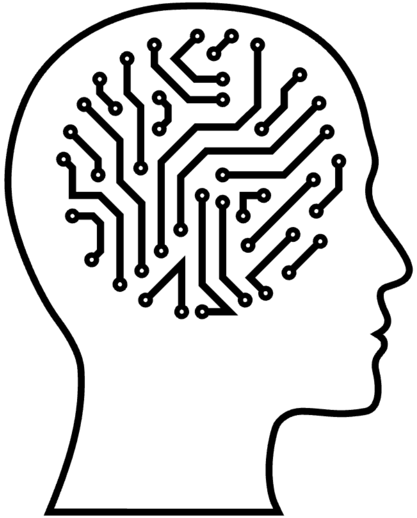


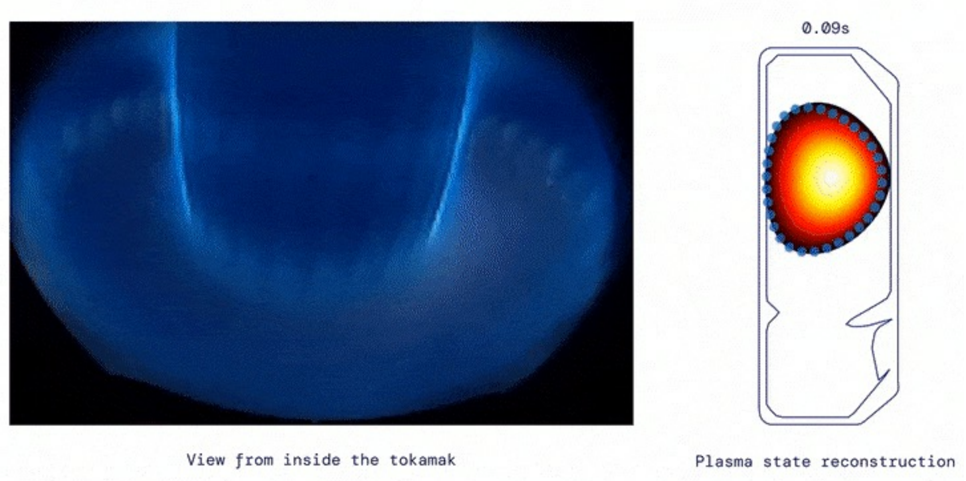
Crash course on Reinforcement Learning

Let's discuss together

05/02/2024 (RL4AA'23 Salzburg)

Support slides: Andrea Santamaría García, Chenran Xu

Control the plasma in a tokamak fusion reactor



ChatGPT: Optimizing Language Models for Dialogue

Methods

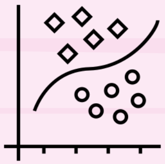
We trained this model using Reinforcement Learning from Human Feedback (RLHF), using the same methods as InstructGPT, but with slight differences in the data collection setup. We trained an initial model using supervised fine-tuning: human AI trainers provided conversations in which they played both sides—the user and an AI assistant. We gave the trainers access to model-written suggestions to help them compose their responses. We mixed this new dialogue dataset with the InstructGPT dataset, which we transformed into a dialogue format.

To create a reward model for reinforcement learning, we needed to collect comparison data, which consisted of two or more model responses ranked by quality. To collect this data, we took conversations that AI trainers had with the chatbot. We randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them. Using these reward models, we can fine-tune the model using Proximal Policy Optimization. We performed several iterations of this process.

SUPERVISED LEARNING

Classification, prediction, forecasting

computer learns by example



- Spam detection
- Weather forecasting
- Housing prices prediction
- Stock market prediction

UNSUPERVISED LEARNING

Segmentation of data

computer learns without prior information about the data

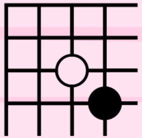
- Medical diagnosis
- Fraud (anomaly) detection
- Market segmentation
- Pattern recognition

MACHINE LEARNING

REINFORCEMENT LEARNING

Real-time decisions

computer learns through trial and error



- Self-driving cars
- Make financial trades
- Gaming (AlphaGo)
- Robotics manipulation

