Reinforcement Learning

How Machines learn like Humans do

PD Dr.-Ing. Marco Huber

Katana
USU Software AG
Karlsruhe
http://katana.usu.de

Interactive Vision and Fusion Lab
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT)
http://www.ies.uka.de

karlsruhe.ai

How do we learn new behaviors?

How do we learn new behaviors?

Positive Reinforcement

(Reward)

How do we learn new behaviors?

Positive Reinforcement

(Reward)

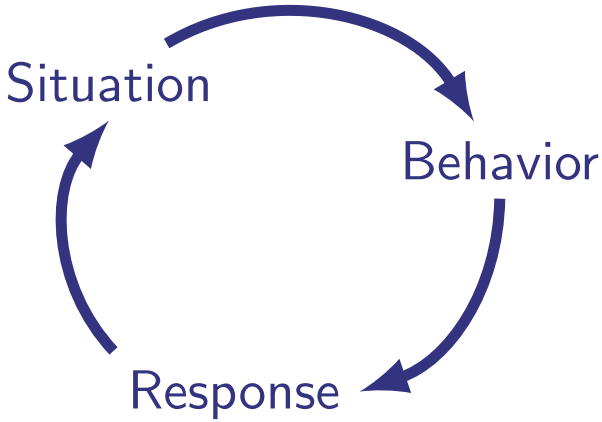
Negative Reinforcement

How do we learn new behaviors?

Law of Effect

Behaviors that are followed by "responses that produce a satisfying effect in a particular **situation** become more likely to occur again in that situation [as the association between situation and behavior is **reinforced**], and responses that produce a discomforting effect become less likely to occur again in that situation."

Edward L. Thorndike, 1898



Experimental Validation

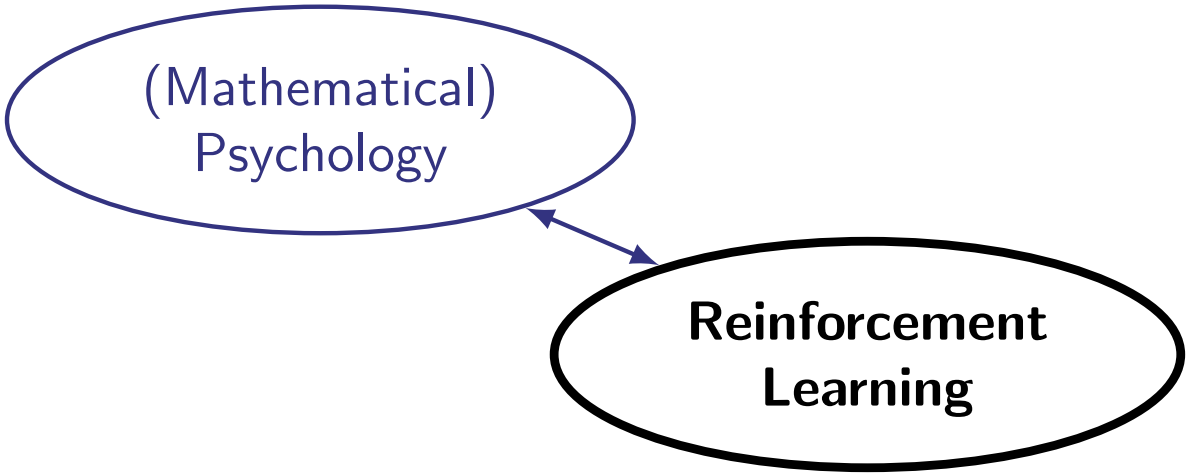
- Pavlov's dogs, 1905
- Skinner box, 1930

Reinforcement Learning (RL)

Goal

Use response (*reinforcement*) of the environment in order to learn what to do in particular situations.

➔ Mapping of situations to actions

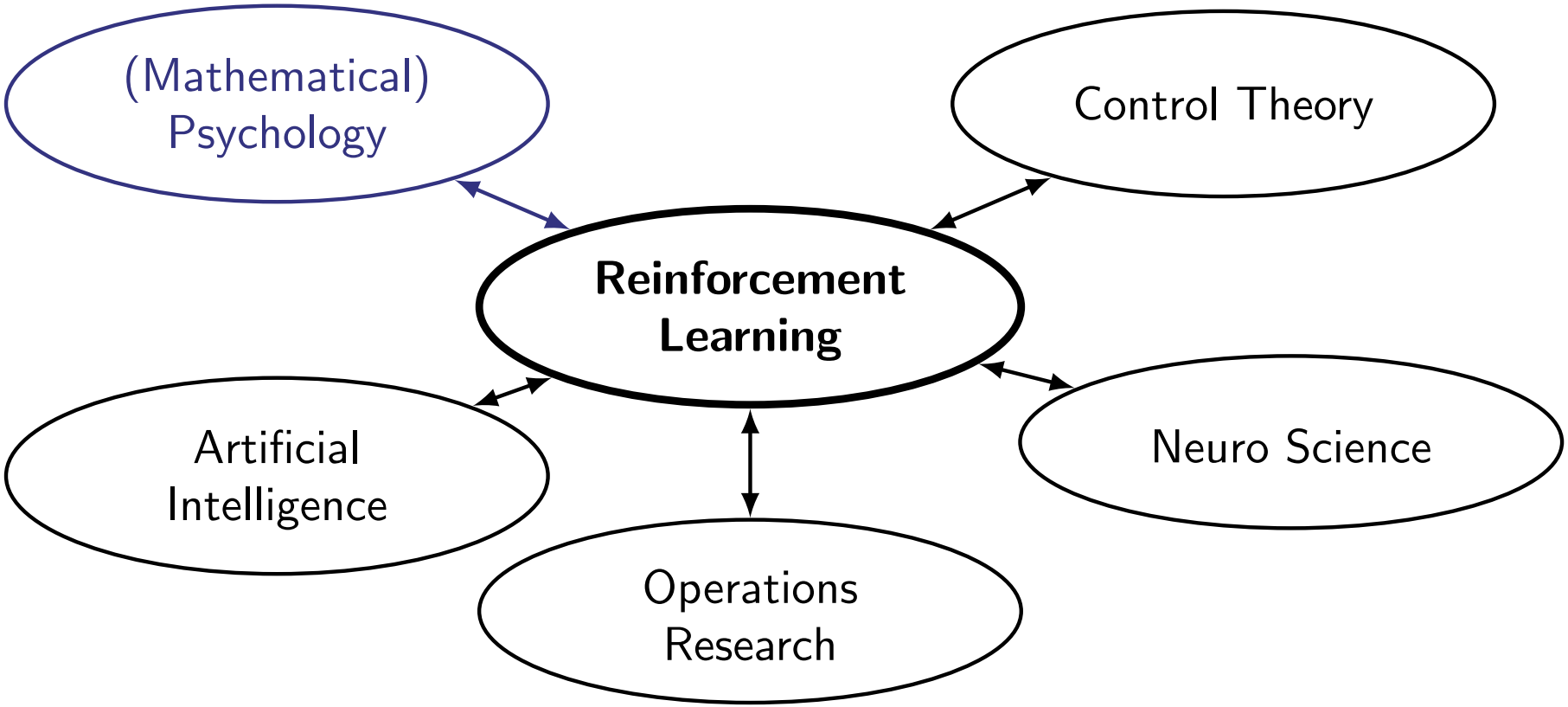


Reinforcement Learning (RL)

Goal

Use response (*reinforcement*) of the environment in order to learn what to do in particular situations.

➔ Mapping of situations to actions



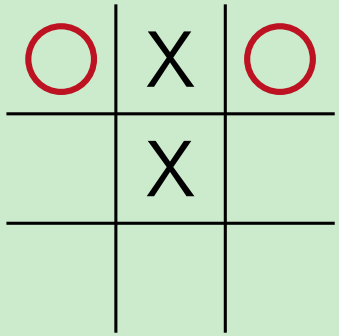
Reinforcement Learning (RL)

Goal

Use response (*reinforcement*) of the environment in order to learn what to do in particular situations.

➔ Mapping of situations to actions

Example: Tic-Tac-Toe



- $3^9 = 19,683$ different situations/states
- $9! = 362,880$ different games
- Opponent is a source of randomness

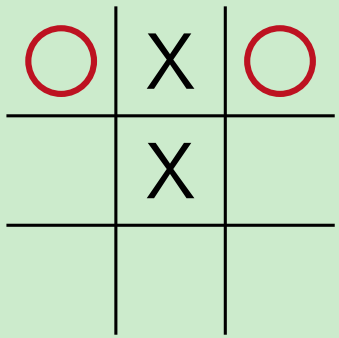
Reinforcement Learning (RL)

Goal

Use response (*reinforcement*) of the environment in order to learn what to do in particular situations.

➔ Mapping of situations to actions

Example: Tic-Tac-Toe



- $3^9 = 19,683$ different situations/states
- $9! = 362,880$ different games
- Opponent is a source of randomness

Problems of Classical Solution Approaches

- High programming effort.
- Problem-specific solution, limited generalization.
- Optimal solution not guaranteed. Intuition often misleading.

First RL Algorithm for Tic-Tac-Toe

MENACE (Matchbox Educable Noughts And Crosses Engine)

- Donald Michie, 1963
- Learns to play Tic-Tac-Toe optimally
- 304 different matchboxes
- Beans in 9 different colors

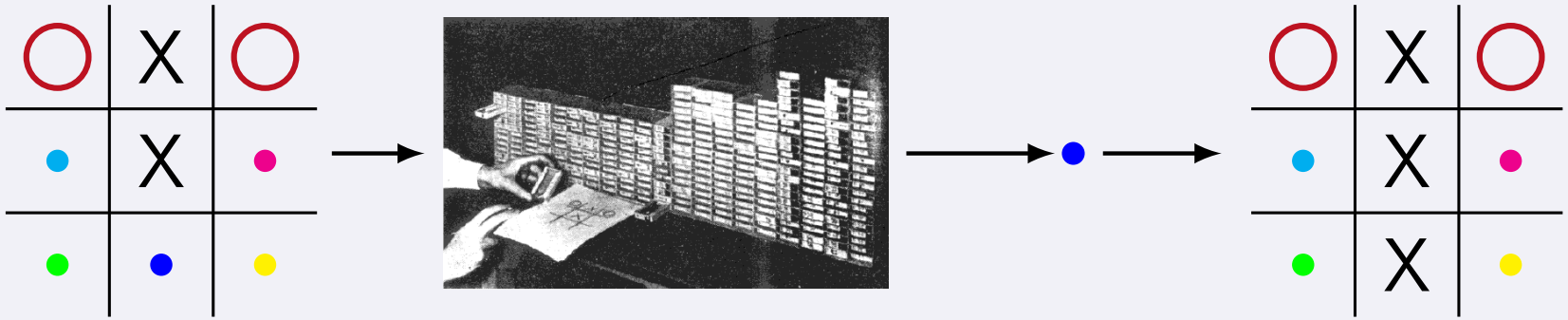
http://shorttermmemoryloss.com/menace/

First RL Algorithm for Tic-Tac-Toe

MENACE (Matchbox Educable Noughts And Crosses Engine)

- Donald Michie, 1963
- Learns to play Tic-Tac-Toe optimally
- 304 different matchboxes
- Beans in 9 different colors

Procedure



First RL Algorithm for Tic-Tac-Toe

MENACE (Matchbox Educable Noughts And Crosses Engine)

- Donald Michie, 1963
- Learns to play Tic-Tac-Toe optimally
- 304 different matchboxes
- Beans in 9 different colors

http://shorttermmemoryloss.com/menace/

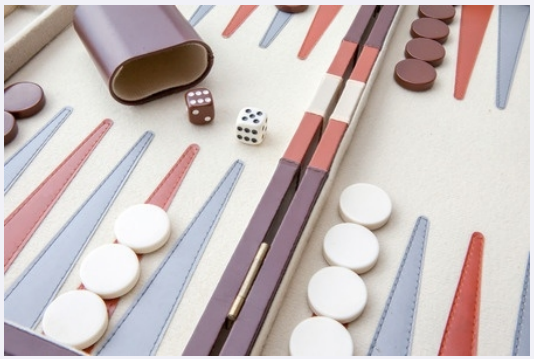
Evaluation

Win	removed beans plus one additional bean of the same color are placed in the opened boxes (positive reinforcement)
Loss	removed beans are retained (negative reinforcement)
Draw	put beans back to boxes (no change)

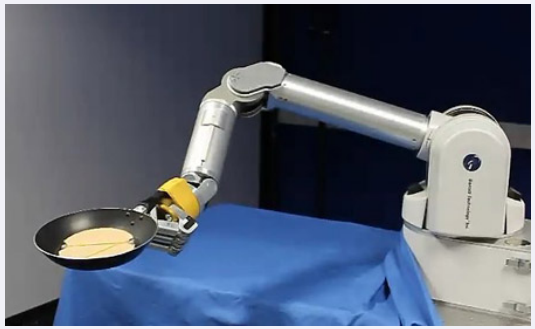
➔ Increases probability of repeating a successful move

Application Examples

Games



Robotics



http://programming-by-demonstration.org

UAV Maneuvering

http://heli.stanford.edu

Trading

Optimal Control

https://wp.doc.ic.ac.uk/sml/

Elevator Scheduling

Formularization

Markov Decision Process (MDP)

States x	belong to finite set $\mathcal{X} = \{1, 2, \ldots, N\}$
Actions a	belong to finite set $\mathcal{A} = \{1, 2, \ldots, M\}$
Time t	discrete time steps with $t = 0, 1, 2, \ldots$
Model	transition probabilities $P(x_{t+1}\|x_t, a_t)$
Reward	according to a reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Assumption	Markov property holds ➔ reward and state transition depend only on previous state

Formularization

Markov Decision Process (MDP)

States x	belong to finite set $\mathcal{X} = \{1, 2, \ldots, N\}$
Actions a	belong to finite set $\mathcal{A} = \{1, 2, \ldots, M\}$
Time t	discrete time steps with $t = 0, 1, 2, \ldots$
Model	transition probabilities $P(x_{t+1} \mid x_t, a_t)$
Reward	according to a reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Assumption	Markov property holds ➔ reward and state transition depend only on previous state

Example: MENACE

States x	matchboxes
Actions a	placement of symbols/removal of beans
Time t	rounds
Model	rules + opponent
Reward	Beans added/retained at end of game

Formularization

Markov Decision Process (MDP)

States x	belong to finite set $\mathcal{X} = \{1, 2, \ldots, N\}$
Actions a	belong to finite set $\mathcal{A} = \{1, 2, \ldots, M\}$
Time t	discrete time steps with $t = 0, 1, 2, \ldots$
Model	transition probabilities $P(x_{t+1}\|x_t, a_t)$
Reward	according to a reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Assumption	Markov property holds ➔ reward and state transition depend only on previous state

Goal

Use response (reinforcement = **reward**) of the environment in order to learn what to do in particular situations = **states**.

➔ Mapping of situations to actions

Formularization

Markov Decision Process (MDP)

States x	belong to finite set $\mathcal{X} = \{1, 2, \ldots, N\}$
Actions a	belong to finite set $\mathcal{A} = \{1, 2, \ldots, M\}$
Time t	discrete time steps with $t = 0, 1, 2, \ldots$
Model	transition probabilities $P(x_{t+1} \mid x_t, a_t)$
Reward	according to a reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Assumption	Markov property holds ➔ reward and state transition depend only on previous state

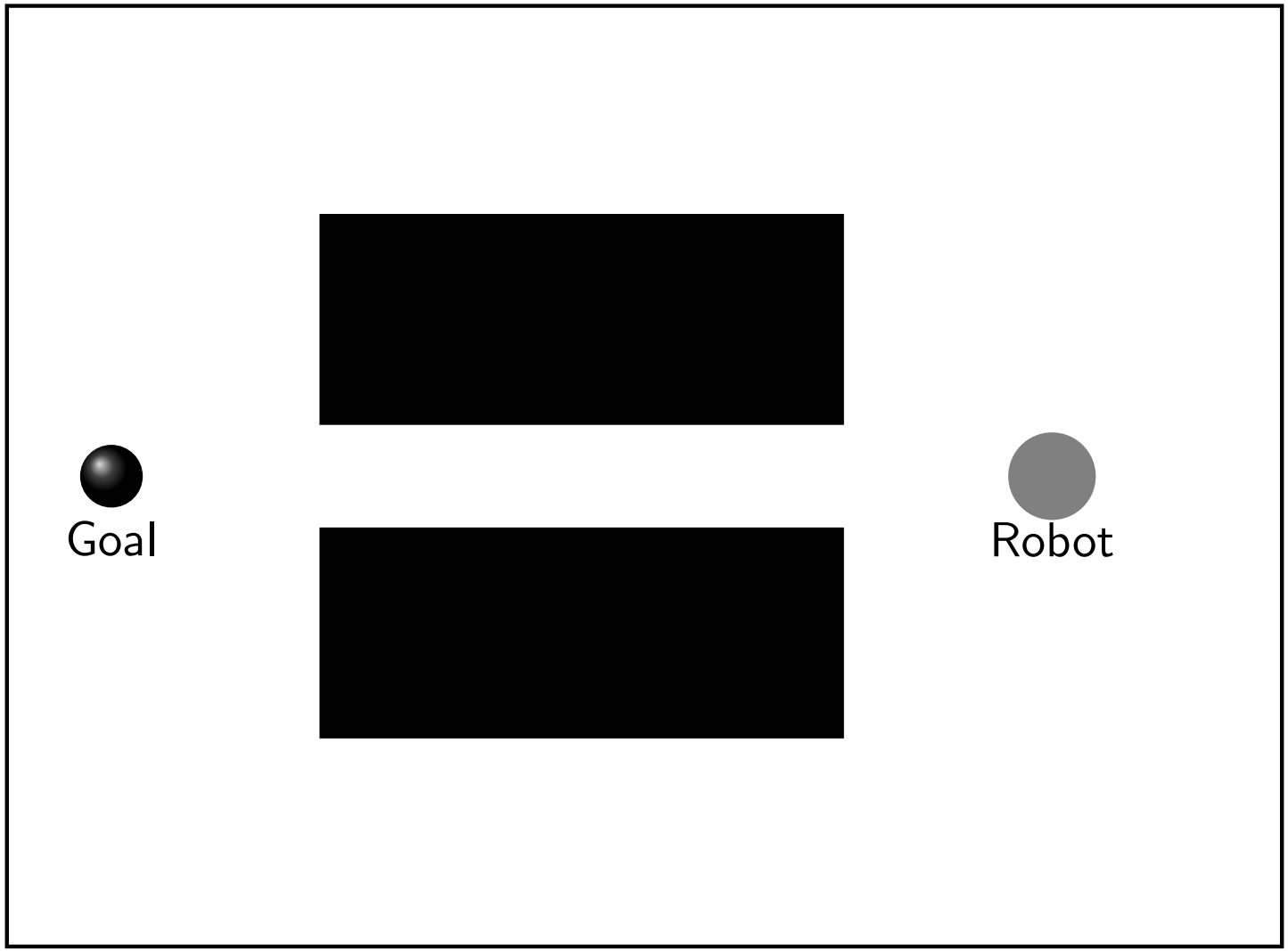
Goal: Optimal Policy

Find the *Policy* $\pi : \mathcal{X} \to \mathcal{A}$ that maximizes the expected cumulative reward

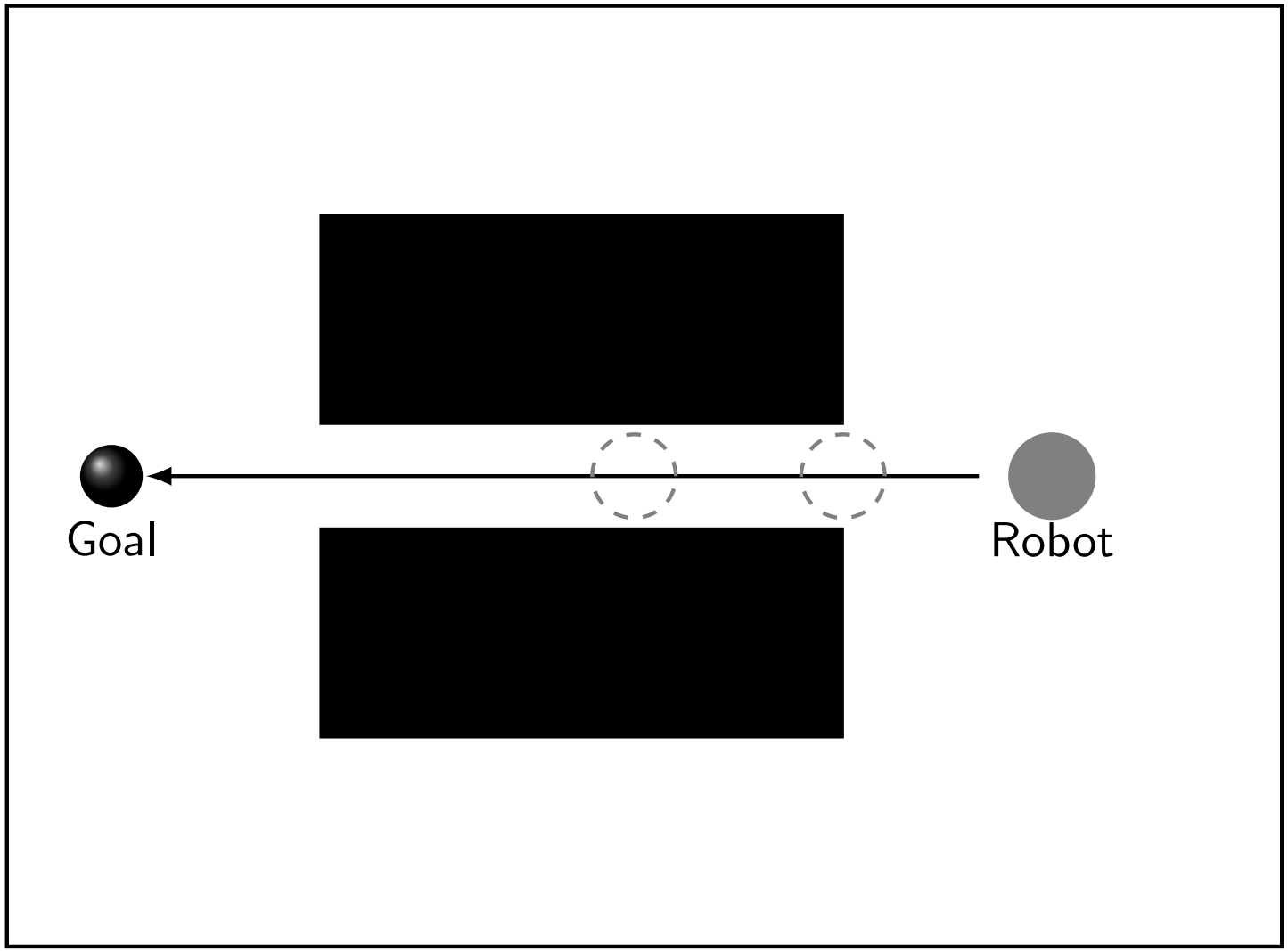
$$V_\pi(x) := \mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot r(x_t, \pi(x_t)) \;\middle|\; x_0 = x \right\}, \qquad \textbf{(value function)}$$

where $\gamma \in [0, 1)$ is a *discount factor*.

