MACHINE LEARNING

REINFORCEMENT LEARNING

AGENDA

01 Introduction

02 Markov Decision Processes

03 Policy and Value Function

04 Value Iteration and Policy Iteration

05 Learning MDP Models

06 Continuous state MDPs

AI

INTRODUCTION

REINFORCEMENT LEARNING

INTRODUCTION

There are many **problems**, where it is **very difficult** to **provide** the "**right**" **answers** to the algorithm, as we have been doing for supervised learning.

For example, if we have just built a **four-legged robot** and are **trying** to **program** it to **walk**, then initially we **have no idea** what the "**correct**" **actions** to take are to make it walk **(credit assignment problem).**

In **reinforcement learning**, we will **provide algorithms** only a **reward function**, to **indicate** when the **learning agent** is **doing well** or **poorly**.

It will be the learning algorithm's job to figure out how to choose actions over time to maximize rewards

REINFORCEMENT LEARNING

I N T R O D U C T I O N

Many of the **applications** of **reinforcement learning include**:

- **Autonomous helicopter flight.**
- **Robot Legged Locomotion.**
- **Cell-phone network routing.**
- **Marketing strategy selection.**
- **Factory control.**
- **Efficient web indexing.**
- **Traffic light control.**
- **Chemical reaction optimization.**
- **Personalized recommendations.**
- **Bidding and Advertisement.**

MARKOV DECISION PROCESSES

MARKOV DECISION PROCESSES

A **Markov decision process** is a **tuple**:

$$(\boldsymbol{S}, \boldsymbol{A}, \{\boldsymbol{P}_{\boldsymbol{sa}}\}, \boldsymbol{\gamma}, \boldsymbol{R})$$

- $\boldsymbol{S}$**:** the set of **states** (i.e. all positions and orientations of a helicopter).
- $\boldsymbol{A}$: the set of **actions** (i.e. all directions in which you can push the helicopter's sticks).
- $\boldsymbol{P}_{\boldsymbol{sa}}$: state **transition probability distributions**. For **each state** $\boldsymbol{s} \in \boldsymbol{S}$ and action $\boldsymbol{a} \in \boldsymbol{A}$**,** $\boldsymbol{P}_{\boldsymbol{sa}}$ is a **distribution** over the **state space.** Thus, $\boldsymbol{P}_{\boldsymbol{sa}}$ **gives** the **distribution** over what **states** the **algorithm** will **transition,** if it takes **action** $\boldsymbol{a}$ in **state** $\boldsymbol{s}$.

$$\sum_{\boldsymbol{s'}} \boldsymbol{P}_{\boldsymbol{sa}}(\boldsymbol{s'}) = \mathbf{1}$$

REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES

A **Markov decision process** is a **tuple**:

$$(\boldsymbol{S}, \boldsymbol{A}, \{\boldsymbol{P}_{\boldsymbol{sa}}\}, \boldsymbol{\gamma}, \boldsymbol{R})$$

- $\boldsymbol{\gamma} \in [\mathbf{0}, \mathbf{1})$: is called the **discount factor**.
- $\boldsymbol{R}: \boldsymbol{S}\ \boldsymbol{X}\ \boldsymbol{A} \rightarrow \mathbb{R}$: the **reward function**. (It can be written only as a function of $\boldsymbol{S}$, $\boldsymbol{R}: \boldsymbol{S} \rightarrow \mathbb{R}$).

REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES

EXAMPLE:

We will **define** a **robot navigation task**, in which you **have** a **robot** in a **grid environment** where **all gray cells** are **obstacles**.

The **reward** is **defined by** the +1 and the **punishment** as -1.

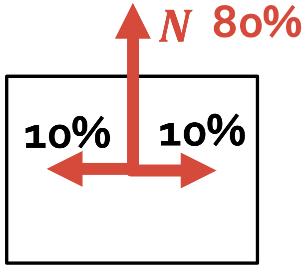
The **robot can move** in **all cells except** for the **gray cells.**

11 states

			+1
			-1

STOCHASTIC ACTIONS

$$\boldsymbol{A} = \{\boldsymbol{N}, \boldsymbol{W}, \boldsymbol{S}, \boldsymbol{E}\}$$



REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES

EXAMPLE:

The **robot** can **transit from** one **state** to another **state** in **single steps.**

When the **robot reaches (4,3) or (4,2)** the **world ends**.