Machine Learning

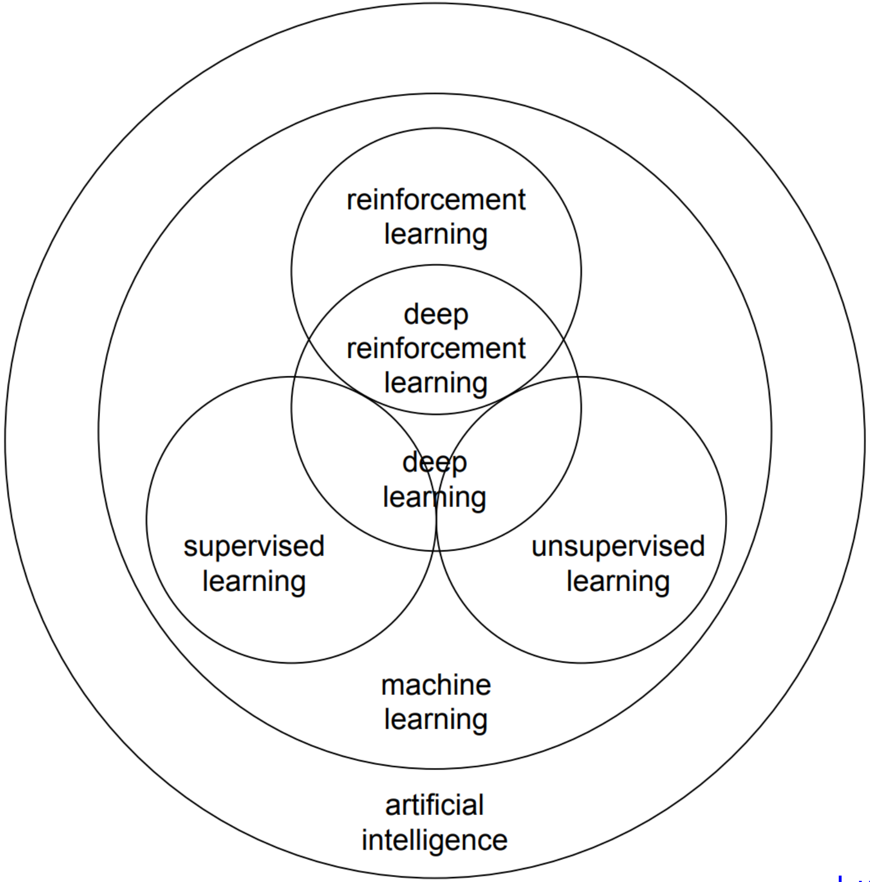
Learning style	Task	Popular algorithms
Supervised Learning We know the input & output (labeled data)	**Classification** discrete variables **Regression** continuous variables	▪ Neural networks (e.g. stochastic gradient descent, backpropagation) ▪ Support Vector Machine ▪ K-nearest neighbor ▪ Decision Tree algorithms (e.g. Classification and Regression Tree) ▪ Random Forest (ensemble) ▪ Uni or multivariate, linear or logistic
Unsupervised Learning We only know the input (unlabeled data)	**Clustering**	▪ K-means ▪ K-medians ▪ Expectation Maximization (EM) ▪ Hierarchical clustering
	Association	▪ Apriori algorithm ▪ Eclat algorithm
Reinforcement Learning	**Control** **Prediction**	▪ Value based (Q-learning) ▪ Policy based ▪ Actor critic ▪ Policy gradient or actor-critic ▪ Model-free or model based

Deep Learning Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short-Term Memory Networks
- Autoencoders
- Deep Boltzmann Machine
- Deep Belief Networks

Bayesian Algorithms
- Naive Bayes
- Gaussian Naive Bayes
- Bayesian Network
- Bayesian Belief Network
- Bayesian optimization

Regularization, dimensionality reduction, ensemble, evolutionary algorithms, computer vision, recommender systems, …

this slide is not exhaustive



https://arxiv.org/pdf/1810.06339.pdf

Reinforcement learning
more than machine learning

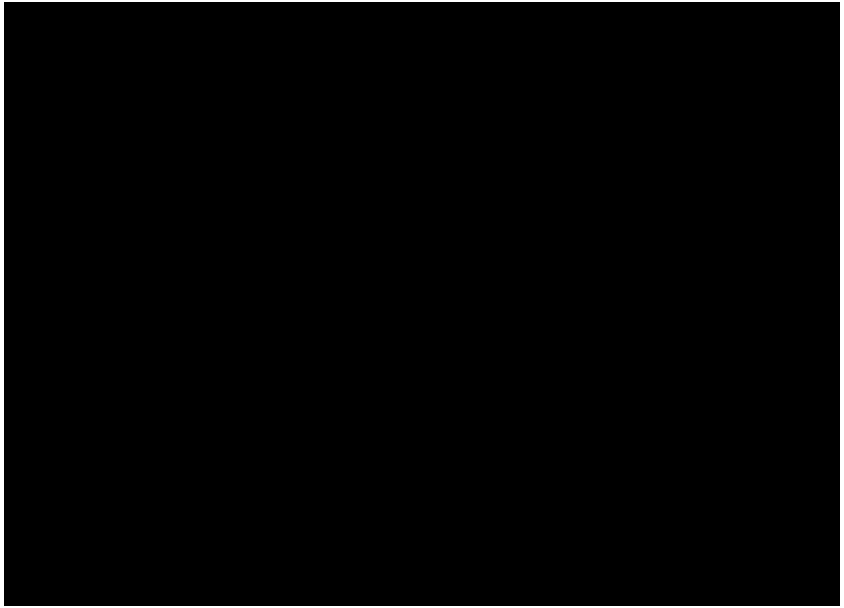
BEHAVIOR LEARNING

Psychology (classical conditioning)
Neuroscience (reward system)
Economics (game theory)
Mathematics (operations research)
Engineering (optimal control, planning)

Reinforcement learning

understanding how the human brain learns makes decisions

https://arxiv.org/abs/1707.02286



https://www.deepmind.com/publications/playing-atari-with-deep-reinforcement-learning

The RL problem

Reward hypothesis

all goals can be described by the maximization of expected cumulative sum of a received scalar signal

"Reward is enough"

Reward

scalar feedback signal $\mathcal{R}_t$ that indicates how well the agent is doing at step t

