Reinforcement Learning: Algorithms and Applications

Dileep Kalathil

Assistant Professor

Department of Electrical and Computer Engineering

RL: A Short Introduction

References:

1. Kevin Murphy, “Machine Learning A Probabilistic Perspective”
2. Prof. David Silver’s Couse, University College, London, Link: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
3. Prof. Sergey Levine’s Couse, UC Berkley, Link: http://rail.eecs.berkeley.edu/deeprlcourse-fa18/
4. Prof. Ben Recht’s (UC Berkeley) notes on RL

What is Reinforcement Learning?

What is Machine Learning?

Three Main Classes of ML

1. Unsupervised Learning
2. Supervised Learning
3. Reinforcement Learning

Unsupervised Learning

- Wikipedia: *“Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of "unlabeled" data (i.e. data that has not been classified or categorized)”*

- Examples: Clustering, Dimensionality Reduction, Matrix Completion, Image Inpainting, Collaborative Filtering

Unsupervised Learning: Clustering

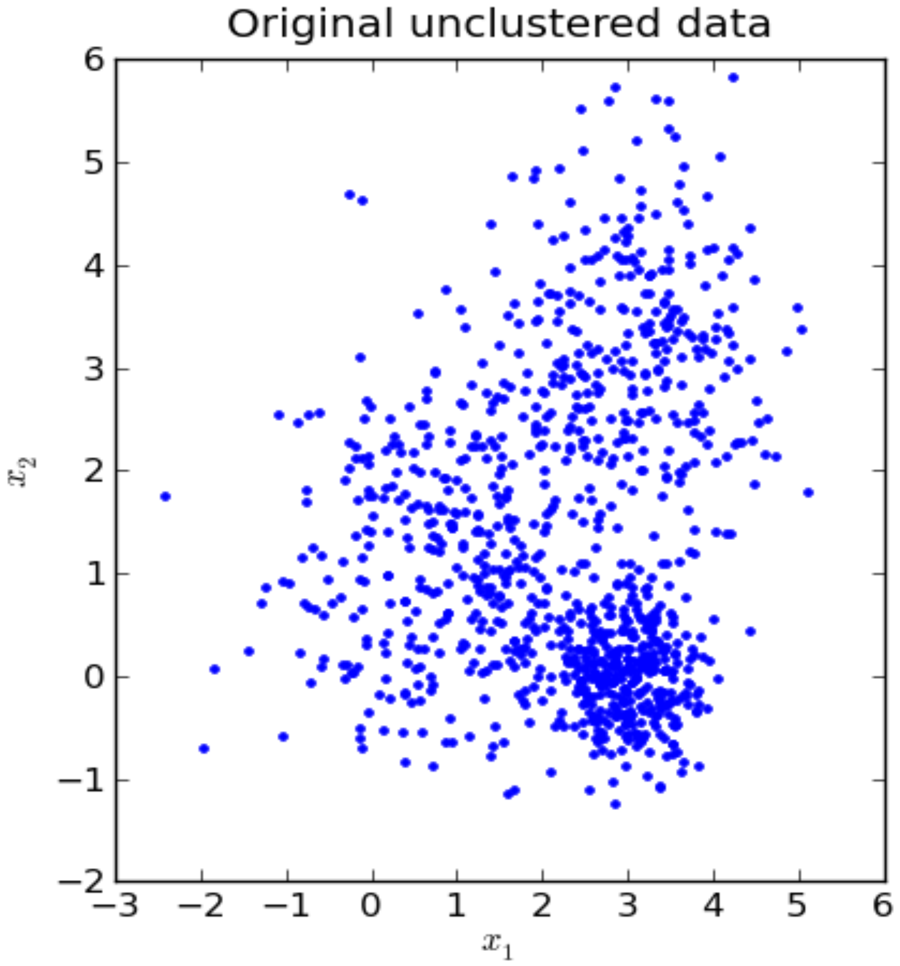


Figure from [1]

Unsupervised Learning: Clustering

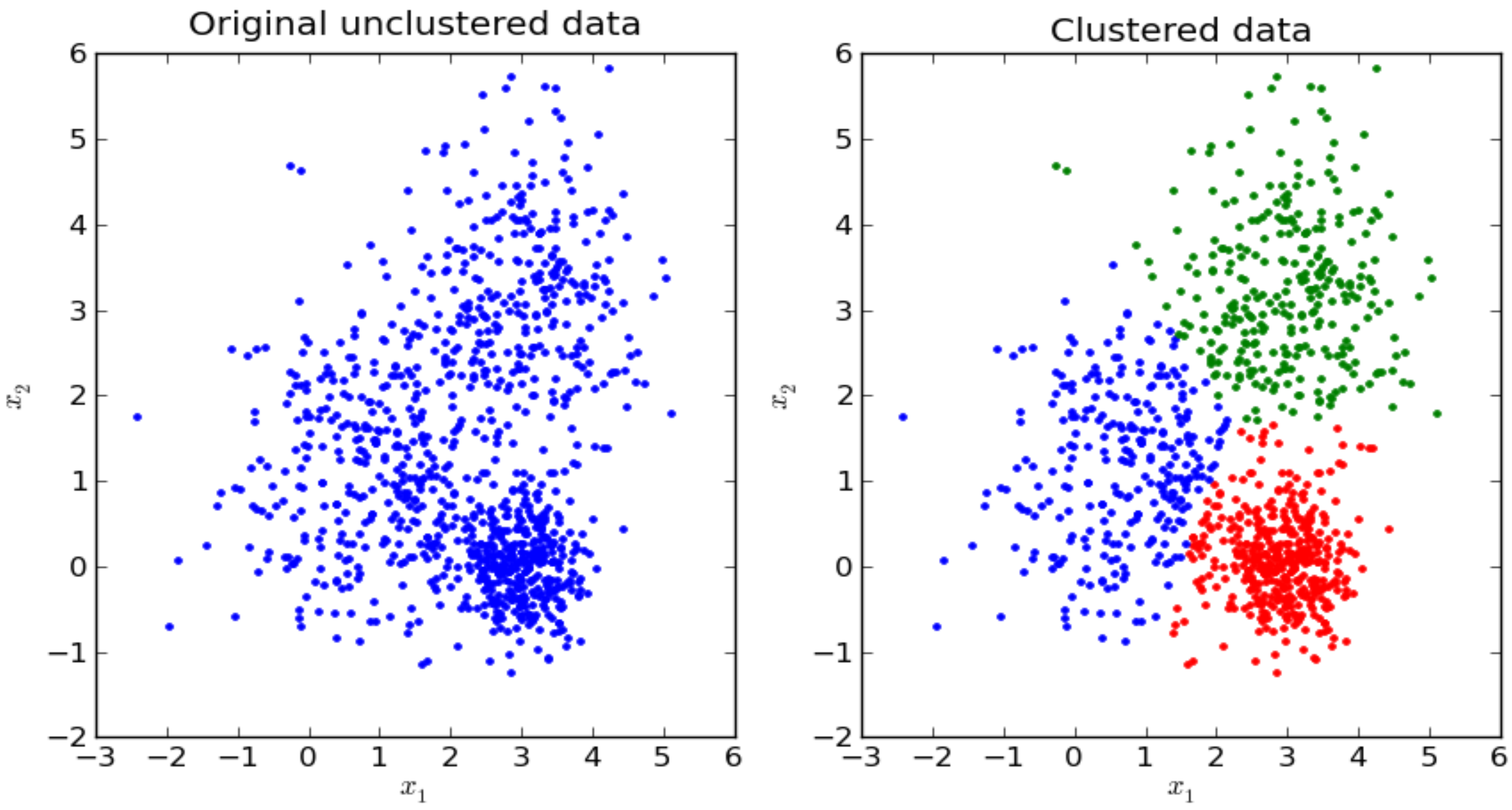


Figure from [1]

Unsupervised Learning

- Wikipedia: *"Unsupervised machine learning is the machine learning task of inferring a function that describes the structure of "unlabeled" data (i.e. data that has not been classified or categorized)"*

- Examples: Clustering, Dimensionality Reduction, Matrix Completion, Image Inpainting, Collaborative Filtering

- Descriptive analytics refers to summarizing data in a way to make it more interpretable

Supervised Learning

- Wikipedia: *"Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs"*

Supervised Learning

- Wikipedia: *"Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs"*
- Examples: Image classification, Handwriting recognition, Email spam filtering, Face recognition, Speech recognition

Supervised Learning: Image Classification

Training Data

Testing Data

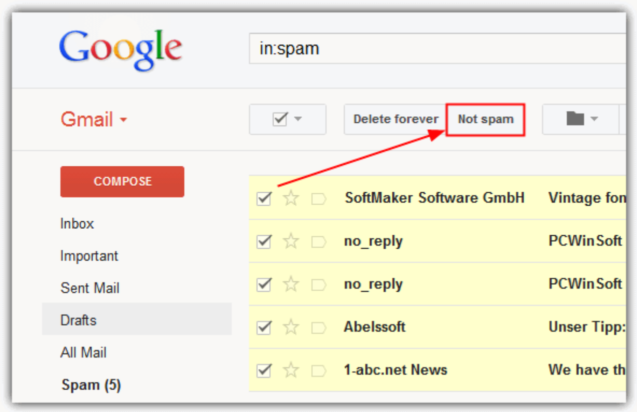
→ Cat

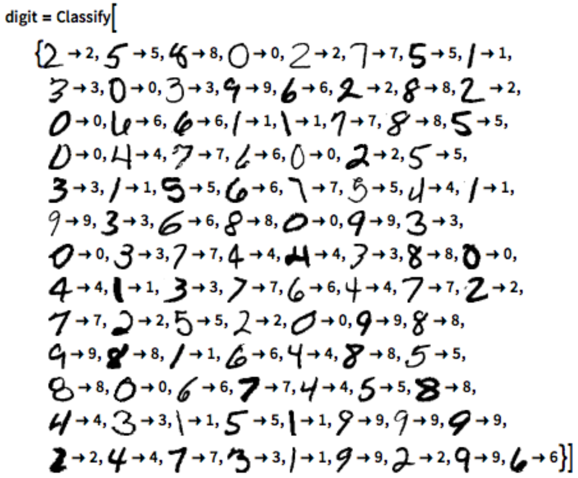
Supervised Learning: Face Recognition

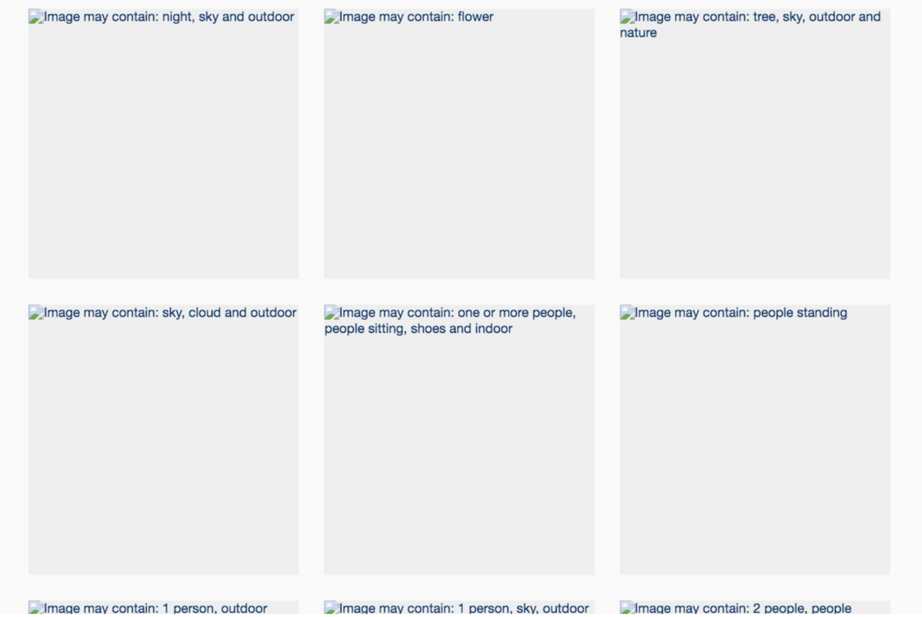
(a) (b)

Figure from [1]

Supervised Learning







Facebook Outage, July, 2019

Supervised Learning

- Wikipedia: *“Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs”*
- Examples: Image classification, Handwriting recognition, Email spam filtering, Face recognition, Speech recognition
- Predictive analytics aims to estimate outcomes from current data

Reinforcement Learning

Wikipedia: *"Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, optimization, multi-agent systems, swarm intelligence, statistics, … "*

Reinforcement Learning

Wikipedia: *"Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, optimization, multi-agent systems, swarm intelligence, statistics, … "*

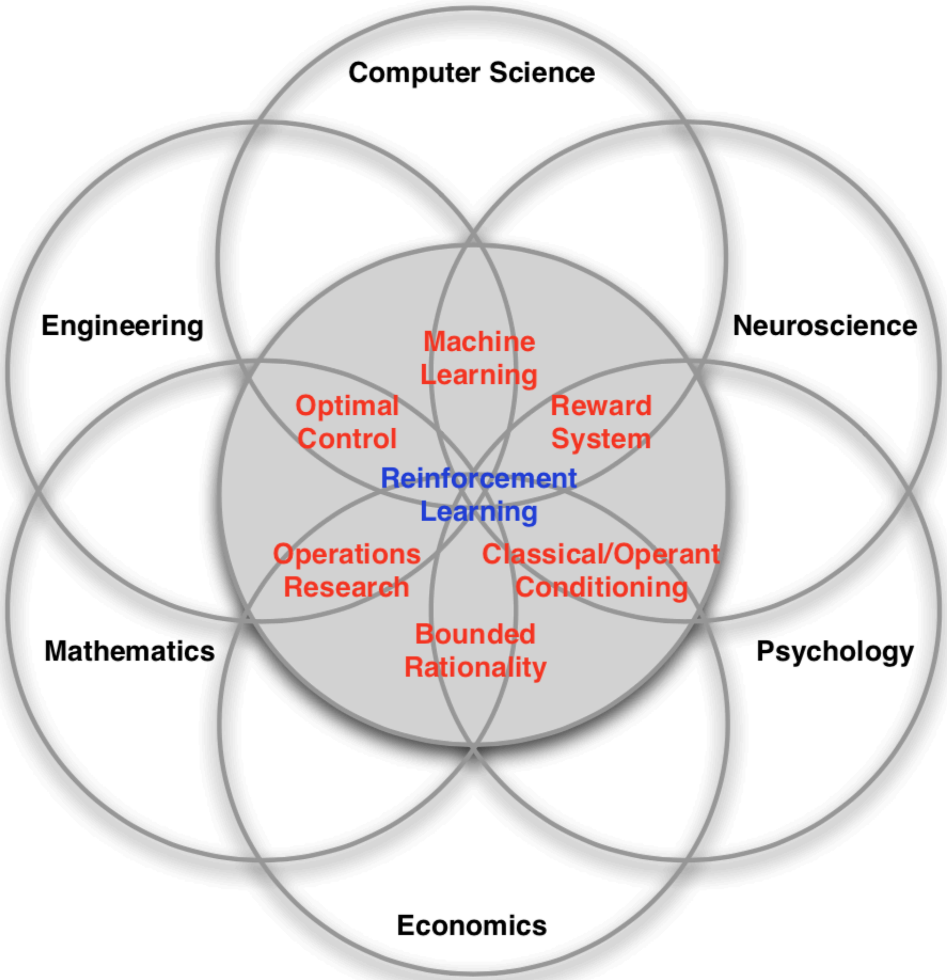
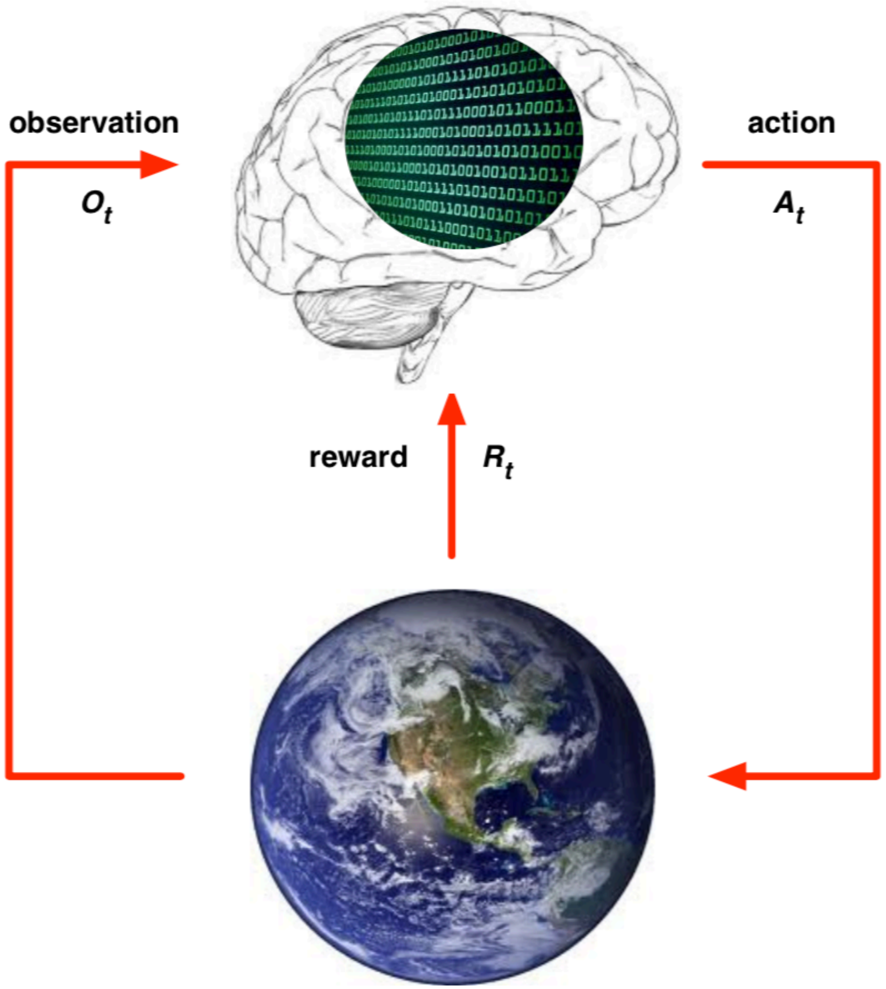


Figure from [2]

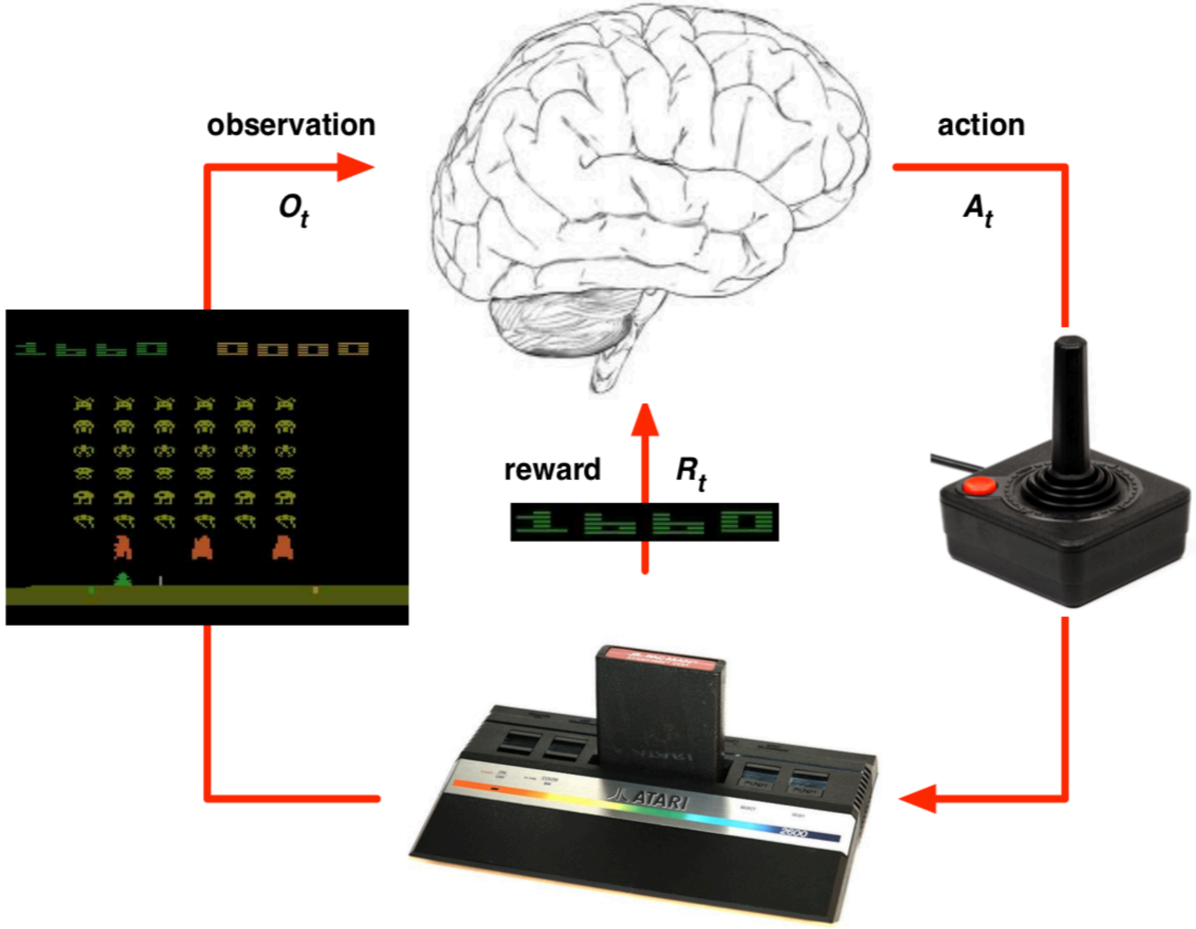
RL: Agent and Environment

- At each time step t the agent:
  - Executes an action A_t
  - Receives reward R_t
  - Receives observation O_{t+1}
- The environment:
  - Receives an action A_t
  - Emits reward R_t
  - Emits reward O_{t+1}
- Time $t \leftarrow t+1$



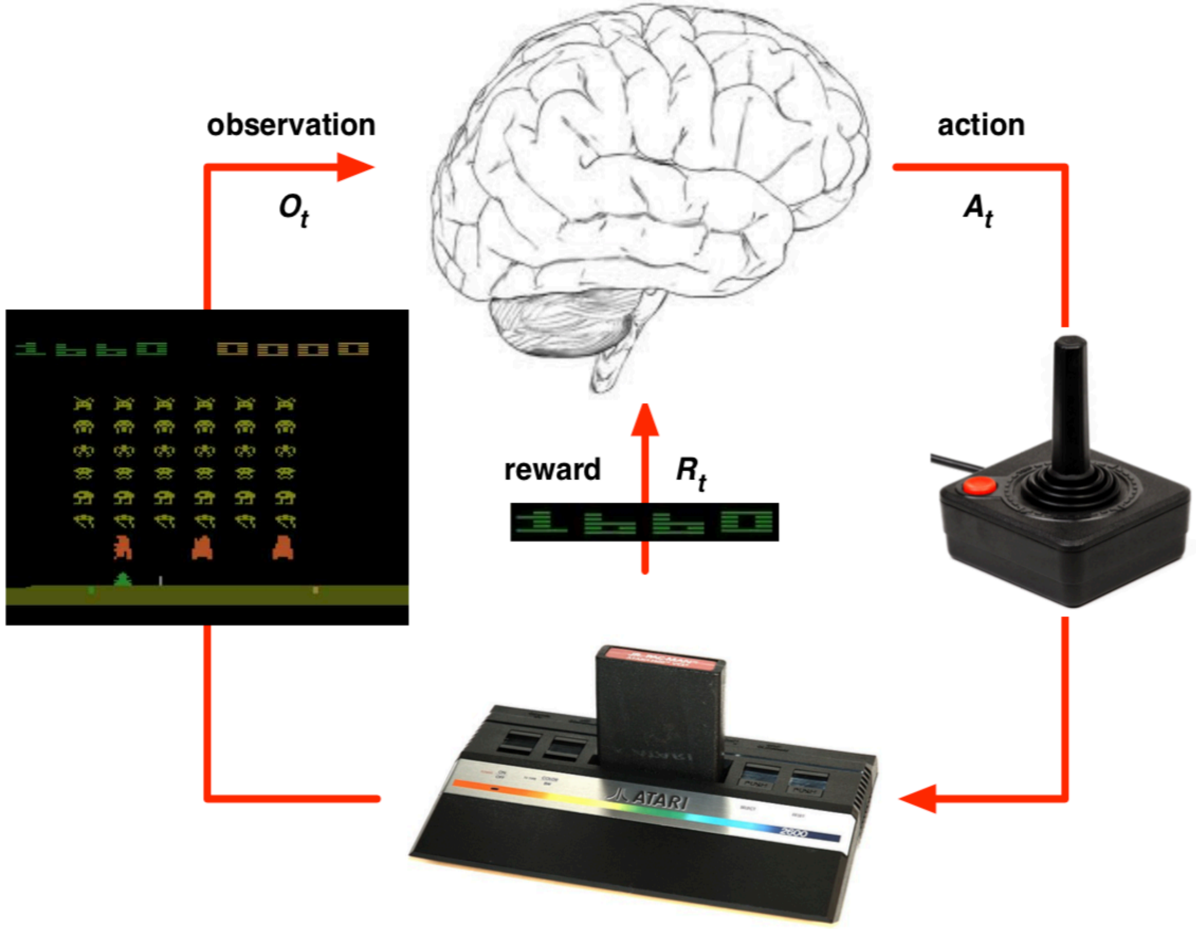
RL: Agent and Environment

- At each time step *t* the agent:
  - Executes an action A_t
  - Receives reward R_t
  - Receives observation O_{t+1}
- The environment:
  - Receives an action A_t
  - Emits reward R_t
  - Emits reward O_{t+1}
- Time $t \leftarrow t+1$



RL: Agent and Environment

- At each time step *t* the agent:
  - Executes an action A_t
  - Receives reward R_t
  - Receives observation O_{t+1}
- The environment:
  - Receives an action A_t
  - Emits reward R_t
  - Emits reward O_{t+1}
- Time $t \leftarrow t+1$



Rules of the game are unknown!

Why RL is Different?

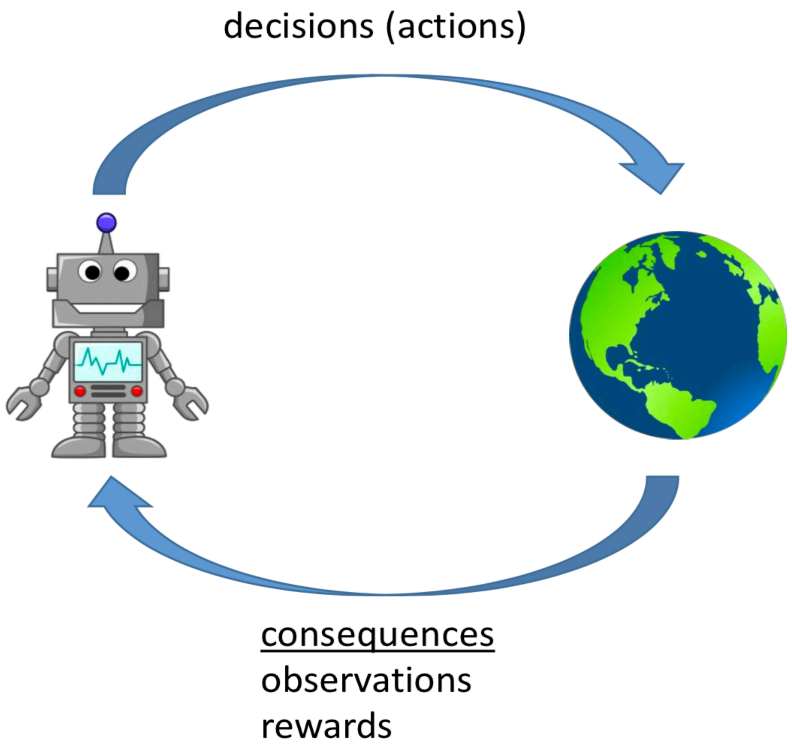
Why RL is Different?

