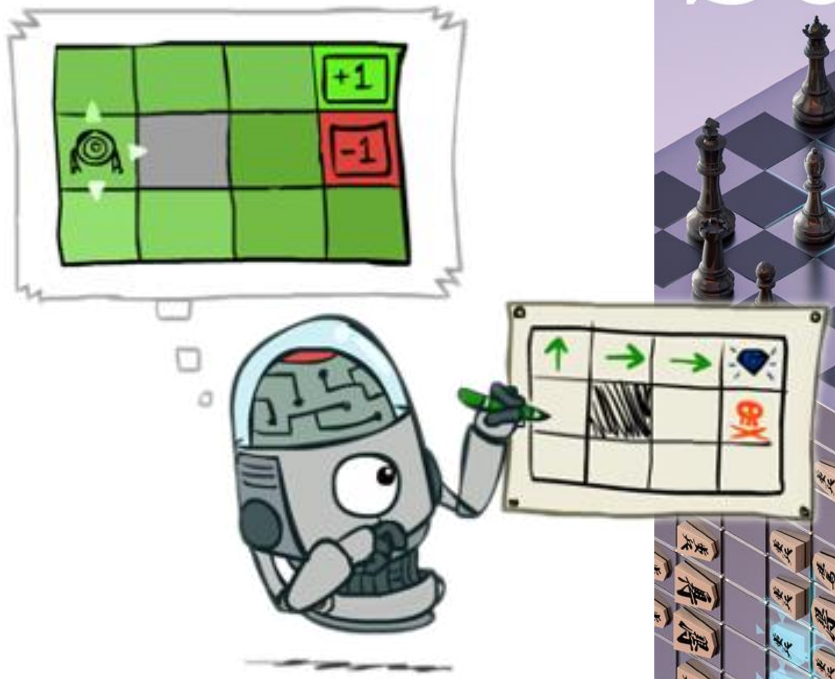


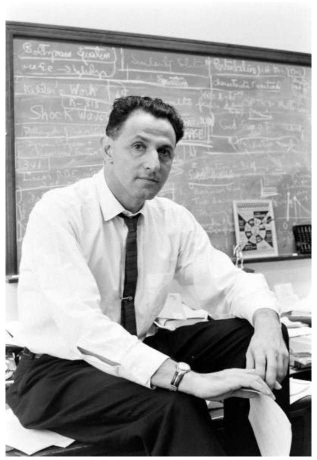
Image: Designed by Klein and Abbeel (CS188, UC Berkeley)

How research funders profit from hidden investments p. 1100
New books for budding scientists p. 1104
Drug leads for malaria pp. 1112 & 1129
Science
$15 7 DECEMBER 2018 sciencemag.org
AAAS
A DIGITAL PRODIGY
AlphaZero teaches itself chess, shogi, and Go
pp. 1087, 1118, & 1140

Image: Cover, Science, 7th December 2018

„An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.“

– Richard Bellman, 1957



AI101

Lecture 13: Introduction into Reinforcement Learning and AlphaZero

Thanks to Dan Klein and Peter Abbeel for making their slides available!

Slides reviewed by: Maria Novitskaja

Slides designed and lecture held by *Johannes Nathanael Czech*, Version 2.0, **Date:** 29.01.2024

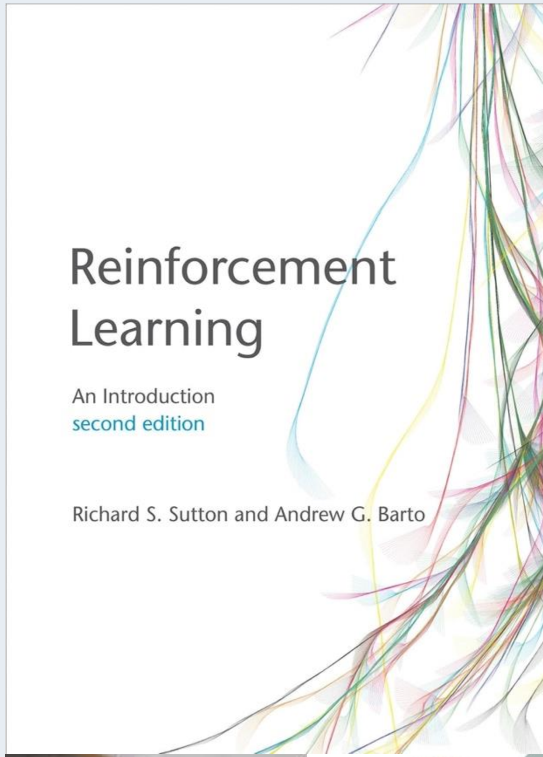
Recap

Introduction into Machine Learning and Neural Networks

- What types of learning exist?
- What is machine learning compared to artificial intelligence?
- What is overfitting and how to deal with it?
- How to evaluate your model?
- How does a (multi-layer) perceptron work?
- How to train a neural network?
- How does backpropagation work?

Today: Reinforcement Learning (RL) and AlphaZero

"RL is a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives when interacting with a complex, uncertain environment." -- Richard Sutton, Andy Barto



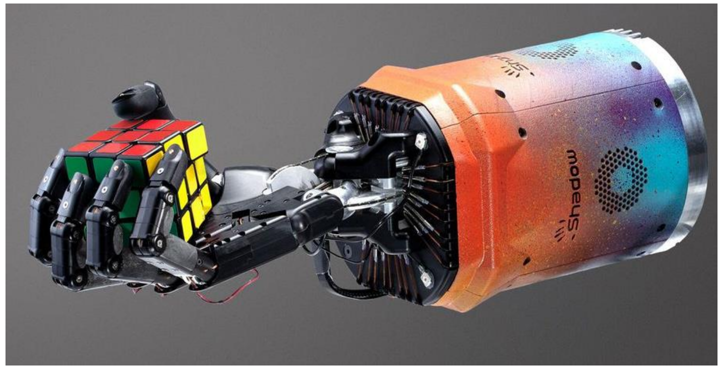
Remarkable Applications

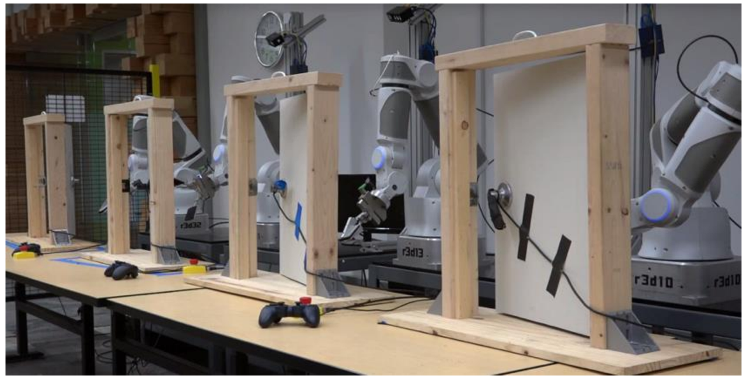
Motivation to learn about reinforcement learning

Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning [DeepMind: Vinyals et al., 2019]

Dota 2 with Large Scale Deep Reinforcement Learning [OpenAI: Berner et al., 2019]

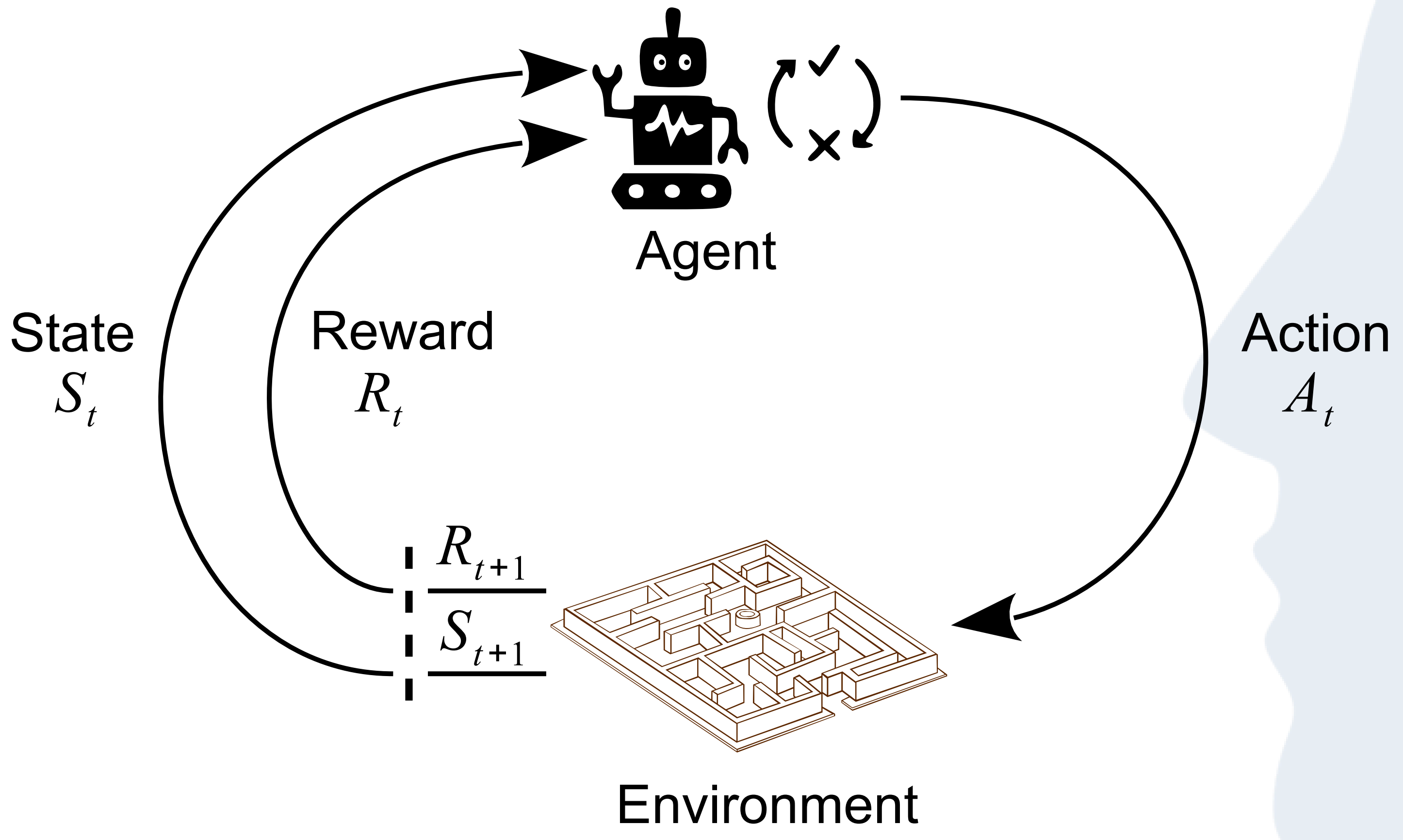
Self-Driving Cars (Tesla Autopilot), Photo: Ian Maddox


Solving Rubik's Cube with a Robot Hand [OpenAI: Akkaya et al., 2019]


Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search [Yahya et al., 2016]

Highly Accurate Protein Structure Prediction with AlphaFold [DeepMind: Jumper et al, 2021]

Reinforcement Learning
Basic Idea



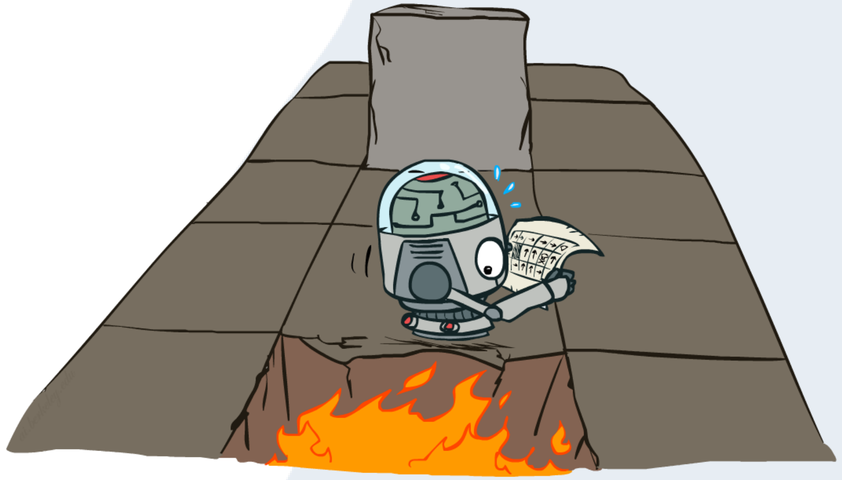
Interaction loop between agent and environment

Reinforcement Learning
Basic Idea

- RL algorithms attempt to find a policy (what action to do in which state) for **maximizing cumulative reward** for the agent over the course of the problem.

