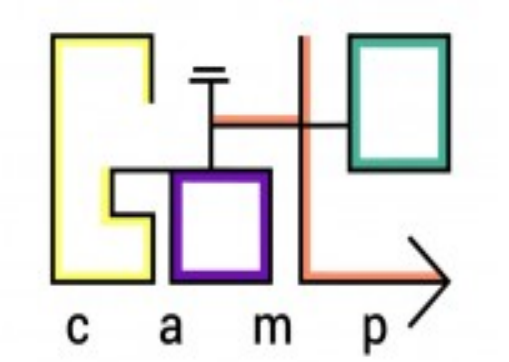
Reinforcement learning

What it does and why should you care?

LAMBDA



British Hedgehog Preservation Society

Terms

All questions are always welcome

Even if question feels stupid,

- Chances are, half of the group is just like you.

<a convenient slide for public survey>

Supervised learning

Given:

- objects and answers (x, y)
- algorithm family $a_\theta(x) \rightarrow y$
- loss function $L(y, a_\theta(x))$

Find:

$$\theta' \leftarrow argmin_\theta L(y, a_\theta(x))$$

Supervised learning

Given:

- objects and answers (x, y)

  [banner,page], ctr

- algorithm family $a_\theta(x) \rightarrow y$

  linear / tree / NN

- loss function $L(y, a_\theta(x))$

  MSE, crossentropy

Find:

$$\theta' \leftarrow argmin_\theta L(y, a_\theta(x))$$

Supervised learning

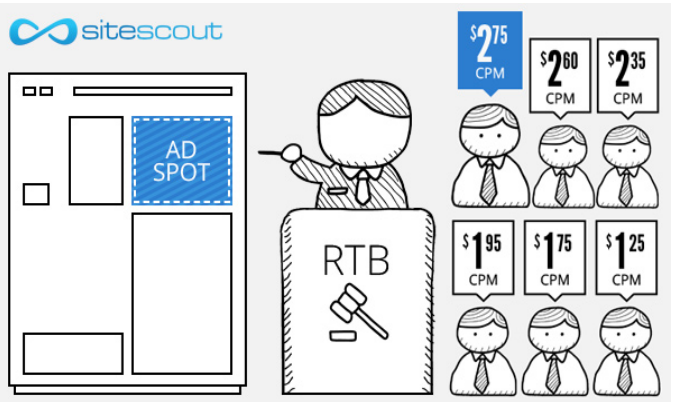
Great... except if we have no reference answers

Online Ads

Great... except if we have no reference answers

We have:

- YouTube at your disposal
- Live data stream
  (banner & video features, #clicked)
- (insert your favorite ML toolkit)



We want:

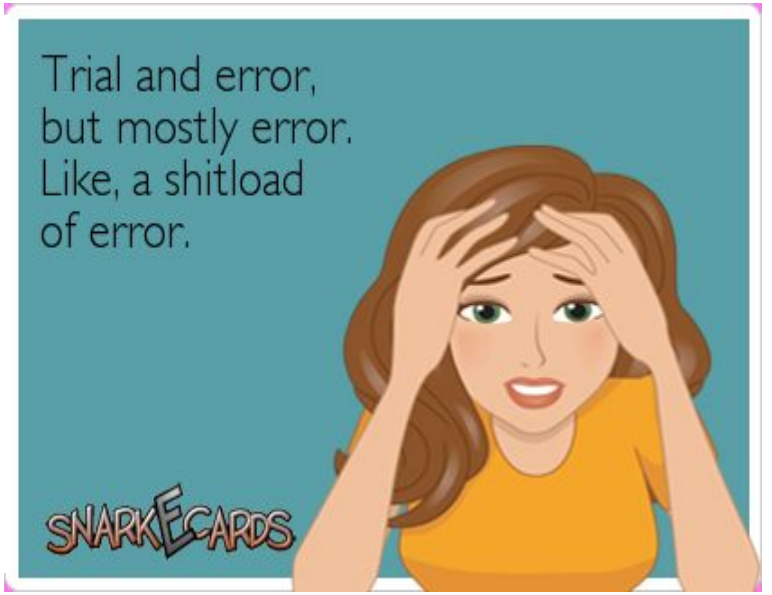
- Learn to pick relevant ads

Ideas?

Duct tape approach

Common idea:

- Initialize with naïve solution
- Get data by trial and error and error and error and error
- Learn (situation) → (optimal action)
- Repeat



Problems

Problem 1:

- What exactly does the “optimal action” mean?

Extract as much money as you can right now

VS

Make user happy so that he would visit you again

Problems

Problem 2:

- If you always follow the “current optimal” strategy, you may never discover something better.

- If you show the same banner to 100% users, you will never learn how other ads affect them.

Ideas?

Duct tape approach

Reinforcement learning

What-what learning?

Supervised learning

- Learning to approximate reference answers
- Needs correct answers
- Model does not affect the input data

Reinforcement learning

- Learning optimal strategy by trial and error
- Needs feedback on agent's own actions
- Agent can affect it's own observations

What-what learning?

Unsupervised learning

- Learning underlying data structure
- No feedback required
- Model does not affect the input data

Reinforcement learning

- Learning optimal strategy by trial and error
- Needs feedback on agent's own actions
- Agent can affect it's own observations

What is: bandit

Examples:

- banner ads (RTB)
- recommendations
- medical treatment

What is: bandit

observation

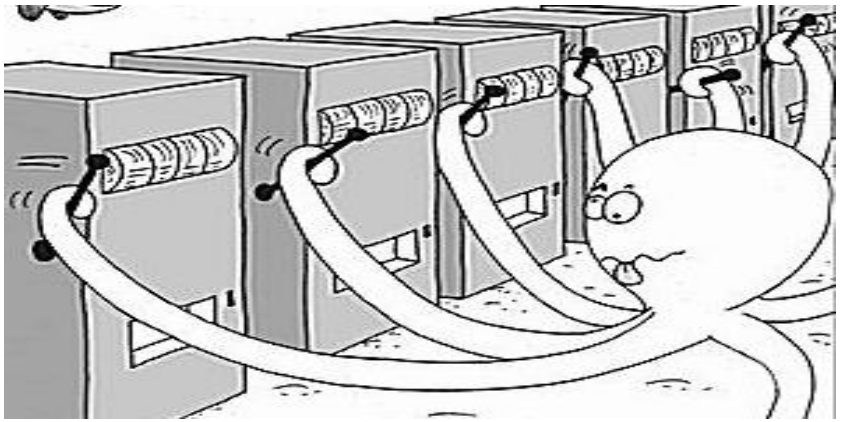
Agent

action

Feedback

Examples:

- banner ads (RTB)
- recommendations
- medical treatment



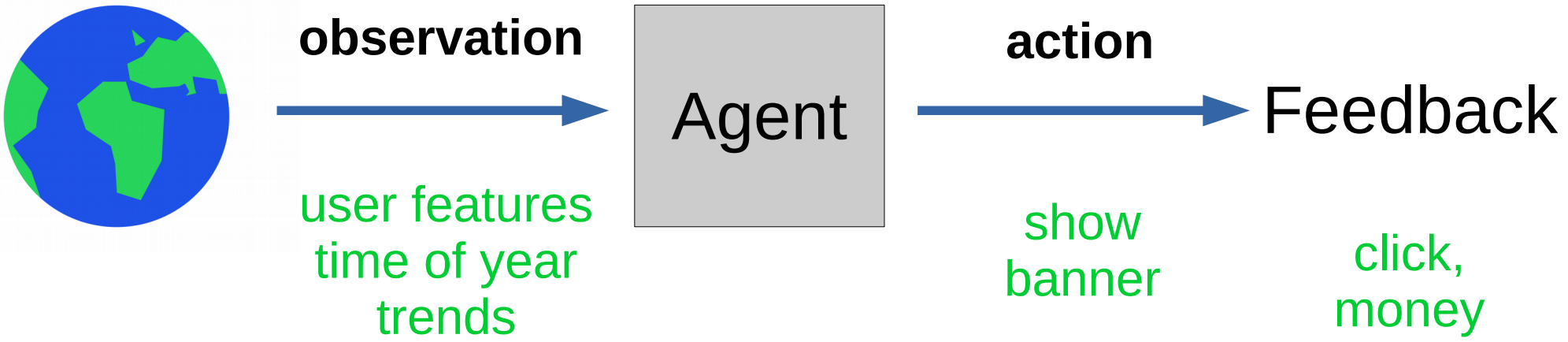
What is: bandit

Examples:

- banner ads (RTB)
- recommendations
- medical treatment

Q: what's observation, action and feedback in the banner ads problem?

What is: bandit



Examples:

- banner ads (RTB)
- recommendations
- medical treatment

What is: bandit

observation

user features
time of year
trends

Agent

action

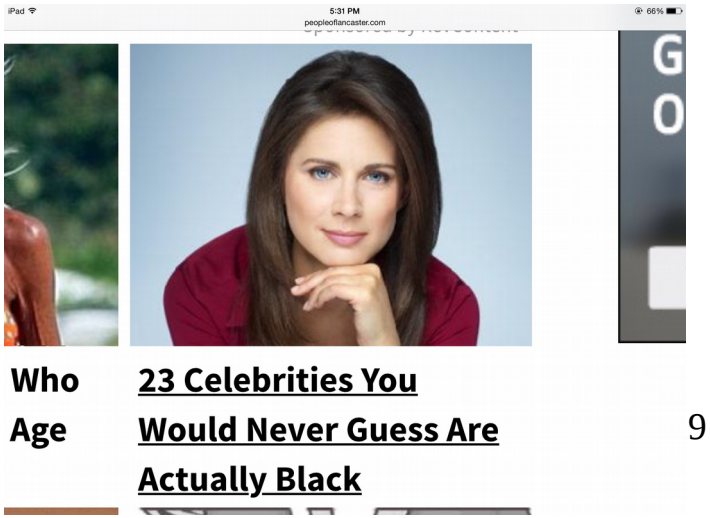
show
banner

Feedback

click,
money

Q: You're Yandex/Google/Youtube. There's a kind of banners that would have great click rates: the “clickbait”.

Is it a good idea to show clickbait?



What is: bandit

observation

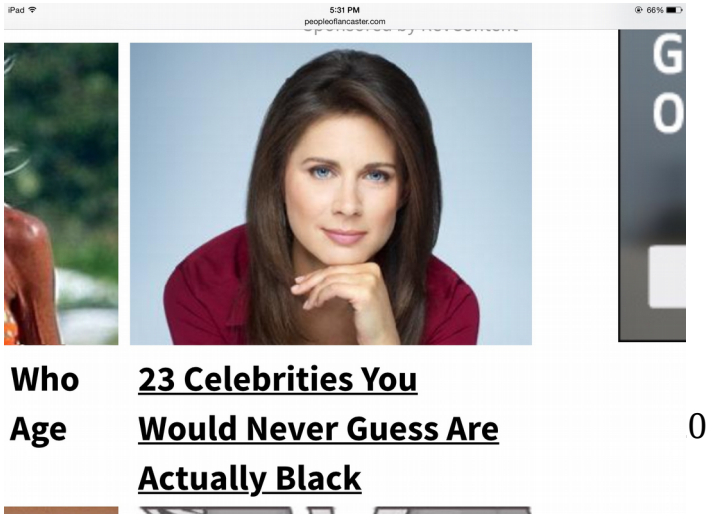
Agent

action

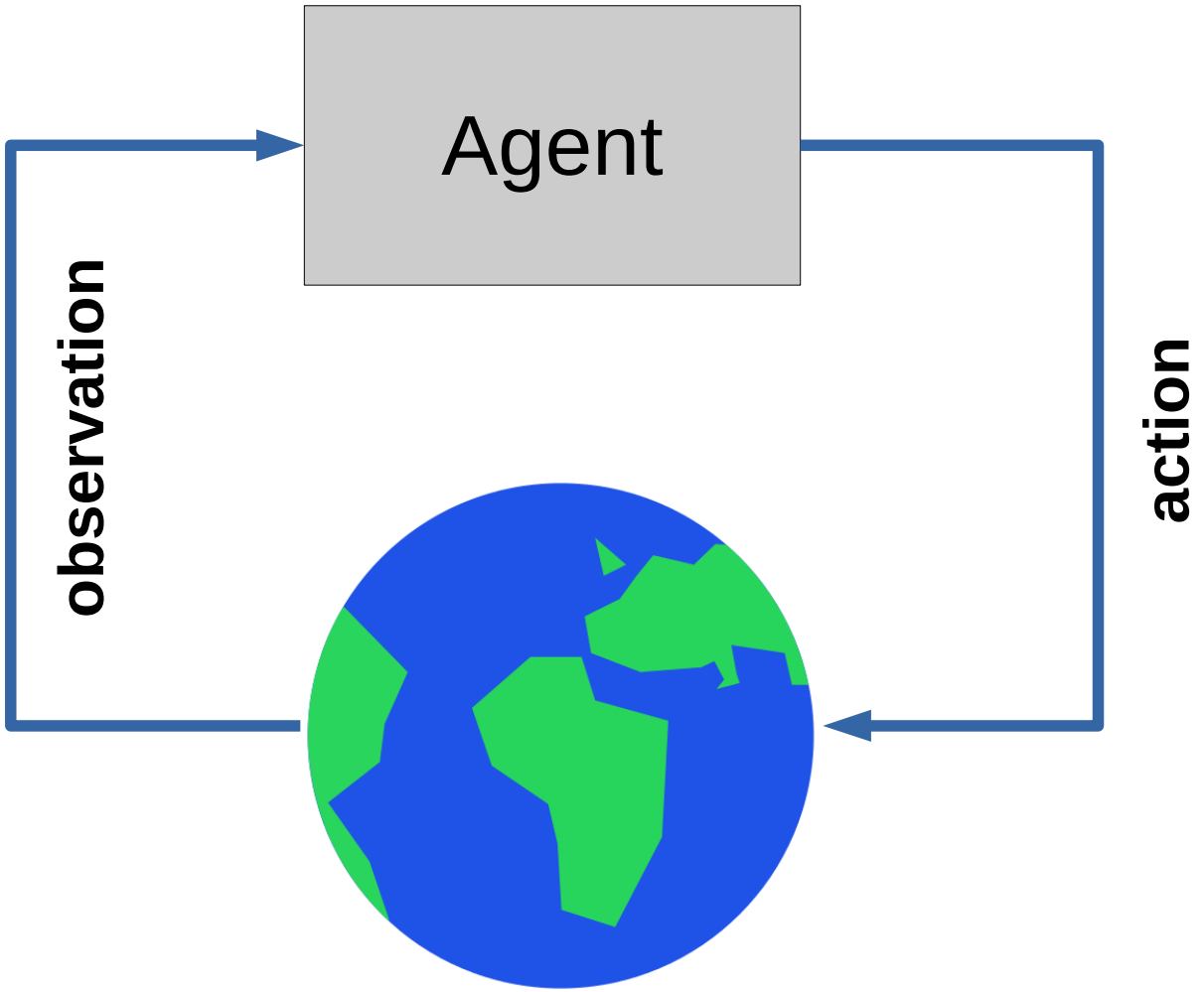
Feedback

Q: You're Google/Yandex/Youtube. There's a kind of banners that would have great click rates: the “clickbait”.