Goal

maximization of cumulative reward through selected actions

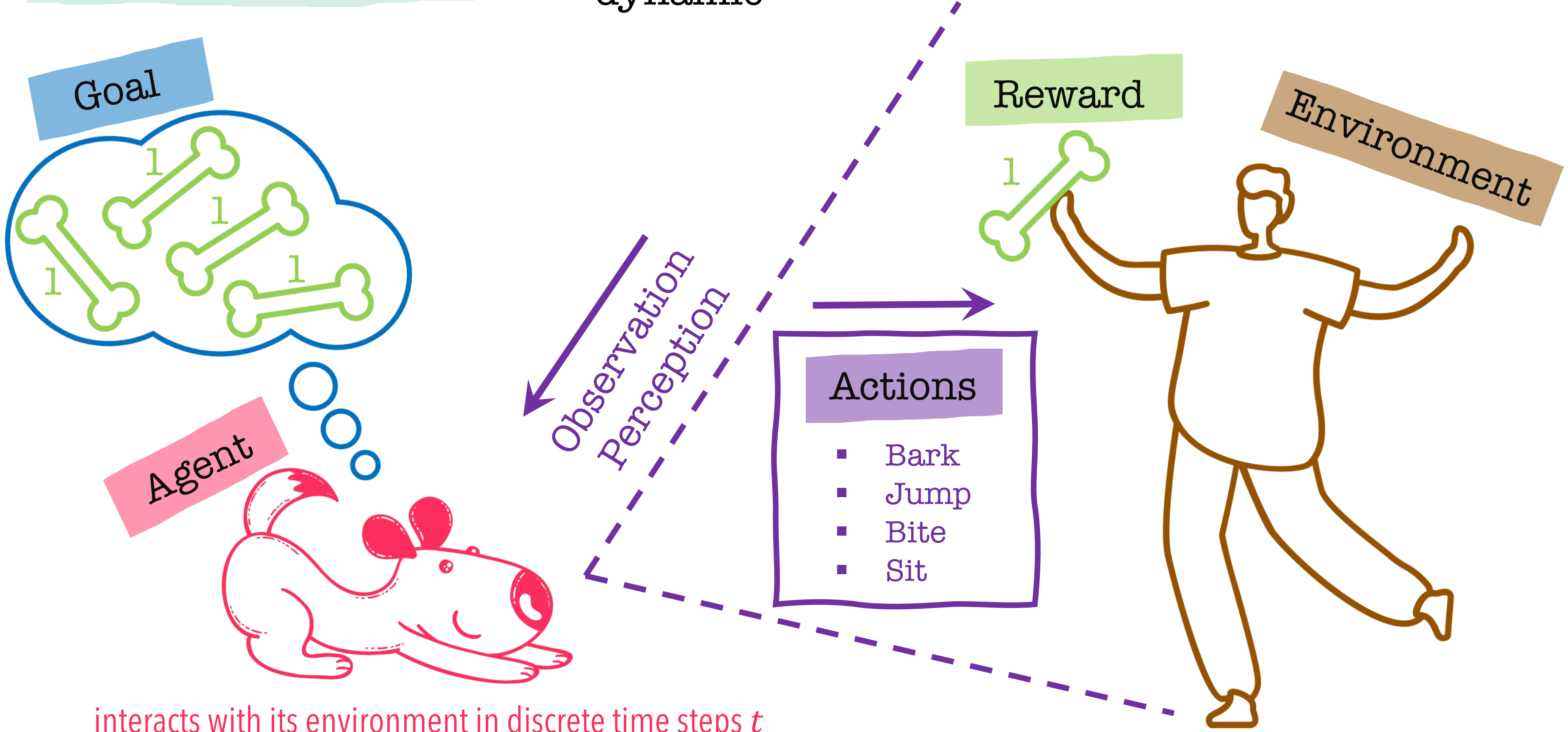
Agent

executes action
→ receives observation
→ receives scalar reward

an agent must learn through trial-and-error interactions with a dynamic environment

The RL problem

interactive
dynamic



interacts with its environment in discrete time steps t

Icons from the noun project

How to cumulate reward?

Agent

Model

agent's representation of the environment

Which behaviors perform well in this environment?

Policy agent's behaviour function (how the agent picks its actions)

Estimate the utility of taking actions in particular states of the environment (evaluation of the policy)

Value function how good each state and/or action are

- **Prediction**: evaluate the future given a policy
- **Control**: optimize the future (find the best policy)

Challenges in RL

Trade-off between exploitation and exploration

- Actions may have long-term consequences
- Reward might be delayed (does not happen immediately)

↳ should the agent sacrifice immediate reward to gain more long term reward?

The agent needs to:

✓ **Exploit** what it has already experienced in order to obtain reward now

✓ **Explore** the environment to select better actions in the future by sacrificing known reward now

…and both cannot be pursued exclusively without failing at the task

The agent

Must:

- Be able to **sense the state** of its environment to some extent
- Be able to **take actions** that affect that state
- **Have a goal** or goals relating to the state of the environment

Sensation

"Free-will"

Motivation

Markov Decision Processes

Include this 3 elements without trivializing any of them

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^{a}, \mathcal{R}_{s}^{a}, \gamma)$ $\mathcal{S}$ = finite set of states

State information used to determine what happens next

A state transition can be:

- **Deterministic** $s_{t+1} = f(\mathcal{H}_t)$
- **Stochastic** $s_{t+1} \sim \mathbb{P}(s_{t+1} | \tau_t)$

Trajectory sequence of states and actions until time t

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \dots)$$

Environment state ($\mathcal{S}^e$): environment's internal representation, usually not visible to the agent

Agent state ($\mathcal{S}^a$): agent's internal representation, used by the RL algorithm to pick the next action

Observation ($\mathcal{O}$):partial description of a state, which may omit information

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^{a}, \mathcal{R}_{s}^{a}, \gamma)$ $\mathcal{S}$ = finite set of states

State information used to determine what happens next

A state transition can be:

- **Deterministic** $s_{t+1} = f(\mathcal{H}_t)$
- **Stochastic** $s_{t+1} \sim \mathbb{P}(s_{t+1} | \tau_t)$

Trajectory sequence of states and actions until time t

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \dots)$$

Markov state / property

A state is Markov if and only if:

$$\mathbb{P}[s_{t+1} | s_t] = \mathbb{P}[s_{t+1} | s_{1,\dots,t}]$$

- The state is a sufficient statistic of the future
- The future is independent of the past, given the present
- Once the state is known, the history may be discarded

state transitions of an MDP satisfy the Markov property

Fully observable environments

$\mathcal{O}_t = \mathcal{S}_t^a = \mathcal{S}_t^e$

- Agent directly observes environment state
- Necessary condition to formalize an RL problem with an MDP

Partially observable environments

$\mathcal{S}_t^a \neq \mathcal{S}_t^e$

Agent constructs its own state representation:

- Complete trajectory: $\mathcal{S}_t^a = \tau_t$
- Beliefs of environment state: $\mathcal{S}_t^a = (\mathbb{P}[\mathcal{S}_t^e = s_1], \dots, \mathbb{P}[\mathcal{S}_t^e = s_n])$
- Recurrent neural networks: $\mathcal{S}_t^a = \sigma(w_0\mathcal{O}_t + w_s\mathcal{S}_{t-1}^a)$

→ Partially observable MDP

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s, \gamma)$

State transition model / probability

Predicts the next state (dynamics of the environment)

$$\mathcal{P}^a_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A} = a]$$

Probability of ending in state s' after taking action a while being in state s

$$\mathcal{P} = \begin{pmatrix} \boxed{\mathcal{P}_{11} & \cdots & \mathcal{P}_{1n}} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{pmatrix} \Sigma=1$$

If probabilities change overtime

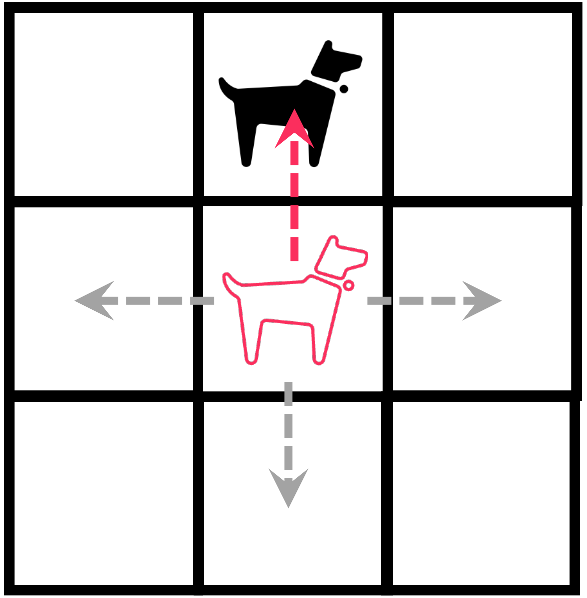
= **non-stationary Markov process**

Transition probabilities from all states and successor states

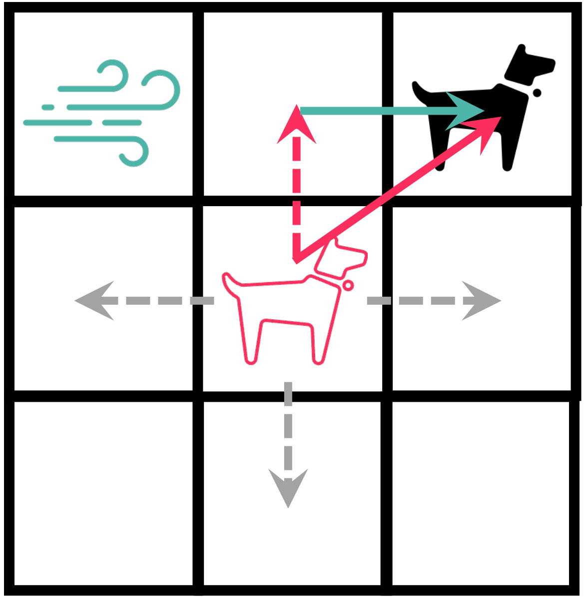
Non-deterministic environment

Taking the same action in the same state on two different occasions may result in different next states

$t = t_0$



$t = t_0 + \tau$



Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s, \gamma)$

Return Total discounted reward from time step t

$$\mathcal{G}_t = \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \cdots = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}$$

"infinite-horizon discounted return"

The goal is to maximize the return

- The discount factor $\gamma \in [0, 1)$ avoids infinite returns (sum converges)
- It values immediate reward over delayed reward (human-like)
- It deals with uncertainty about the future (no perfect model of env.)

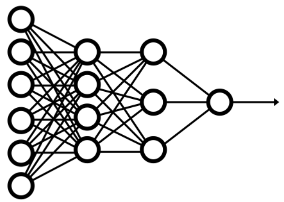
Side notes:

- There are also undiscounted Markov processes if all sequences terminate (episodic)
- Model-based: there is an expectation of a reward (but not in model-free)

Policy

Map from state to action

- Policy π completely defines how the agent will behave
- It's a distribution over actions given a certain state



Deterministic: $a = \pi(s)$

Stochastic: $\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a | \mathcal{S}_t = s]$

Probability of taking a specific action by being in a specific state

Categorical (discrete action spaces)
Gaussian (continuous action spaces)

Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π:

$$\mathcal{P}^{\pi}_{s,s'} = \sum_{a \epsilon \mathcal{A}} \pi(a|s)\, \mathcal{P}^{a}_{s,s'} \qquad \mathcal{R}^{\pi}_{s} = \sum_{a \epsilon \mathcal{A}} \pi(a|s)\, \mathcal{R}^{a}_{s}$$

Value function

Estimation of expected future reward

A way to compare policies

- Used to choose between states depending on how much reward we expect to get
- Depends on the agent's behavior (policy)

State-value function

Expected return starting from state s and following policy π (evaluates the policy)

$$\mathcal{V}_{\pi}(s) = \mathbb{E}_{\pi}[\mathcal{G}_t \mid \mathcal{S}_t = s]$$

given policy

Action-value function

Expected return starting from state s , taking action a , and following policy π

$$\mathcal{Q}_{\pi}(s, a) = \mathbb{E}_{\pi}[\mathcal{G}_t \mid \mathcal{S}_t = s, \mathcal{A}_t = a]$$

"Q function"

Bellman optimality equation

The state-value function can be decomposed into:

- **immediate reward $\mathcal{R}_{t+1}$**
- **discounted value of next state $\gamma\, v(\mathcal{S}_{t+1})$**

$$\mathcal{V}(s) = \mathbb{E}[\mathcal{G}_t \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{R}_{t+2} + \gamma^2\, \mathcal{R}_{t+3} \ldots \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, (\mathcal{R}_{t+2} + \gamma\, \mathcal{R}_{t+3} \ldots) \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{G}_{t+1} \mid \mathcal{S}_t = s]$$