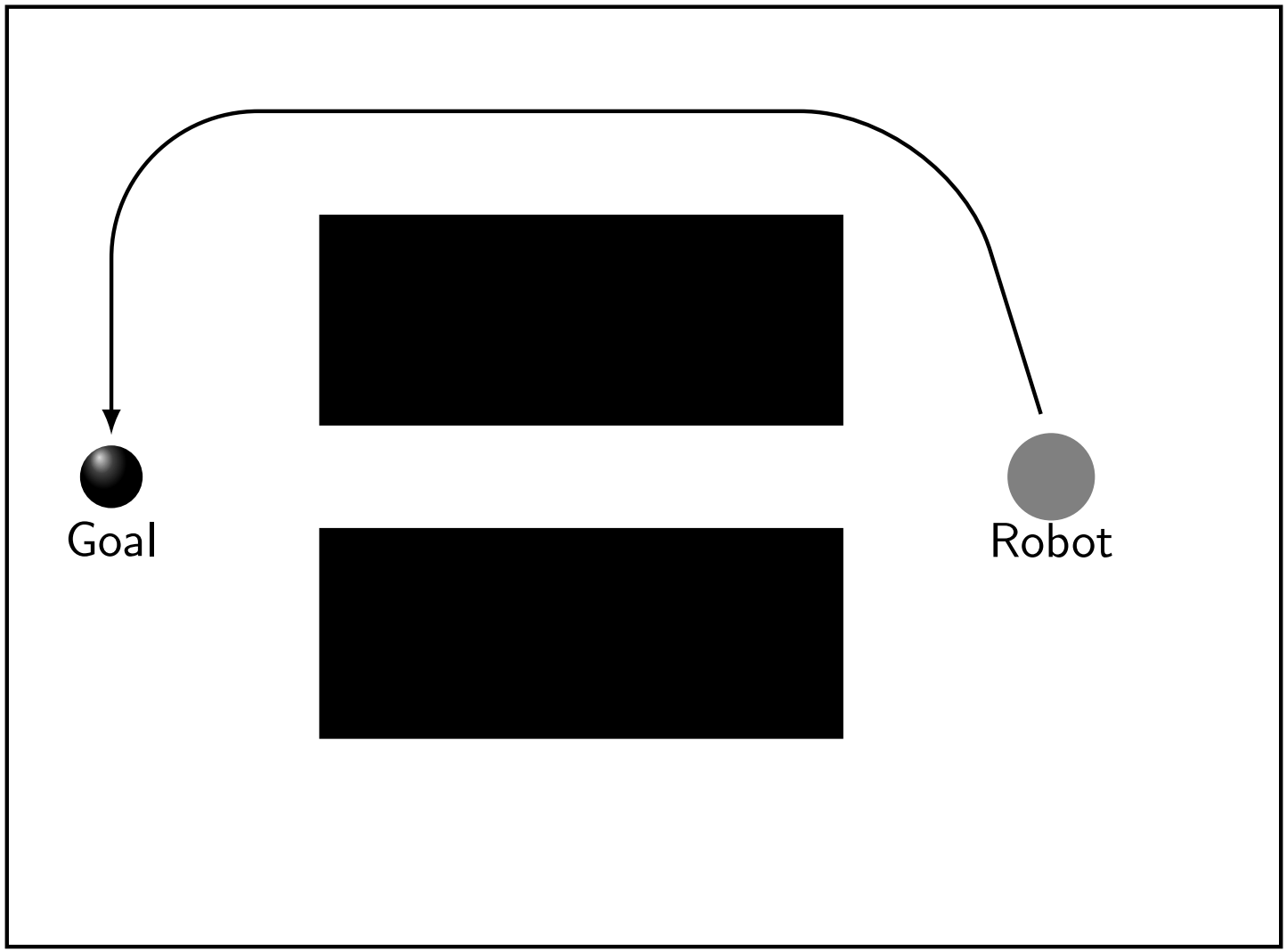
Policy vs Plan: Deterministic



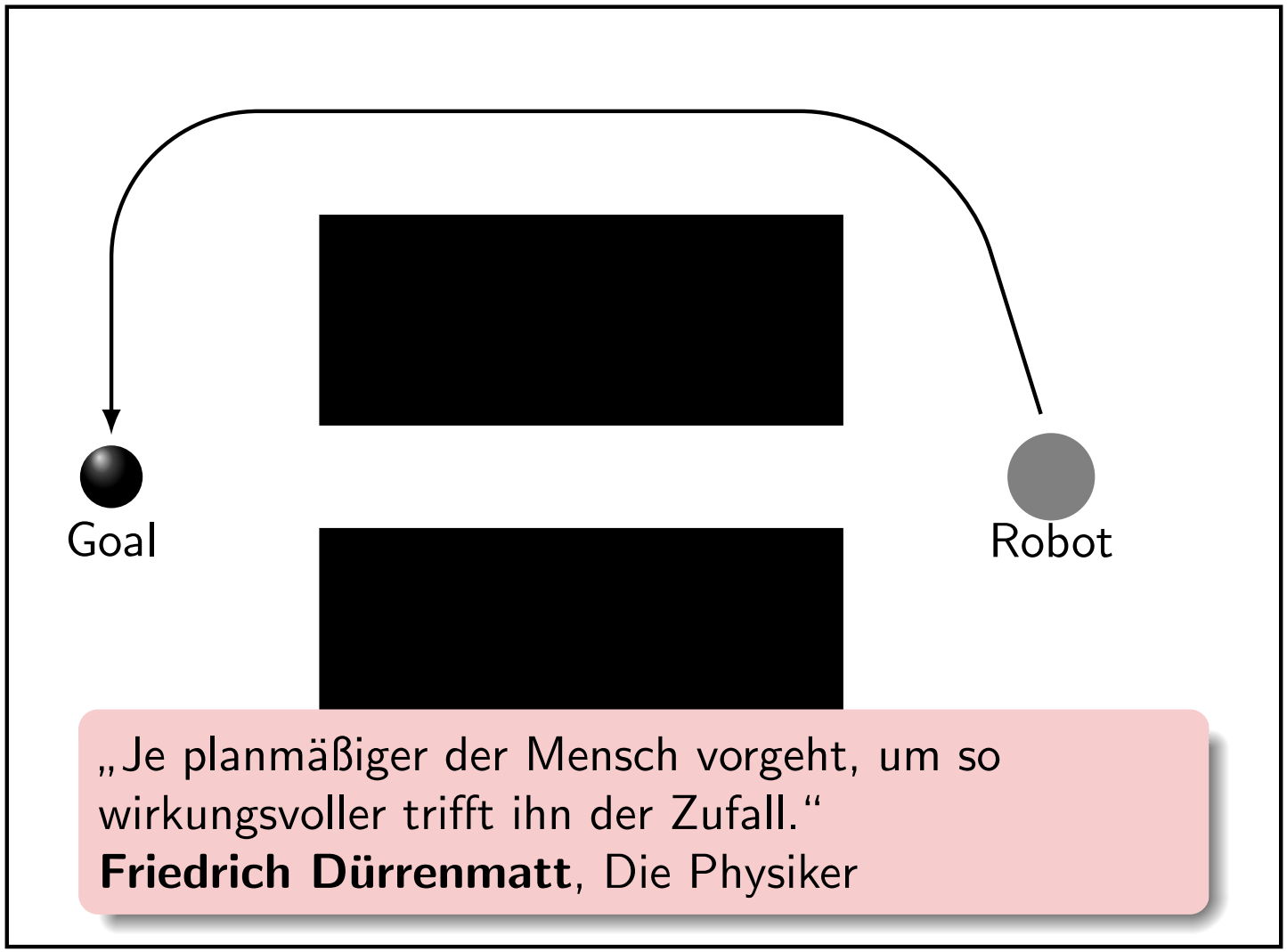
Policy vs Plan: Deterministic



Policy vs Plan: Plan

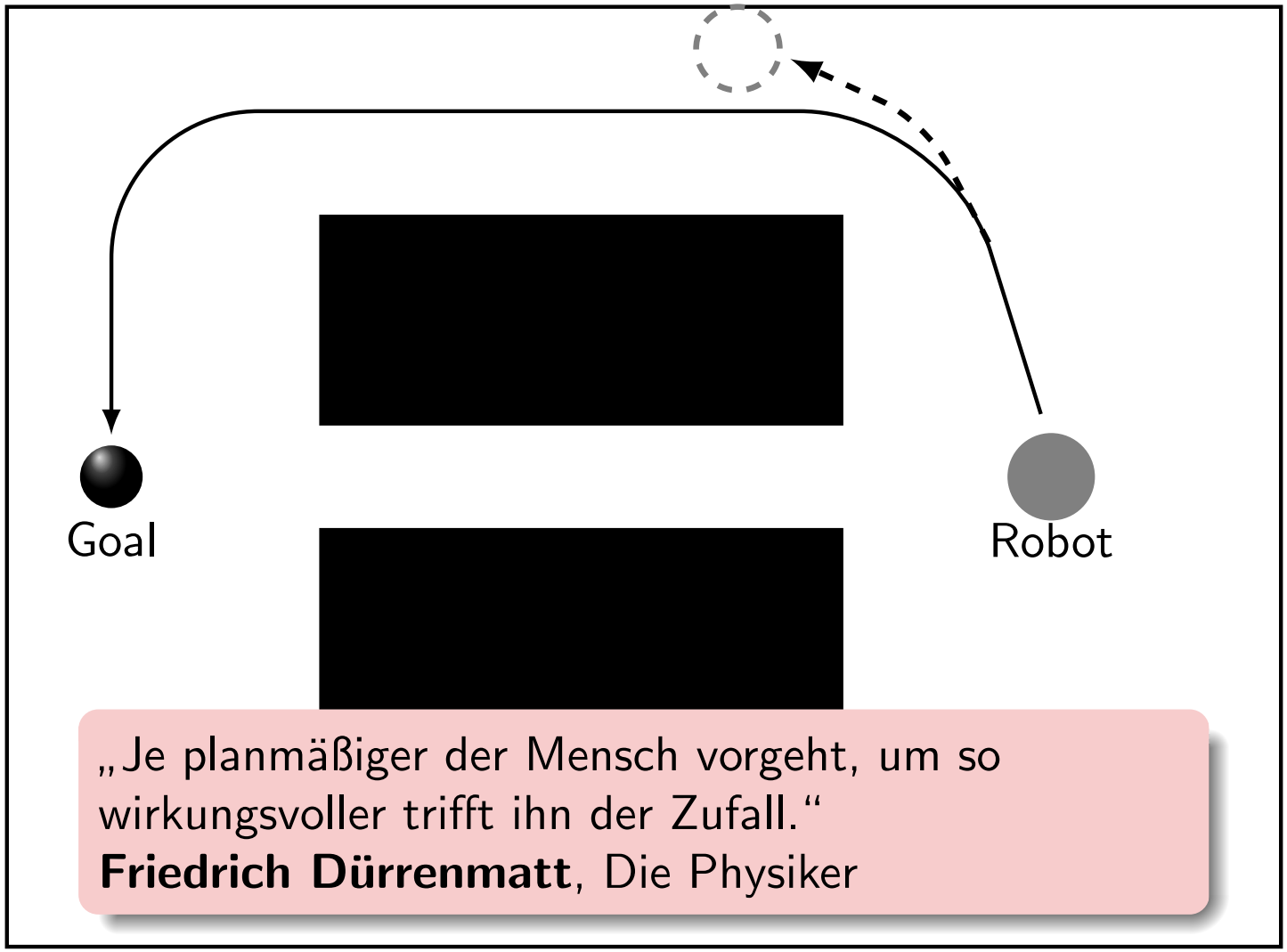


Policy vs Plan: Plan



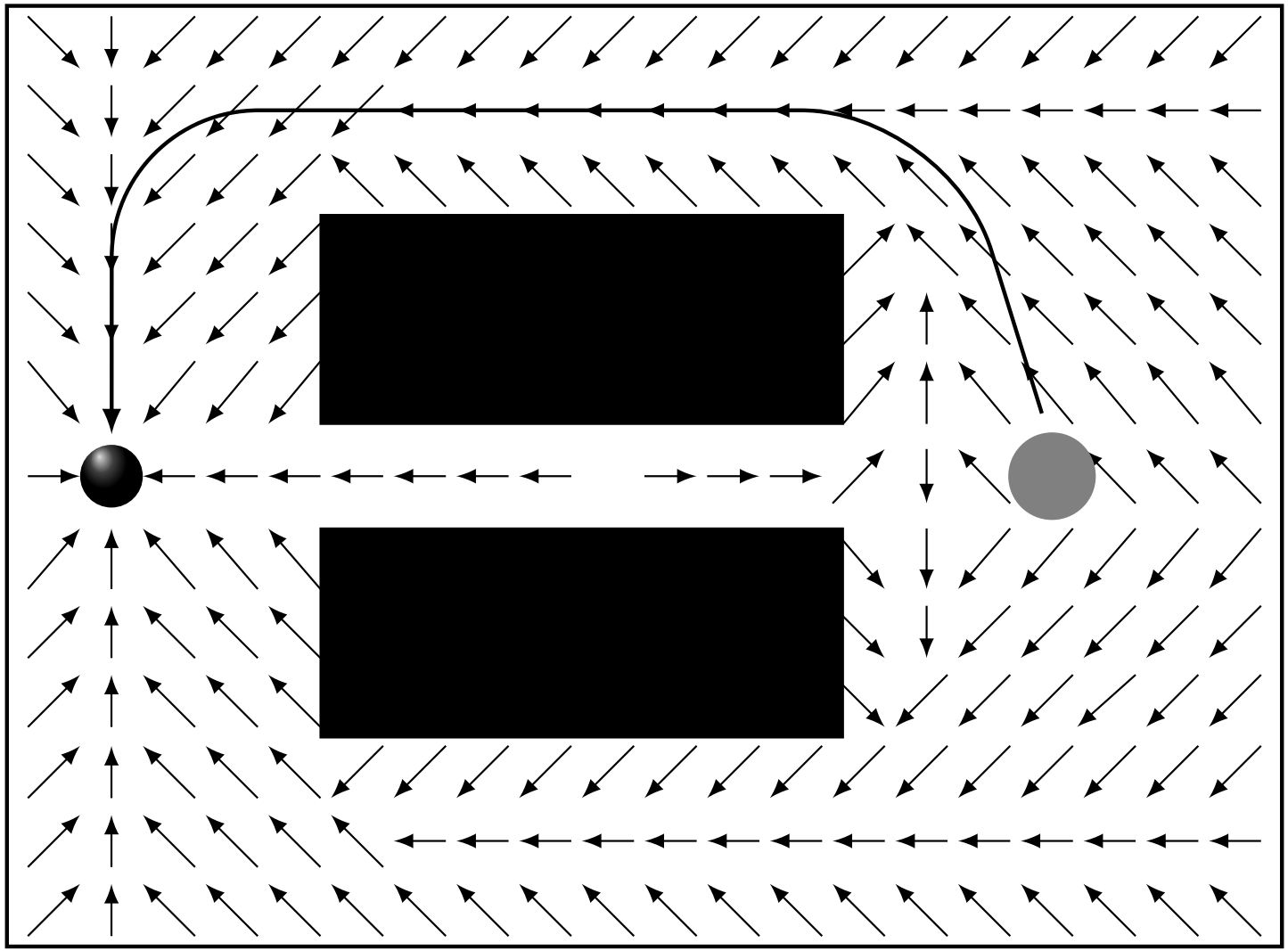
„Je planmäßiger der Mensch vorgeht, um so wirkungsvoller trifft ihn der Zufall.“
Friedrich Dürrenmatt, Die Physiker

Policy vs Plan: Plan



„Je planmäßiger der Mensch vorgeht, um so wirkungsvoller trifft ihn der Zufall.“
Friedrich Dürrenmatt, Die Physiker

Policy vs Plan: Policy



Solution

Optimal Policy: $\pi^*(x) = \arg \max_{\pi} V_\pi(x)$

Solution

Optimal Policy: $\pi^*(x) = \arg \max_{\pi} V_\pi(x)$

$$V_\pi(x) = \mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot \underbrace{r(x_t, \pi(x_t))}_{=:r_t} \;\middle|\; x_0 = x \right\}$$

Solution

Optimal Policy: $\pi^*(x) = \arg \max_{\pi} V_\pi(x)$

$$V_\pi(x) = \mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot \underbrace{r(x_t, \pi(x_t))}_{=:r_t} \;\middle|\; x_0 = x \right\}$$

$$= \mathrm{E}\left\{ r_0 + \gamma \cdot \sum_{t=0}^{\infty} \gamma^t \cdot r_{t+1} \;\middle|\; x_0 = x \right\}$$

Solution

Optimal Policy: $\pi^*(x) = \arg\max_{\pi} V_\pi(x)$

$$
\begin{aligned}
V_\pi(x) &= \mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot \underbrace{r(x_t, \pi(x_t))}_{=:r_t} \;\middle|\; x_0 = x \right\} \\
&= \mathrm{E}\left\{ r_0 + \gamma \cdot \sum_{t=0}^{\infty} \gamma^t \cdot r_{t+1} \;\middle|\; x_0 = x \right\} \\
&= r_0 + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi(x)) \cdot \underbrace{\mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot r_{t+1} \;\middle|\; x_1 = x' \right\}}_{=V_\pi(x')}
\end{aligned}
$$

Solution

Optimal Policy: $\pi^*(x) = \arg\max_{\pi} V_\pi(x)$

$$
\begin{aligned}
V_\pi(x) &= \mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot \underbrace{r(x_t, \pi(x_t))}_{=:r_t} \;\middle|\; x_0 = x \right\} \\
&= \mathrm{E}\left\{ r_0 + \gamma \cdot \sum_{t=0}^{\infty} \gamma^t \cdot r_{t+1} \;\middle|\; x_0 = x \right\} \\
&= r_0 + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi(x)) \cdot \underbrace{\mathrm{E}\left\{ \sum_{t=0}^{\infty} \gamma^t \cdot r_{t+1} \;\middle|\; x_1 = x' \right\}}_{=V_\pi(x')}
\end{aligned}
$$

Recursion, especially holds for the optimal policy!

Solution

Optimal Policy: $\pi^*(x) = \arg\max_{\pi} V_\pi(x)$

$$
\begin{aligned}
V^*(x) &:= V_{\pi^*}(x) \\
&= r(x, \pi^*(x)) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi^*(x)) \cdot V^*(x')
\end{aligned}
$$

Solution

Optimal Policy: $\pi^*(x) = \arg \max_{\pi} V_\pi(x)$

$$
\begin{aligned}
V^*(x) &:= V_{\pi^*}(x) \\
&= r(x, \pi^*(x)) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi^*(x)) \cdot V^*(x') \\
&= \max_{a \in \mathcal{A}} \left\{ r(x, a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, a) \cdot V^*(x') \right\}
\end{aligned}
$$

Solution

Optimal Policy: $\pi^*(x) = \arg\max_{\pi} V_\pi(x)$

$$
\begin{aligned}
V^*(x) &:= V_{\pi^*}(x) \\
&= r(x, \pi^*(x)) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi^*(x)) \cdot V^*(x') \\
&= \max_{a \in \mathcal{A}} \Bigg\{ \underbrace{r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot V^*(x')}_{:= Q^*(x,a) \quad \textbf{(Q-function)}} \Bigg\}
\end{aligned}
$$

Solution

Optimal Policy: $\pi^*(x) = \arg\max_{\pi} V_\pi(x)$

$$V^*(x) := V_{\pi^*}(x)$$
$$= r(x, \pi^*(x)) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x, \pi^*(x)) \cdot V^*(x')$$
$$= \max_{a \in \mathcal{A}} \left\{ \underbrace{r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot V^*(x')}_{:= Q^*(x,a) \quad \textbf{(Q-function)}} \right\}$$

Bellman Equation / Dynamic Programing (DP)

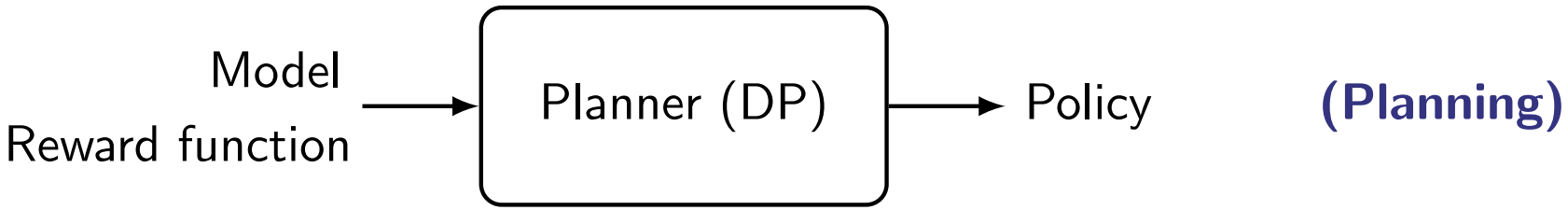
Contraction fix-point iteration yields optimal $V^*(x)$ independent of initialization

Optimal Policy $\pi^*(x) = \arg\max_{a \in \mathcal{A}} Q^*(x,a) = \arg\max_{\pi} V_\pi(x)$

Planning vs Reinforcement Learning

Problem

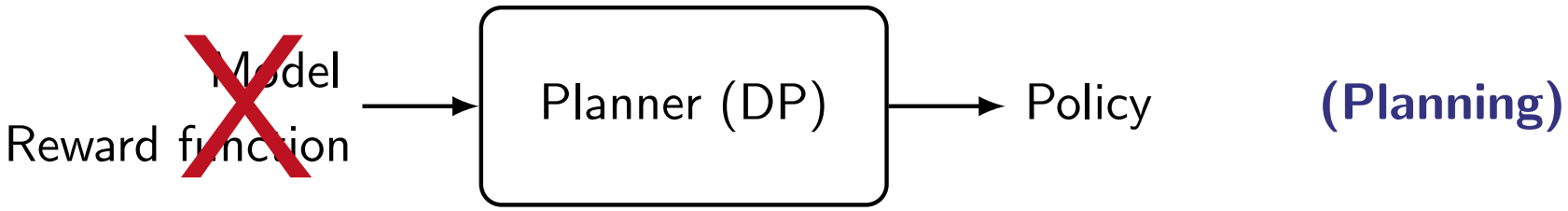
Up to now we assumed that model and reward function are known
➔ Not realistic in many applications



