Reinforcement learning

Episodes 0 & 2

Introduction & Temporal Difference

Yandex Data Factory

LAMBDA

British Hedgehog Preservation Society

Supervised learning

Given:

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

Find:

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

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

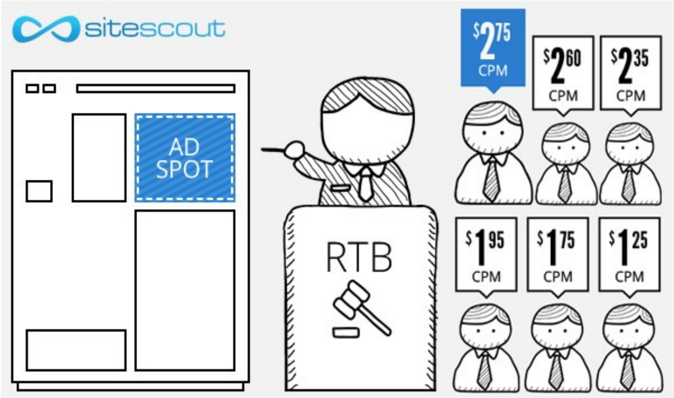
Great... except if we have no reference answers

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

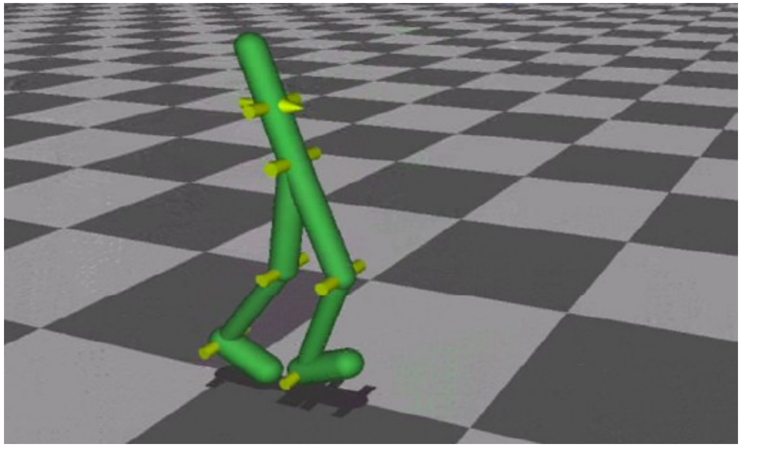
Ideas?

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 :)

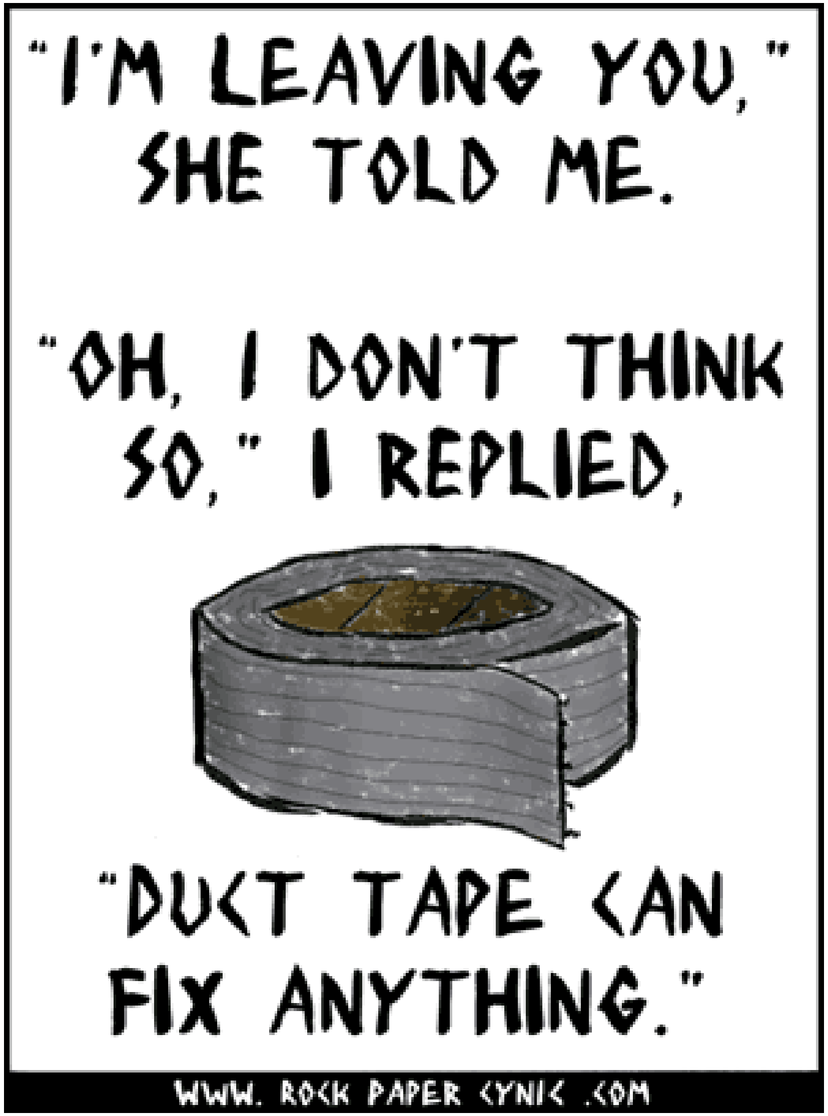


We want:

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

Ideas?

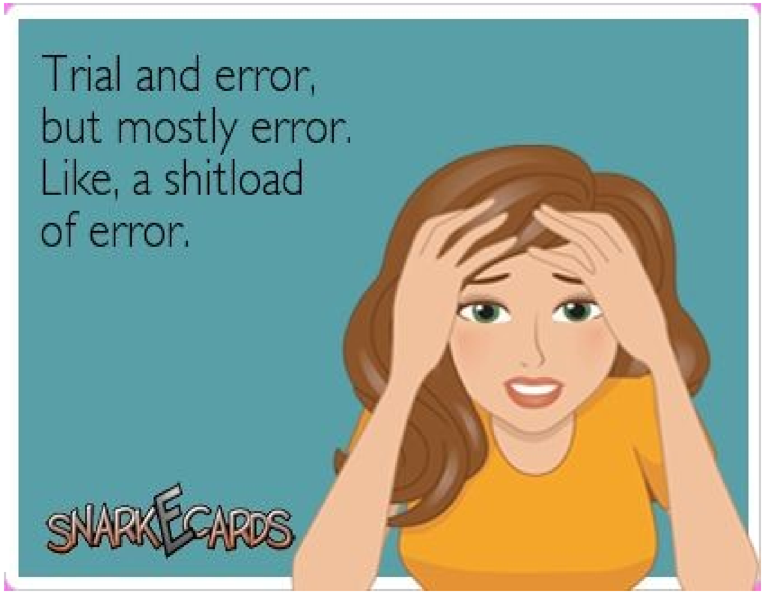
Duct tape approach



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:

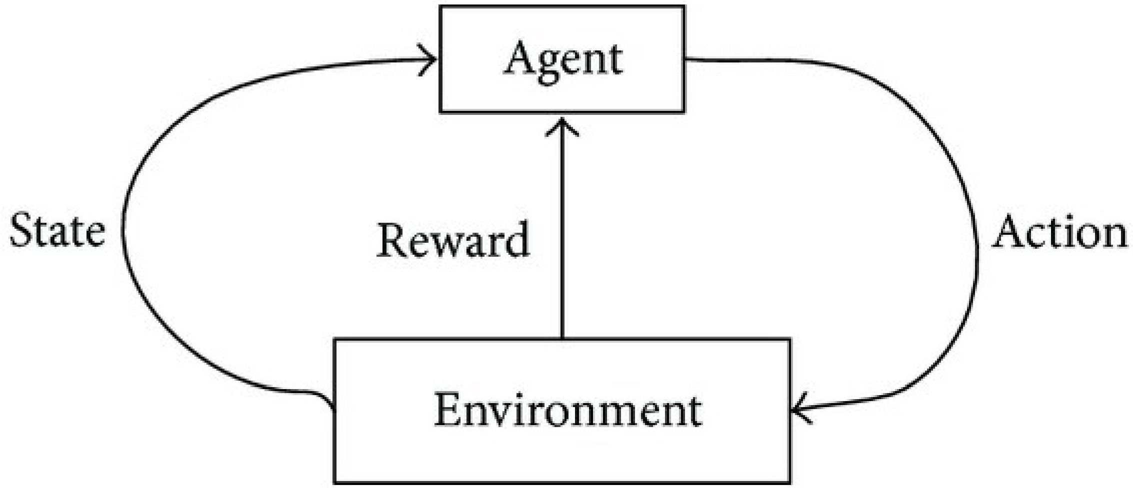
- If you only act by the “current optimal” policy, you may never hit the global optimum.