$\mathbb{E}(f) = \mathbb{E}(\mathbb{E}(f))$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{V}(\mathcal{S}_{t+1}) \mid \mathcal{S}_t = s]$$

Reward you expect to get from being in your current state

Expected value of wherever state you land next

$$\boxed{\mathcal{V}(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}\, \mathcal{V}(s')}$$

Bellman expectation equation

Considering the policy π we get:

$$\boxed{\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi\,(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)}$$

Direct solution only for small MDPs

- System of $\mathcal{S}$ simultaneous linear equations with $\mathcal{S}$ unknowns

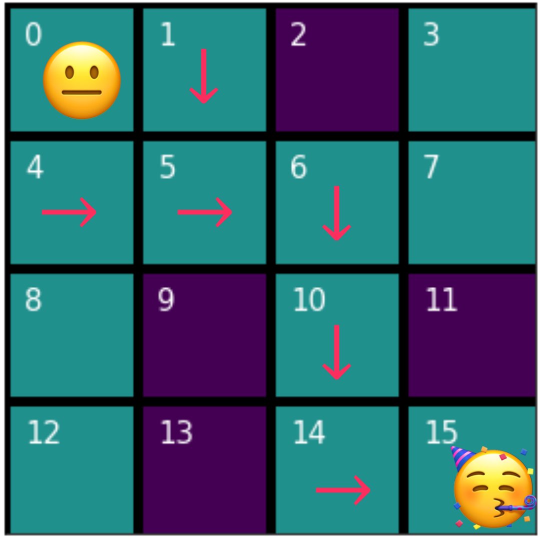
Other ways of solving it:

- Iteratively (dynamic programming)
- Sampling (Monte-Carlo evaluation)
- Approximation (temporal-difference learning)

Example: gridworld

The agent needs to get from state **0** to state **15** to get out of the maze

States

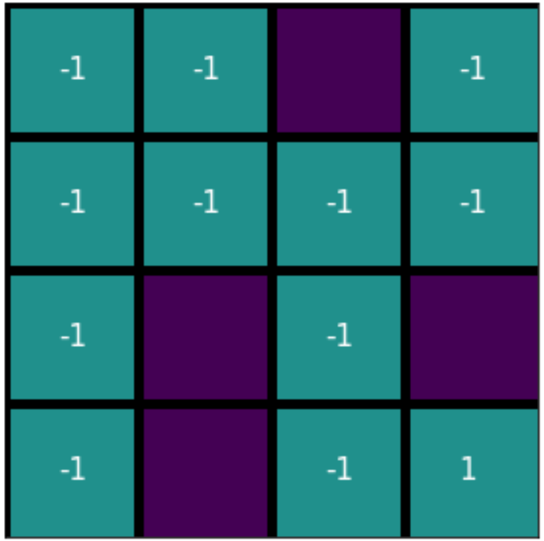


Actions $\mathcal{A} = (\uparrow, \downarrow, \leftarrow, \rightarrow)$

Deterministic env: $\mathcal{P}^a_{s,s'} = 1$

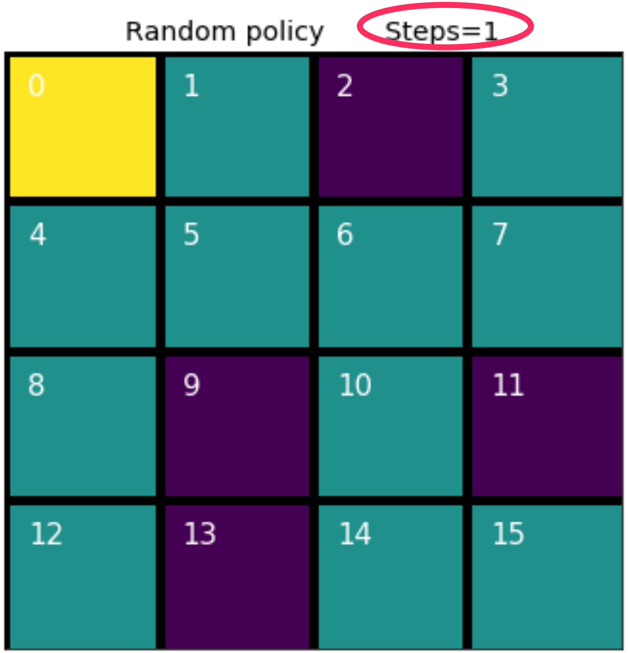
Rewards

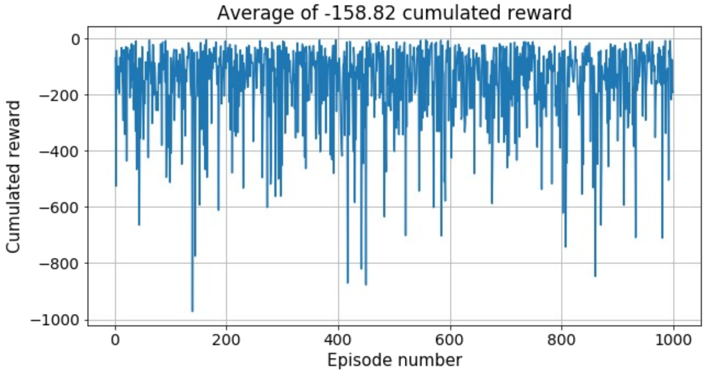
no discount γ



Example: gridworld

Policy $\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a | \mathcal{S}_t = s] \longrightarrow \pi(a|s) = \overbrace{\mathbb{P}[\uparrow,\downarrow,\leftarrow,\rightarrow \mid \mathcal{S}_t] = 0.25}^{\text{random policy}}$