- Mathematically, this is typically represented by a **Markov Decision Process (MDP)**

- Keep in mind RL differs from supervised learning in that **correct input/output pairs are never presented**, nor sub-optimal actions explicitly corrected.



Grid world environment

Designed by Dan Klein and Pieter Abbeel (CS188 Intro to AI at UC Berkeley, http://ai.berkeley.edu)

Credit Assignment Problem

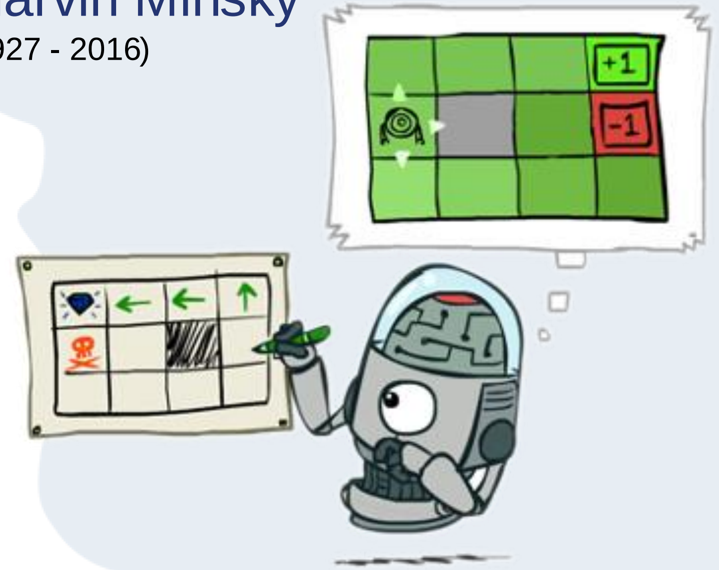
Unfortunately, we face the credit assignment problem

We have to determine how the **success** of a system's overall **performance** is due to the various contributions of the system's **components** (Marvin Minsky, 1963)

In RL, i.e., the temporal CAP:

- Given a **sequence of states and actions**, and the final sum of time-discounted future rewards, how do we infer which **actions** were **effective** at producing lots of reward and which actions were not effective?
- How do we **assign credit** for the observed rewards given a sequence of actions over time?
- Every reinforcement learning algorithm must **address this problem**

Marvin Minsky
(1927 - 2016)

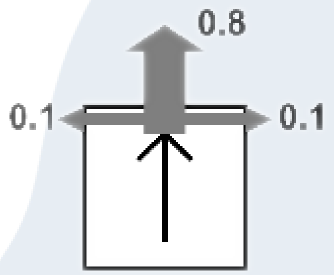


Finding the right plan
Image: Designed by Klein and Abbeel
(CS188, UC Berkeley)

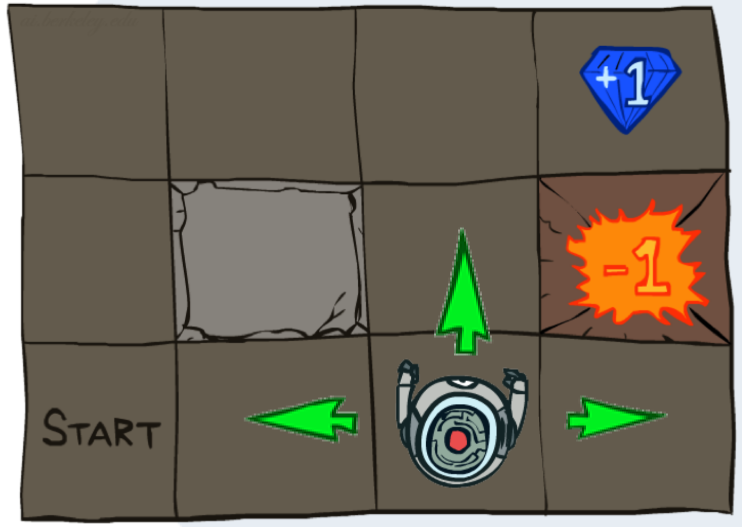
The Fruit Fly of RL: Grid World

An illustrative example

- The agent lives in a **grid**
- **Walls** block the agent's path
- The agent's actions are **noise**, i.e., they do not always go as planned)
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East (if there is no wall there)
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- **Small "living" reward** each step
- **Big rewards** come at the **end**
- **Goal:** maximize sum of rewards*



Stochastic movement

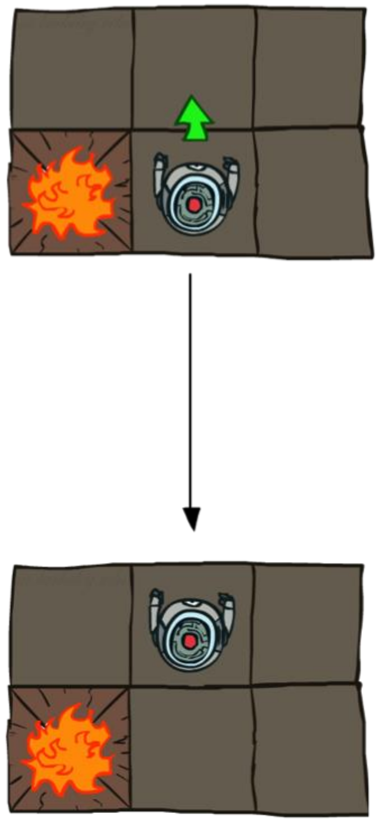


Top-down view of the grid world

Designed by Klein and Abbeel (CS188, UC Berkeley)

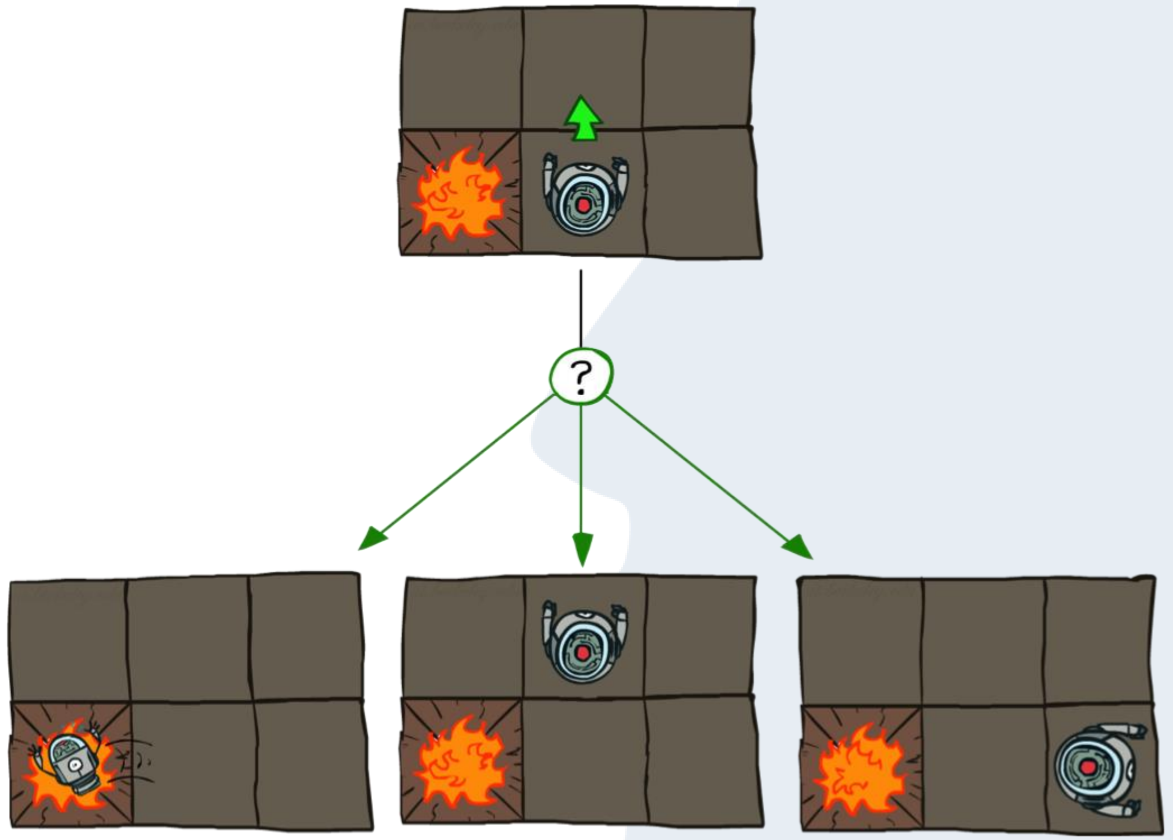
One can imagine two “Grid Futures”

Grid-World Actions



Deterministic grid world

Designed by Klein and Abbeel (CS188, UC Berkeley)



Stochastic grid world

Designed by Klein and Abbeel (CS188, UC Berkeley)

Now, what are Markov Decision Processes (MDP)?

Definition of MDPs

"Markov(ian)"

Generally, means that given the present state, the future and the past are independent.

- An MDP is defined by a 4d-tuple (S, A, T_a, R_a), where
  - S : a **set of states s ∊ S**
  - A : a **set of actions a ∊ A**
  - T_a: a **transition function T(s, a, s')**
    - Probability that a from s leads to s'
    - i.e., P(s' | s,a), also called the model
  - R_a: a **reward function R(s, a, s')**
    - Sometimes just R(s) or R(s')
- A **start state** (or distribution)
- An (optional) **terminal state**
- A **discount factor: γ**

Andrei Markov
(1856-1922)

Now, what are Markov Decision Processes (MDP)?

Definition of MDPs

- MDPs are a family of non-deterministic search problems
- Reinforcement learning: MDPs where we don't know the transition or reward functions

$$\begin{tikzcd} s_0 \arrow[r, "a_0", "r_0"'] & s_1 \arrow[r, "a_1", "r_1"'] & s_2 \arrow[r, "a_2", "r_2"'] & s_3 \end{tikzcd}$$

- For Markov decision processes, "Markov" means:

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \dots S_0 = s_0)$$

$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Andrei Markov
(1856-1922)

What does it mean to solve MDPs?