RL: Learning to make a optimal *sequence* of decisions under uncertainty

Why RL is Different?

RL: Learning to make a optimal *sequence* of decisions under uncertainty

- No supervisor, only a reward signal
- Feedback is delayed, not instantaneous
- Sequential decision making
- Actions have long-term consequences
- Non i.i.d. data
- Agent’s actions affect the subsequent data it receives



Helicopter Maneuvers

P. Abbeel, A. Coates, M. Quigley, A. Ng. "An application of reinforcement learning to aerobatic helicopter flight”, *NeurIPS*, 2007.

Robotic Hand Solving Rubik’s Cube

Akkayaet al. "Solving Rubik's Cube with a Robot Hand”, *2019*.

Playing Atari

V. Mnih et al. “Playing Atari with Deep Reinforcement Learning”, *NeurIPS,* 2013.

Robotic Arms

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”, *International Journal of Robotics Research*, 2017

Learning to Walk

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, “Learning to Walk via Deep Reinforcement Learning”, *Robotics: Science and Systems (RSS)*, 2019

Playing Go

- D. Silver, et al. “Mastering the game of Go with deep neural networks and tree search”, *Nature* (2016)

Reinforcement Learning

Wikipedia: *"Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, optimization, multi-agent systems, swarm intelligence, statistics, … "*

Prescriptive analytics guides actions to take in order to guarantee outcomes

RL: Exploration vs Exploitation

- Unlike supervised and unsupervised learning, data is not given before
- Agent learns about the environment by trying things out
  - RL in some way a trial-and-error learning
- Agent should learn a good control policy:
  - From its experiences of the environment but without loosing too much reward along the way

- Online decision-making involves a fundamental choice:
- Exploitation: Make the best decision given current information
- Exploration: Gather more information to make the best decisions
- The optimal long-term strategy may involve sub-optimal short-term decisions

RL: Exploration vs Exploitation

- Unlike supervised and unsupervised learning, data is not given before
- Agent learns about the environment by trying things out
  - RL in some way a trial-and-error learning
- Agent should learn a good control policy:
  - From its experiences of the environment but without loosing too much reward along the way

- Online decision-making involves a fundamental choice:
- Exploitation: Make the best decision given current information
- Exploration: Gather more information to make the best decisions
- The optimal long-term strategy may involve sub-optimal short-term decisions

How to balance exploration and exploitation?

Exploration vs Exploitation: Examples

- Restaurant Selection
  - Exploitation: Go to your favorite restaurant
  - Exploration: Try a new restaurant
- Online advertisements
  - Exploitation: Show the most successful ad
  - Exploration: Show a different ad
- Oil Drilling
  - Exploitation: Drill at best known location
  - Exploration: Drill at new location
- Games
  - Exploitation: Play the move you believe is best
  - Exploration: Play an experimental move

Reinforcement Learning Problem

- Agent doesn’t know how the environment works
- Agent has to interact with the environment to learn
- Agent gets two feedback:
  - It can observe the state of the environment at each step
  - It gets a reward at each step
- Agent has to learn a control policy
  - Algorithm to select action sequentially
- Agent’s objective is to maximize the expected cumulative reward

Markov Decision Processes

References:

1. Some figures for this lecture are taken from UC Berkeley CS188 course, with permission: https://inst.eecs.berkeley.edu/~cs188/fa18/index.html

MDP Example: Grid World

- Agent lives in a grid world

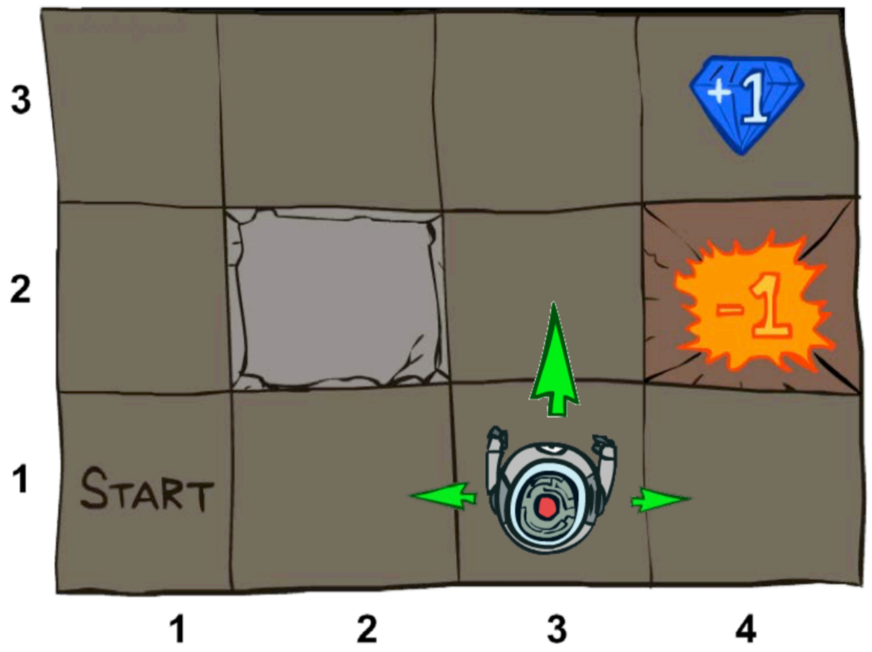


Figure from [1]

MDP Example: Grid World

- Agent lives in a grid world
- This world is non-deterministic
  - Inherent uncertainties in the environment
  - Actions don’t go always go as planned

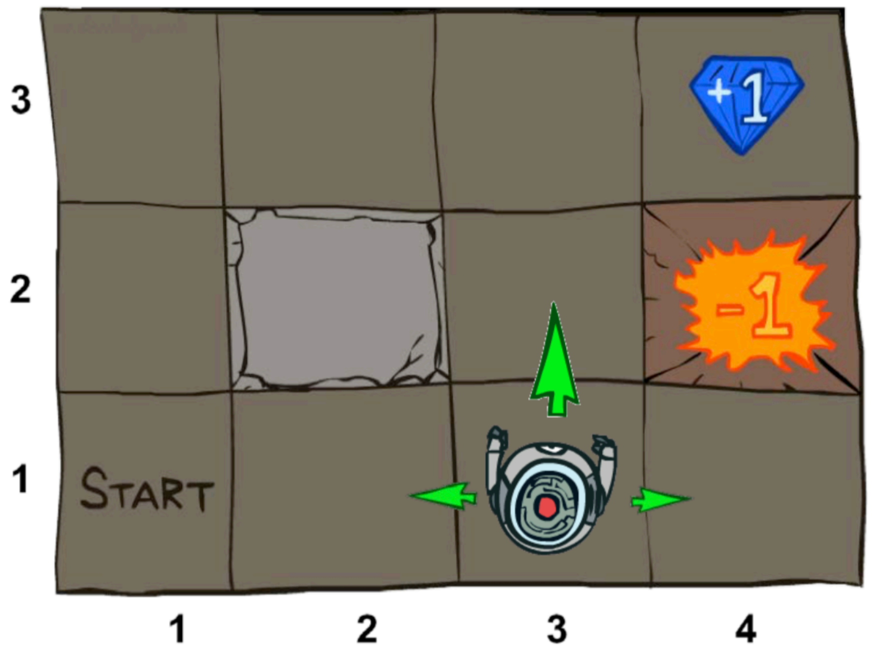


Figure from [1]

MDP Example: Grid World

- Agent lives in a grid world
- This world is non-deterministic
  - Inherent uncertainties in the environment
  - Actions don't go always go as planned

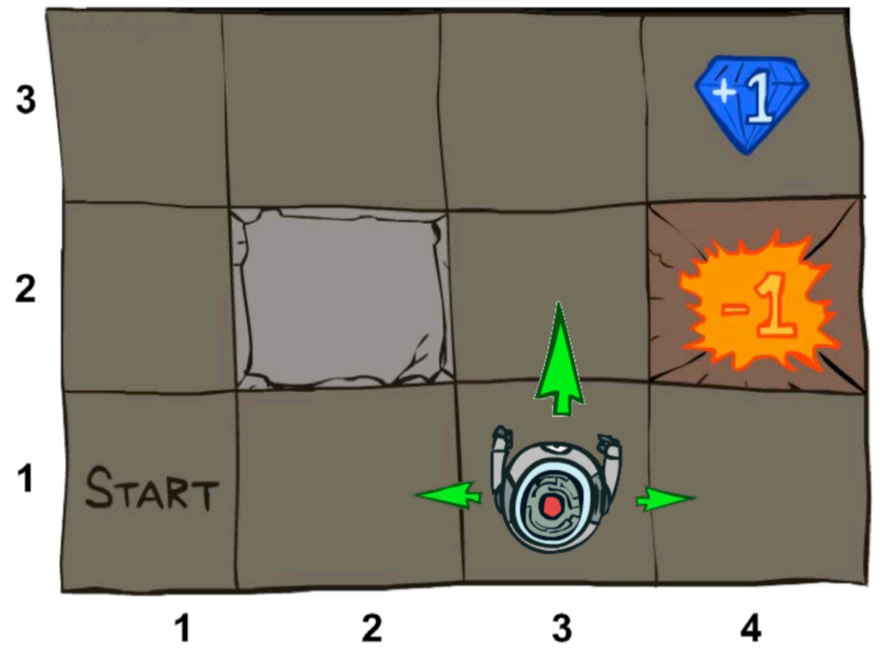
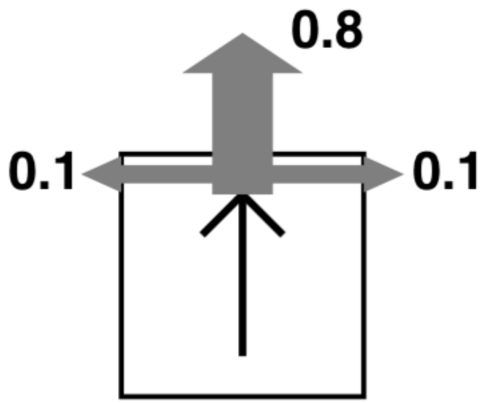


Figure from [1]



MDP Example: Grid World

- Agent lives in a grid world
- This world is non-deterministic
  - Inherent uncertainties in the environment
  - Actions don't go always go as planned
- Agent receives rewards each time step
  - Reward will depend on the current state/action

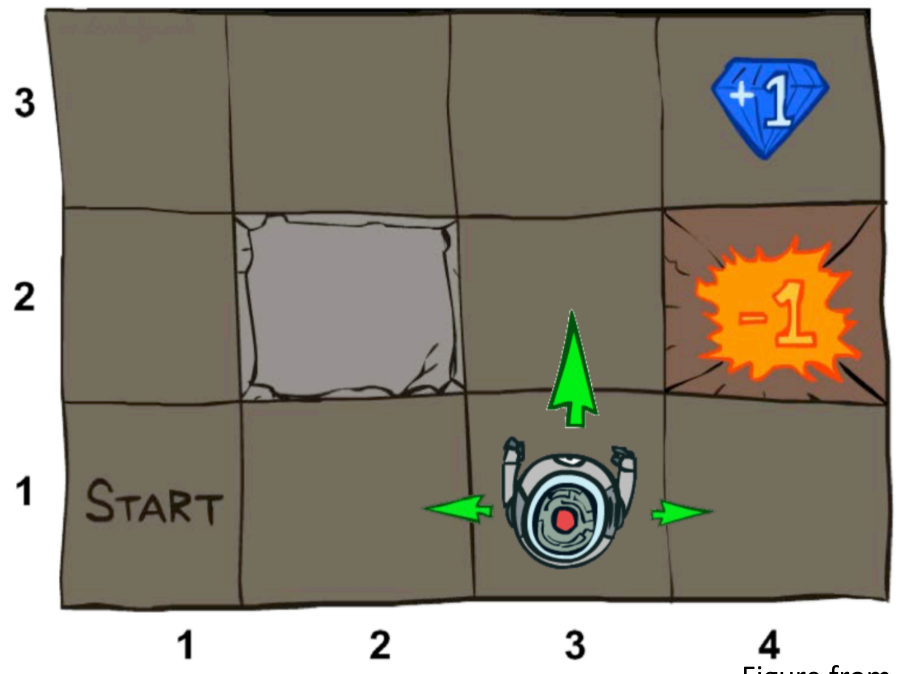


Figure from [1]

Markov Decision Processes (MDP)

- A set of states: x ∈ X
- A set of actions: a ∈ A

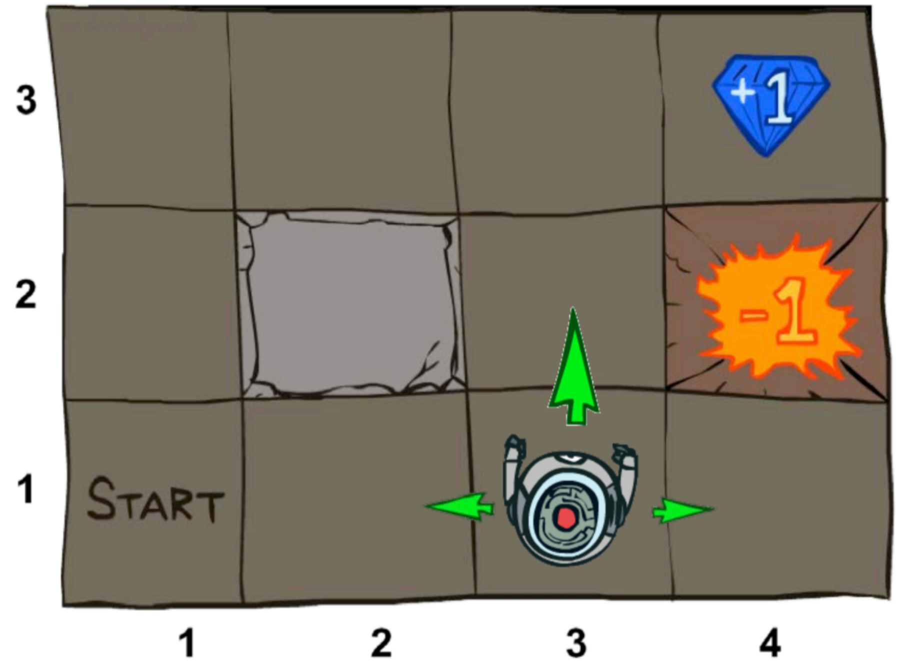


Figure from [1]

Markov Decision Processes (MDP)

- A set of states: x ∈ X
- A set of actions: a ∈ A
- A transition function: P(x’|x, a)
  - Probability moving from x to x’ if action a is taken
  - Also called model or dynamics of the system

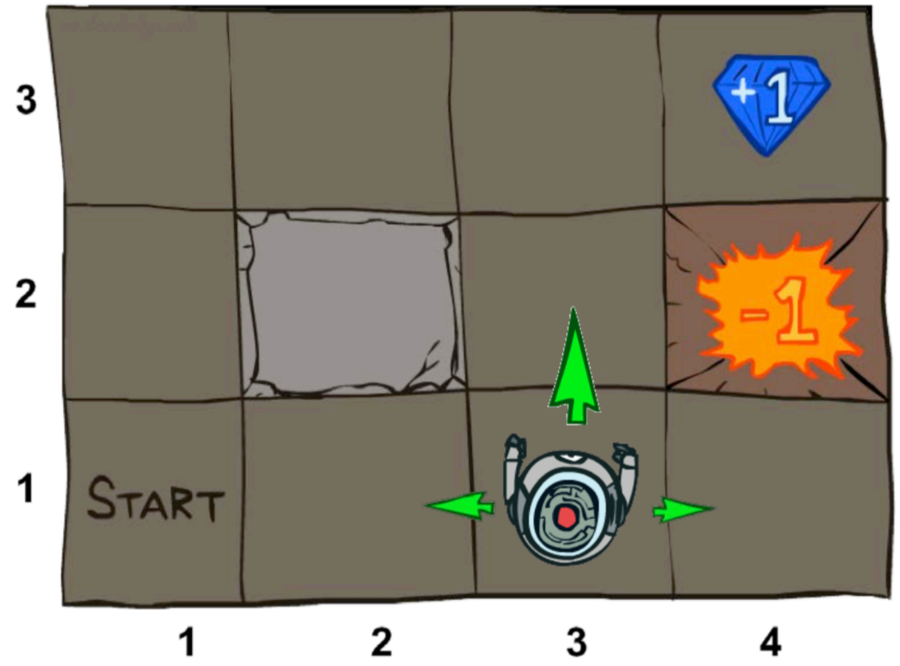


Figure from [1]

Markov Decision Processes (MDP)

- A set of states: $x \in X$
- A set of actions: $a \in A$
- A transition function: $P(x'|x, a)$
  - Probability moving from x to x' if action a is taken
  - Also called model or dynamics of the system
- Reward function: $R(x, a)$ or $R(x, a, x')$

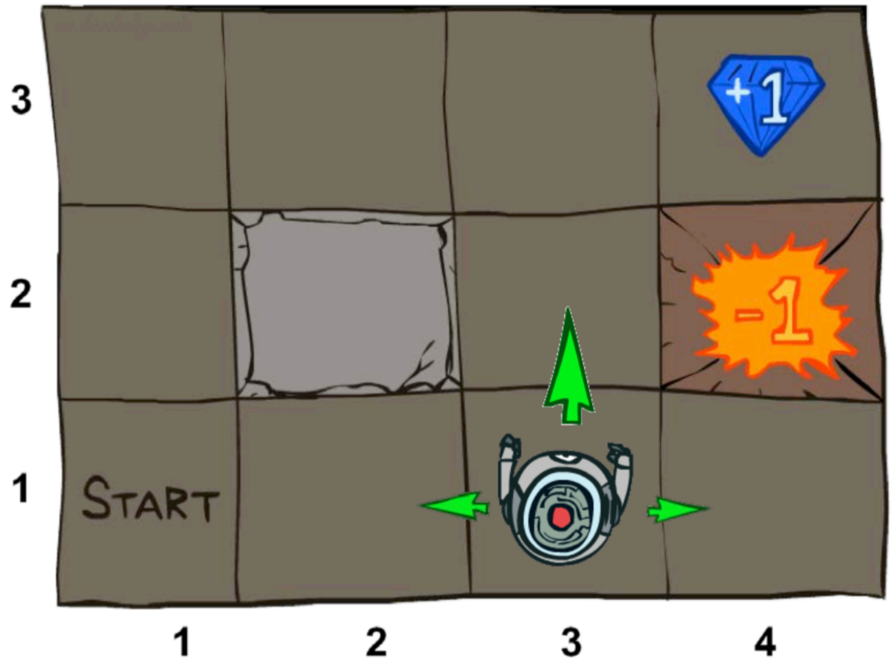


Figure from [1]

Markov Decision Processes (MDP)

- A set of states: $x \in X$
- A set of actions: $a \in A$
- A transition function: $P(x'|x, a)$
  - Probability moving from x to x' if action a is taken
  - Also called model or dynamics of the system
- Reward function: $R(x, a)$ or $R(x, a, x')$
- A start state and/or a terminal state

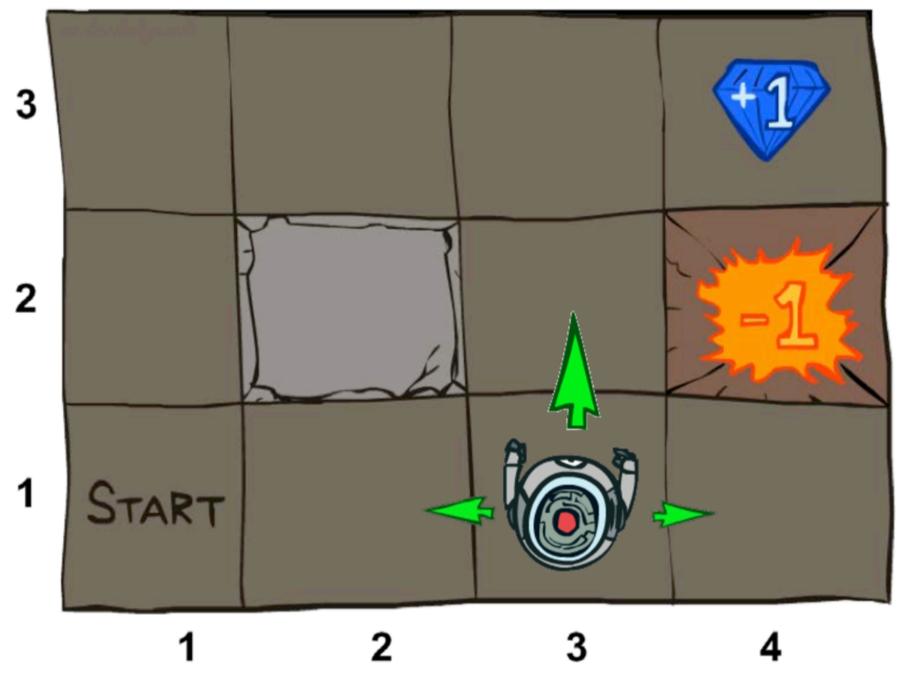


Figure from [1]

Markov Decision Processes (MDP)

- A set of states: x ∈ X
- A set of actions: a ∈ A
- A transition function: P(x’|x, a)
  - Probability moving from x to x’ if action a is taken
  - Also called model or dynamics of the system
- Reward function: R(x, a) or R(x, a, x’)
- A start state and/or a terminal state
- A very useful model for approximating real-world systems!

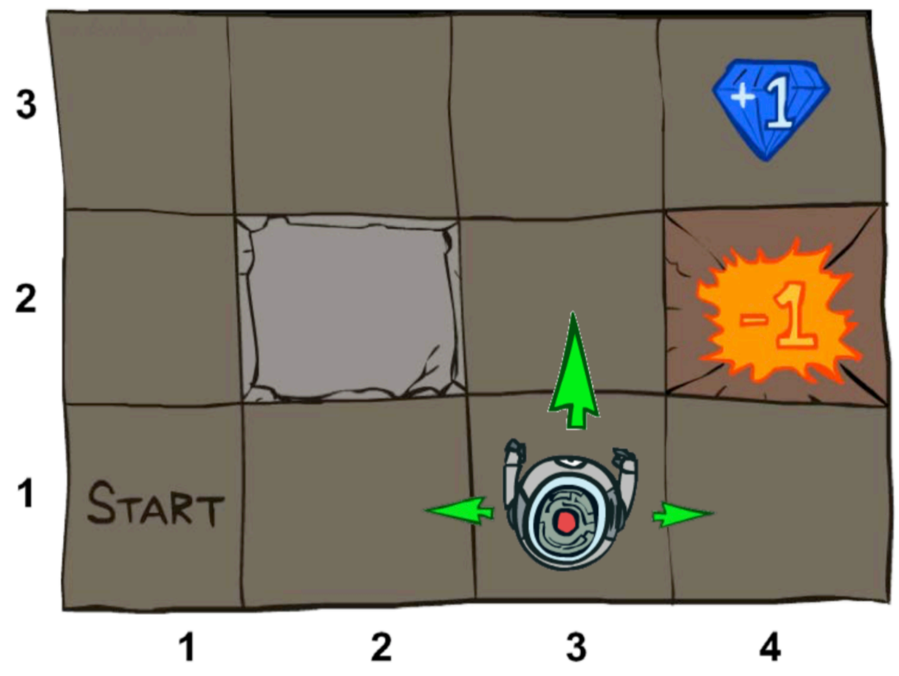


Figure from [1]

Markov Property

- “Markov” generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \dots, X_0 = x_0, A_0 = a_0) =$$

Markov Property

- “Markov” generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

- If you know the current state and current action, history is irrelevant to predict the future states

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

- If you know the current state and current action, history is irrelevant to predict the future states
- Is the world Markov?

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

- If you know the current state and current action, history is irrelevant to predict the future states
- Is the world Markov?
  - A very useful modeling assumption for a large class of real world problems!

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

- If you know the current state and current action, history is irrelevant to predict the future states
- Is the world Markov?
  - A very useful modeling assumption for a large class of real world problems!
- Why is it useful?

Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

- If you know the current state and current action, history is irrelevant to predict the future states
- Is the world Markov?
  - A very useful modeling assumption for a large class of real world problems!
- Why is it useful?
  - Tremendous reduction in memory/computation

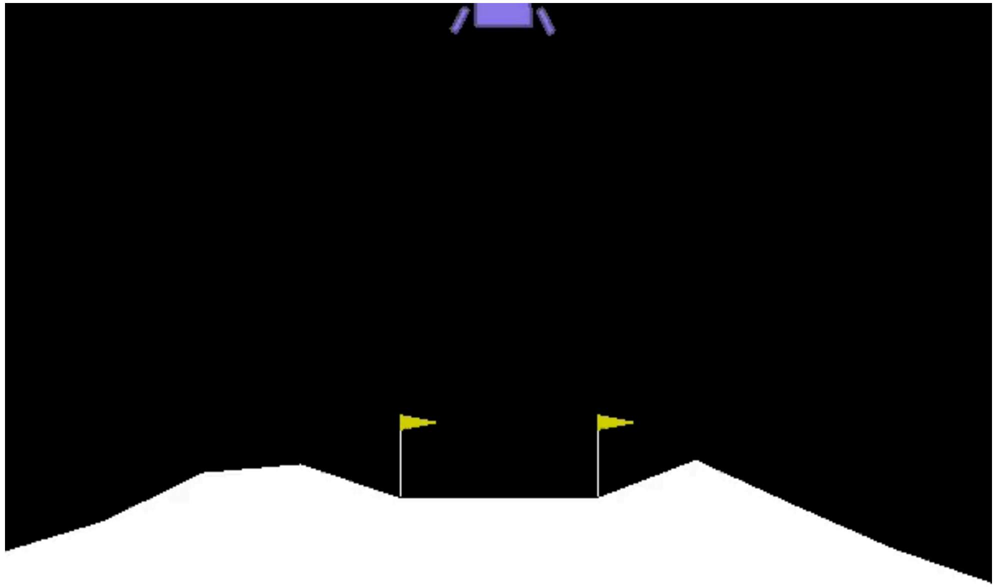
Markov Property

- "Markov" generally means that given the present, the future and the past are independent

$$\mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t, \ldots, X_0 = x_0, A_0 = a_0) = \mathbb{P}(X_{t+1} = x_{t+1} | X_t = x_t, A_t = a_t)$$
$$= P(x_{t+1} | x_t, a_t)$$

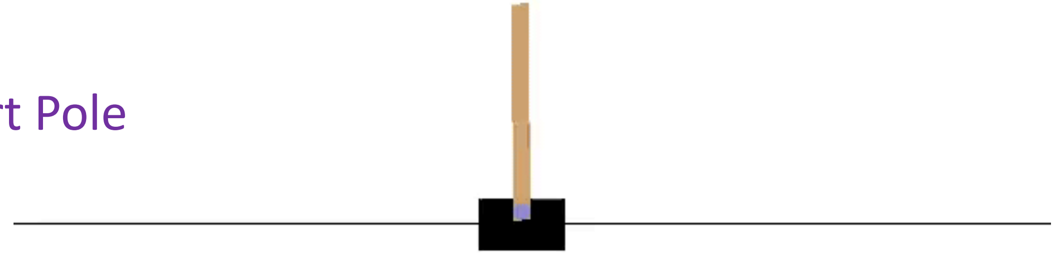
- If you know the current state and current action, history is irrelevant to predict the future states
- Is the world Markov?
  - A very useful modeling assumption for a large class of real world problems!
- Why is it useful?
  - Tremendous reduction in memory/computation
  - History explodes with time. But no need to store the entire history!