$(1,3)$	$(2,3)$	$(3,3)$	$(4,3)$ **+1**
$(1,2)$	$(2,2)$	$(3,2)$ 80%	$(4,2)$ **-1**
$(1,1)$	$(2,1)$	10% 10% $(3,1)$	$(4,1)$

Transitions

$$P_{s_t a_t}(s')$$

$$P_{(3,1)N}\big((3,2)\big) = 0.8$$

$$P_{(3,1)N}\big((4,1)\big) = 0.1$$

$$P_{(3,1)N}\big((2,1)\big) = 0.1$$

$$P_{(3,1)N}\big((3,3)\big) = 0$$

$$\vdots$$

Rewards

$$R(s)$$

$$R\big((4,3)\big) = +1$$

$$R\big((4,2)\big) = -1$$

$$R(s) = -0.2$$

(fuel consumption)

$$\vdots$$

REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES

The dynamics of a **Markov decision process** is as follows:

1. The **algorithm starts** in a **state** $\boldsymbol{s_0}$ and **choose** an **action** $\boldsymbol{a_0} \in \boldsymbol{A}$.

2. There is a **random transition from state** $\boldsymbol{s_0}$ to **state** $\boldsymbol{s_1}$ **drawn according** to $\boldsymbol{s_1 \sim P_{s_0 a_0}}$.

3. The **process repeats indefinitely** where there is a **random selection** to **traverse** from state $\boldsymbol{s_t}$ to state $\boldsymbol{s_{t+1}}$, which is **drawn from** $\boldsymbol{s_{t+1} \sim P_{s_t a_t}}$.

$$\boldsymbol{s_0 \underset{a_0}{\rightarrow} s_1 \underset{a_1}{\rightarrow} s_2 \underset{a_2}{\rightarrow} \ldots}$$

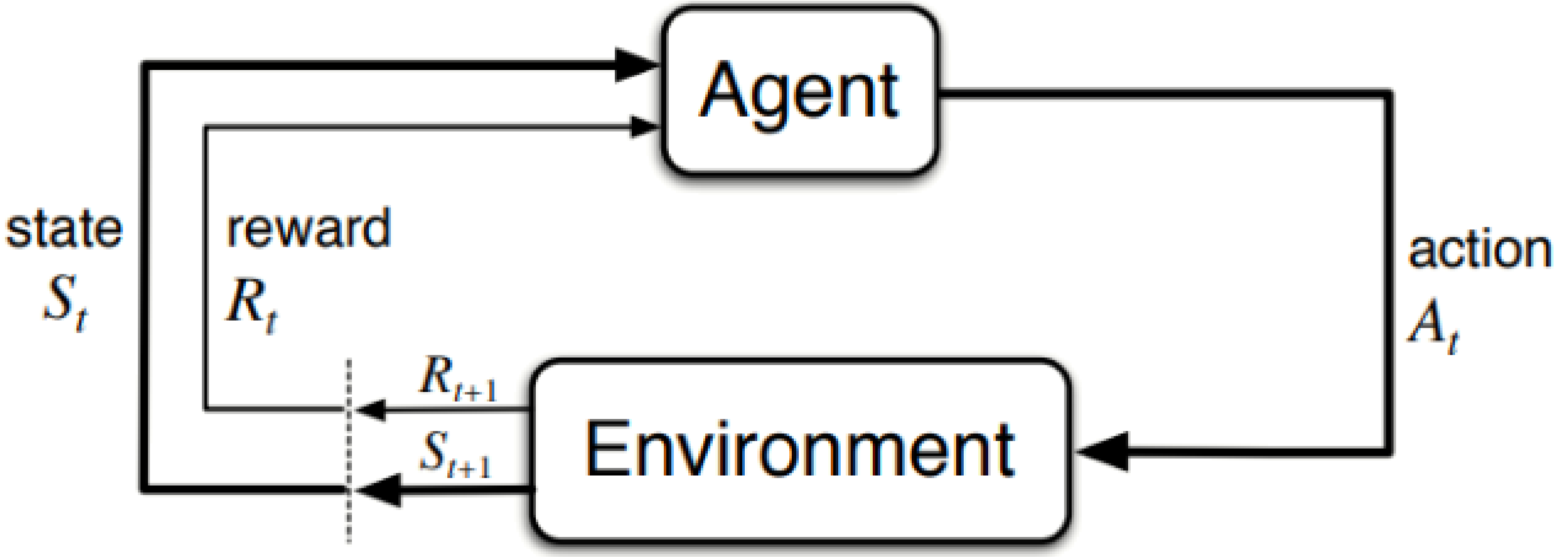
4. The **total payoff** is be given by:

$$\boldsymbol{R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots}$$

(Rewards diminish as times goes by).

REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES



REINFORCEMENT LEARNING

MARKOV DECISION PROCESSES

The **objective** in **reinforcement learning** is to **choose actions over time** to **maximize** the **expected value** of the **total payoff**:

$$E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots\right]$$

The **reward** at **timestep** t is **discounted** by a **factor** of γ^t where $\gamma \in [0,1)$ is the **discount factor**.

Therefore, to **make** this **expectation large**, we would like to:

- **Obtain positive rewards** as **soon** as **possible**.
- **Postpone negative rewards,** as **long as possible**.

POLICY AND VALUE FUNCTION

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

POLICY

A **policy** is **any function** $\pi : S \rightarrow A$ that **maps from states** to **actions (for every state, what action** is **recommended** to **take** in **that state)**.

EXAMPLE: (**Optimal Policy**)

$(1,3)$ →	$(2,3)$ →	$(3,3)$ →	+1
$(1,2)$ ↑	$(2,2)$	$(3,2)$ ↑	-1
$(1,1)$ ↑	$(2,1)$ ←	$(3,1)$ ←	$(4,1)$ ←

MDPs are very good in finding the best tradeoffs between actions (policies).

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

VALUE FUNCTION:

For **any policy** $\boldsymbol{\pi}$, we **define** the **value function** $\boldsymbol{V^{\pi}}$: $\boldsymbol{S} \rightarrow \mathbb{R}$ as the **expected total payoff** upon **starting** in **state** $\boldsymbol{s}$ and **taking actions according** to the **policy** $\boldsymbol{\pi}$**.**

$$\boldsymbol{V^{\pi}(s) = E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi\right]}$$

As a **side note**, this **notation** is **technically incorrect** because we **cannot condition** on $\boldsymbol{\pi}$, which **does not represent** a **random variable**.

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

EXAMPLE: Value Function and Policy

Very bad policy π

$(1,3)$ →	$(2,3)$ →	$(3,3)$ →	+1
$(1,2)$ ↓	$(2,2)$	$(3,2)$ →	-1
$(1,1)$ →	$(2,1)$ →	$(3,1)$ ↑	$(4,1)$ ↑

Value Function $V^{\pi}(s)$

$V^{\pi}((1,3))$ 0.52	$V^{\pi}((2,3))$ 0.73	$V^{\pi}((3,3))$ 0.77	+1
$V^{\pi}((1,2))$ -0.90		$V^{\pi}((3,2))$ -0.82	-1
$V^{\pi}((1,1))$ -0.88	$V^{\pi}((2,1))$ -0.87	$V^{\pi}((3,1))$ -0.85	$V^{\pi}((4,1))$ -1

$$V^{\pi}(s) = E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s, \pi\right]$$

Better to start at the top

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

BELLMAN EQUATIONS:

We can **represent** the **value function** in the **following way**:

$$V^{\pi}(s) = E\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s, \pi\right]$$

$$V^{\pi}(s) = E[R(s_0) + \gamma \underbrace{(R(s_1) + \gamma R(s_2) + \cdots)}_{\substack{\text{Expected sum of future} \\ \text{discounted rewards.} \\ V^{\pi}(s_1)}} | s_0 = s, \pi]$$

Immediate reward
for starting in state s

Therefore, we can **write** the **equation** as a **recursive form** of **itself** ($s_0 \rightarrow s$ and $s_0 \rightarrow s'$) **using** the **definition** of **expected value** $E[X] = \sum x_i p_i$ (s' is a **random variable**):

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s')\, V^{\pi}(s')$$

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

BELLMAN EQUATIONS:

Given a **fixed policy** $\boldsymbol{\pi}$**,** its **value function** $\boldsymbol{V^{\pi}}$ **satisfies Bellman equations**:

$$\boldsymbol{V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s')\, V^{\pi}(s')}$$

The **first term** is the **immediate reward** for **starting** in **state** $\boldsymbol{s}$**.**

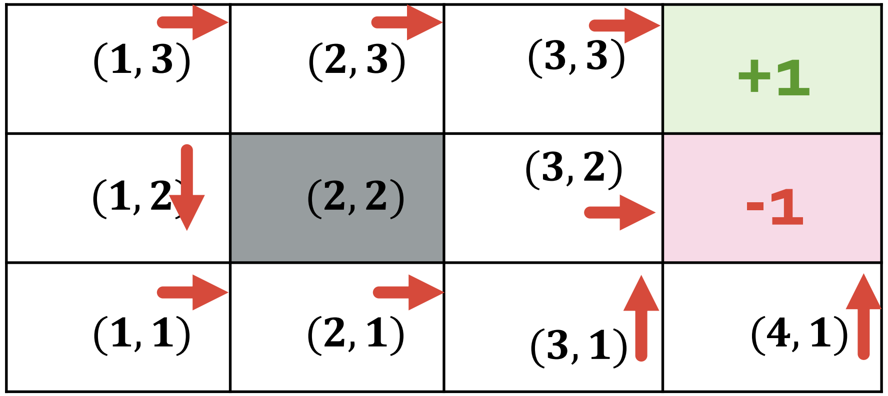
The **second term** gives the **expected sum** of **discounted rewards** obtained after the **first step** in the **MDP.**

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

EXAMPLE: Bellman Equation

Let us **assume** that we **start** at **state** (**3,1**) and the **policy** $\boldsymbol{\pi}$ **indicates** to take a **step** in the **north direction** $\boldsymbol{\pi}\big((3,1)\big) = \boldsymbol{N}$.



The **Bellman Equation** is:

$$\boldsymbol{V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s')\, V^{\pi}(s')}$$

$$\boldsymbol{V^{\pi}\big((3,1)\big) = R\big((3,1)\big) + \gamma\big[0.8V^{\pi}\big((3,2)\big) + 0.1V^{\pi}\big((4,1)\big) + 0.1V^{\pi}\big((2,1)\big)\big]}$$

You will have **11 variables** and **11 equations**, **each** one **corresponding** to **each state**.

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

OPTIMAL VALUE FUNCTION:

The **optimal value function** will be the **best possible expected sum** of **discounted rewards** that can be attained **using any policy**.

$$\boldsymbol{V^*(s) = \max_{\pi} V^{\pi}(s)}$$

The **Bellman representation** will be:

Expected future rewards if we take the BEST action $\boldsymbol{a}$ in state $\boldsymbol{s}$.

$$\boldsymbol{V^*(s) = R(s) + \boxed{\max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')\, V^*(s')}}$$

The **first term** above is the **immediate reward**. The **second term** is the **maximum over all actions $\boldsymbol{a}$** of the **expected future sum** of **discounted rewards** we'll get **upon after action $\boldsymbol{a}$.**

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

OPTIMAL POLICY:

The **optimal policy** π^* indicates the **best action** a to **take** in **state** s, which **maximizes** the **expected future rewards**.

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')\, V^*(s')$$

This **derives** in the **following expression**:

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

In conclusion, π^* is the **optimal policy for all states** s.

We can use the same policy π^* no matter what the initial state of our MDP is.

REINFORCEMENT LEARNING

POLICY AND VALUE FUNCTION

OPTIMAL POLICY:

To **find** the **optimal policy** $\boldsymbol{\pi}^*(\boldsymbol{s})$, we would **need** to **obtain** the **best value function** $\boldsymbol{V}^*(\boldsymbol{s}')$ and **substitute it** in the **equation** we have just obtained:

$$\boldsymbol{\pi}^*(\boldsymbol{s}) = \underset{\mathbf{a} \in \mathbf{A}}{\mathbf{arg\,max}} \sum_{\boldsymbol{s'} \in \boldsymbol{S}} \boldsymbol{P}_{\boldsymbol{sa}}(\boldsymbol{s}')\,\boldsymbol{V}^*(\boldsymbol{s}')$$

The **problem** is **finding** $\boldsymbol{V}^*(\boldsymbol{s}')$ which corresponds to:

$$\boldsymbol{V}^*(\boldsymbol{s}) = \max_{\boldsymbol{\pi}} \boldsymbol{V}^{\boldsymbol{\pi}}(\boldsymbol{s})$$

The **bottleneck** is that you would **need** to **solve** an **exponentially large number** of **systems** of **equations because** there is an **exponential number** of **policies** $\boldsymbol{\pi}$.