Finding the optimal plan

Recall, in deterministic single-agent search problems we want an optimal **plan**, or sequence of actions, from start to a goal

In an MDP, we want to compute an optimal policy $\pi^*: S \rightarrow A$

- A policy π gives an action for each state
- An optimal policy maximizes expected utility if followed
- Defines a reflex agent

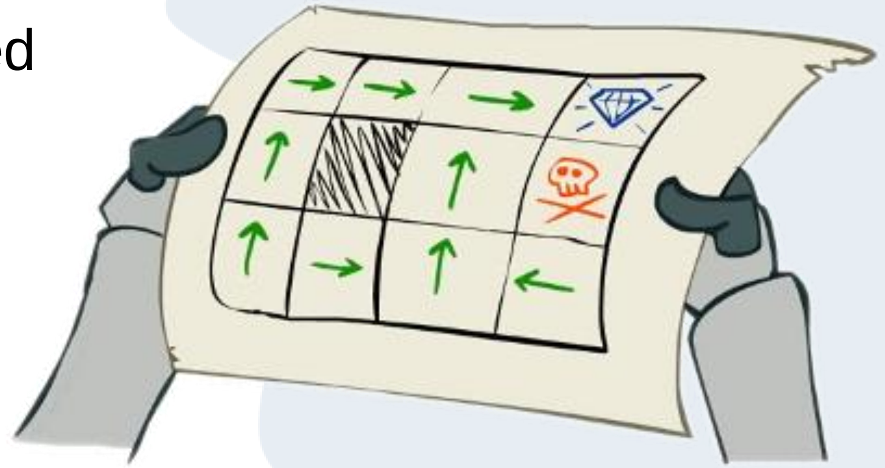
- Policy π: Rule to decide what actions to take

$$a_t = \mu(s_t)$$

Deterministic policy

$$a_t \sim \pi(\cdot|s_t)$$

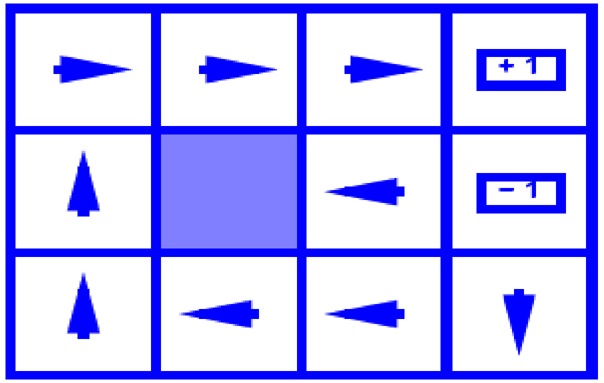
Stochastic policy



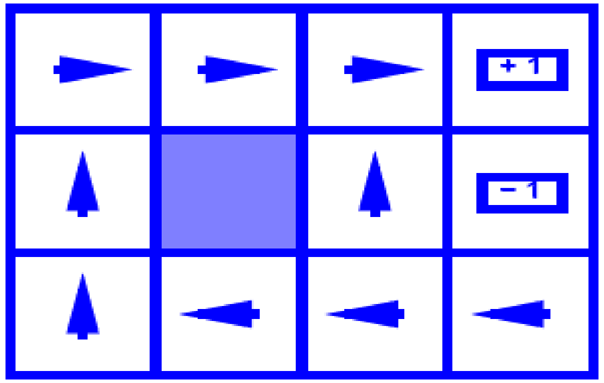
Finding the correct actions

Designed by Klein and Abbeel (CS188, UC Berkeley)

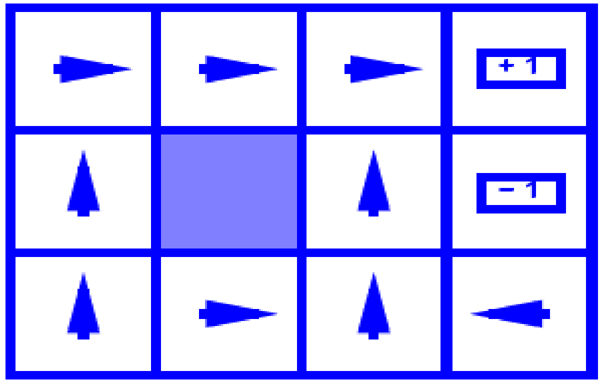
What does it mean to solve MDPs?



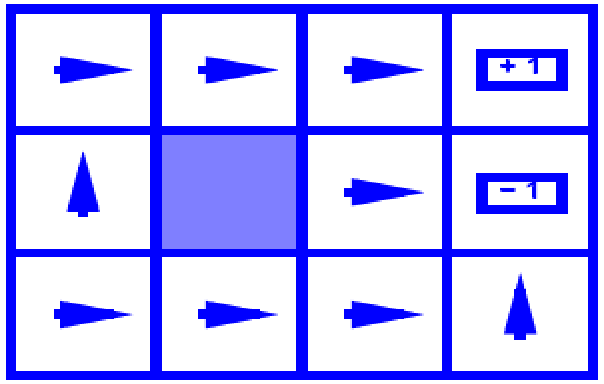
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Depending on the action rewards/costs, optimal policies may look differently

Recap: Defining MDPs

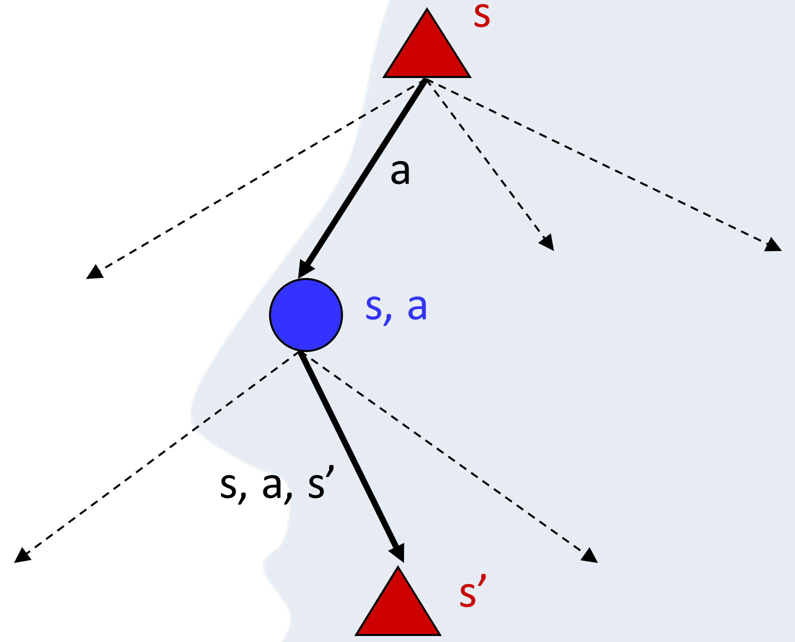
Formal definition

Markov decision processes:

- States S
- Start state s_0
- Actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (and discount γ)

MDP **quantities** so far:

- Policy = Choice of action for each state
- Utility (or return) = sum of discounted rewards



Graph of an MDP

Intuition of Action Selection

Formalizing our goal

At time t, the agent tries to select an action to maximize the sum G_t of discounted rewards received in the future

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

Maximizing the discounted sum of rewards

Designed by Klein and Abbeel (CS188, UC Berkeley)

Why Discounted Rewards?
Formalizing our goal

Rewards in the future are **worth less** than an immediate reward. This helps to ensure convergence of the expectations.

To this end, we introduce a **Discount factor $\gamma \leq 1$**

(often $\gamma = 0.9$)

- Assume reward n years in the future is only worth $(\gamma)^n$ of the value of immediate reward
    - $(0.9)^6 * 10{,}000 = 0.531 * 10{,}000 = 5310$

For each state, calculate a **utility value** equal to the

Sum of Future Discounted Rewards

Optimal Utilities

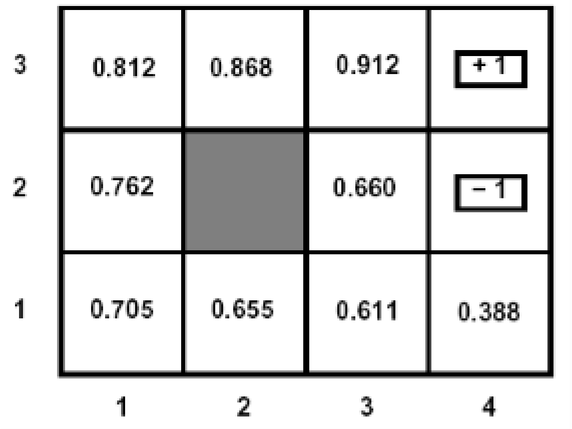
Fundamental operation of computing the values (optimal expectimax utilities) of states s (based on the rewards)

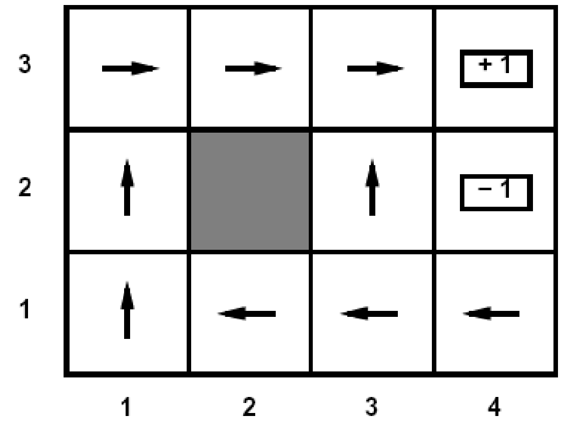
- Why? Optimal values define optimal policies!

- **Value Function** defines the value of a state s:
  - $V^*(s)$ = expected utility starting in s and thereafter acting optimally

- **Action-Value Function** defines the value of a q-state (s, a):
  - $Q^*(s,a)$ = expected utility starting in s, **taking action a** and thereafter acting optimally

- Define the **optimal policy**:
  - $\pi^*(s)$ = optimal action from state s





Values of each state and its corresponding policy

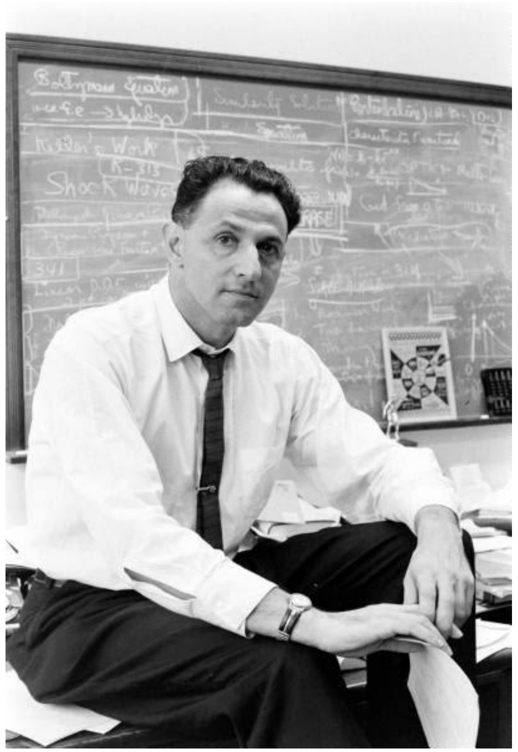
Optimality
Brief Introduction

- A policy is defined to be **better than or equal** to another policy if its expected return is greater than or equal to that of the other policy for all states

- There is **always** at least **one optimal policy π^*** that is better than or equal to all other policies

- **All optimal policies** share the **same optimal state-value function v***, which gives the maximum expected return for any state s over all possible policies

- **All optimal policies** share the **same optimal action-value function q***, which gives the maximum expected return for any state-action pair (s, a) over all possible policies


Richard Bellman
(1920 – 1984)

How to find a Solution that is optimal?
Formalization

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s))\, V^{\pi}(s')$$

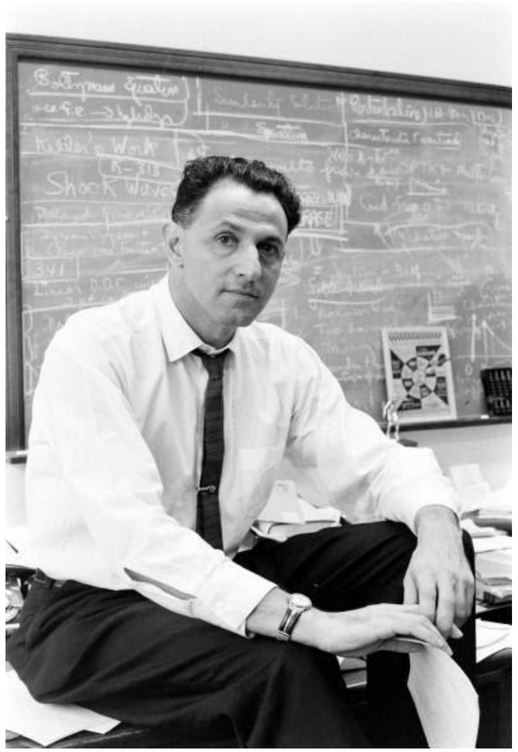
Bellman Equation of Optimality

V^{π}: value of state s when acting according to π

$R\big(s, \pi(s)\big)$: immediate reward

γ: discount factor

$T(s' \mid s, \pi(s))$: transition probability to arrive in s'


Richard Bellman
(1920 – 1984)

How can we compute, e.g., the value function?

