Machine Learning/Advanced Machine Learning

Lecture 8.2:
Reinforcement Learning

Sami S. Brandt

Department of Computer Science
IT University of Copenhagen

Based on Slides by D. Silver, UCL and F.F. Li, Stanford, and Ch 18 in Alpaydin, 2014
24 October 2019

IT UNIVERSITY OF COPENHAGEN

Learning Objectives for Thursday

- Reflect the characteristics of reinforcement learning
- Explain the concepts of model, policy and value in relation to RL
- Explain the concepts of environment, agent, state, and action in relation to RL
- Implement simple RL agents based on value iteration or policy iteration
- Explain and apply Bellman equation in Q-learning.

Outline of lecture

Introduction to Reinforcement Learning

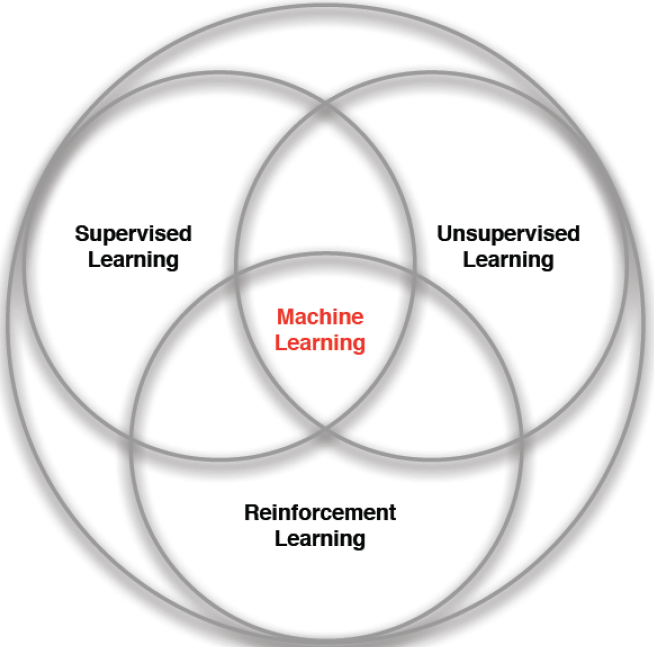
Elements of the Reinforcement Learning

Categorisation of the RL agents

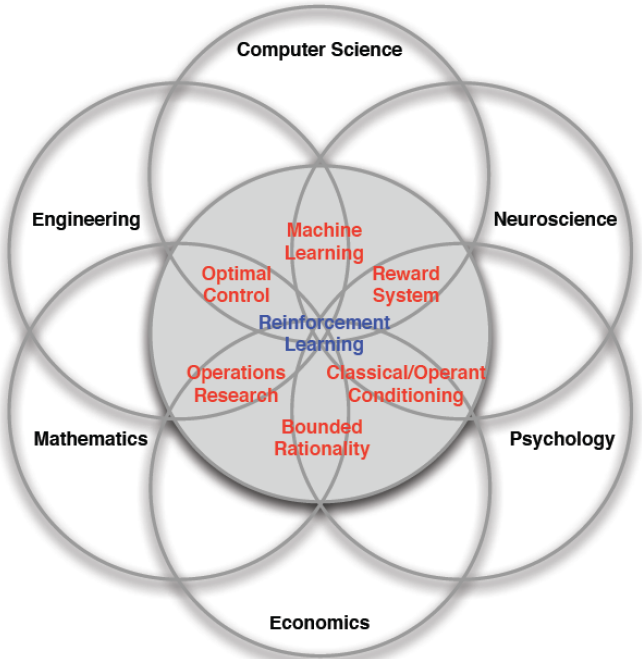
Model-Based Learning

Model Free Learning

RL as part of Machine Learning



Many Faces of Reinforcement Learning



Characteristics of Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

- There is **no supervisor**, only a **reward** signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

Examples of Reinforcement Learning

- Fly stunt manoeuvres in a helicopter
- Defeat the world champion at Backgammon
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