- 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

The MDP formalism



Classic MDP(Markov Decision Process)
Agent interacts with environment

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

Optimal policy formalism

Objective:
Total reward

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

γ ~ patience
Cake tomorrow is γ as good as now

Reinforcement learning:
- Find policy that maximizes the expected reward

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

Optimal policy formalism

Objective:
Total reward

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \ldots + \gamma^n \cdot r_{t+n}$$

$$R_t = \sum_i \gamma^i \cdot r_{t+i} \qquad \gamma \in (0,1)\, const$$

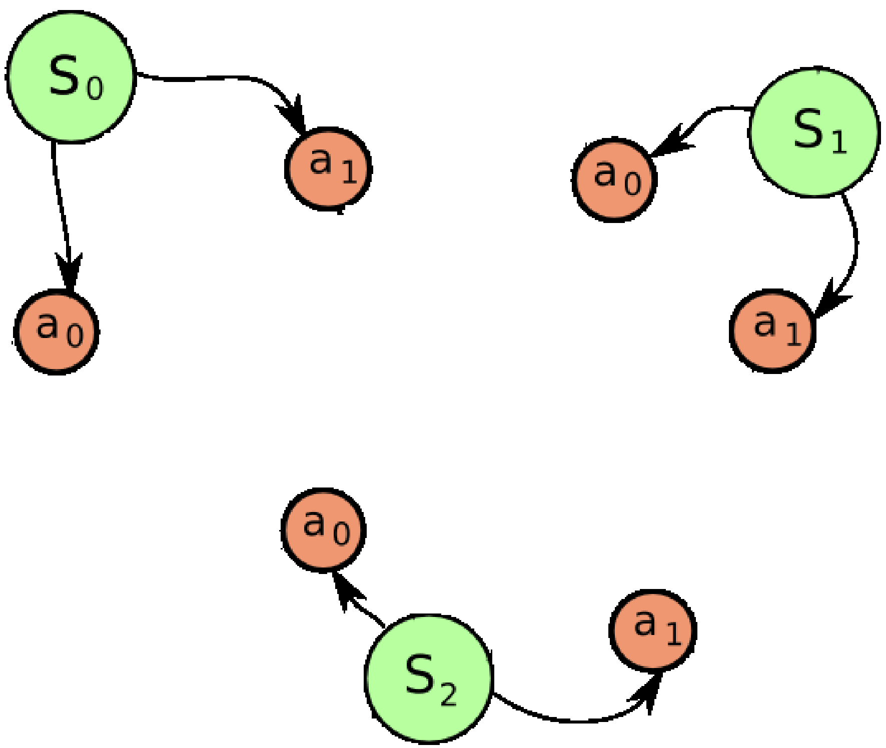
Trivia: which γ implies that "only the present time matters"?

Reinforcement learning:

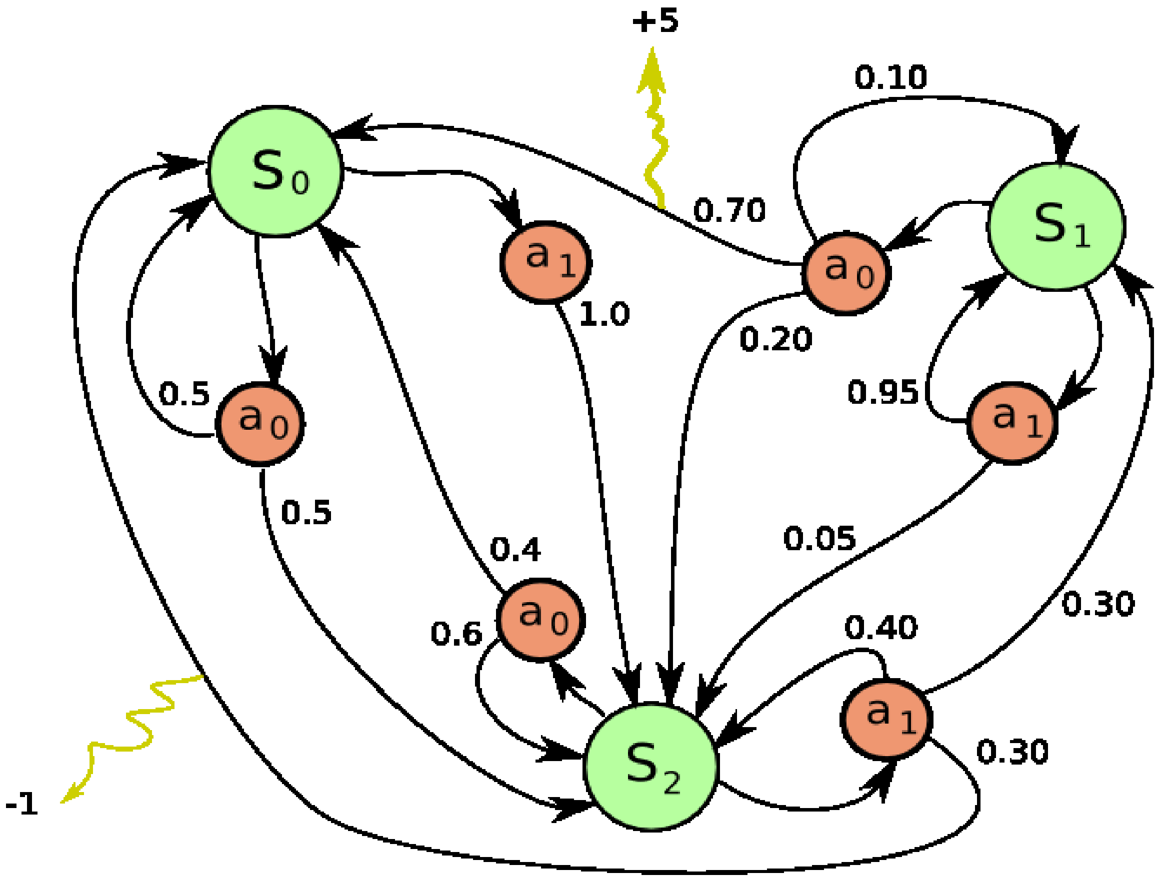
- Find policy that maximizes the expected reward

