Reinforcement learning I

DMIA 04.17

Why should I care?

Yandex
Data Factory

LAMBDA

British Hedgehog
Preservation Society

Terms

- **Ask**!
  - Even if the question feels stupid.
  - Chances are, half of the group is just like you.
  - If it's necessary, interrupt the speaker.

- **Contribute!**
  - Found an error? Got useful link? Ported the seminar to py3 from py2? Answered peer's question in the chat?
  - You're awesome!

<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 $\left(x, y\right)$

[banner,page], ctr

- algorithm family $a_{\theta}\left(x\right) \rightarrow y$

linear / tree / NN

- loss function $L\left(y, a_{\theta}\left(x\right)\right)$

MSE, crossentropy

Find:

$$\theta' \leftarrow argmin_{\theta} L\left(y, a_{\theta}\left(x\right)\right)$$

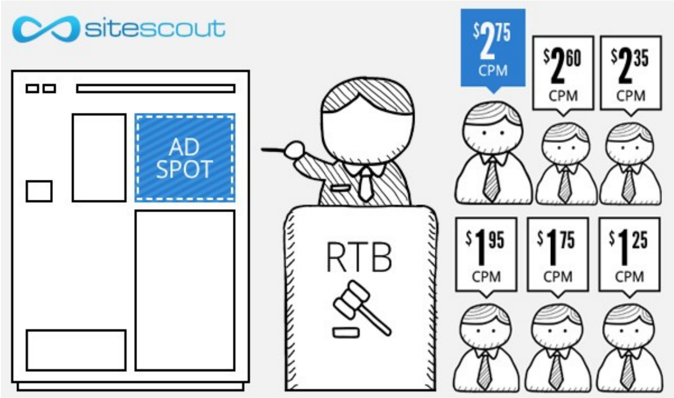
Supervised learning

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

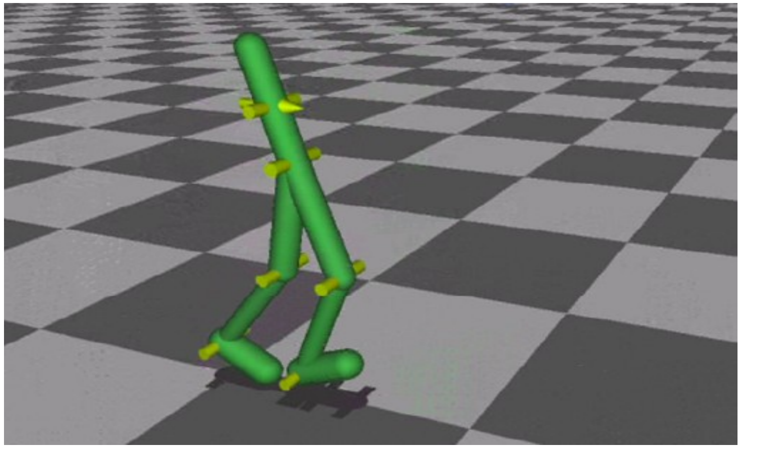
Ideas?

Giant Death Robot (GDR)

Great... except if we have no reference answers

We have:

- Evil humanoid robot
- A lot of spare parts
  to repair it :)

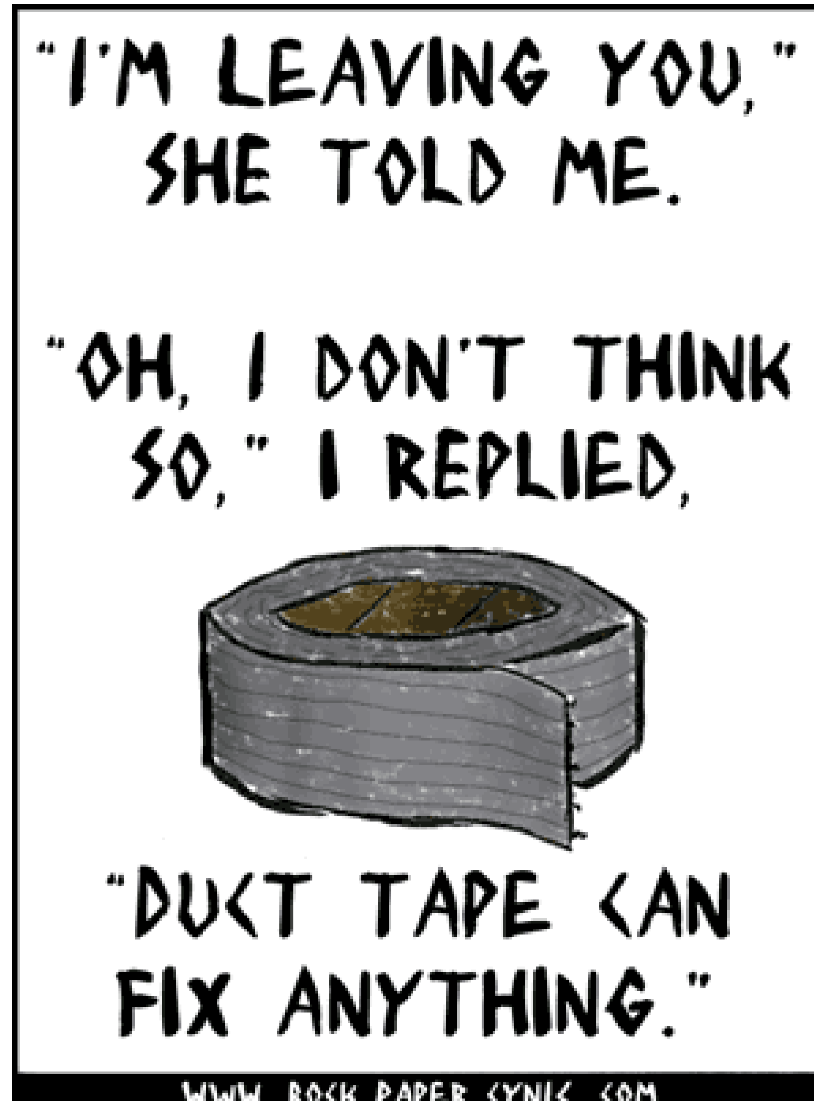


We want:

- ~~Enslave humanity~~
- Learn to walk forward

Ideas?