MDP Examples



Lunar Lander

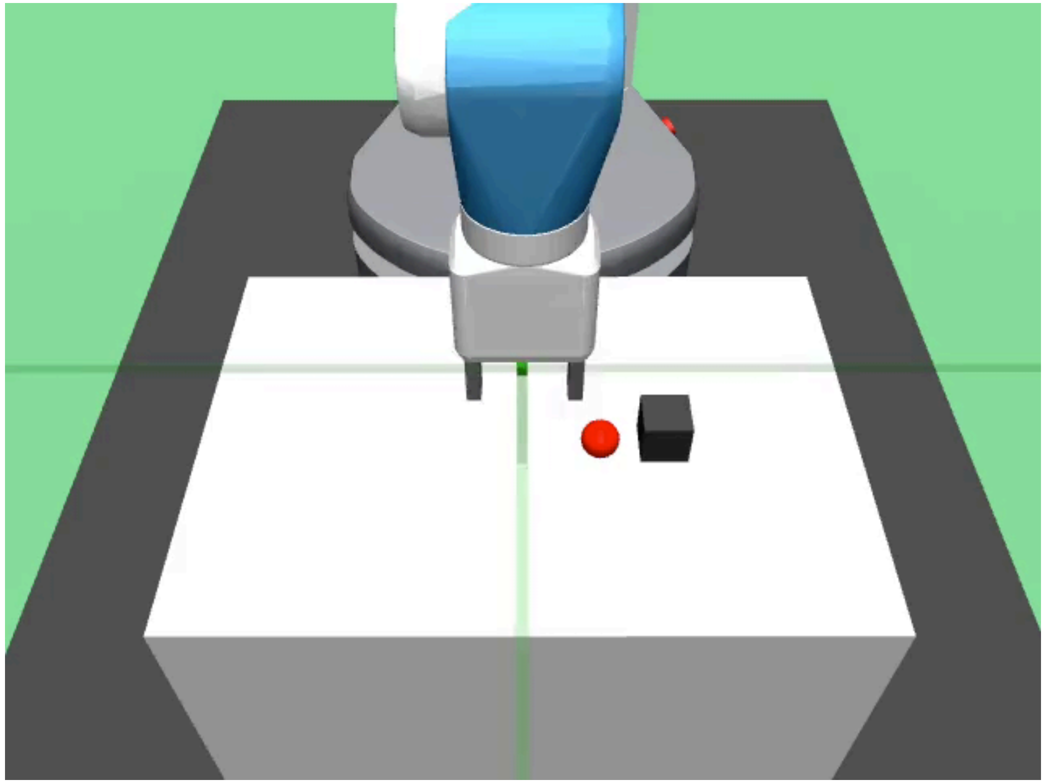
Inverted Pendulum



Cart Pole

Videos are from OpenAI Gym

MDP Examples



Robotic Arm 1

Robotic Arm 2

Videos are from OpenAI Gym

Markov Decision Processes

Definition. A Markov Decision Process (MDP) is a tuple $(\mathcal{X}, \mathcal{A}, P, R)$, where,

- $\mathcal{X}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- P is a transition probability matrix, $P(x'|x, a) = \mathbb{P}(x_{t+1} = x'|x_t = x, a_t = a)$
- R is a reward function, $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

Control Policy

- Which action to take in each state?

Control Policy

- Which action to take in each state?
- A control policy specifies the action to take given the current state
  - Can be deterministic or stochastic

$$\pi(a|x) = \mathbb{P}(a_t = a | x_t = x)$$

  - Conditional probability of taking action *a* given the state *x*

Control Policy

- Which action to take in each state?
- A control policy specifies the action to take given the current state
  - Can be deterministic or stochastic

$$\pi(a|x) = \mathbb{P}(a_t = a | x_t = x)$$

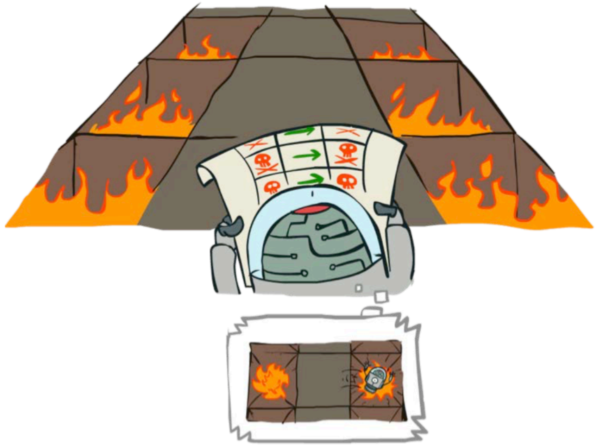
  - Conditional probability of taking action *a* given the state *x*

- A policy fully defines the behavior of an agent

Value of a Policy

Value of a Policy

Always Go Right



Always Go Forward

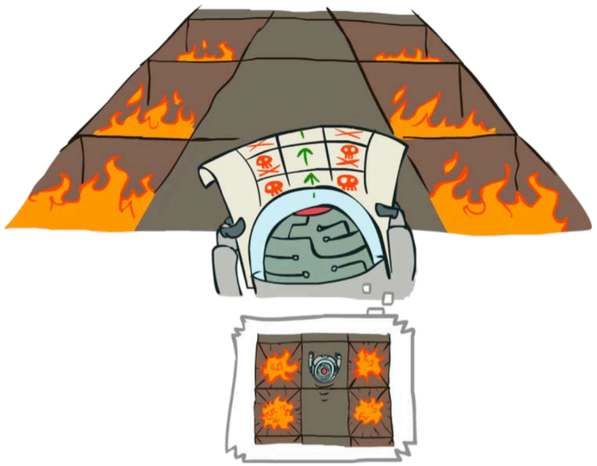


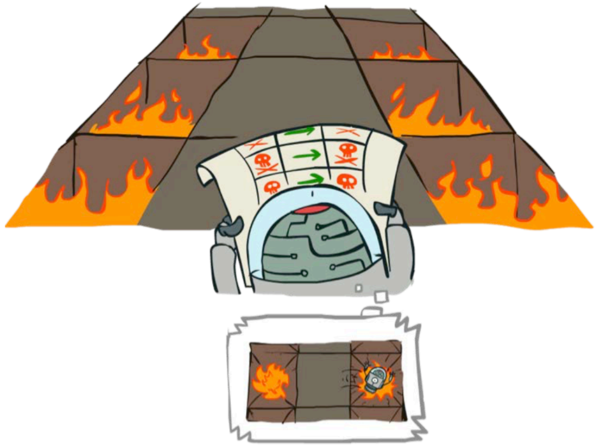
Figure from [1]

Value of a Policy

- Value of a policy evaluated at state *x* is the expected cumulative (discounted) rewards obtained by taking action according that policy, starting from *x*

$$V_\pi(x) = \mathbb{E}\left[R(x_0, a_0) + \gamma R(x_1, a_1) + \gamma^2 R(x_2, a_2) + \ldots + \gamma^t R(x_t, a_t) + \ldots \;\middle|\, x_0 = x, a_t \sim \pi(\cdot|x_t)\right]$$

Always Go Right



Always Go Forward

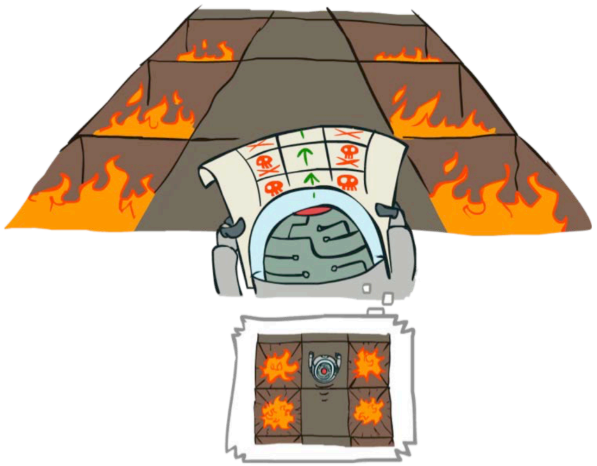


Figure from [1]

Value of a Policy

- Value of a policy evaluated at state *x* is the expected cumulative (discounted) rewards obtained by taking action according that policy, starting from *x*

$$V_\pi(x) = \mathbb{E}\left[R(x_0, a_0) + \gamma R(x_1, a_1) + \gamma^2 R(x_2, a_2) + \ldots + \gamma^t R(x_t, a_t) + \ldots \;\middle|\; x_0 = x, a_t \sim \pi(\cdot|x_t)\right]$$

Always Go Right

Always Go Forward

Figure from [1]

MDP Questions

1. How to evaluate a policy?

   Given a policy π, how to compute the value function V_π of that policy?

   $$V_\pi(x) = \mathbb{E}\left[R(x_0, a_0) + \gamma R(x_1, a_1) + \gamma^2 R(x_2, a_2) + \ldots + \gamma^t R(x_t, a_t) + \ldots \;\middle|\; x_0 = x, a_t \sim \pi(\cdot|x_t)\right]$$

MDP Questions

1. How to evaluate a policy?

   Given a policy π, how to compute the value function V_π of that policy?

   $$V_\pi(x) = \mathbb{E}\left[R(x_0, a_0) + \gamma R(x_1, a_1) + \gamma^2 R(x_2, a_2) + \ldots + \gamma^t R(x_t, a_t) + \ldots \;\middle|\; x_0 = x, a_t \sim \pi(\cdot|x_t)\right]$$

2. How to compute the optimal value function V^*?

   $$V^*(x) = \max_\pi V_\pi(x)$$

MDP Questions

1. How to evaluate a policy?

   Given a policy π, how to compute the value function V_π of that policy?

   $$V_\pi(x) = \mathbb{E}\left[R(x_0, a_0) + \gamma R(x_1, a_1) + \gamma^2 R(x_2, a_2) + \ldots + \gamma^t R(x_t, a_t) + \ldots \;\middle|\; x_0 = x, a_t \sim \pi(\cdot|x_t)\right]$$

2. How to compute the optimal value function V^*?

   $$V^*(x) = \max_\pi V_\pi(x)$$

3. How to compute the optimal policy π^*?

   $$\pi^*(x) = \arg\max_\pi V_\pi(x)$$

Bellman Optimality Equation

Let V^* be the optimal value function. Then V^* satisfies the equation

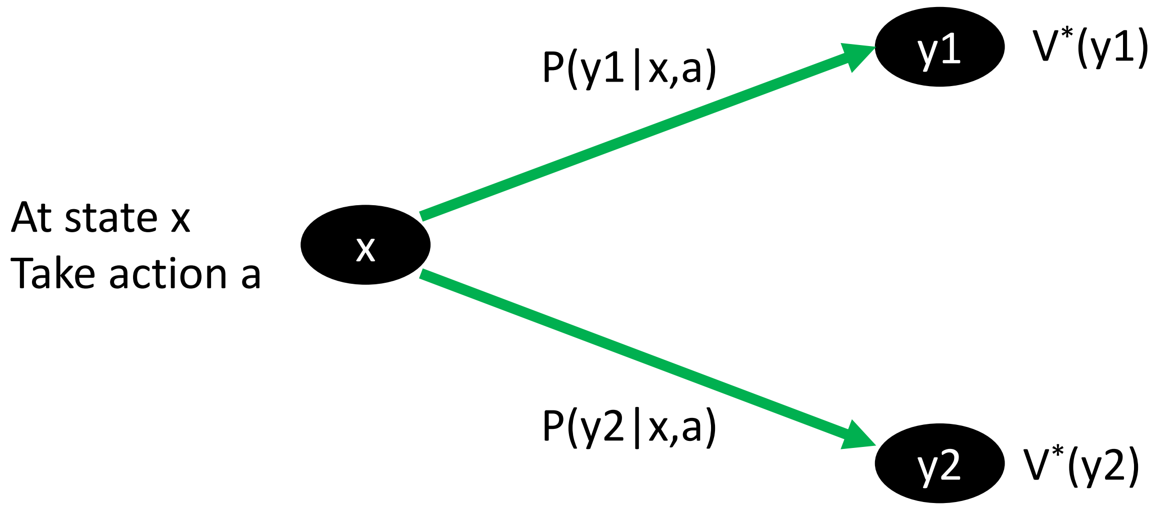
$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x, a) + \gamma \sum_{y \in \mathcal{X}} P(y|x, a) V^*(y) \right)$$

Bellman Optimality Equation

Let V^* be the optimal value function. Then V^* satisfies the equation

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right)$$

- Intuition: Suppose V^* is the optimal value function

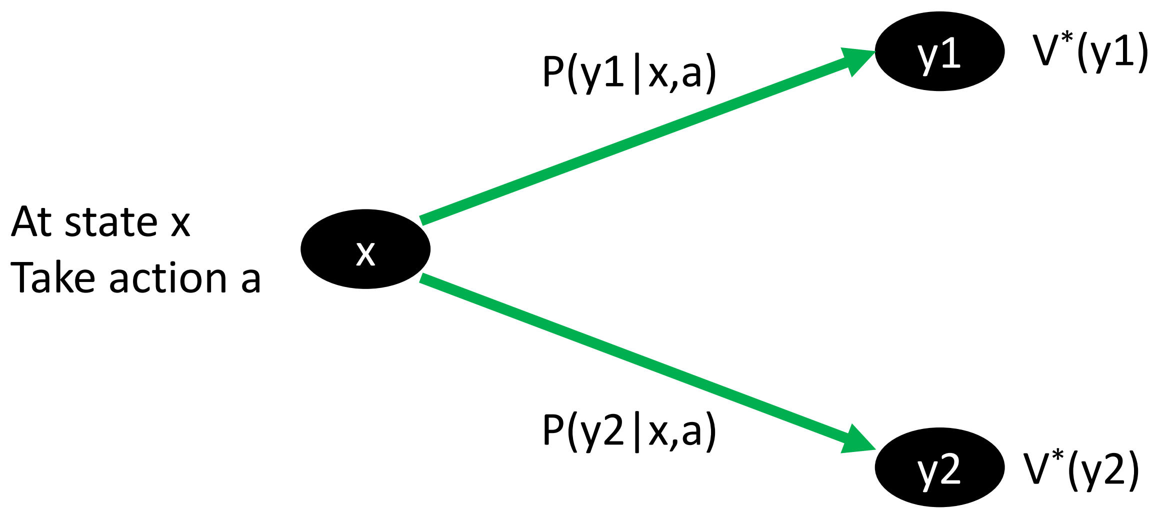


Bellman Optimality Equation

Let V^* be the optimal value function. Then V^* satisfies the equation

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right)$$

- Intuition: Suppose V^* is the optimal value function



Value at state x by taking action a is

$$\left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right)$$

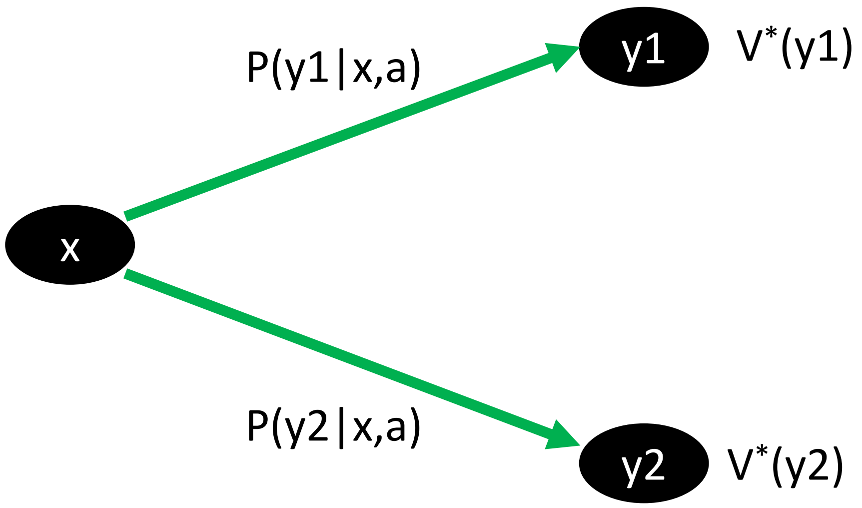
Bellman Optimality Equation

Let V^* be the optimal value function. Then V^* satisfies the equation

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right)$$

- Intuition: Suppose V^* is the optimal value function

At state x
Take action a



Value at state x by taking action a is

$$\left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y) \right)$$

But we want to take the best action

Bellman Operator and Value Iteration

Define the mapping $T : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ as

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator and Value Iteration

Define the mapping $T : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ as

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

Bellman Operator and Value Iteration

Define the mapping $T : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ as

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y) \right)$$

Bellman Operator

Define the iteration, $V_{k+1} = TV_k$

This will give a sequence $V_0, V_1, V_2, \ldots$

Bellman Operator and Value Iteration

Define the mapping $T : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ as

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

Define the iteration, $V_{k+1} = TV_k$

This will give a sequence $V_0, V_1, V_2, \ldots$

Value Iteration

Bellman Operator and Value Iteration

Define the mapping $T : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$ as

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

Define the iteration, $V_{k+1} = TV_k$

This will give a sequence $V_0, V_1, V_2, \ldots$

Value Iteration

- Will the VI converge?
- Will it converge to V^*?
- Is V^* unique?
- How fast does it converge?
- How to get π^* from V^*?

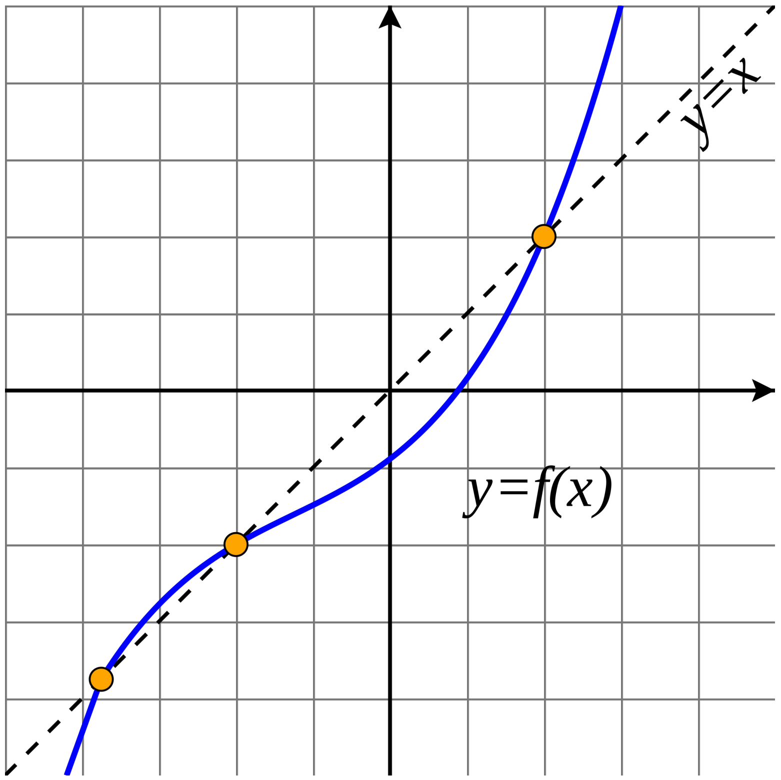
Fixed Point of the Bellman Operator

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point)**.** *Let* $f : \mathcal{X} \to \mathcal{X}$*.* x^* *is a fixed point of* f*, if* $f(x^*) = x^*$

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point)**.** *Let* $f : \mathcal{X} \to \mathcal{X}$. x^* *is a fixed point of* f, *if* $f(x^*) = x^*$



Fixed Point of the Bellman Operator

Definition 1 (Fixed Point). *Let* $f : \mathcal{X} \to \mathcal{X}$. x^* *is a fixed point of* f, *if* $f(x^*) = x^*$

V^* is a fixed point of the Bellman Operator T

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point). *Let* $f : \mathcal{X} \to \mathcal{X}$. x^* *is a fixed point of* f, *if* $f(x^*) = x^*$

V^* is a fixed point of the Bellman Operator T

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point)**.** *Let* $f : \mathcal{X} \to \mathcal{X}$. x^* *is a fixed point of* f, *if* $f(x^*) = x^*$

V* is a fixed point of the Bellman Operator T

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V^*(y)\right)$$

Bellman Optimality Equation

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point). *Let $f : \mathcal{X} \to \mathcal{X}$. x^* is a fixed point of f, if $f(x^*) = x^*$*

V^* is a fixed point of the Bellman Operator T $\quad V^* = TV^*$

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V^*(y)\right)$$

Bellman Optimality Equation

Fixed Point of the Bellman Operator

Definition 1 (Fixed Point). *Let* $f : \mathcal{X} \to \mathcal{X}$. x^* *is a fixed point of* f, *if* $f(x^*) = x^*$

V^* is a fixed point of the Bellman Operator T $\quad V^* = TV^*$

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

Bellman Operator

$$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V^*(y)\right)$$

Bellman Optimality Equation

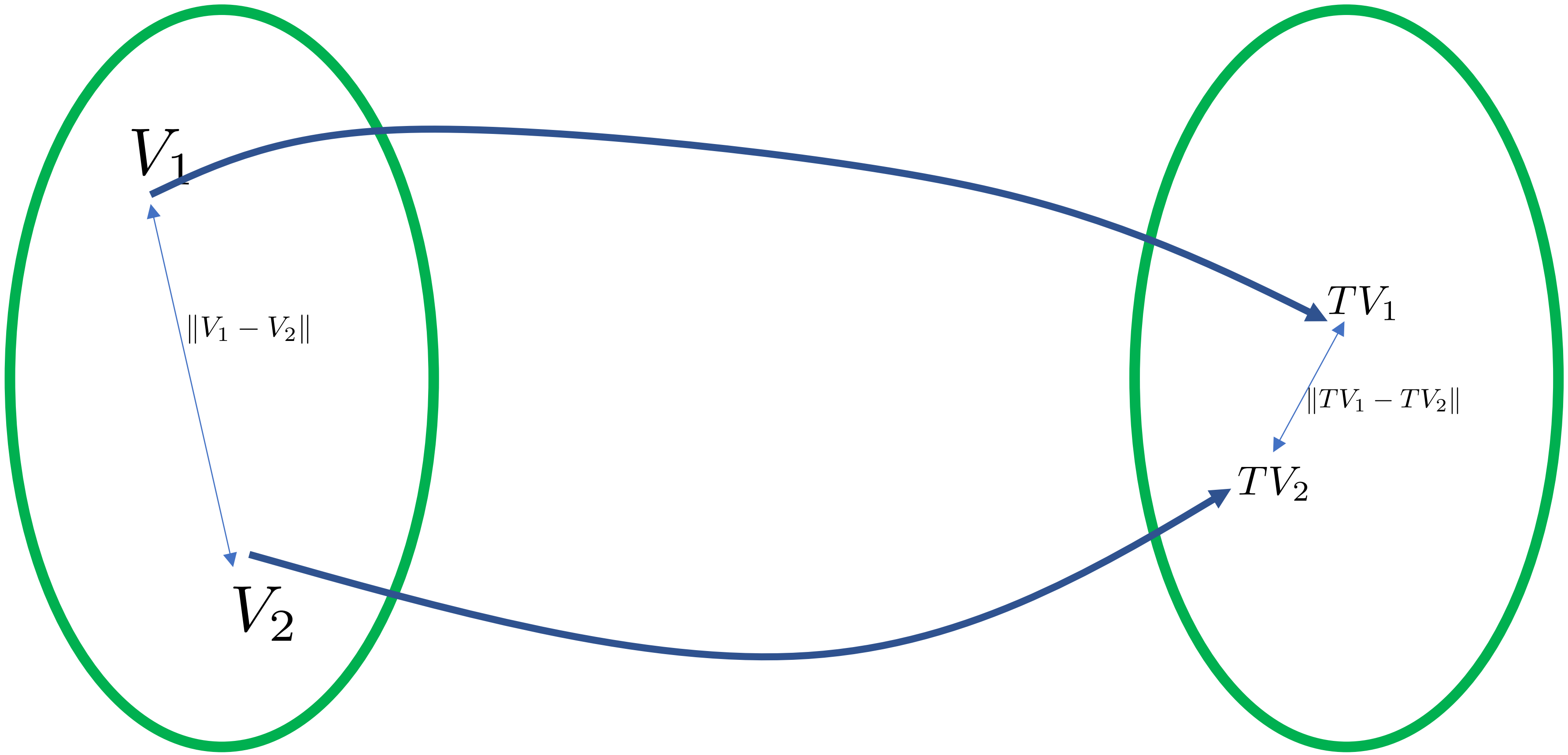
- Computing optimal value function is equivalent to computing the fixed point of the Bellman Operator

Contraction Mapping

Proposition 2 (Contraction Mapping)**.** *T is a contraction mapping. For any $V_1, V_2 \in \mathcal{R}^{n_x}$, $\|TV_1 - TV_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$*

Contraction Mapping

Proposition 2 (Contraction Mapping). *T is a contraction mapping. For any $V_1, V_2 \in \mathcal{R}^{n_x}$, $\|TV_1 - TV_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$*



Convergence of Value Iteration

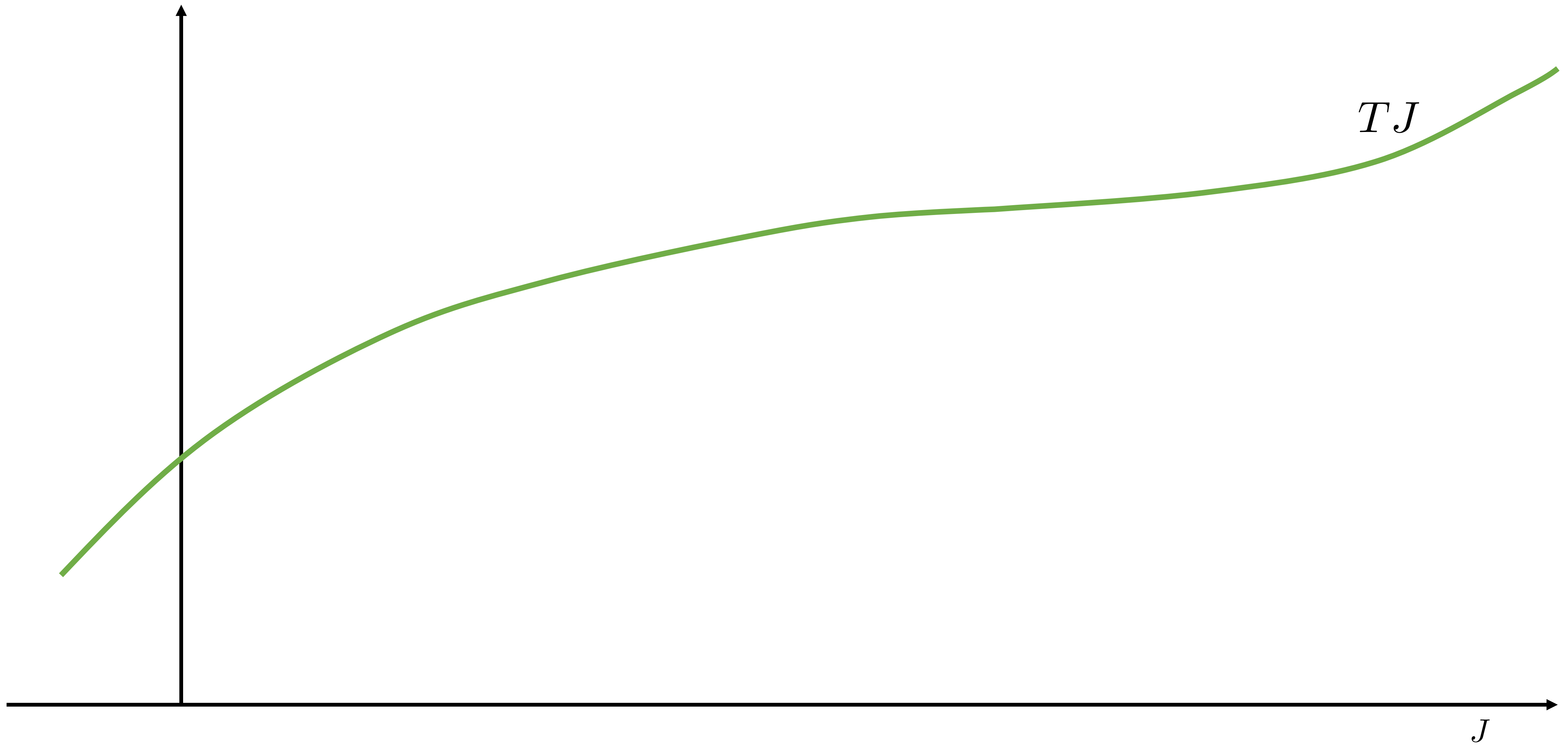
$$\|V_{k+1} - V^*\| \;\leq\; \|TV_k - TV^*\| \;\leq\; \gamma\;\|V_k - V^*\|$$