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

Simple example



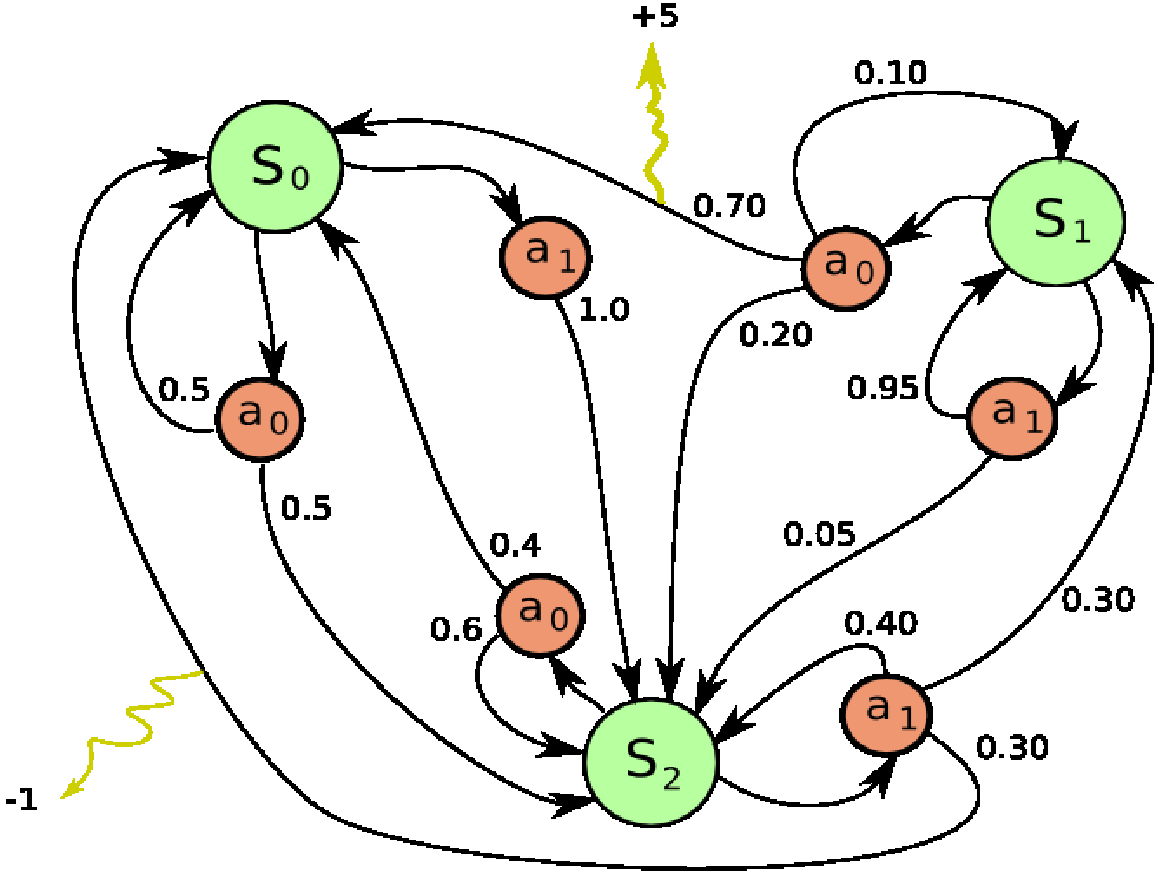
Simple example



Trivia:
What are the optimal actions for each state?

- **γ = 0**

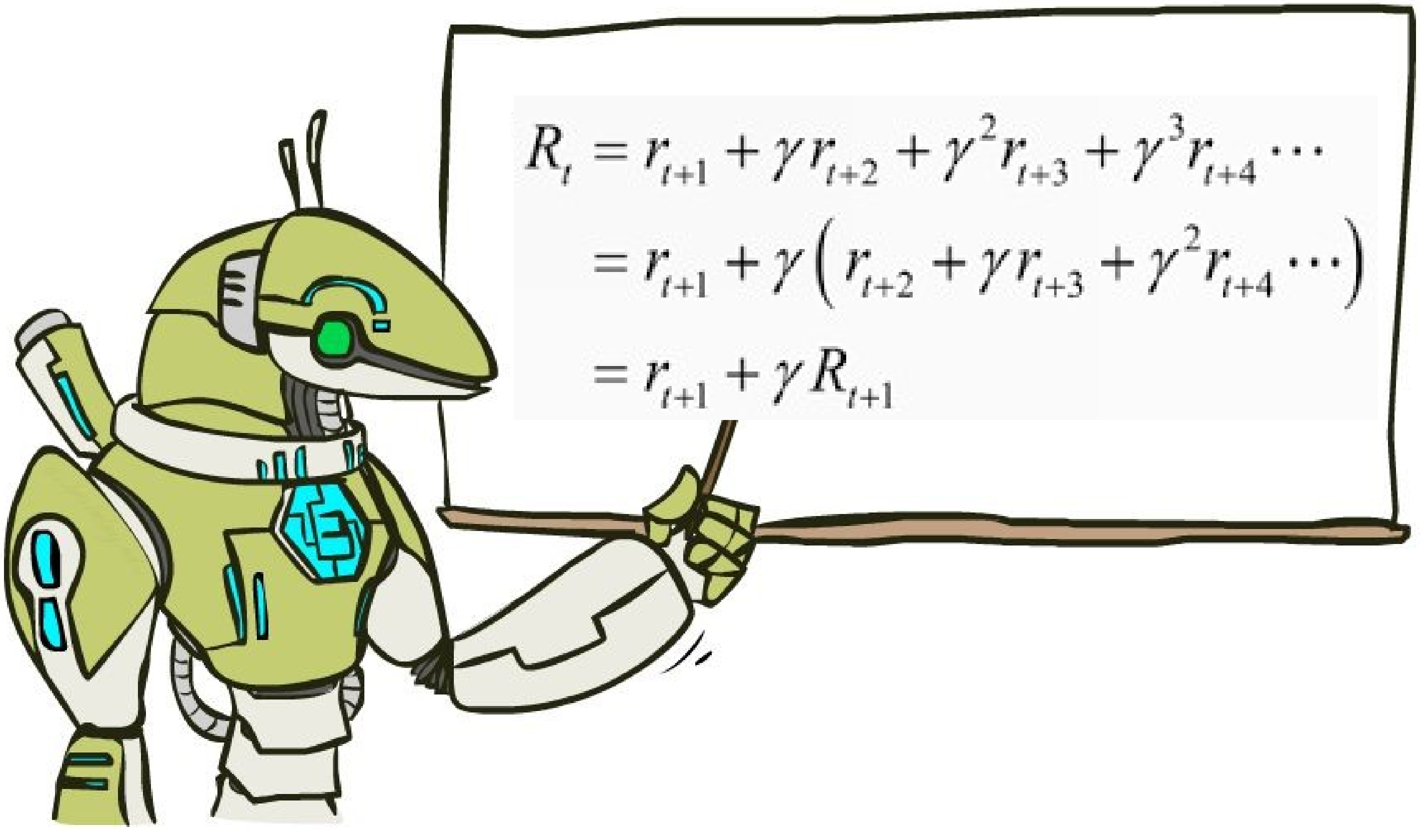
Simple example



Trivia:
What are the optimal actions for each state?

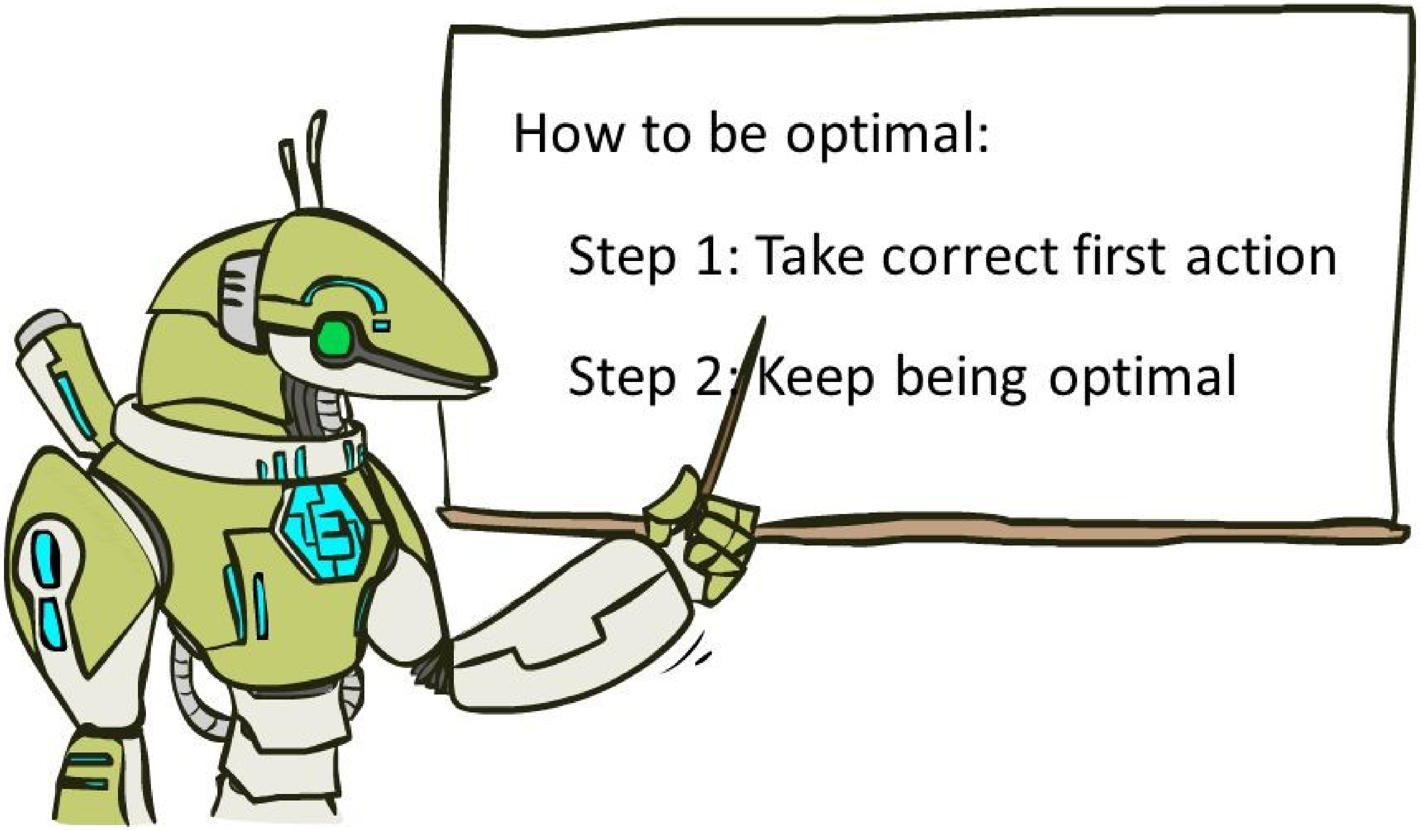
- ~~γ = 0~~
- **γ = 0.99**

Optimal policy



We rewrite R with sheer power of math!!