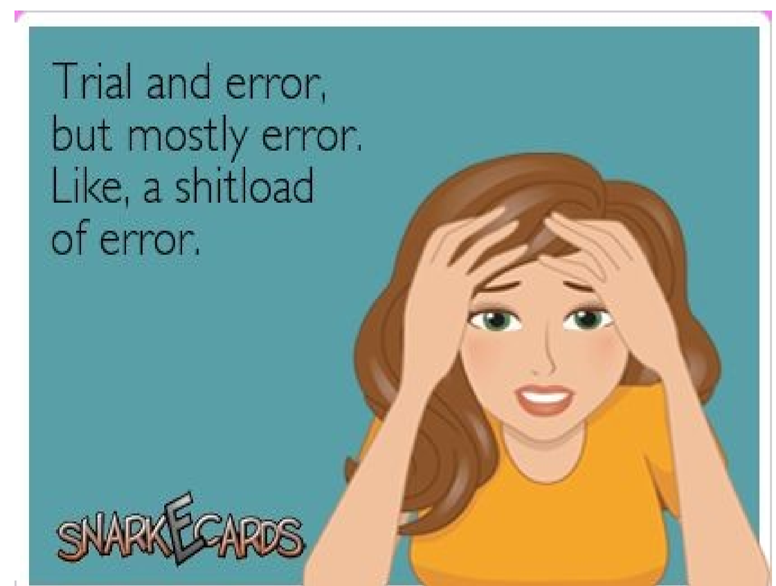
Duct tape approach



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



Duct tape approach

Problem 1:

- What exactly does the “optimal action” mean in the Giant Death Robot setting?

Push yourself forward as far as you can at each tick

VS

Do what allows you to walk farther over next N seconds

Duct tape approach

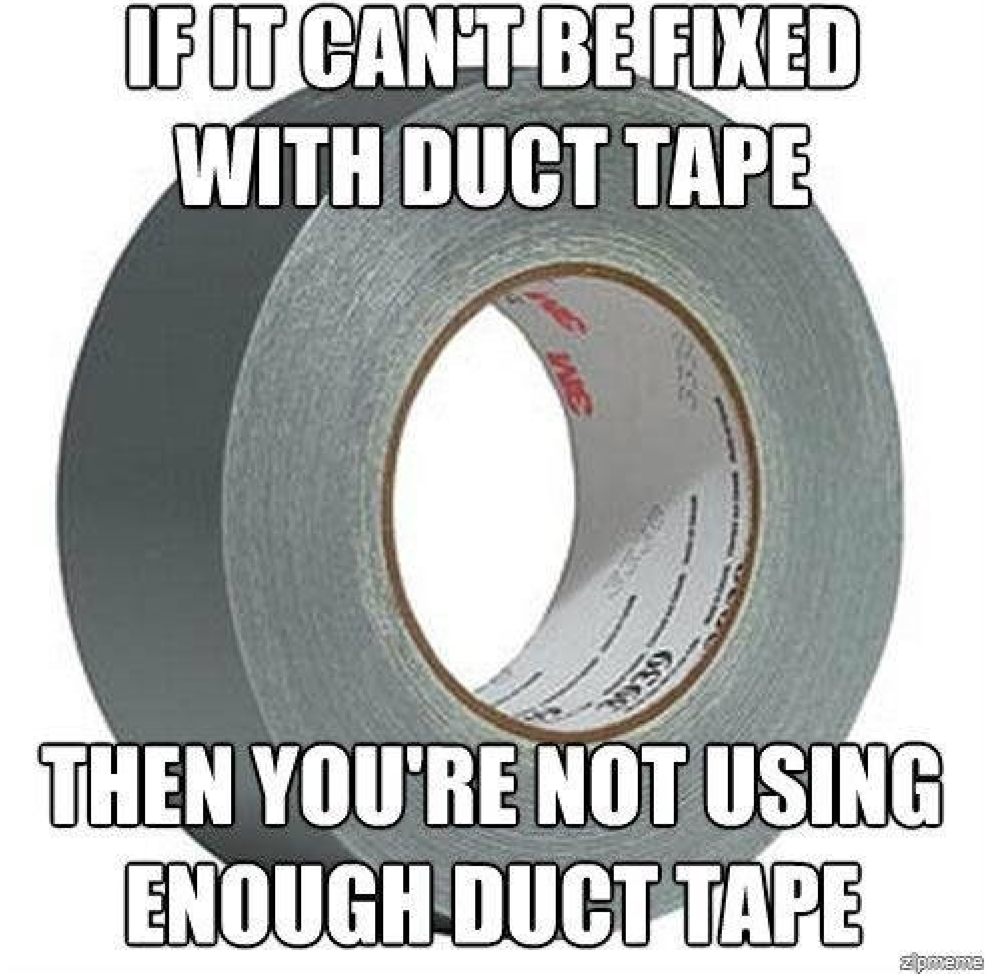
Problem 2:

- If you only act by the “current optimal” policy, you may never hit the global optimum.

- If your learned to fall down and crawl forward, that it will never get examples of how to walk because it always crawls.