Planning vs Reinforcement Learning

Problem

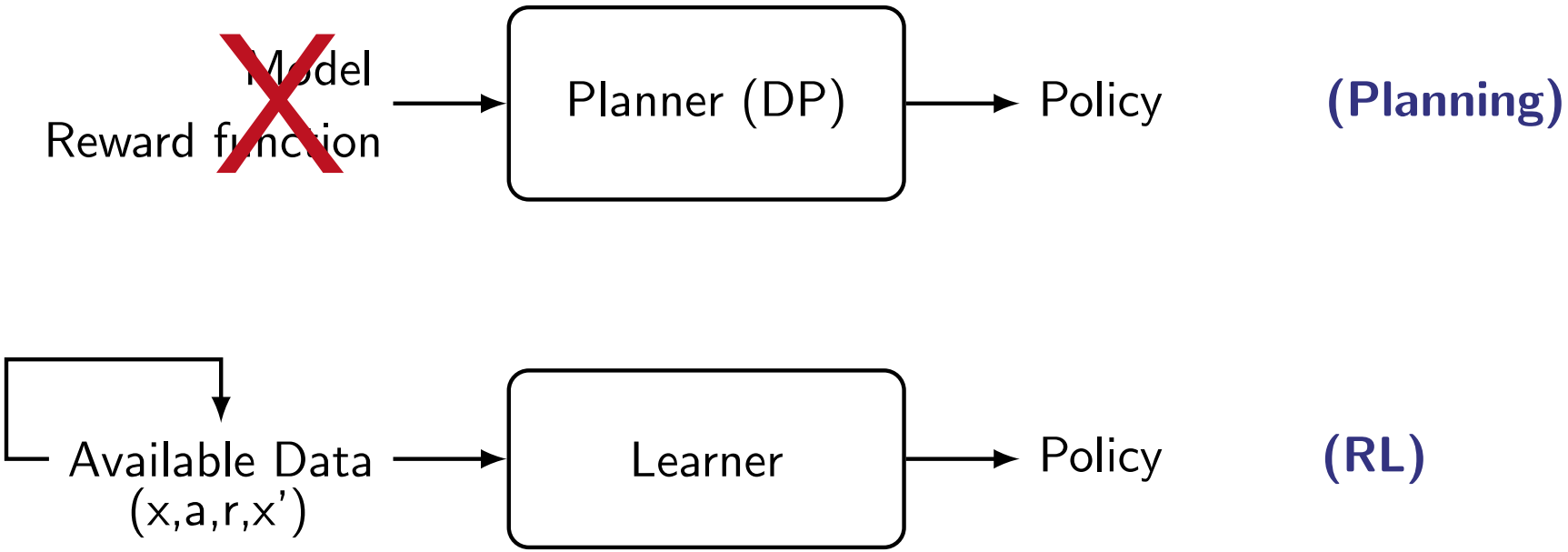
Up to now we assumed that model and reward function are known
➔ Not realistic in many applications



Planning vs Reinforcement Learning

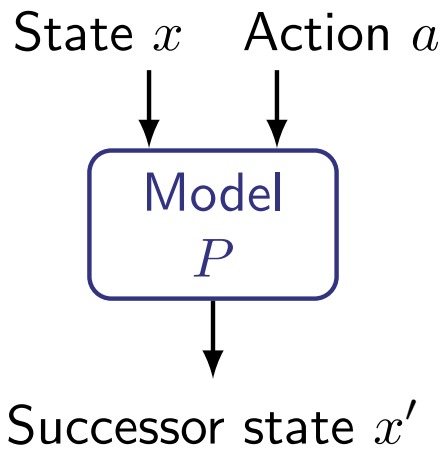
Problem

Up to now we assumed that model and reward function are known
➔ Not realistic in many applications



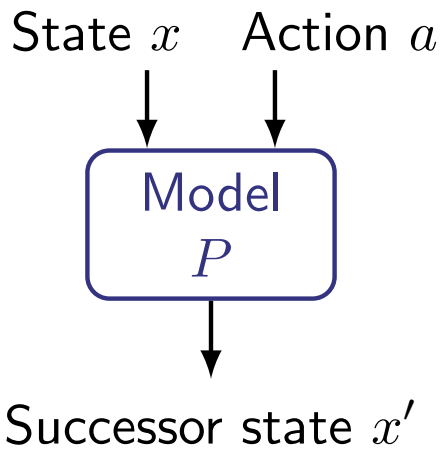
Fundamental Types of Reinforcement Learning

Model-based

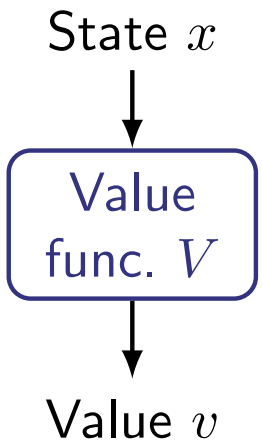


Fundamental Types of Reinforcement Learning

Model-based

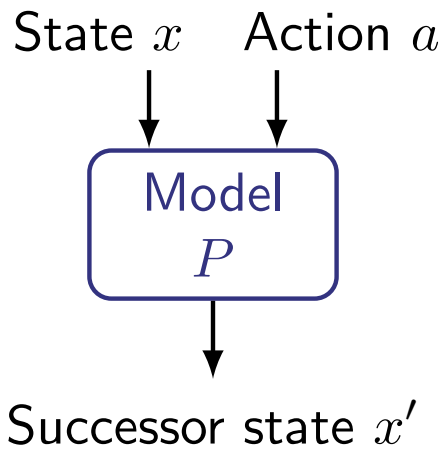


Value function based

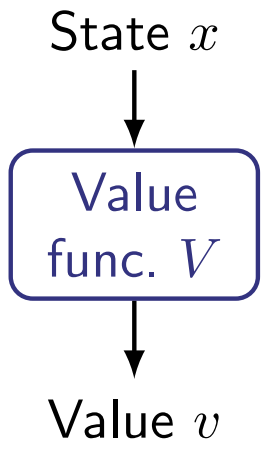


Fundamental Types of Reinforcement Learning

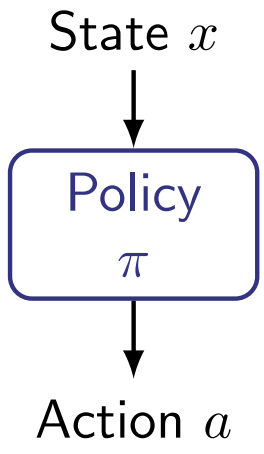
Model-based



Value function based

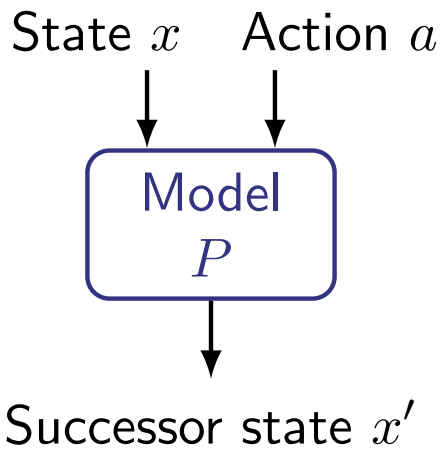


Policy Search

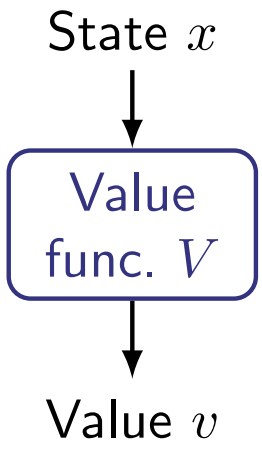


Fundamental Types of Reinforcement Learning

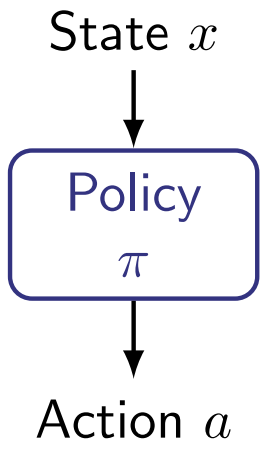
Model-based



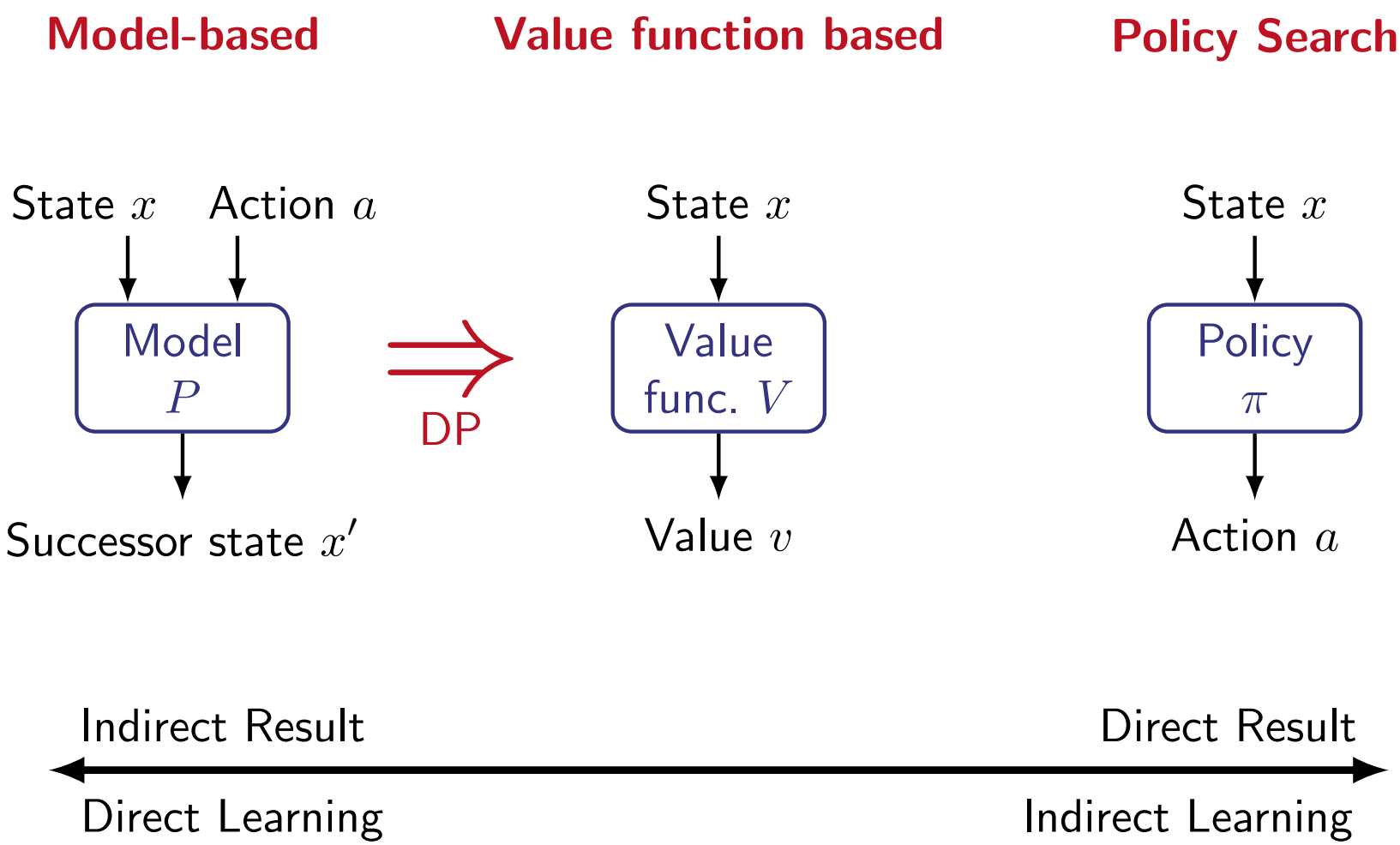
Value function based



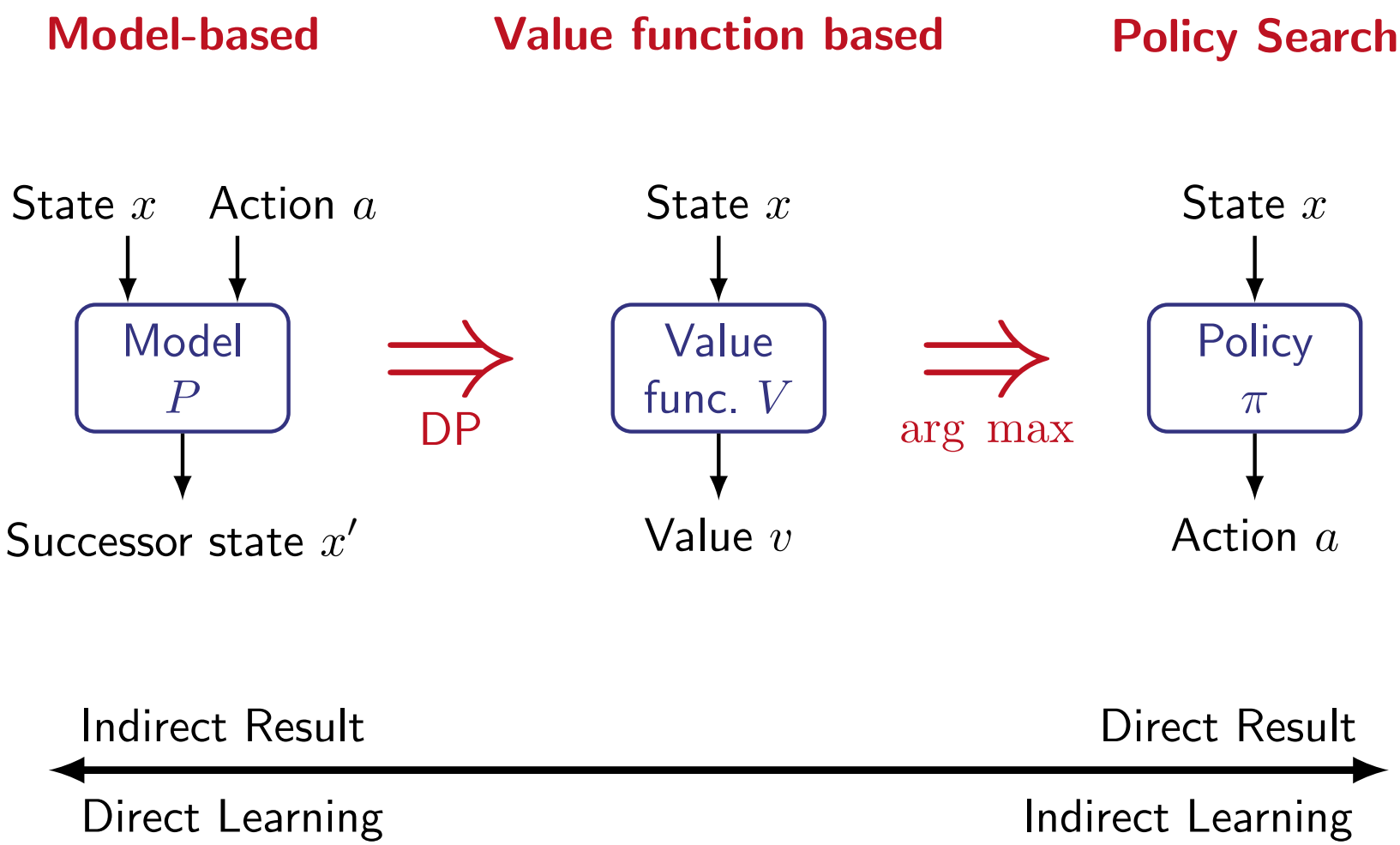
Policy Search



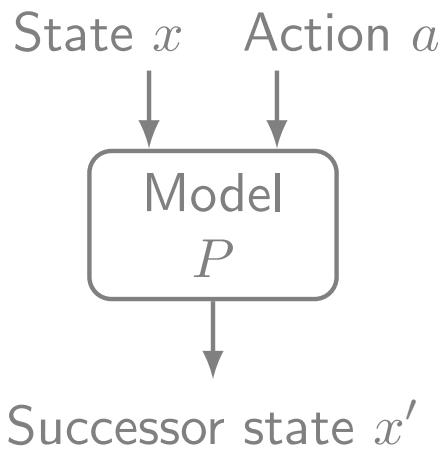
Fundamental Types of Reinforcement Learning

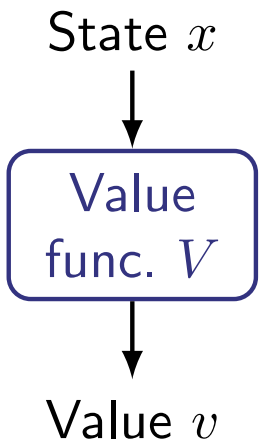


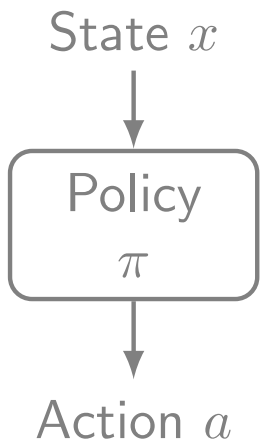
Fundamental Types of Reinforcement Learning



Model-based | **Value function based** | Policy Search







Q-learning

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot V^*(x')$$

where

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

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

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot V^*(x')$$

where

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

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

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning with a Single Sample

Recursive estimation of the Q-function via transition data (x, a, r, x'):

$$\hat{Q}(x,a) \leftarrow (1-\alpha) \cdot \hat{Q}(x,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x',a') \right)$$

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning with a Single Sample

Recursive estimation of the Q-function via transition data (x, a, r, x'):

$$\hat{Q}(x,a) \leftarrow (1-\alpha) \cdot \hat{Q}(x,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x',a') \right)$$

current value — expected value

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning with a Single Sample

Recursive estimation of the Q-function via transition data (x, a, r, x'):

$$\hat{Q}(x,a) \leftarrow \hat{Q}(x,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x',a') - \hat{Q}(x,a) \right)$$

Learning the Q-function

Recall: Q-function

$$Q^*(x,a) = r(x,a) + \gamma \cdot \sum_{x' \in \mathcal{X}} P(x'|x,a) \cdot \max_{a' \in \mathcal{A}} Q^*(x',a')$$

where

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

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

Learning with a Single Sample

Recursive estimation of the Q-function via transition data (x, a, r, x'):

$$\hat{Q}(x,a) \leftarrow \hat{Q}(x,a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x',a') - \hat{Q}(x,a) \right)$$

Learning rate — Error / temporal difference

Q-learning Algorithm (v0)

1: Initialize $\hat{Q}(x, a)$ arbitrary $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}$
2: **while** true **do**
3: Choose initial state x
4: **repeat** (*for every time step*)
5: Choose action a
6: Execute a: observe reward r, successor state x'
7: $\hat{Q}(x, a) \leftarrow \hat{Q}(x, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x', a') - \hat{Q}(x, a) \right)$
8: $x \leftarrow x'$
9: **until** State x is terminal
10: **end while**

Q-learning Algorithm (v0)

1: Initialize $\hat{Q}(x, a)$ arbitrary $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}$
2: **while** true **do**
3: Choose initial state x
4: **repeat** (*for every time step*)
5: Choose action a
6: Execute a: observe reward r, successor state x'
7: $$\hat{Q}(x, a) \leftarrow \hat{Q}(x, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x', a') - \hat{Q}(x, a) \right)$$
8: $x \leftarrow x'$
9: **until** State x is terminal
10: **end while**

Q-learning **converges**, i.e., $\hat{Q} \rightarrow Q^*$ if

- $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$
- Every state-action pair (x, a) is visited infinitely often

Q-learning Algorithm (v0)

1: Initialize $\hat{Q}(x, a)$ arbitrary $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}$
2: **while** true **do**
3: Choose initial state x
4: **repeat** (*for every time step*)
5: Choose action a
6: Execute a: observe reward r, successor state x'
7: $$\hat{Q}(x, a) \leftarrow \hat{Q}(x, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x', a') - \hat{Q}(x, a) \right)$$
8: $x \leftarrow x'$
9: **until** State x is terminal
10: **end while**

Q-learning **converges**, i.e., $\hat{Q} \rightarrow Q^*$ if

- $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$
- Every state-action pair (x, a) is visited infinitely often

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence
- Using $\hat{Q}$ ➔ convergence towards local maximum

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence **Exploration**
- Using $\hat{Q}$ ➔ convergence towards local maximum

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence **Exploration**
- Using $\hat{Q}$ ➔ convergence towards local maximum **Exploitation**

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence — **Exploration**
- Using $\hat{Q}$ ➔ convergence towards local maximum — **Exploitation**

Combination of both Approaches

1. **Begin with exploration:** Try new "paths" through the state-action space to possibly discover better policies.
2. **Gradually transit to Exploitation:** Utilize existing experience to gain a preferably high reward.

Exploration vs Exploitation

Choose Actions

- Static choice ➔ no convergence
- Random choice ➔ slow convergence — **Exploration**
- Using $\hat{Q}$ ➔ convergence towards local maximum — **Exploitation**

Combination of both Approaches

1. **Begin with exploration:** Try new "paths" through the state-action space to possibly discover better policies.
2. **Gradually transit to Exploitation:** Utilize existing experience to gain a preferably high reward.