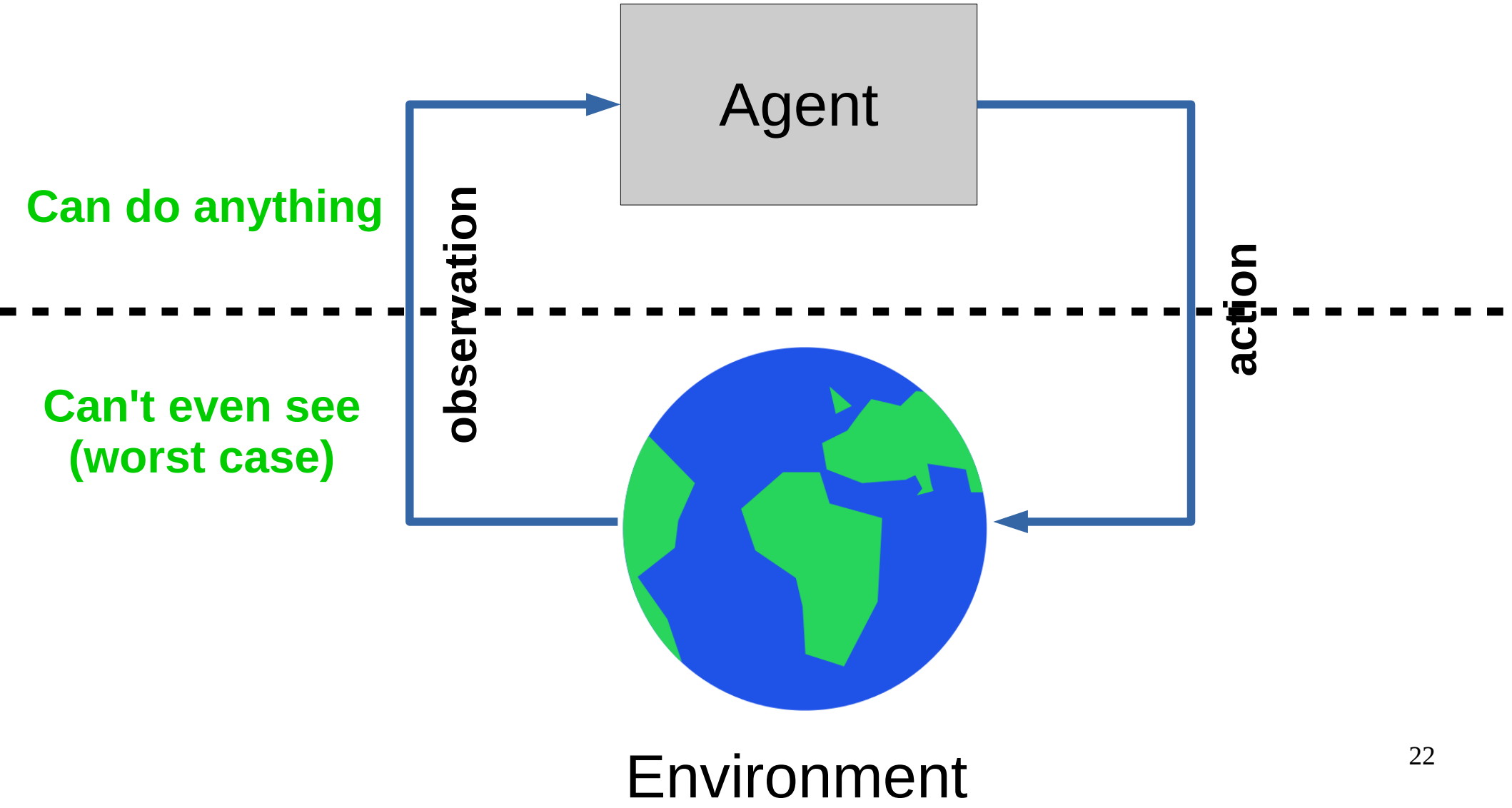
Is it a good idea to show clickbait?
No, no one will trust you after that!



What is: decision process



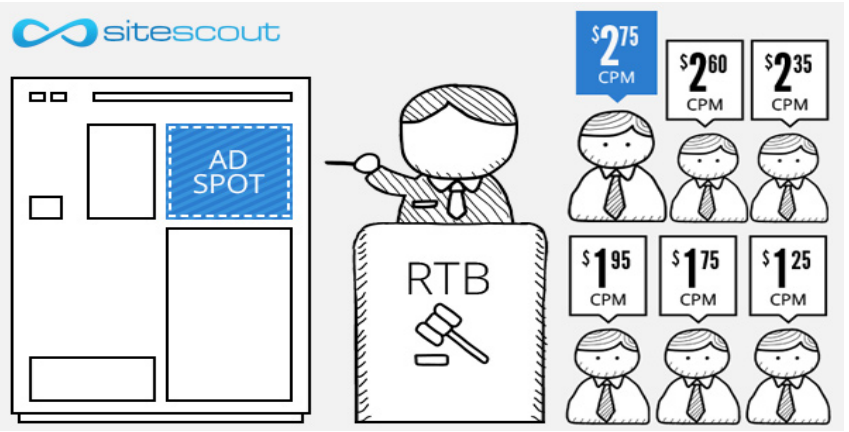
What is: decision process

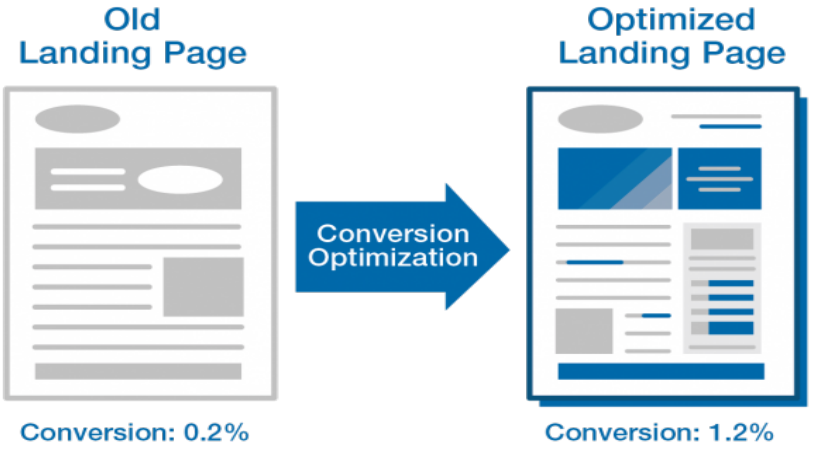


Applications: web

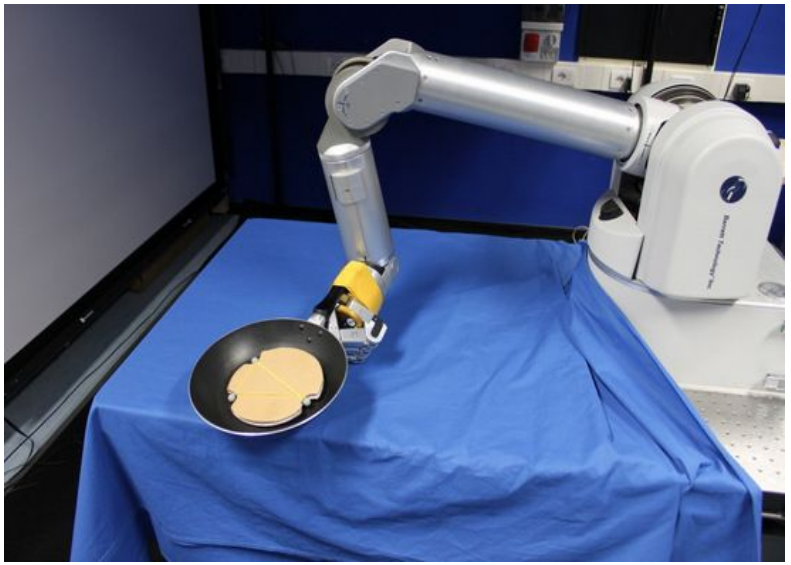
- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend movies to users
  - Find pages relevant to queries

- **Example**
  - Observation – user features
  - Action – show banner #i
  - Feedback – did user click?





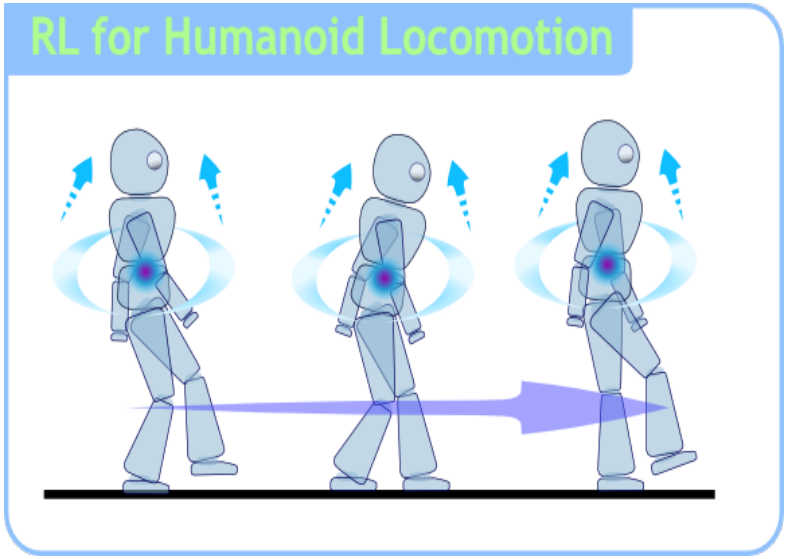
Applications: dynamic systems



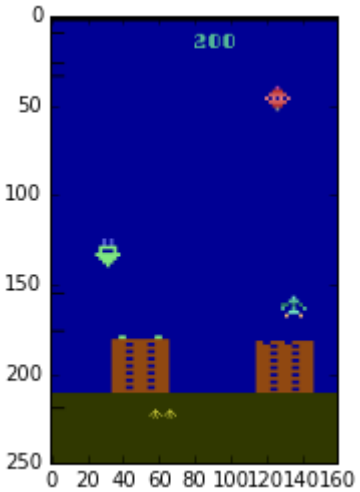
Applications: dynamic systems

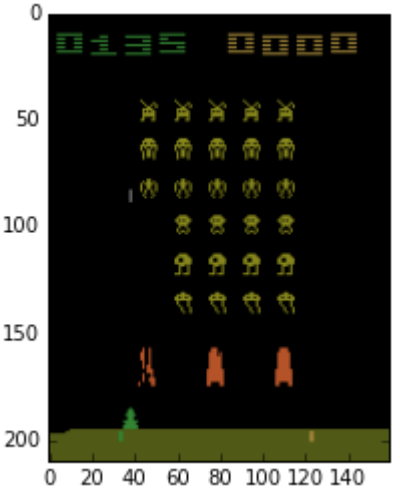
- **Cases:**
  - Robots
  - Self-driving vehicles
  - Pilot assistant
  - More robots!

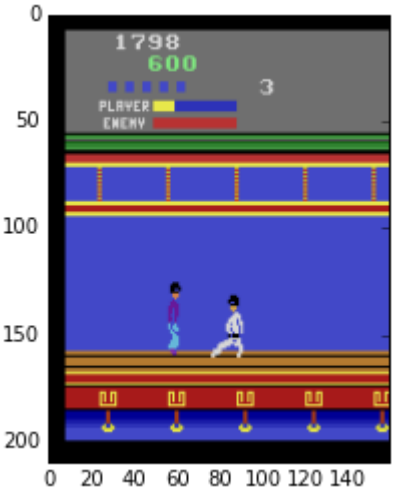
- **Example**
  - Observation: sensor feed
  - Action: voltage sent to motors
  - Feedback: how far did it move forward before falling



~~Applications~~: videogames

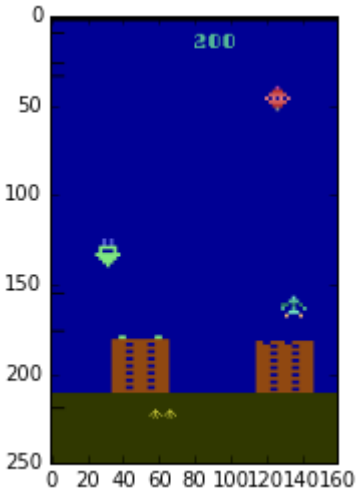


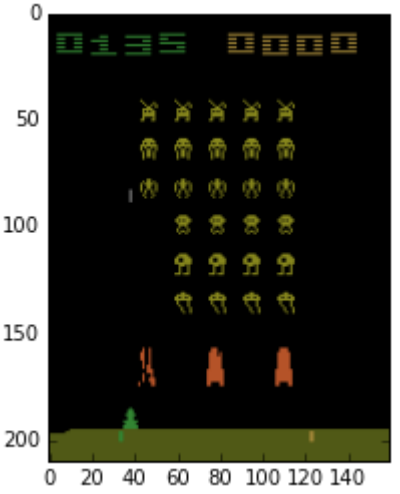


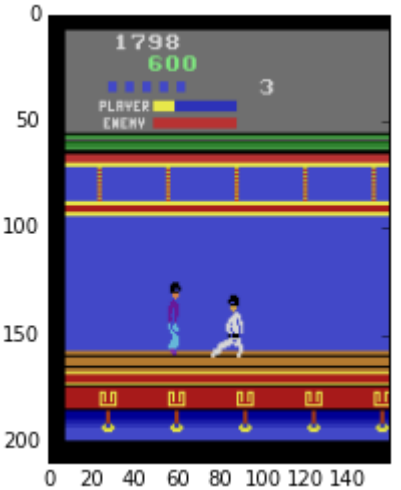


- **Q:** What are observations, actions and feedback?

~~Applications~~: videogames





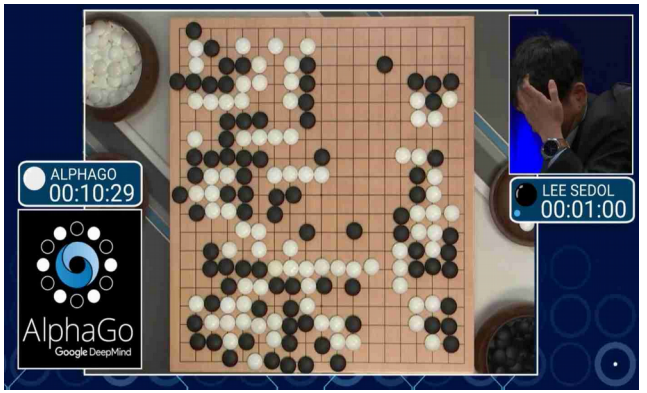


- **Q:** What are observations, actions and feedback?

More use cases

- Personalized medical treatment

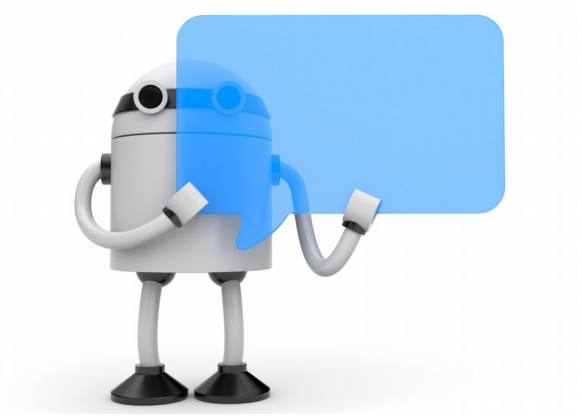
- Even more games (Go, chess, etc)

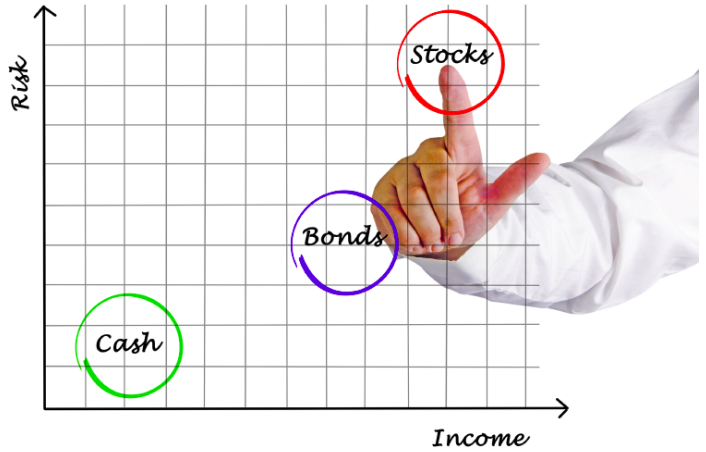


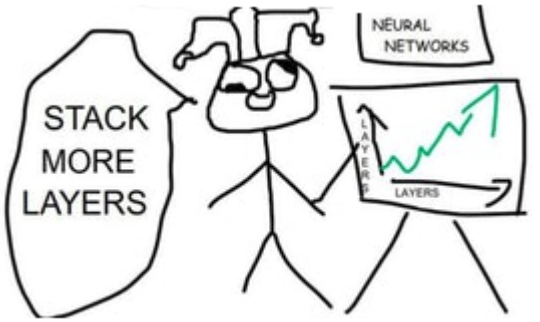
- **Q:** What are observations, actions and feedback?