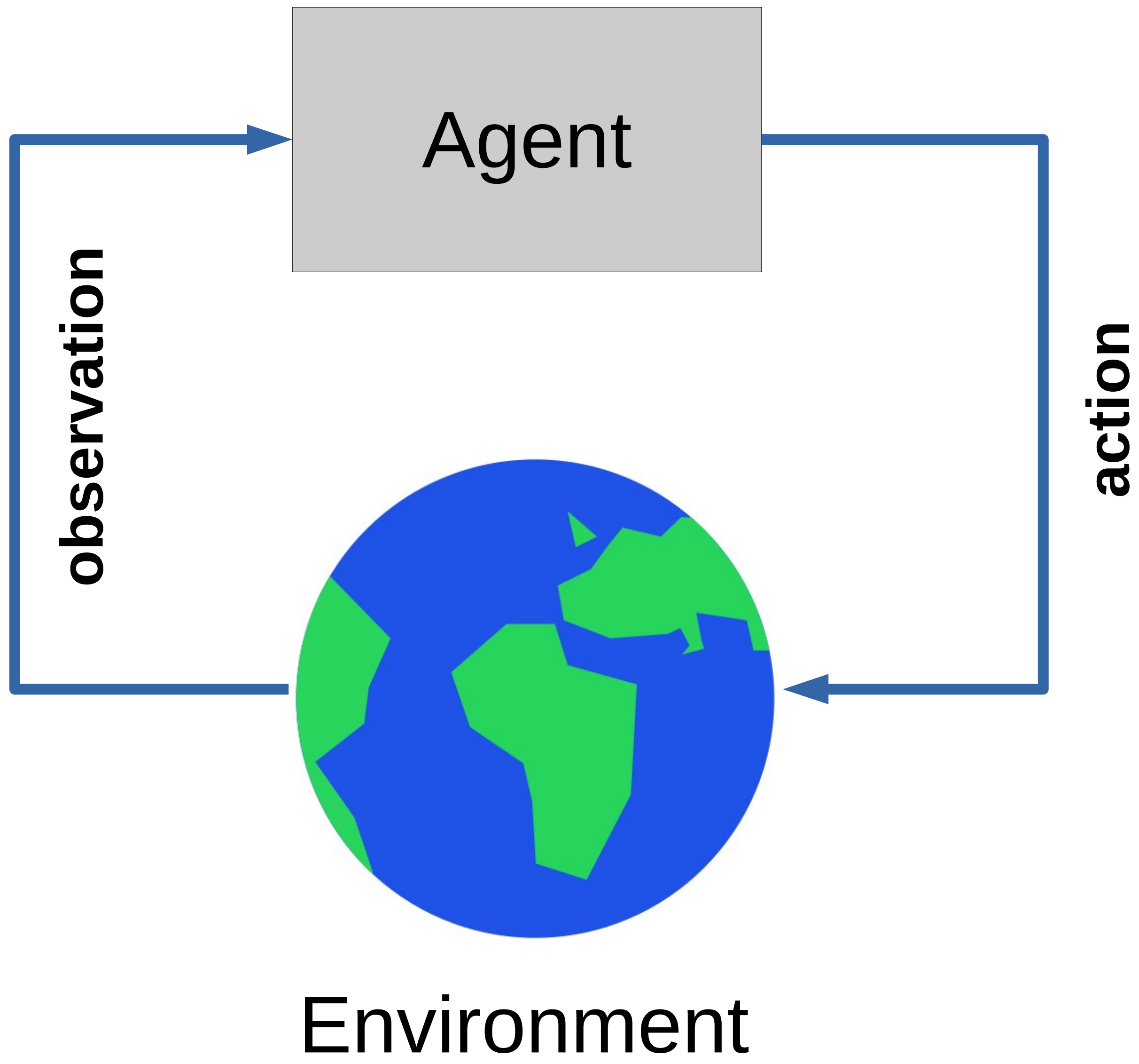
- **Ideas?**

Duct tape approach

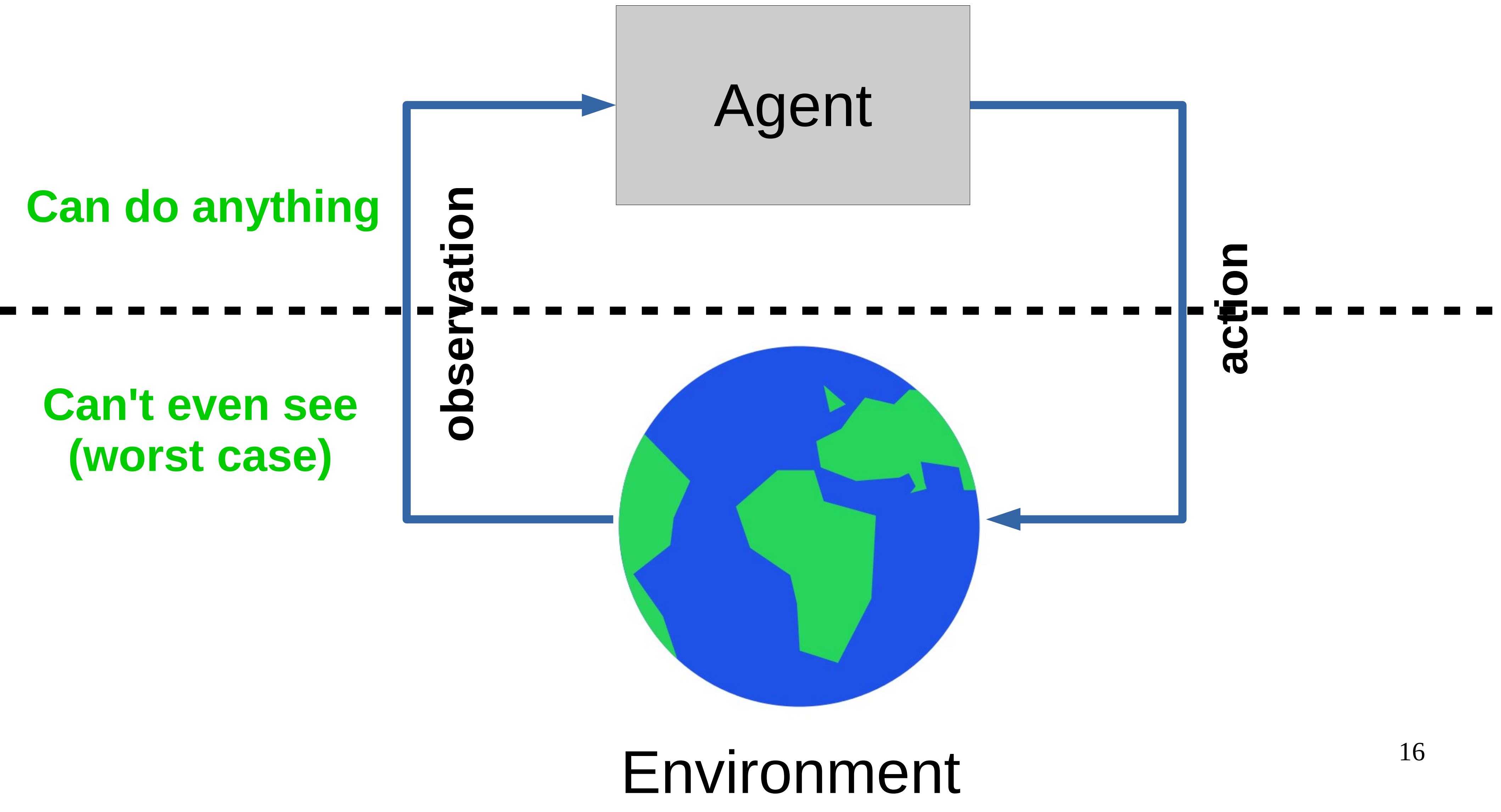


What is: reinforcement learning

What is: reinforcement learning



What is: reinforcement learning



What is: reinforcement learning

- Agent interacts with environment
  - site interacts with user
  - robot interacts with the physical world