➔ Occasionally take a random action (similar to "Simulated Annealing"):

$$\pi(x) = \begin{cases} \text{rand}(\mathcal{A}) & \text{, with probability } \epsilon \\ \arg\max_a \hat{Q}(x,a) & \text{, with probability } 1-\epsilon \end{cases}$$

Q-learning Algorithm (v1)

1: Initialize $\hat{Q}(x, a)$ arbitrary $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}$
2: **while** true **do**
3: Choose initial state x
4: **repeat** (*for every time step*)
5: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $$\hat{Q}(x, a) \leftarrow \hat{Q}(x, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x', a') - \hat{Q}(x, a) \right)$$
9: $x \leftarrow x'$
10: **until** State x is terminal
11: **end while**

Q-learning Algorithm (v1)

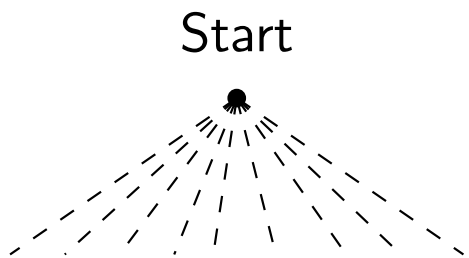
1: Initialize $\hat{Q}(x, a)$ arbitrary $\forall x \in \mathcal{X}, \forall a \in \mathcal{A}$
2: **while** true **do**
3: Choose initial state x
4: **repeat** (*for every time step*)
5: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $$\hat{Q}(x, a) \leftarrow \hat{Q}(x, a) + \alpha \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}} \hat{Q}(x', a') - \hat{Q}(x, a) \right)$$
9: $x \leftarrow x'$
10: **until** State x is terminal
11: **end while**

Convergence

For **GLIE**-policies = **g**reedy in the **l**imit with **i**nfinite **e**xploration
➔ satisfied if ϵ declines over time: $\epsilon(x) := c/n(x)$ with $c \in (0, 1)$