Other use cases

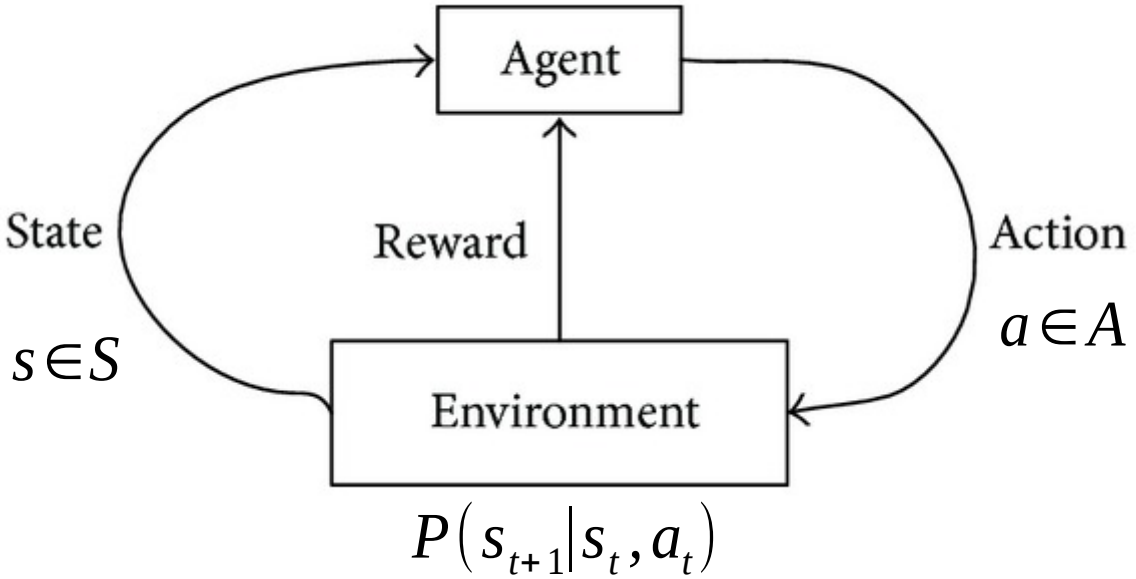
- Conversation systems
  - learning to make user happy
- Quantitative finance
  - portfolio management
- Deep learning
  - optimizing non-differentiable loss
  - finding optimal architecture







The MDP formalism

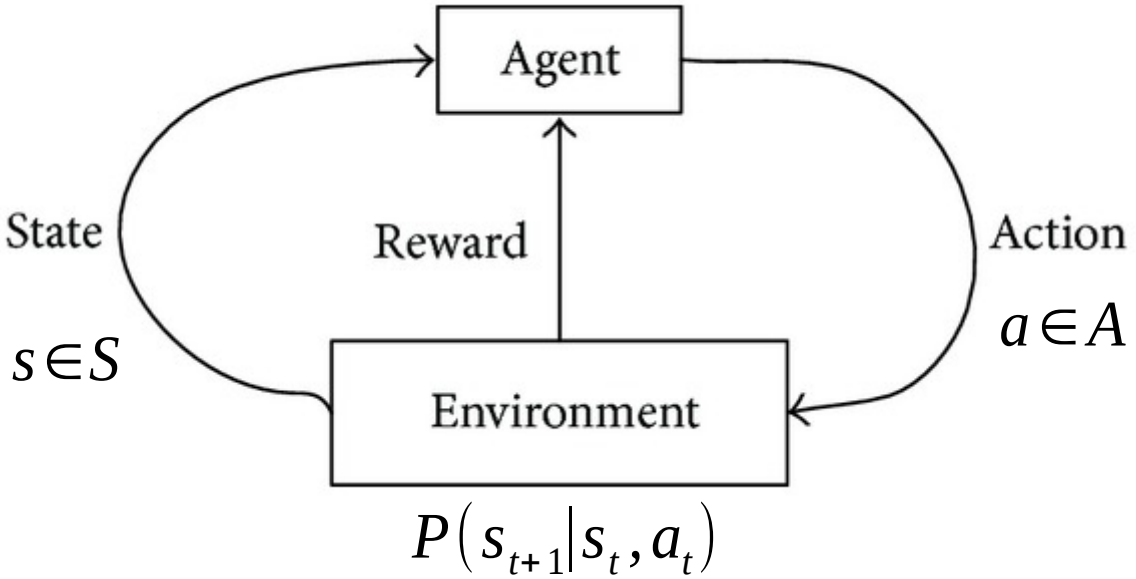


Markov Decision Process

- Environment states: $s \in S$
- Agent actions: $a \in A$
- Rewards $r \in \mathbb{R}$

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

The MDP formalism

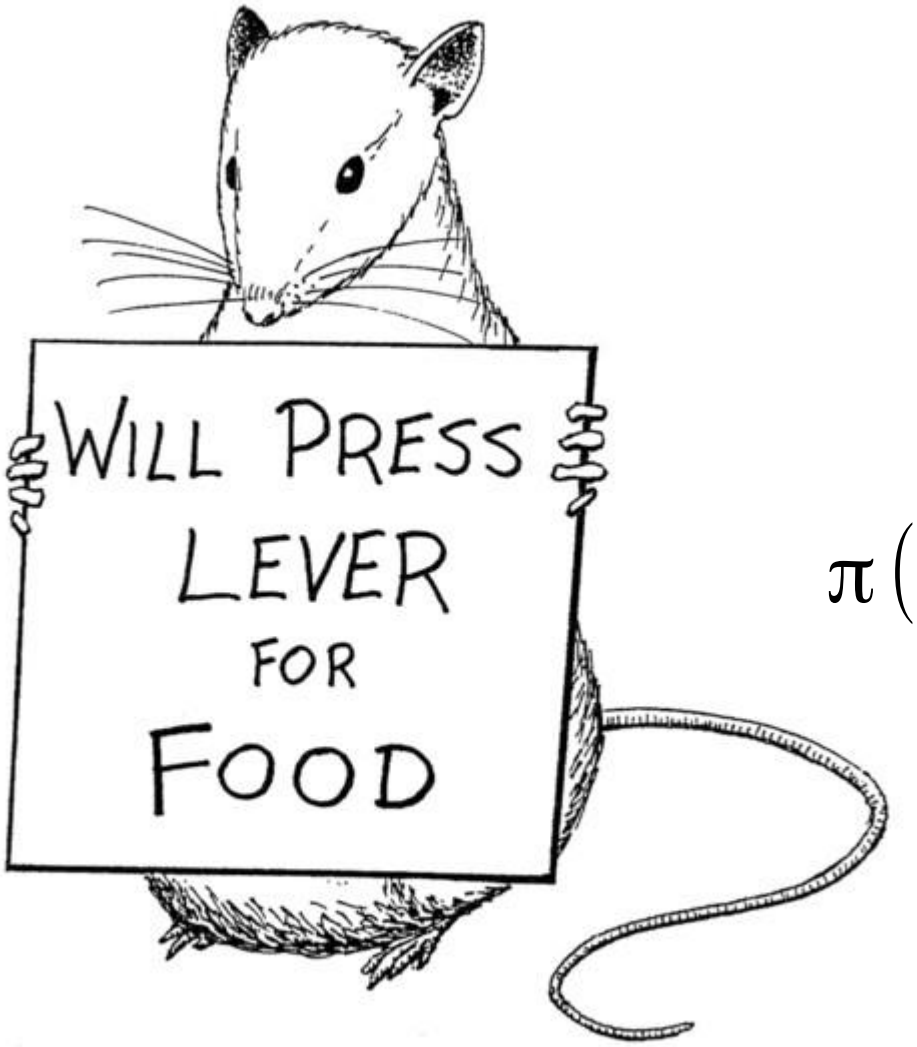


Markov Decision Process

Markov assumption

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}) = P(s_{t+1}|s_t, a_t)$$

Total reward



Total reward for session:

$$R = \sum_t r_t$$

Agent's policy:

$$\pi(a|s) = P(\textit{take action } a | \text{in } \textit{state } s)$$

Problem: find policy with highest reward:

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

Objective

The easy way:

$E_\pi R$ is an expected sum of rewards that agent with policy π earns per session

The hard way:

$$\underset{s_0 \sim p(s_0),\, a_0 \sim \pi(a|s_0),\, s_1, r_0 \sim P(s', r|s, a)}{E} \quad \underset{}{E} \quad \underset{}{E} \quad \ldots \quad \underset{s_n, r_n \sim \pi(a|s_{n-1})}{E} \left[r_0 + r_1 + r_2 + \ldots + r_n \right]$$

How do we solve it?

Value-based approach:

What is the expected reward I get if I take this action?

Value Iteration, Q-learning, SARSA, DQN

...

How do we solve it?

Value-based approach:

What is the expected reward I get if I take this action?

Value Iteration, Q-learning, SARSA, DQN

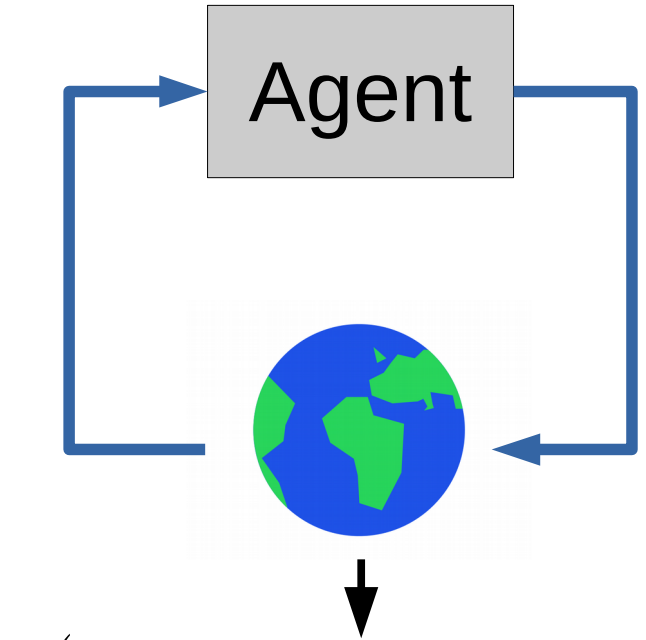
Policy-based:

Environment takes policy and computes expected reward.

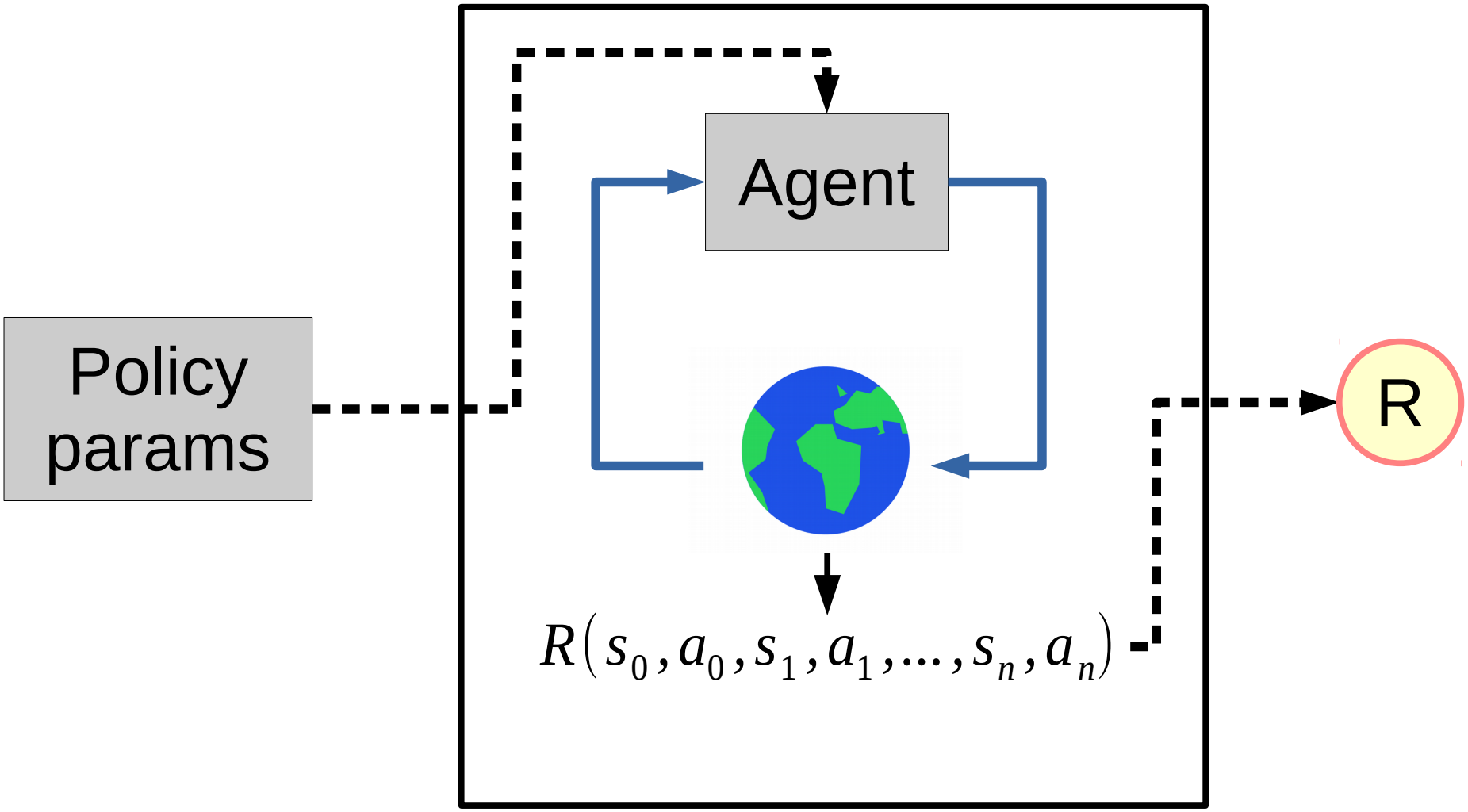
Find such policy that produces best reward!

