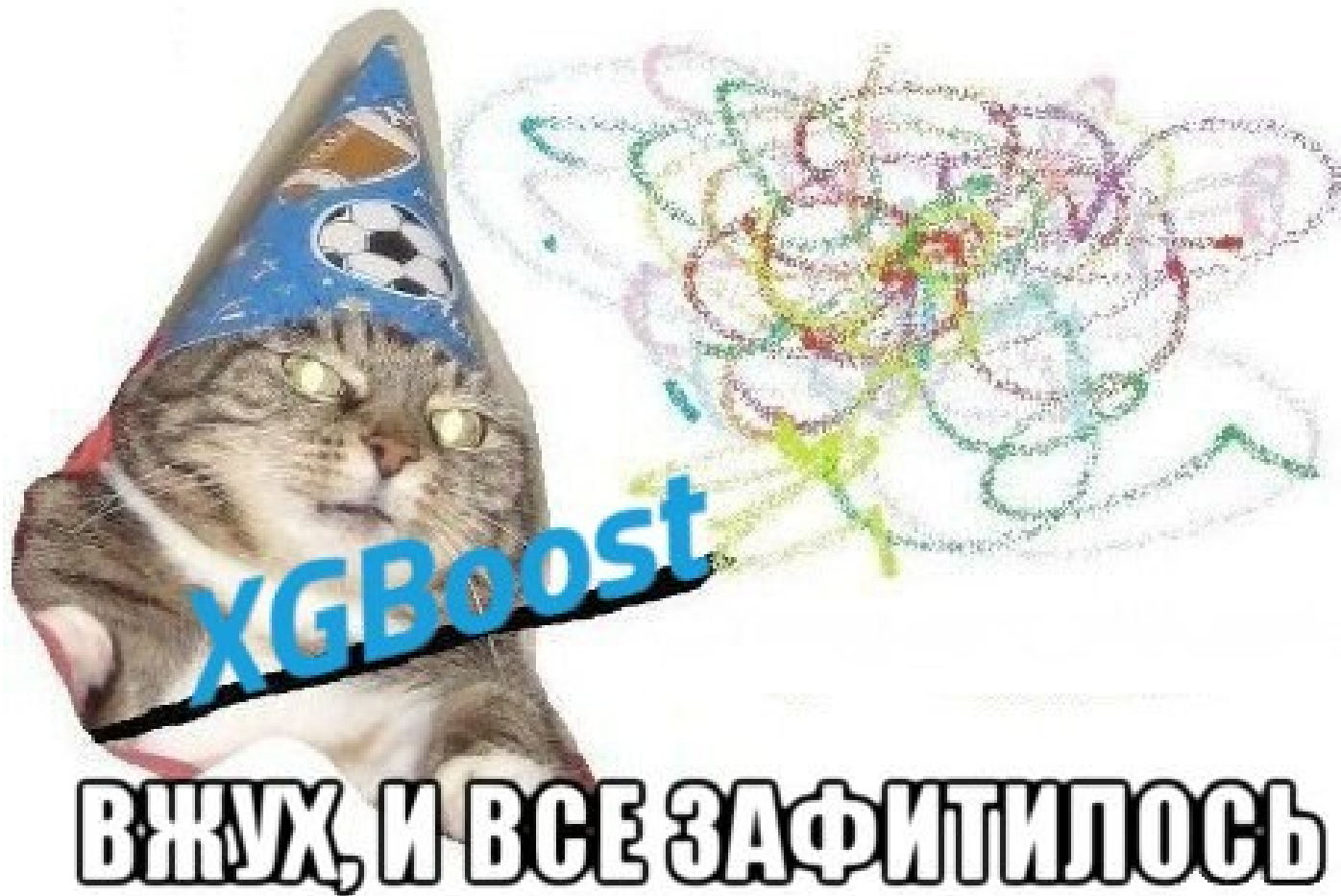
All state-action pairs from M best sessions

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

Maximize likelihood of actions in “best” games
conveniently,

$$nn.fit(elite_states, elite_actions)$$

Approximate (deep) version



Approximate (deep) version

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

Approximate (deep) version

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

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
 - σ is a parameter or yet another network output
- Loop:
 - Sample N sessions
 - elite = take M best sessions and concatenate
 - $$W_{i+1} = W_i + \alpha \nabla \left[\sum_{s_i, a_i \in \text{Elite}} \log \pi_{W_i}(a_i | s_i) \right]$$

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
 - σ is a parameter or yet another network output
- Loop:
 - Sample N sessions
 - elite = take M best sessions and concatenate
 - $$W_{i+1} = W_i + \alpha \nabla \left[\sum_{s_i, a_i \in \text{Elite}} \log \pi_{W_i}(a_i | s_i) \right]$$

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

Nothing!