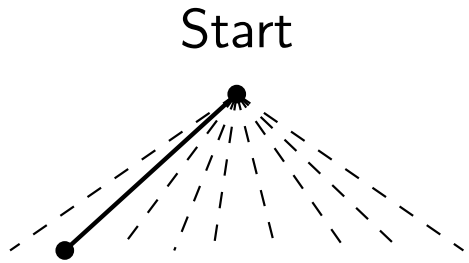
Example: Tic-Tac-Toe

RL {

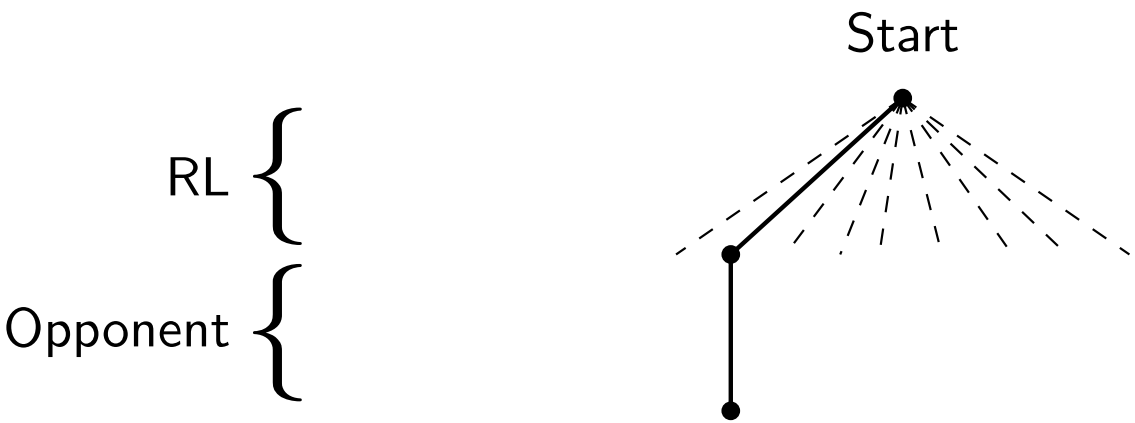


Example: Tic-Tac-Toe

RL {



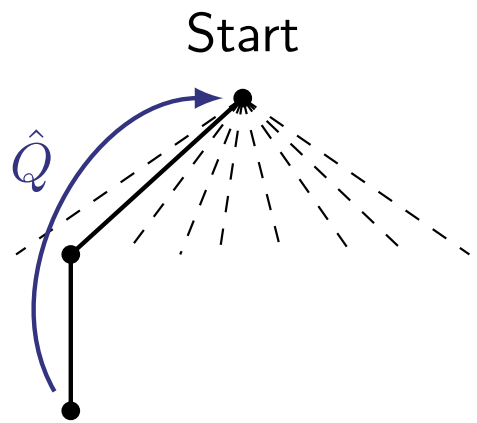
Example: Tic-Tac-Toe



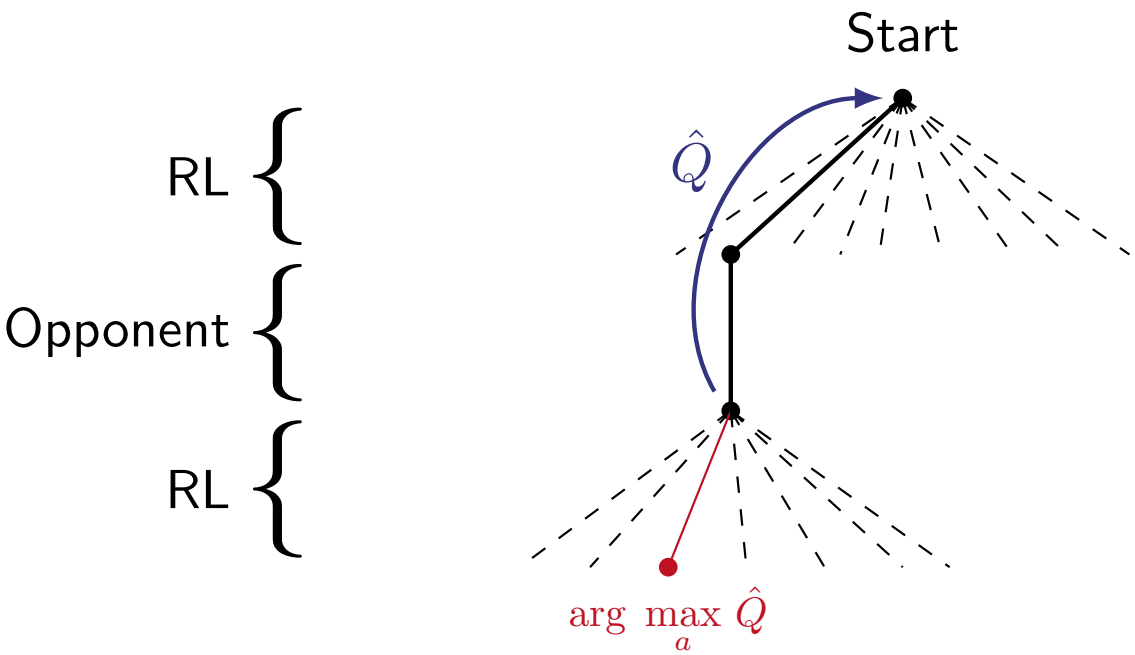
Example: Tic-Tac-Toe

RL {

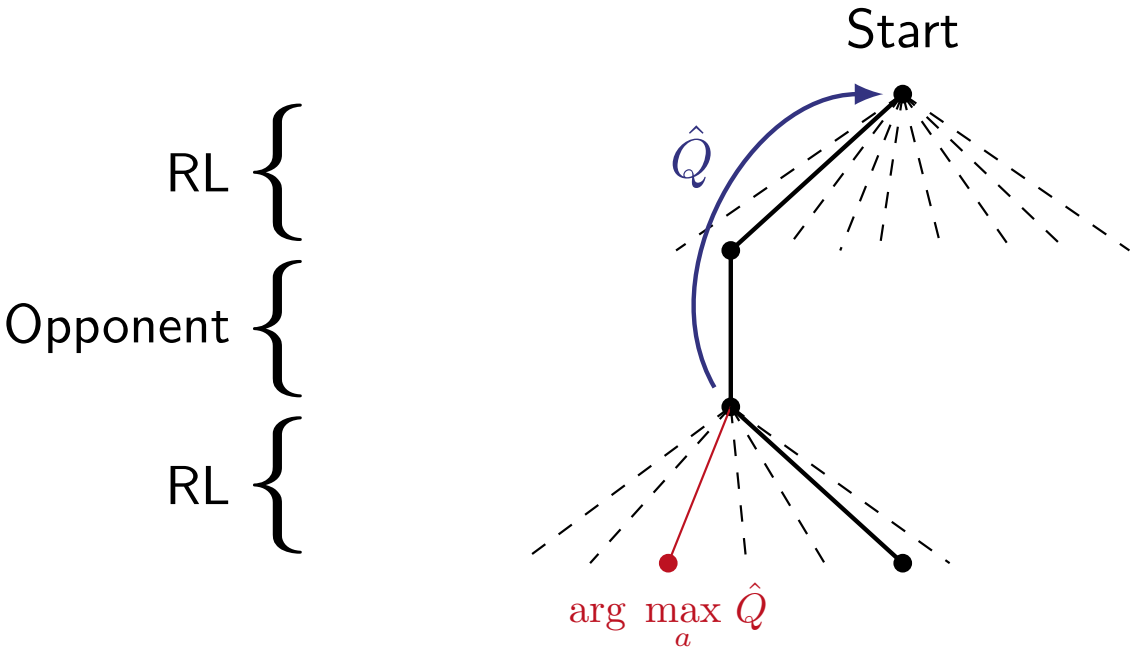
Opponent {



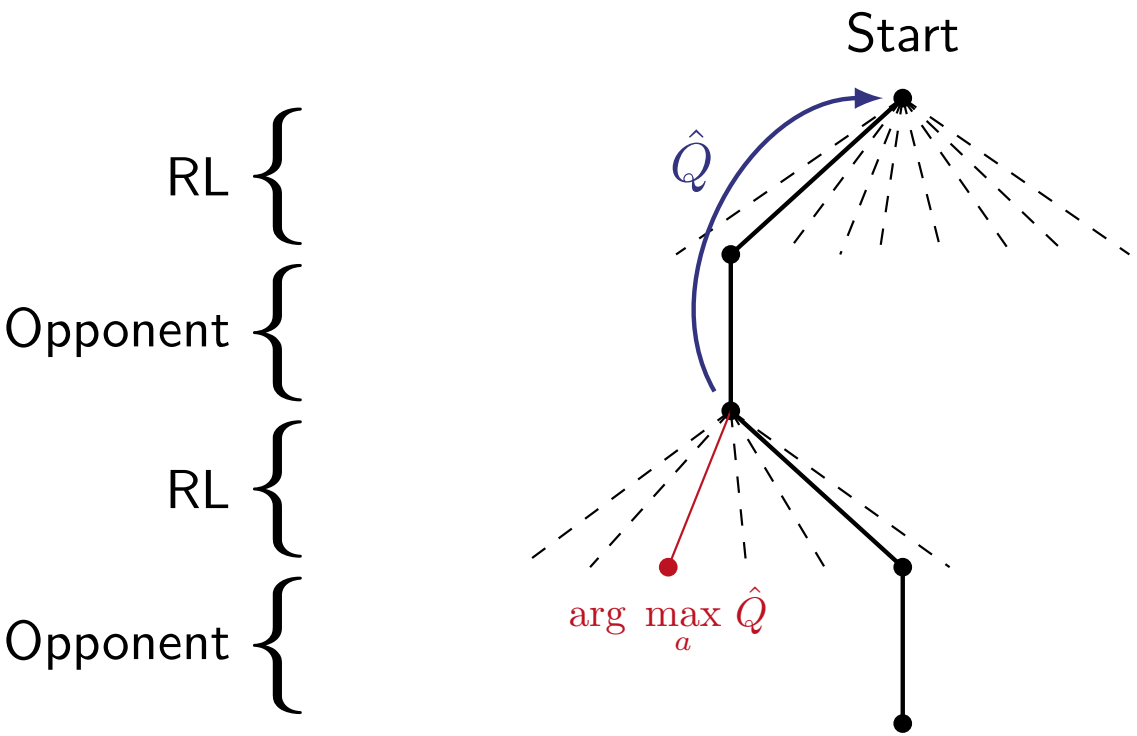
Example: Tic-Tac-Toe



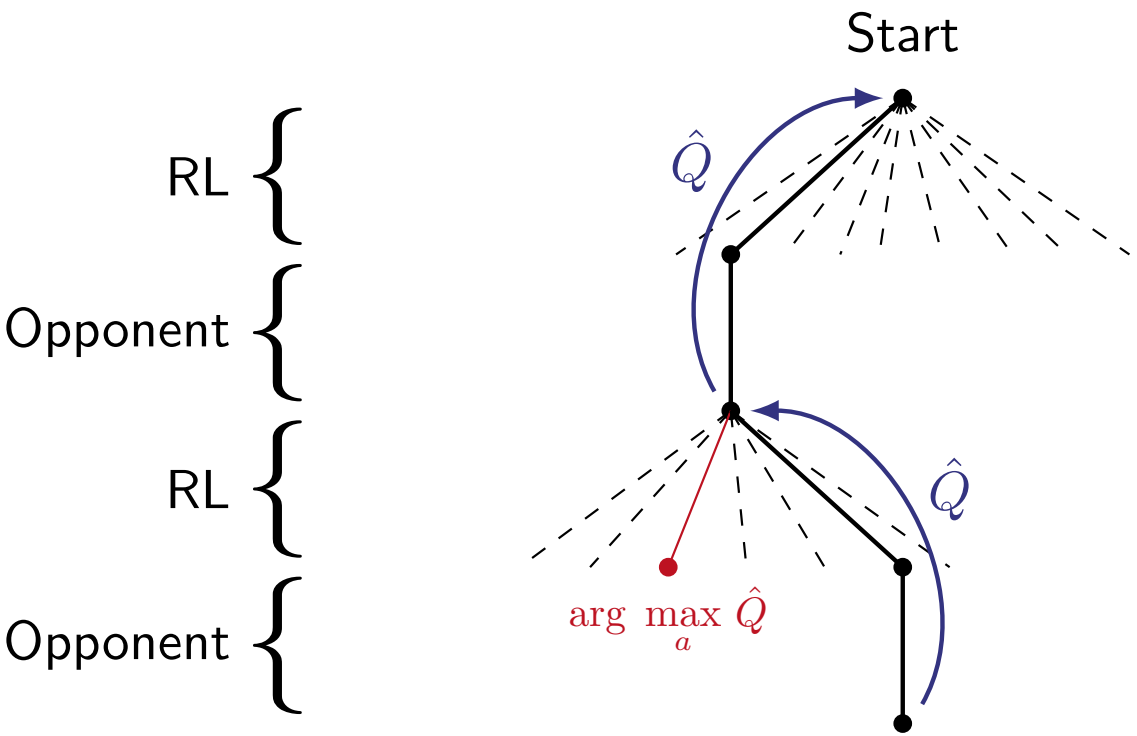
Example: Tic-Tac-Toe



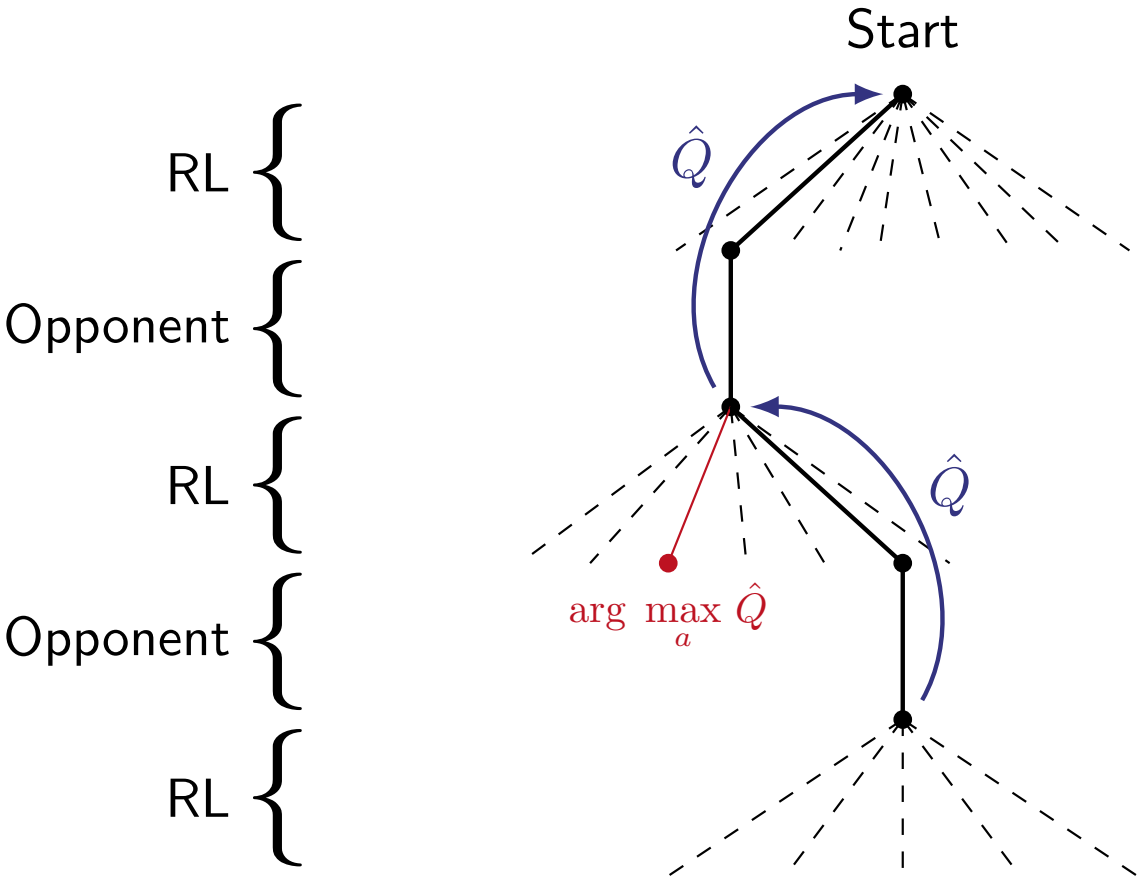
Example: Tic-Tac-Toe



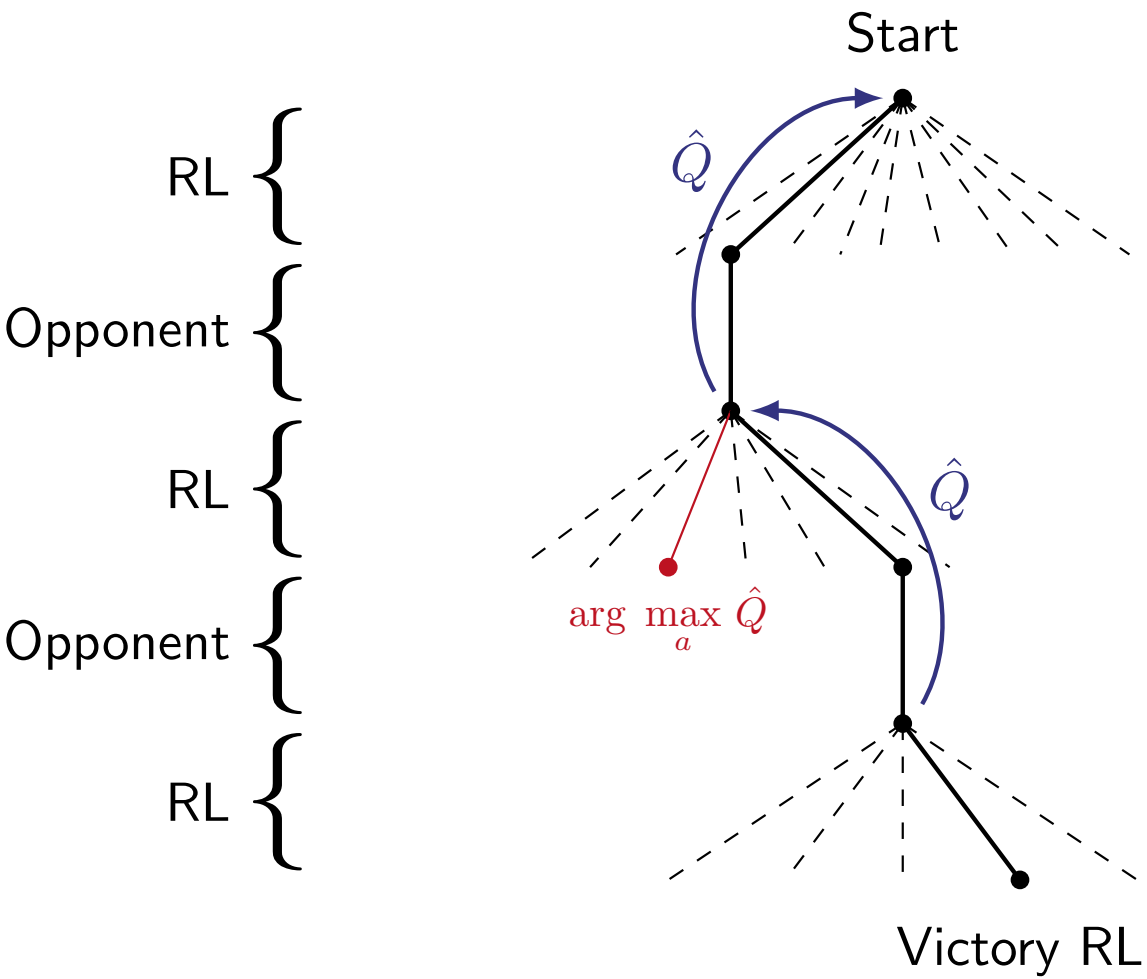
Example: Tic-Tac-Toe



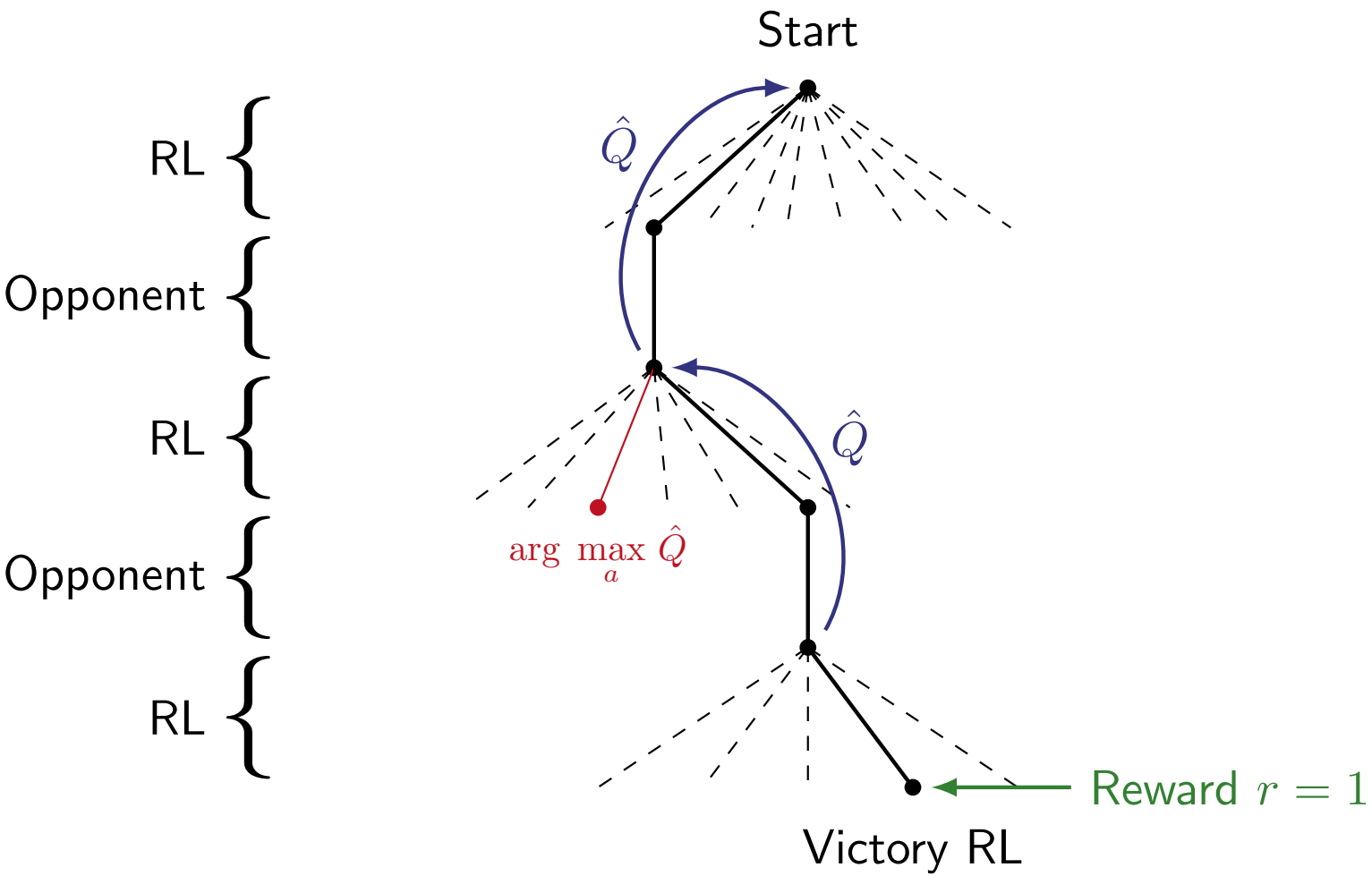
Example: Tic-Tac-Toe



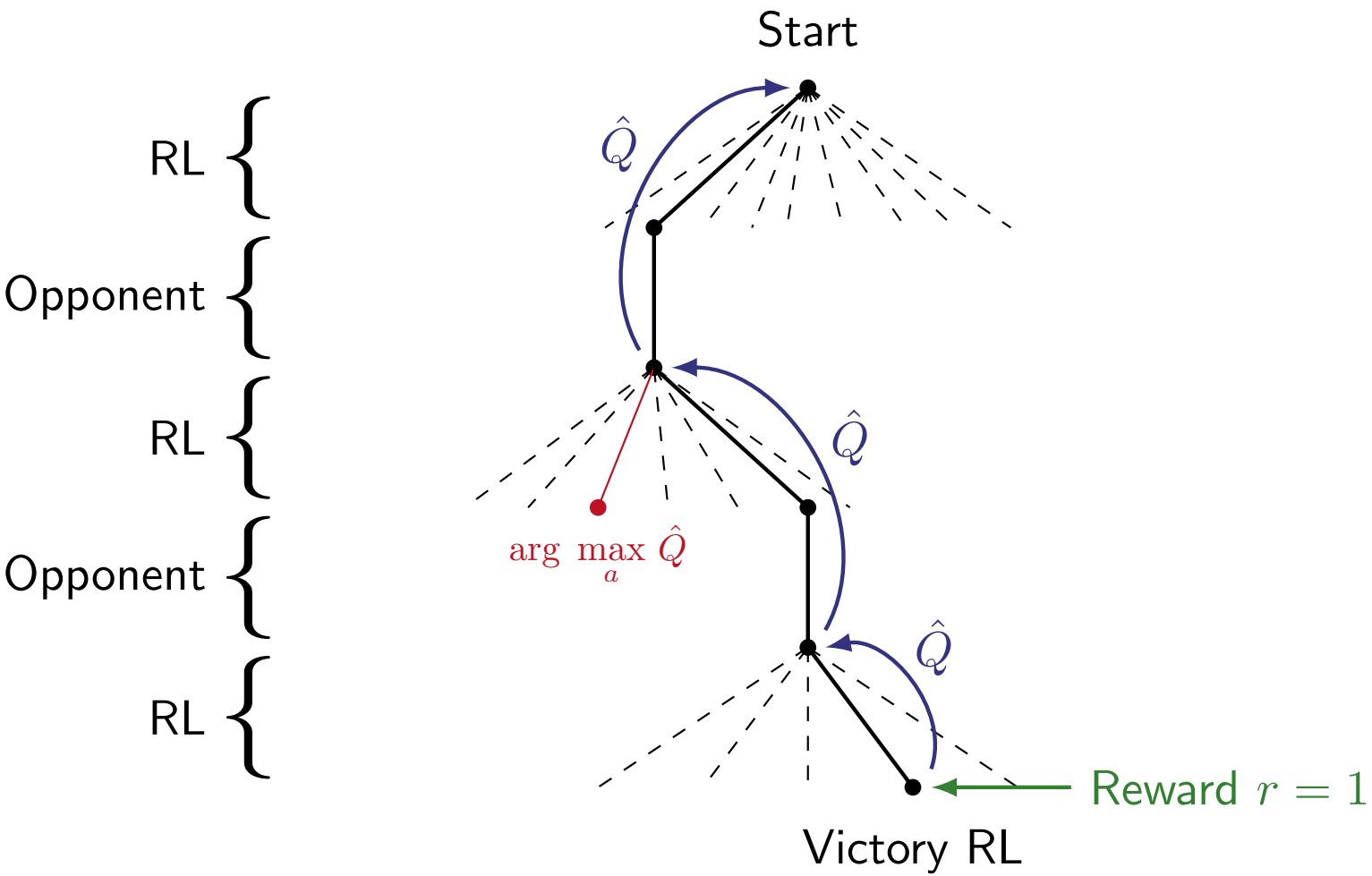
Example: Tic-Tac-Toe



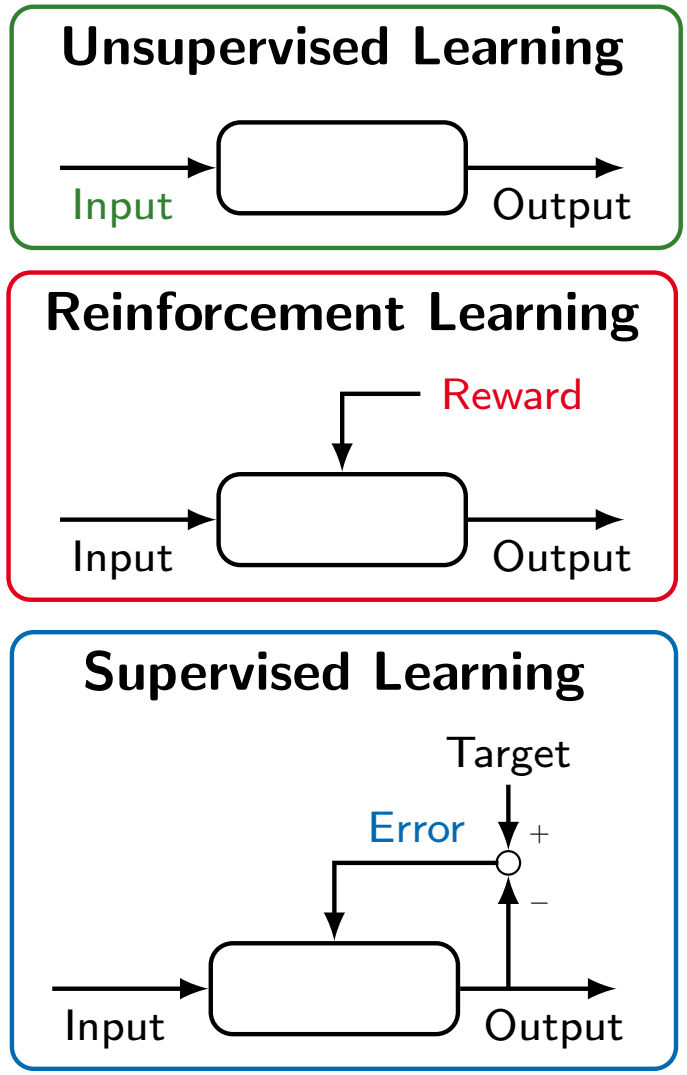
Example: Tic-Tac-Toe



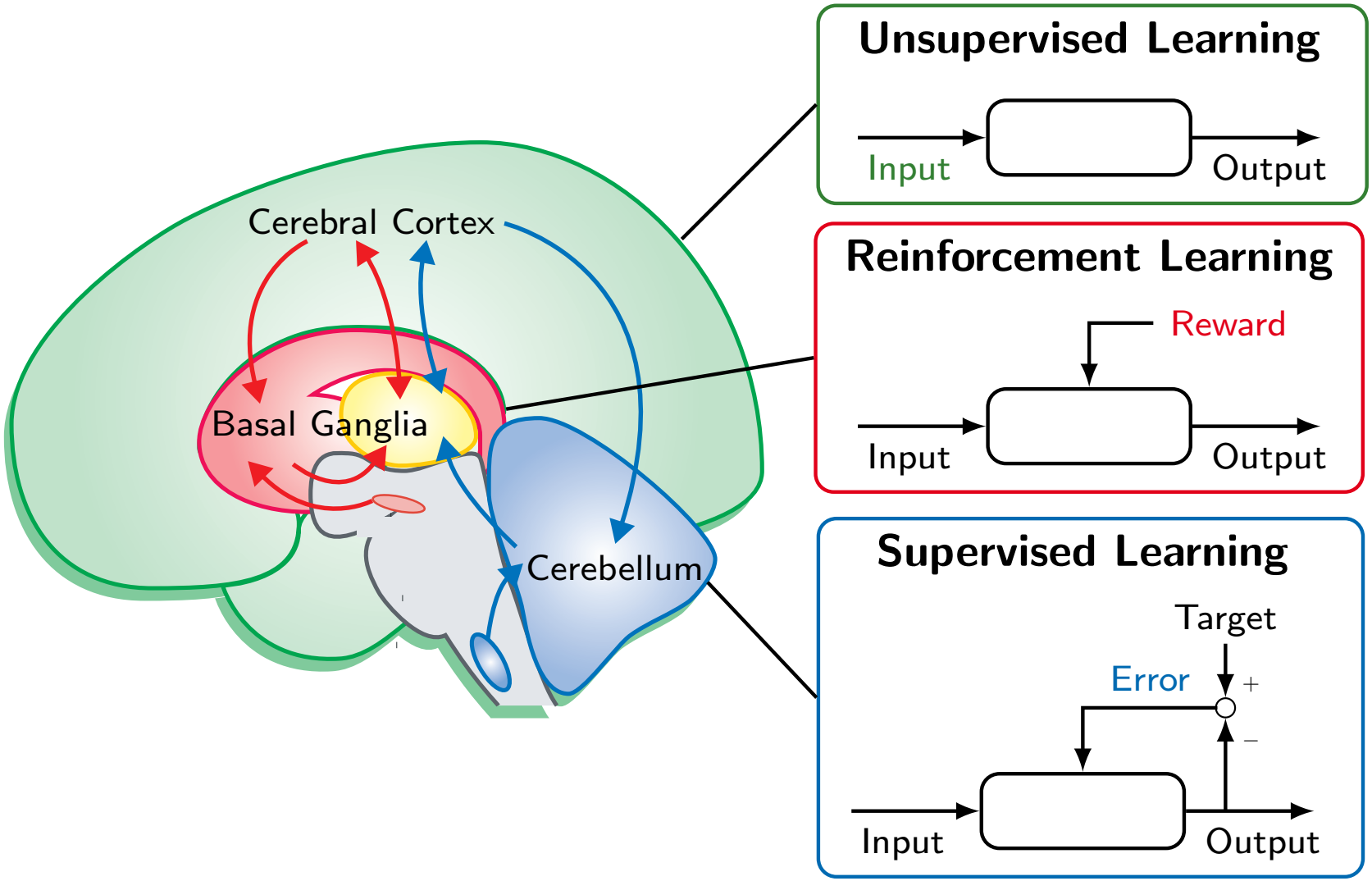
Example: Tic-Tac-Toe



Types of (Machine) Learning

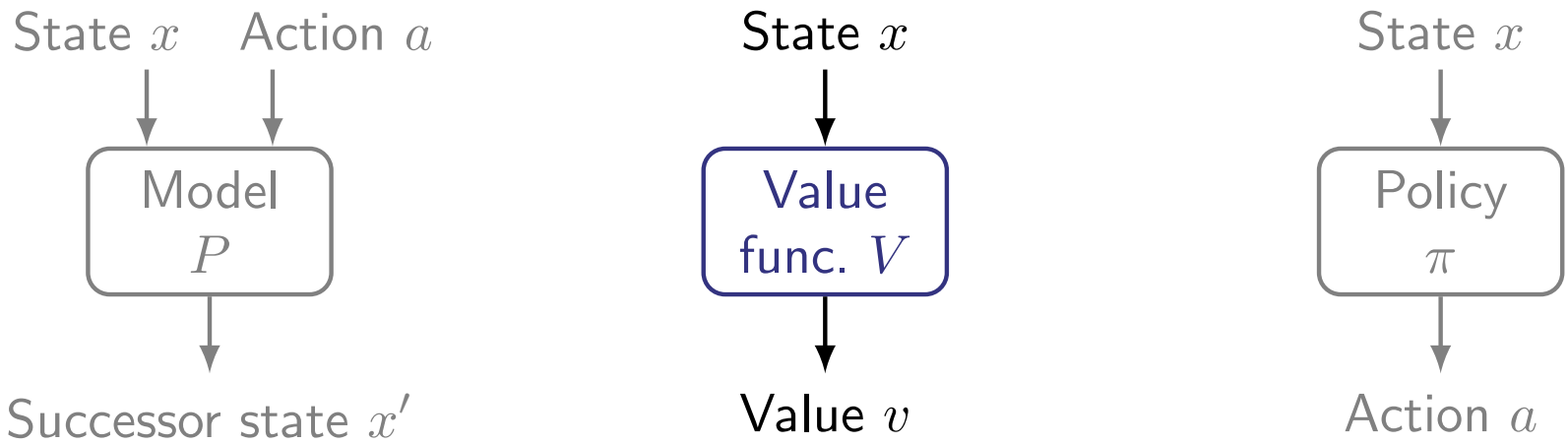


Types of (Machine) Learning



Kenji Doya: *Complementary roles of basal ganglia and cerebellum in learning and motor control*, Current Opinion in Neurobiology, 2000.

Model-based **Value function based** **Policy Search**



Function Approximation

Basic Idea

So far

Assumption of finite state and action spaces!

- continuous spaces?
- finite but huge spaces (e. g., 10^{20} states)?

➔ Visit of all state-action pairs no longer possible

➔ Generalization over large unvisited parts

Basic Idea

So far

Assumption of finite state and action spaces!

- continuous spaces?
- finite but huge spaces (e. g., 10^{20} states)?

➔ Visit of all state-action pairs no longer possible

➔ Generalization over large unvisited parts

Function Approximation

Function approximators for representing the value function / Q-function:

Parametric	Polynomial, artificial neural networks, fuzzy sets
Non-parameteric	support vector regression, Gaussian processes

Basic Idea

So far

Assumption of finite state and action spaces!

- continuous spaces?
- finite but huge spaces (e. g., 10^{20} states)?

➔ Visit of all state-action pairs no longer possible

➔ Generalization over large unvisited parts

Function Approximation

Function approximators for representing the value function / Q-function:

Parametric	Polynomial, artificial neural networks, fuzzy sets
Non-parameteric	support vector regression, Gaussian processes

Challenges of RL

- non-static training sets
- target functions (V, Q) are non-stationary

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \left(Q^*(x, a) - Q(x, a|\underline{\theta})\right)^2$$

for given x and a.

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \left(Q^*(x, a) - Q(x, a|\underline{\theta})\right)^2$$

for given x and a.

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \left(Q^*(x, a) - Q(x, a|\underline{\theta})\right)^2$$

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big)^2$$

for given x and a.

$$\begin{aligned}\underline{\theta} &\leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big)^2 \\ &\leftarrow \underline{\theta} - \alpha \cdot \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x, a|\underline{\theta})\end{aligned}$$

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \left(Q^*(x, a) - Q(x, a|\underline{\theta}) \right)^2$$

for given x and a.

$$\begin{aligned} \underline{\theta} &\leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \left(Q^*(x, a) - Q(x, a|\underline{\theta}) \right)^2 \\ &\leftarrow \underline{\theta} - \alpha \cdot \Big(\underbrace{Q^*(x, a)}_{\approx r + \gamma \cdot \max_{a'} Q(x', a'|\underline{\theta})} - Q(x, a|\underline{\theta}) \Big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x, a|\underline{\theta}) \end{aligned}$$

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x,a)$ via $Q(x,a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big)^2$$

for given x and a.

$$\begin{aligned}
\underline{\theta} &\leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big)^2 \\
&\leftarrow \underline{\theta} - \alpha \cdot \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x,a|\underline{\theta}) \\
&\leftarrow \underline{\theta} - \alpha \cdot \Big(r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(x',a|\underline{\theta}) - Q(x,a|\underline{\theta}) \Big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x,a|\underline{\theta})
\end{aligned}$$

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x, a)$ via $Q(x, a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big)^2$$

for given x and a.

$$\begin{aligned}
\underline{\theta} &\leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big)^2 \\
&\leftarrow \underline{\theta} - \alpha \cdot \big(Q^*(x, a) - Q(x, a|\underline{\theta})\big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x, a|\underline{\theta}) \\
&\leftarrow \underline{\theta} - \alpha \cdot \Big(r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(x', a|\underline{\theta}) - Q(x, a|\underline{\theta}) \Big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x, a|\underline{\theta})
\end{aligned}$$

temporal difference

Example: Q-learning

Parametric Q-function

Approximation of optimal Q-function $Q^*(x,a)$ via $Q(x,a|\underline{\theta})$

Instead of Global Optimization

Step-wise calculation of parameter vector $\underline{\theta}$ via **gradient ascent**

$$\underline{\theta} \leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big)^2$$

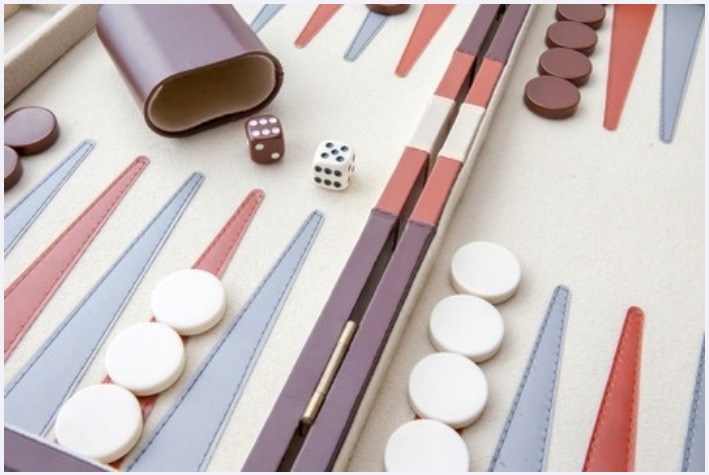
for given x and a.

$$\begin{aligned}
\underline{\theta} &\leftarrow \underline{\theta} + \tfrac{1}{2} \cdot \alpha \cdot \frac{\partial}{\partial \underline{\theta}} \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big)^2 \\
&\leftarrow \underline{\theta} - \alpha \cdot \big(Q^*(x,a) - Q(x,a|\underline{\theta})\big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x,a|\underline{\theta}) \\
&\leftarrow \underline{\theta} - \alpha \cdot \Big(r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(x',a|\underline{\theta}) - Q(x,a|\underline{\theta}) \Big) \cdot \frac{\partial}{\partial \underline{\theta}} Q(x,a|\underline{\theta})
\end{aligned}$$

Convergence: No general statements. Only for linear approximators.

Application: Backgammon

TD-Gammon



- Gerald Tesauro, 1992–1995
- Artificial neural network for value function approximation
- Gradient ascent for learning the network weights
- 1.5 mio. games against itself

Results

- Better than any former Backgammon program
- Able to beat (human) world-class players
- Played particular game openings contrary to conventions
  ➔ Adopted by the best (human) players

Application: Go

AlphaGo

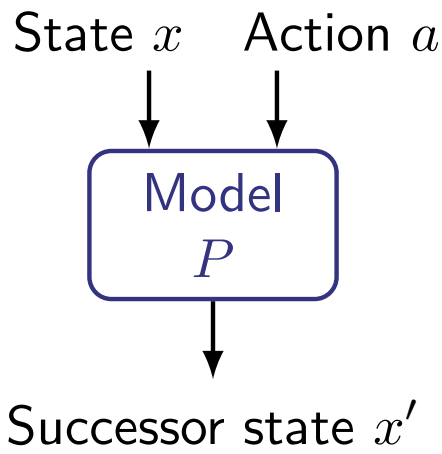
- Google DeepMind, 2014–today
- Deep neural networks for policy and value function approximation
- Learns from human games and games against itself
- Monte Carlo tree search for action selection

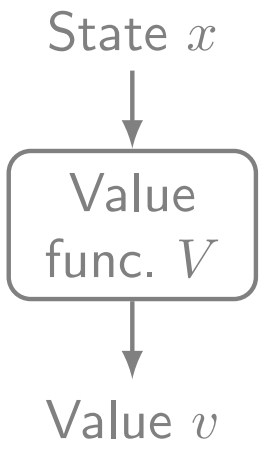
Results

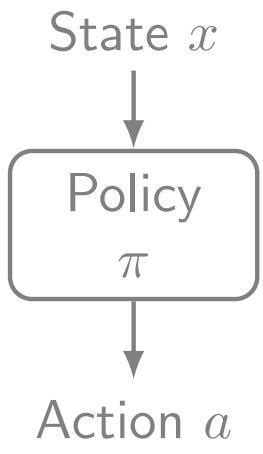
- Better than any former Go program, even in limited versions
- 5-0 victory against European champion Fan Hui, October 2015
  4-1 victory against world champion Lee Sedol, March 2016
  3-0 victory against number-one ranked Ke Jie, May 2017