Crossentropy Method, Policy Gradient, TRPO, ES

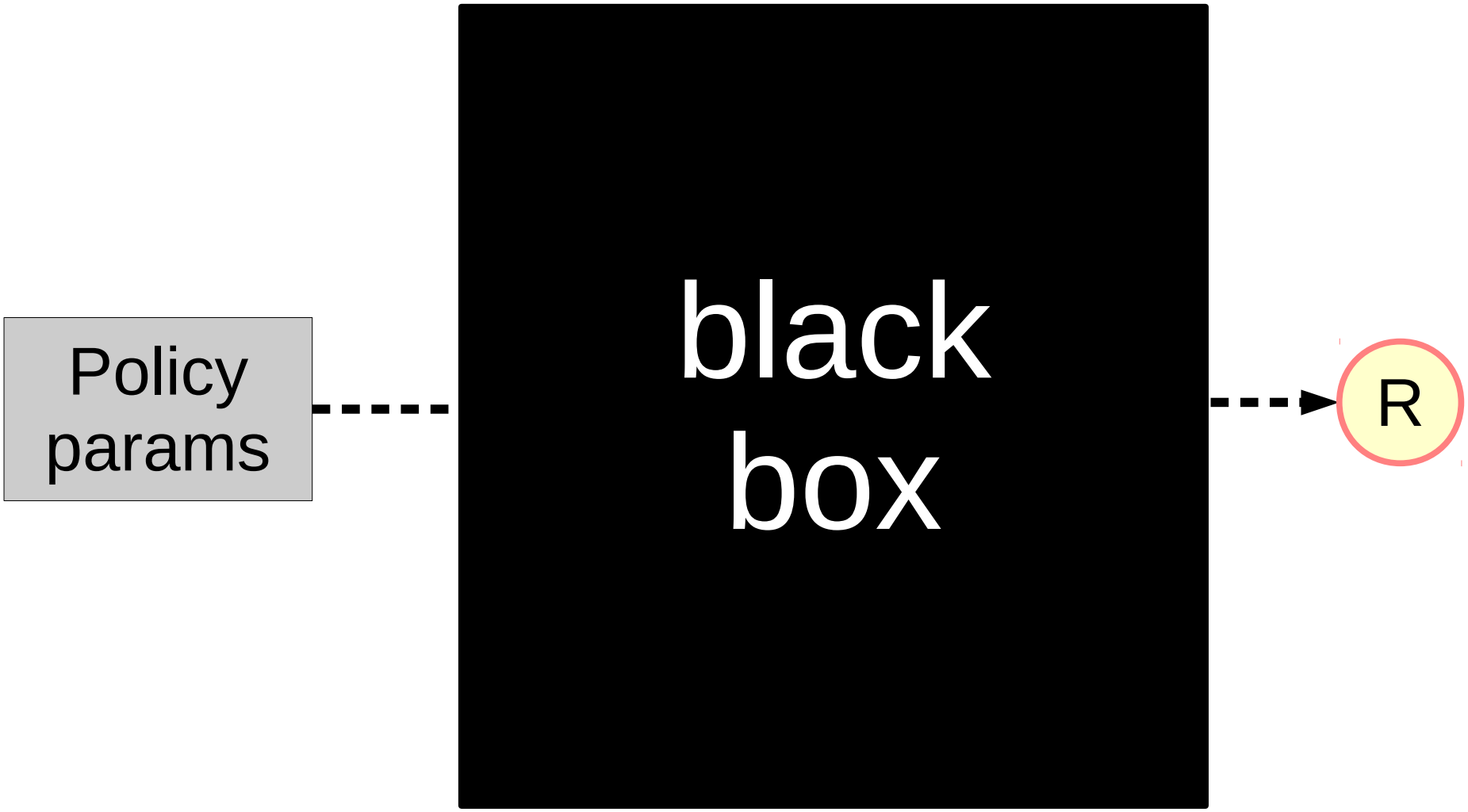
Black box optimization setup



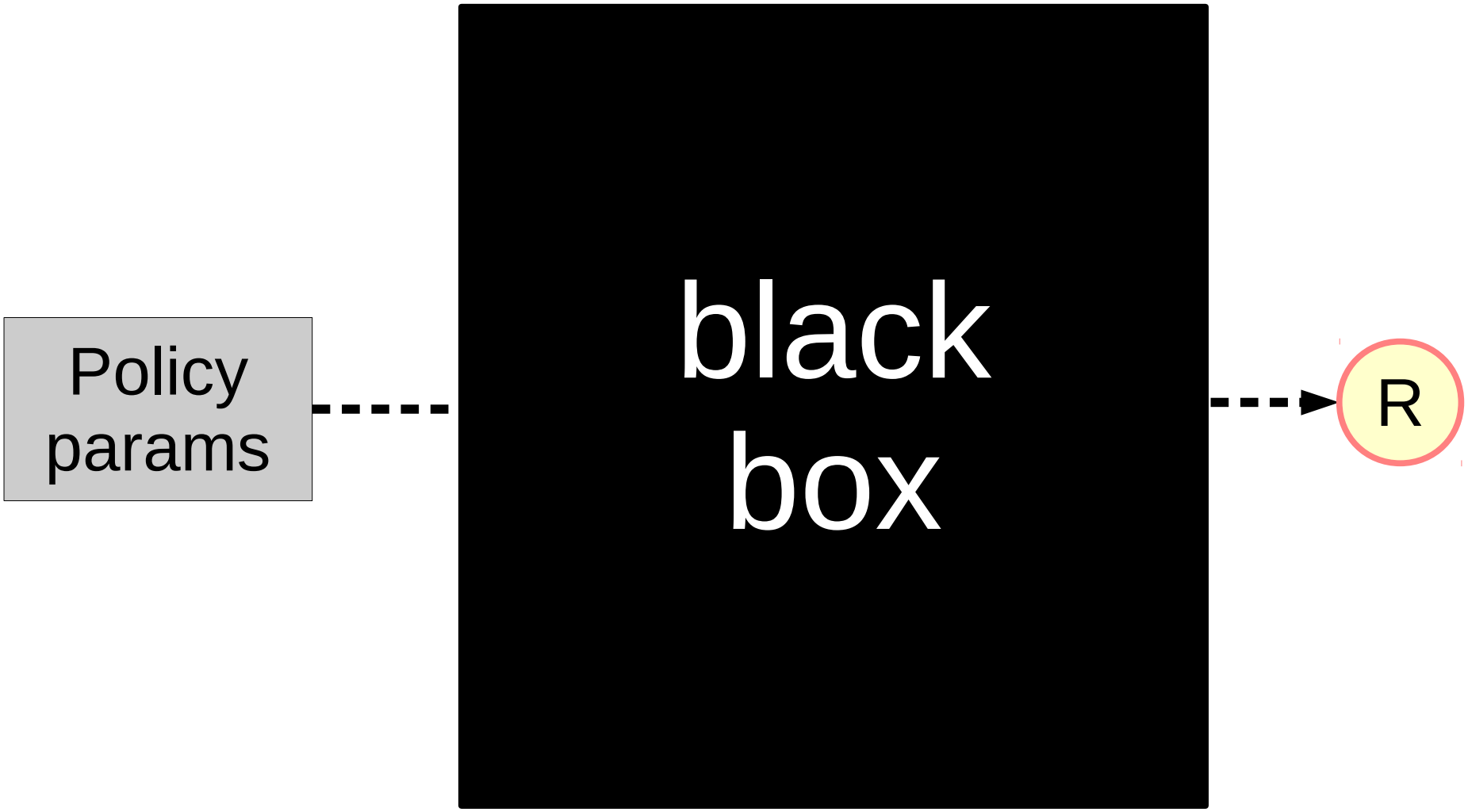
Black box optimization setup



Black box optimization setup



Black box optimization setup



How to find argmax R?

How do we solve it?

General idea:

Play a few sessions

Update your policy

Repeat

Crossentropy method

Intialize policy

Repeat:

- Sample N[100] sessions
- Pick M[20] best sessions, called **elite** sessions
- Change policy so that it prioritizes actions from elite sessions

Tabular crossentropy method

- Policy is a matrix

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

- Sample N games with that policy
- Get M best sessions (elites)

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

Tabular crossentropy method

- Policy is a matrix

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

- Sample N games with that policy
- Take M best sessions (elites)
- 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 sessions (elite)
- 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

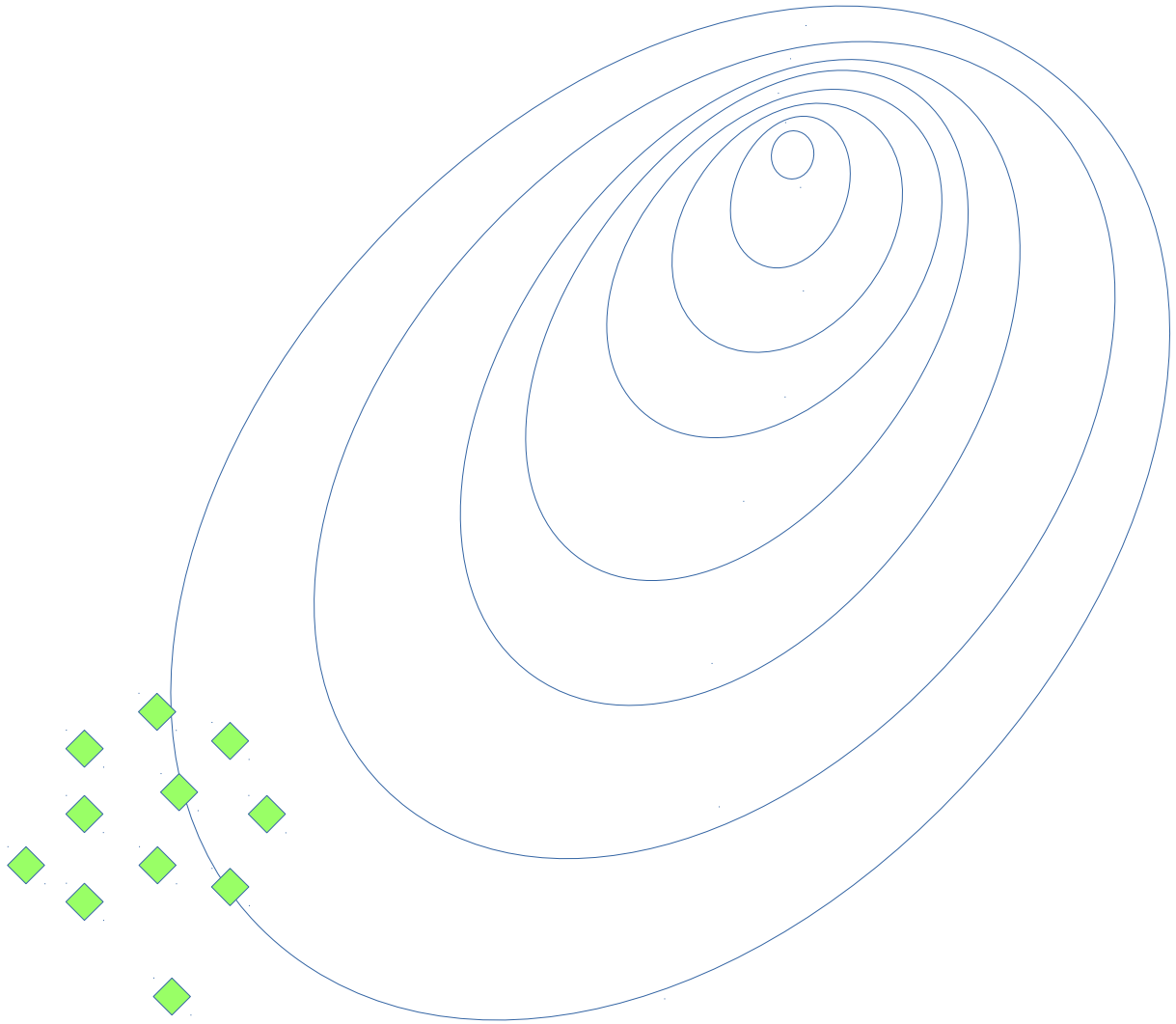
$$\pi_{i+1}(a|s) = \alpha \cdot \pi_{opt} + (1-\alpha)\pi_i(a|s)$$

Stochastic MDPs

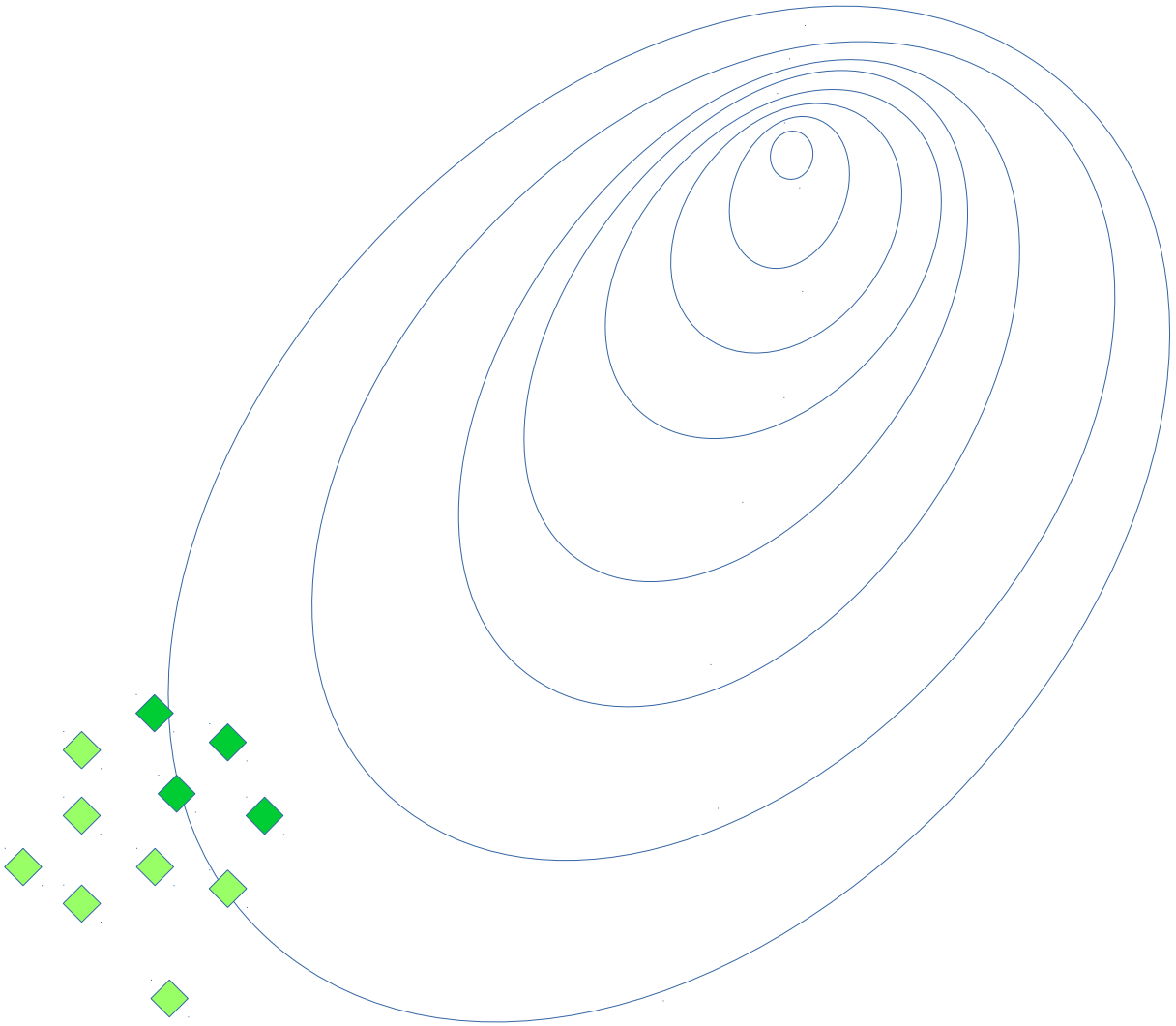
- If there's randomness in environment, algorithm will prefer “lucky” sessions.
- Training on lucky sessions is no good