- Feedback on agent performance
  - Agent receives feedback on **his** performance
  - Usually a real number (more=better)

What is: reinforcement learning

- Agent interacts with environment
  - site interacts with user
  - robot interacts with the physical world

You get to pick actions, not just observe data

- Feedback on agent performance
  - Agent receives feedback on **his** performance
  - Usually a real number (more=better)

Not given optimal actions as feedback

Reinforcement learning Vs regular ML

- Algorithm can influence what samples it gets
- Data is not i.i.d.
- Goal is to learn optimal policy
  - (observation → what to do)

Reinforcement learning Vs regular ML

- Algorithm can influence what samples it gets

  Similar to “active learning”

- Data is not i.i.d.

  Many optimization/inference require i.i.d.

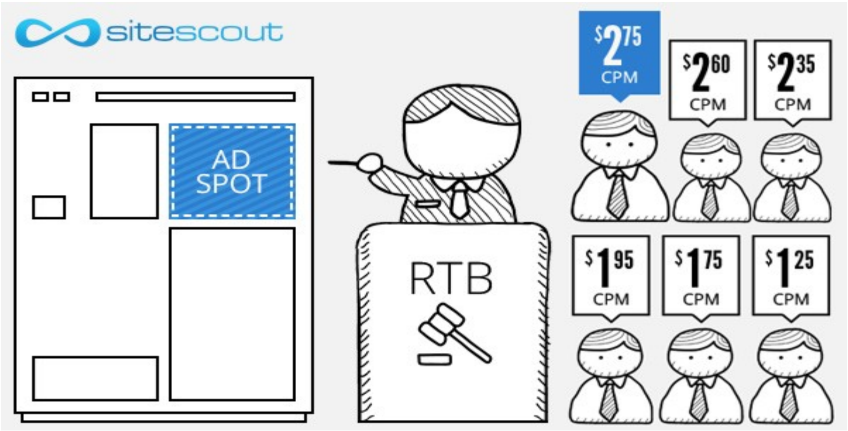
- Goal is to learn optimal policy
  - (observation → what to do)

RL can be viewed as supervised learning*

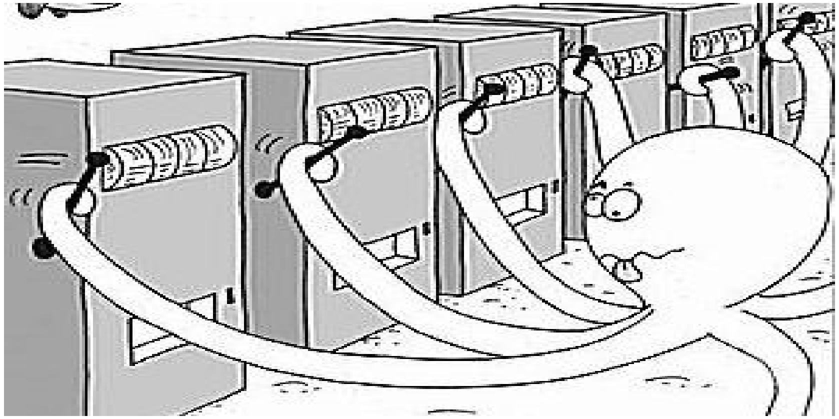
* with some duct tape

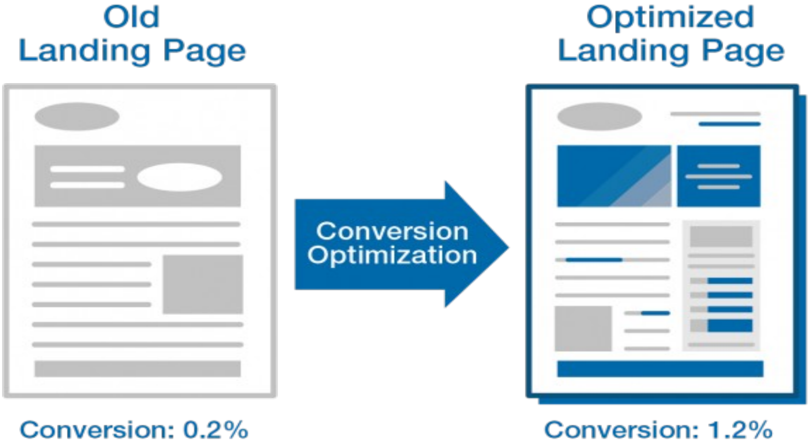
Reality check: web

- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend items to users

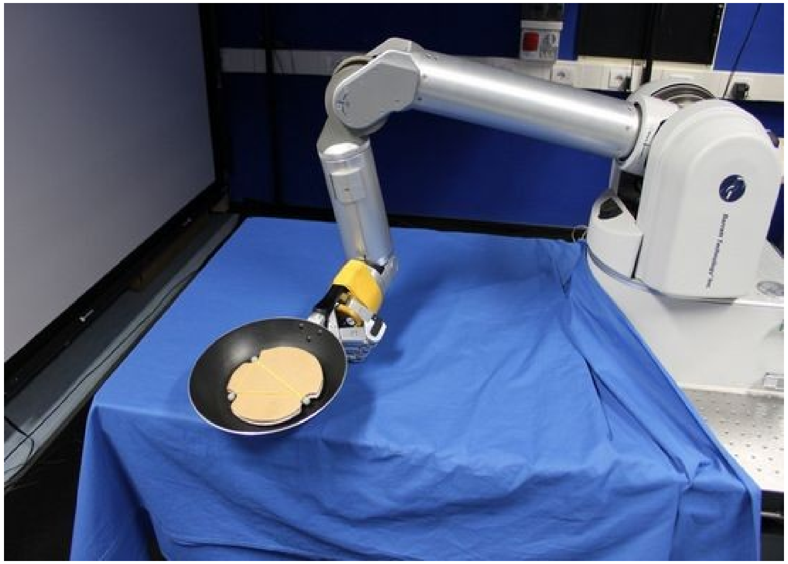


- **Example**
  - Observation – user features
  - Action – show banner #i
  - Feedback – did user click?