One step contraction

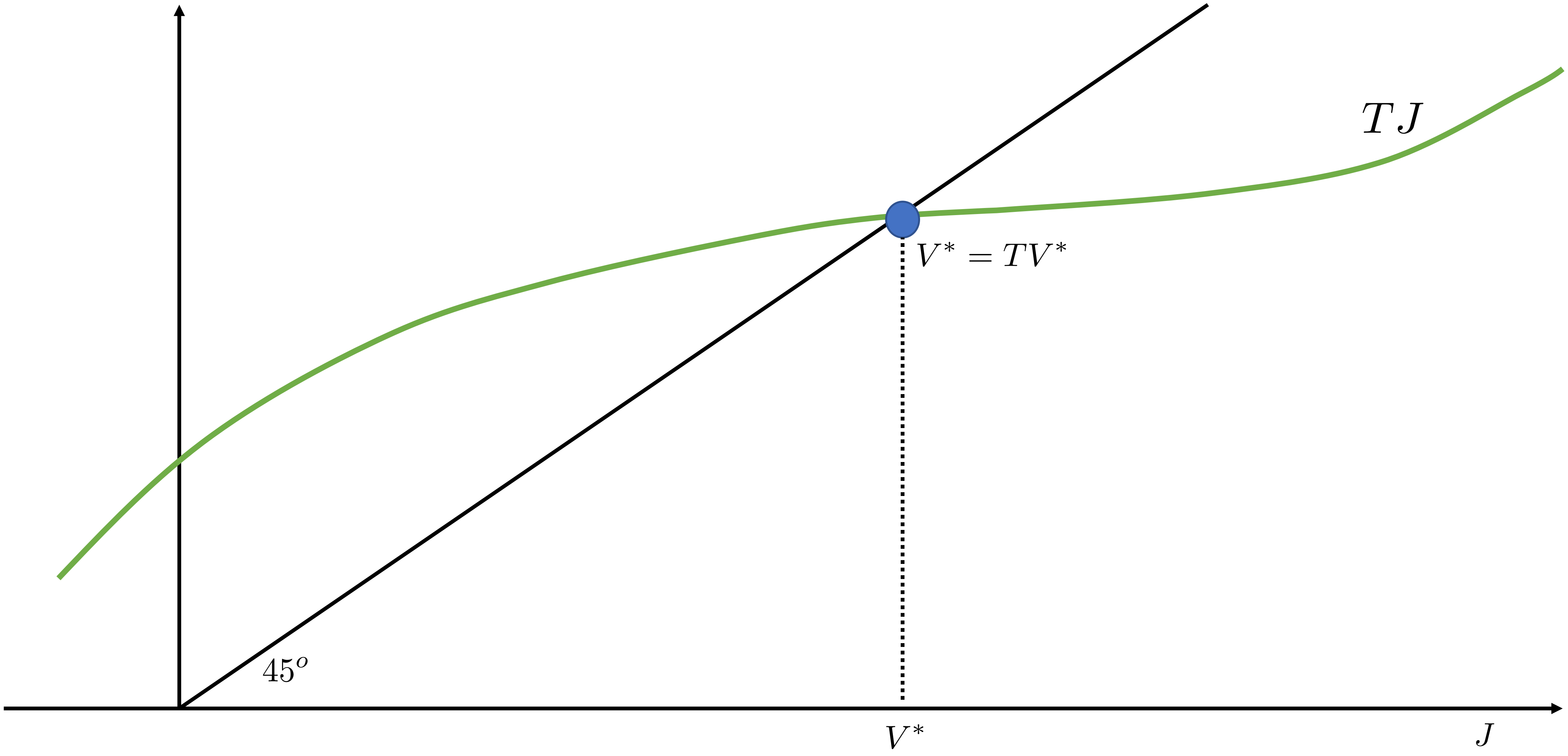
$$\|V_{k+1} - V^*\| \;\leq\; \gamma^{k+1}\;\|V_0 - V^*\|$$

(k+1) step contraction

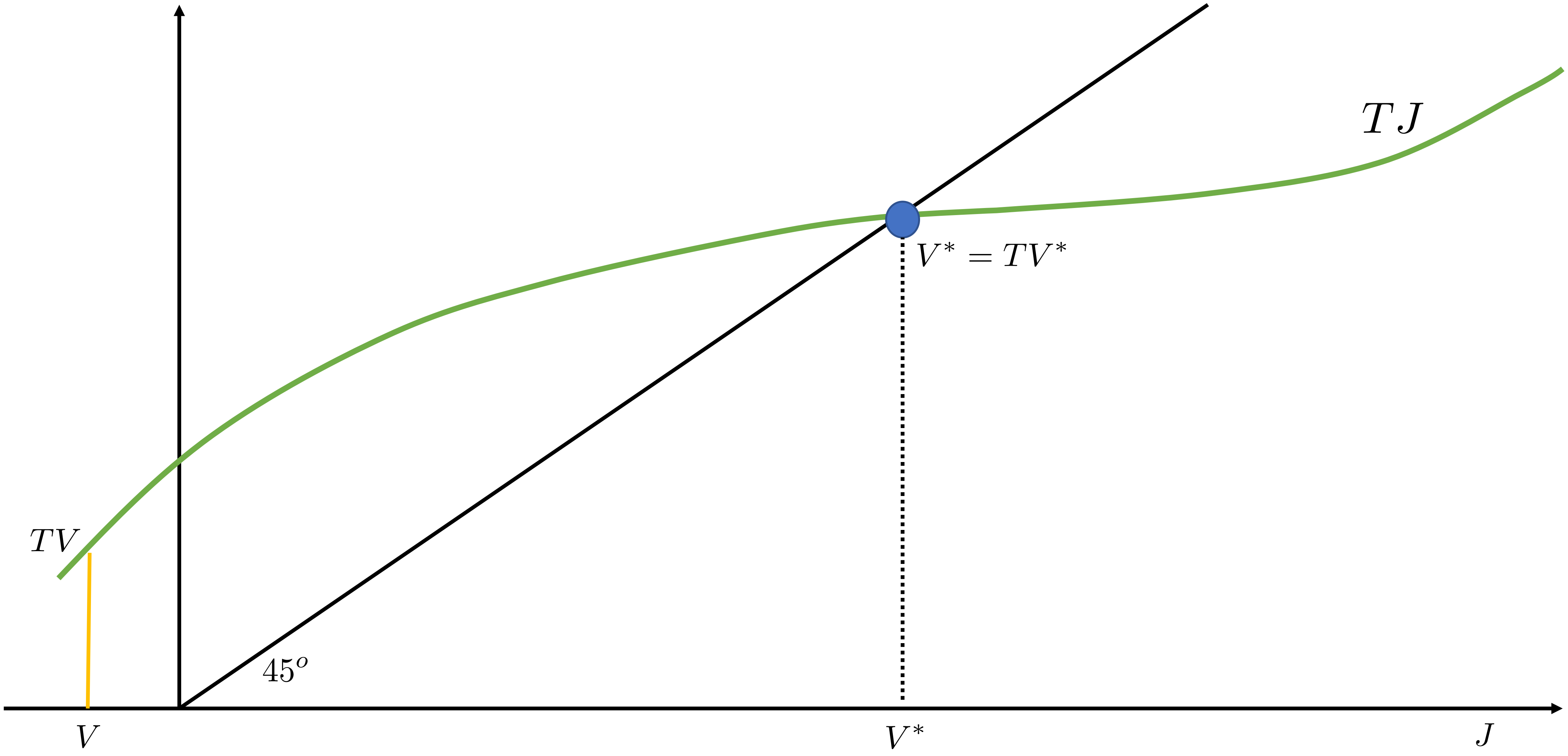
Contraction, Fixed Point, Convergence



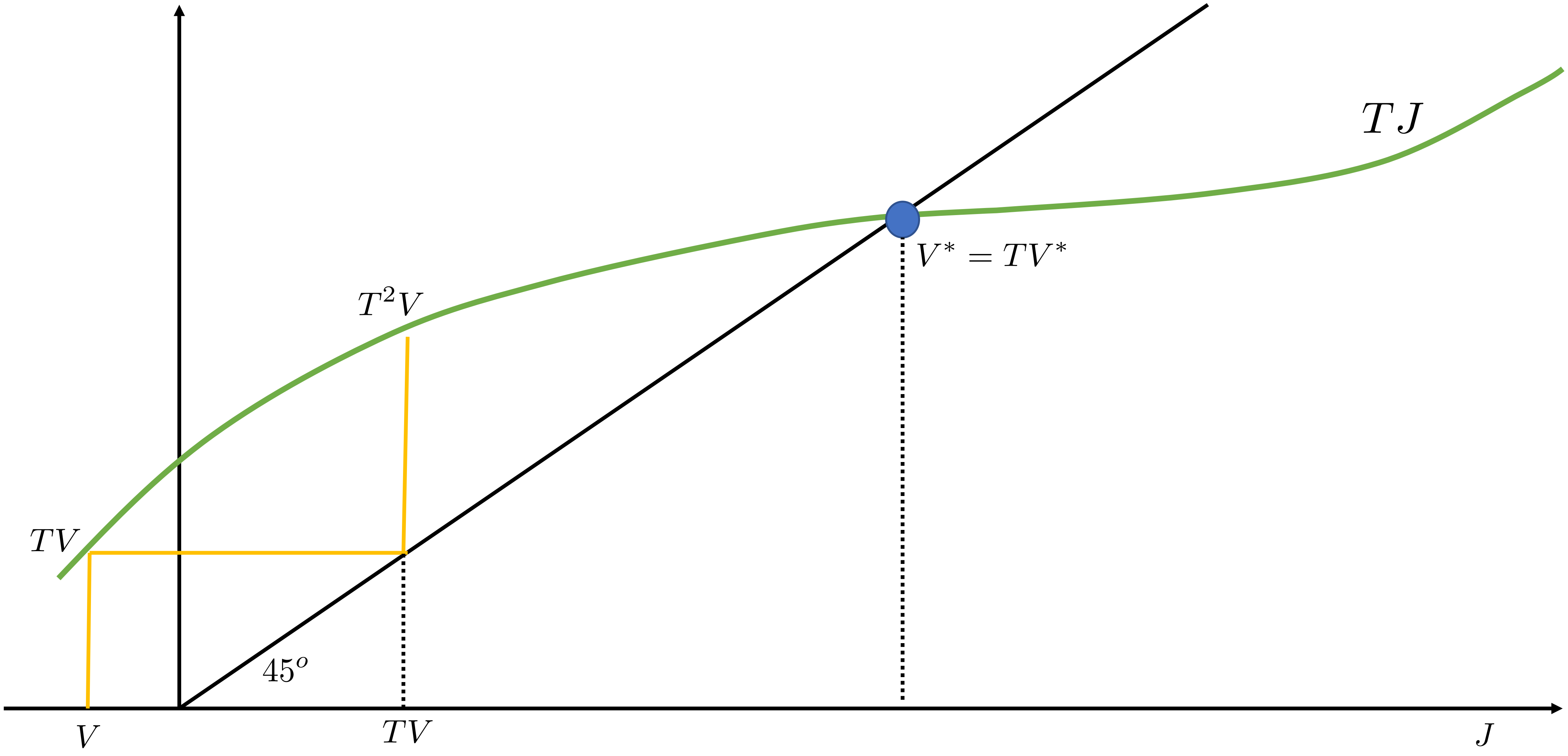
Contraction, Fixed Point, Convergence



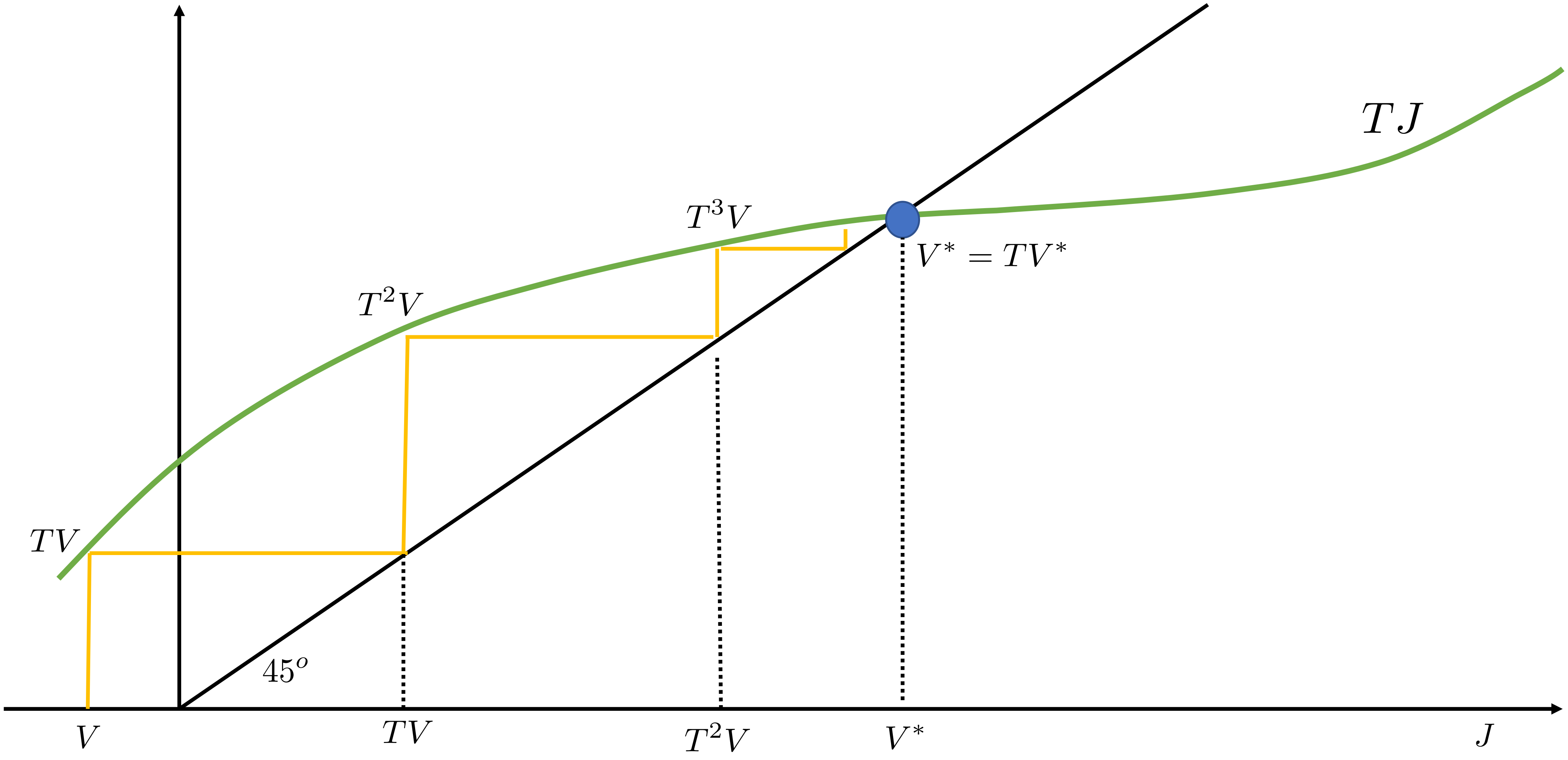
Contraction, Fixed Point, Convergence



Contraction, Fixed Point, Convergence



Contraction, Fixed Point, Convergence



Summary: Computing V^*

- Optimal value function V^* satisfies the Bellman optimality equation

Summary: Computing V^*

- Optimal value function V^* satisfies the Bellman optimality equation

- V^* is the unique fixed point of the Bellman operator $V^* = TV^*$

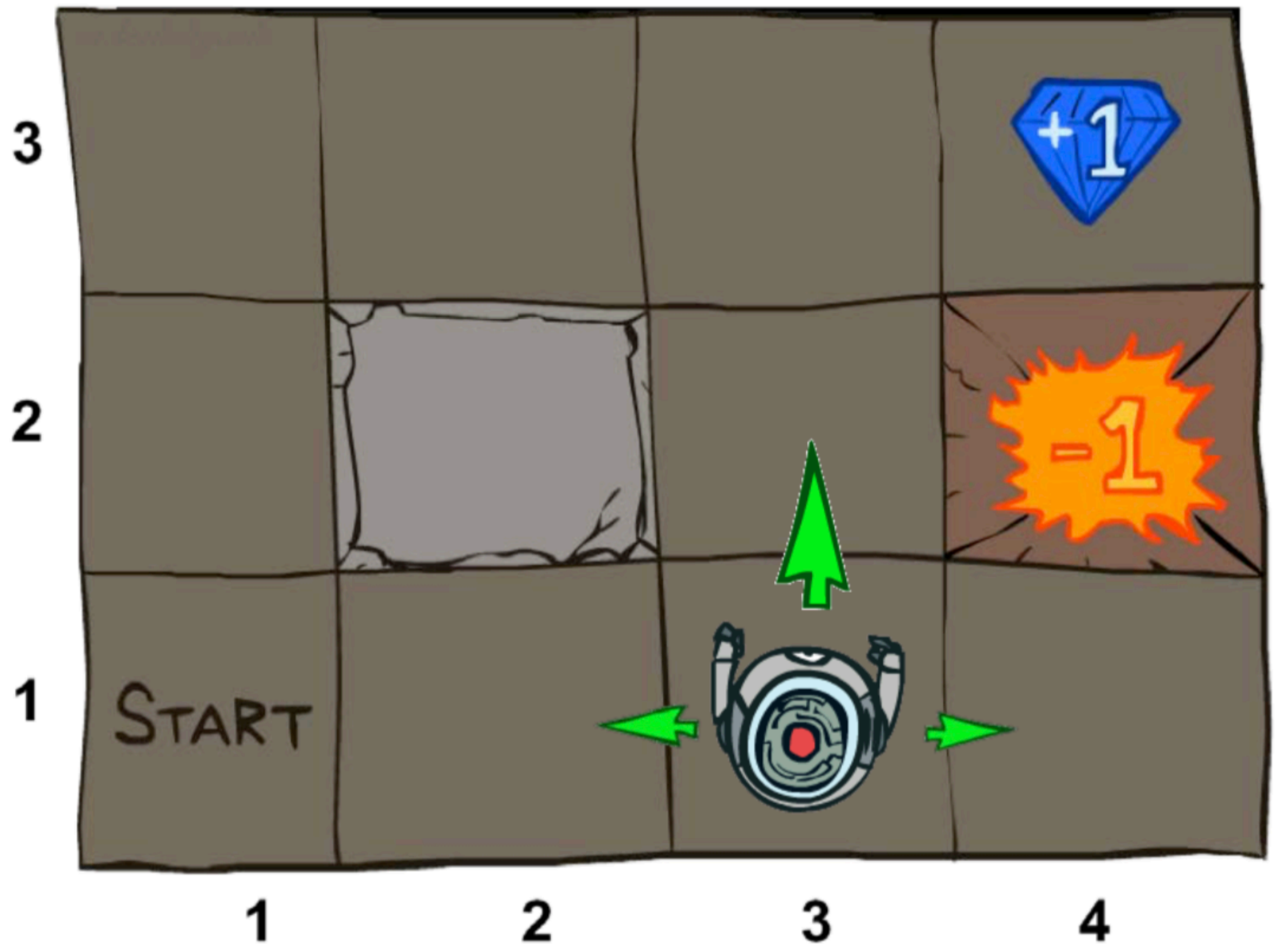
Summary: Computing V^*

- Optimal value function V^* satisfies the Bellman optimality equation

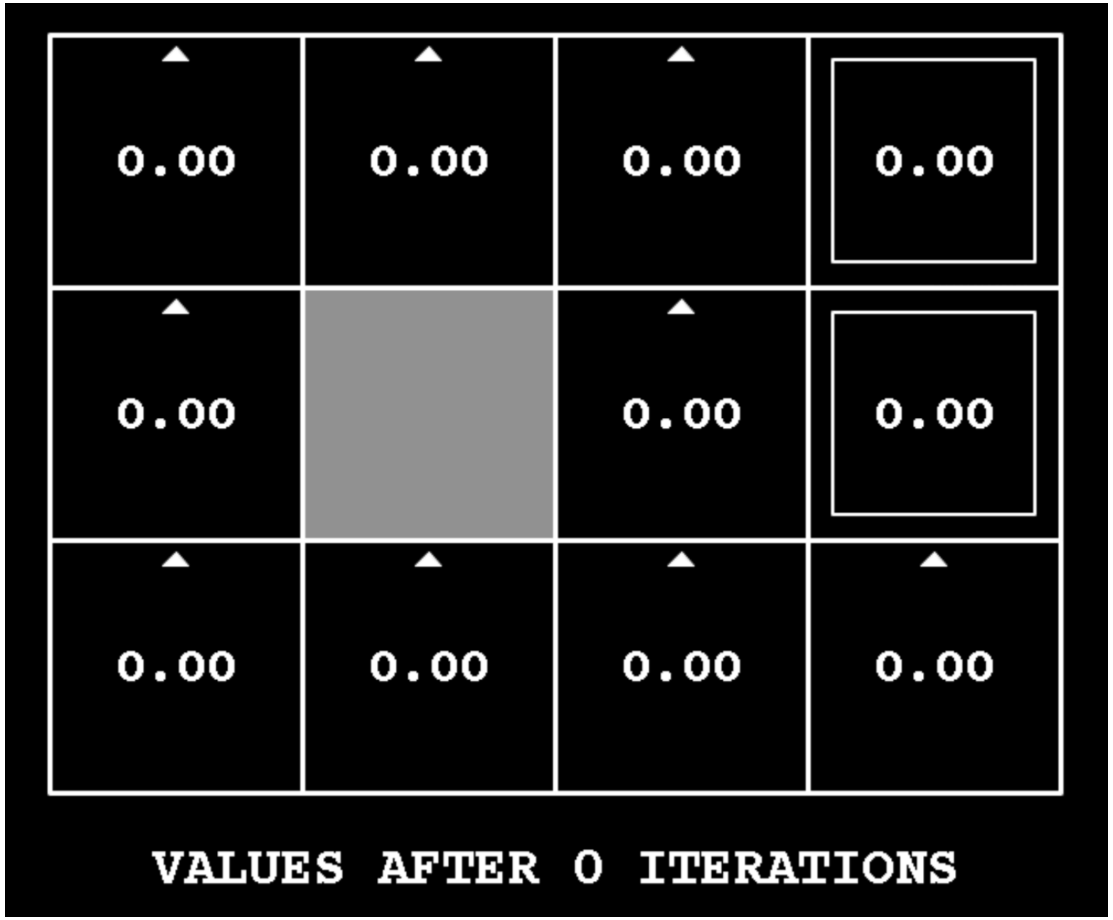
- V^* is the unique fixed point of the Bellman operator $V^* = TV^*$