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

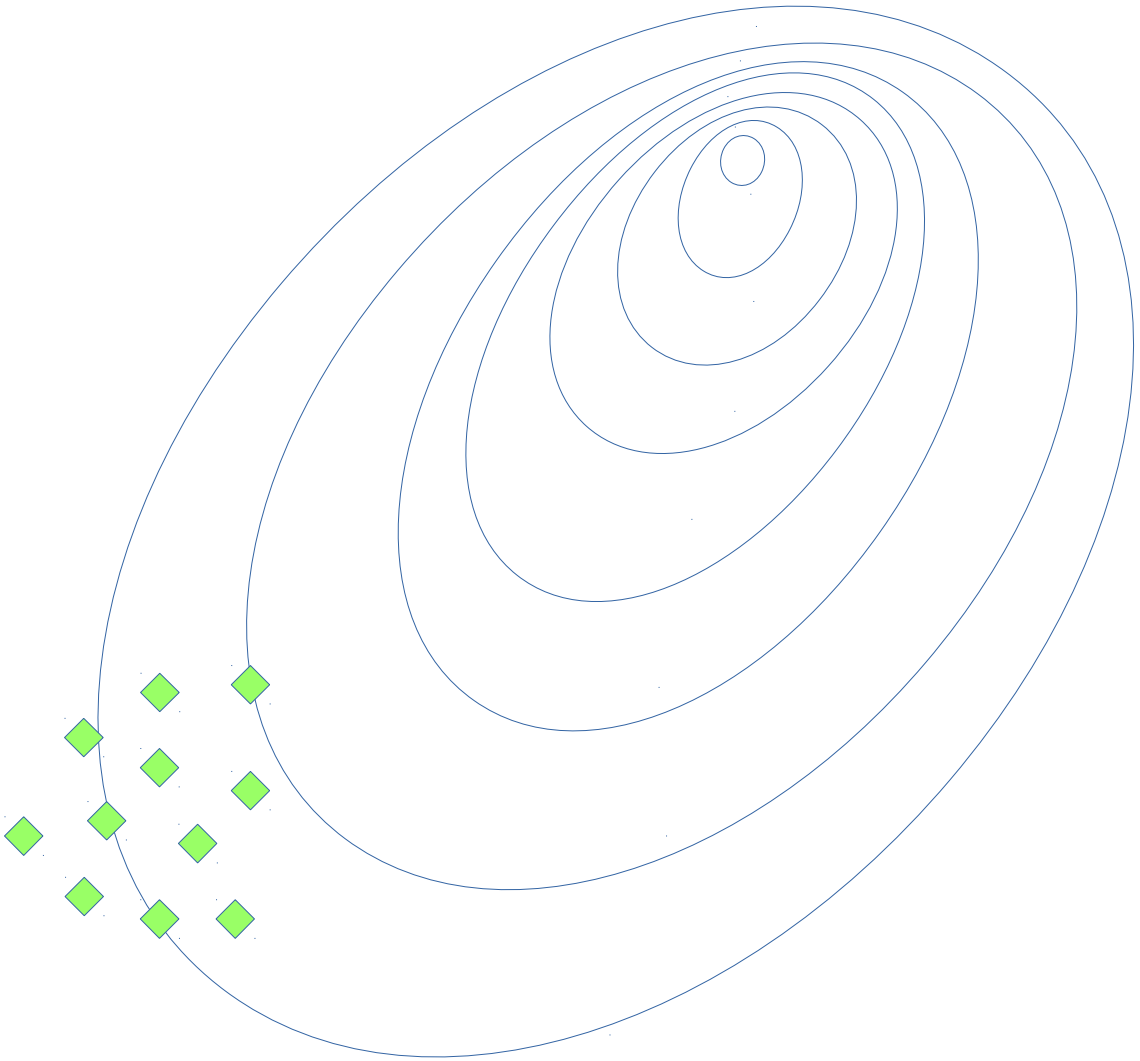
Crossentropy Method



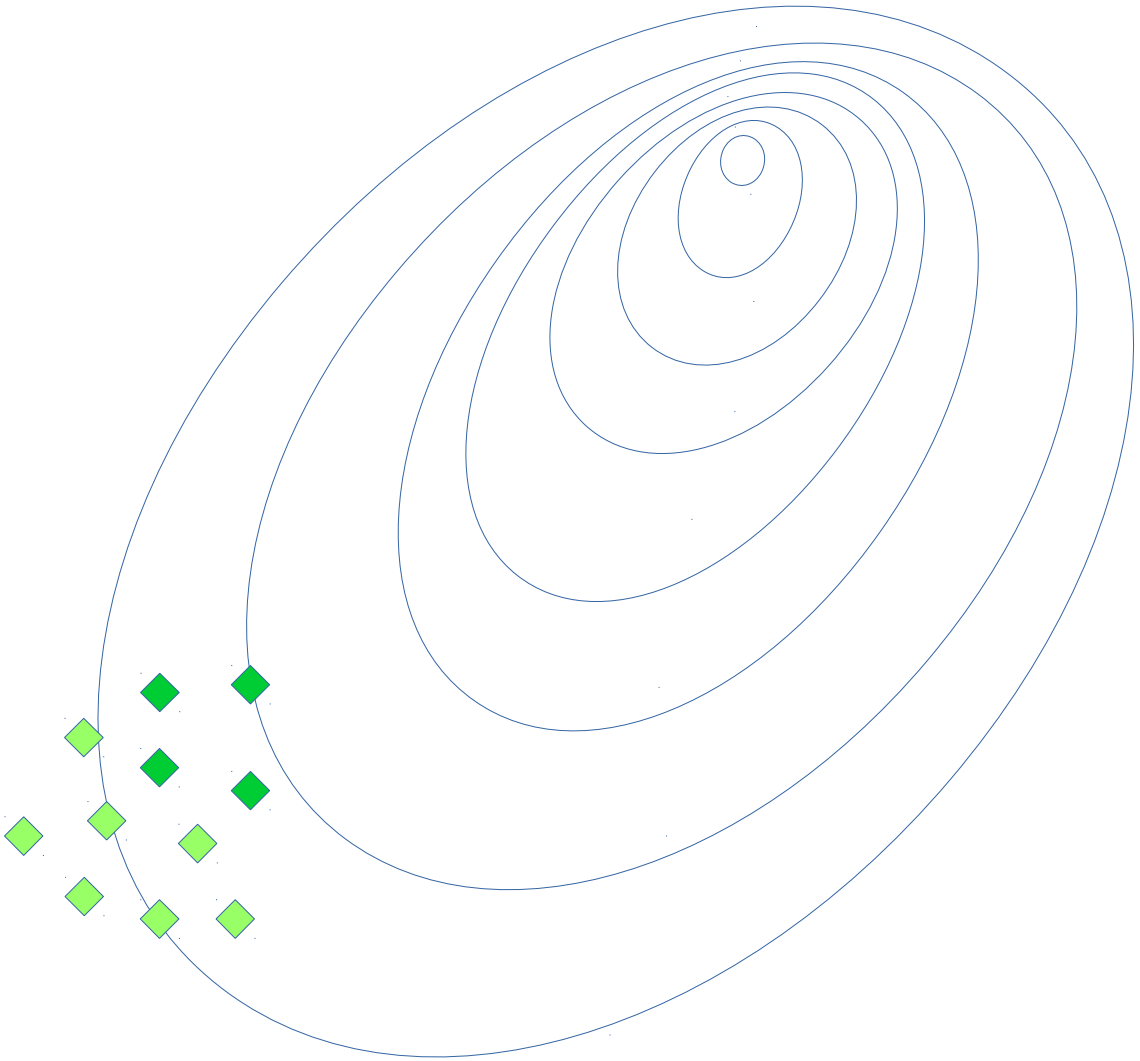
Crossentropy Method



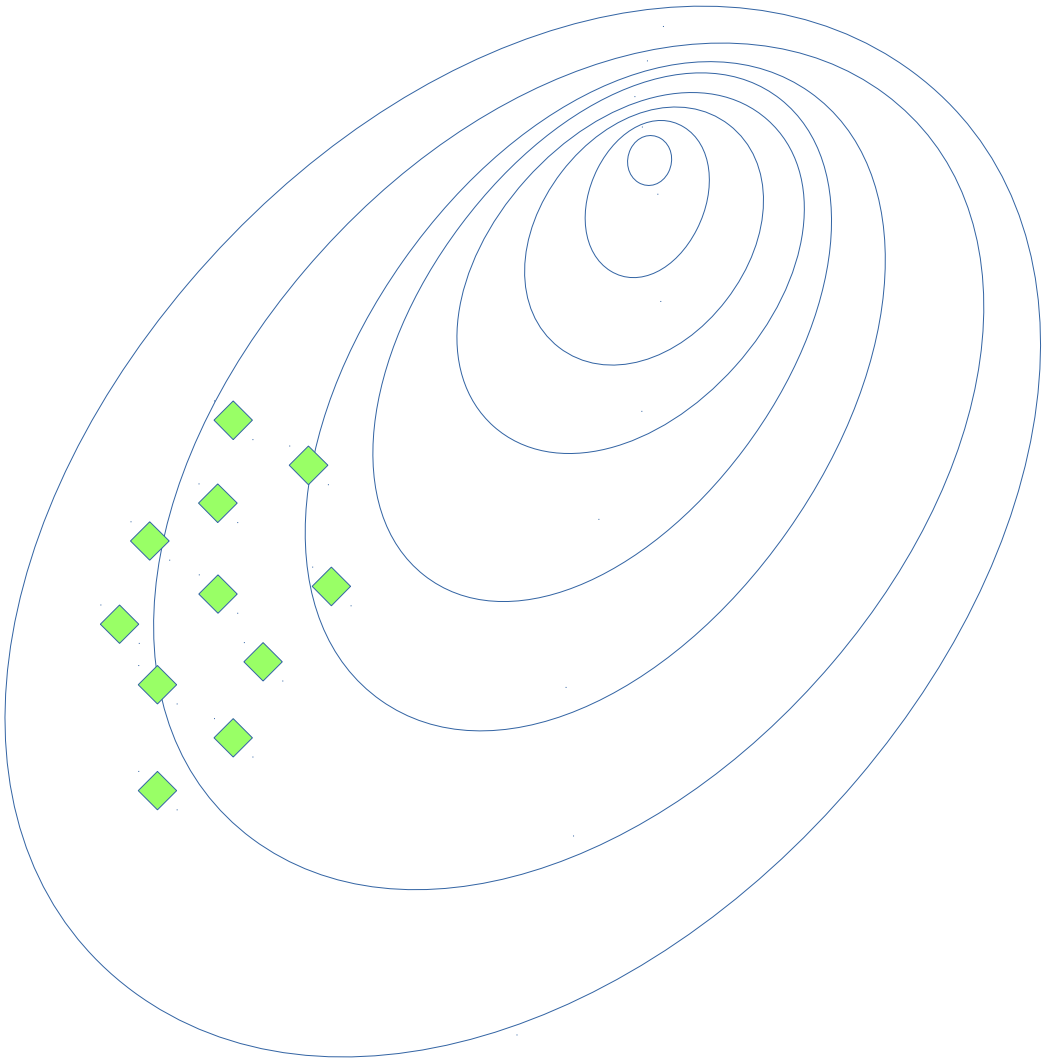
Crossentropy Method



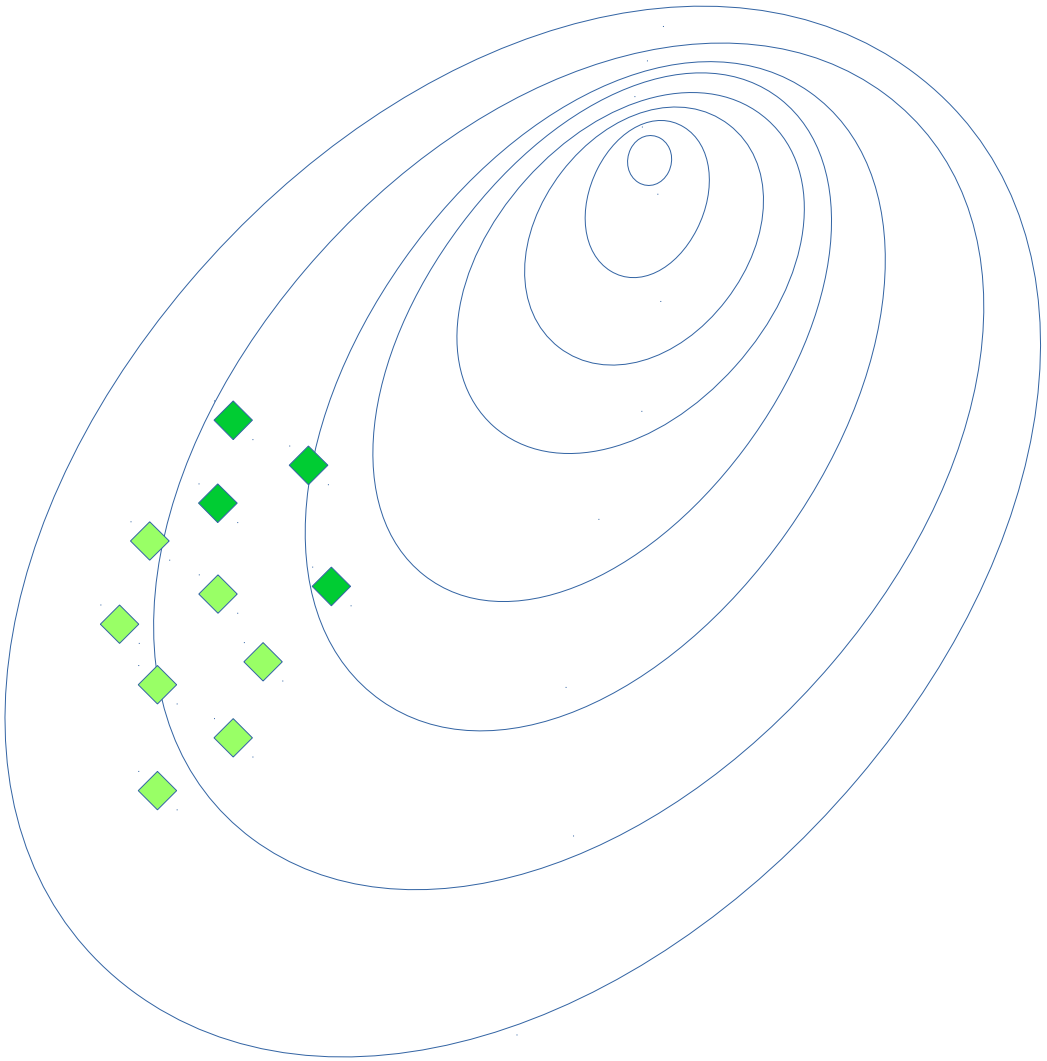
Crossentropy Method



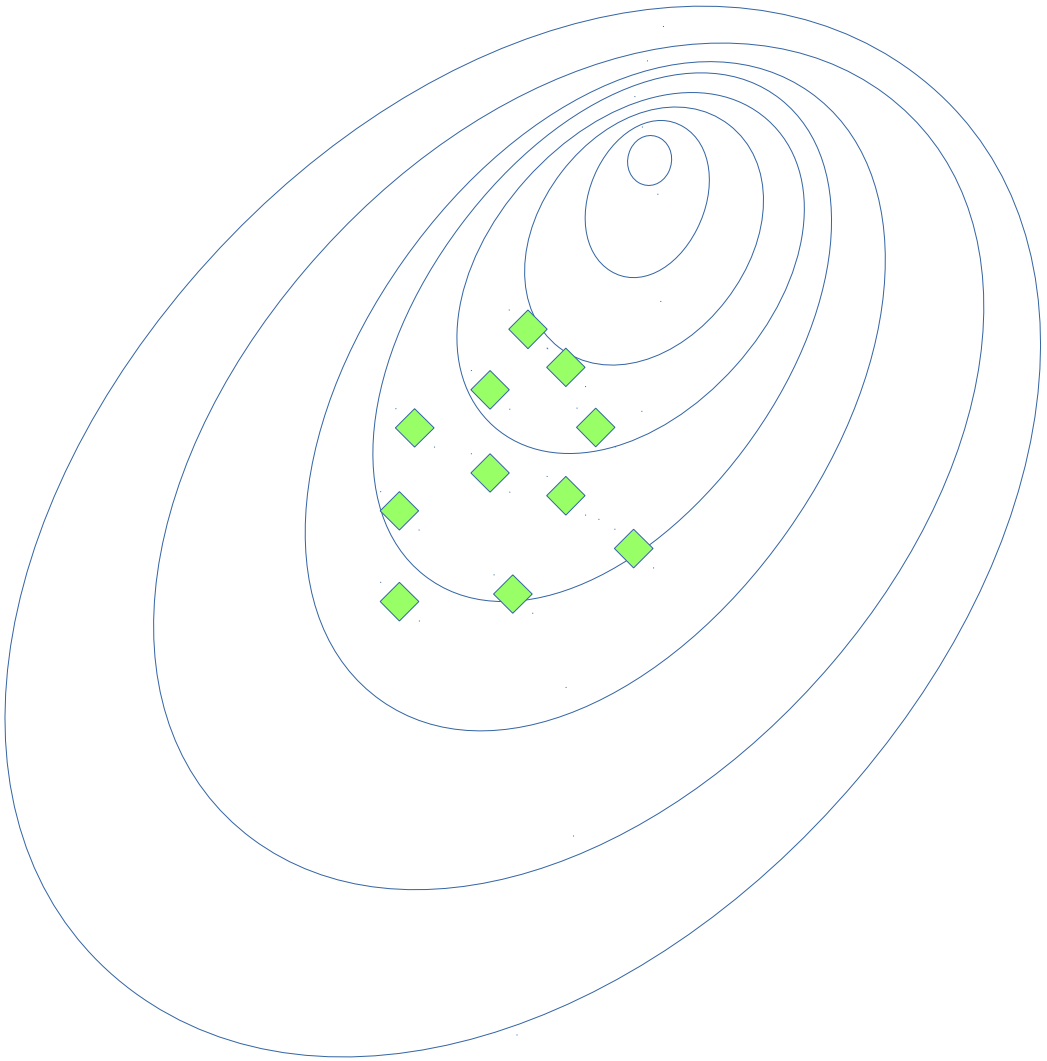
Crossentropy Method



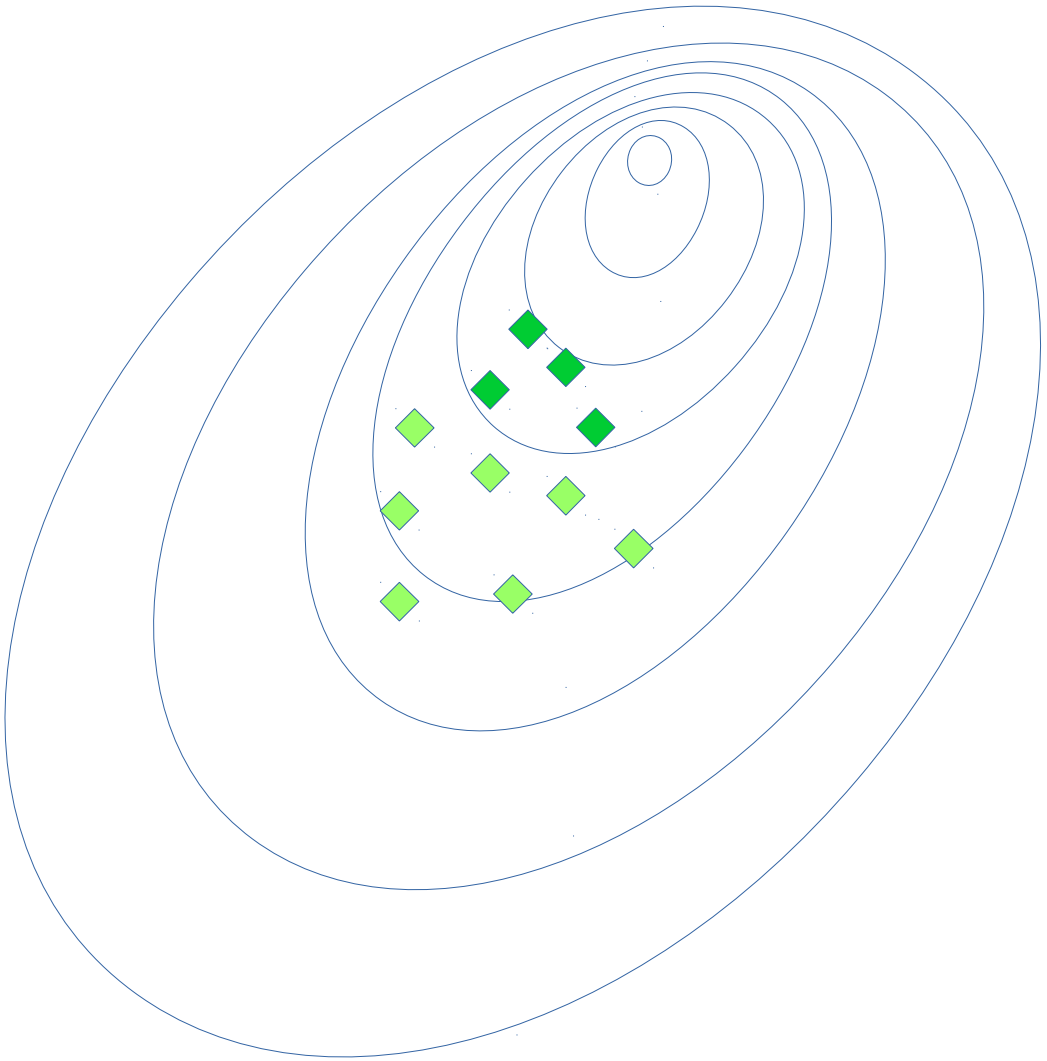
Crossentropy Method