Stanford Autonomous Helicopter

https://www.youtube.com/watch?v=0JL04JJjocc

Google’s DeepMind AI Taught How to Walk

https://www.youtube.com/watch?v=gn4nRCC9TwQ

Google Deep Mind's Deep Q Learning Playing Atari Breakout

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Outline of lecture

Introduction to Reinforcement Learning

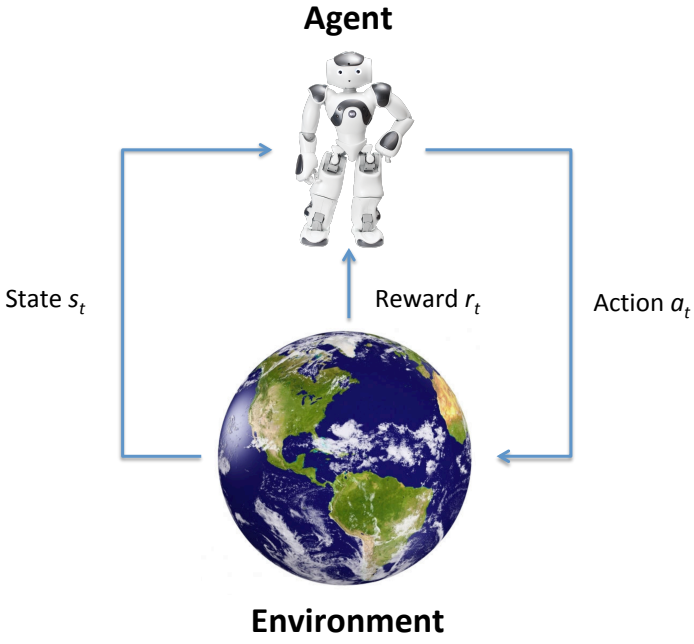
Elements of the Reinforcement Learning

Categorisation of the RL agents

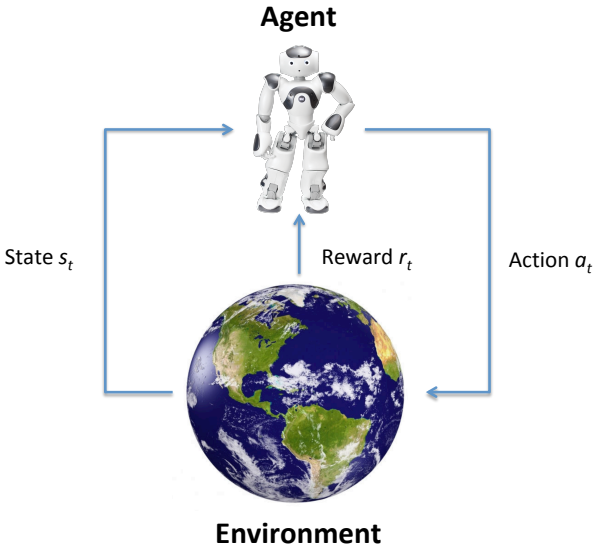
Model-Based Learning

Model Free Learning

Agent and Environment



Agent and Environment



At each time instant t

- There is the **state** s_t of the agent/environment
- Agent makes and **action** a_t
- The agent receives the **reward** r_t

Reward Hypothesis

A **reward** r_t is a scalar feedback signal of the action

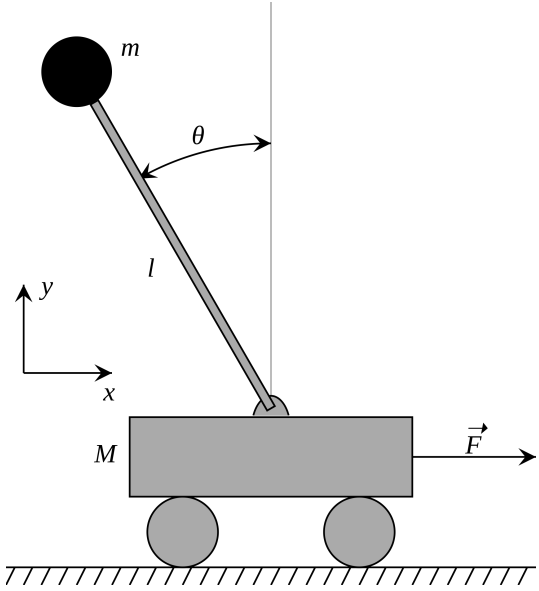
- Indicates how well agent is doing at step t
- The agent's job is to maximise **cumulative reward**