Dynamic Programming: Value Iteration

Idea:

- Start with $V_0^*(s) = 0$
- Given V_i^*, calculate the values for all states for depth i+1:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma\, V_k(s')]$$

- This is called a **value update** or **Bellman update**
- Repeat until convergence
- Complexity of each iteration: $O(S^2 A)$

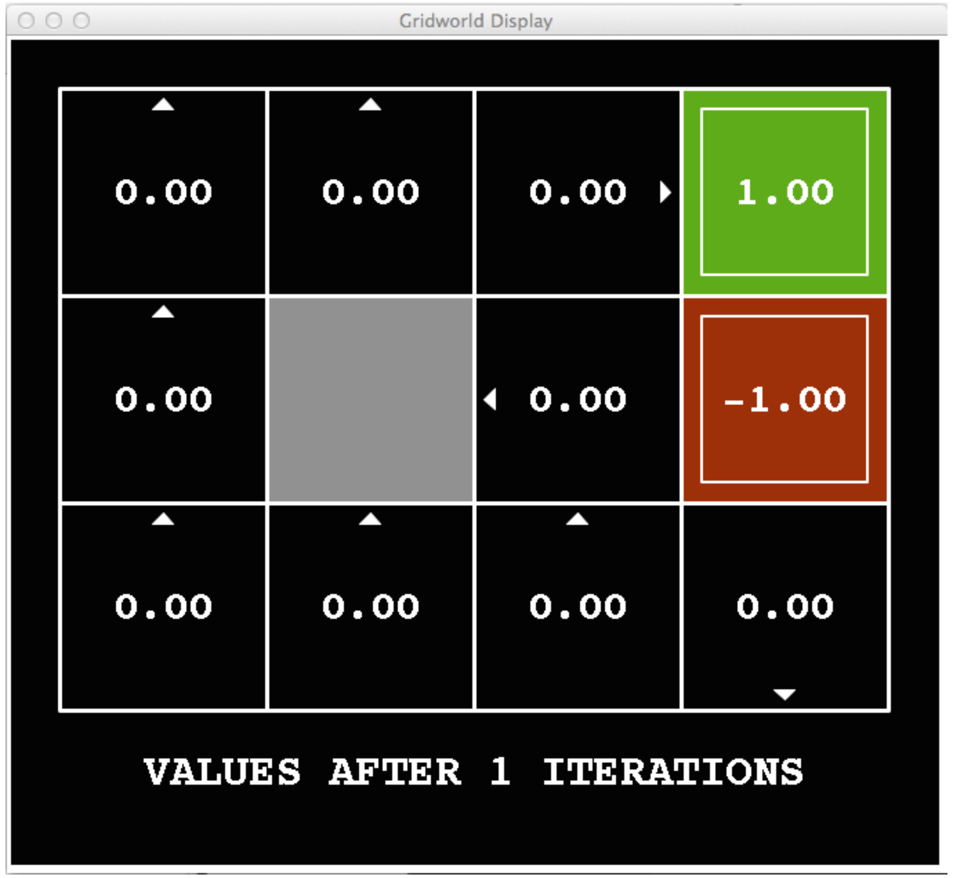
Using value iteration

Designed by Klein and Abbeel (CS188, UC Berkeley)

Theorem: will converge to unique optimal values

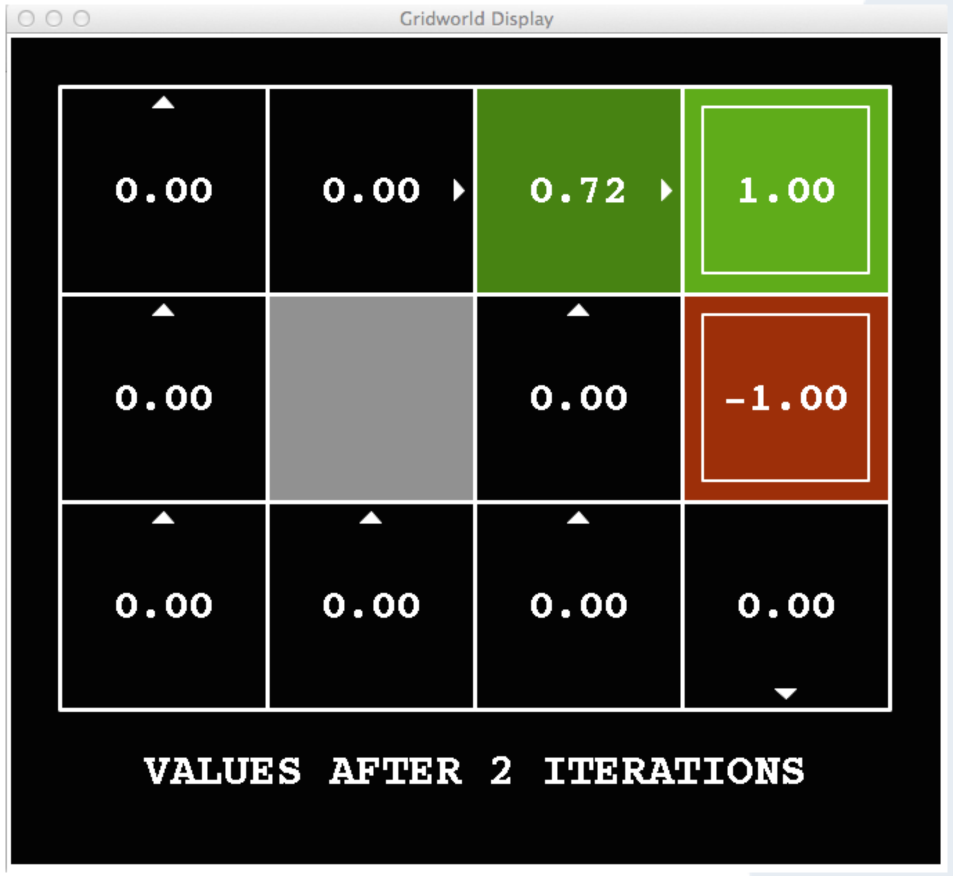
- Basic idea: approximations get refined towards optimal values
- Policy may converge long before values do

Value Iteration
Example



0.00	0.00	0.00	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

VALUES AFTER 1 ITERATIONS



0.00	0.00	0.72	1.00
0.00		0.00	-1.00
0.00	0.00	0.00	0.00

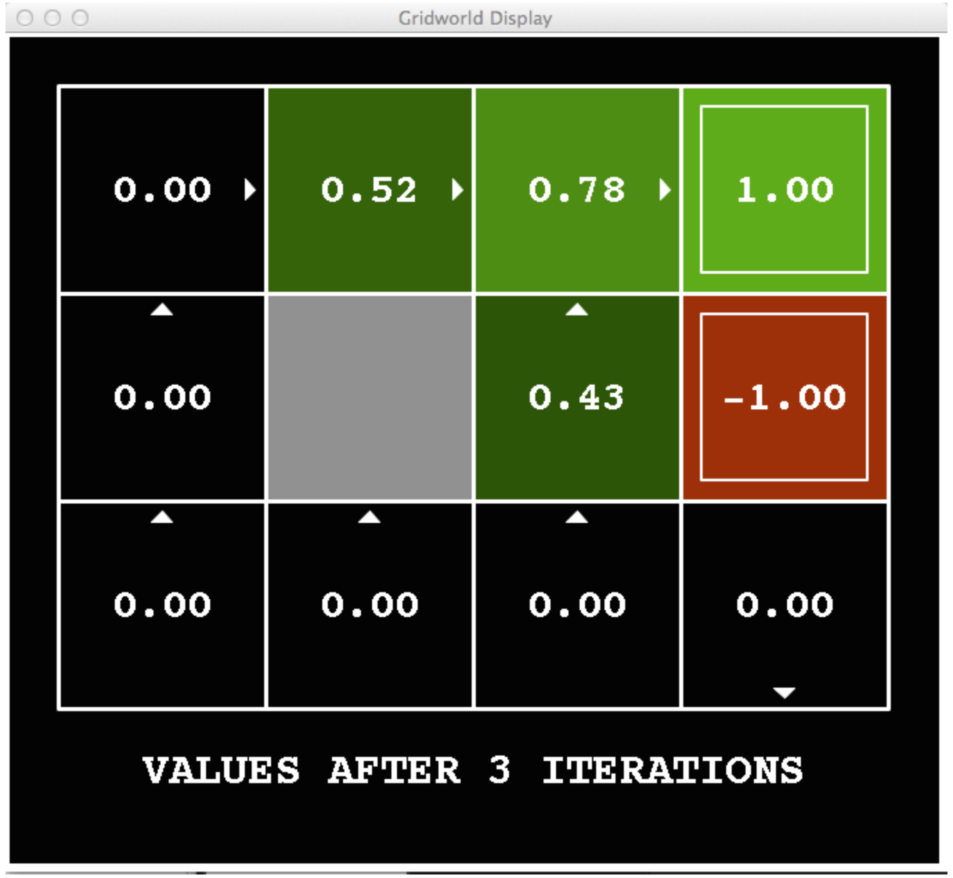
VALUES AFTER 2 ITERATIONS

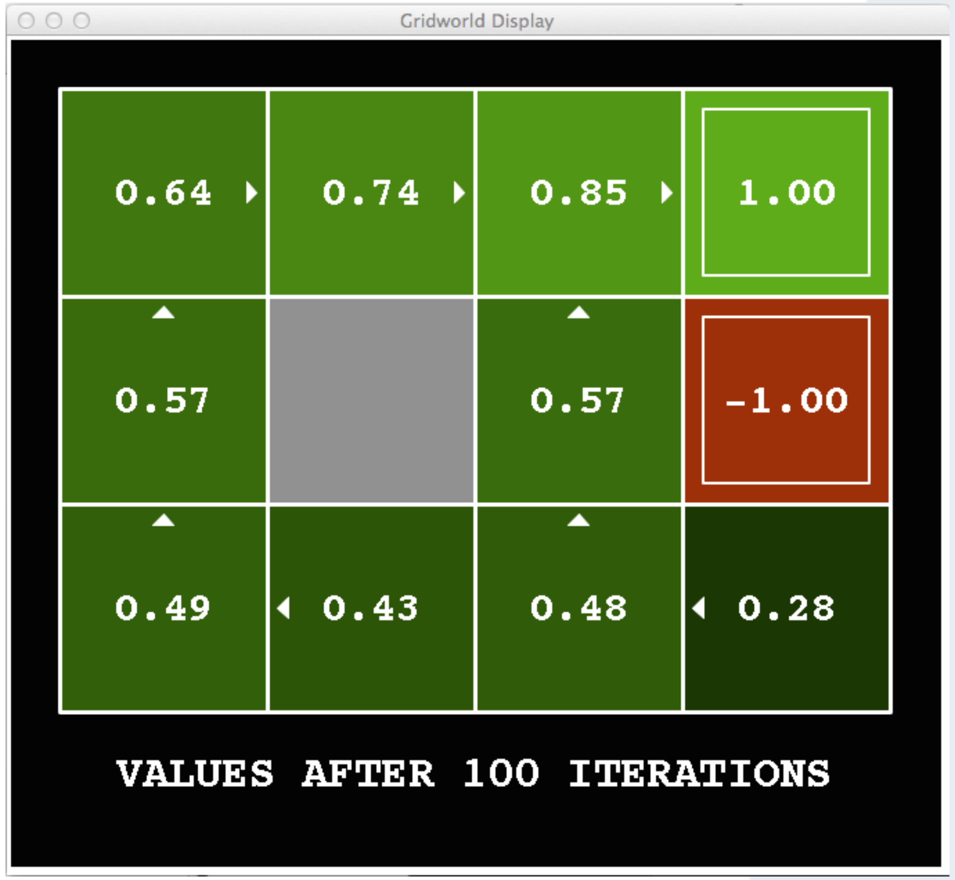
Parameters:
Noise = 0.2
Discount = 0.9
Living reward = 0

Source: Klein and Abbeel (CS188, UC Berkeley)

Information propagates outward from terminal states and eventually all states have correct value estimates

Value Iteration
Example





Parameters:
Noise = 0.2
Discount = 0.9
Living reward = 0

Source: Klein and Abbeel (CS188, UC Berkeley)

Information propagates outward from terminal states and eventually all states have correct value estimates

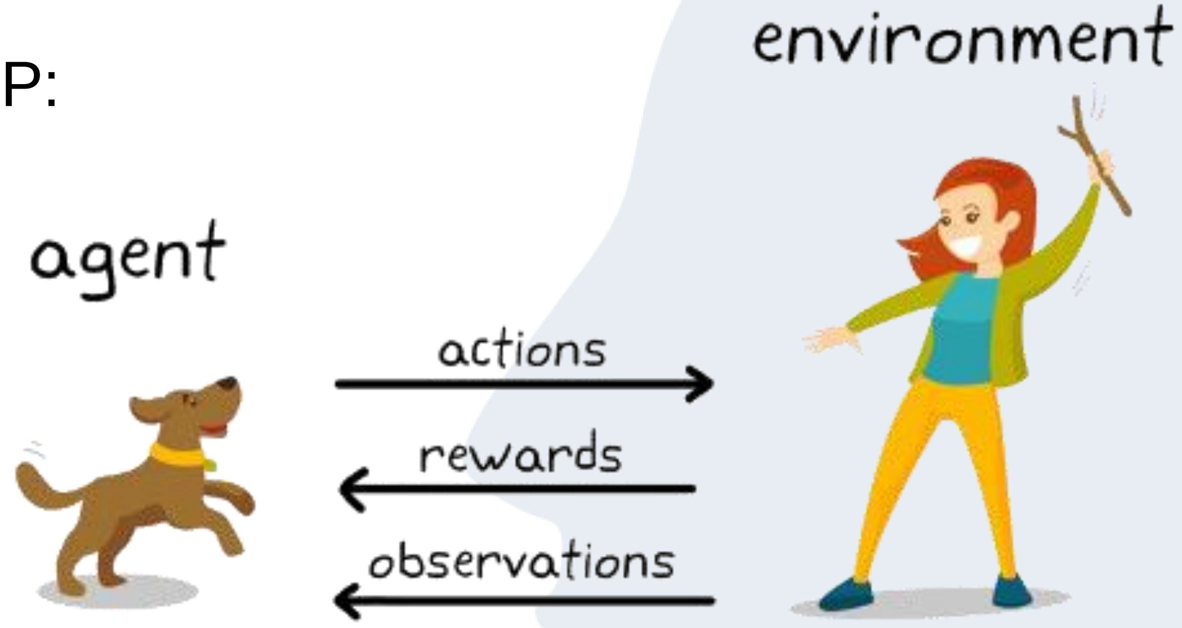
OK, but what is now Reinforcement Learning?