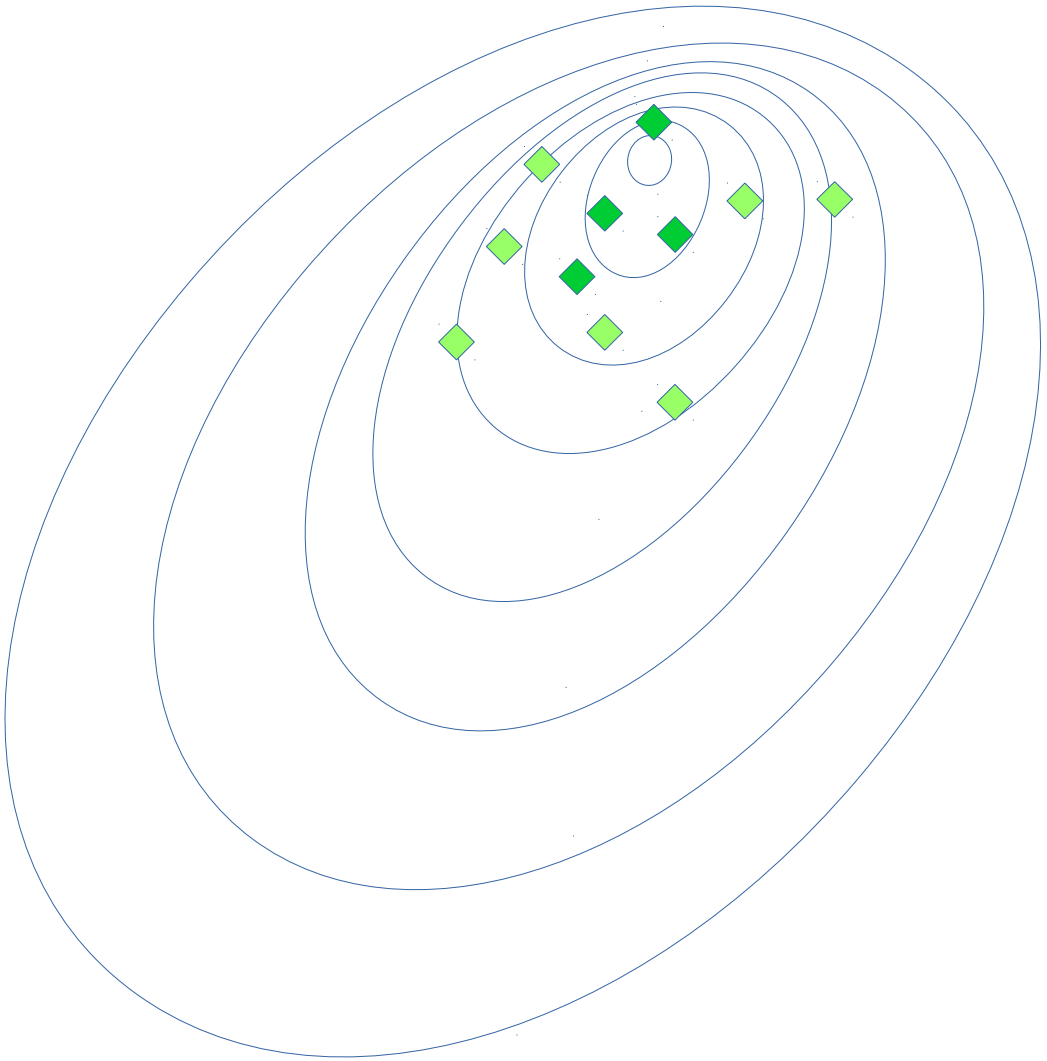
Crossentropy Method



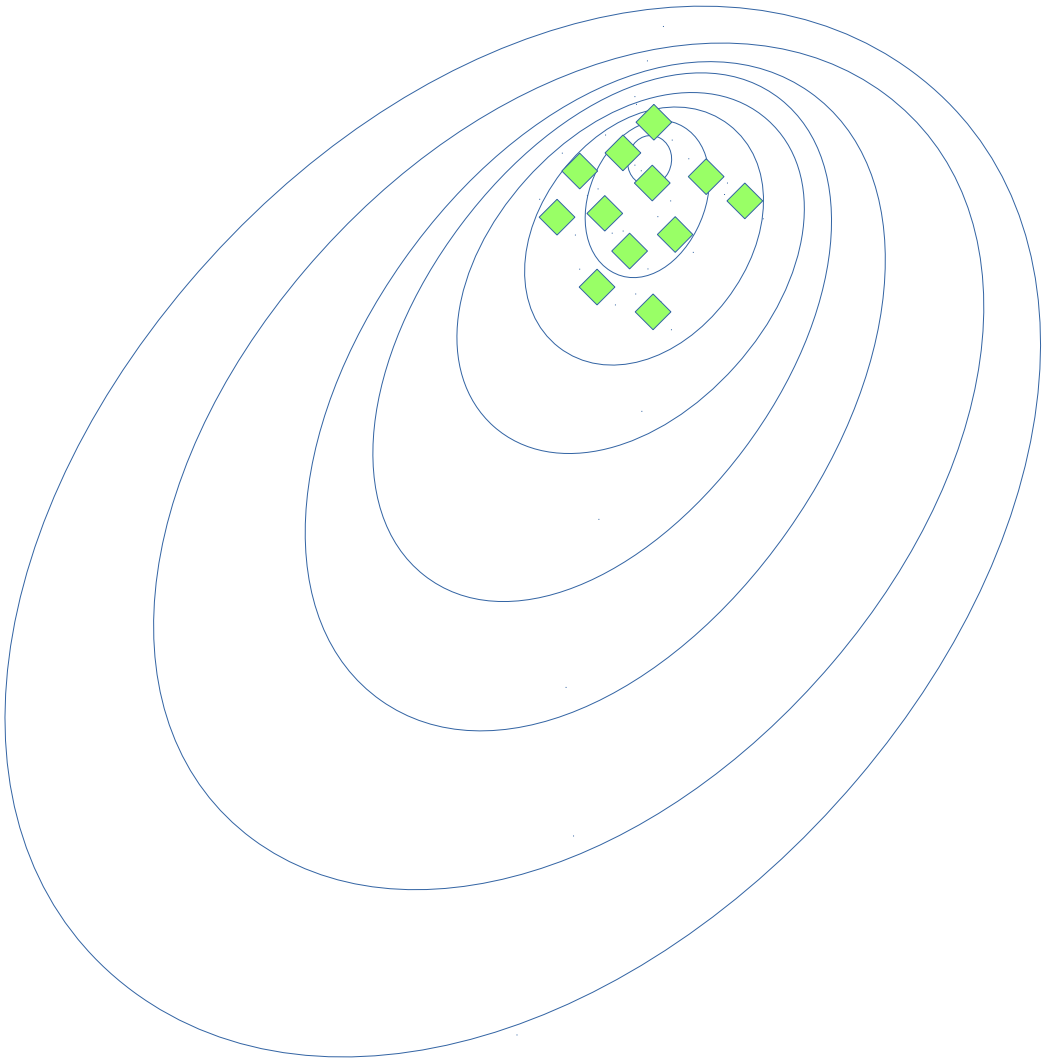
Crossentropy Method



Crossentropy Method



Crossentropy Method



Grim reality

If your environment has infinite/large state space

Approximate crossentropy method

- 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

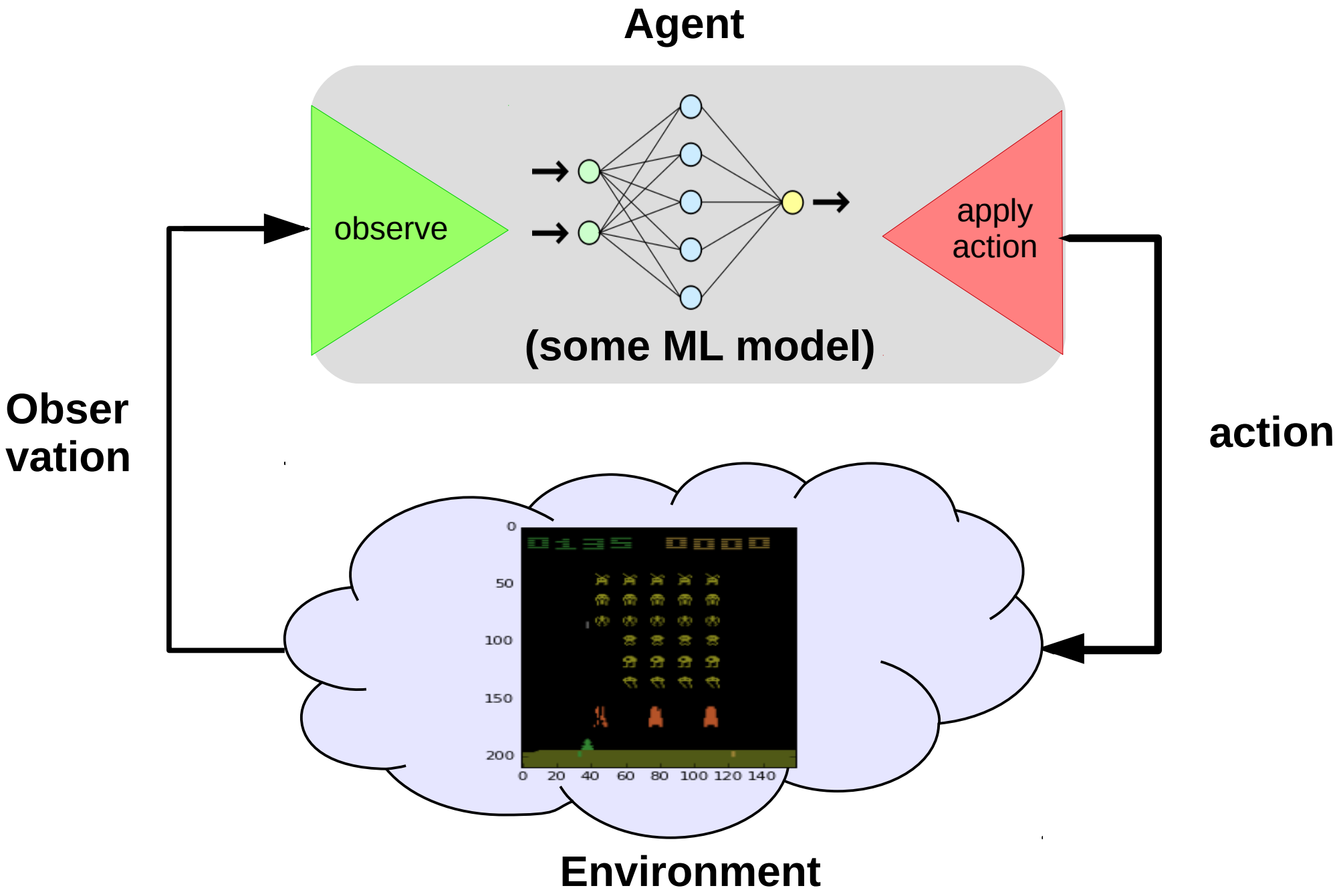
$$Elite = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_k, a_k)]$$