Optimal policy



Recurrent optimal strategy definition

Value iteration (Temporal Difference)

Idea:

- For each state, obtain V(state)

$$V(s)=max_a[r(s,a)+\gamma\cdot V(s'(s,a))]$$

Value iteration (Temporal Difference)

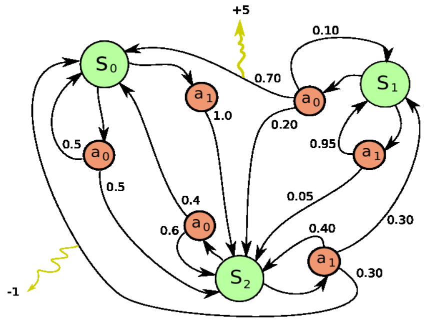
Idea:

- For each state, obtain V(state)

$$V(s) = max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V(s')]$$

Stochastic
action outcome

Trivia: if we know the exact V(s) for all states,
how do we determine the best actions?

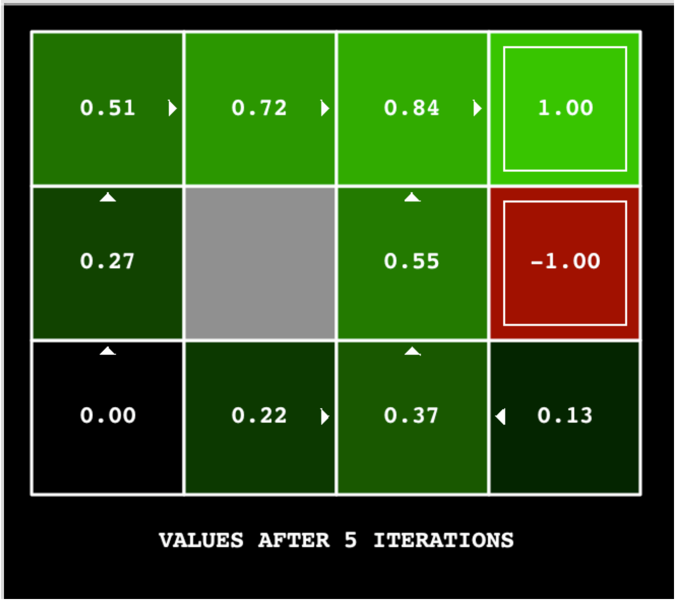


Value iteration (TD)

Idea:
- Iterative updates

$$\forall s, V_0(s) := 0$$

$$V_{i+1}(s) := max_a [r(s,a) + \gamma \cdot E_{s' \sim P(s'|s,a)} V_i(s')]$$



Voila! We've solved the reinforcement learning!

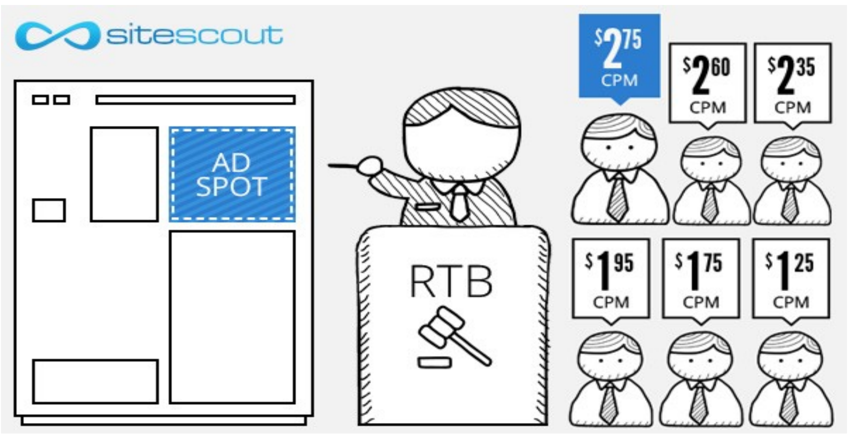
Voila! We've solved the reinforcement learning!

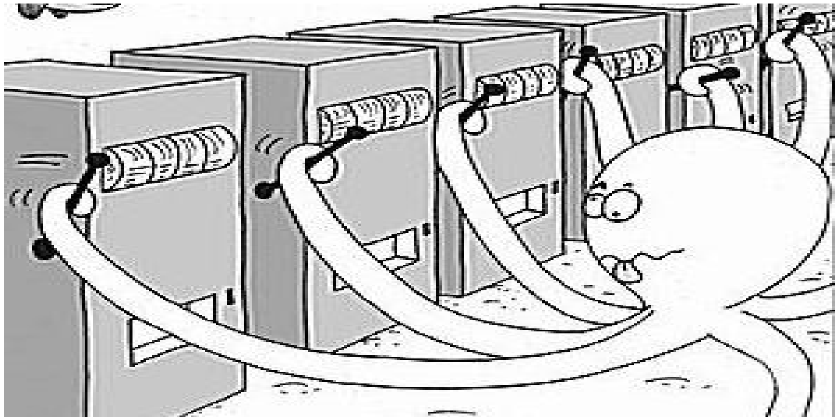
Or have we?

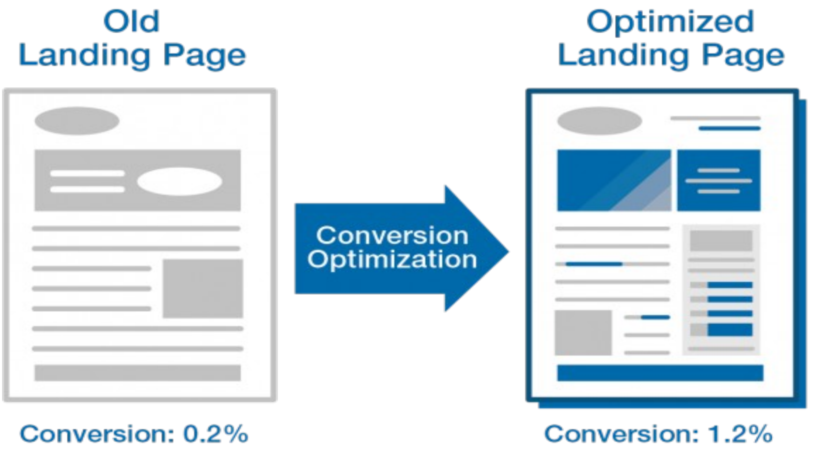
What happens if we apply it to real world problems?