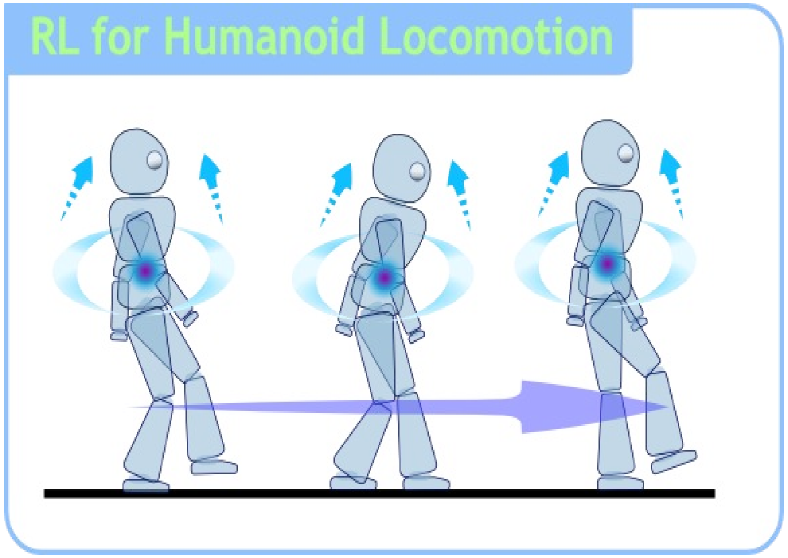
Reality check: dynamic systems



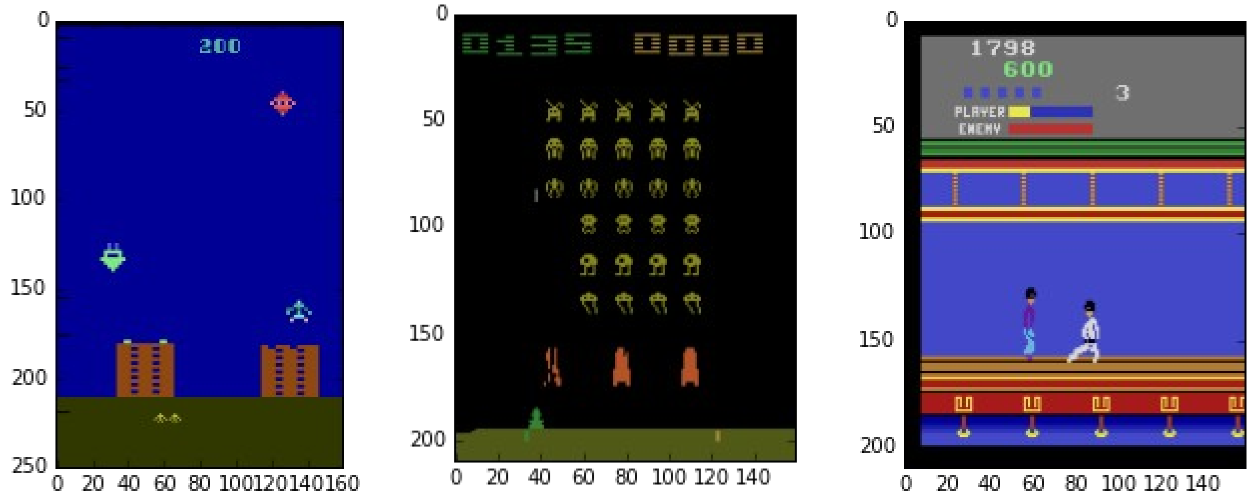
Reality check: MOAR

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

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



~~Reality~~ check: videogames

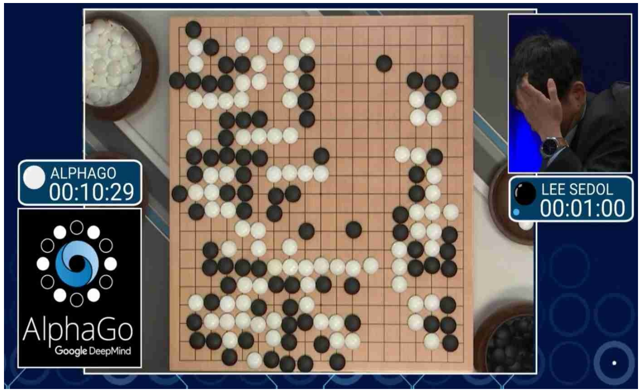


- **Trivia:** What are observations, actions and feedback?

Other use cases

- Personalized medical treatment

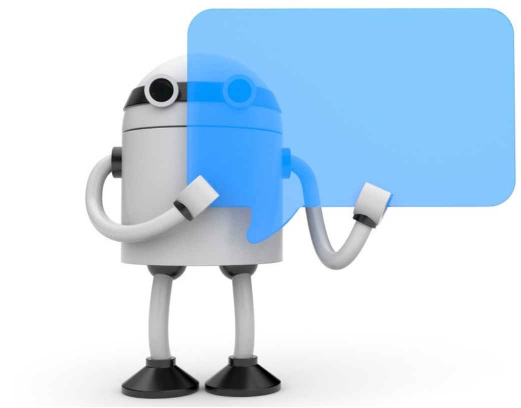
- Even more games (Go, chess, etc)