Approximate crossentropy method

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

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

Approximate RL



Approximate crossentropy method

- 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 crossentropy method

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 crossentropy method

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 crossentropy method

- Initialize NN weights $W_0 \leftarrow random$

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

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

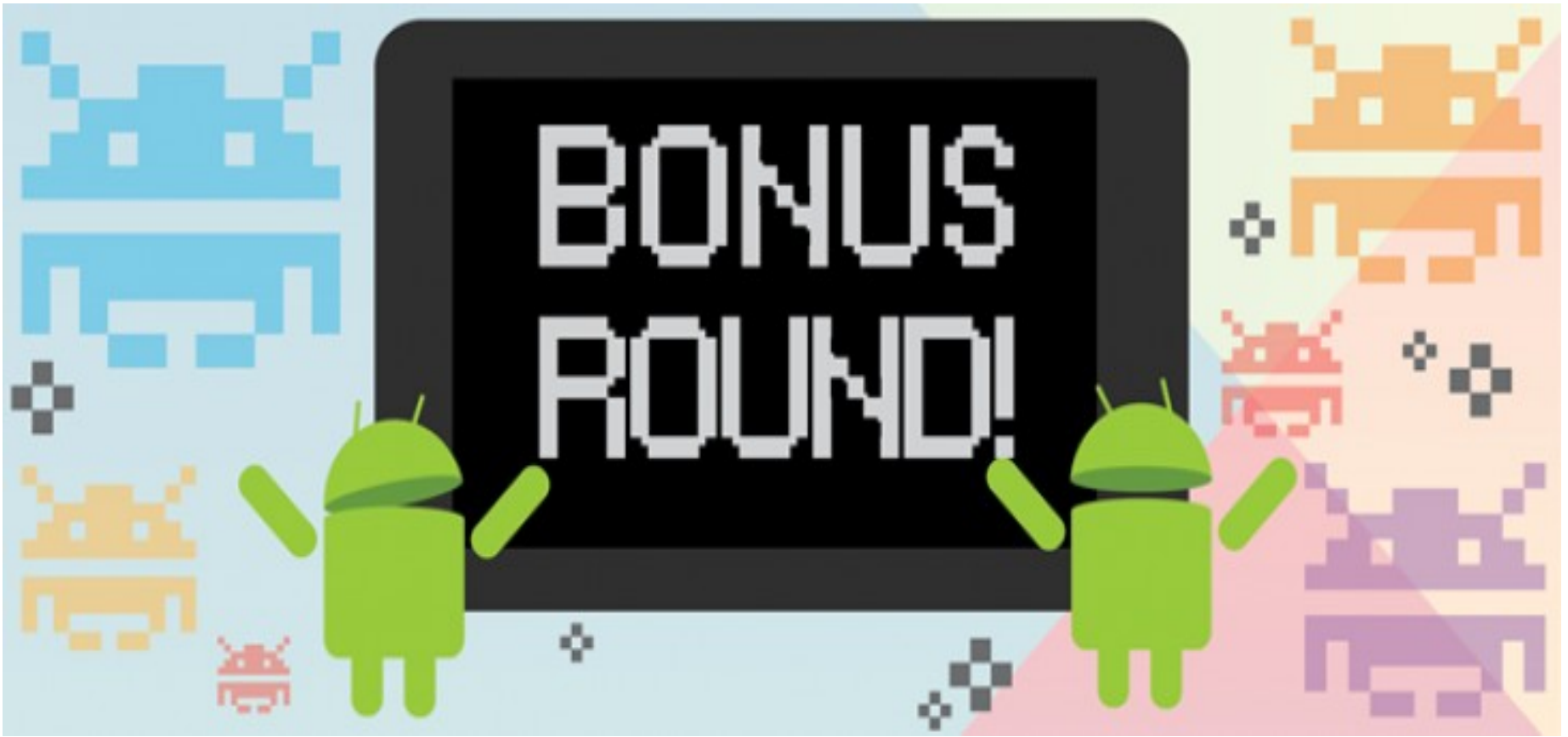
- Continuous state space **model = MLPRegressor()**
- Model $\pi_W(a|s) = N(\mu(s), \sigma^2)$
  - Mu(s) is neural network output
  - Sigma is a parameter or yet another network output
- Loop:
  - Sample N sessions
  - elite = take M best sessions and concatenate
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i,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

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 (later)


BONUS
ROUND!

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

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

Evolution Strategies

- Introduce distribution over weights

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

- Maximize expected reward

$$J = \mathop{E}_{\theta \sim N\left(\theta|\mu,\sigma^2\right)} \mathop{E}_{\tau(\theta)=s,a,s',a',\ldots} R\left(\tau\right)$$

Evolution Strategies

- Expected reward (using math. Expectation)

$$J = \mathop{E}_{\theta \sim N(\theta|\mu,\sigma^2)} \; \mathop{E}_{\tau(\theta)=s,a,s',a',\ldots} R(\tau)$$

Q: How can we estimate J in practice?

for large/infinite state space

Evolution Strategies

- Expected reward (using math. Expectation)

$$J = \mathop{E}_{\theta \sim N(\theta|\mu,\sigma^2)} \; \mathop{E}_{\tau(\theta)=s,a,s',a',\ldots} R(\tau)$$

- Approximate with sampling

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

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

Evolution Strategies

- Expected reward (using math. Expectation)

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

- Approximate with sampling

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

sample full episodes

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

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

- What we need

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

Q: you compute ∇J w.r.t. what?

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

- What we need

$$\nabla J = \frac{\partial J}{\partial \mu, \partial \sigma^2}$$

Gradient w.r.t. distribution parameters

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

- What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

- What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

Evolution Strategies

- Expected reward (using math. expectation)

$$J = \int N(\theta|\mu, \sigma^2) \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

- What we need

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

Can't estimate by sampling!
∇ 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

$$\nabla J = \int \nabla [N(\theta|\mu, \sigma^2)] \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

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

Evolution Strategies

Analytical inference

$$\nabla J = \int [N(\theta|\mu, \sigma^2) \cdot \nabla \log N(\theta|\mu, \sigma^2)] \cdot \int P(\tau|\theta) R(\tau) d\tau d\theta$$

Q: How can we estimate ∇J in now?

Evolution Strategies

Analytical inference

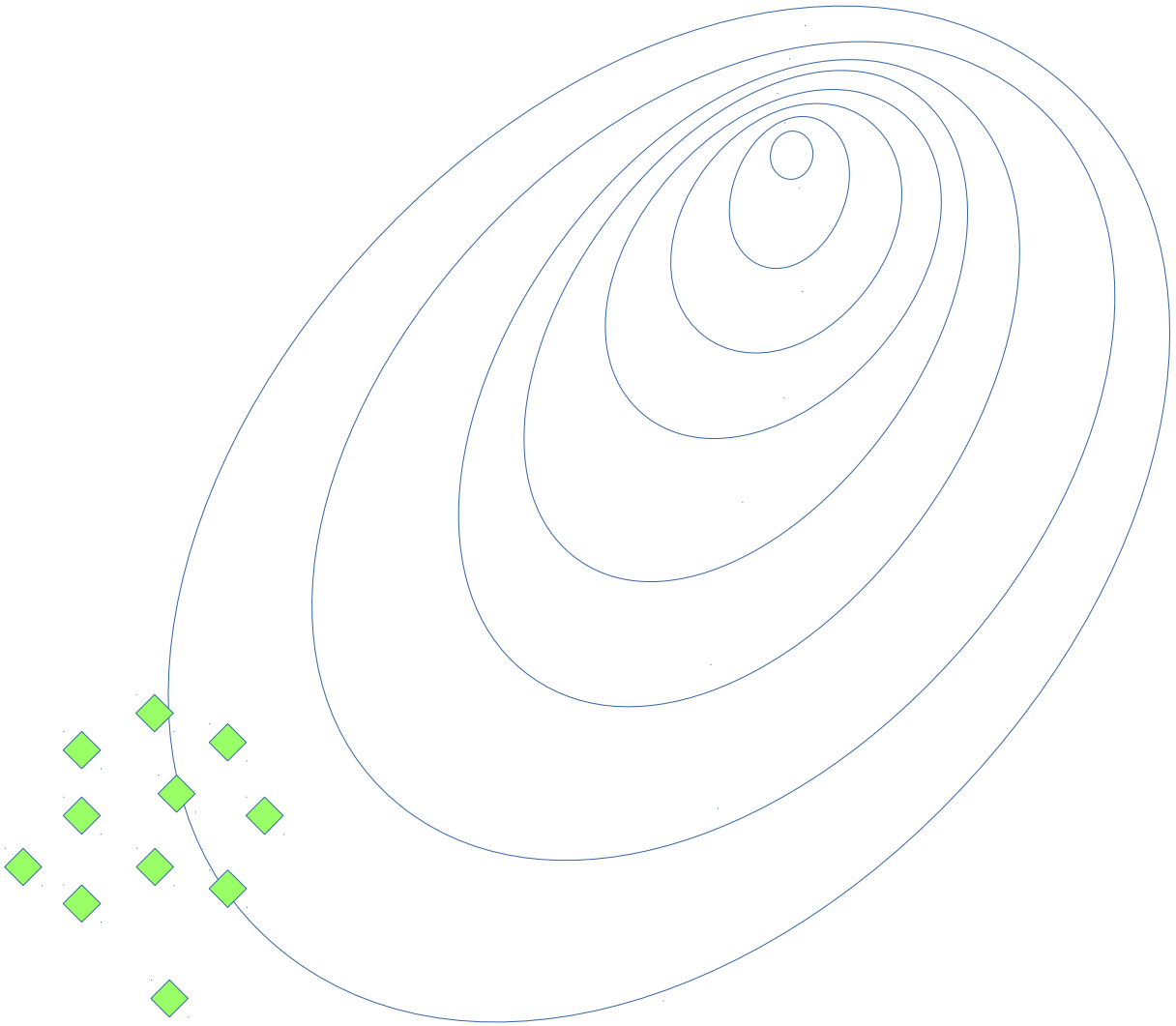
$$\nabla J = \mathop{E}_{\theta \sim N(\theta|\mu,\sigma^2)} \nabla \log N(\theta|\mu,\sigma^2) \cdot \mathop{E}_{\tau(\theta)=s,a,s',\ldots} R(\tau)$$

Sampled estimate

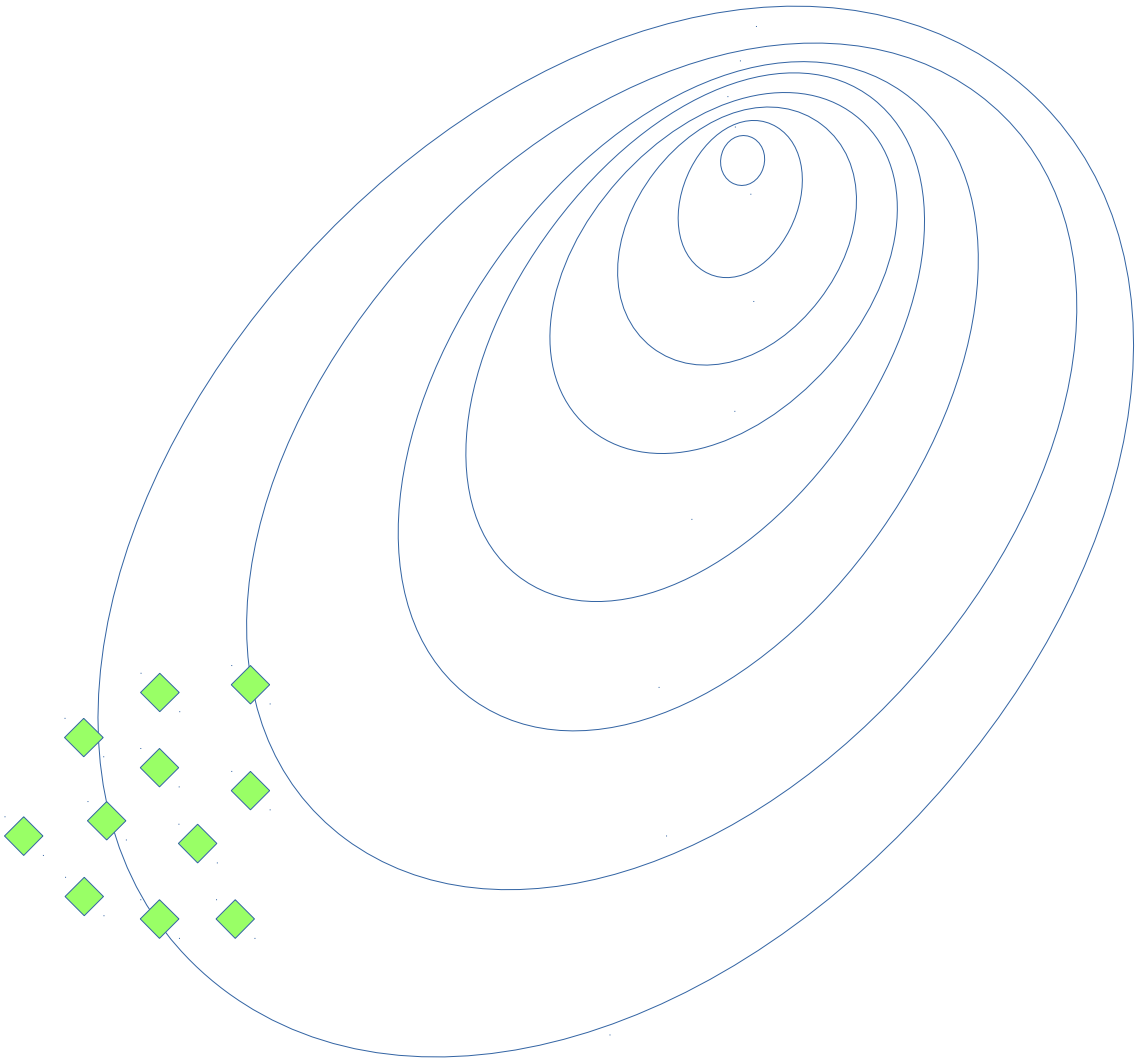
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log N(\theta|\mu,\sigma^2) \cdot \sum_{s,a \in z_i} R(s,a,\ldots)$$