Reinforcement learning is based on the **reward hypothesis**

Definition

Reward Hypothesis is the assumption that all goals can be described by the maximisation of expected cumulative reward

Example: Cart Pole Problem



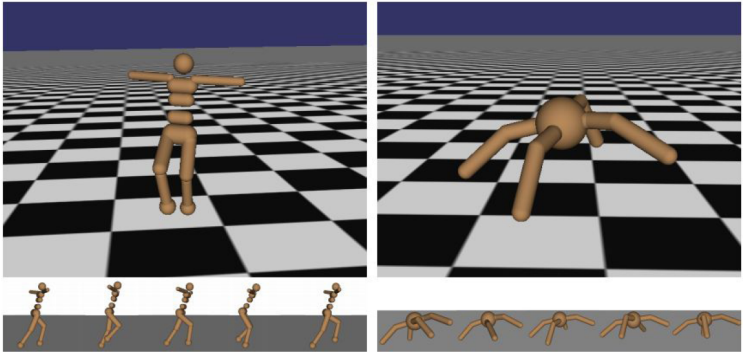
Objective: Balance a pole on top of a movable cart

Action: Horizontal force applied on the cart

State: Angle, angular speed, position, horizontal velocity

Reward: 1 at each time step if the pole is upright

Example: Robot Locomotion



Objective: Make the robot move forward

Action: Torques applied on joints

State: Angle and position of the joints

Reward: 1 at each time step upright + forward movement

Example: Atari Games

Objective: Complete the game with the highest score

Action: Game controls e.g. Left, Right, Up, Down

State: Raw pixel inputs of the game state

Reward: Score increase/decrease at each time step

Sequential Decision Making

Goal: select actions to maximise total **future** reward

- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward

Examples:

- A financial investment (may take months to mature)
- Refuelling a helicopter (might prevent a crash in several hours)
- Blocking opponent moves (might help winning chances many moves from now)

Markov System

Definition

A variable s_t follows a Markov system if and only if

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, \ldots, s_t). \tag{1}$$

"The future is independent of the past given the present"

- Once the state is known, the history may be thrown away
- The state is a sufficient statistic of the future

Mathematical formulation of the RL problem

Markov Decision Process

1. At time step $t = 0$, environment samples the initial state s_0 from $p(s_0)$
2. Repeat until done:
   - Select action a_t. **(Agent)**
   - Sample reward r_t from $P(r_t|s_t, a_t)$. **(Environment)**
   - Sample next state s_{t+1} from $P(s_{t+1}|s_t, a_t)$. **(Environment)**
   - Increment t.

The **model** predicts what the environment will do next

- $P(s_t|s_t, a_t)$ predicts the next state
- $P(r_t|s_t, a_t)$ predicts the next (immediate) reward

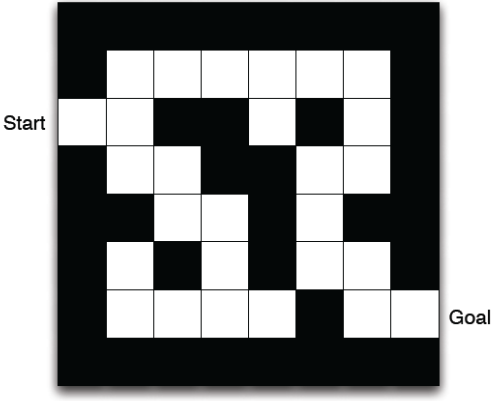
The **policy** $\pi : \mathcal{S} \to \mathcal{A}$, a mapping from the space of states to the space of actions

- It *defines the action* to be taken at state s_t, i.e., $a_t = \pi(s_t)$.

The **value** $V^\pi(s_t)$ of a policy is the *expected cumulative reward* achievable from state s_t.

Objective: find the policy π^* that maximises the value.