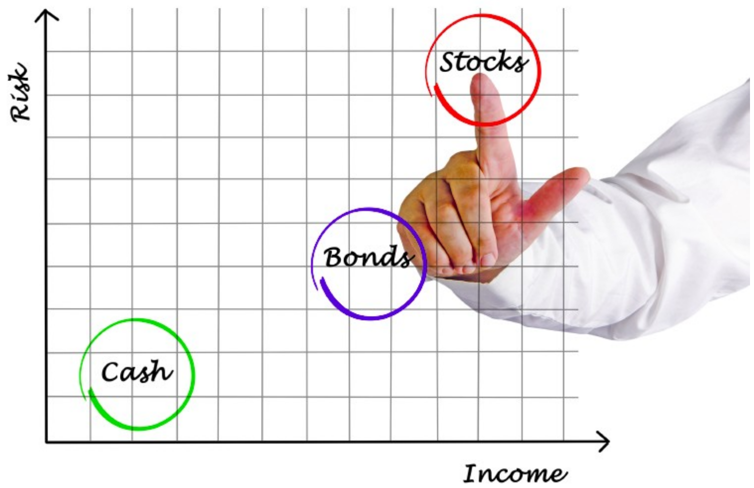
- **Trivia:** What are observations, actions and feedback?

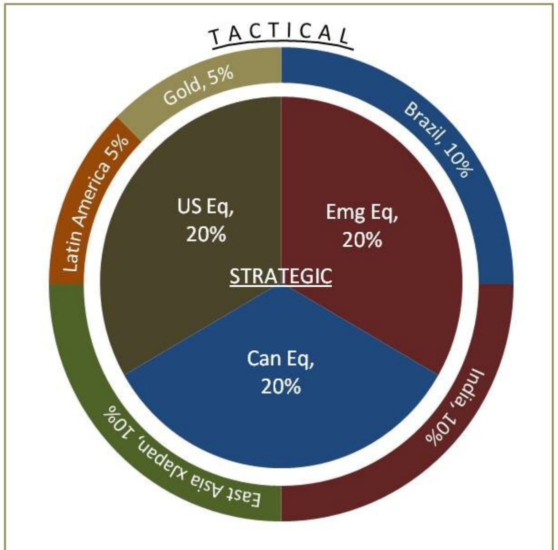
Other use cases

- Conversation systems (additional goals)

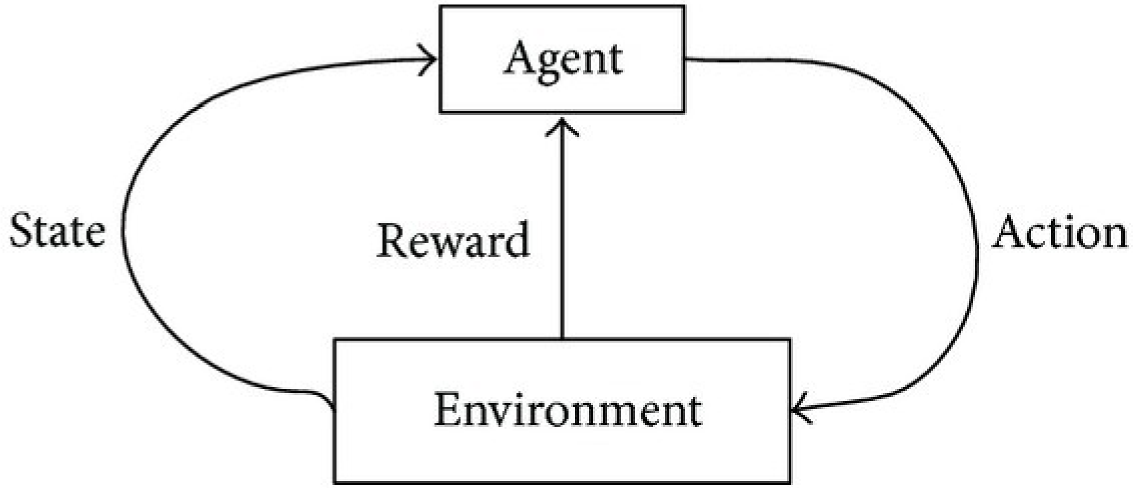


- Portfolio management (aka asset allocation)





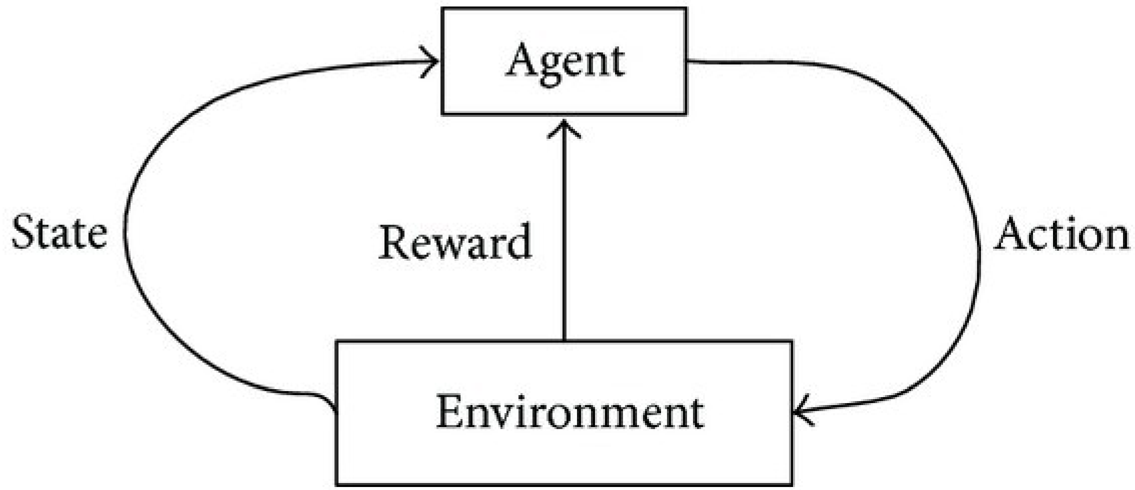
The MDP formalism



Classic MDP(Markov Decision Process)
Agent interacts with environment

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

The MDP formalism



Classic MDP(Markov Decision Process)
Agent interacts with environment

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

Markov assumption

Optimal policy (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

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

Context: FrozenLake

- A grid world with a goal tile and ice holes

SFFF (S: starting point, safe)
FHFH (F: frozen surface, safe)
FFFH (H: hole, fall to your doom)
HFFG (G: goal, where the frisbee is located)

Quiz: what states, actions and rewards are used?