- Value iteration, $V_{k+1} = TV_k$, converges to V^*

Value Iteration

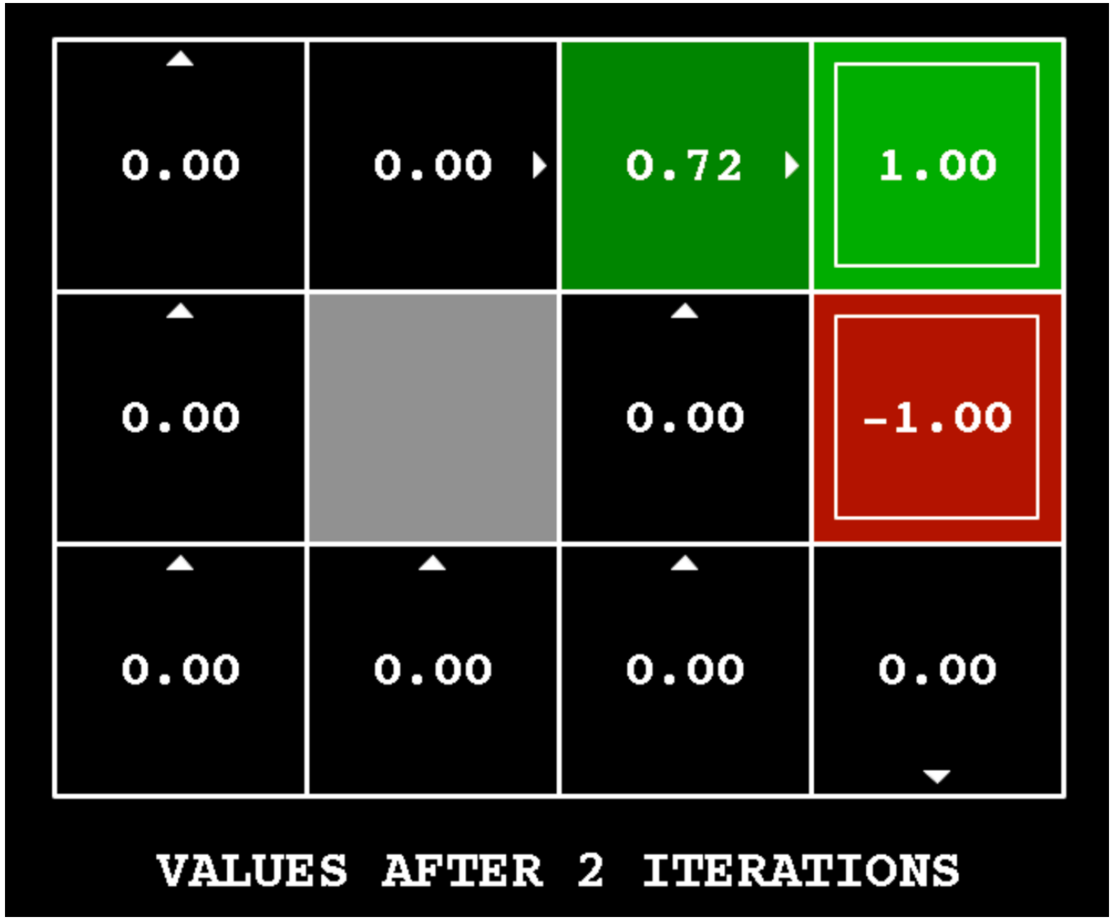


Value Iteration

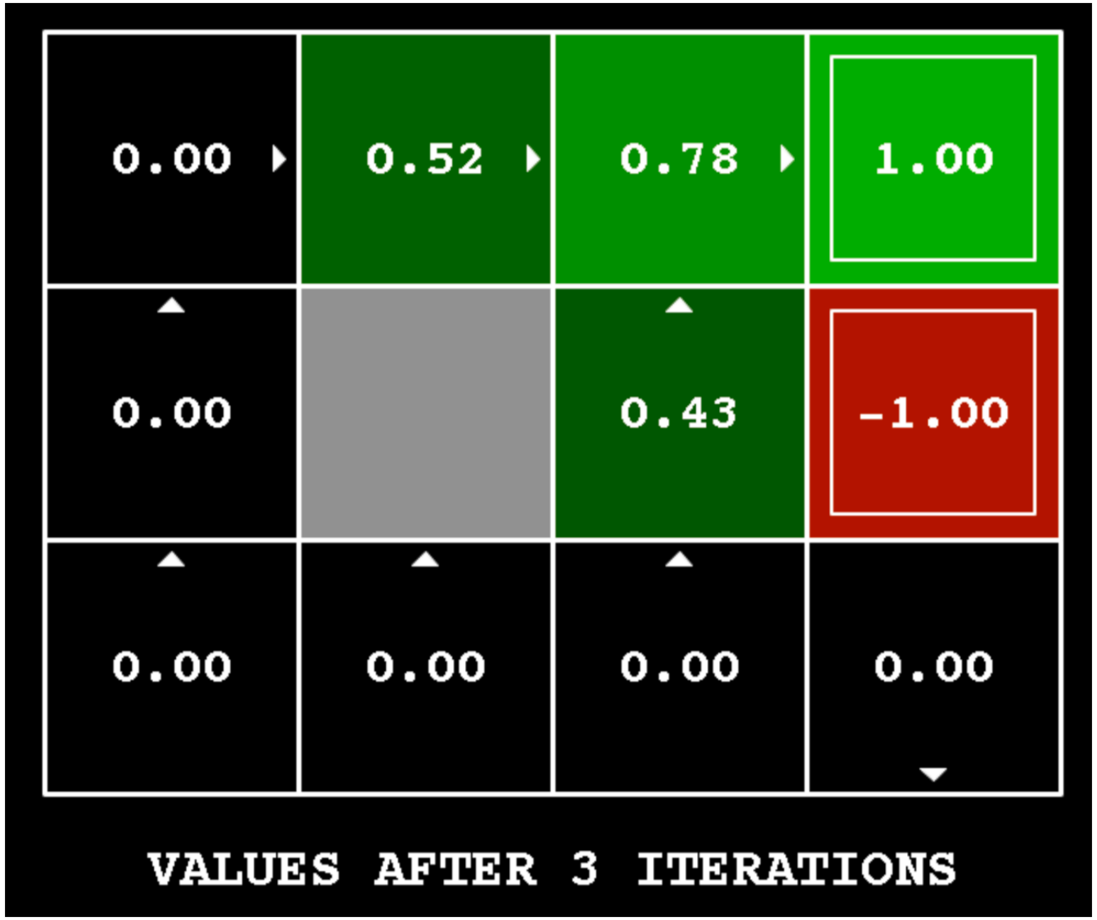


Value Iteration

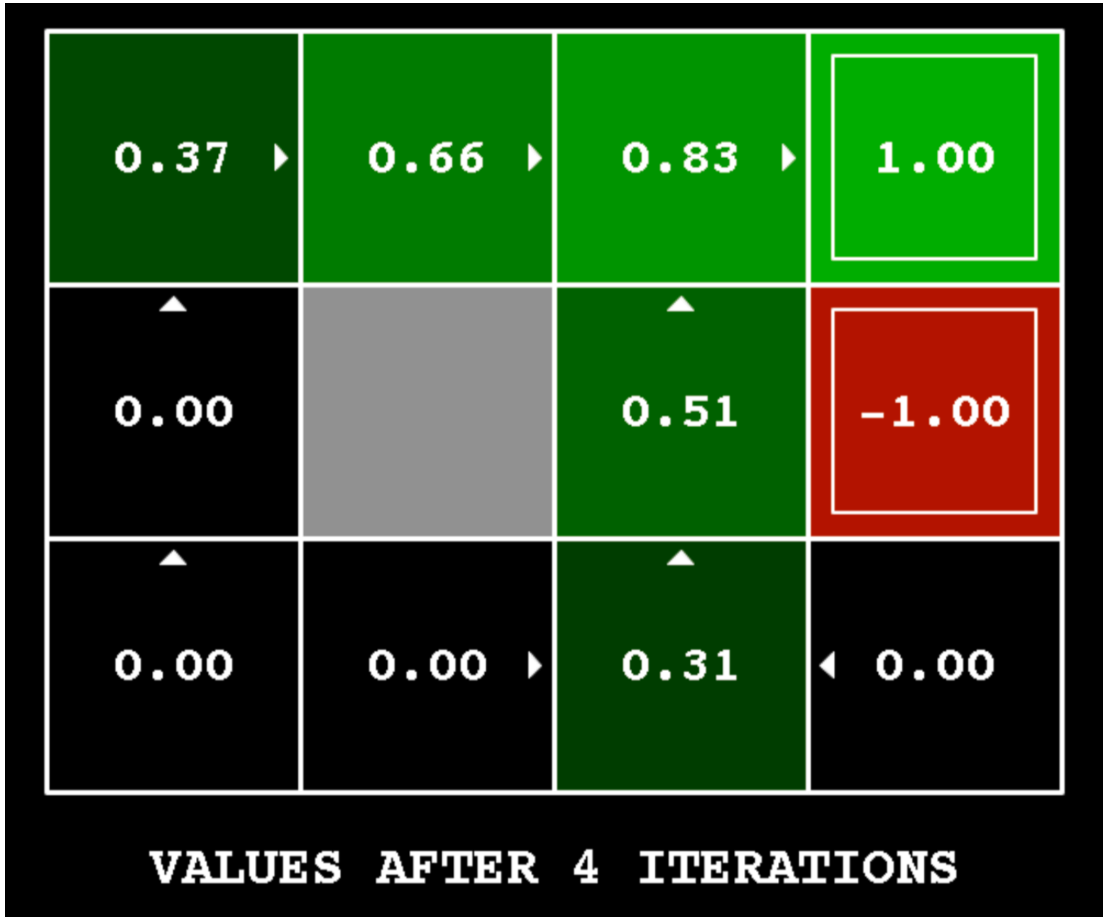
Value Iteration



Value Iteration



Value Iteration



Value Iteration

Value Iteration

Value Iteration

Value Iteration

Value Iteration

Value Iteration

Value Iteration

Value Iteration

State-Action Value Function (Q-function)

- Value of a policy:

$$V_\pi(x) = \mathbb{E}_\pi[R(x_0, \pi(x_0)) + \gamma R(x_1, \pi(x_1)) + \gamma^2 R(x_2, \pi(x_2)) + \gamma^3 R(x_3, \pi(x_3)) + \ldots \mid x_0 = x]$$

  - Expected cumulative discounted reward obtained by starting from state x and following the policy π

State-Action Value Function (Q-function)

- Value of a policy:

$$V_\pi(x) = \mathbb{E}_\pi[R(x_0, \pi(x_0)) + \gamma R(x_1, \pi(x_1)) + \gamma^2 R(x_2, \pi(x_2)) + \gamma^3 R(x_3, \pi(x_3)) + \ldots \mid x_0 = x]$$

  - Expected cumulative discounted reward obtained by starting from state x and following the policy π

- Q-Value of a policy:

$$Q_\pi(x, a) = \mathbb{E}_\pi[R(x_0, a_0)) + \gamma R(x_1, \pi(x_1)) + \gamma^2 R(x_2, \pi(x_2)) + \gamma^3 R(x_3, \pi(x_3)) + \ldots \mid x_0 = x, a_0 = a]$$

  - Expected cumulative discounted reward obtained by starting from state x, taking action a, and then following the policy π

Optimal Q-Function

Q-function for a given policy: $$Q_\pi(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_\pi(y)$$

Optimal Q-Function

Q-function for a given policy: $$Q_\pi(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_\pi(y)$$

Optimal Q-function: $$Q^*(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)$$

Optimal Q-Function

Q-function for a given policy: $$Q_\pi(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_\pi(y)$$

Optimal Q-function: $$Q^*(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)$$

How do we get V* from Q^* ?

Optimal Q-Function

Q-function for a given policy: $$Q_\pi(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_\pi(y)$$

Optimal Q-function: $$Q^*(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)$$

How do we get V* from Q^* ? $$V^*(x) = \max_{a \in \mathcal{A}} Q^*(x,a)$$

Recall optimal Value function: $$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)\right)$$

Optimal Q-Function

Q-function for a given policy: $$Q_\pi(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_\pi(y)$$

Optimal Q-function: $$Q^*(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)$$

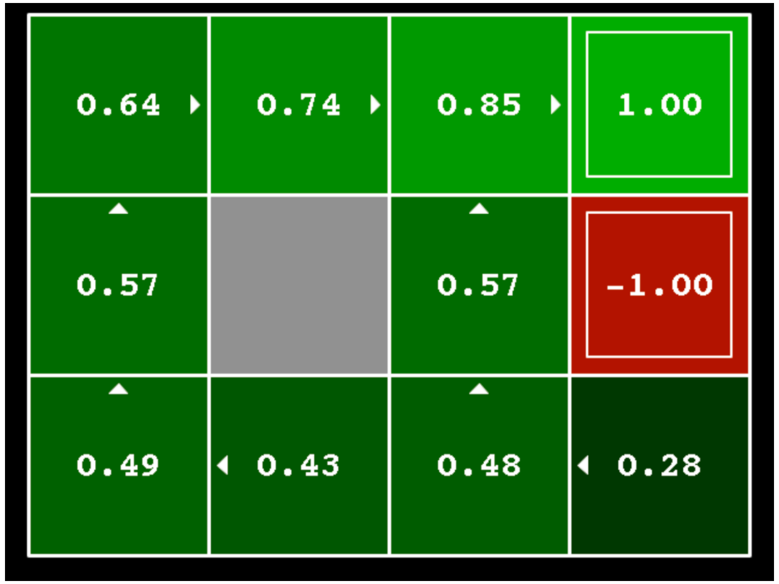
How do we get V* from Q* ? $$V^*(x) = \max_{a \in \mathcal{A}} Q^*(x,a)$$

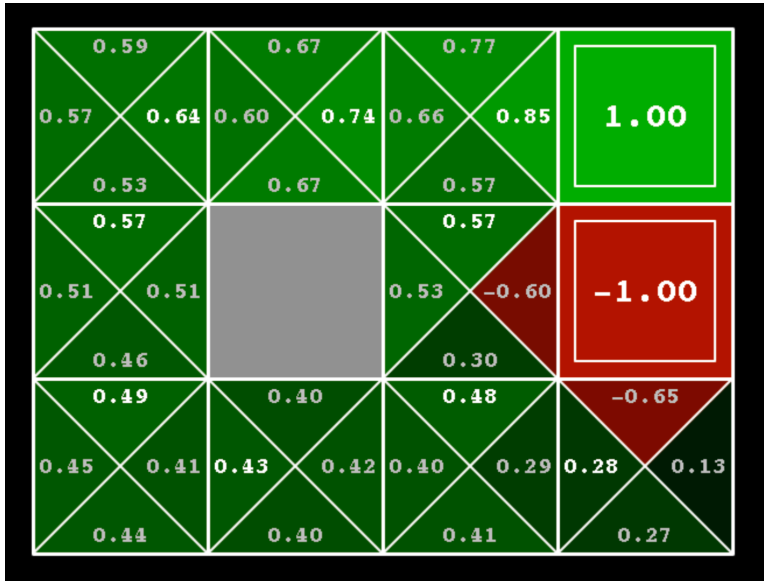
Recall optimal Value function: $$V^*(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V^*(y)\right)$$

Bellman Equation for Q* :

$$Q^*(x,a) = R(x,a) + \gamma \sum_{y} P(y|x,a) \max_{b} Q^*(y,b)$$

Value, Q-value and Policy





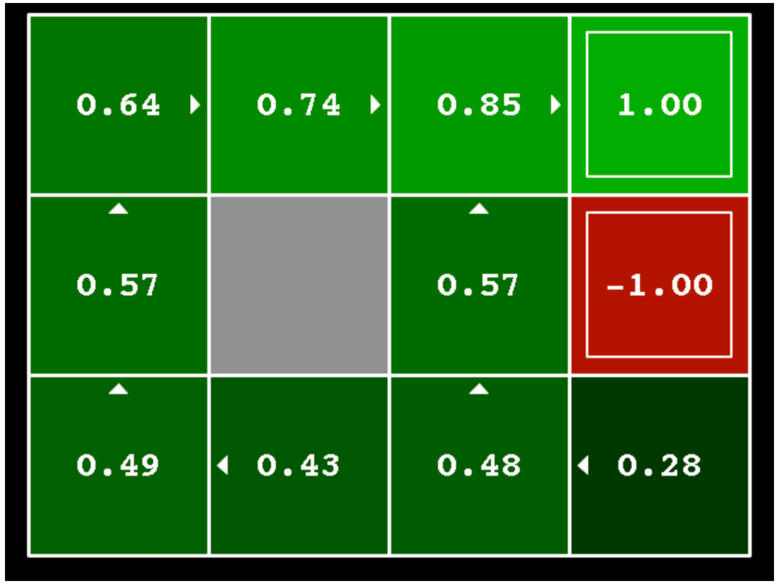
V^* to Q^* :

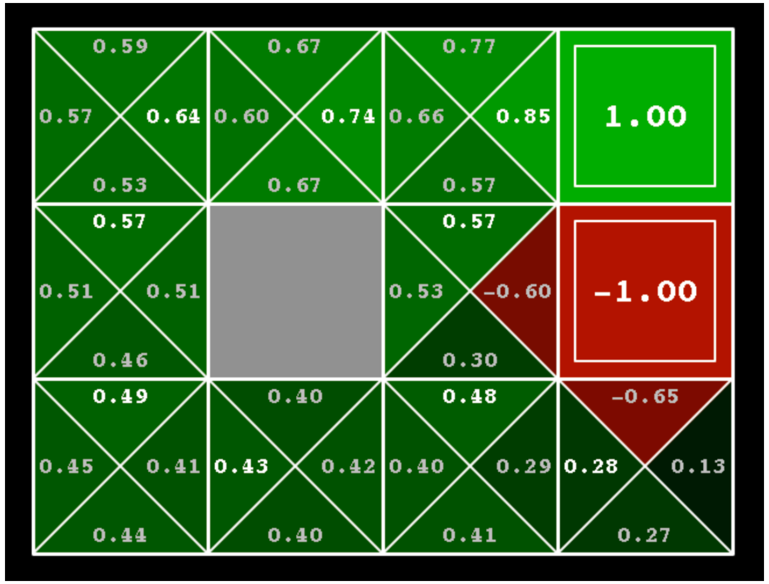
$$Q^*(x,a) = R(x,a) + \gamma \ \mathbb{E}[V^*(y)|x,a]$$

Q^* to V^* :

$$V^*(x) = \max_{a \in \mathcal{A}} Q^*(x,a)$$

Value, Q-value and Policy





V* to Q* :

$$Q^*(x,a) = R(x,a) + \gamma \; \mathbb{E}[V^*(y)|x,a]$$

Q* to V* :

$$V^*(x) = \max_{a \in \mathcal{A}} Q^*(x,a)$$

Computing optimal policy: $\pi^*(x) = \arg\max_a \; Q^*(x,a)$

Policy Iteration

Iteration over policy space to find the optimal policy

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0

2. At each iteration k

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0
2. At each iteration k
   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0
2. At each iteration k
   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)
   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

Greedy update

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0
2. At each iteration k
   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)
   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y)\right)$$

Greedy update

(select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Policy Iteration

Iteration over policy space to find the optimal policy

1. Start with an arbitrary policy π_0
2. At each iteration k
   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)
   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

Greedy update

(select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Policy Iteration converges in finite number of steps!

Policy Iteration

2. At each iteration k

   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

   (select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Policy Iteration

2. At each iteration k

   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

   (select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Theorem 3. *In Policy Iteration, there exists a finite number k_0 such that $V_{\pi_k} = V^*$ for all $k \geq k_0$.*

Policy Iteration

2. At each iteration k

   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

   (select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Theorem 3. *In Policy Iteration, there exists a finite number k_0 such that $V_{\pi_k} = V^*$ for all $k \geq k_0$.*

The sequence $\{V_{\pi_0}, V_{\pi_1}, V_{\pi_2}, \ldots\}$ is a monotonically increasing sequence bounded above by V^*, and the sequence must therefore converge. As there are finitely many deterministic policies, this convergence must happen in finite time.

Policy Iteration

2. At each iteration k

   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

   (select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Theorem 3. *In Policy Iteration, there exists a finite number k_0 such that $V_{\pi_k} = V^*$ for all $k \geq k_0$.*

The sequence $\{V_{\pi_0}, V_{\pi_1}, V_{\pi_2}, \dots\}$ is a monotonically increasing sequence bounded above by V^*, and the sequence must therefore converge. As there are finitely many deterministic policies, this convergence must happen in finite time.

Bad news: It can take up to $n_a^{n_x}$ steps

Policy Iteration

2. At each iteration k

   (a) Evaluate policy π_k to get V_{π_k} (solve the equation $V_{\pi_k} = T_{\pi_k} V_{\pi_k}$)

   (b) Update the policy to get π_{k+1}

$$\pi_{k+1}(x) = \arg\max_{a \in \mathcal{A}} (R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V_{\pi_k}(y))$$

   (select π_{k+1} s.t. $T_{\pi_{k+1}} V_{\pi_k} = T V_{\pi_k}$)

Theorem 3. *In Policy Iteration, there exists a finite number k_0 such that $V_{\pi_k} = V^*$ for all $k \geq k_0$.*

The sequence $\{V_{\pi_0}, V_{\pi_1}, V_{\pi_2}, \ldots\}$ is a monotonically increasing sequence bounded above by V^*, and the sequence must therefore converge. As there are finitely many deterministic policies, this convergence must happen in finite time.

Bad news: It can take up to $n_a^{n_x}$ steps

Good news: Much faster in practice!

Dynamic Programming

1. How to evaluate a policy?

   Given a policy π, $V_{k+1} = T_{\pi_k} V_k$. Then $V_k \to V_\pi$

2. How to compute the optimal value function V^*?

   Value Iteration: $V_{k+1} = TV_k$. Then $V_k \to V^*$

3. How to compute the optimal policy π^*?

   Value Iteration, Policy Iteration

Q Learning

References:

1. "Reinforcement Learning: An Introduction", Richard S. Sutton and Andrew G. Barto , Ch. 5, Ch. 6
2. "Algorithms for Reinforcement Learning", Csaba Szepesvari, Ch. 3
3. "Neuro-Dynamic Programming", Dimitri P. Bertsekas and John Tsitsiklis, Ch. 5

Dynamic Programming

1. How to evaluate a policy?

   Given a policy π, $V_{k+1} = T_{\pi_k} V_k$. Then $V_k \to V_\pi$

2. How to compute the optimal value function V^*?

   Value Iteration: $V_{k+1} = T V_k$. Then $V_k \to V^*$

3. How to compute the optimal policy π^*?

   Value Iteration, Policy Iteration

Requires the system model ***P***

Reinforcement Learning

1. How to evaluate a policy?
2. How to compute the optimal value function V^*?
3. How to compute the optimal policy π^*?

When the system model is unknown

Reinforcement Learning

1. How to evaluate a policy?
2. How to compute the optimal value function V^*?
3. How to compute the optimal policy π^*?

When the system model is unknown

- Need to learn from experiences and observations
  - Observations: Sequences of states, actions, and rewards

Bellman Operator for Q-Function

- Recall Bellman Operator for V:

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

- Bellman Operator for Q

$$(FQ)(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \max_{b \in \mathcal{A}} Q(y,b)$$

Bellman Operator for Q-Function

- Recall Bellman Operator for V:

$$(TV)(x) = \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a)V(y)\right)$$

- Bellman Operator for Q

$$(FQ)(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \max_{b \in \mathcal{A}} Q(y,b)$$

Proposition 1. *(i) F is a monotone mapping. (ii) F is a contraction mapping*

Why Q-function?

- Q to action is simple: $\pi^*(x) = \arg\max_{a \in \mathcal{A}} Q^*(x, a)$

Why Q-function?

- Obtaining unbiased samples of (***FQ***) is easier than obtaining unbiased samples of (***TV***)

Why Q-function?

- Obtaining unbiased samples of (***FQ***) is easier than obtaining unbiased samples of (***TV***)

$$
\begin{aligned}
(FQ)(x,a) &= R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \max_{b \in \mathcal{A}} Q(y,b) \\
&= R(x,a) + \gamma \, \mathbb{E}[\max_{b \in \mathcal{A}} Q(x_{t+1},b) | x_t = x, a_t = a]
\end{aligned}
$$

Why Q-function?

- Obtaining unbiased samples of (***FQ***) is easier than obtaining unbiased samples of (***TV***)

$$(FQ)(x,a) = R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \max_{b \in \mathcal{A}} Q(y,b)$$
$$= R(x,a) + \gamma \, \mathbb{E}[\max_{b \in \mathcal{A}} Q(x_{t+1},b) | x_t = x, a_t = a]$$

$R(x_t,a_t) + \gamma \max_{b \in \mathcal{A}} Q(x_{t+1},b)$ is an unbiased sample of $(FQ)(x_t,a_t)$ if $x_{t+1} \sim P(\cdot|x_t,a_t)$

Why Q-function?

- Obtaining unbiased samples of (***FQ***) is easier than obtaining unbiased samples of (***TV***)

$$
\begin{aligned}
(FQ)(x,a) &= R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) \max_{b \in \mathcal{A}} Q(y,b) \\
&= R(x,a) + \gamma\, \mathbb{E}[\max_{b \in \mathcal{A}} Q(x_{t+1},b) | x_t = x, a_t = a]
\end{aligned}
$$

$R(x_t,a_t) + \gamma \max_{b \in \mathcal{A}} Q(x_{t+1},b)$ is an unbiased sample of $(FQ)(x_t,a_t)$ if $x_{t+1} \sim P(\cdot|x_t,a_t)$

$$
\begin{aligned}
(TV)(x) &= \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma \sum_{y \in \mathcal{X}} P(y|x,a) V(y)\right) \\
&= \max_{a \in \mathcal{A}} \left(R(x,a) + \gamma\, \mathbb{E}[V(x_{t+1}) | x_t = x, a_t = a]\right)
\end{aligned}
$$

$\max_{a \in \mathcal{A}} \left(R(x_t,a_t) + \gamma\, V(x_{t+1})\right)$ is not an unbiased sample of $(TV)(x_t)$

Q-Learning Algorithm

- Learning the optimal value function directly
  - Action can be taken with respect to a *behavioral* policy

Q-Learning Algorithm

- Learning the optimal value function directly
  - Action can be taken with respect to a *behavioral* policy

---

Algorithm Q-Learning

Initialization: A behavioral policy μ, $t = 0$, initial state $s_t = s_0$

for each $t = 0, 1, 2, \ldots$ **do**

  Observe the current state s_t

  Take action a_t according to the policy μ: $a_t \sim \mu(s_t)$

  Observe the reward R_t and the next state s_{t+1}

  Update Q:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(R_t + \gamma \max_b Q(s_{t+1}, b) - Q(s_t, a_t)\right)$$

end for

Q-Learning Algorithm

Theorem. If: (i) all state-action pairs are visited infinitely often, and (ii) step size satisfies Robbins-Munro condition, $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$, then Q-Learning will converge, i.e., $Q_t \to Q^*$ almost surely.

Q-Learning Algorithm

Theorem. If: (i) all state-action pairs are visited infinitely often, and (ii) step size satisfies Robbins-Munro condition, $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$, then Q-Learning will converge, i.e., $Q_t \to Q^*$ almost surely.

- It is a very surprising result!
  - Convergence to the optimal Q-value even if you are acting sub-optimally (according to a behavioral policy)

Q-Learning Algorithm

Theorem. If: (i) all state-action pairs are visited infinitely often, and (ii) step size satisfies Robbins-Munro condition, $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$, then Q-Learning will converge, i.e., $Q_t \to Q^*$ almost surely.

- It is a very surprising result!
  - Convergence to the optimal Q-value even if you are acting sub-optimally (according to a behavioral policy)
- This is called off-policy learning

Q-Learning Algorithm

Theorem. If: (i) all state-action pairs are visited infinitely often, and (ii) step size satisfies Robbins-Munro condition, $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$, then Q-Learning will converge, i.e., $Q_t \to Q^*$ almost surely.

- It is a very surprising result!
  - Convergence to the optimal Q-value even if you are acting sub-optimally (according to a behavioral policy)
- This is called off-policy learning
- Caveats:
  - Need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
  - Need to use the correct learning rate

Q-Learning Algorithm: Behavioral Policy

- Q-Learning is an off-policy learning algorithm
  - However, need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
- Standard behavioral policy: ε-greedy policy

$$\pi_{k+1}(s) = \epsilon\text{-greedy}(Q_k) = \begin{cases} a^* = \arg\max_a Q_k(s,a) & \text{with prob. } 1 - \epsilon\ (|\mathcal{A}| - 1) \\ a & \text{with prob } \epsilon \end{cases}$$

Q-Learning Algorithm: Behavioral Policy

- Q-Learning is an off-policy learning algorithm
  - However, need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
- Standard behavioral policy: ε-greedy policy

$$\pi_{k+1}(s) = \epsilon\text{-greedy}(Q_k) = \begin{cases} a^* = \arg\max_a Q_k(s,a) & \text{with prob. } 1 - \epsilon\ (|\mathcal{A}| - 1) \\ a & \text{with prob } \epsilon \end{cases}$$

  - Under some conditions on the transition probability matrix, all state-action pairs will be visited infinitely often (in the limit)

Q-Learning Algorithm: Behavioral Policy

- Q-Learning is an off-policy learning algorithm
  - However, need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
- Standard behavioral policy: ε-greedy policy

$$\pi_{k+1}(s) = \epsilon\text{-greedy}(Q_k) = \begin{cases} a^* = \arg\max_a Q_k(s,a) & \text{with prob. } 1 - \epsilon\ (|\mathcal{A}| - 1) \\ a & \text{with prob } \epsilon \end{cases}$$

  - Under some conditions on the transition probability matrix, all state-action pairs will be visited infinitely often (in the limit)

Q-Learning Algorithm: Behavioral Policy

- Q-Learning is an off-policy learning algorithm
  - However, need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
- Standard behavioral policy: ε-greedy policy

$$\pi_{k+1}(s) = \epsilon\text{-greedy}(Q_k) = \begin{cases} a^* = \arg\max_a Q_k(s,a) & \text{with prob. } 1-\epsilon\ (|\mathcal{A}|-1) \\ a & \text{with prob } \epsilon \end{cases}$$

  - Under some conditions on the transition probability matrix, all state-action pairs will be visited infinitely often (in the limit)
- Is ε-greedy (Q) is the optimal online control policy?

Q-Learning Algorithm: Behavioral Policy

- Q-Learning is an off-policy learning algorithm
  - However, need to *explore* enough (make sure that behavioral policy will visit all state-action pairs infinitely often)
- Standard behavioral policy: ε-greedy policy

$$\pi_{k+1}(s) = \epsilon\text{-greedy}(Q_k) = \begin{cases} a^* = \arg\max_a Q_k(s,a) & \text{with prob. } 1-\epsilon\ (|\mathcal{A}|-1) \\ a & \text{with prob } \epsilon \end{cases}$$

  - Under some conditions on the transition probability matrix, all state-action pairs will be visited infinitely often (in the limit)
- Is ε-greedy (Q) is the optimal online control policy?
- What do we mean by optimal online control policy?

Exploration vs Exploitation, Sample Complexity, Safety, Stability

Deep Q Learning

Problems with Large States Space

- RL should be useful to solve large problems:
  - Backgammon: 10^{20}
  - Go: 10^{170} (how many atoms are in the universe?)
  - Helicopter: continuous state

Problems with Large States Space

- RL should be useful to solve large problems:
  - Backgammon: 10^{20}
  - Go: 10^{170} (how many atoms are in the universe?)
  - Helicopter: continuous state
- Recall Q-learning assumptions for convergence

Problems with Large States Space

- RL should be useful to solve large problems:
  - Backgammon: 10^{20}
  - Go: 10^{170} (how many atoms are in the universe?)
  - Helicopter: continuous state
- Recall Q-learning assumptions for convergence

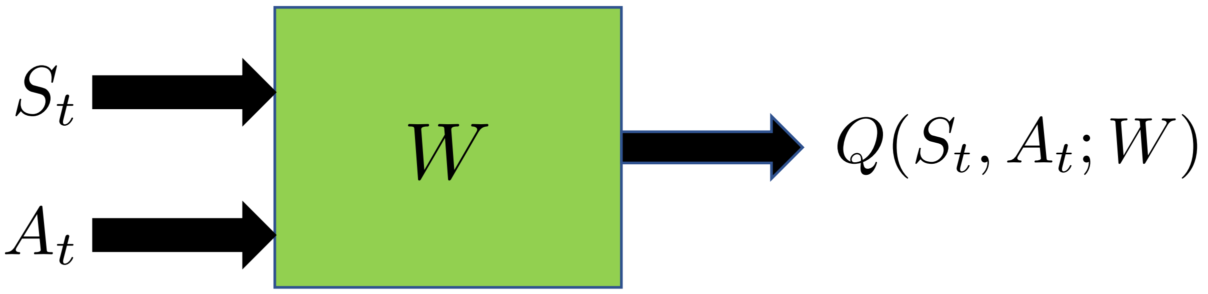
- Will we ever solve such a large scale systems?