Why does value iteration not always work?

Reinforcement learning still assumes an MDP:

- A set of states $s \in S$
- A set of actions (per state) A
- A model T(s, a, s')
- A reward function R(s, a, s')
- A discount factor γ (could be 1)

and we still seek for a policy $\pi(s)$



Reinforcement learning in dog training

Source: https://analyticsindiamag.com/three-things-to-know-about-reinforcement-learning/

New twist: we don't know T(s, a, s') or R(s, a, s')

- i.e. don't know which states are good or what the actions do
- Must actually try actions and states out to learn

Model-based v.s. Model-free

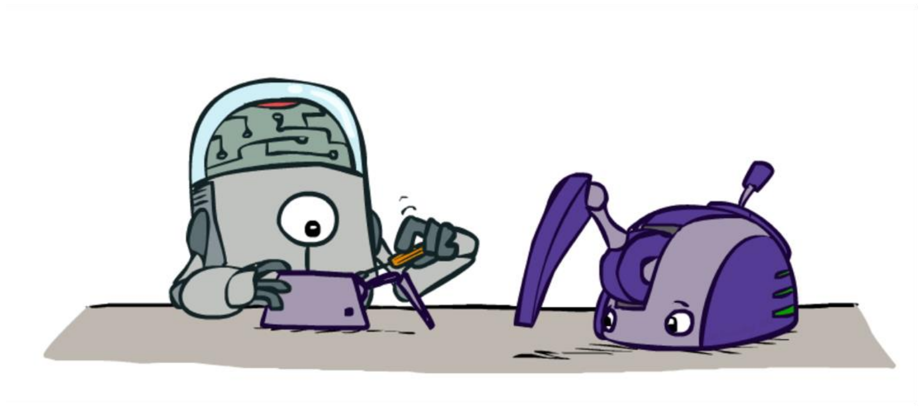
Types of Reinforcement Learning

Model-based approach RL

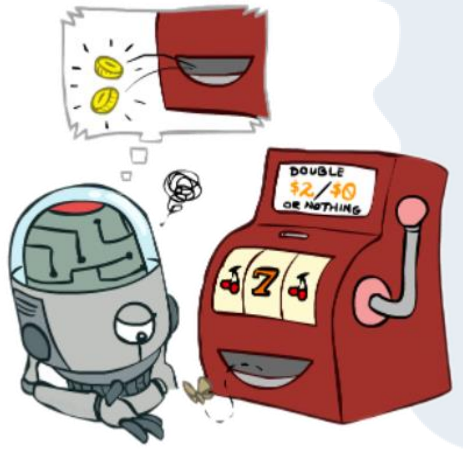
Learn (or use) the model and use it to derive the optimal policy.

Model-free approach RL

Derive the optimal policy without learning the model.



Model-based learning
Designed by Klein and Abbeel
(CS188, UC Berkeley)



Model-free learning
Designed by Klein and Abbeel
(CS188, UC Berkeley)

On-policy v.s. Off-policy
Types of Reinforcement Learning

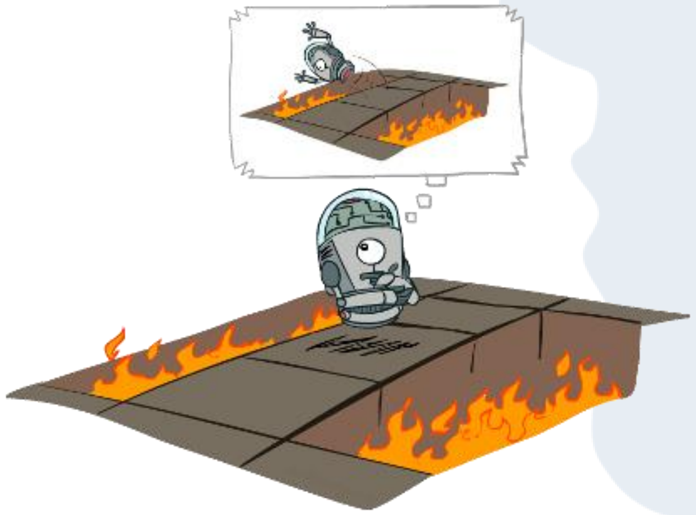
On-policy RL

An on-policy agent learns only about the policy that it is executing.

Off-policy RL

An off-policy agent learns about a policy or policies different from the one that it is executing.

Online learning
Designed by Klein and Abbeel
(CS188, UC Berkeley)



Offline solution
Designed by Klein and Abbeel
(CS188, UC Berkeley)

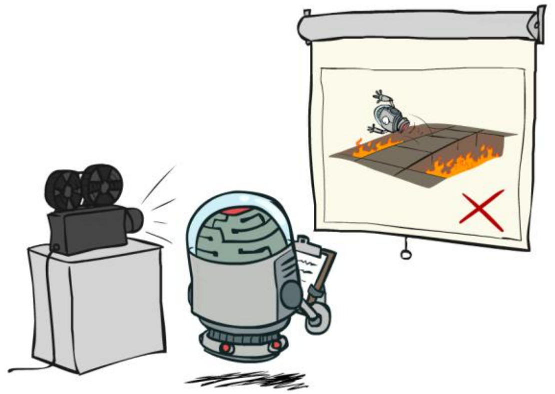
Passive learning v.s. Active learning
Types of Reinforcement Learning

Passive Learning

The agent simply watches the world going by and tries to learn the utilities of being in various states. The goal is to execute a fixed policy (sequence of actions) and evaluate it.

Active Learning

The agent not simply watches, but also acts. The goal is to act and learn an optimal policy.


Passive learning
Designed by Klein and Abbeel (CS188, UC Berkeley)

Active learning
Designed by Klein and Abbeel (CS188, UC Berkeley)

Passive Learning

Types of Reinforcement Learning

Simplified task

- You don't know the transitions T(s,a,s')
- You don't know the rewards R(s,a,s')
- You are given a policy π(s)
- **Goal:** learn the state values of the policy

In this case:

- Learner "along for the ride"
- No choice about what actions to take
- Just execute the policy and learn from experience
- We'll get to the active case soon
- This is **NOT offline planning!** You actually take actions in the world and see what happens…

Model-Based Learning

Types of Reinforcement Learning

Idea:

- Learn the model empirically through experience
- Solve for values as if the learned model were correct

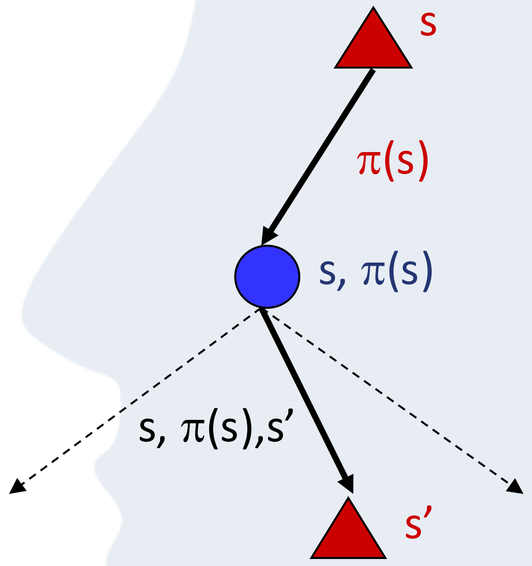
Simple empirical model learning

- Count outcomes for each s, a
- Normalize to give estimate of **T(s, a, s')**
- Discover **R(s, a, s')** when we experience (s, a, s')

Solving the MDP with the learned model

- Iterative policy evaluation, for example

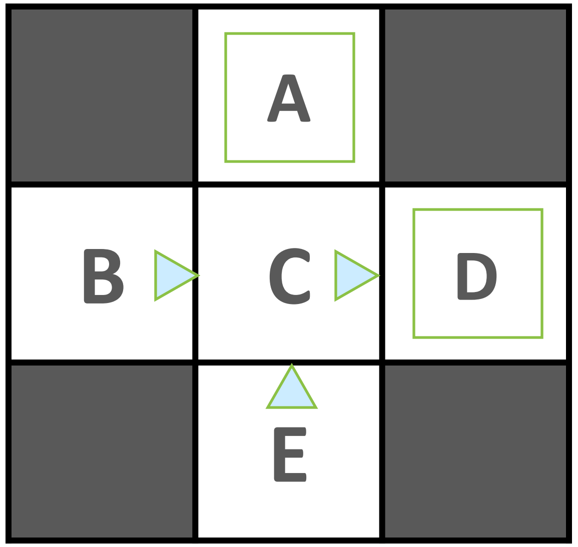
$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_i^{\pi}(s')]$$



Graph of an MDP

Model-Based Learning
Example

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
…

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
…

Sample-Based Policy Evaluation?

Types of Reinforcement Learning

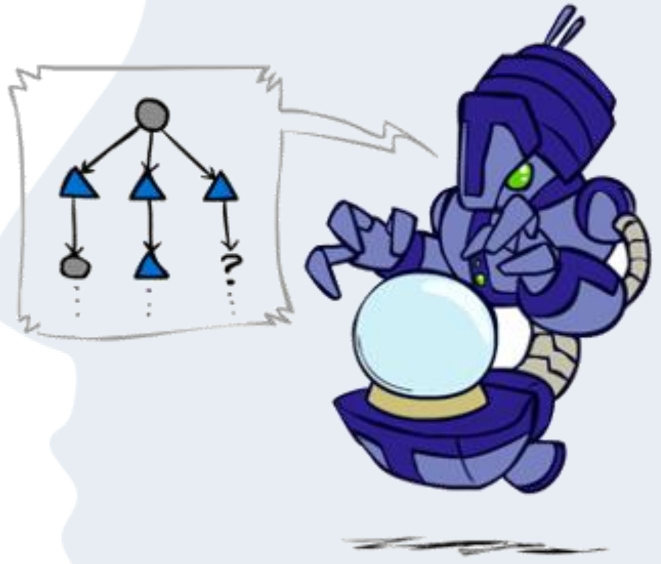
We want to improve our estimate of V by computing these averages:

$$V_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i^{\pi}(s')]$$

Idea: Take samples of outcomes s' (by doing the action!) and average

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$
$$\dots$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i sample_i$$



Trying out sample-based policy evaluation

Designed by Klein and Abbeel (CS188, UC Berkeley)

Temporal-Difference (TD) Learning

Types of Reinforcement Learning

Big idea: learn from every experience!

- Update V(s) each time we experience (s, a, s', r)
- Likely s' will contribute updates more often

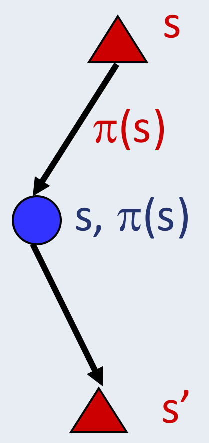
Temporal difference learning

- Policy still fixed!
- Move values toward value of whatever successor occurs: running average!