Model-based RL

- Imagine you have an accurate model of the world
  - e.g. physics model for robot
- You know exactly:
  - $P(s_{t+1} | s_t, a_t)$
  - $R(z)$
  - Forall $s \in S \quad a \in A$

- How can you get optimal action?

Black box optimization

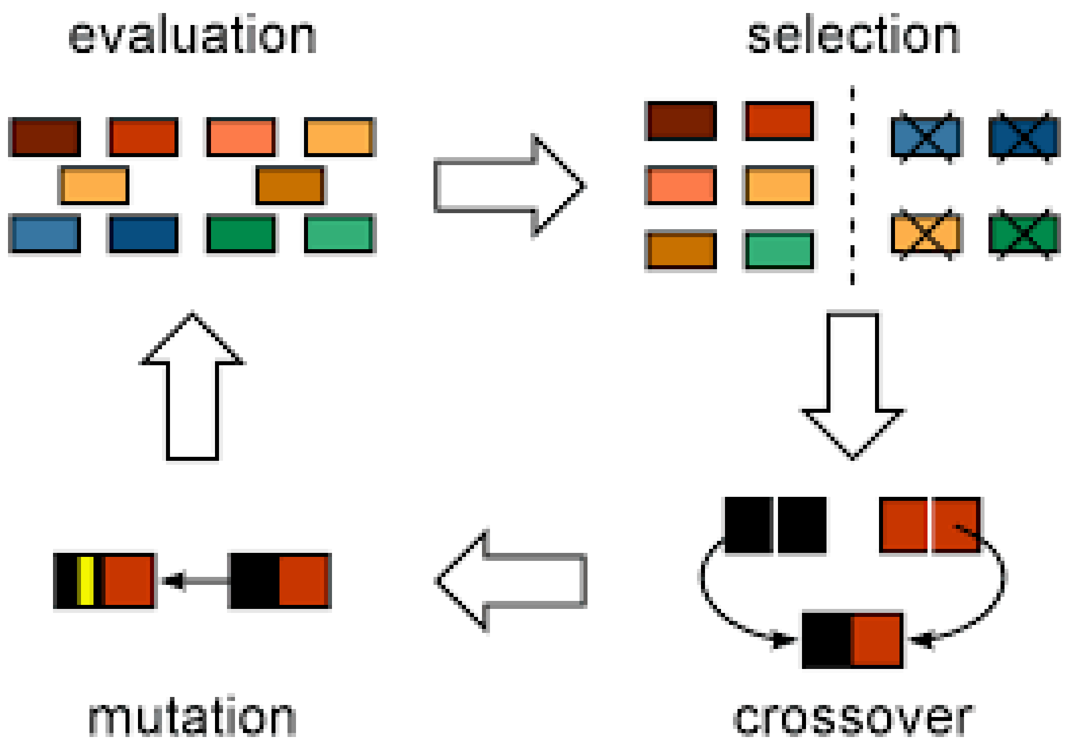
- Maximize score over policy
- No gradient
- Naive solution: iterate over all policies
  - Any problems with that?

Black box optimization

- Maximize score over policy
- No gradient
- Naive solution: iterate over all policies
  - Bizillion candidates
- Efficient algorithms for particular problems
- Heuristics!

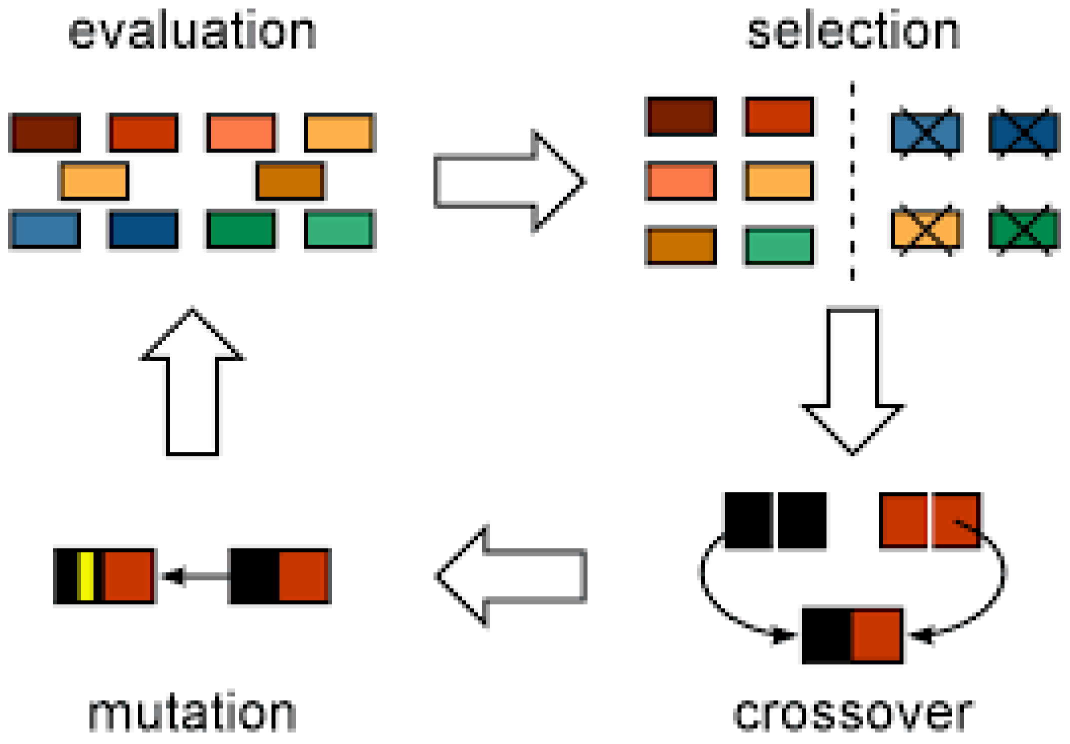
Genetic algorithms

- biologically inspired heuristic
- maintain a population of policies
- reproduce (crossover) → mutate → prune