Reality check: web

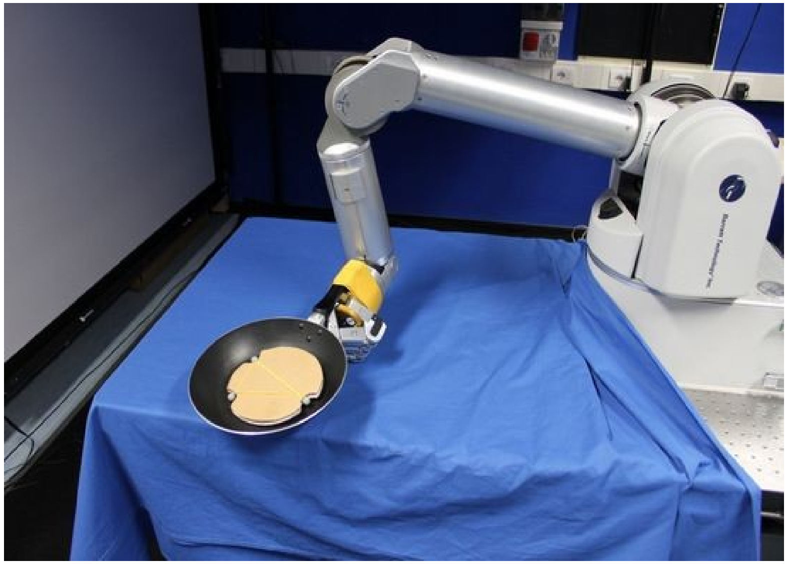
- **Cases:**
  - Pick ads to maximize profit
  - Design landing page to maximize user retention
  - Recommend items to users
- **Common traits:**
  - Independent states
  - Large action space

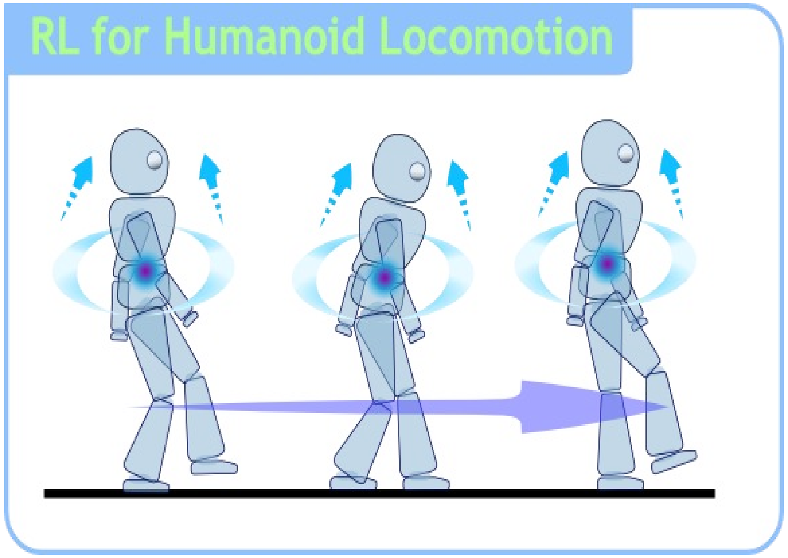






Reality check: dynamic systems



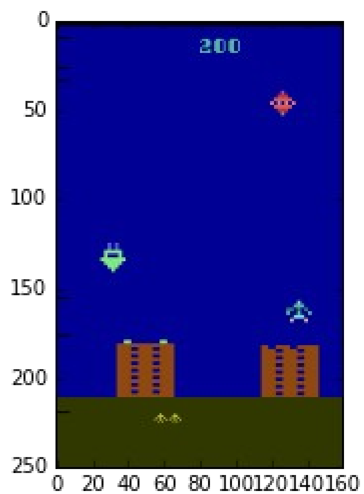


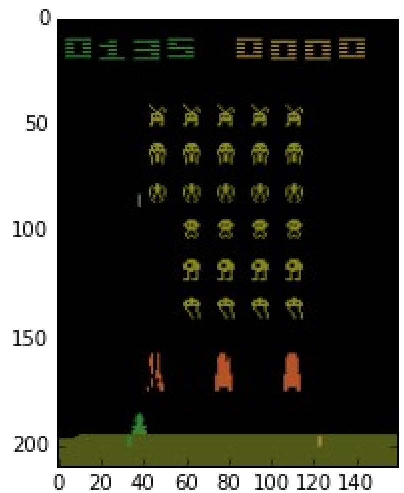
Reality check: MOAR

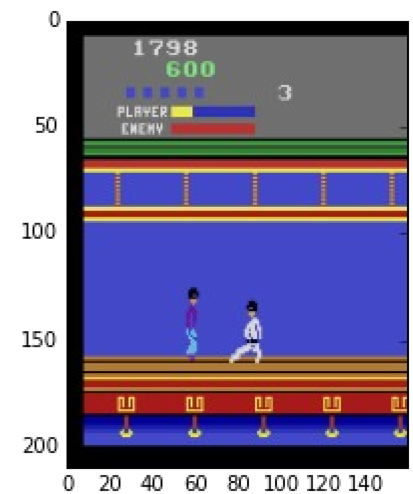
- **Cases:**
  - Robots
  - Self-driving vehicles
  - Pilot assistant
  - More robots!

- **Common traits:**
  - Continuous state space
  - Continuous action space
  - Partially-observable environment
  - LONG sessions

~~Reality~~ check: videogames





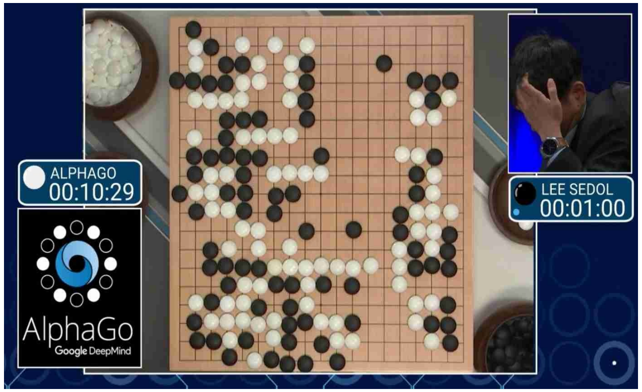


- **Trivia:** What are the states and actions?

Other use cases

- Personalized medical treatment

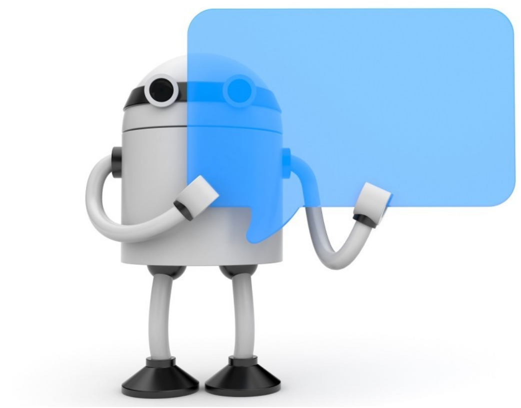
- Even more games (Go, chess, etc)



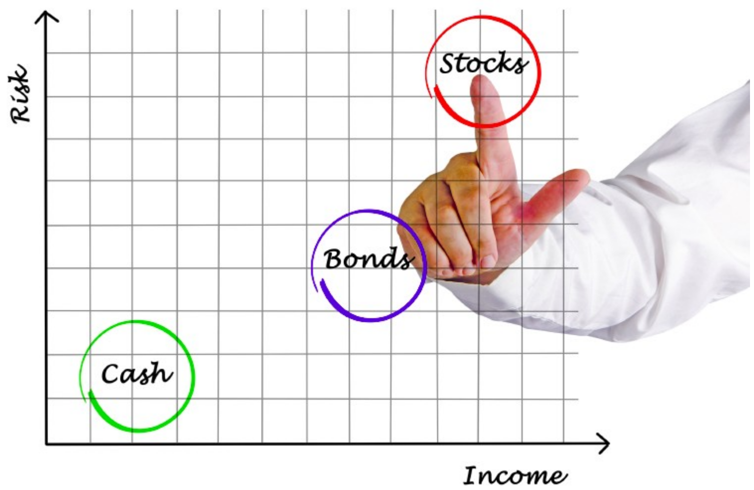
- **Trivia:** What are the states and actions?