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha) V^{\pi}(s) + (\alpha) sample$

Same update:
(Explains the term TD) $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

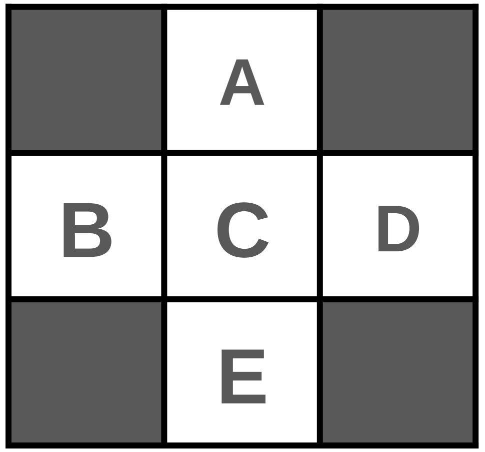


Graph of an MDP

Temporal-Difference (TD) Learning

Example

States

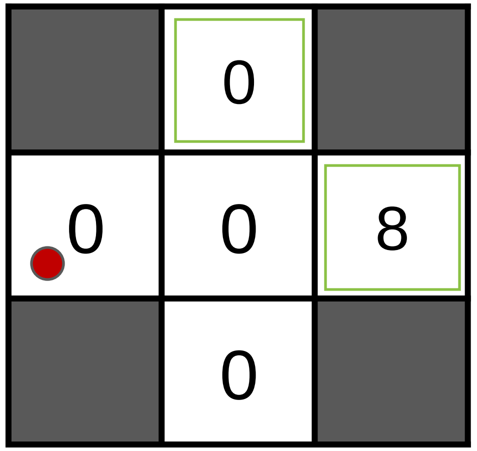


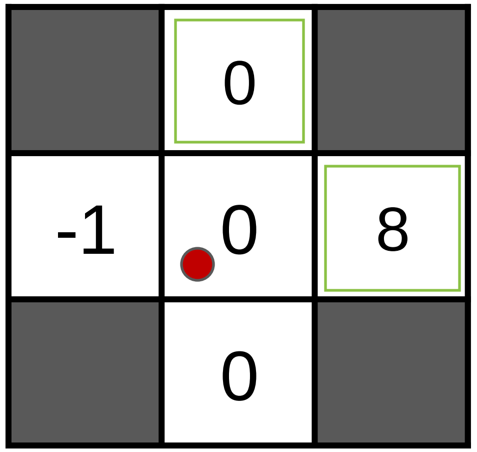
Assume: $\gamma = 1$, $\alpha = 1/2$

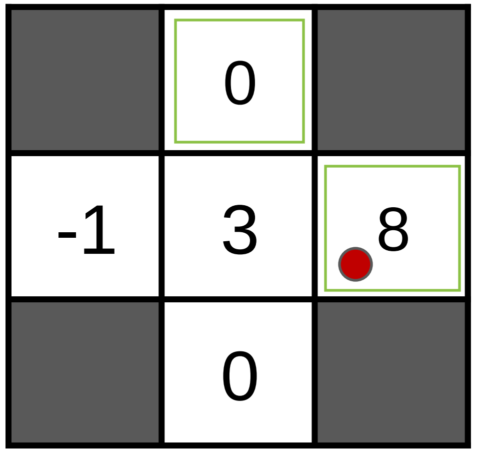
Observed Transitions

B, east, C, -2

C, east, D, -2







$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^{\pi}(s')\right]$$

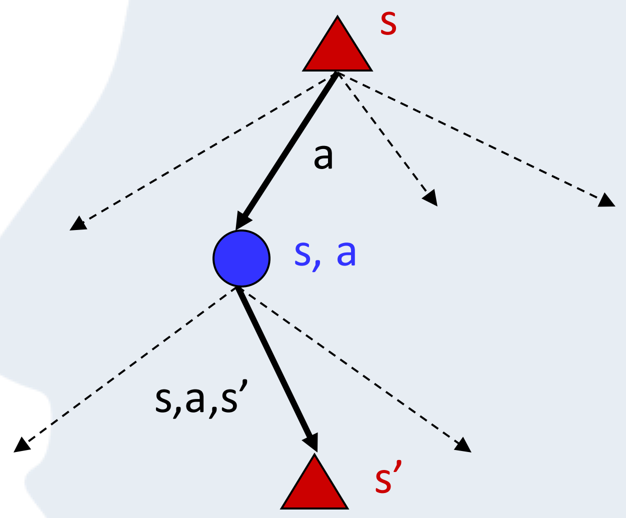
Problems with TD Value Learning
Types of Reinforcement Learning

TD value learning is a model-free way to do policy evaluation. However, if we want to turn values into a (new) policy, we're sunk:

$$\pi(s) = \arg\max_a Q(s,a)$$

$$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$$

- **Idea:** learn Q-values directly
- Makes action selection model-free too!



Graph of an MDP

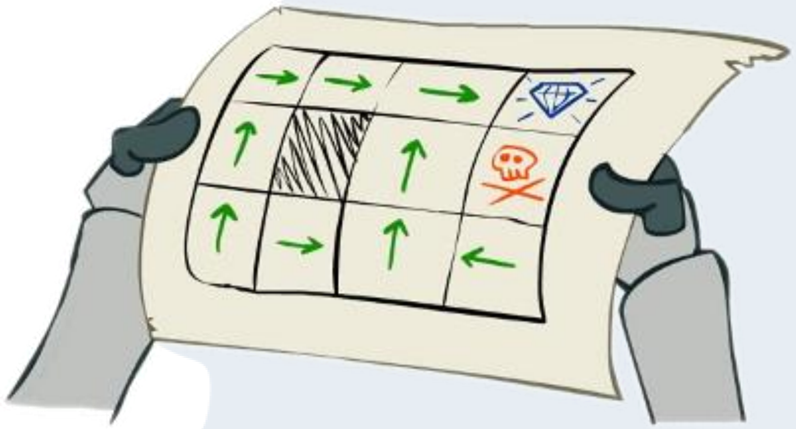
Active Learning
Types of Reinforcement Learning

Full reinforcement learning

- You don’t know the transitions T(s,a,s’)
- You don’t know the rewards R(s,a,s’)
- You can choose any actions you like
- **Goal:** learn the optimal policy

In this case:

- Learner makes choices!
- Fundamental trade-off: exploration vs. exploitation
- This is NOT offline planning! You actually take actions in the world and find out what happens…



Finding the correct actions

Designed by Klein and Abbeel (CS188, UC Berkeley)

Q-Learning

Types of Reinforcement Learning

Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

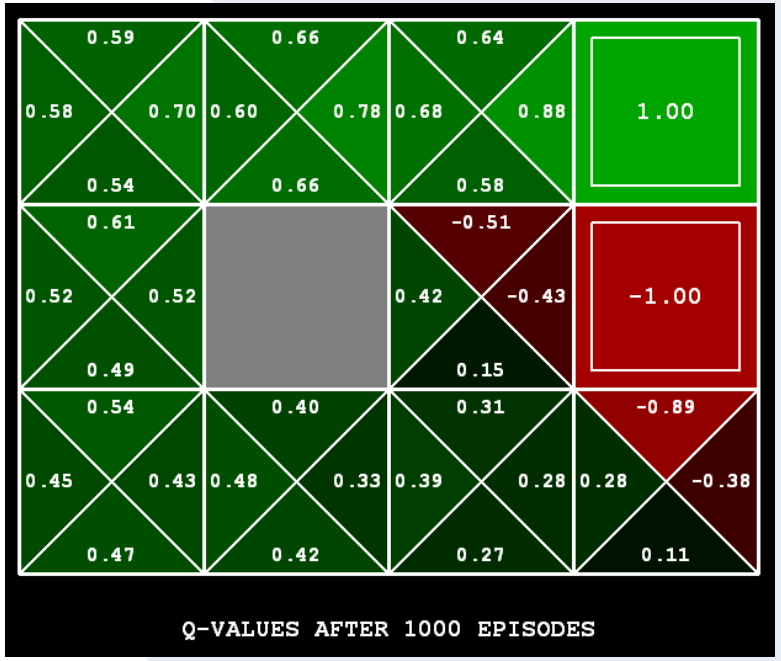
Learn Q*(s,a) values

- Receive a sample (s, a, s’, r)
- Consider your old estimate: $Q(s,a)$
- Consider your new sample estimate:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

- Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[sample\right]$$



Converged Q-Values for initial grid-world environment

Source: Klein and Abbeel (CS188, UC Berkeley)

Q-Learning
Properties

- **Amazing result:** Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Basically, doesn't matter how you select actions (!)

- **Problems:** basic Q-Learning keeps a table of all q-values
  - In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

Storing Q-values in a gigantic Q-table

Designed by Klein and Abbeel (CS188, UC Berkeley)

Exploration / Exploitation
Dilemma

Several schemes for forcing exploration

- Simplest: random actions (ε **greedy**)
  - Every time step, flip a coin
  - With probability ε, act randomly
  - With probability 1-ε, act according to current policy

- Problems with random actions?
  - You do explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

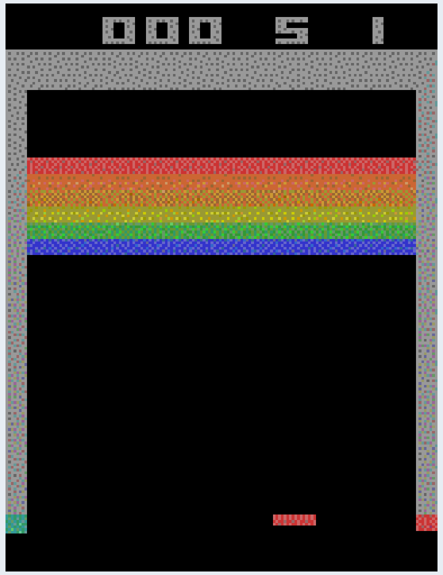
Exploring your environment

Designed by Klein and Abbeel (CS188, UC Berkeley)

Deep Q-Networks (DQN)
Properties

- Deep Q-Networks is Q-learning with a deep neural network function approximator called the Q-network
  - Discrete and finite set of actions A
  - Uses epsilon-greedy policy to select actions

- **Example:** Breakout has 3 actions
  - Move left
  - Move right
  - Idle (no movement)



Breakout Atari environment

Source:https://www.gymlibrary.dev/environments/atari/breakout

The Story So Far: MDPs and RL
Overview

Things we know how to do:

- If we know the MDP
  - Compute V*, Q*, π* exactly
  - Evaluate a fixed policy π
- If we don't know the MDP
  - We can estimate the MDP then solve
  - We can estimate V for a fixed policy π
  - We can estimate Q*(s,a) for the optimal policy while executing an exploration policy

Techniques:

- **Model-based DPs**
  - Value and policy Iteration
  - Policy evaluation
- **Model-based RL**
- **Model-free RL:**
  - Value learning
  - Q-learning

Policy Search

Types of Reinforcement Learning

Problem:

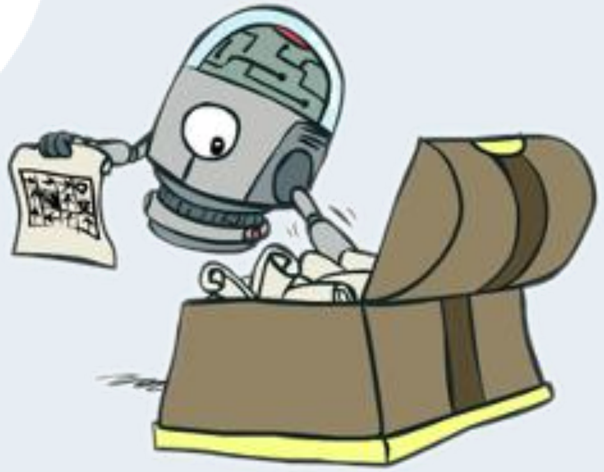
often the feature-based policies that work well aren't the ones that approximate V / Q best

Solution:

learn the policy that maximizes rewards rather than the value that predicts rewards

We may parameterize the policy and then run some gradient optimization, e.g., REINFORCE

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta}))} \qquad \boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\gamma^t G_t \frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}.$$