  ➔ AlphaGo awarded an honorary 9-dan professional

Model-based | **Value function based** | **Policy Search**

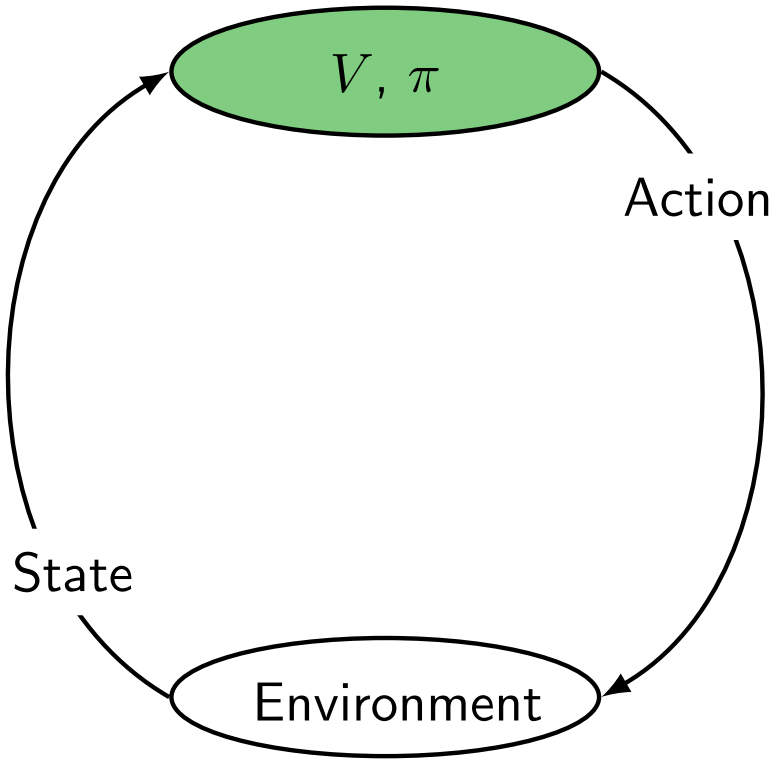






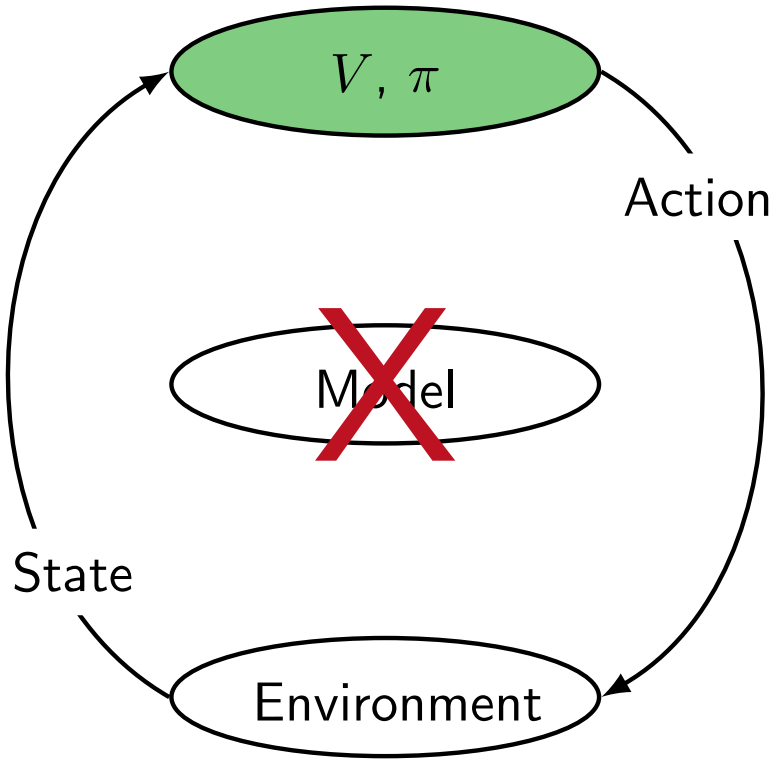
Learning and Planning

Learning and Planning



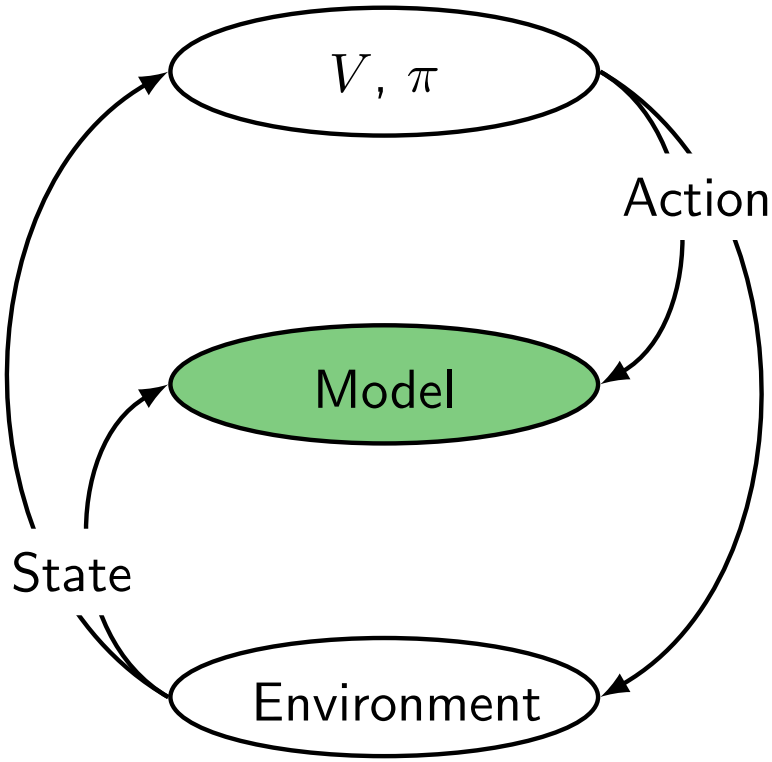
So far

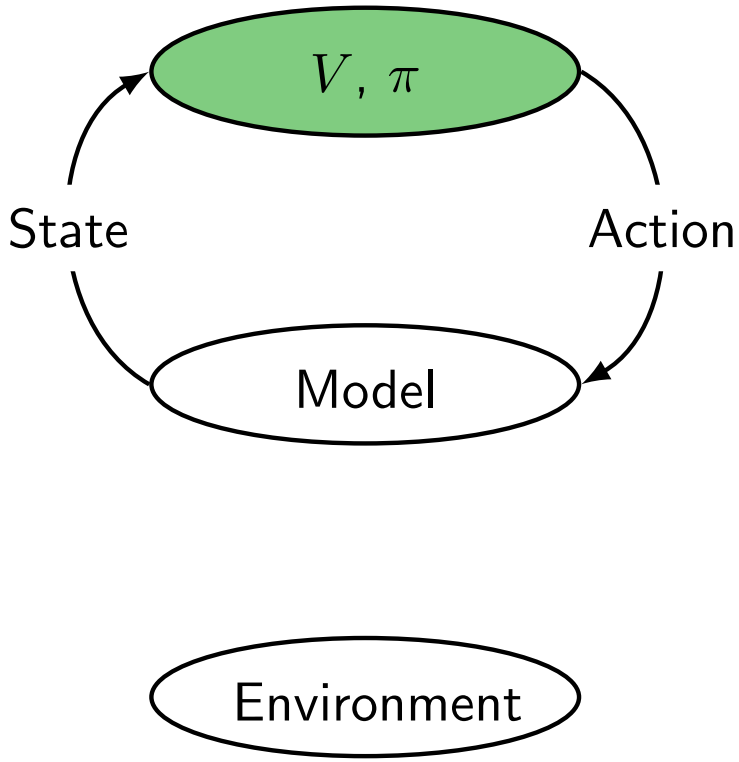
Learning and Planning



So far

Learning and Planning

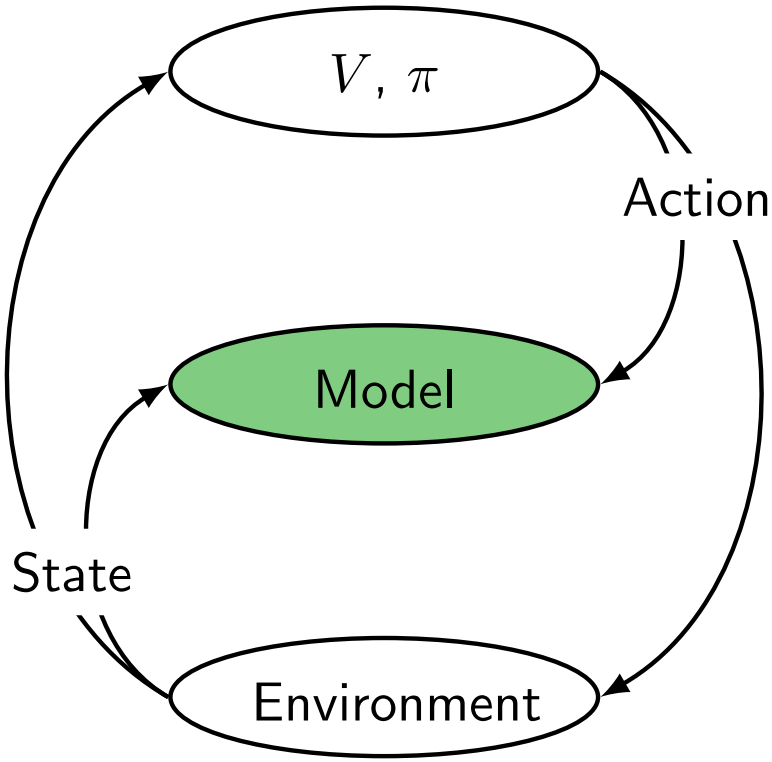


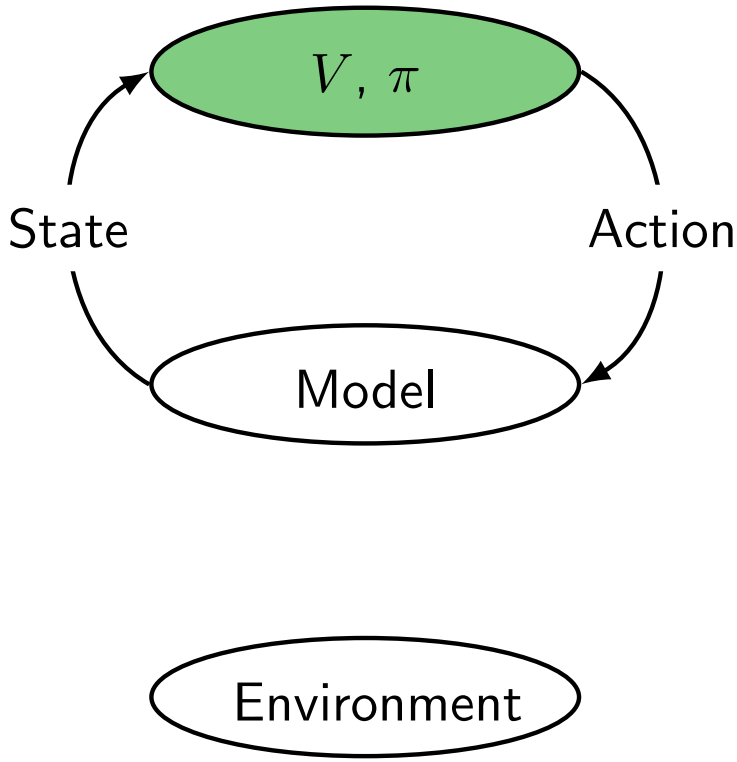


Learning by interaction with the environment

Planning by simulation

Learning and Planning



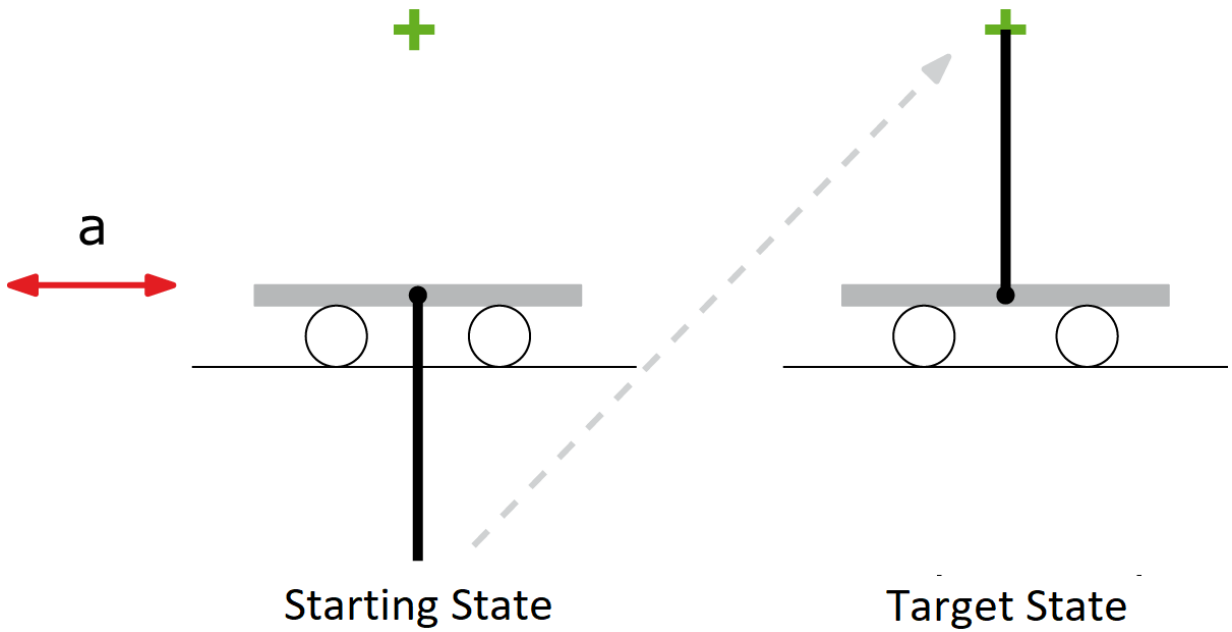


Learning by interaction with the environment ⟷ **Planning** by simulation

Application: Control of Dynamical Systems

Stabilization of Inverse Pendulum

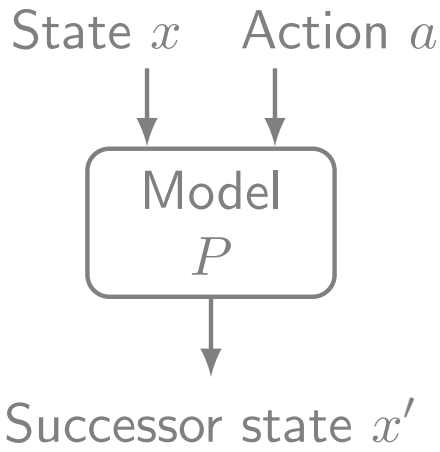
- Dynamical motion model unknown
  ➔ System of four ordinary differential equations
- **Actions:** acceleration of the cart
- Standard benchmark for control algorithms



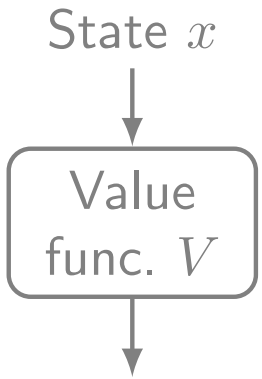
Application: Stabilization of Inverse Pendulum

Source: Marc Deisenroth, Imperial College London. `https://youtu.be/XiigTGKZfks`

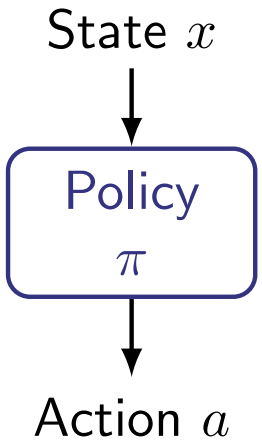
Model-based



Value function based



Policy Search



Basic Idea

Basic Idea

Parametric Policy

1. Representation of policy via function approximator $\pi(x|\underline{\theta})$ (instead of Q)
2. Solving optimization problems: $\underline{\theta}^* = \arg \max_{\underline{\theta}} V_\pi(x)$

Basic Idea

Parametric Policy

1. Representation of policy via function approximator $\pi(x|\underline{\theta})$ (instead of Q)
2. Solving optimization problems: $\underline{\theta}^* = \arg \max_{\underline{\theta}} V_\pi(x)$

Problems

- Non-convex optimization problem ➔ only sub-optimal solution
- No analytical gradient ➔ gradient approximation via sampling

Basic Idea

Parametric Policy

1. Representation of policy via function approximator $\pi(x|\underline{\theta})$ (instead of Q)
2. Solving optimization problems: $\underline{\theta}^* = \arg \max_{\underline{\theta}} V_\pi(x)$

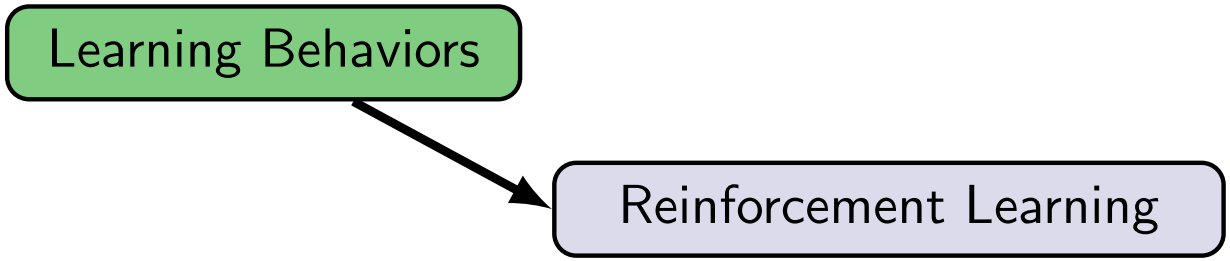
Problems

- Non-convex optimization problem ➔ only sub-optimal solution
- No analytical gradient ➔ gradient approximation via sampling

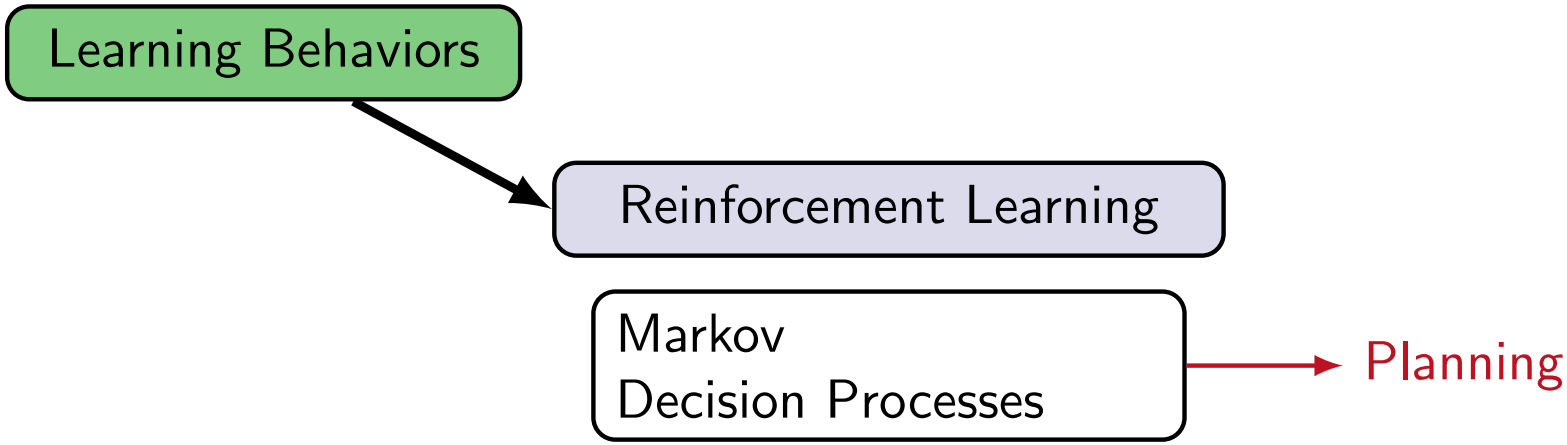
Meta-Algorithm

1. n runs with perturbed parameters $\underline{\theta}_1, \ldots, \underline{\theta}_n$
2. Calculate cumulative rewards $V_1, \ldots, V_n$ mit $V_i := \sum_t \gamma^t \cdot r_t$
3. Learn function approximator $f(\underline{\theta})$ such that $V_i \approx f(\underline{\theta}_i)$, $\forall i = 1 \ldots n$
4. Update parameter: $\underline{\theta} \leftarrow \underline{\theta} + \alpha \cdot \nabla_{\underline{\theta}} f(\underline{\theta})$