Tabular Form to Function Approximation

- We have represented V, Q, π in tabular form
  - One element for each state / (state, action)

- Function Approximation:

$$Q(s,a) \approx Q(s,a;w)$$



Generalize from seen states to unseen states

What Do We Need

- A good class of function approximators
  - Linear combinations of features
  - Decision trees
  - Tile coding
  - Fourier basis
  - Reproducing Kernel Hilbert Spaces
  - Neural networks

- A good training algorithm that is suitable for non-iid and non-stationary data

Q-Learning with FA

- Q-Learning with FA:

$$w_{t+1} = w_t + \alpha_t \left(R_t + \gamma \max_b Q(s_{t+1}, b, w) - Q(s_t, a_t, w)\right) \nabla Q(s_t, a_t, w)$$

Q-Learning with FA

- Q-Learning with FA:

$$w_{t+1} = w_t + \alpha_t \left(R_t + \gamma \max_b Q(s_{t+1}, b, w) - Q(s_t, a_t, w)\right) \nabla Q(s_t, a_t, w)$$

- Does it converge?
  - No convergence proofs even with linear function approximation!

Q-Learning with FA

- Q-Learning with FA:

$$w_{t+1} = w_t + \alpha_t \left(R_t + \gamma \max_b Q(s_{t+1}, b, w) - Q(s_t, a_t, w)\right) \nabla Q(s_t, a_t, w)$$

- Does it converge?
  - No convergence proofs even with linear function approximation!

"Deadly Triad" according to Sutton and Barto (2018)

Function approximation A powerful, scalable way of generalizing from a state space much larger than the memory and computational resources (e.g., linear function approximation or ANNs).

Bootstrapping Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods).

Off-policy training Training on a distribution of transitions other than that produced by the target policy. Sweeping through the state space and updating all states uniformly, as in dynamic programming, does not respect the target policy and is an example of off-policy training.

Correlation and Forgetting in Q-Learning

- Q-Learning:

1. Take some actions a_i and get the data point $e_i = (s_i, a_i, R_i, s'_i)$
2. Update $w = w + \alpha \left(R_i + \gamma \max_b Q(s'_i, b, w) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$

Correlation and Forgetting in Q-Learning

- Q-Learning:

1. Take some actions a_i and get the data point $e_i = (s_i, a_i, R_i, s'_i)$

2. Update $w = w + \alpha \left(R_i + \gamma \max_b Q(s'_i, b, w) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$

  - Data is used in sequentially, in the same way as they are observed
  - Past data is discarded

Correlation and Forgetting in Q-Learning

- Q-Learning:

  1. Take some actions a_i and get the data point $e_i = (s_i, a_i, R_i, s'_i)$
  2. Update $w = w + \alpha \left(R_i + \gamma \max_b Q(s'_i, b, w) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$

  - Data is used in sequentially, in the same way as they are observed
  - Past data is discarded

- Correlation: of sequential data leads to (strongly) correlated updates
  - This breaks the i.i.d. assumption of many popular SGD algorithms

Correlation and Forgetting in Q-Learning

- Q-Learning:

  1. Take some actions a_i and get the data point $e_i = (s_i, a_i, R_i, s'_i)$
  2. Update $w = w + \alpha \left(R_i + \gamma \max_b Q(s'_i, b, w) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$

  - Data is used in sequentially, in the same way as they are observed
  - Past data is discarded

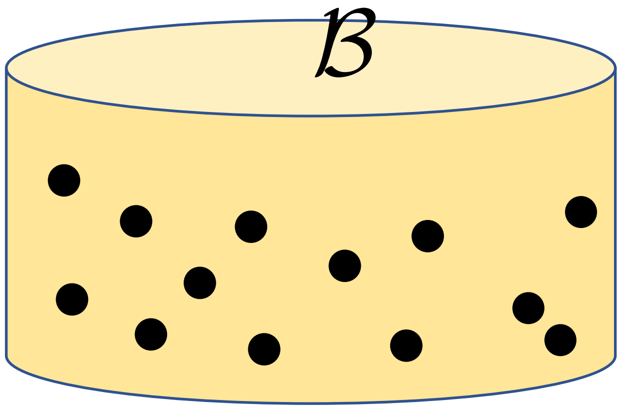
- Correlation: of sequential data leads to (strongly) correlated updates
  - This breaks the i.i.d. assumption of many popular SGD algorithms
- Rapid forgetting: of possibly rare experiences does not effectively use the information (that would be useful later on)

Experience Relay

- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update

Experience Relay

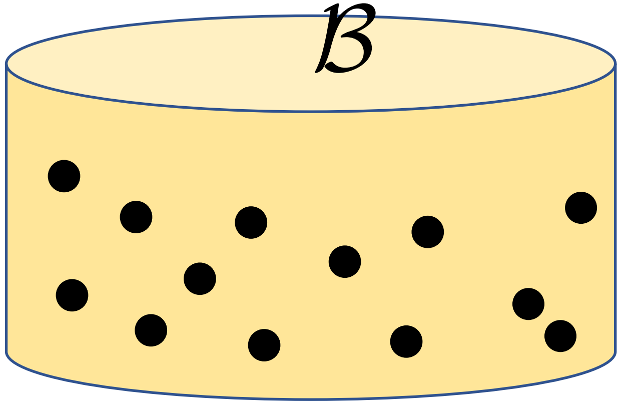
- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update



1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

Experience Relay

- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update

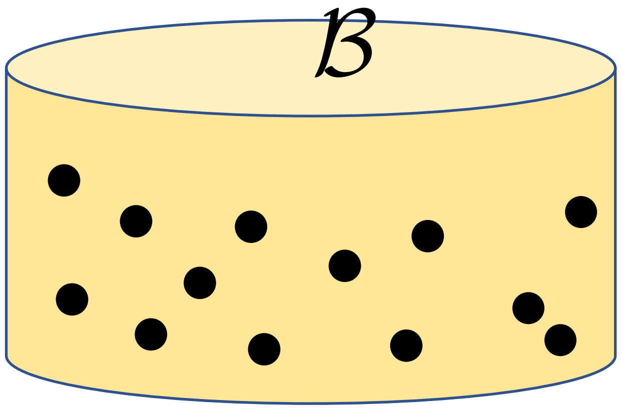


1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

2. At each time t, sample a random mini-batch of transitions $\{e_k\}$ from $\mathcal{B}$

Experience Relay

- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update



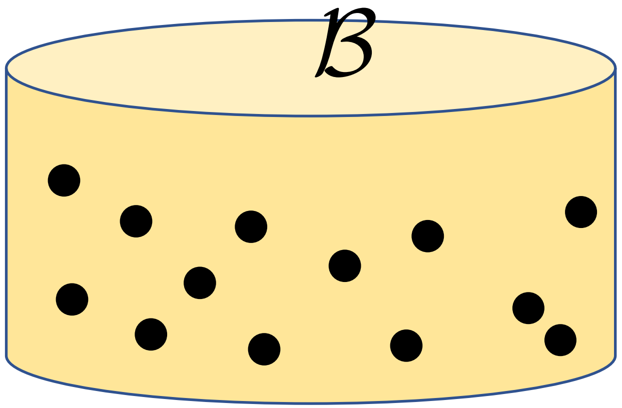
1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

2. At each time t, sample a random mini-batch of transitions $\{e_k\}$ from $\mathcal{B}$

3. For each data point e_k in the sampled mini-batch, update $w = w + \alpha\ (R_k + \gamma \max_b Q(s'_k, b, w) - Q(s_k, a_k, w))\ \nabla Q(s_k, a_k, w)$

Experience Relay

- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update



1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

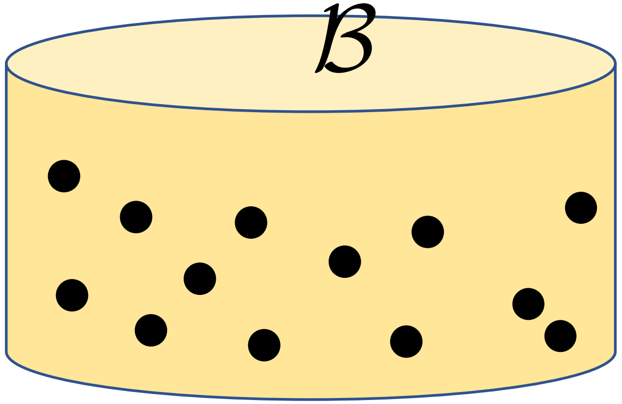
2. At each time t, sample a random mini-batch of transitions $\{e_k\}$ from $\mathcal{B}$

3. For each data point e_k in the sampled mini-batch, update $w = w + \alpha\, (R_k + \gamma \max_b Q(s'_k, b, w) - Q(s_k, a_k, w))\, \nabla Q(s_k, a_k, w)$

4. Select an action a_t according to ϵ-greedy policy

Experience Relay

- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update



1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.

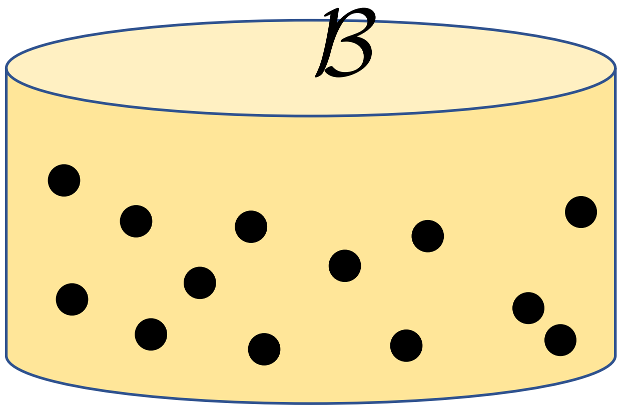
2. At each time t, sample a random mini-batch of transitions $\{e_k\}$ from $\mathcal{B}$

3. For each data point e_k in the sampled mini-batch, update
   $w = w + \alpha\,(R_k + \gamma \max_b Q(s'_k, b, w) - Q(s_k, a_k, w))\,\nabla Q(s_k, a_k, w)$

4. Select an action a_t according to ϵ-greedy policy

5. Add the new data point $e_t = (s_t, a_t, R_t, s'_t)$ to $\mathcal{B}$.

Experience Relay



- Experience Replay:
  - Break the temporal correlations by mixing more and less recent experience for the updates
  - Using rare experience for more than just a single update

1. Initialization; Collect data set $\{e_j = (s_j, a_j, R_j, s'_j)\}$ using some policy. Add it to the replay buffer $\mathcal{B}$.
2. At each time t, sample a random mini-batch of transitions $\{e_k\}$ from $\mathcal{B}$
3. For each data point e_k in the sampled mini-batch, update $w = w + \alpha\,(R_k + \gamma \max_b Q(s'_k, b, w) - Q(s_k, a_k, w))\,\nabla Q(s_k, a_k, w)$
4. Select an action a_t according to ϵ-greedy policy
5. Add the new data point $e_t = (s_t, a_t, R_t, s'_t)$ to $\mathcal{B}$.

Necessary to do off-policy learning for ER!

QL-is off-policy learning!

Target Network

- QL gradient update:

$$w = w + \alpha \left(\underline{R_i + \gamma \max_b Q(s_i', b, w)} - Q(s_i, a_i, w) \right) \nabla Q(s_i, a_i, w)$$

Target for gradient update

Target Network

- QL gradient update:

$$w = w + \alpha \left(\underbrace{R_i + \gamma \max_b Q(s'_i, b, w)}_{\text{Target for gradient update}} - Q(s_i, a_i, w) \right) \nabla Q(s_i, a_i, w)$$

- In supervised learning, the target doesn't depend on the parameter
- In QL, target itself depend on the parameter
  - Target is moving as the parameter is updated
  - This leads to oscillation and instability

Target Network

- QL gradient update:

$$w = w + \alpha \left(\underline{R_i + \gamma \max_b Q(s_i', b, w)} - Q(s_i, a_i, w) \right) \nabla Q(s_i, a_i, w)$$

Target for gradient update

- In supervised learning, the target doesn't depend on the parameter
- In QL, target itself depend on the parameter
  - Target is moving as the parameter is updated
  - This leads to oscillation and instability
- Solution:
  - Use a separate target network
  - Keep the target network unchanged for multiple updates

Target Network

- QL gradient update:

$$w = w + \alpha \left(R_i + \gamma \max_b Q(s_i', b, w) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$$

Target for gradient update

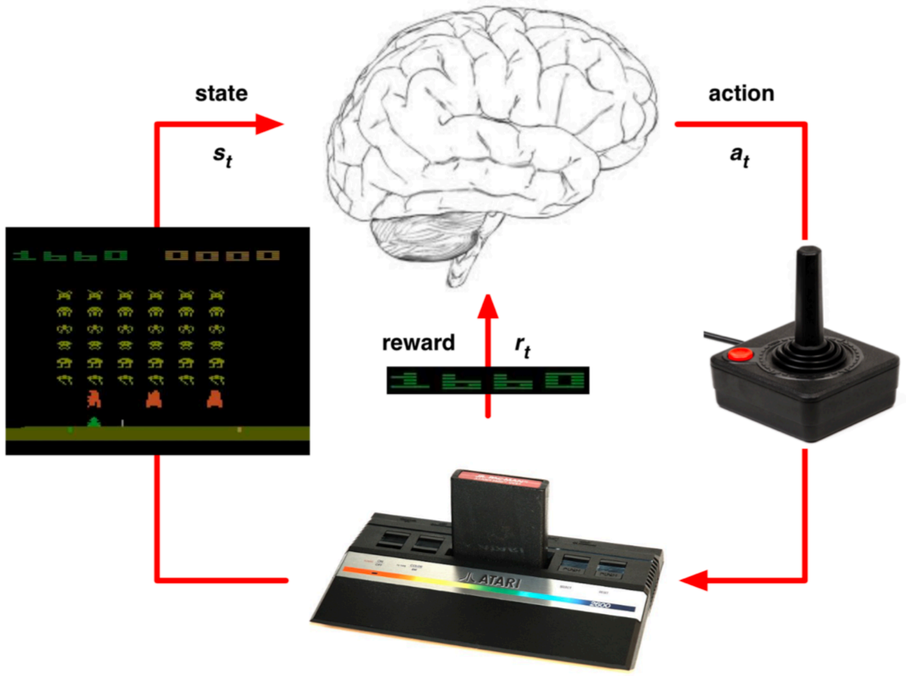
- QL with target network

$$w = w + \alpha \left(R_i + \gamma \max_b Q(s_i', b, w^-) - Q(s_i, a_i, w)\right) \nabla Q(s_i, a_i, w)$$

$$w^- = w \text{ after every } N \text{ steps}$$

Deep Q-Networks (DQN) for Atari Games

- State: Screen Images (history)
  - 210 x 160 pixel, 128 color
- Action: Joystick position
  - 18 different positions
- Reward: Game score

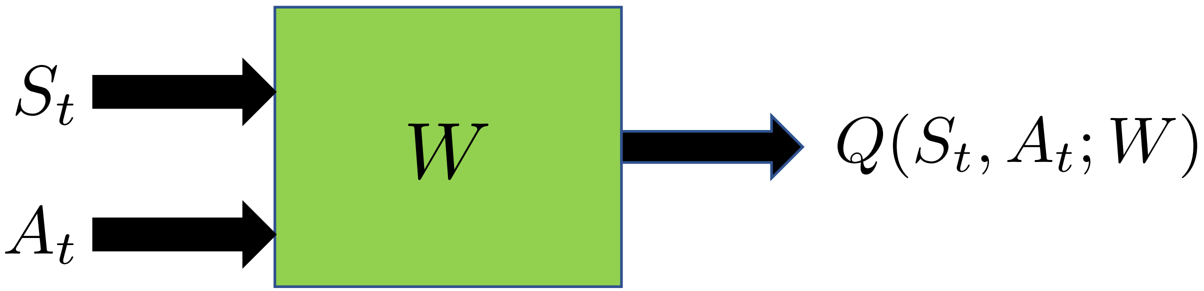


- Objective: Win the game (a control policy that maximizes game score)

Mnih et al, Human-level control through deep reinforcement learning, Nature, 2015

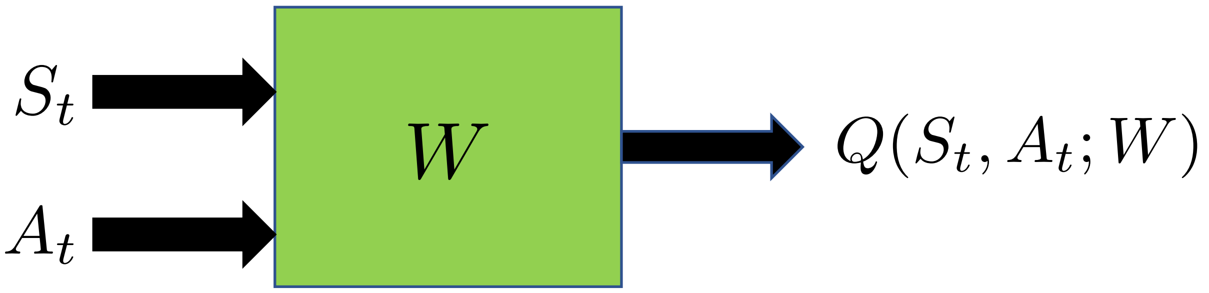
QL with Function Approximation

- Function Approximation

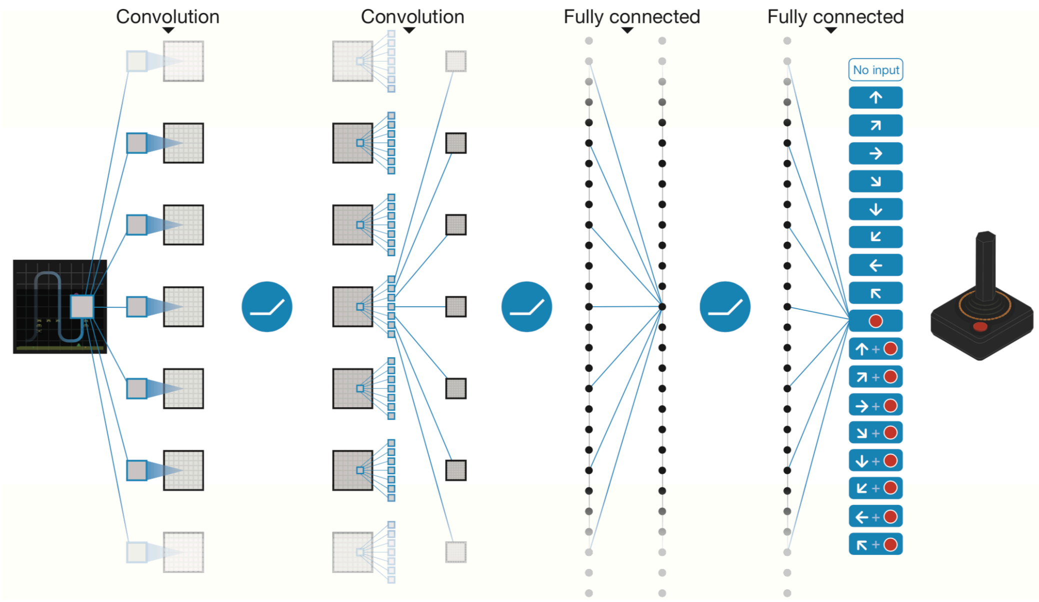


QL with Function Approximation

- Function Approximation



Use a (deep) NN for function approximation



QL with (Deep) NN

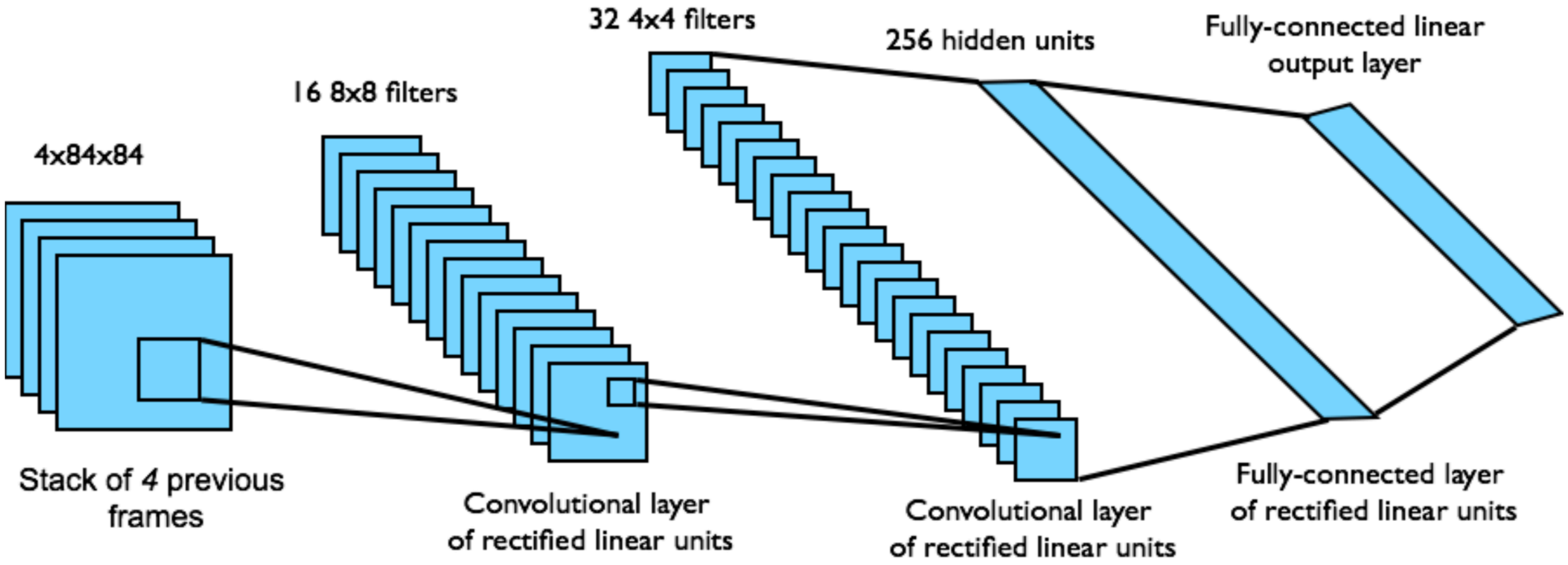
- Preprocessing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Preprocessing: Reduce it to to 84 x 84 images
  - History: Use the 4 most recent frame
  - State size = 84 x 84 x 4

QL with (Deep) NN

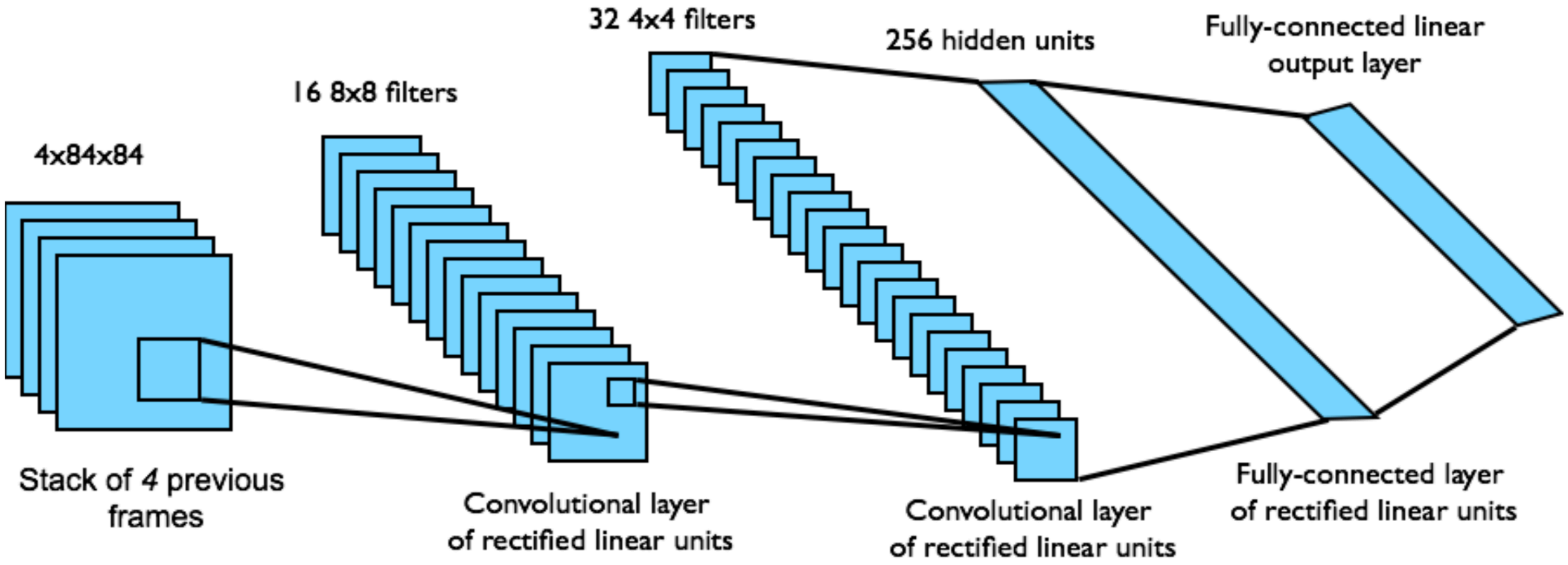
- Preprocessing
  - Each original frame is 210 x 160 pixel images with a 128-colour palette
  - Preprocessing: Reduce it to to 84 x 84 images
  - History: Use the 4 most recent frame
  - State size = 84 x 84 x 4

- Architecture
  - Input state (s), output Q(s, a), for each *a*
    - Avoid forward pass for each possible action
  - Three CNN and two fully connected layers with ReLU