Learning the policy directly

Designed by Klein and Abbeel (CS188, UC Berkeley)

Policy Search
Properties and limitations

Simplest policy search:

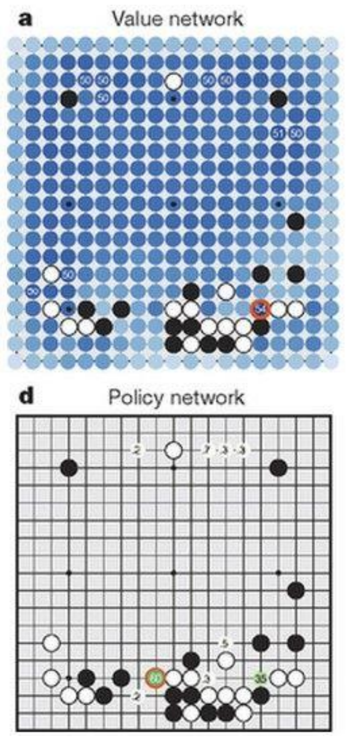
- Start with an initial linear value function or q-function
- Nudge each feature weight up and down and see if your policy is better than before

Problems:

- How do we tell the policy got better?
- Need to run many sample episodes!
- If there are a lot of features, this can be impractical

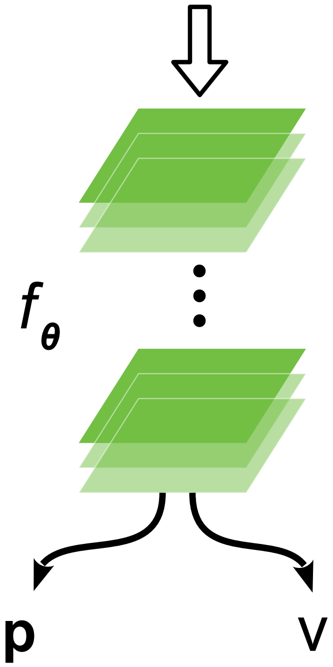
Or one may go for deep neural networks

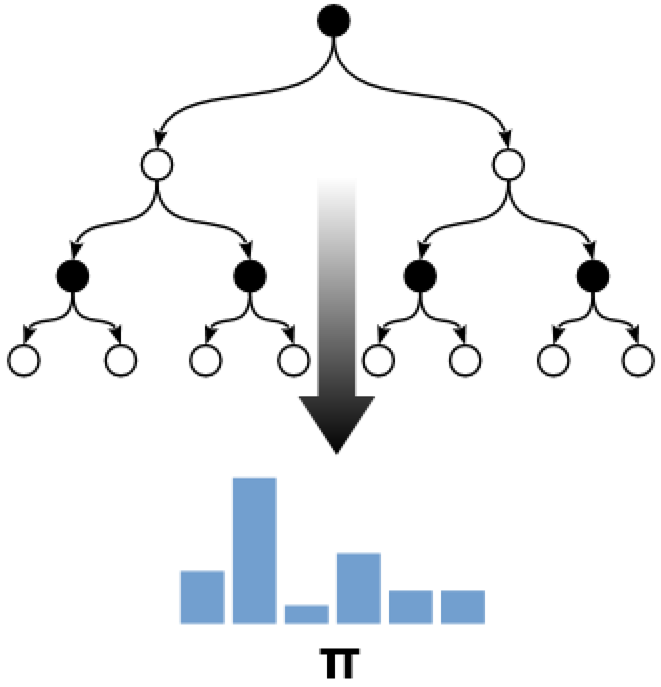
Introduction to AlphaGo / AlphaGoZero / AlphaZero

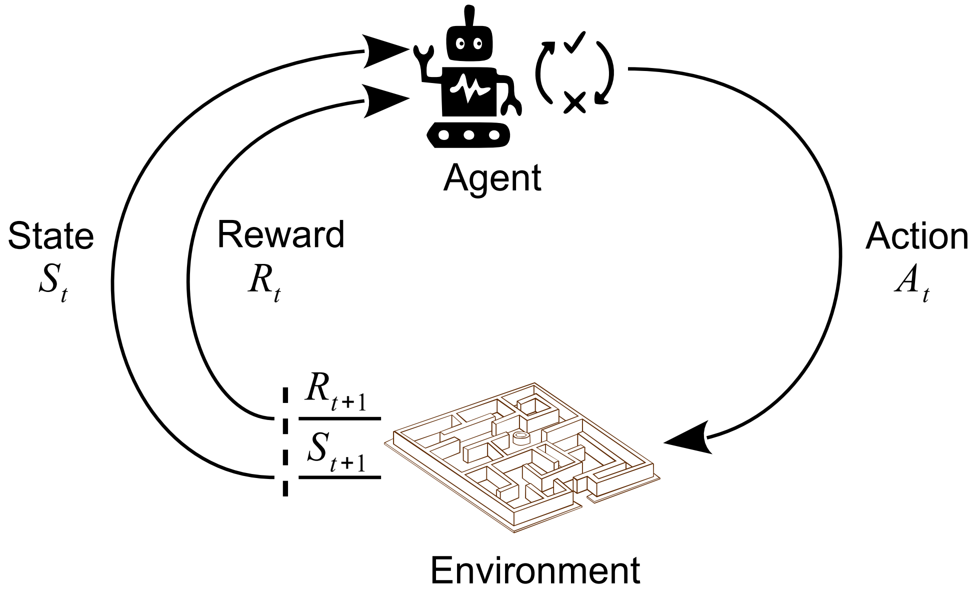


- Deep policy network is trained to produce probability map of promising moves. The deep value network is used to prune the (mcmc) search tree

Components of AlphaZero
Overview







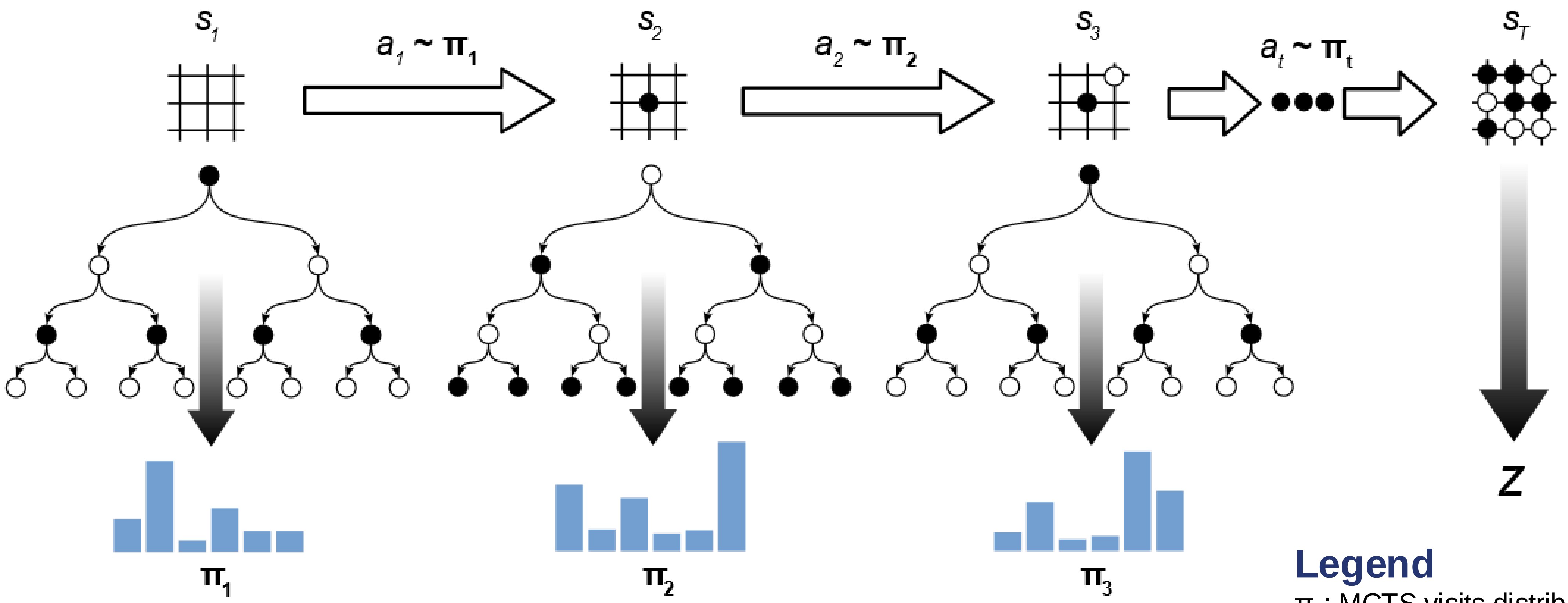
Deep learning network
Value & Policy Network

Tree search algorithm
Predictor Upper Confidence Bounds for Trees Algorithm (PUCT)

Neural network optimization
Supervised Training / Reinforcement Learning

Combining Tree Search with Neural Networks

Generating games as training data



Selfplay iteration to build target policy

Based on design by DeepMind

Legend

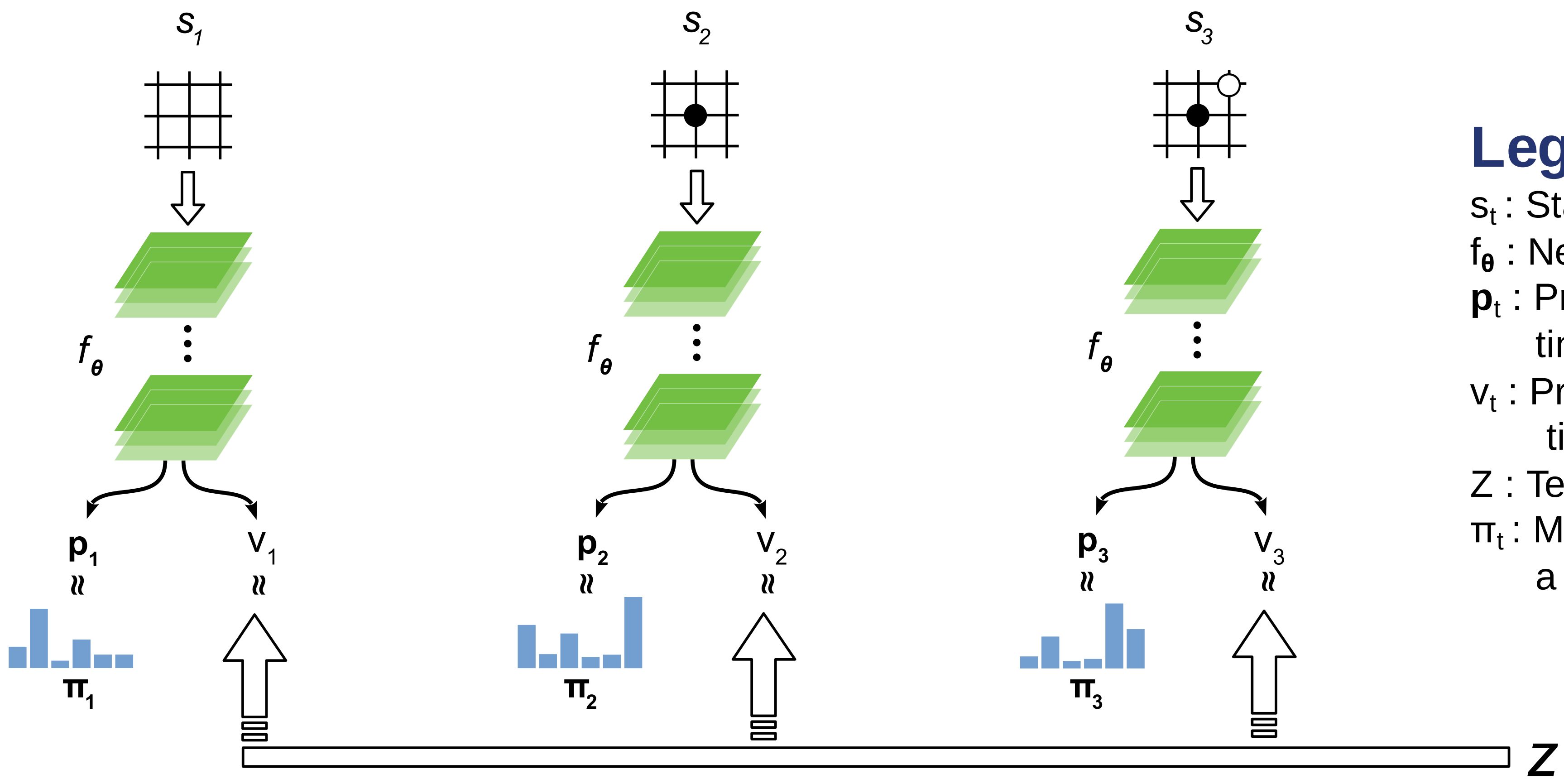
π_t : MCTS visits distribution used as a policy for move selection