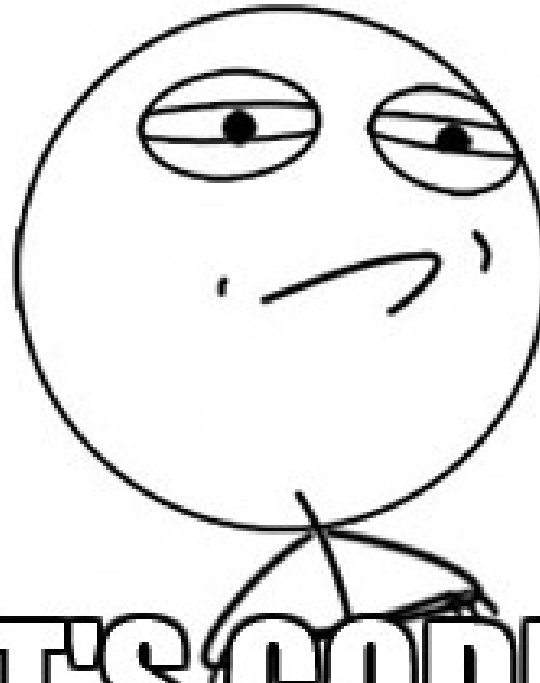
Tricks

- 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



Seminar

CHALLENGE ACCEPTED



LET'S CODE IT

memegenerator.net

Bonus round!

Crossentropy method

- Stochastic optimization
- Not specific to RL
- Has theoretical issues (stochastic MDP)

Policy gradient (monte-carlo)

- Log-derivative trick
- Almost specific to RL
- Has practical issues

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

- Straightforward way

$$J = \underset{z \sim \pi}{E} R(z) = \int_z \pi(z) R(z) dz$$

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

- Straightforward way

$$J = \underset{z \sim \pi}{E} R(z) = \int \pi(z) R(z) dz$$

over all possible
sessions...

Probability of such session
under current policy
(math slang)

any issues with that formula?

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

- Approximate way

$$J = \underset{z \sim \pi}{E} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

- Approximate way

$$J = \underset{z \sim \pi}{E} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

$$\nabla J = \frac{\partial J}{\partial \pi} = ???$$

Policy gradient

- We want to optimize expected reward

$$\underset{\pi}{\operatorname{argmax}} \underset{z \sim \pi}{E} R(z)$$

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

- Approximate way

$$J = \underset{z \sim \pi}{E} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

$$\nabla J = \frac{\partial J}{\partial \pi} = ???$$

Can't compute gradient
over sampling



Policy gradient

- Objective

$$J = E_{z \sim \pi} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

- Gradient

$$\nabla J = \int \nabla \pi(z) R(z) dz$$

Logderivative trick

Simple math

$$\nabla \log \pi(z) = ? ? ?$$

(try chain rule)

Logderivative trick

Simple math

$$\nabla \log \pi(z) = \frac{1}{\pi(z)} \cdot \nabla \pi(z)$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

Alternative in some cases: reparameterization trick

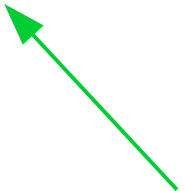
Policy gradient

- Objective

$$J = E_{z \sim \pi} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

- Gradient

$$\nabla J = \int \nabla \pi(z) R(z) dz$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$


Policy gradient

- Objective

$$J = E_{z \sim \pi} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

- Gradient

$$\nabla J = \int_z \pi \cdot \nabla \log \pi(z) R(z) dz$$

Policy gradient

- Objective

$$J = E_{z \sim \pi} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

- Gradient

$$\nabla J = \int \pi \cdot \nabla \log \pi(z) R(z) dz$$

Can we approximate this?