QL with (Deep) NN



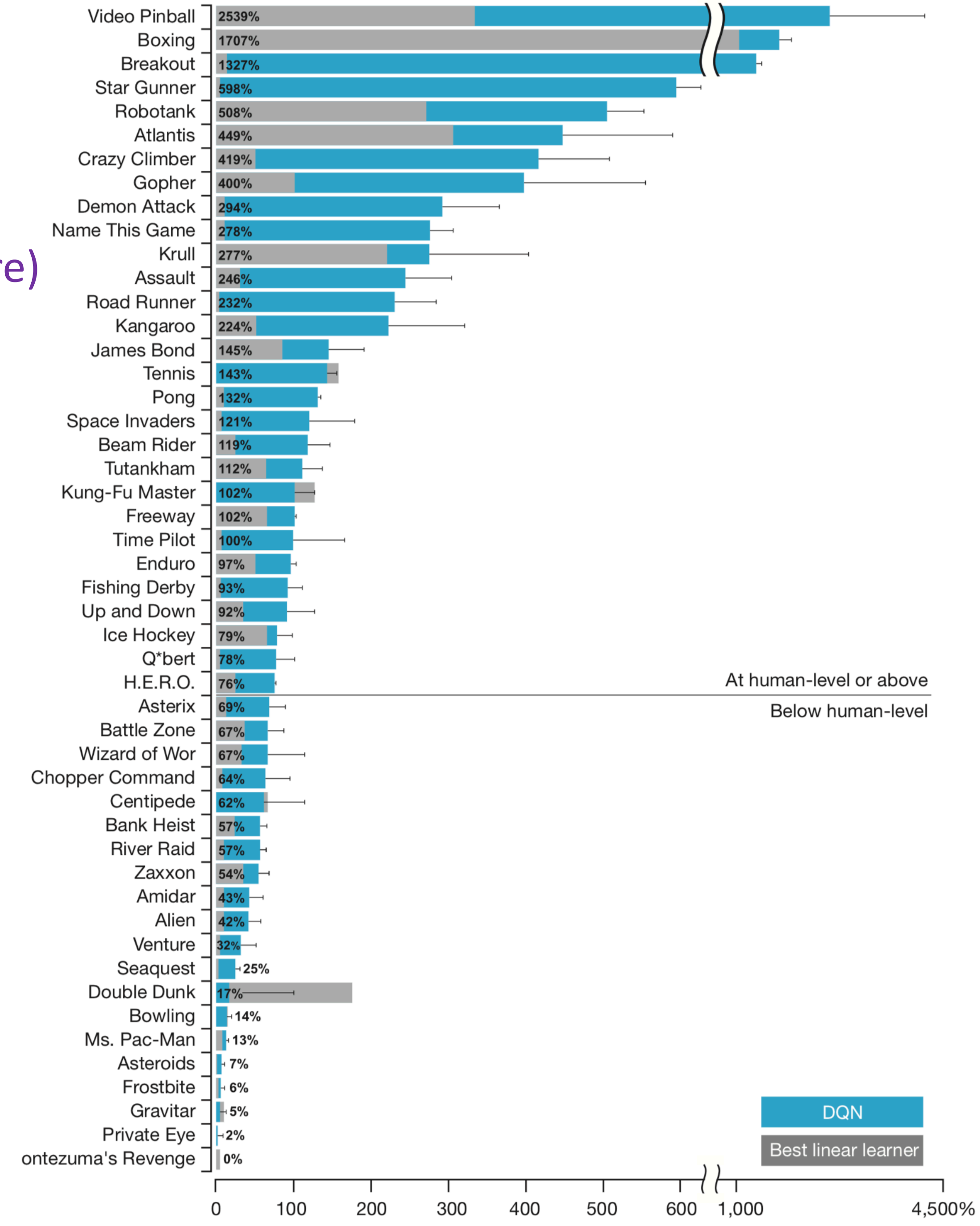
QL with (Deep) NN



Same network architecture, hyperparameters for 49 games in Atari

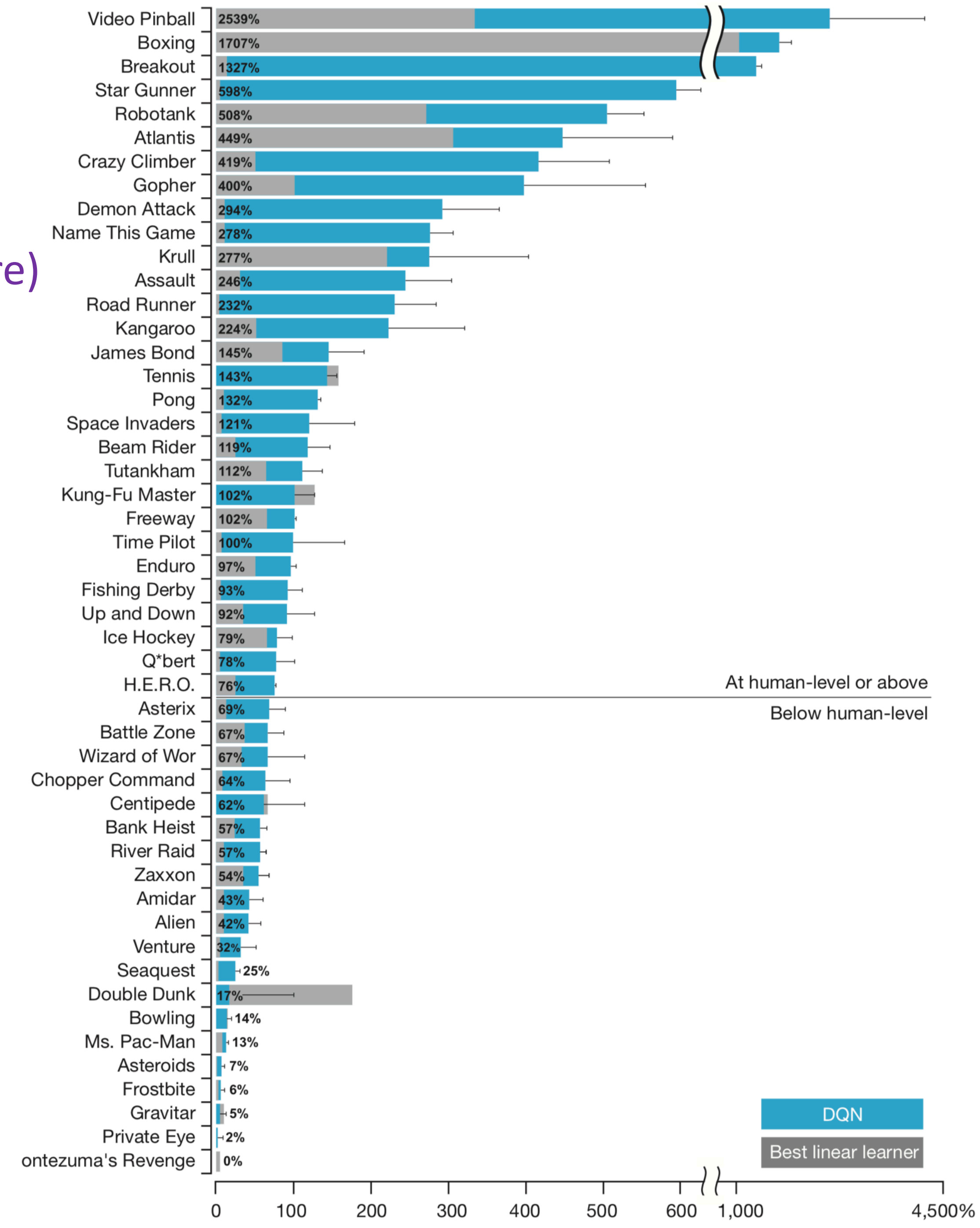
DQN Performance in Atari

- 100 x (DQN score - random play score)/ (human score - random play score)



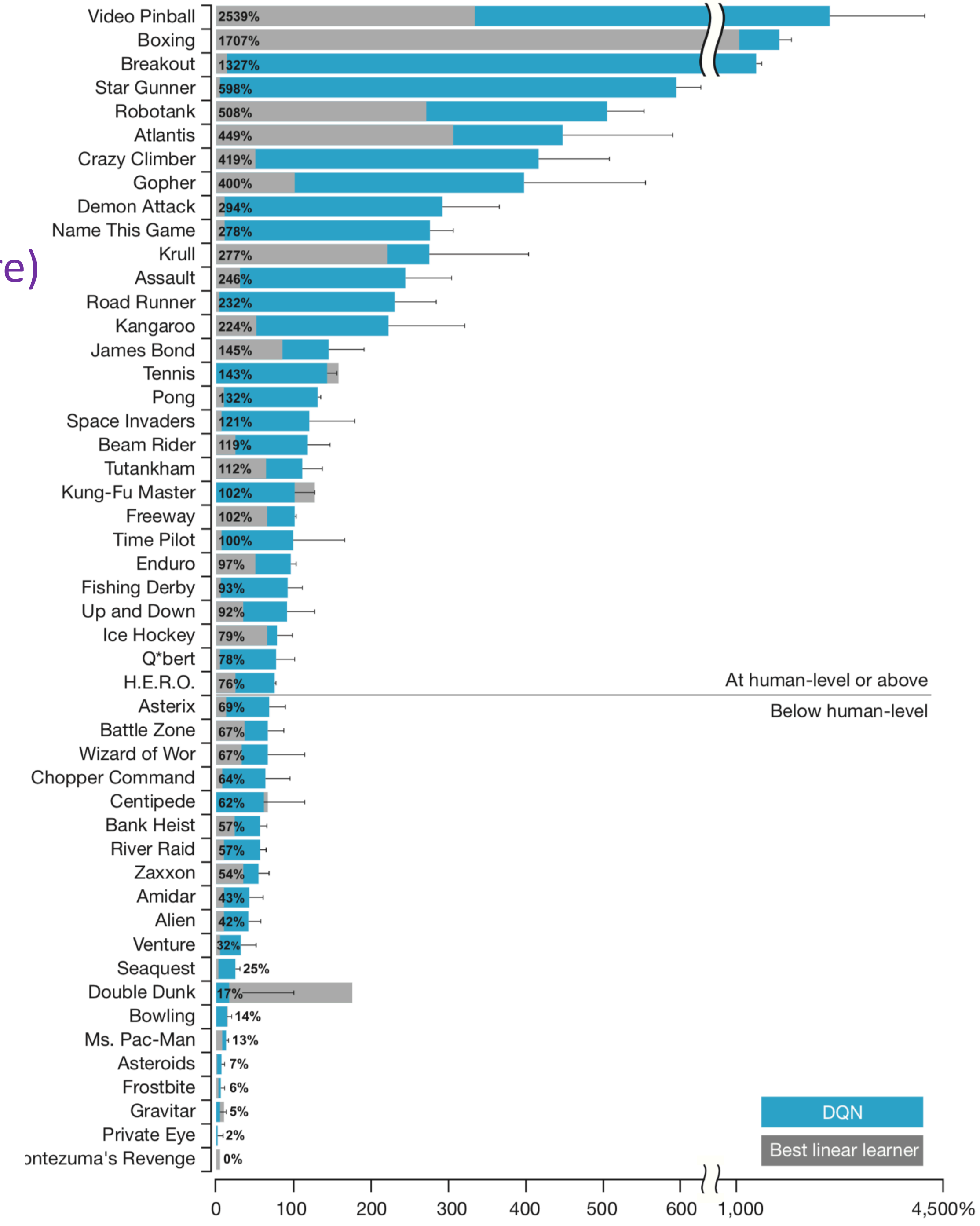
DQN Performance in Atari

- 100 x (DQN score - random play score)/
  (human score - random play score)
- Score for each game is averaged over 30 sessions on each game, each lasting up to 5 minutes and beginning with a random initial game state



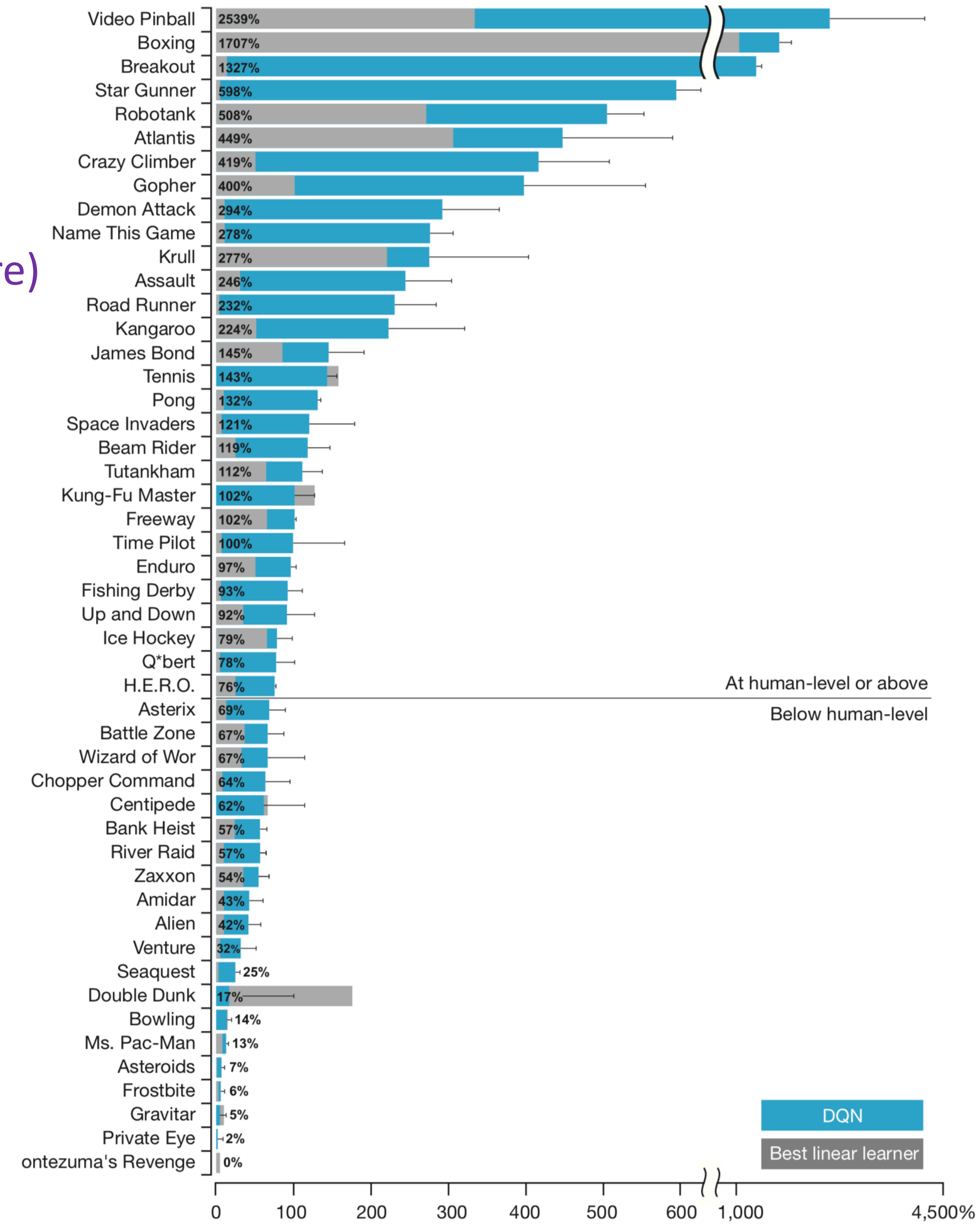
DQN Performance in Atari

- 100 x (DQN score - random play score)/ (human score - random play score)
- Score for each game is averaged over 30 sessions on each game, each lasting up to 5 minutes and beginning with a random initial game state
- The professional human tester played using the same emulator (with the sound turned off). After 2 hours of practice, the human played about 20 episodes of each game for up to 5 minutes each and was not allowed to take any break during this time



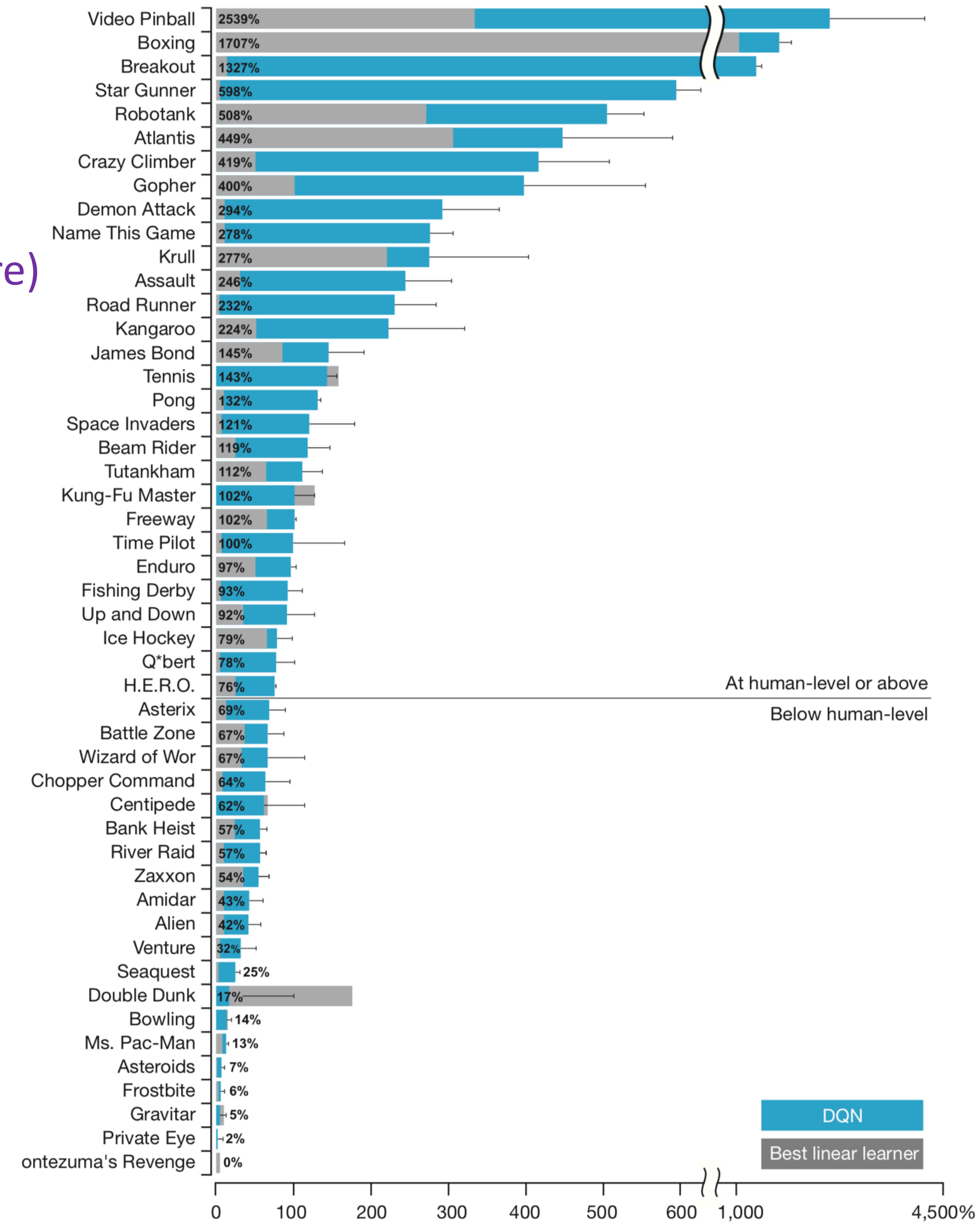
DQN Performance in Atari

- 100 x (DQN score - random play score)/ (human score - random play score)
- Score for each game is averaged over 30 sessions on each game, each lasting up to 5 minutes and beginning with a random initial game state
- The professional human tester played using the same emulator (with the sound turned off). After 2 hours of practice, the human played about 20 episodes of each game for up to 5 minutes each and was not allowed to take any break during this time
- DQN played better than the human player on 22 of the games.



DQN Performance in Atari

- 100 x (DQN score - random play score)/
  (human score - random play score)
- Score for each game is averaged over 30 sessions on each game, each lasting up to 5 minutes and beginning with a random initial game state
- The professional human tester played using the same emulator (with the sound turned off). After 2 hours of practice, the human played about 20 episodes of each game for up to 5 minutes each and was not allowed to take any break during this time
- DQN played better than the human player on 22 of the games.
- By considering any performance that scored at or above 75% of the human score to be comparable to, or better than, human-level play, Mnih et al. concluded that the levels of play DQN learned reached or exceeded human level on 29 of the 46 games



DQN Training

- Training time: Training over 50 million frames
  - 38 days of game experience in total
- Replay memory: Recent 1 million frames
- Minibatch size: 32
- Target network update frequency: After every 10k parameter updates
- Action repeat: Repeat the same action for k (= 4) frames
- SGD: RMSProp, with learning rate 0.00025
- Exploration: Epsilon-greedy policy
  - Epsilon decreasing from 1.0 to 0.1 over first million frames and then fixed after
- Discount factor: 0.99

Effect of Replay and Target Network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN vs Linear

Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3

DQN Algorithm

Code: https://sites.google.com/a/deepmind.com/dqn/

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

DQN Improvements

- Prioritized Experience Replay (Shaul et al, ICLR 2016)
  - In experience replay, experience transitions were uniformly sampled from a replay memory
  - However, some transitions may be more *informative* than others, but may occur less frequently
  - Uniform sampling may not select these experiences
  - Can we prioritize the samples to accelerate the learning progress?
  - Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)

DQN Improvements

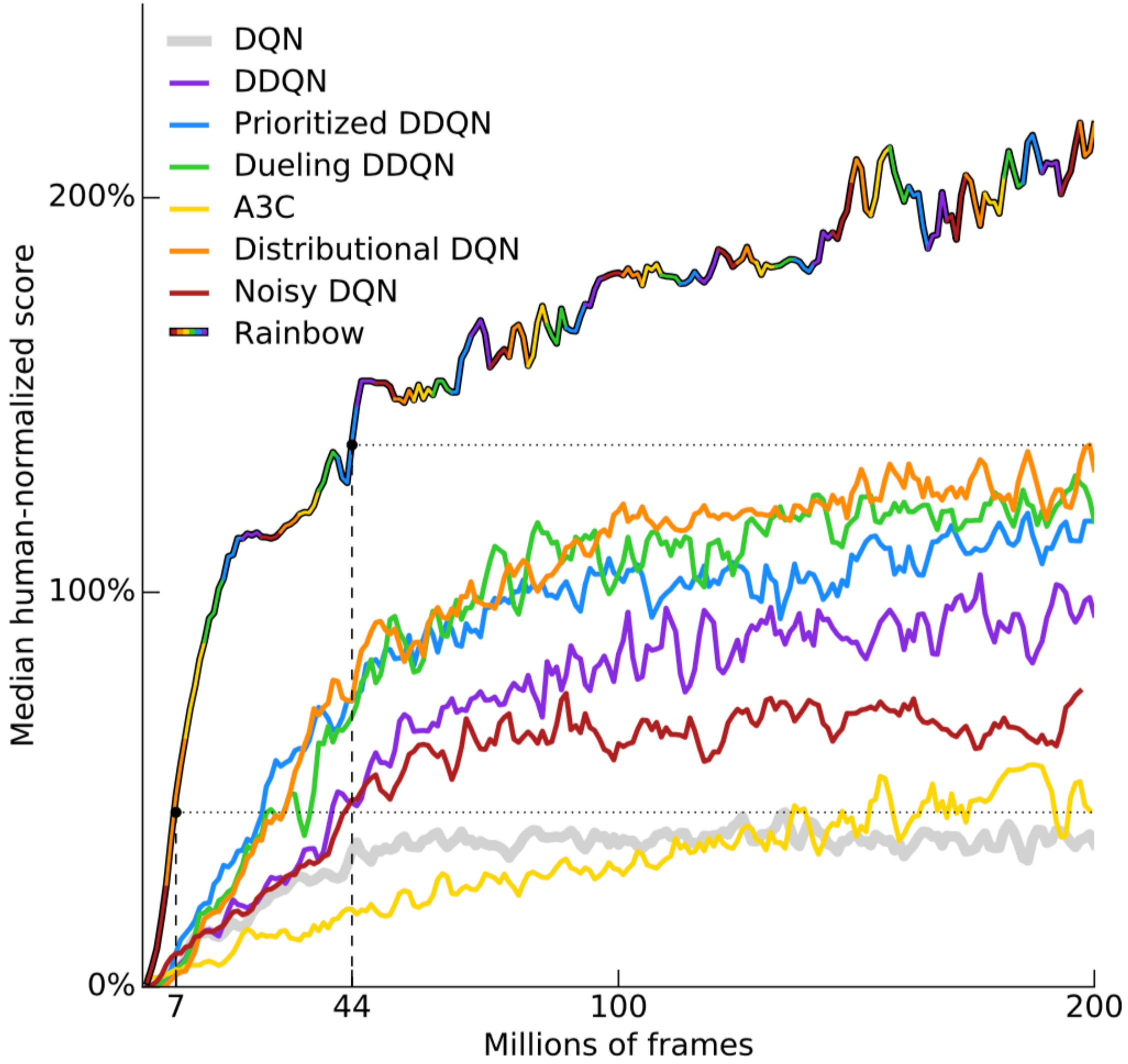
- Prioritized Experience Replay (Shaul et al, ICLR 2016)
  - In experience replay, experience transitions were uniformly sampled from a replay memory
  - However, some transitions may be more *informative* than others, but may occur less frequently
  - Uniform sampling may not select these experiences
  - Can we prioritize the samples to accelerate the learning progress?
  - Intuition: Prioritize a sample based on how much can learn from a transition (expected learning progress)

- Double DQN (Van Hasselt, Guez and Silver, AAAI 2026)
  - Overcome the maximization bias in Q-learning

DQN Improvements

- Dueling DQN (Wang et al, ICML 2016)
  - Two streams to separately estimate (scalar) state-value and the advantages for each action
- Noisy Networks for Exploration (Fortunato sselt, ICLR 2018)
  - Exploration via adding noise to the neural network parameters
- Distributional Reinforcement Learning (Bellemare, ICML, 2018)
  - Tracks the distribution of the Q-values instead of a point estimate
- Rainbow (Hessel et al, AAAI 2018)
  - Combining all the DQN improvements

DQN Rainbow Performance



Recent Improvements

"Agent57: Outperforming the human Atari benchmark"
https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark

"We've developed Agent57, the first deep reinforcement learning agent to obtain a score that is above the human baseline on all 57 Atari 2600 games"

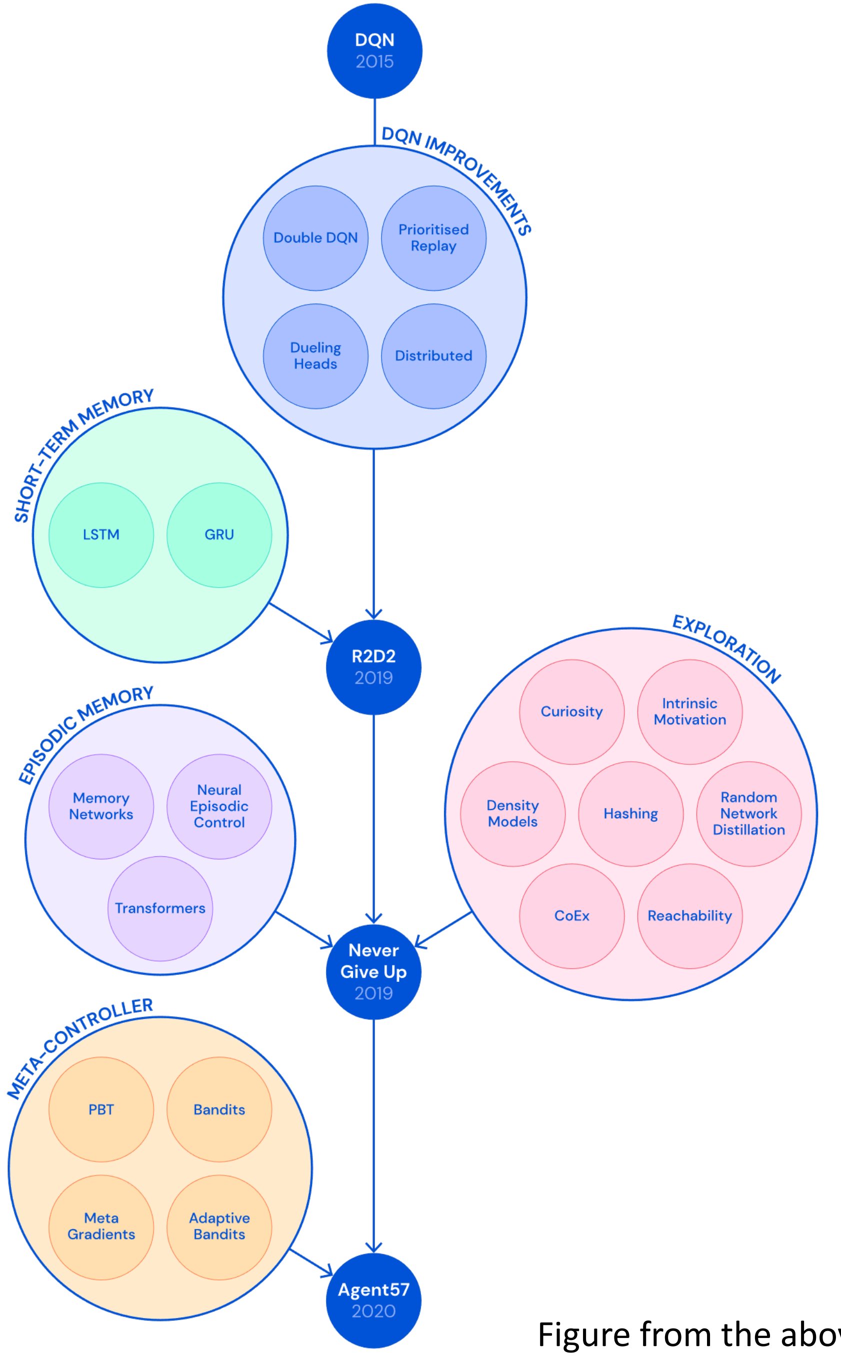


Figure from the above link

Policy Gradient Algorithms

Reinforcement Learning

1. How to evaluate a policy?
2. How to compute the optimal value function V^*?
3. How to compute the optimal policy π^*?

When the system model is unknown

- Q Learning: Learn optimal value (Q-value) function
  - Compute the policy from the learned Q-function
- Can we learn the policy directly?