VALUE ITERATION AND POLICY ITERATION

REINFORCEMENT LEARNING

VALUE ITERATION AND POLICY ITERATION

VALUE ITERATION:

1. **For each state** s, **initialize** $V(s) := 0 \ \forall s$.

2. **Repeat until convergence**{

   For every state, **update** $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$.

   }

3. **Substitute** $V^*(s)$ in $\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$.

The **algorithm repeatedly tries** to **update** the **estimated value function** using **Bellman Equations.** By **implementing this** you **ensure** that the **value function converges** to the **optimum**.

$$V(s) \rightarrow V^*(s)$$

REINFORCEMENT LEARNING

VALUE ITERATION AND POLICY ITERATION

VALUE ITERATION:

There are **two possible ways** of **performing** the **updates** in the **inner loop**.

$$\boldsymbol{V}\,(\boldsymbol{s}) := \boldsymbol{R}(\boldsymbol{s}) + \max_{\boldsymbol{a}\in\boldsymbol{A}} \boldsymbol{\gamma} \sum_{\boldsymbol{s}'\in\boldsymbol{S}} \boldsymbol{P}_{\boldsymbol{sa}}(\boldsymbol{s}')\,\boldsymbol{V}(\boldsymbol{s}').$$

1. **Synchronous update** (current estimate → new estimate):
   - **Compute** the **new values** of $V(s)$.
   - **Overwrite** the **old values with** the **new values**.

2. **Asynchronous updates**
   - **Loop over** the **states** (**predefined order**).
   - **Update** the **values** of $V(s)$ **one** at a **time**.

VALUE ITERATION AND POLICY ITERATION

EXAMPLE: Value Iteration

Running value iteration on our **previous example**, we obtain the **following results** for $V^*(s)$:

$\boxed{V^*(s)}$

$V^\pi((1,3))$ $\mathbf{0.86}$	$V^\pi((2,3))$ $\mathbf{0.90}$	$V^\pi((3,3))$ $\mathbf{0.93}$	**+1**
$V^\pi((1,2))$ $\mathbf{0.82}$		$V^\pi((3,2))$ $\mathbf{0.69}$	**-1**
$V^\pi((1,1))$ $\mathbf{0.78}$	$V^\pi((2,1))$ $\mathbf{0.75}$	$V^\pi((3,1))$ $\mathbf{0.71}$	$V^\pi((4,1))$ $\mathbf{0.49}$

REINFORCEMENT LEARNING

VALUE ITERATION AND POLICY ITERATION

EXAMPLE: Value Iteration

Let us see how we **obtain $\pi^*(s)$** for a **single state:** $(3,1)$

$$W = \sum_{s' \in S} P_{sa}(s')V(s') = (0.8 * 0.75) + (0.1 * 0.69) + (0.1 * 0.71) = 0.78$$

The action west maximizes our future rewards if we start at state (3,1).

$$N = \sum_{s' \in S} P_{sa}(s')V(s') = (0.8 * 0.69) + (0.1 * 0.75) + (0.1 * 0.49) = 0.67$$

$$S = \sum_{s' \in S} P_{sa}(s')V(s') = (0.8 * 0.71) + (0.1 * 0.75) + (0.1 * 0.49) = 0.69$$

$$E = \sum_{s' \in S} P_{sa}(s')V(s') = (0.8 * 0.49) + (0.1 * 0.69) + (0.1 * 0.71) = 0.53$$

$\boxed{V^*(s)}$

$V^\pi((1,3))$ 0.86	$V^\pi((2,3))$ 0.90	$V^\pi((3,3))$ 0.93	+1
$V^\pi((1,2))$ 0.82		$V^\pi((3,2))$ 0.69	-1
$V^\pi((1,1))$ 0.78	$V^\pi((2,1))$ 0.75	$V^\pi((3,1))$ 0.71	$V^\pi((4,1))$ 0.49