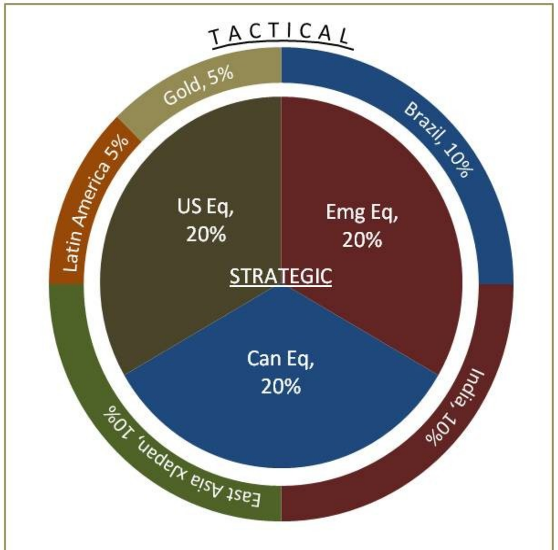
Other use cases

- Conversation systems (additional goals)

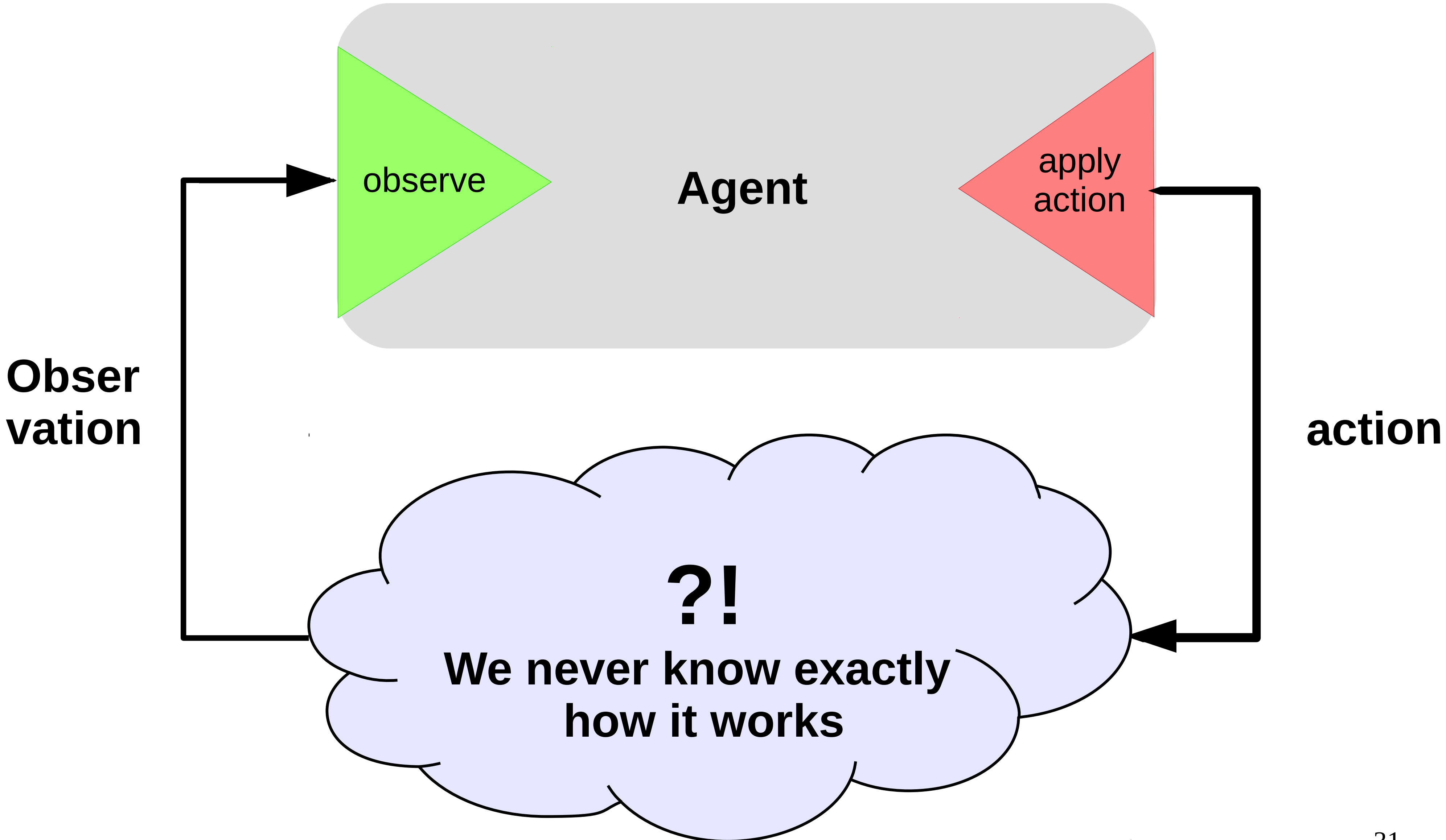


- Portfolio management (aka asset allocation)



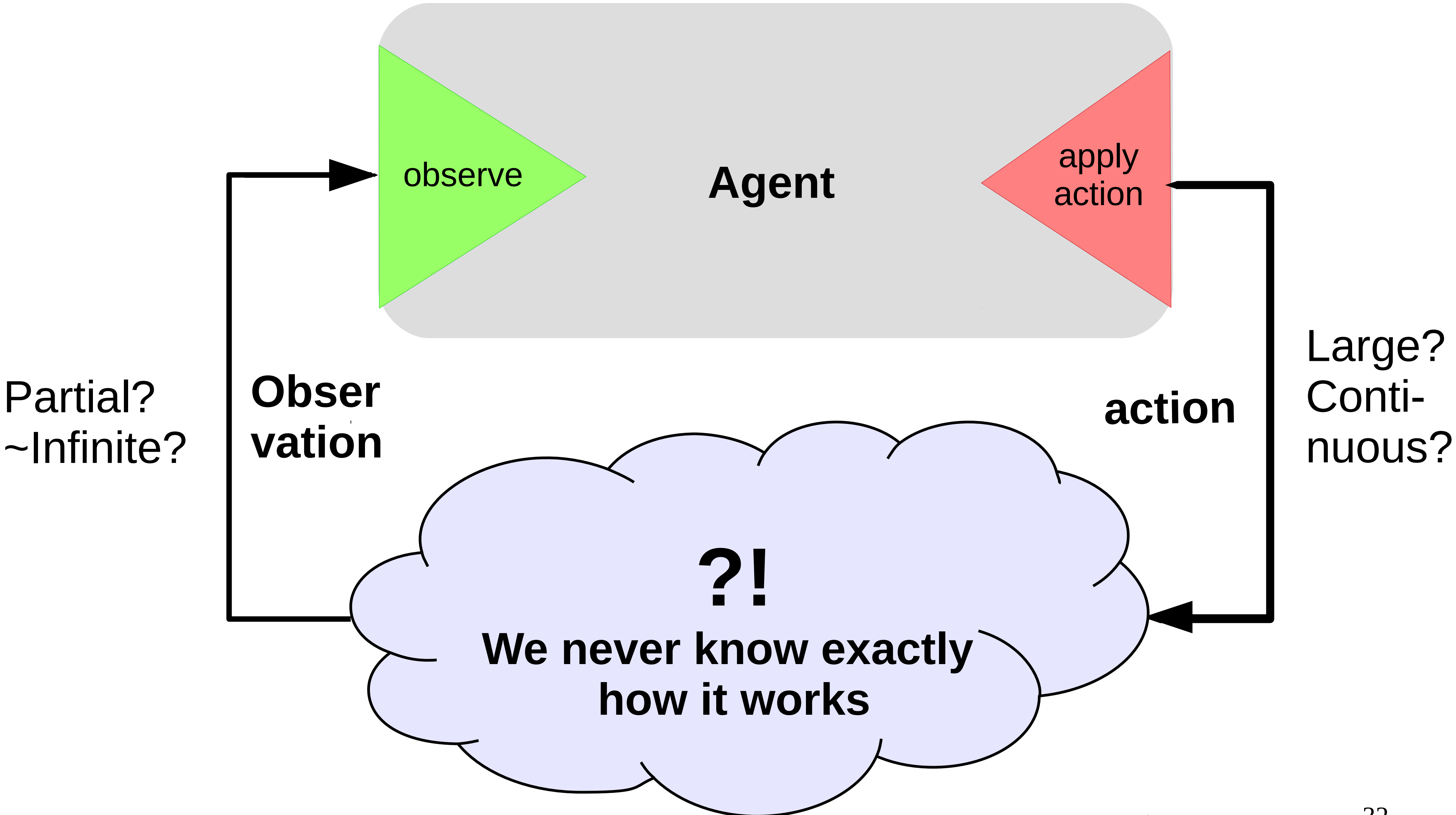


Real world



observe
Agent
apply
action
Obser
vation
action
?!
We never know exactly
how it works

Real world



observe
Agent
apply
action
Partial?
~Infinite?
Obser
vation
action
Large?
Conti-
nuous?
?!
We never know exactly
how it works

Problem:

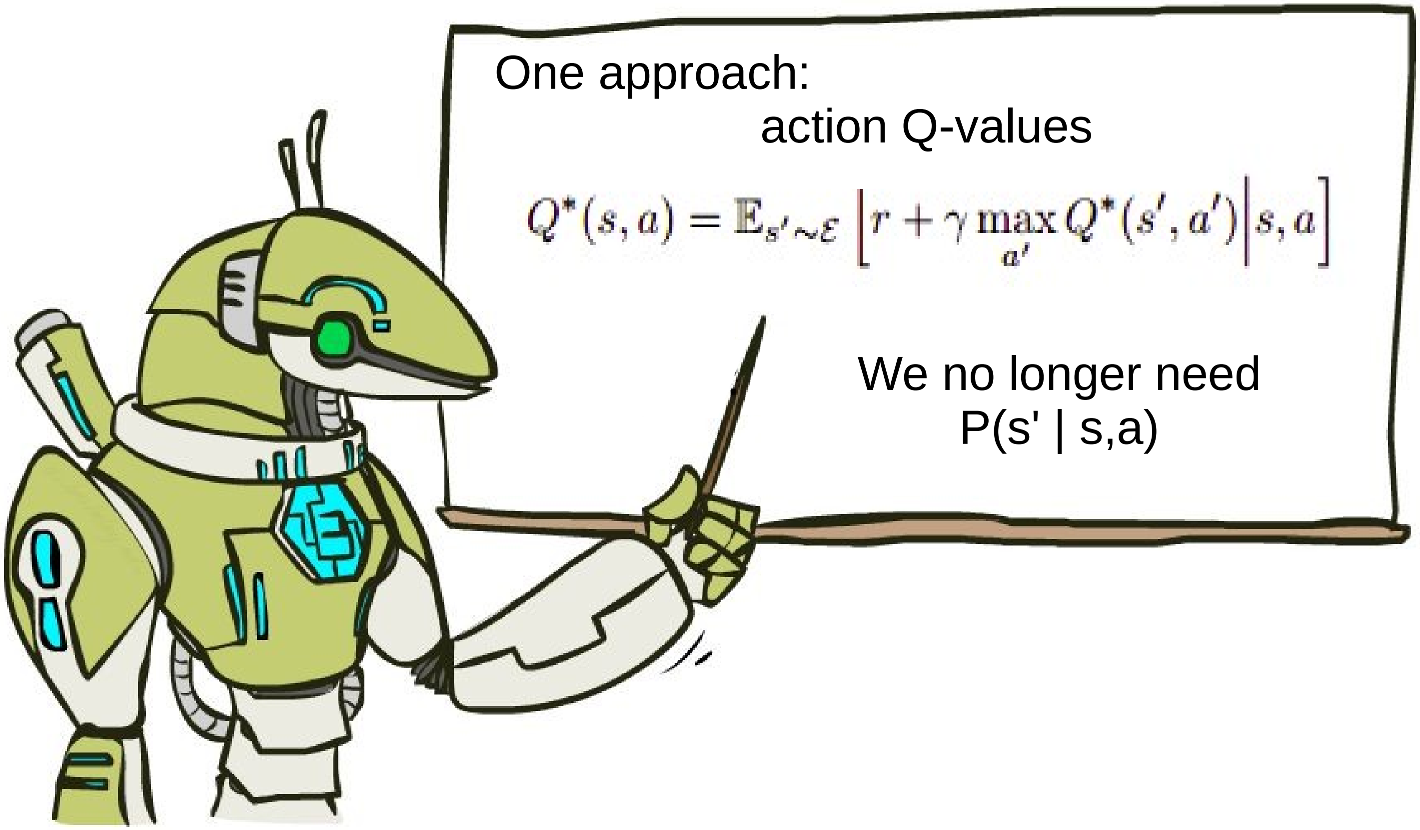
We never know actual

$P(s'|s,a)$

Learn it?

Get rid of it?

From V to Q

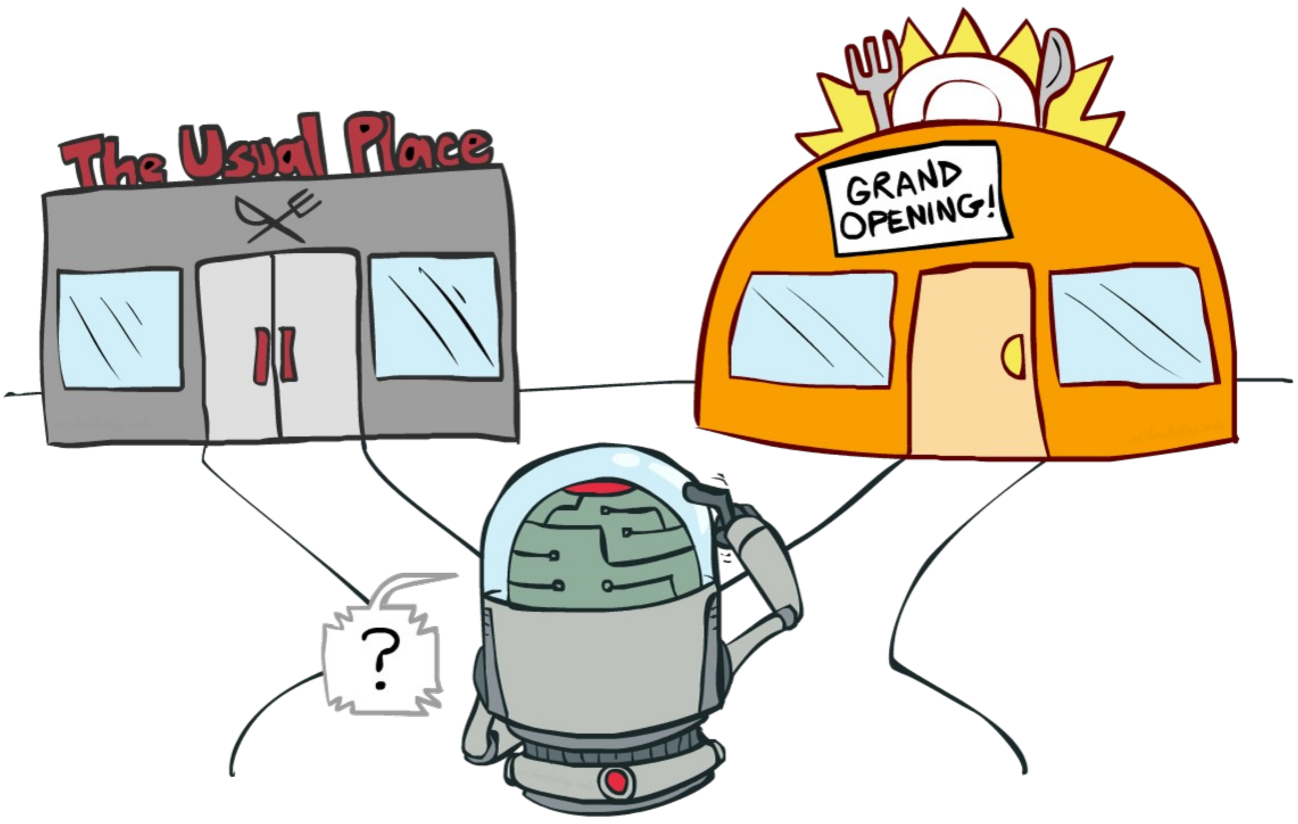


$$argmin_Q\left(Q(s_t,a_t) - [r_t + \gamma \cdot max_{a'} Q(s_{t+1},a')]\right)^2$$

$$\pi(s): argmax_a Q(s,a)$$

Exploration Vs Exploitation

Balance between using what you learned and trying to find something even better



Exploration Vs Exploitation

Strategies:

- ε-greedy
  - With probability ε take a uniformly random action; otherwise take optimal action.
- Softmax

  Pick action proportional to softmax of shifted normalized Q-values.

$$P(a) = softmax\left(\frac{Q(a) - Qmean}{Qvariance}\right)$$

- Some methods have a built-in exploration strategy (e.g. A2c)

Problem:

State space is usually large,

sometimes continuous.

And so is action space;

However, states do have a structure, similar states have similar action outcomes.

From tables to approximations

- Before:
  - For all states, for all actions, remember Q(s,a)

- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

$$argmin_{w,b}(Q(s_t,a_t)-[r_t+\gamma \cdot max_{a'} Q(s_{t+1},a')])^2$$

Trivia: should we use linear regression or logistic regression?