Policy gradient

- Objective

$$J = E_{z \sim \pi} R(z) \approx \frac{1}{N} \sum_{z_i \sim \pi}^N R(z)$$

- Gradient

$$\nabla J \approx \frac{1}{N} \sum_{z_i \sim \pi}^N \nabla \log \pi(z) R(z)$$

Approximate policy gradient

- Initialize NN weights $W_0 \leftarrow \text{random}$
- Loop:
 - Sample N sessions (states, actions, final reward)
 - Update weights

$$W_{i+1} = W_i + \alpha \nabla \left[\sum_{k=0}^N R_k \cdot \log \pi_{W_i}(z_k) \right]$$

Approximate policy gradient

- Initialize NN weights $W_0 \leftarrow \text{random}$
- Loop:
 - Sample N sessions (states, actions, final reward)
 - Update weights

$$W_{i+1} = W_i + \alpha \nabla \sum_{k=0}^N R_k \cdot \log \pi_{W_i}(z_k)$$

**That's slang.
Let's be more explicit!**

Approximate policy gradient

- Initialize NN weights $W_0 \leftarrow \text{random}$
- Loop:
 - Sample N sessions (states, actions, final reward)
 - Update weights

$$W_{i+1} = W_i + \alpha \nabla \left[\sum_{k=0}^N R_k \cdot \sum_{s_j, a_j \in \mathcal{Z}_k} \log \pi_{W_i}(a_i | s_i) \right]$$

Approximate policy gradient

- Initialize NN weights $W_0 \leftarrow \text{random}$
model = MLPClassifier()/MLPRegressor() or whatever
- Loop:
 - Sample N sessions (states, actions, final reward)
 - Update weights

$$W_{i+1} = W_i + \alpha \nabla \left[\sum_{k=0}^N R_k \cdot \sum_{s_j, a_j \in \mathcal{Z}_k} \log \pi_{W_i}(a_i | s_i) \right]$$

model.fit(states, actions, sample_weight=total_rewards)

(sklearn v0.18 neural networks don't support weights!)

Comparison

Crossentropy method

Minimize crossentropy (with threshold)

$$\nabla L \approx -\frac{1}{N} \sum_{z_i \sim \pi}^N \nabla \log \pi(z) [R(z) \geq \psi]$$

Policy gradient

Maximize expected reward

$$\nabla J \approx \frac{1}{N} \sum_{z_i \sim \pi}^N \nabla \log \pi(z) R(z)$$

Tricks (policy gradient edition)

- Regularize with entropy
 - to prevent premature convergence.
 - Parallelize sampling
 - Use RNNs if partially-observable
 - Reduce variance
 - Design $R(z)$ so that it's less noisy
- More on that later!**



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



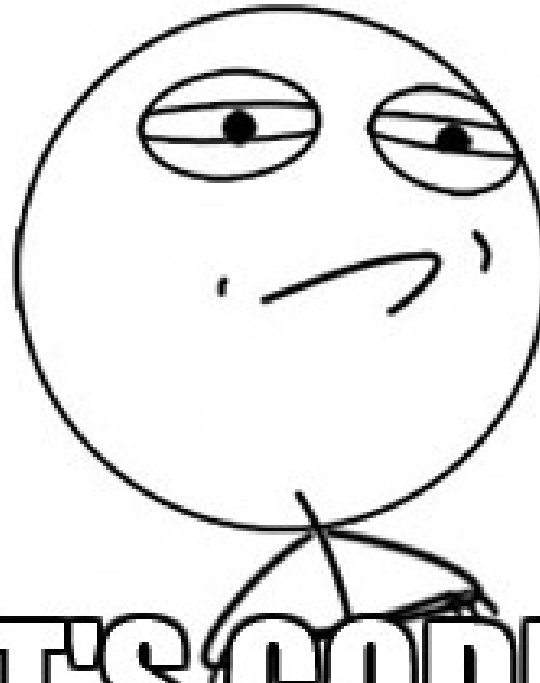
Gonna fix that next lecture!

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



After 70+ slides...

CHALLENGE ACCEPTED



LET'S CODE IT

memegenerator.net