Genetic algorithms

- biologically inspired heuristic   no guarantees
- maintain a population of policies
- reproduce (crossover) → mutate → prune

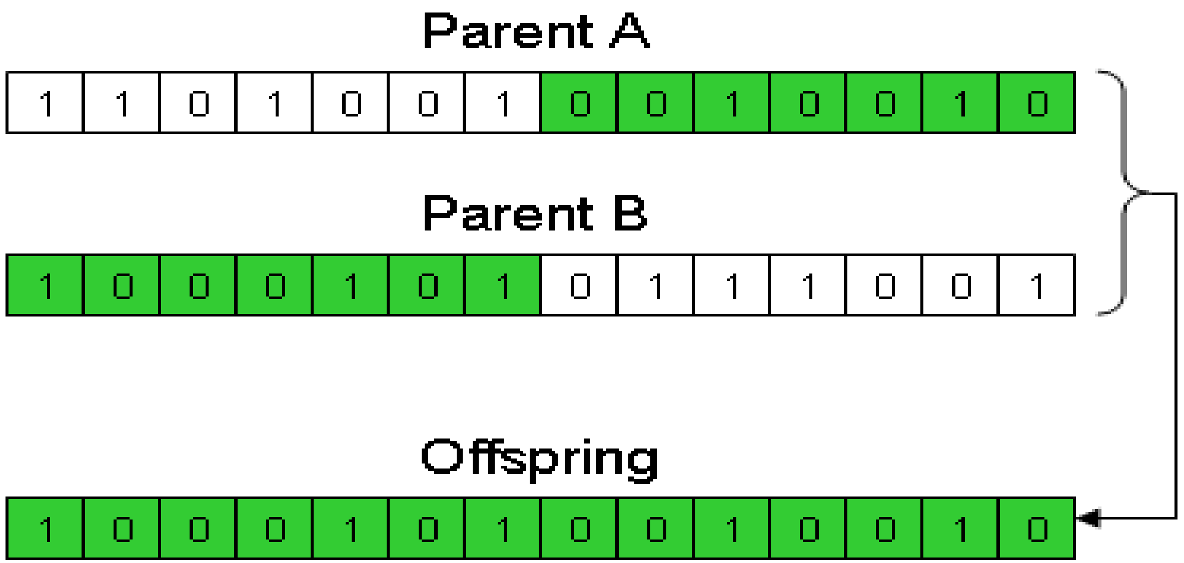


Genetic algorithms

- Keep a pool of N policies
- On each step,
  - Modify existing pool by mutating/mixing strategies
  - compute fitness ~ how good each policy is
  - Take top-N policies with highest fitness to next step

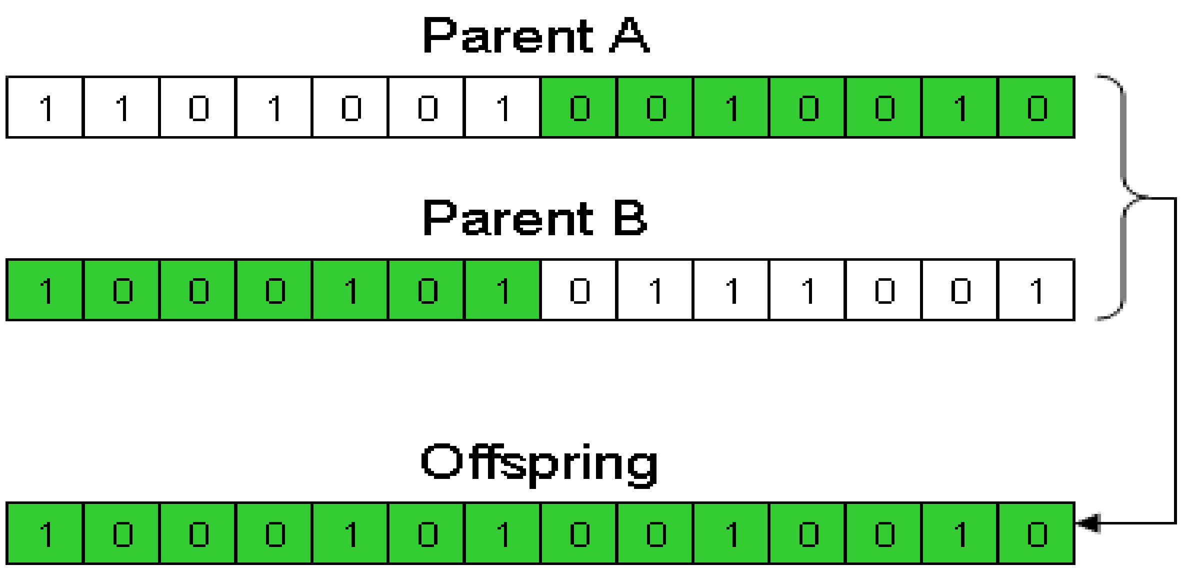
Crossover

- Take 2 random policies ("**parents**")
- For each state, flip a coin
  - If **heads**, take action from the first parent
  - If **tails**, take action from the **second parent**



Crossover

- Take 2 random policies ("**parents**")
- For each state, flip a coin
  - If **heads**, take action from the first parent → If **heads**, take action from the **first parent**
  - If **tails**, take action from the **second parent**



Any other ways to make it?

Genetic Algorithms

- Cons
  - Convergence not guaranteed
  - Requires a lot of samples
  - A lot of parameter tuning
  - Hard to scale on large state spaces

- Pros
  - It sorta sometimes works
  - Very easy to scale (multi-cpu, cluster)
  - You get to be a god!

Crossentropy method

- Sample N(e.g. 100) sessions
- Take M(e.g. 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 to get actual score

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

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

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
  - $W_{i+1} = W_i + \alpha \nabla [\sum_{s_i, a_i \in Elite} \log \pi_{W_i}(a_i | s_i)]$

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

No time to explain

- CEM is a stochastic optimization method
  - Even got probabilistic guarantees of convergence
- Connected to general EM algorithm
- Works embarrassingly well (c) joshu

Problem with GA & CEM

- need a full sessions to start learning
  - A LOT of those to learn reliably
- requires a lot of interaction
  - A lot of crashed robots / simulations

Seminar