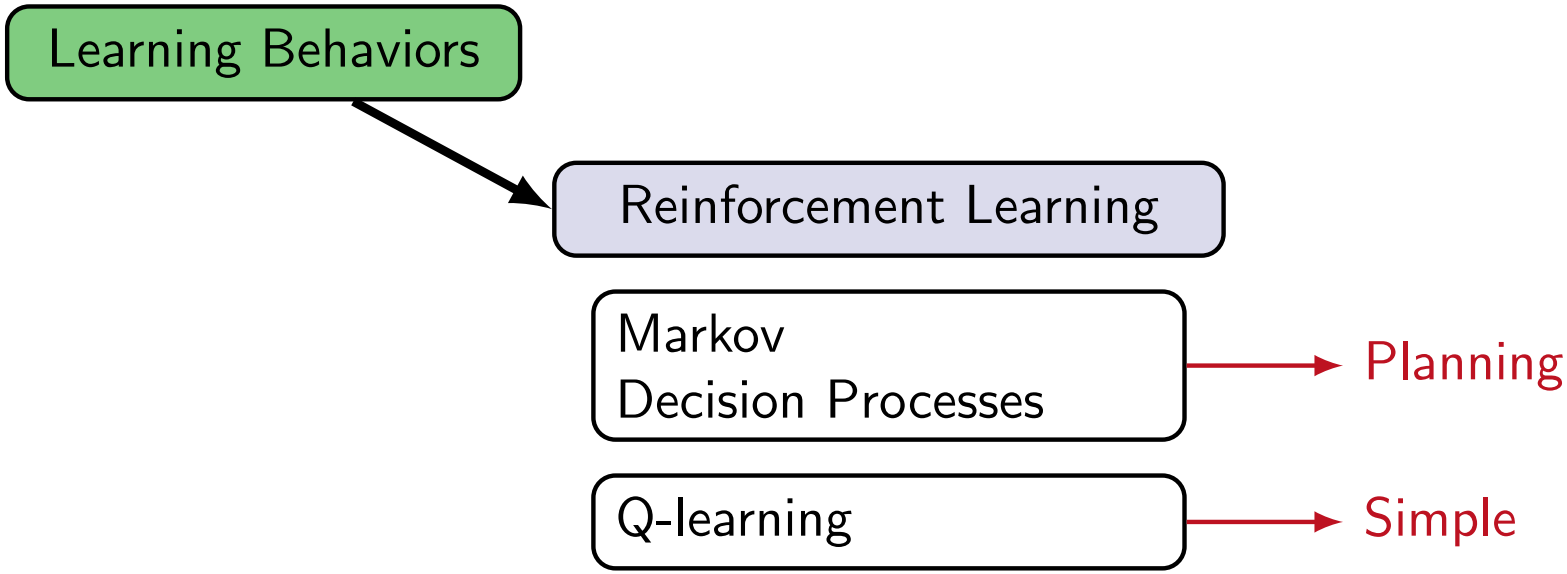
Summary



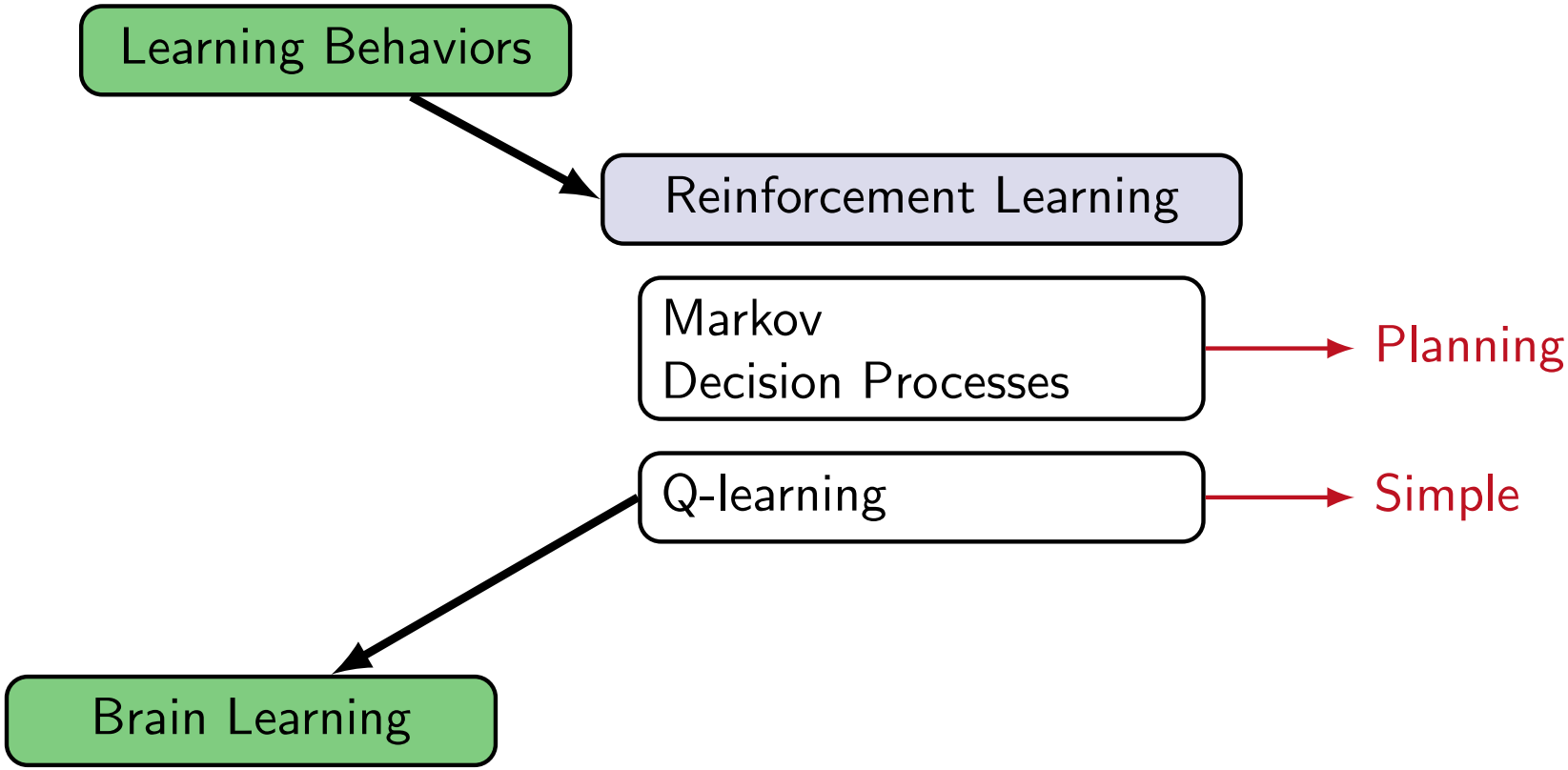
Summary


Learning Behaviors
Reinforcement Learning
Markov
Decision Processes
Planning

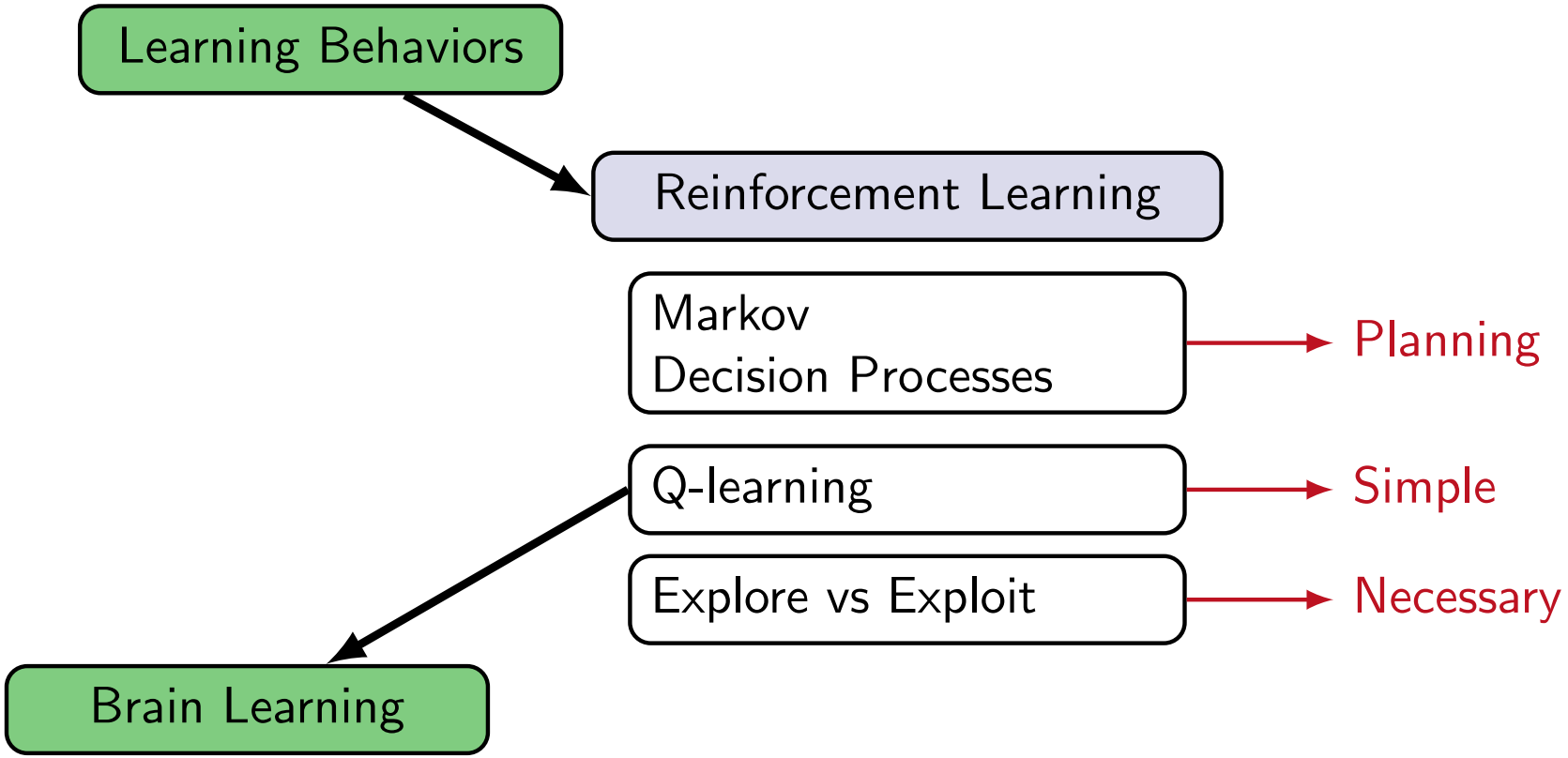
Summary


Learning Behaviors
Reinforcement Learning
Markov
Decision Processes
Planning
Q-learning
Simple

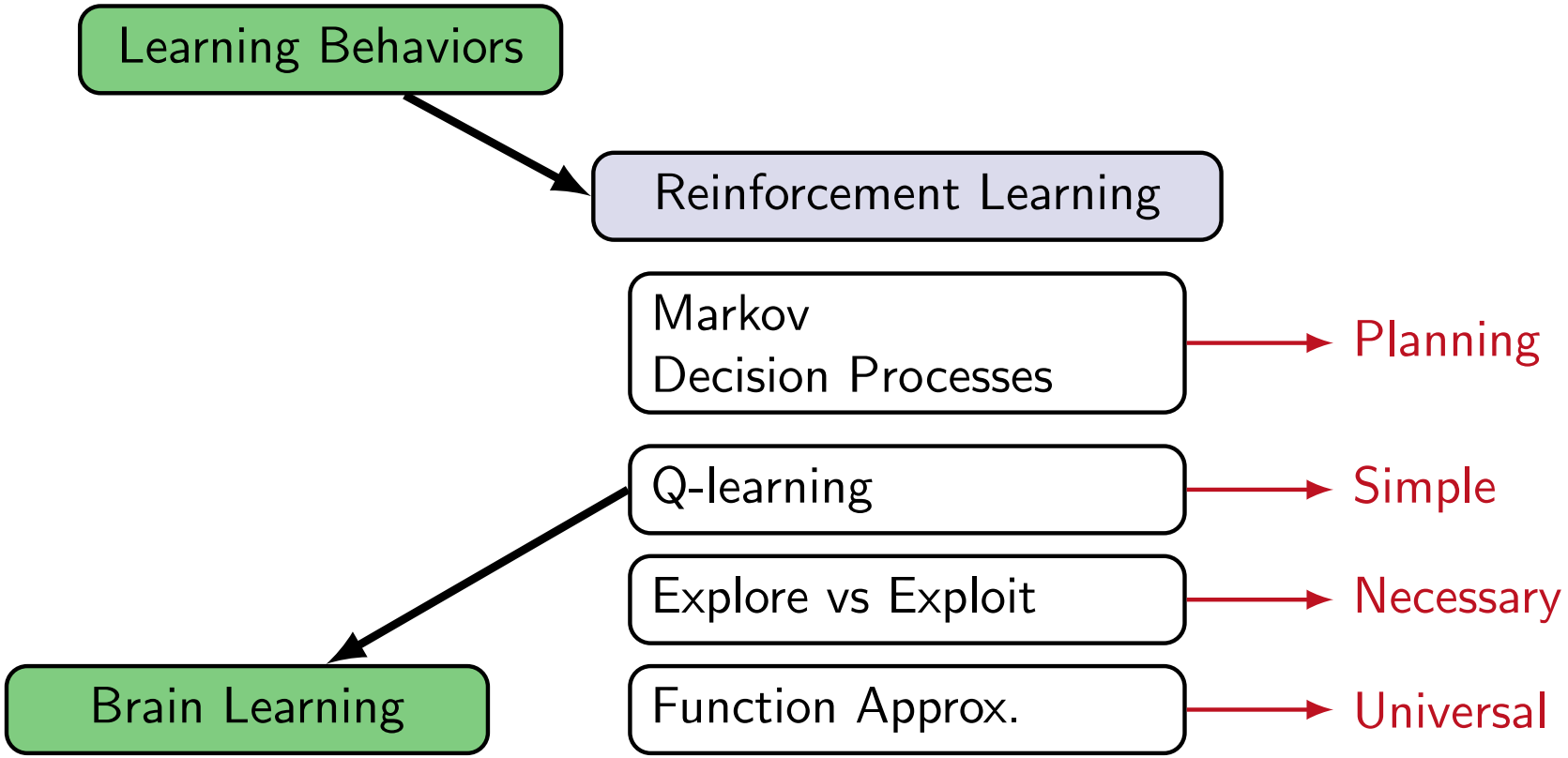
Summary


Learning Behaviors
Reinforcement Learning
Markov
Decision Processes
Planning
Q-learning
Simple
Brain Learning

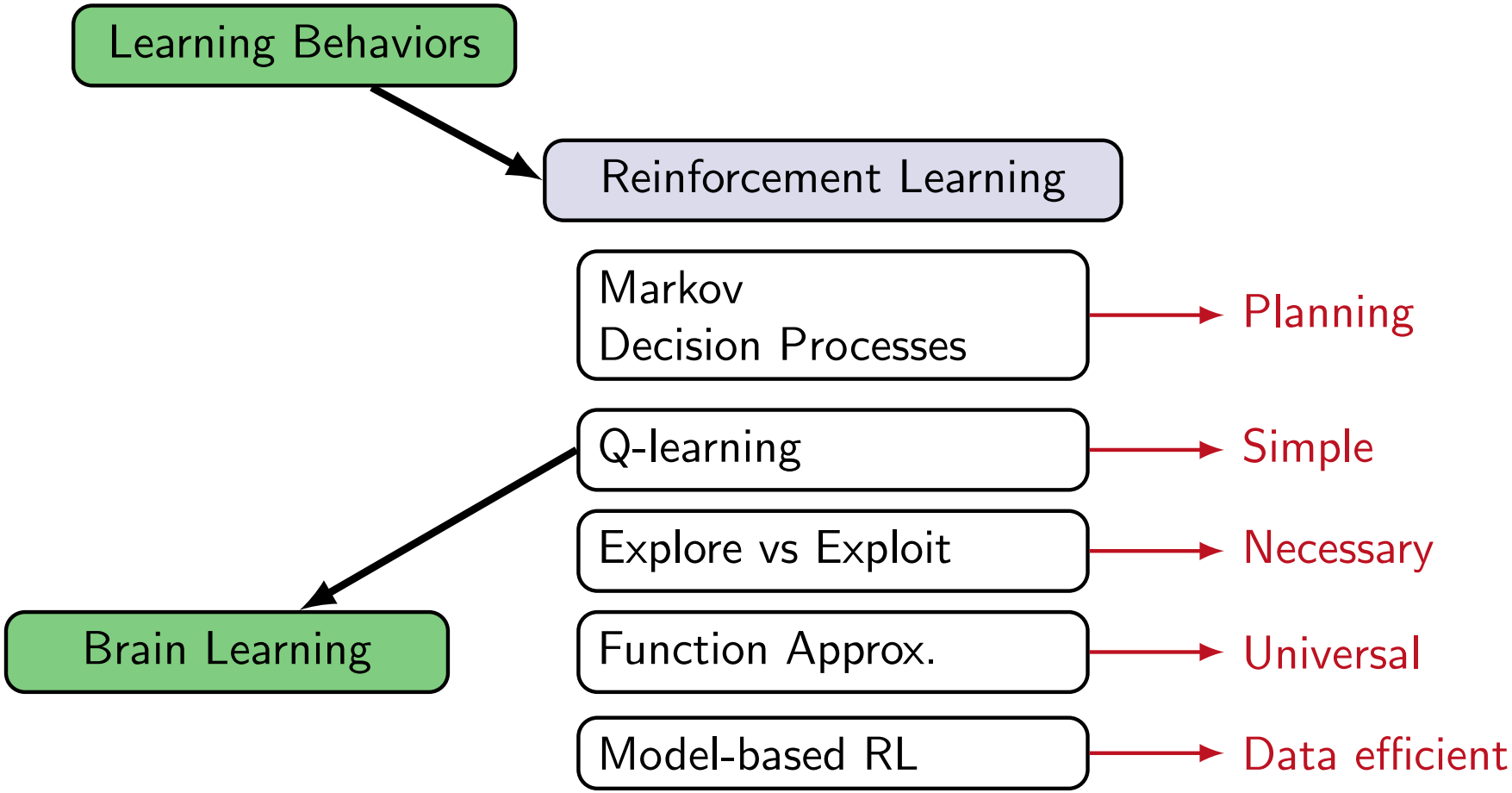
Summary


Learning Behaviors
Reinforcement Learning
Markov
Decision Processes
Planning
Q-learning
Simple
Explore vs Exploit
Necessary
Brain Learning

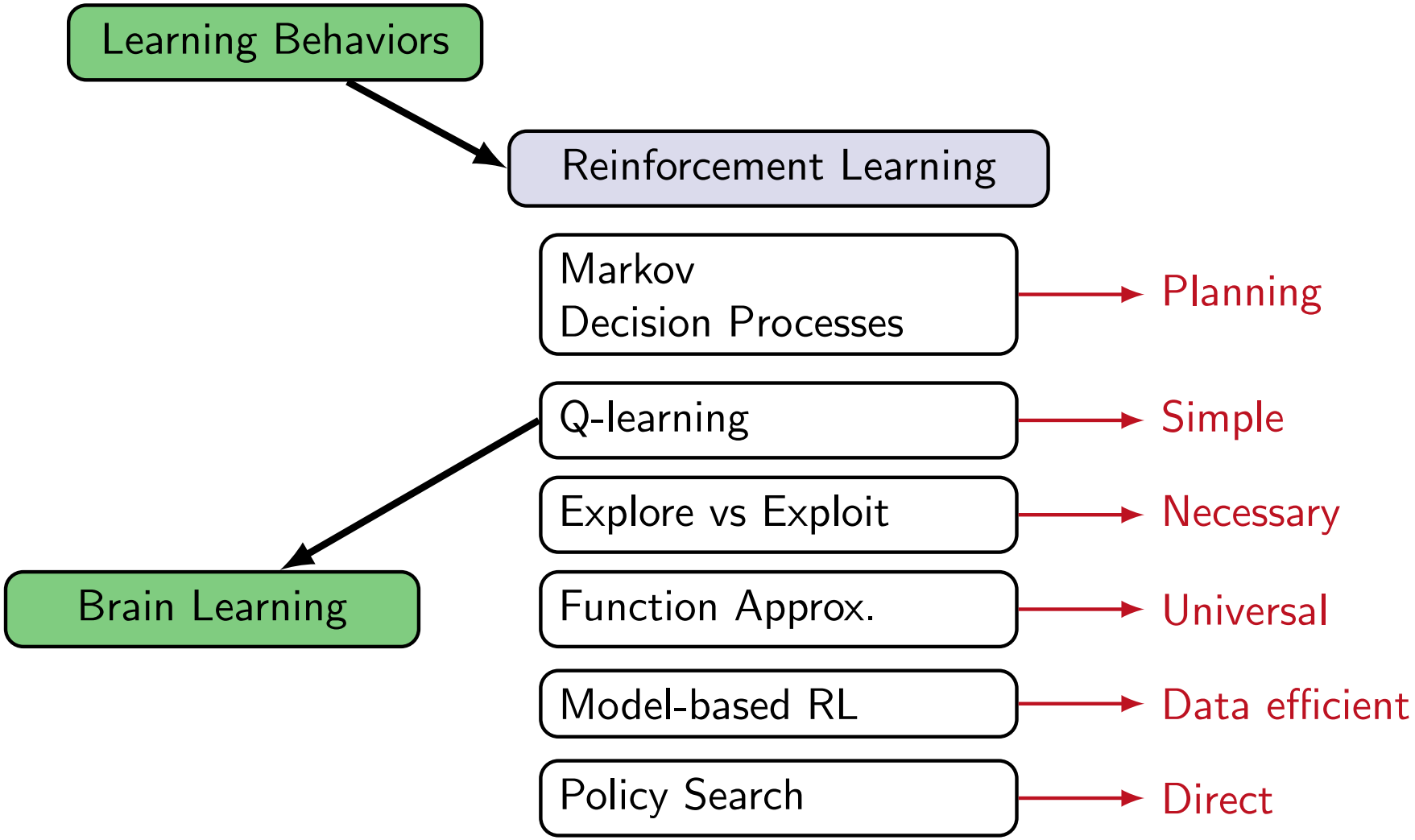
Summary


Learning Behaviors
Reinforcement Learning
Markov
Decision Processes
Planning
Q-learning
Simple
Explore vs Exploit
Necessary
Brain Learning
Function Approx.
Universal

Summary


Learning Behaviors
Reinforcement Learning
Markov Decision Processes
Planning
Q-learning
Simple
Explore vs Exploit
Necessary
Brain Learning
Function Approx.
Universal
Model-based RL
Data efficient

Summary


Learning Behaviors
Reinforcement Learning
Markov Decision Processes
Planning
Q-learning
Simple
Explore vs Exploit
Necessary
Brain Learning
Function Approx.
Universal
Model-based RL
Data efficient
Policy Search
Direct


i Love
Learning

References

Atkeson, C. G. und Santamaría, J. C. (1997). A Comparison of Direct and Model-Based Reinforcement Learning, *Proceedings of the International Conference on Robotics and Automation*.

Bertsekas, D. P. und Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*, Athena Scientific.

Busoniu, L., Babuska, R., Schutter, B. D. und Ernst, D. (2010). *Reinforcement Learning and Dynamic Programming Using Function Approximators*, CRC Press.

Deisenroth, M. P. (2009). *Efficient Reinforcement Learning Using Gaussian Processes*, Doktorarbeit, Karlsruher Institut für Technologie (KIT).

Doya, K. (2000). Complementary roles of basal ganglia and cerebellum in learning and motor control, *Current Opinion in Neurobiology* **10**: 732–739.

Doya, K. (2007). Reinforcement Learning: Computational theory and biological mechanisms, *HFSP Journal* **1**: 30–40.

Kober, J., Bagnelli, J. A. und Peters, J. (2013). Reinforcement Learning in Robotics: A Survey, *International Journal of Robotics Research* **32**(11): 1238–1274.

Melo, F. S. und Ribeiro, M. I. (2007). Convergence of Q-Learning with linear function approximation, *Proceedings of the European Control Conference*.

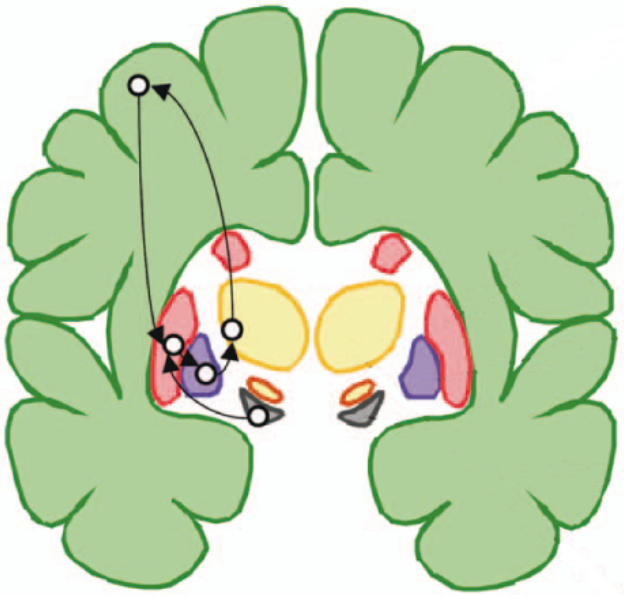
Mitchell, T. (1997). *Machine Learning*, McGraw-Hill.

Russel, S. und Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, Prentice Hall.

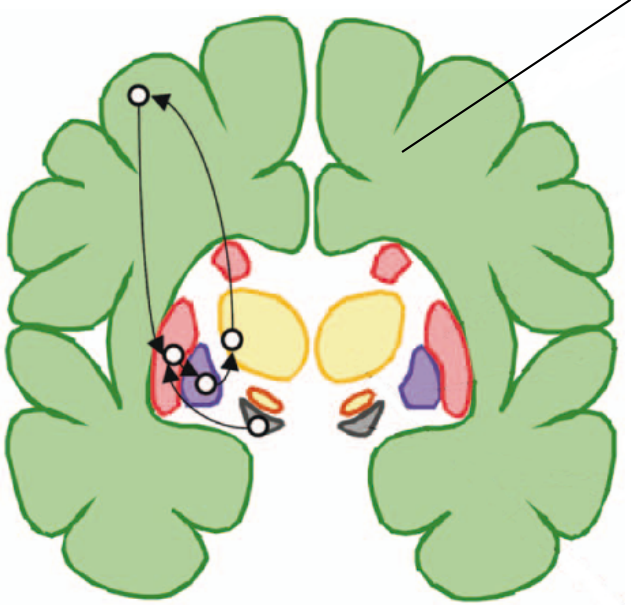
Singh, S., Jaakkola, T., Littman, M. L. und Szepesvári, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms, *Machine Learning* **39**: 287–308.

Sutton, R. S. und Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, The MIT Press.