EXAMPLE: Value Iteration

Applying the **previous procedure** to **all states,** we **obtain** our **optimal policy** (**best actions** $\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s})$ to take **at every state** $\boldsymbol{s}$**).**

$\boxed{\boldsymbol{\pi^*(s)}}$

$(\mathbf{1,3})$ →	$(\mathbf{2,3})$ →	$(\mathbf{3,3})$ →	**+1**
$(\mathbf{1,2})$ ↑	$(\mathbf{2,2})$	$(\mathbf{3,2})$ ↑	**-1**
$(\mathbf{1,1})$ ↑	$(\mathbf{2,1})$ ←	$(\mathbf{3,1})$ ←	$(\mathbf{4,1})$ ←

VALUE ITERATION AND POLICY ITERATION

POLICY ITERATION:

1. **Initialize π randomly.**
2. **Repeat until convergence**{

   a) Let $V := V^{\pi}$ **(Solve Bellman equations using policy π).**

   b) For every state, **compute** $\pi(s) := R(s) \underset{a \in A}{\operatorname{argmax}} \sum_{s' \in S} P_{sa}(s') V(s')$.

}

The **policy π found** in **step (b)** is also **called** the **policy** that is **greedy** with **respect** to V.
By **implementing this** you **ensure** that the **value function converges** to the **optimum**.

$$V(s) \rightarrow V^{*}(s)$$
$$\pi(s) \rightarrow \pi^{*}(s)$$

REINFORCEMENT LEARNING

VALUE ITERATION AND POLICY ITERATION

POLICY ITERATION VS VALUE ITERATION:

POLICY ITERATION	VALUE ITERATION
For **small MDPs** it is **very fast.**	For **small MDPs** it is **slower.**
For **MDPs** with **large state spaces** it is **computationally expensive** (large matrices)	For **MDPs** with **large state spaces** it is has **less computational expense**.

Value iteration seems to be **used more often** than policy iteration.

LEARNING MDP MODELS

REINFORCEMENT LEARNING

LEARNING MDP MODELS

In **many realistic problems**, we are **not given** state **transition probabilities** and **rewards explicitly**. Therefore, we need to **estimate them from data.**

More generally, in a Markov Decision Process $(\boldsymbol{S}, \boldsymbol{A}, \{\boldsymbol{P}_{sa}\}, \boldsymbol{\gamma}, \boldsymbol{R})$ the **states $\boldsymbol{S}$,** the **actions $\boldsymbol{A}$** and the **discount factors $\boldsymbol{\gamma}$** are **almost** always **known.** The **discount factor** is **chosen based** on **how much tradeoff** do you want **between current** and **future rewards**.

We can **easily derive** the **maximum likelihood estimates** for the **state transition probabilities**:

$$\boldsymbol{P}_{sa}(\boldsymbol{s}') = \frac{\#\ \boldsymbol{times\ we\ took\ action\ a\ in\ state\ s\ and\ got\ to\ s'}}{\#\ \boldsymbol{times\ we\ took\ action\ a\ in\ state\ s}}$$

If there is the **case** that you have **never taken action $\boldsymbol{a}$ in state $\boldsymbol{s}$** (0/0) we can **compute** the **uniform distribution over all states** :

$$\boldsymbol{P}_{sa}(\boldsymbol{s}') = \frac{\mathbf{1}}{|\boldsymbol{S}|}$$

REINFORCEMENT LEARNING

LEARNING MDP MODELS

Putting together model learning and **value iteration**, here is **one possible algorithm** for **learning** in an **MDP** with **unknown state transition probabilities**:

1. **Initialize π randomly.**

2. **Repeat** {

   a) Execute π in the **MDP** for some **number** of **trials**

   b) Using the **accumulated experience** in the **MDP**, **update** our **estimates** for P_{sa} (and R, **if applicable**).

   c) Apply value iteration with the **estimated state transition probabilities** and **rewards** to **get** a **new estimated value function V** (**use** this **update** to **initialize V** in **next iteration**).

   (d) Update π to be the **greedy policy** with **respect** to V.

}