Example: Maze

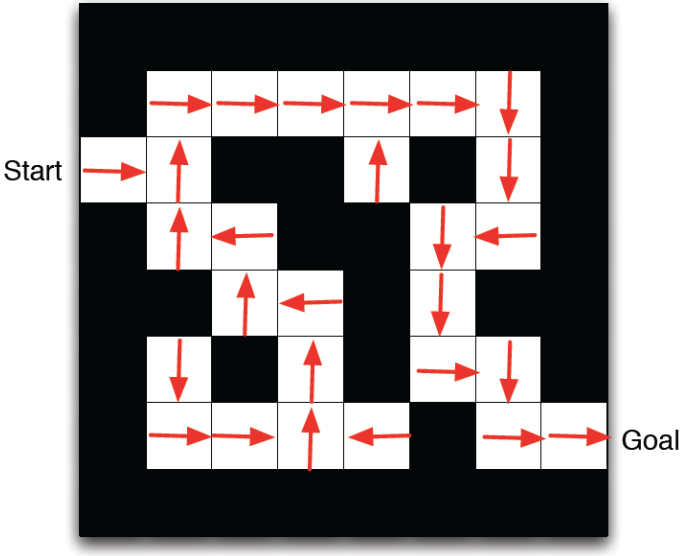


State: Agent's location

Actions: N, E, S, W

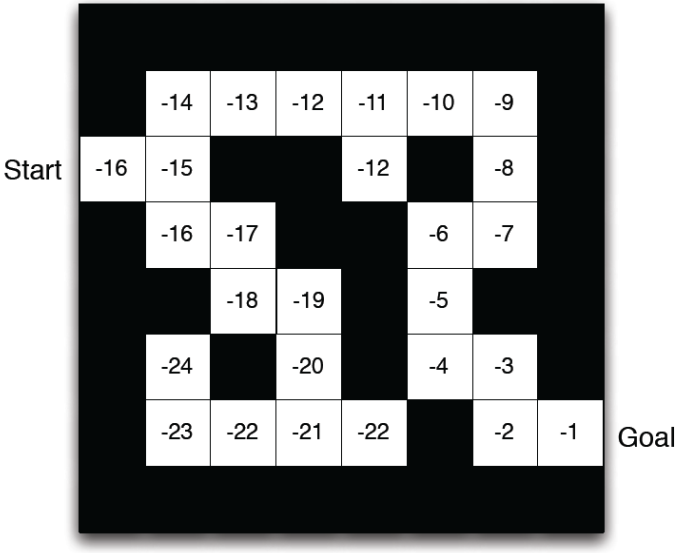
Reward: -1 for each time step

Example: Maze - Policy



Arrows represent the (optimal) policy π for each state s.

Example: Maze - Value Function



Numbers represent the value $V^{\pi}(s_t) = \mathrm{E}[\sum_{i=1}^{T} r_{t+T}]$ for each state s (**finite horizon model**).

Value Function and Q-Value Function

The **value function**, in an **infinite horizon model**, is the expected sum of rewards

$$V^{\pi}(s_t) = \mathrm{E}[\sum_{i=1}^{T} \gamma^{i-1} r_{t+i}]$$

where γ is the **discount rate**.

- Quantifies each state.

The **Q-value function** $Q^*(s_t, a_t)$ is the expected cumulative reward of state–action pair when optimal policy is followed thereafter.

The **optimal policy** π^* that maximises the expected sum of rewards, or,

$$\begin{aligned} V^*(s_t) &= \max_{\pi} V^{\pi}(s_t), \forall s_t \\ &= \max_{a_t} Q^*(s_t, a_t) \end{aligned}$$

Optimal Policy

The expected sum of rewards by the optimal policy hence is

$$
\begin{aligned}
V^*(s_t) &= \max_{a_t} Q^*(s_t, a_t) \\
&= \max_{a_t} \mathrm{E}\left[\sum_{i=1}^{T} \gamma^{i-1} r_{t+i}\right] \\
&= \max_{a_t} \mathrm{E}\left[r_{t+1} + \sum_{i=1}^{T} \gamma^{i+1-1} r_{t+i+1}\right] \\
&= \max_{a_t} \mathrm{E}\left[r_{t+1} + \gamma V^*(s_{t+1})\right]
\end{aligned}
$$

Bellman Equation

$$
V^*(s_t) = \max_{a_t} \left(\mathrm{E}\left[r_{t+1}\right] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \right)
$$

Optimal Policy with the Q-Value Function

From the Bellman equation

$$Q^*(s_t, a_t) = \mathrm{E}\left[r_{t+1}\right] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}).$$

Optimal Policy

$$\pi^*(s_t) : \text{Choose } a_t^* = \arg\max_{a_t} Q^*(s_t, a_t) \tag{2}$$

The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*.