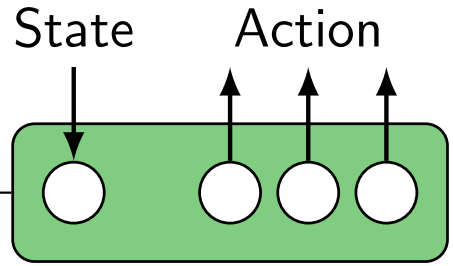
Reinforcement Learning inside the Brain



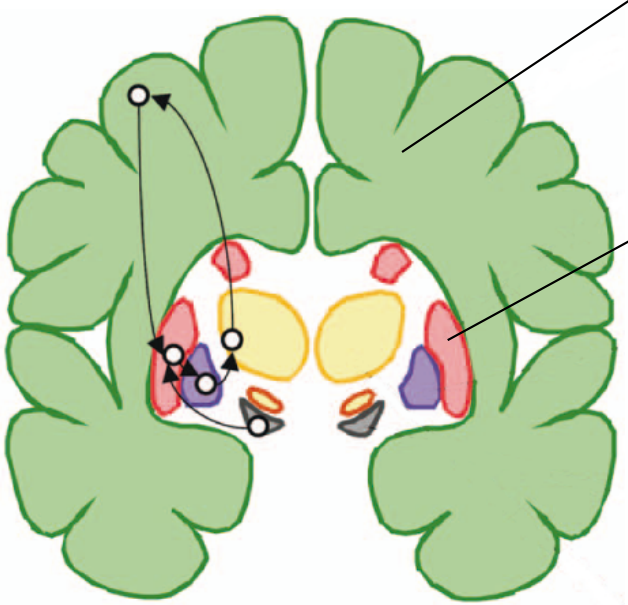
Reinforcement Learning inside the Brain



Cerebral Cortex
state-action coding

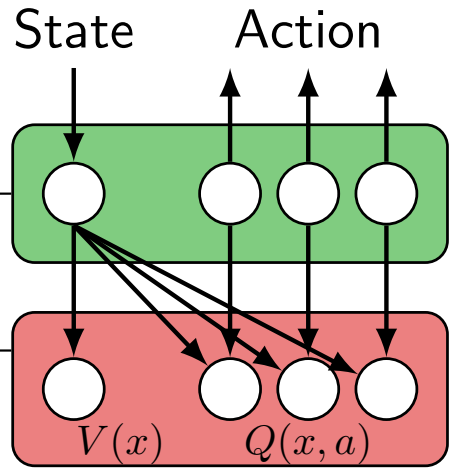


Reinforcement Learning inside the Brain

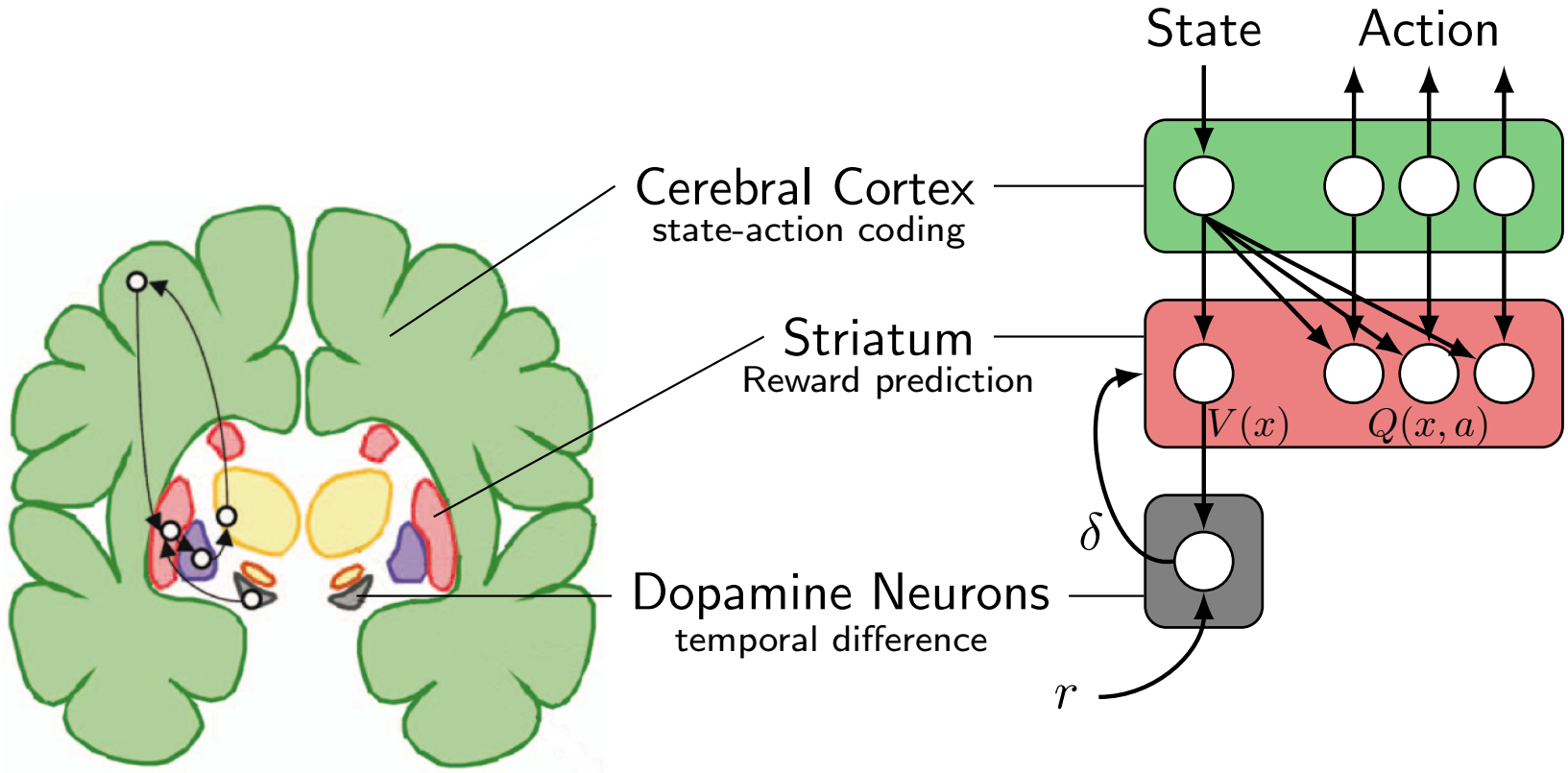


Cerebral Cortex
state-action coding

Striatum
Reward prediction

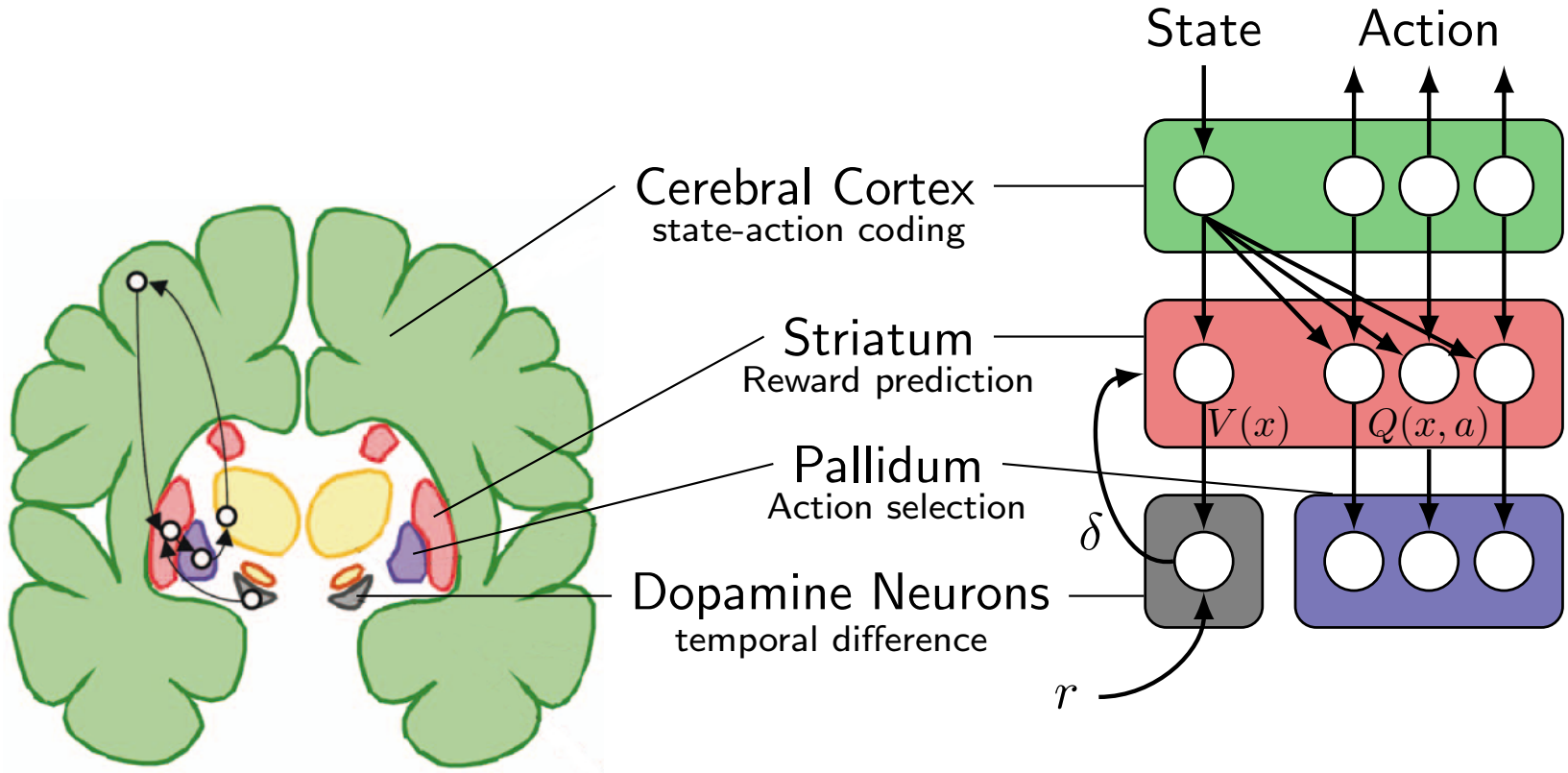


Reinforcement Learning inside the Brain



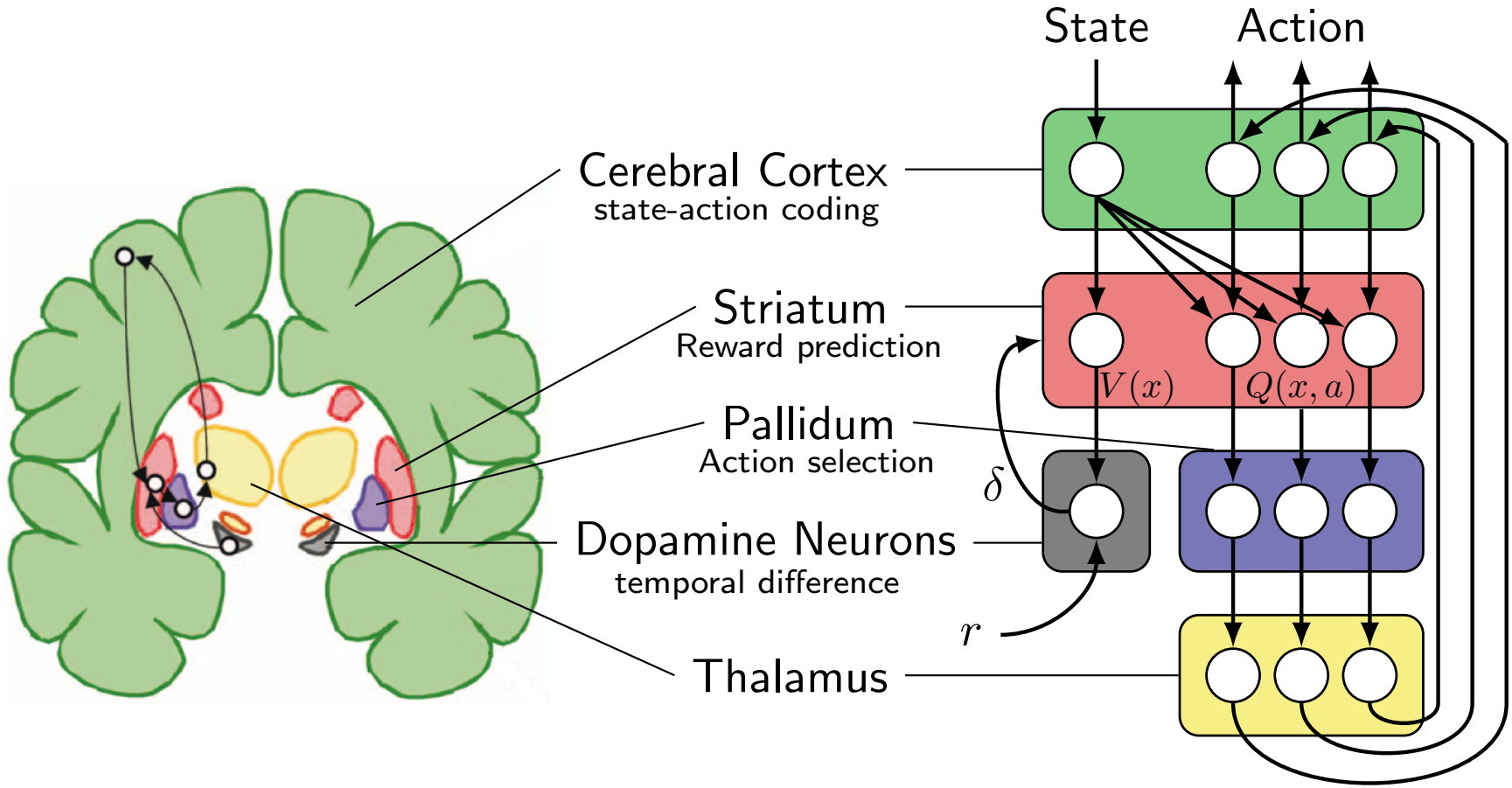
Value function: $$V(x) \leftarrow V(x) + \alpha \cdot \Big(\underbrace{r + \gamma \cdot V(x') - V(x)}_{\text{temporal difference } \delta(x)} \Big)$$

Reinforcement Learning inside the Brain



Value function: $$V(x) \leftarrow V(x) + \alpha \cdot \Big(\underbrace{r + \gamma \cdot V(x') - V(x)}_{\text{temporal difference } \delta(x)} \Big)$$

Reinforcement Learning inside the Brain



Kenji Doya: *Reinforcement Learning: Computational theory and biological mechanisms*, HFSP Journal, 2007.

Value function: $$V(x) \leftarrow V(x) + \alpha \cdot \Big(\underbrace{r + \gamma \cdot V(x') - V(x)}_{\text{temporal difference } \delta(x)} \Big)$$

Example: Finite State-Action Sets

1: Initialize $\hat{Q}$ arbitrary and $m(x, a, x') = n(x, a) = 0$
2: **while** true **do**
3: Choose initial state x
4: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
5: **for** k time steps **do**
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $m(x, a, x') \leftarrow m(x, a, x') + 1$
9: $n(x, a) \leftarrow n(x, a) + 1$
10: $x \leftarrow x'$
11: **end for**
12: $P(x'|x, a) \leftarrow \frac{m(x,a,x')}{n(x,a)}$
13: Solve MDP to obtain new Q-function $\hat{Q}$
14: **end while**

Example: Finite State-Action Sets

1: Initialize $\hat{Q}$ arbitrary and $m(x, a, x') = n(x, a) = 0$
2: **while** true **do**
3: Choose initial state x
4: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
5: **for** k time steps **do**
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $m(x, a, x') \leftarrow m(x, a, x') + 1$
9: $n(x, a) \leftarrow n(x, a) + 1$
10: $x \leftarrow x'$
11: **end for**
12: $P(x'|x, a) \leftarrow \frac{m(x,a,x')}{n(x,a)}$ **Learning**
13: Solve MDP to obtain new Q-function $\hat{Q}$
14: **end while**

Example: Finite State-Action Sets

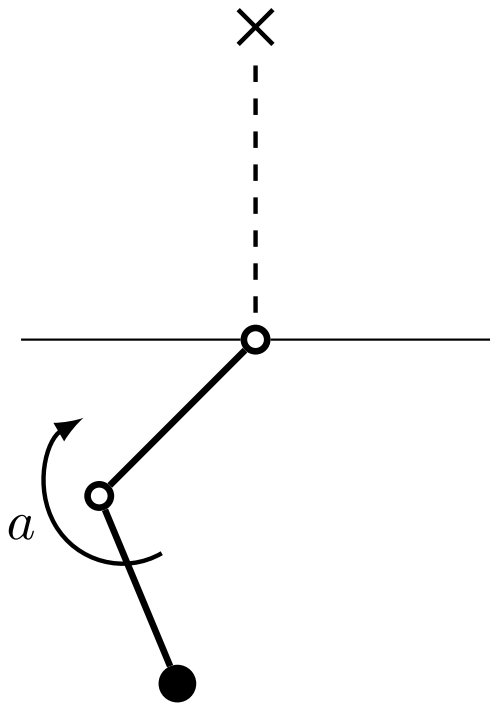
1: Initialize $\hat{Q}$ arbitrary and $m(x, a, x') = n(x, a) = 0$
2: **while** true **do**
3: Choose initial state x
4: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
5: **for** k time steps **do**
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $m(x, a, x') \leftarrow m(x, a, x') + 1$
9: $n(x, a) \leftarrow n(x, a) + 1$
10: $x \leftarrow x'$
11: **end for**
12: $P(x'|x, a) \leftarrow \frac{m(x,a,x')}{n(x,a)}$
13: Solve MDP to obtain new Q-function $\hat{Q}$ **Planning**
14: **end while**

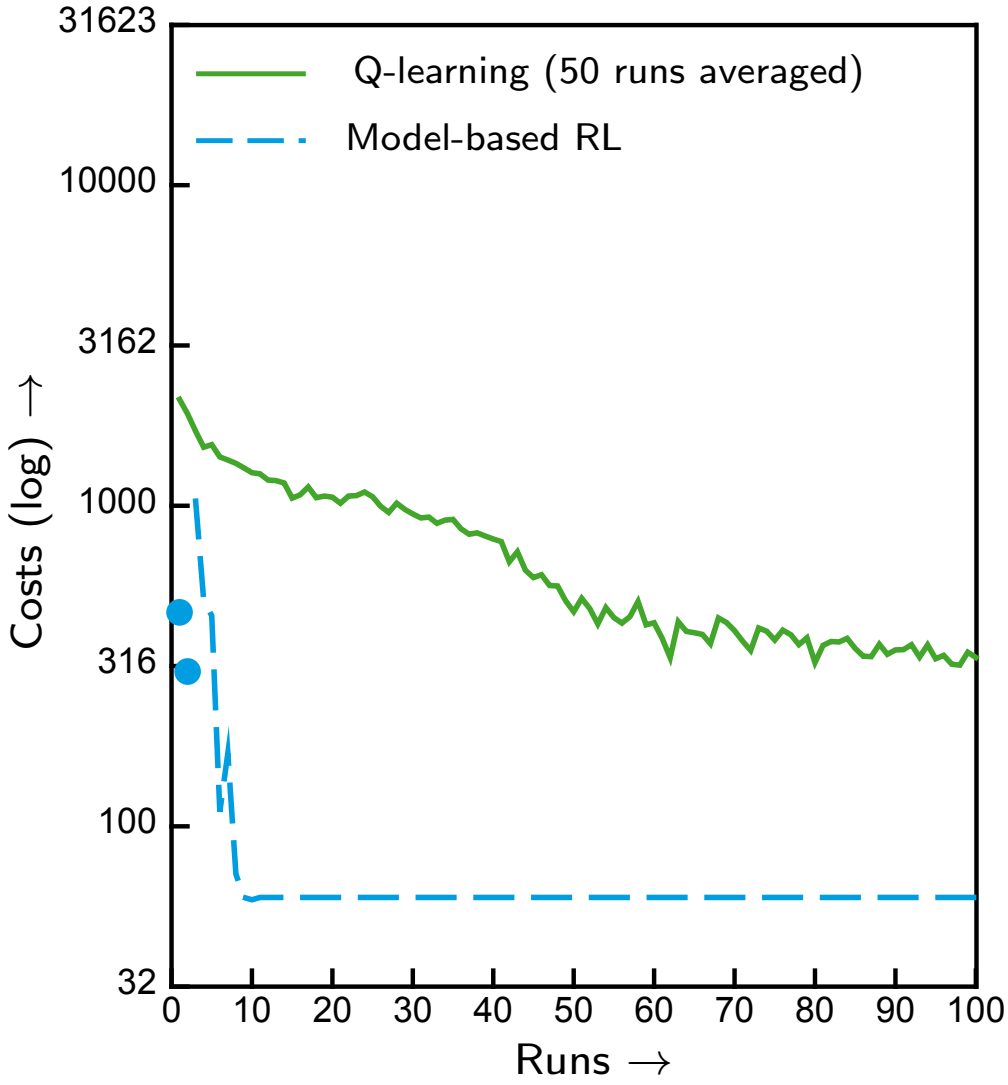
Example: Finite State-Action Sets

1: Initialize $\hat{Q}$ arbitrary and $m(x, a, x') = n(x, a) = 0$
2: **while** true **do**
3: Choose initial state x
4: Calculate policy π based on $\hat{Q}(x, a)$ and ϵ
5: **for** k time steps **do**
6: Choose action a from state x according to policy π
7: Execute a: observe reward r, successor state x'
8: $m(x, a, x') \leftarrow m(x, a, x') + 1$
9: $n(x, a) \leftarrow n(x, a) + 1$
10: $x \leftarrow x'$
11: **end for**
12: $P(x'|x, a) \leftarrow \frac{m(x,a,x')}{n(x,a)}$
13: Solve MDP to obtain new Q-function $\hat{Q}$
14: **end while**

Converges towards true model and thus, to Q^*/π^* (if GLIE)

Application: Acrobot





Atkeson und Santamaría (1997)

Inverses Reinforcement Learning

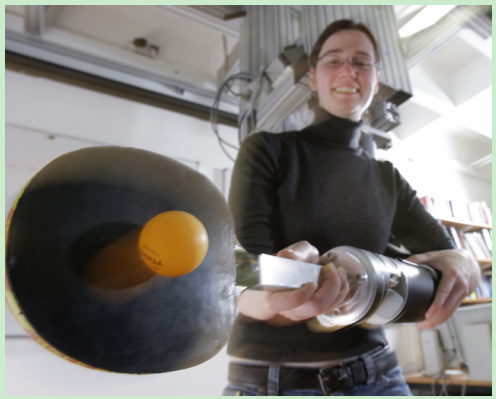
Annahme

- MDP mit bekannten Transitionswahrscheinlichkeiten
- **Aber:** Belohnungsfunktion ist unbekannt
- **Stattdessen:** Trajektorien $x_0, a_0, x_1, a_1, \ldots$ der optimalen Strategie π^*

Ziel

1. Lernen der Belohnungsfunktion
2. Bestimmen einer guten Strategie mittels gelernter Belohnungsfunktion

Anwendung: Programmieren durch Vormachen



http://www.ausy.tu-darmstadt.de

- Experte (sog. Teacher) demonstriert die zu lernende Aufgabe
- Kein überwachtes Lernen, d. h. Roboter soll den Experten *nicht* kopieren!
  ➔ geringe Generalisierung