S_T : Terminal Node

Z : Terminal Return {-1, 0, +1}

Using Search as an Improvement Operator

Using the generated data to update the neural network



Legend

s_t : State at time step t

f_θ : Neural network with weights θ

$\mathbf{p}_t$: Predicted policy distribution at time step t

v_t : Predicted value for state at time step t

Z : Terminal Return {-1, 0, +1}

π_t : MCTS visits distribution used as a policy for move selection

Neural network optimization

Based on design by DeepMind

Training Objective
Loss formulization

$$l = \underbrace{\alpha(z - v)^2}_{\text{Mean Squared Error}} - \underbrace{\pi^T \log \mathbf{p}}_{\text{Cross Entropy}} + \underbrace{c\|\theta\|^2}_{\text{Regularizer}}$$

Loss function definition
Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm [Silver et al., 2017]

α: value loss factor

z: target value

v: predicted value

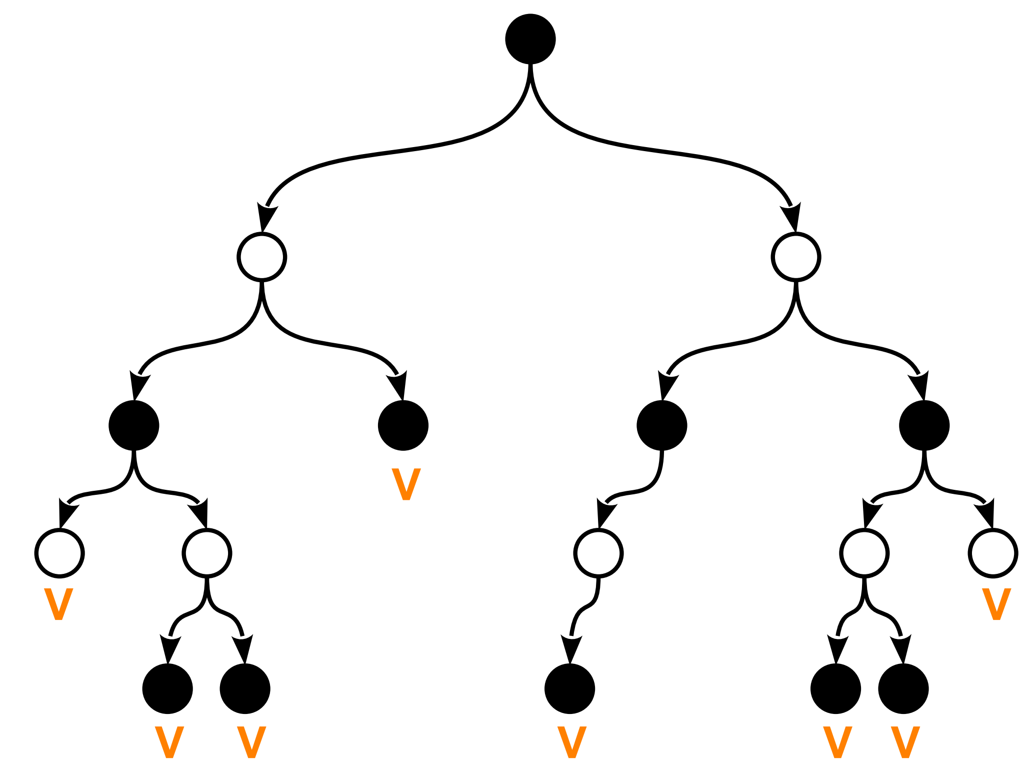
$\boldsymbol{\pi}^{\boldsymbol{T}}$: MCTS simulation distribution

$\boldsymbol{p}$: policy head output

c: L_2 regularization constant

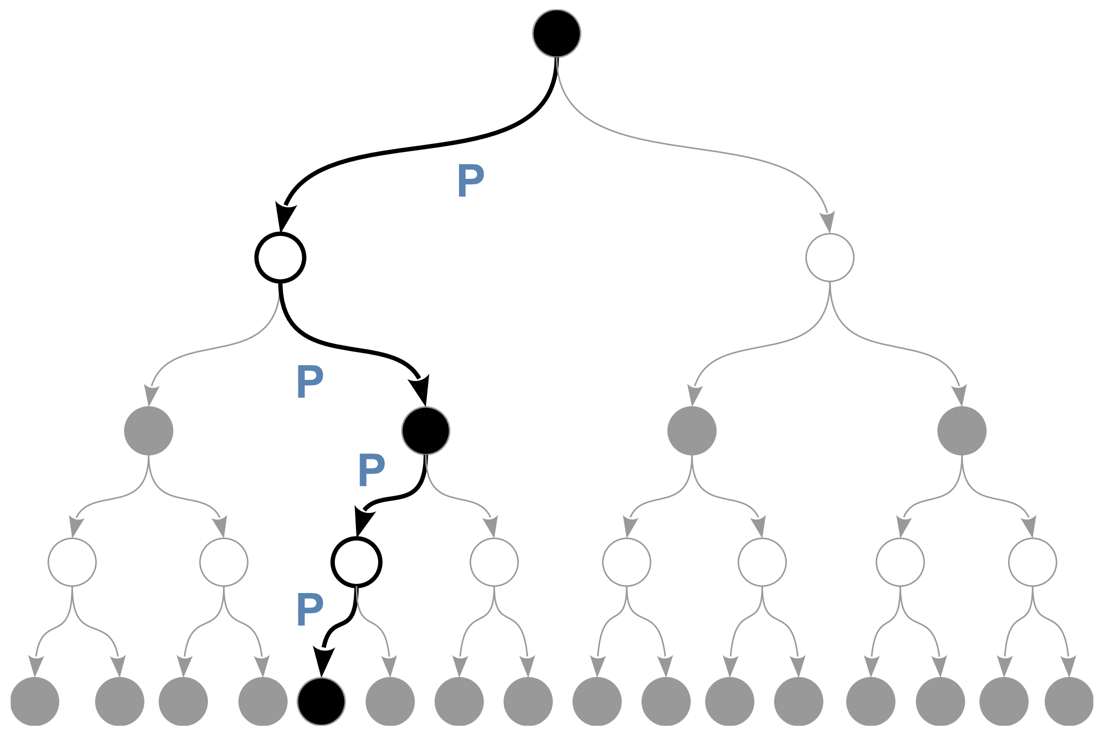
Search Principles of AlphaZero

Using a shared value / policy neural network



Limit depth of the search using the value prediction

Based on design by DeepMind



Reduce breadth of the search using the policy prediction

Based on design by DeepMind

PUCT-Algorithm

Predictor Upper Confidence Bounds for Tree Algorithm (PUCT) [Rosin, 2011]

- Exploration by rollouts
- Move selection:

$$a_t = \mathrm{argmax}_a\big(Q(s_t, a) + U(s_t\, , a)\big) \quad ,$$

$$\text{where} \quad U(s_t, a) = c_{\text{puct}}\, P(s,a)\, \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \quad .$$

- Update Q-values by simple moving average (SMA):

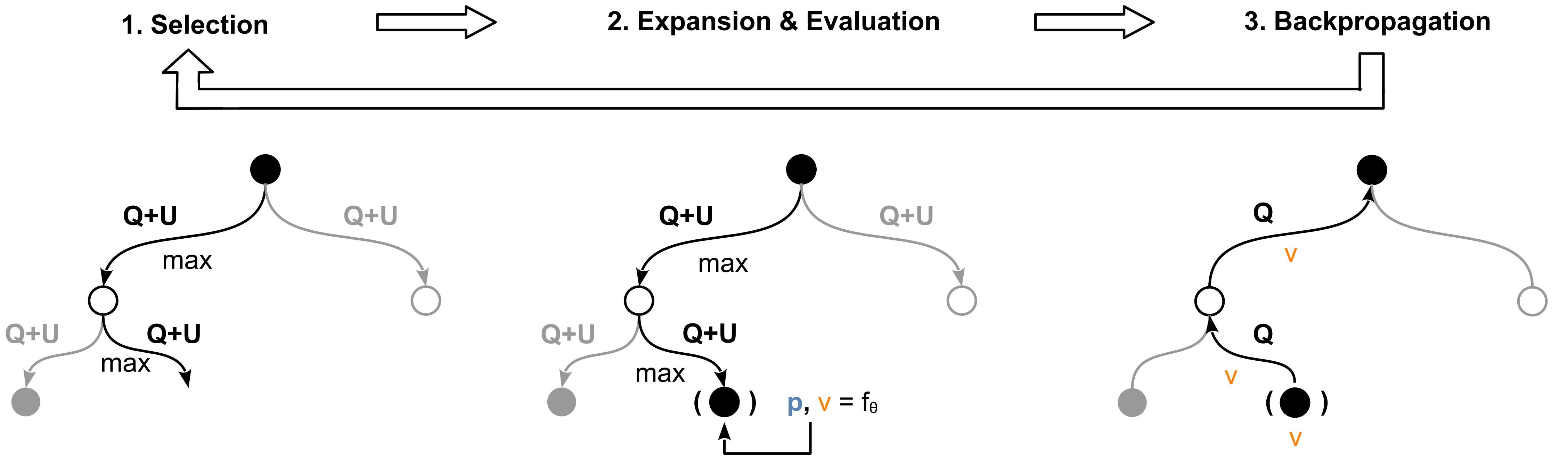
$$Q'(s_t, a) \leftarrow Q(s_t, a) + \frac{1}{n}\left[v - Q(s_t, a)\right] \, .$$

Legend

c_{puct} : scalar value to manage exploration
U(s_t , a) : utility values
N(s, a) : number of visits of action a
Q(s_t , a) : state action values
s_t : state s at time step t
a : action a
v : predicted value by neural network in (-1, +1)

Monte-Carlo Tree Search (MCTS)

Phases of MCTS



MCTS phases in AlphaZero

Based on design by DeepMind

Follow up
Video series, lecture series, coding tutorials

Video Courses

RL Course by David Silver 2015 (DeepMind)
https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ

DeepMind x UCL RL Lecture Series 2021
https://www.youtube.com/watch?v=TCCjZe0y4Qc

Stanford CS234: Reinforcement Learning | Winter 2019
https://www.youtube.com/watch?v=FgzM3zpZ55o&list=PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u

Reinforcement Learning - Developing Intelligent Agents
https://www.youtube.com/watch?v=nyjbcRQ-uQ8&list=PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv

Recommended Books

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto
http://incompleteideas.net/book/the-book-2nd.html

Online-Book: Spinning Up in Deep RL! (OpenAI)
https://spinningup.openai.com/en/latest/

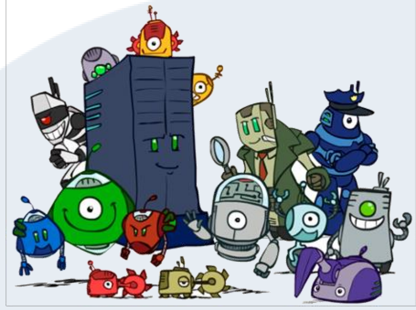
Lectures

- Reinforcement Learning (RL)
- Robot Learning

Online Resources

- Python Atari Gym Environment
  https://www.gymlibrary.dev/environments/atari/
- PySC2 - StarCraft II Learning Environment
  https://github.com/deepmind/pysc2

Summary


Designed by Klein and Abbeel (CS188)

- What is Reinforcement Learning?
- What is the Credit Assignment Problem?
- How does Value Iteration work?
- What types of Reinforcement Learning exists?
- How does Q-Learning, TD-Learning and Policy Search work?
- How does AlphaZero work?

You should be able to:

- describe the reinforcement learning loop
- apply value iteration on a grid world example
- calculate state-value functions and state-action value functions (q-values)
- explain the core concepts of different reinforcement learning approaches
- describe the main components of AlphaZero and their function

Next Week: Reinforcement Learning