From tables to approximations

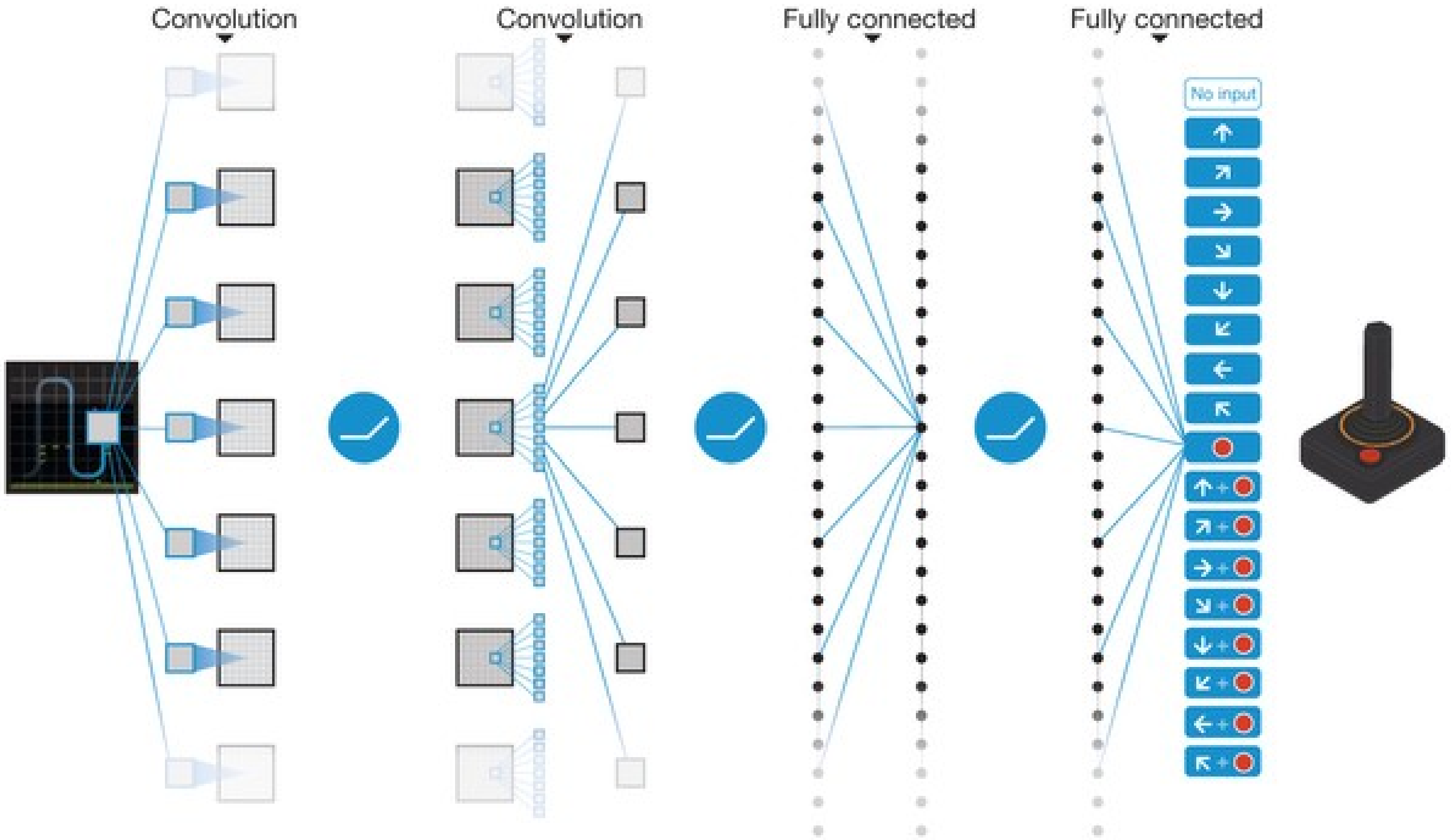
- Before:
  - For all states, for all actions, remember Q(s,a)

- Now:
  - Approximate Q(s,a) with some function
  - e.g. linear model over state features

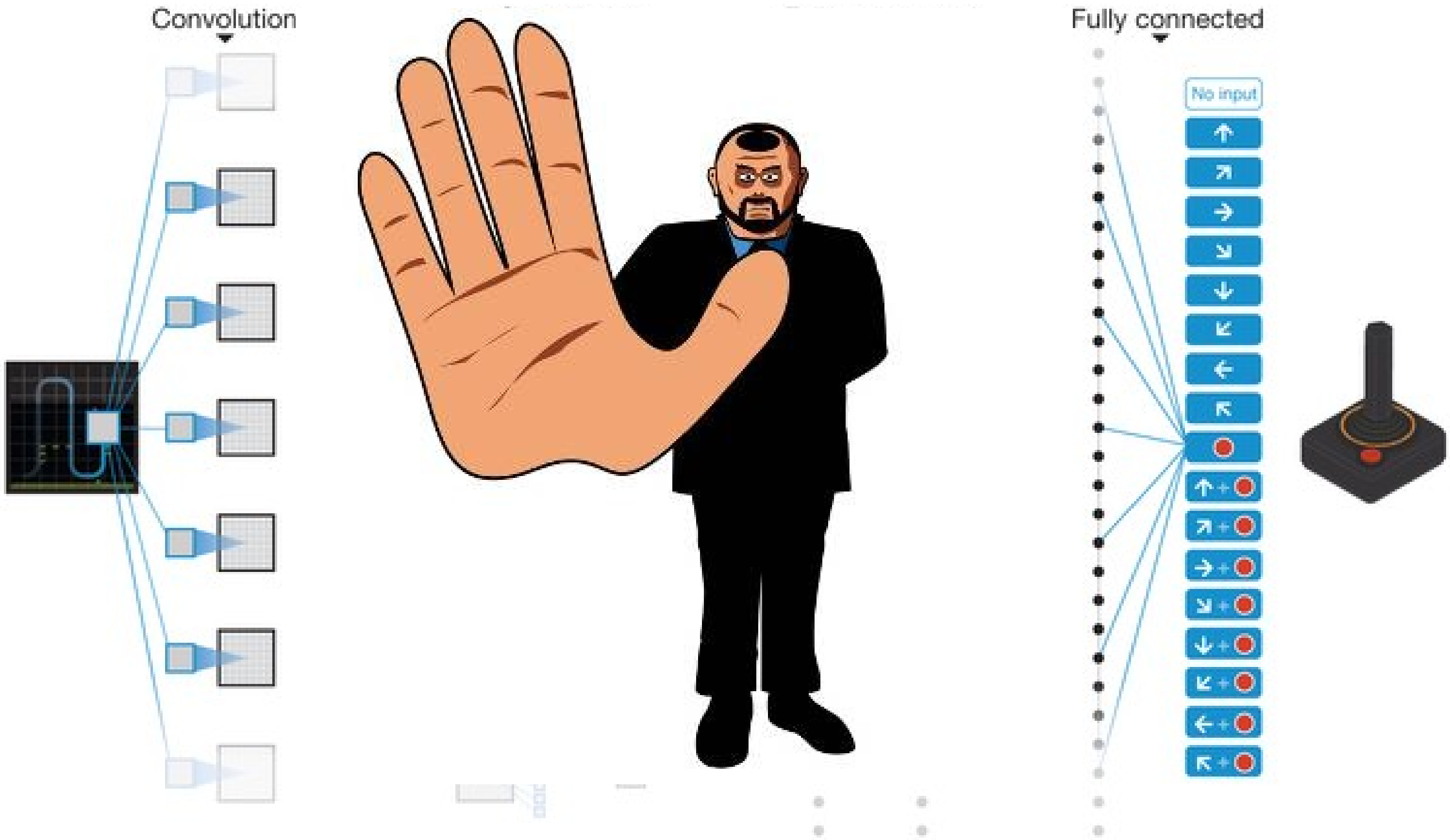
$$argmin_{w,b}\left(Q(s_t,a_t)-\left[r_t+\gamma\cdot max_{a'}Q(s_{t+1},a')\right]\right)^2$$

$$Q(s,a)=\sum_i W_{a,i}\cdot f_i(s)+b_a$$

Smells like a neural network



Not so fast...



Let's write some code!