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

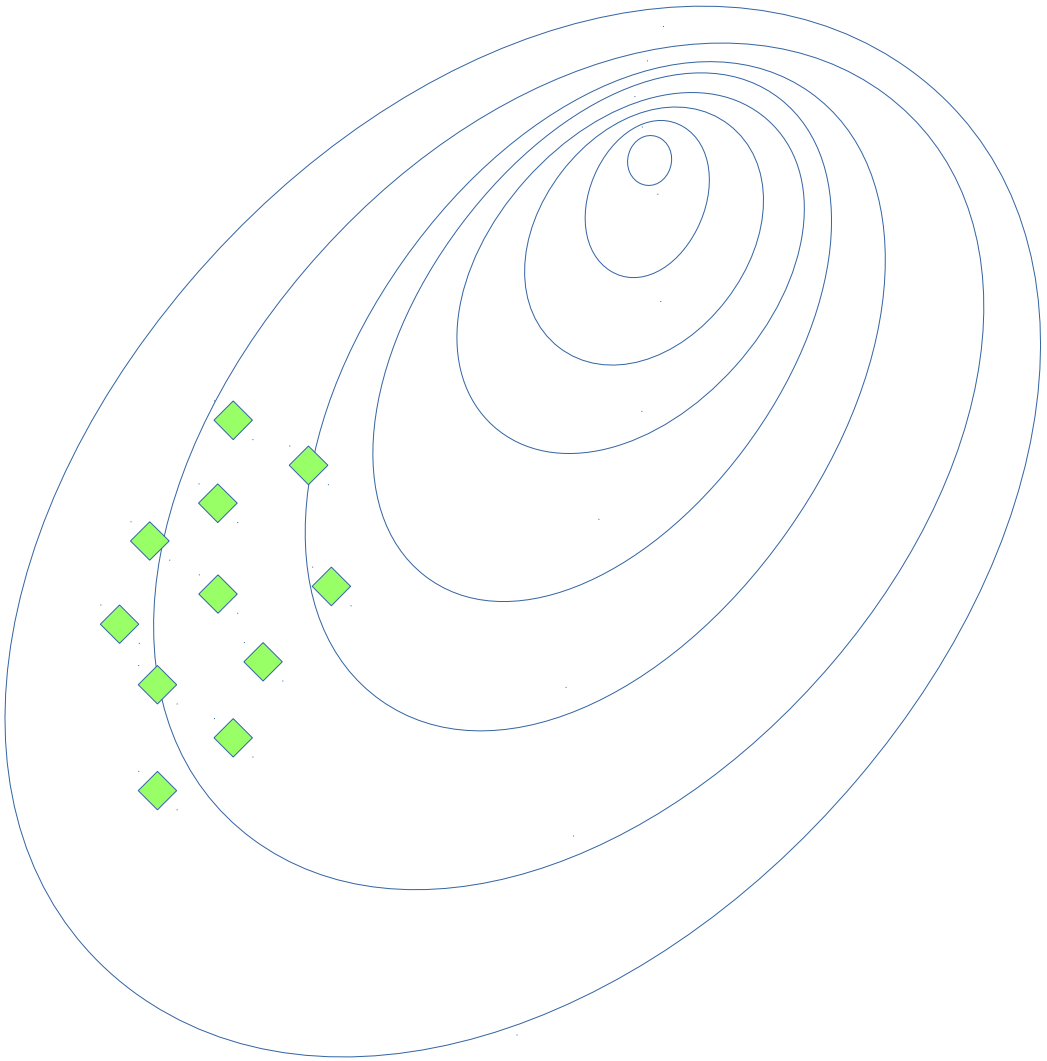
Evolution Strategies



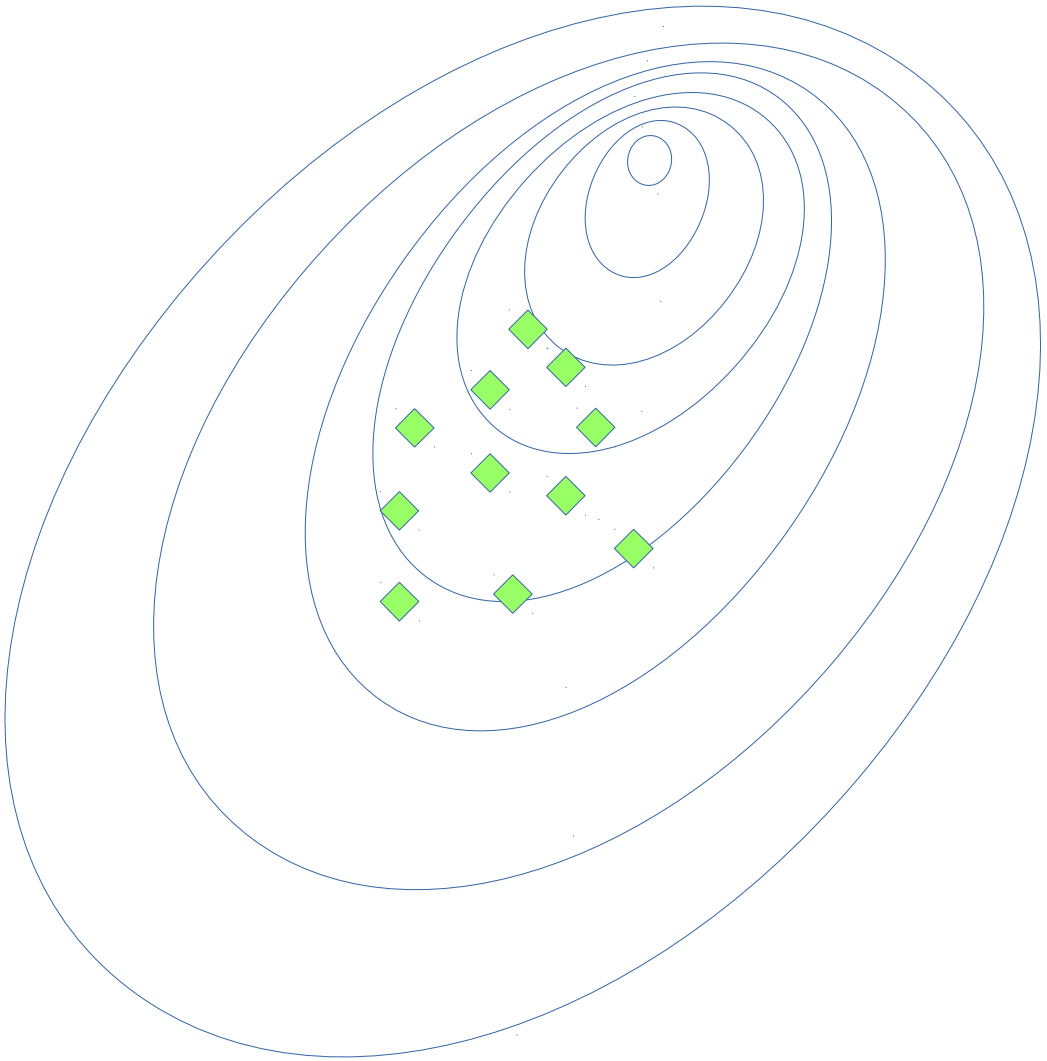
Evolution Strategies



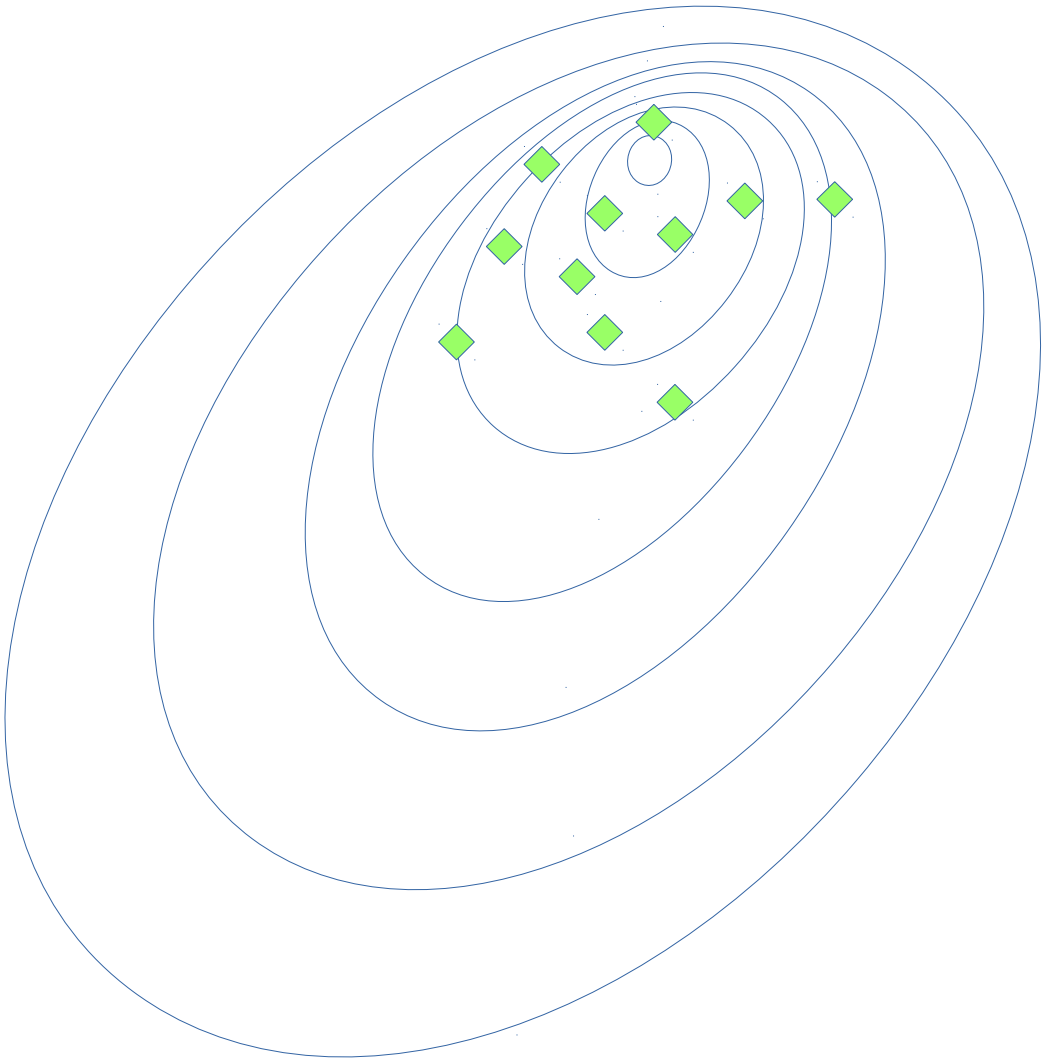
Evolution Strategies



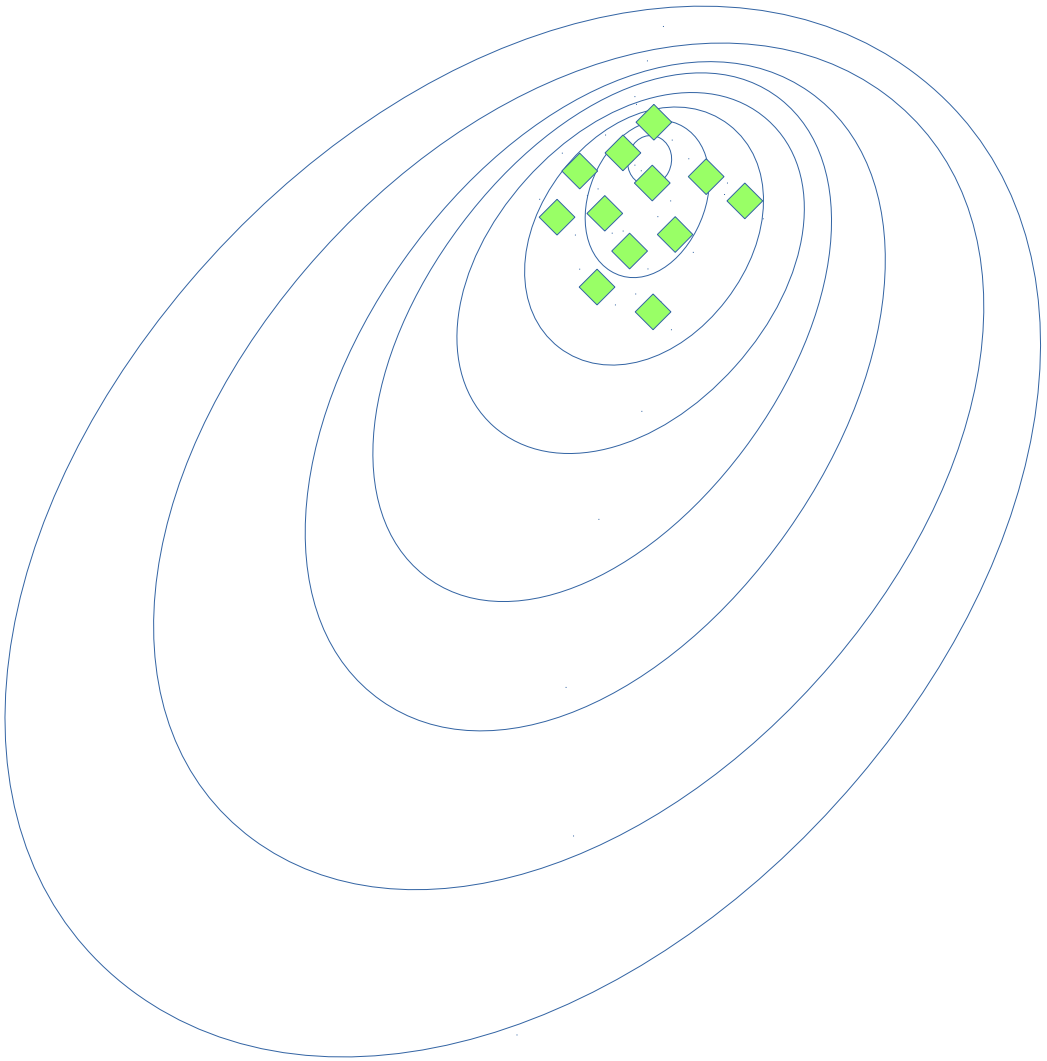
Evolution Strategies



Evolution Strategies



Evolution Strategies



Evolution strategies

Algorithm

1. Initialize μ_0, σ_0^2

2. Forever:

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

Evolution strategies

Algorithm

1. Initialize μ_0, σ_0^2

2. Forever:

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

$$\mu = \mu + \alpha \cdot \nabla_{\mu} J \qquad \sigma^2 = \sigma^2 + \alpha \cdot \nabla_{\sigma^2} J$$

Evolution strategies

Algorithm

1. Initialize μ_0, σ_0^2

2. Forever:

$$\theta = \mu + \sigma \cdot \xi \; : \; \xi \sim N(0,1)$$

sample $\theta_i, \tau_i, R_i : R_i = R(\tau_i) = R(\tau(\theta_i))$

$$\mu = \mu + \alpha \cdot \frac{1}{N \cdot \sigma} \cdot \sum_i \xi_i \cdot R_i \qquad \sigma^2 = \ldots$$

Reward baselines

TL;DR normalize reward

$$\underset{N(\theta|\mu,\sigma^2)}{argmax}\ E\ R = \underset{N(\theta|\mu,\sigma^2)}{argmax}\ E\ \frac{R-mean}{std}$$

$$A = \frac{R - E\,R}{Var\ R}$$

Evolution strategies

Features

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

Evolution strategies

Features

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

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

Q: You have 1000 CPUs.

Optimize this formula!

Evolution strategies

Features

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

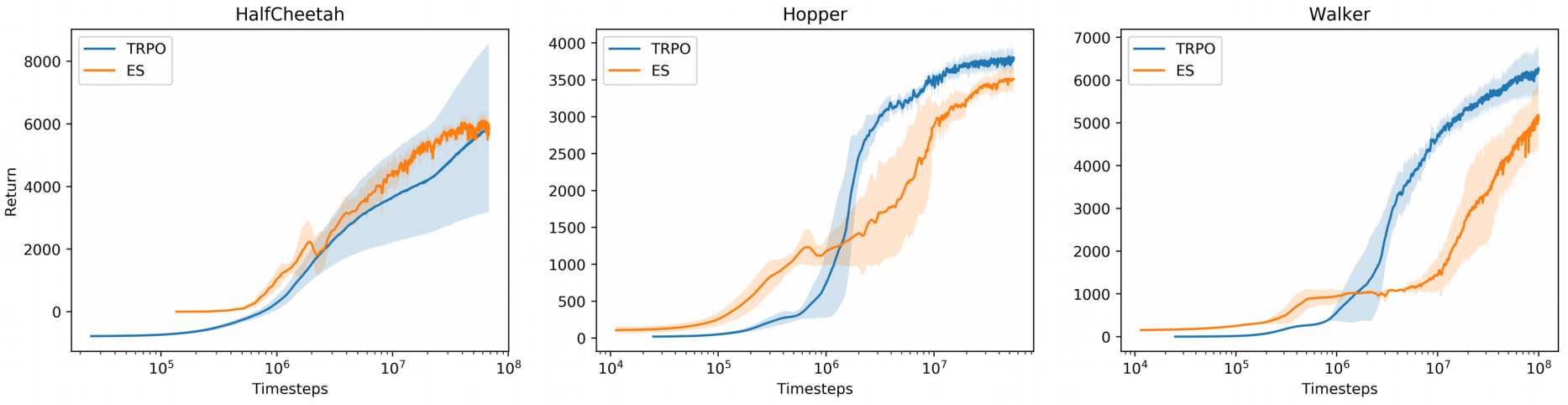
$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \nabla \log N\left(\theta | \mu, \sigma^2\right) \cdot \sum_{\tau_i = s, a, s', \ldots} R\left(\tau_i\right)$$

You can compute every session in parallel

Evolution strategies

Features

- A general black box optimization
- Some results on gym



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

Common drawback

Both CEM, ES and all similar methods

- Train from full sessions only
- Require a lot of samples

Common drawback

Both CEM, ES and all similar methods

- Train from full sessions only
- Require a lot of samples