Reinforcement Learning

1. How to evaluate a policy?
2. How to compute the optimal value function V^*?
3. How to compute the optimal policy π^*?

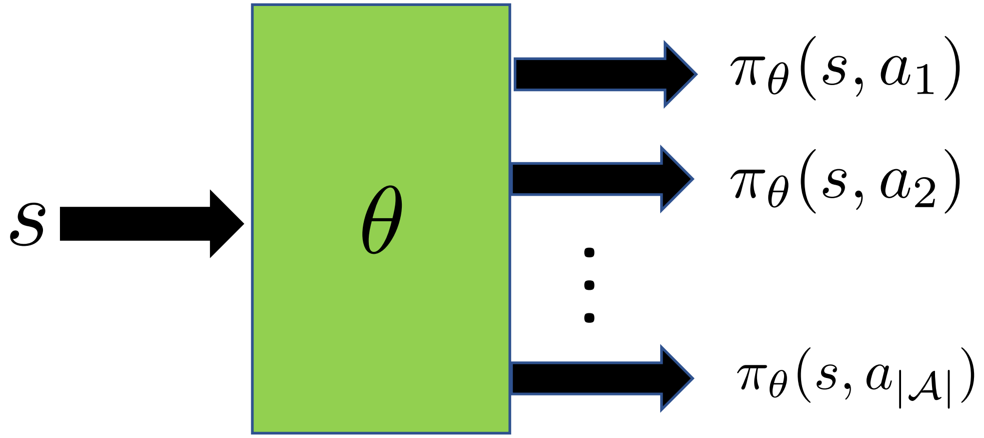
When the system model is unknown

- Q Learning: Learn optimal value (Q-value) function
  - Compute the policy from the learned Q-function
- Can we learn the policy directly?

Policy Gradient Algorithms

Parametric Policy

- Consider a (stochastic) policy $\pi_\theta(s, a)$



Parametric Policy

- Objective: Find the optimal policy π^* that maximizes V_π

Parametric Policy

- Objective: Find the optimal policy π^* that maximizes V_π
- Consider a (stochastic) policy $\pi_\theta(s, a)$
- Define $J(\theta) = V_{\pi_\theta}$
- Goal: Find $\theta^* = \arg\max_\theta J(\theta)$

Parametric Policy

- Objective: Find the optimal policy π^* that maximizes V_π
- Consider a (stochastic) policy $\pi_\theta(s, a)$
- Define $J(\theta) = V_{\pi_\theta}$
- Goal: Find $\theta^* = \arg\max_\theta J(\theta)$

- Parametric policy examples:
  - Softmax policy: $\pi_\theta(s, a) = \frac{e^{(\phi(s,a)^\top \theta)}}{\sum_b e^{(\phi(s,b)^\top \theta)}}$
  - Gaussian policy (for continuous action space): $a \sim \mathcal{N}(\phi(s)^\top \theta, \sigma^2)$

Policy Gradient

- Objective: Find the optimal policy π^* that maximizes V_π
- Consider a (stochastic) policy $\pi_\theta(s, a)$
- Define $J(\theta) = V_{\pi_\theta}$
- Goal: Find $\theta^* = \arg\max_\theta J(\theta)$

- Policy gradient intuition: $\theta = \theta + \alpha \nabla J(\theta)$
  - Will this converge?
  - How do we estimate the gradient?

Why Policy Gradient?

- Advantages:
  - In many problems, policy may be a simpler function to approximate
  - Choice of policy parameterization is a good way of incorporating domain knowledge about the system into the reinforcement learning algorithm

Why Policy Gradient?

- Advantages:
  - In many problems, policy may be a simpler function to approximate
  - Choice of policy parameterization is a good way of incorporating domain knowledge about the system into the reinforcement learning algorithm
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies

Why Policy Gradient?

- Advantages:
  - In many problems, policy may be a simpler function to approximate
  - Choice of policy parameterization is a good way of incorporating domain knowledge about the system into the reinforcement learning algorithm
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies

- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Why it is difficult?

- Value of a policy depends on both action selections (policy) and the (stationary) distribution of the states resulting from the policy

Why it is difficult?

- Value of a policy depends on both action selections (policy) and the (stationary) distribution of the states resulting from the policy
- Given a state s, $\nabla \pi_\theta(s, \cdot)$ can be computed

Why it is difficult?

- Value of a policy depends on both action selections (policy) and the (stationary) distribution of the states resulting from the policy
- Given a state s, $\nabla\pi_\theta(s,\cdot)$ can be computed
- But, the gradient of the (stationary) distribution of the states induced by the policy cannot be computed easily (why?)

Why it is difficult?

- Value of a policy depends on both action selections (policy) and the (stationary) distribution of the states resulting from the policy
- Given a state s, $\nabla \pi_\theta(s, \cdot)$ can be computed
- But, the gradient of the (stationary) distribution of the states induced by the policy cannot be computed easily
  - Requires the knowledge of the transition probability
  - Transition probability is unknown!

Why it is difficult?

- Value of a policy depends on both action selections (policy) and the (stationary) distribution of the states resulting from the policy
- Given a state s, $\nabla \pi_\theta(s, \cdot)$ can be computed
- But, the gradient of the (stationary) distribution of the states induced by the policy cannot be computed easily
  - Requires the knowledge of the transition probability
  - Transition probability is unknown!
- How do we estimate the gradient?

Policy Gradient Theorem

Theorem 2. *For infinite horizon discounted reward MDP,*

$$\nabla J(\theta) = \sum_{s} \mu_{\pi_\theta}(s) \sum_{a} Q_{\pi_\theta}(s,a)\ \nabla \pi_\theta(s,a)$$

where, $\mu_{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t\ \mathbb{P}(s_t = s|s_0)$. *This can be represented as*

$$\nabla J(\theta) = \mathbb{E}_{s\sim\mu_{\pi_\theta(\cdot)}}\ \mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[Q_{\pi_\theta}(s,a)\nabla \log \pi_\theta(s,a)\right] = \mathbb{E}_{\pi_\theta}\left[Q_{\pi_\theta}(s,a)\nabla \log \pi_\theta(s,a)\right]$$

Policy Gradient Theorem

Theorem 2. *For infinite horizon discounted reward MDP,*

$$\nabla J(\theta) = \sum_s \mu_{\pi_\theta}(s) \sum_a Q_{\pi_\theta}(s,a)\ \nabla \pi_\theta(s,a)$$

where, $\mu_{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t\ \mathbb{P}(s_t = s|s_0)$. *This can be represented as*

$$\nabla J(\theta) = \mathbb{E}_{s\sim\mu_{\pi_\theta(\cdot)}}\ \mathbb{E}_{a\sim\pi_\theta(s,\cdot)}\left[Q_{\pi_\theta}(s,a)\nabla \log \pi_\theta(s,a)\right] = \mathbb{E}_{\pi_\theta}\left[Q_{\pi_\theta}(s,a)\nabla \log \pi_\theta(s,a)\right]$$

- Doesn't depend on the gradient of the (stationary) distribution induced by the policy!
- Can be estimated from the sample trajectories

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Can we do the update

$$\theta = \theta + \alpha\ Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))$$

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Can we do the update

$$\theta = \theta + \alpha\ Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))$$

Yes, but what is the w to be used?

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Can we do the update

$$\theta = \theta + \alpha\ Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))$$

Yes, but what is the w to be used?

- w should give a good approximation $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Can we do the update

$$\theta = \theta + \alpha\ Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))$$

Yes, but what is the w to be used?

- w should give a good approximation $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$
- Q_{π_θ} is unknown. So, need to learn w

Actor-Critic Algorithm

- We have $\nabla J(\theta) = \mathbb{E}\left[Q_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$

- Approximate Q function: $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$. Then,

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Can we do the update

$$\theta = \theta + \alpha\ Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))$$

Yes, but what is the w to be used?

- w should give a good approximation $Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$
- Q_{π_θ} is unknown. So, need to learn w
- θ changing. So, w should also change

Actor-Critic Algorithm

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Critic: Updates Q-value parameter *w*

  (*Evaluate* the policy corresponding to θ)

- Actor: Update policy parameter θ

  (*Improve* the policy in the direction suggested by critic)

Actor-Critic Algorithm

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Critic: Updates Q-value parameter *w*

  (*Evaluate* the policy corresponding to θ)

  Can be done by TD

- Actor: Update policy parameter θ

  (*Improve* the policy in the direction suggested by critic)

Actor-Critic Algorithm

$$\nabla J(\theta) \approx \mathbb{E}\left[Q_w(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

- Critic: Updates Q-value parameter *w*

  (*Evaluate* the policy corresponding to θ)

  Can be done by TD

- Actor: Update policy parameter θ

  (*Improve* the policy in the direction suggested by critic)

for each step **do**

$\theta = \theta + \alpha_\theta\ Q_w(s,a)\ \nabla_\theta \log(\pi_\theta(s,a))$

Sample the next state s', sample the next action $a' \sim \pi_\theta(s',\cdot)$

$\delta = R(s,a) + \gamma\ Q_w(s',a') - Q_w(s,a)$

$w = w + \alpha_w\ \delta\ \nabla_w Q_w(s,a)$

end for

Advantage Actor-Critic

- Reduce variance using baseline: $\nabla J(\theta) = \mathbb{E}\left[\left(Q_{\pi_\theta}(s,a) - b(s)\right) \nabla \log(\pi_\theta(s,a))\right]$

Advantage Actor-Critic

- Reduce variance using baseline: $\nabla J(\theta) = \mathbb{E}\left[\left(Q_{\pi_\theta}(s,a) - b(s)\right) \nabla \log(\pi_\theta(s,a))\right]$

- Value function is a good baseline: $\nabla J(\theta) = \mathbb{E}\left[\left(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)\right) \nabla \log(\pi_\theta(s,a))\right]$

Advantage Actor-Critic

- Reduce variance using baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - b(s)) \nabla \log(\pi_\theta(s,a)) \right]$

- Value function is a good baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s)) \nabla \log(\pi_\theta(s,a)) \right]$

Advantage Actor-Critic

- Reduce variance using baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - b(s))\ \nabla \log(\pi_\theta(s,a))\right]$
- Value function is a good baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))\ \nabla \log(\pi_\theta(s,a))\right]$

Advantage Function

Advantage Actor-Critic

- Reduce variance using baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - b(s))\ \nabla \log(\pi_\theta(s,a))\right]$
- Value function is a good baseline: $\nabla J(\theta) = \mathbb{E}\left[(Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s))\ \nabla \log(\pi_\theta(s,a))\right]$

$$\nabla J(\theta) = \mathbb{E}\left[A_{\pi_\theta}(s,a)\ \nabla \log(\pi_\theta(s,a))\right]$$

Advantage Function

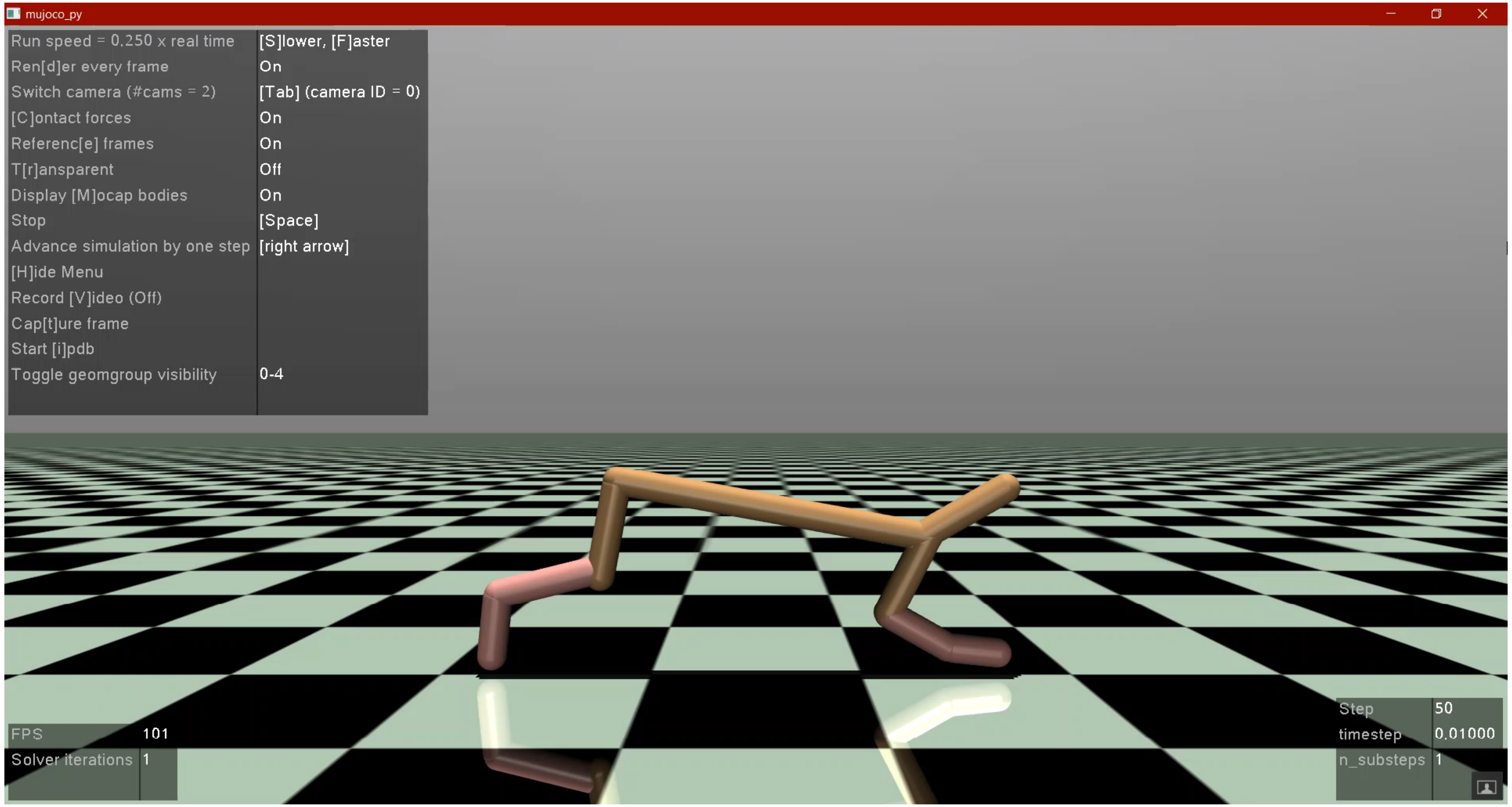
Advanced Policy Gradient Algorithms

- Natural Policy Gradient Algorithm (Kakade, NIPS, 2002)

- Trust Region Policy Optimization (TRPO) (Schulman et al, ICML, 2015)

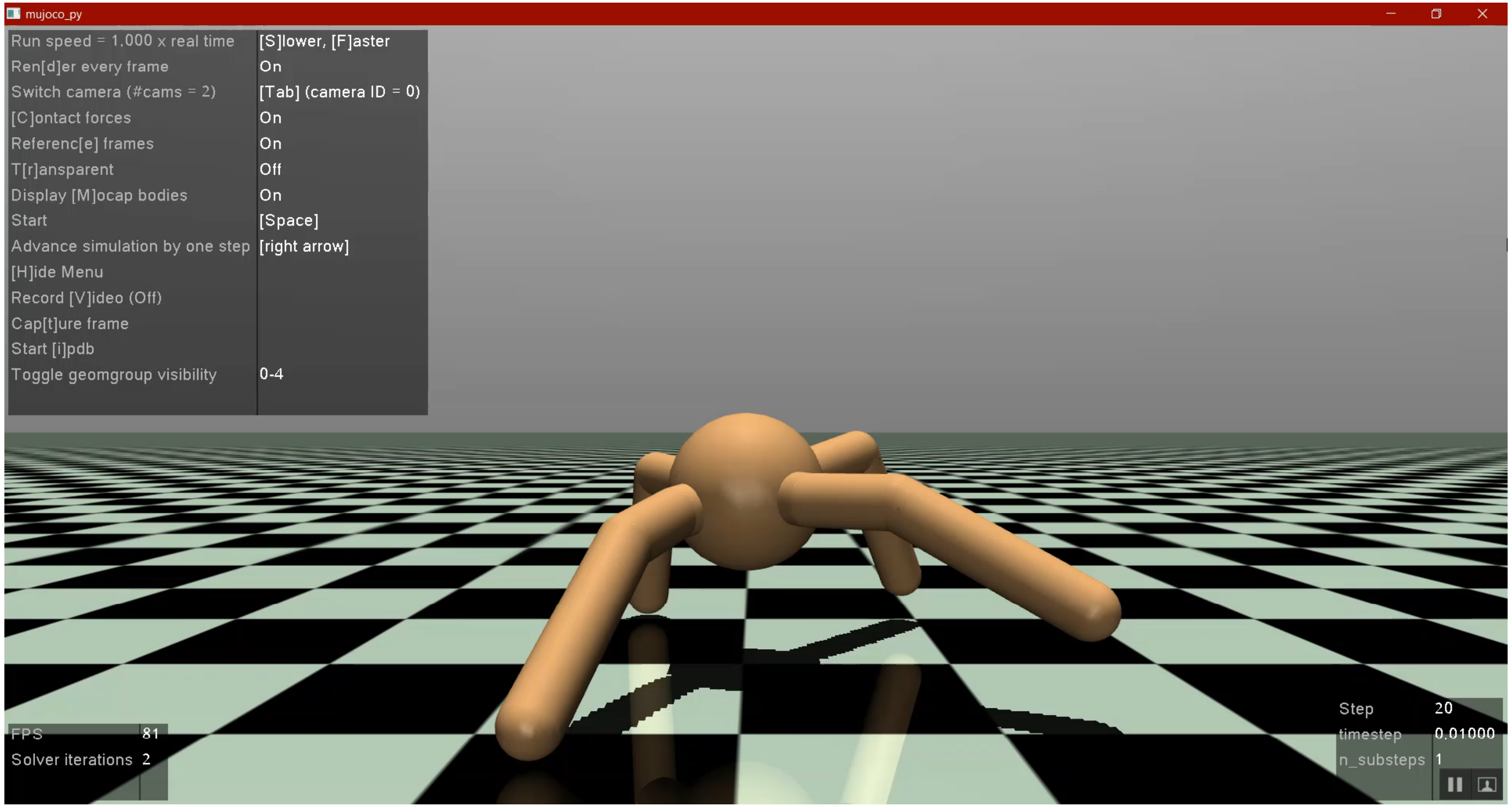
- Proximal Policy Optimization (PPO) (Schulman et al, 2017)

Some Examples with PPO



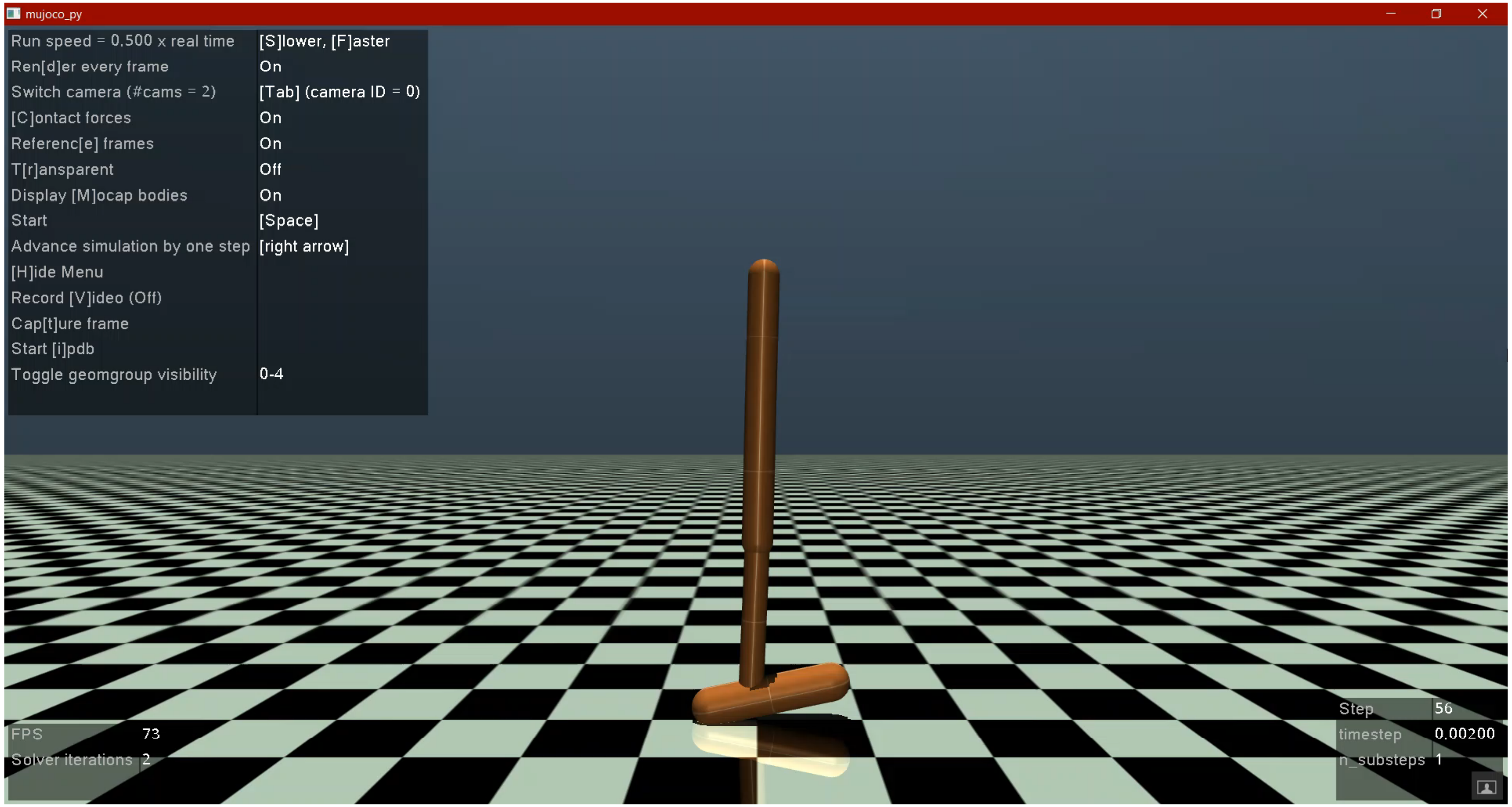
Trained by Rayan El Helo (ECE, TAMU)

Some Examples with PPO



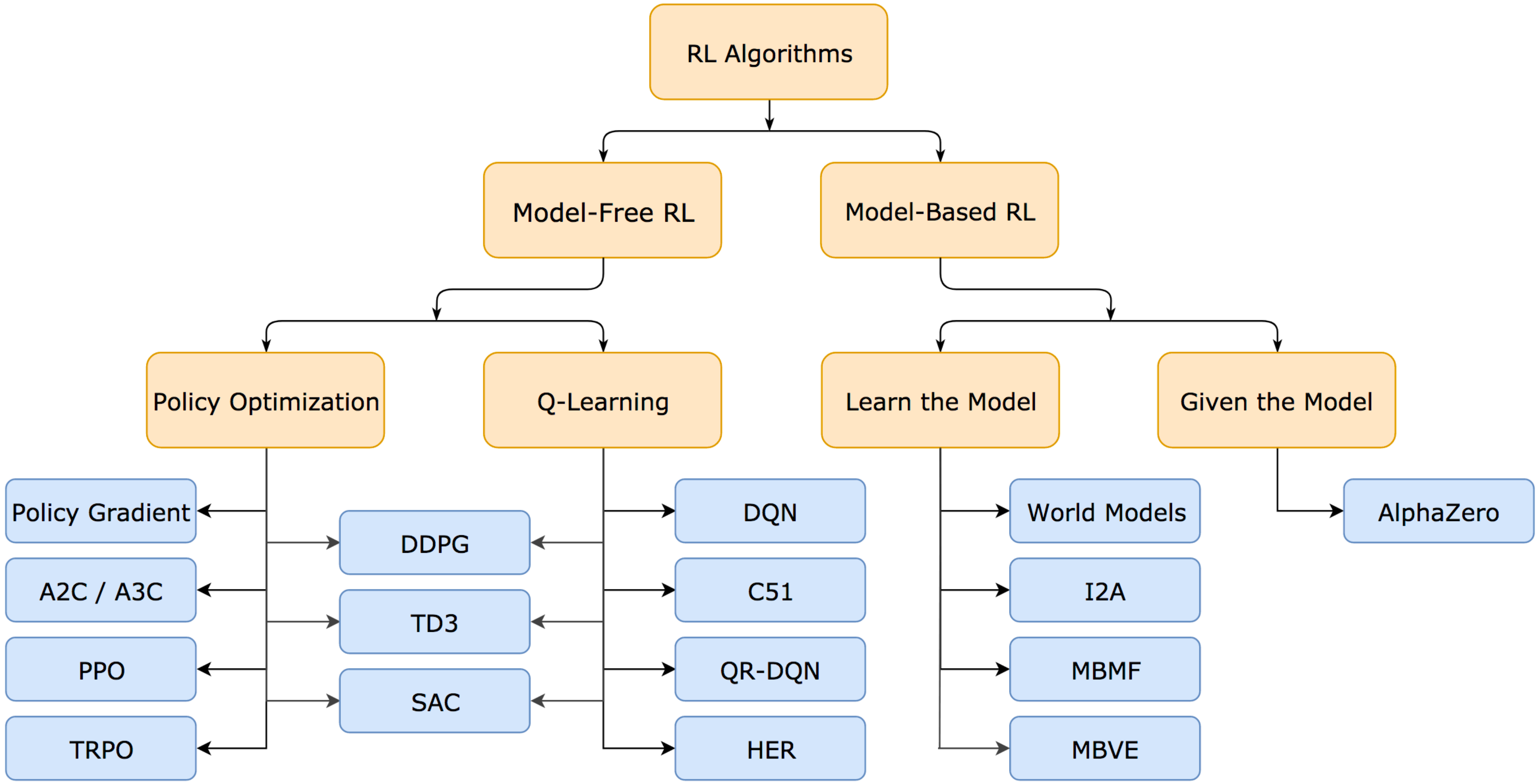
Trained by Rayan El Helo (ECE, TAMU)

Some Examples with PPO



Trained by Rayan El Helo (ECE, TAMU)

Taxonomy of RL Algorithms



[1] https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#id20

Conclusion

- RL provides a general-purpose framework for AI
- RL problems can be solved by end-to-end deep learning
- Reinforcement Learning + Deep Learning = AI ?

Conclusion

- RL is a very active research area!!
  - How do we learn fast? (RL is infamous for being data hungry)
  - How do we learn safely? (Don't want my drone to crash during training/testing)
  - How do we use *memory* for transfer/meta learning? (Learning one task should be useful to execute other tasks)
  - How do we represent and learn hierarchical features? (Breaking down a very large task to simple tasks, to reduce the complexity)
  - How do we learn from an expert? (Can the AI agent learn from human agents' demonstrations?)
  - How do we learn with multiple agents? (Especially if the agents are rational and selfish)
- Would you like to do some cool research on RL?
  - Please send me an email!