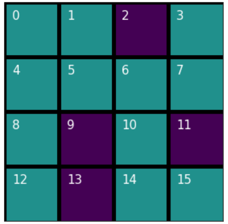




Example: gridworld

Value function

$$\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)$$

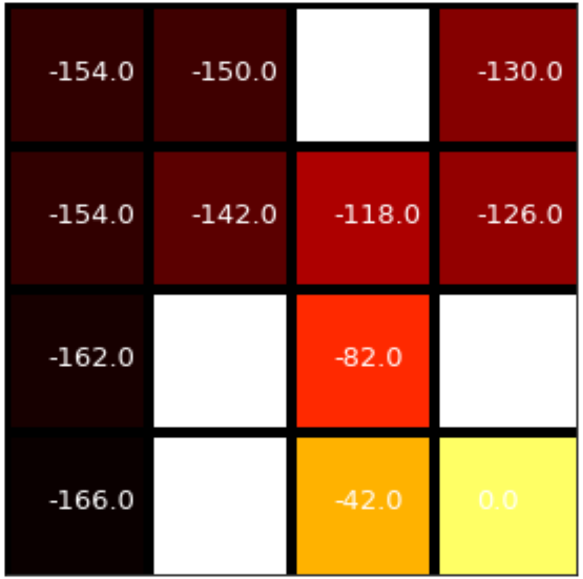


Solving simultaneously linear set of equations:

- environment's dynamics are completely known

```
0.5*v0 - 0.25*v1 - 0.25*v4 + 1.0 = 0
-0.25*v0 + 0.5*v1 - 0.25*v5 + 1.0 = 0
0.25*v3 - 0.25*v7 + 1.0 = 0
-0.25*v0 + 0.75*v4 - 0.25*v5 - 0.25*v8 + 1.0 = 0
-0.25*v1 - 0.25*v4 + 0.75*v5 - 0.25*v6 + 1.0 = 0
-0.25*v10 - 0.25*v5 + 0.75*v6 - 0.25*v7 + 1.0 = 0
-0.25*v3 - 0.25*v6 + 0.5*v7 + 1.0 = 0
-0.25*v12 - 0.25*v4 + 0.5*v8 + 1.0 = 0
0.5*v10 - 0.25*v14 - 0.25*v6 + 1.0 = 0
0.25*v12 - 0.25*v8 + 1.0 = 0
-0.25*v10 + 0.5*v14 + 0.5 = 0

11 variables,  11 equations
```



$\pi \rightarrow \mathcal{V}_\pi$ = policy evaluation

how much value this policy has?

Example: gridworld

Value function

$$\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)$$

Solving iteratively:

- Bellman equation becomes an update rule

$$\mathcal{V}_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}_k(s') \right)$$

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$

Loop:

  $\Delta \leftarrow 0$

  Loop for each $s \in \mathcal{S}$:

    $V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$

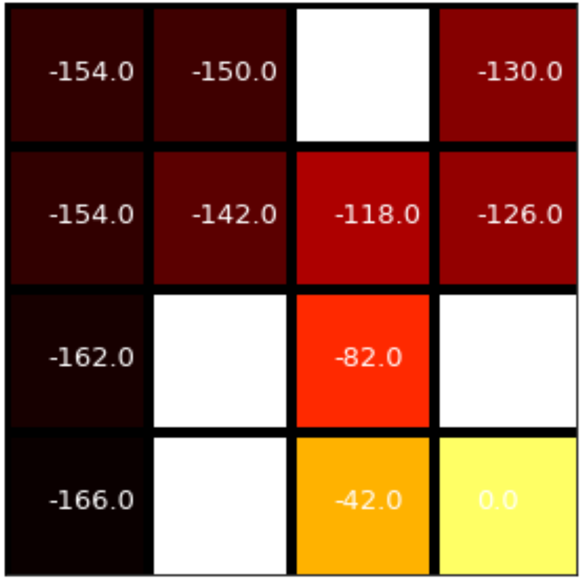
    $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$

  $V \leftarrow V'$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Coursera



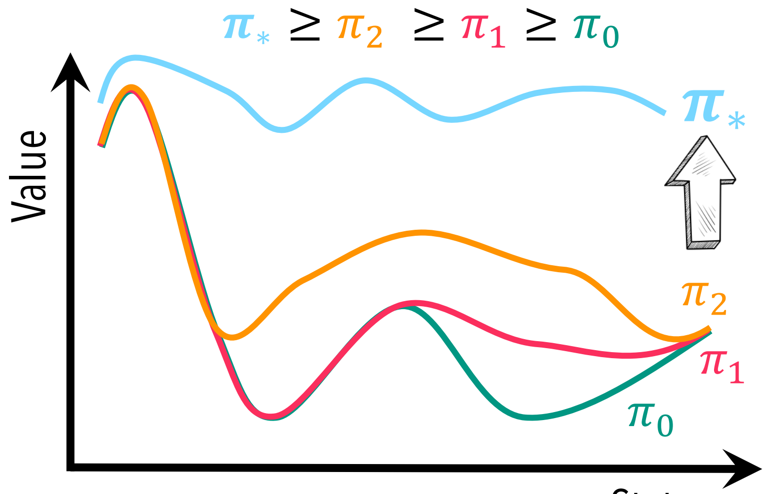
$\pi \rightarrow \mathcal{V}_\pi$ = **policy evaluation**

how much value this policy has?

Dynamic programming algorithms

turn the Bellman eq. into update rules

- ✅ **Prediction**: what's the value for a specific policy?
- **Control**: which policy gives as much reward as possible?
  → the policy with more value!



$\pi \geq \pi' \ if \ \mathcal{V}_\pi(s) \geq \mathcal{V}'_\pi, \forall \ s\epsilon\mathcal{S}$

For any MDP:

- There exists an <u>optimal policy $\boldsymbol{\pi}_*$</u> that is better or equal to all other policies $\pi_* \geq \pi \ \forall\pi$
- All optimal policies achieve the optimal value function $\mathcal{V}_{\pi_*} = \mathcal{V}_*(s)$ and $Q_{\pi_*} = Q_*(s,a)$

So…do I have to calculate the value of every policy and compare them?

$|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies in an MDP

$4^{11} \approx$ 4 million policies for simple gridworld example

Bellman optimality equations

$$\mathcal{V}_{\pi*}(s) = \mathbb{E}_{\pi*}[\mathcal{G}_t \mid \mathcal{S}_t = s] = \max_{\pi} \mathcal{V}_{\pi}(s) \qquad \forall\, s \epsilon \mathcal{S}$$

$$\mathcal{Q}_{\pi*}(s) = \max_{\pi} \mathcal{Q}_{\pi}(s) \qquad \forall\, s \epsilon \mathcal{S}, a \epsilon \mathcal{A}$$

Optimal value functions

By replacing the optimal policy on the Bellman equations we get:

π_* assigns probability 1 to the action that receives the highest value

$$\boldsymbol{\mathcal{V}_*(s)} = \max_{a} \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right)$$

maximum value over every next possible state

$$\boldsymbol{\mathcal{Q}_*(s,a)} = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \max_{a'} \mathcal{Q}_*(s', a')$$

- <u>Nonlinear (max)</u>, no closed-form solution
- Dynamic programming solutions only applicable <u>if the dynamics of the system $\mathcal{P}$ are known</u>

Determining an optimal policy

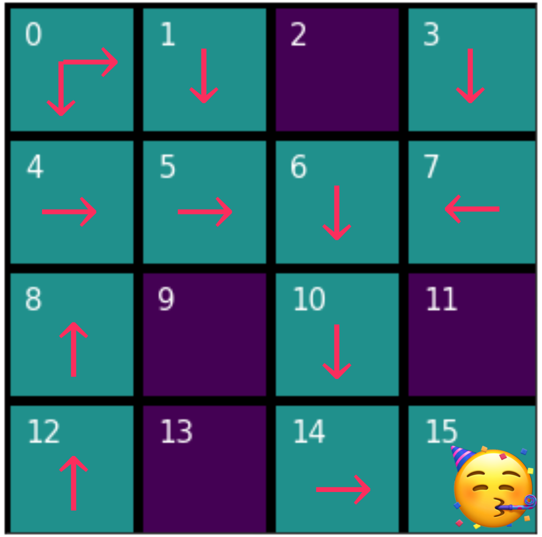
$$\boldsymbol{\mathcal{V}}_*(\boldsymbol{s}) = \max_a \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right)$$

maximum over all actions

For any state we look at each available action and take the one that maximizes the argument

$$\boldsymbol{\pi}_*(\boldsymbol{s}) = \underset{a}{\operatorname{argmax}} \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right)$$

particular action that achieves that maximum (greedy action)



$$\boldsymbol{\pi}_*(\boldsymbol{s}) = \underset{a}{\operatorname{argmax}}\, Q_*$$

Policy improvement & iteration

Let's consider a value function $\mathcal{V}_\pi$ that is non-optimal, and we select an action that is greedy with respect to it:

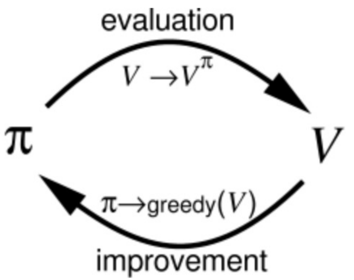
$$\boldsymbol{\pi}'(\boldsymbol{s}) = \underset{a}{\operatorname{argmax}} \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_\pi(s') \right)$$

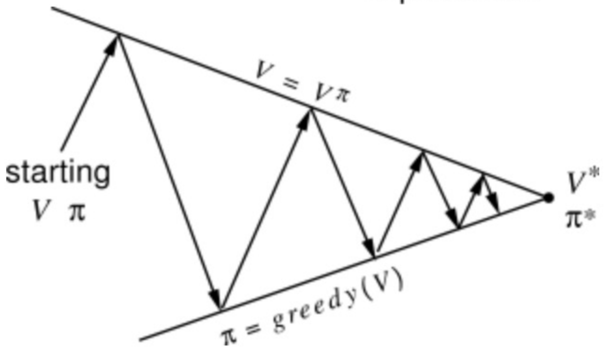
- If the action has a higher value, the policy is better
- $\mathcal{V}_*$ is the unique solution to the Bellman optimality eq.
- If this greedy operation does not change $\mathcal{V}$, then it converged to the optimal policy because it satisfies the Bellman optimality eq.

improvement

$$\pi_1 \to \mathcal{V}_{\pi 1} \to \pi_2 \to \cdots \to \pi_*$$

evaluation





Images from http://incompleteideas.net/book/ebook/node46.html

Dynamic programming algorithms

turn the Bellman eq. into update rules

Problem	Bellman equation	Algorithm	Sample-based version
Prediction	Expectation equation	Iterative policy evaluation	Temporal difference
Control	Expectation equation + greedy policy	Policy iteration	Sarsa
Control	Optimality equation	Value iteration	Q-learning

when we don't know $\mathcal{P}$

Off-policy learning

On-policy: improve and evaluate the policy being used to select actions

Off-policy: improve and evaluate a different policy from the one used to select actions

- ➢ Learn a target policy π (optimal policy) while…
- ➢ …selecting actions from behavior policy b (exploratory policy)

Provides another strategy for continuous exploration (experiences a larger # of states)

Temporal difference learning

Learning method specialized for multi-step **prediction learning**

- TD learning is learning a prediction from another, later learned prediction
  - ➢ learning a guess from a guess (you don't know the true $\mathcal{V}$)

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \alpha[\mathcal{R} + \gamma\mathcal{V}(s') - \mathcal{V}(s)]$$

- Difference between both predictions = temporal difference
- No $\mathcal{P}$ model needed (unlike in dynamic programming)

- Allows you to estimate the value function before the episode is finished
- Making long-term predictions is exponentially complex
  - ➢ Memory scales with the #steps of the prediction
- TD model = standard model of reward systems in the brain

Q-learning

Off-policy TD control

$$\mathcal{Q}(s,a) \leftarrow \mathcal{Q}(s,a) + \alpha[\mathcal{R} + \gamma \max \mathcal{Q}(s',a) - \mathcal{Q}(s,a)]$$

Converges to the optimal value function as long as the agent continues to explore sampling the state-action space

Overview of RL methods

Tabular solution methods

- Iterative (dynamic programming)
- Sample-based (Monte-Carlo evaluation)
- Temporal-difference learning

- Used to solve finite MDPs
- Value functions are stored as arrays (tables)
- Methods can often find exact solutions

In real-life situations, we cannot store the values of each possible state in an array, especially in continuous problems

- Autonomous driving: array per possible image the camera sees?

Approximate solution methods

- Value-based
- Policy-based
- Policy gradient
- Actor-critic

- Approximate value by function parametrized by a weight vector
  --> **neural networks (learning!)**
- Applicable to partially observable problems

Approximate solution methods

Value-based
contains a value function, policy is implicit

- Sample efficient
- Computationally fast
- Unstable (bias, don't know true $\mathcal{V}$)

DQN, NAF

Policy-based
does not store the value function, only the policy

Policy gradient
optimizes parametrized policies with gradient descent

- Convergence guarantees
- Sensitive to stepsize choice
- Poor sample efficiency
- Large variance

PPO, TD3, DDPG

Actor-critic
stores both the policy and value function

ACER, A2C/A3C, SAC

Policy Gradient

Parameterize the policy with a parameter set θ $\quad \pi = \pi_\theta(a|s)$

E.g. using a Neural Network

Optimize the cumulative reward $\quad \max_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t\right]$

Return G_t

Generic gradient ascent step: $\quad \theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$

How to calculate the gradient?

Policy Gradient Theorem

$$\nabla_\theta J(\pi_\theta) \propto \mathbb{E}_{\pi_\theta}\left[\sum_t Q_\pi(s,a) \nabla \pi_\theta(a|s)\right]$$

Reinforce

- Initialize the policy parameter θ at random
- Loop forever:
  - Generate an trajectory (episode) using π_θ : $S_0, A_0, R_1, S_1, A_1, \ldots,$
  - Loop for each step in episode $t = 0, 1, \ldots, T - 1$:
    - Estimate the return G_t
    - Perform one gradient update: $\theta \leftarrow \theta + \alpha\, \gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$

Monte-Carlo Sampling

Gradient update after sampling the whole episode

The MC-estimate has a high **variance**, we can subtract an **unbiased** baseline from G_t

Common choice is the state-value function:

Advantage function: $A(s,a) = Q(s,a) - V(s)$

Use **actor** and **critic** to approximate the functions

Trust-region Policy Optimization (TRPO)

Idea: constrain the policy updates within a small trust-region, using KL-divergence as a *difference measure* between the new and old policies

Importance sampling

Advantage function (estimated)

$$\text{maximize } J^{\text{TRPO}}(\theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_{\text{old}}(s,a)\right],$$

$$\text{subject to } \mathrm{KL}(\pi_{\theta_{old}} || \pi_\theta) \leq \delta$$

Note: Even though TRPO is an on-policy algorithm, the policy being optimized π_θ is not always the behaviour policy $\pi_{\theta_{\text{old}}}$

TRPO guarantees a **monotonic improvement** over policy iteration!

https://jonathan-hui.medium.com/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

Proximal Policy Optimization (PPO)

TRPO has nice sample efficiency and improvement guarantees, but is **complicated to optimize**

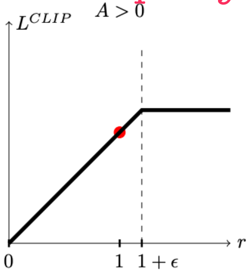
PPO uses a **clipped reward** so that the ratio between the policies stays within a small interval

$$J^{\mathrm{PPO}}(\theta) = \mathbb{E}\left[\min\left(\underbrace{r(\theta)A_{\theta_{\mathrm{old}}}(s,a)}_{\text{TRPO reward}}, \underbrace{\mathrm{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A_{\theta_{\mathrm{old}}}(s,a)}_{\text{Clipped reward}}\right)\right]$$

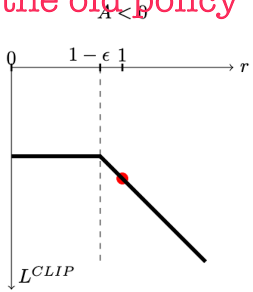
e.g. in [0.8, 1.2]

with $r(\theta) = \dfrac{\pi_\theta(a|s)}{\pi_{\theta_{\mathrm{old}}}(a|s)}$

A>0: Better than the old policy



A<0: Worse than the old policy



	Description	Policy	Action space	State space	Operator
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor Critic	Off-policy	Continuous	Continuous	Advantage

Thank you for your attention!

What questions do you have for me?

- Sutton & Barto book
- https://arxiv.org/pdf/cs/9605103.pdf
- Reinforcement learning lectures by David Silver
- https://spinningup.openai.com/en/latest/
- Coursera RL specialization
- https://arxiv.org/pdf/1810.06339.pdf

Let's connect! andrea.santamaria@kit.edu / @ansantam

 All icons from this talk from TheNounProject