Outline of lecture

Introduction to Reinforcement Learning

Elements of the Reinforcement Learning

Categorisation of the RL agents

Model-Based Learning

Model Free Learning

Categorisation of the RL agents

Value-based

- *No policy (implicit)*
- Value function

Policy-based

- Policy
- *No Value function*

Actor Critic

- Policy
- Value function

Another Categorisation of the RL agents

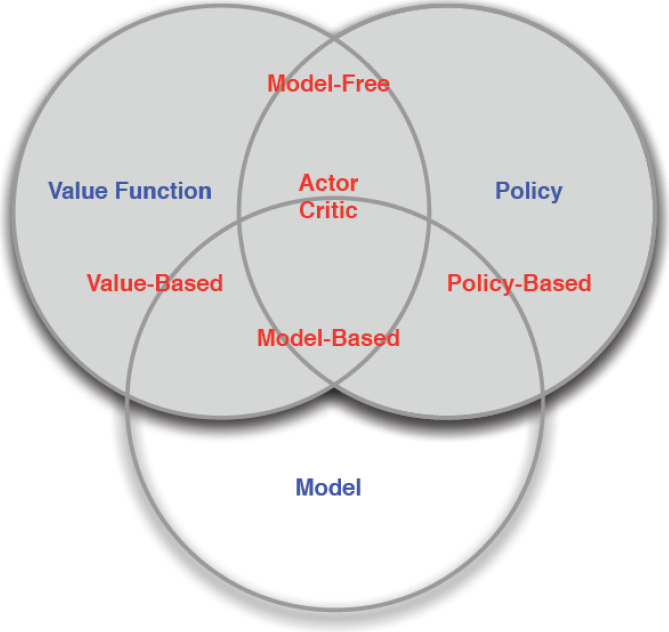
Model-based

- Policy and/or Value Function
- Model

Model free

- Policy and/or Value Function
- *No Model*

RL Agent Taxonomy



Outline of lecture

Introduction to Reinforcement Learning

Elements of the Reinforcement Learning

Categorisation of the RL agents

Model-Based Learning

Model Free Learning

Model-Based Learning: Value Iteration

Assume that we **know** the model parameters $P(s_t|s_t, a_t)$ and $P(r_t|s_t, a_t)$.

The optimal policy can be found by **Value Iteration**

- Shown to converge to the correct values V^*.

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in S$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Figure 18.2 Value iteration algorithm for model-based learning.

Model-Based Learning: Policy Iteration

Assume that we **know** the model parameters $P(s_t|s_t, a_t)$ and $P(r_t|s_t, a_t)$.

In **Policy Iteration** we store and update the policy rather than the values

- Converges faster than value iteration.

Initialize a policy π' arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi}(s'))$
Until $\pi = \pi'$

Figure 18.3 Policy iteration algorithm for model-based learning.

Outline of lecture

Introduction to Reinforcement Learning

Elements of the Reinforcement Learning

Categorisation of the RL agents

Model-Based Learning

Model Free Learning

Model-Free Learning

Model is defined by the **reward** and **the next state probability distributions**

We seldom have such knowledge of the environment

Exploration of the environment is required to query the model

Option: ϵ-greedy search

- With probability ϵ, choose one action randomly;
- With probability $1 - \epsilon$, choose the best action.

Q-learning

We may learn Q by querying the action state pairs.

In principle, we use the Bellman equation

$$Q_{i+1}(s,a) = \mathrm{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right] \tag{3}$$

to compute the update for Q.

Note: in Alpaydin, 2014 Q-learning defined as computing the running average in the non-deterministic case

Deep Q-learning

Updating Q-directly is not **scalable**

- We would need to compute $Q(s, a)$ for every state–action pair!
- E.g. in Atari games the current state is a bitmap image (huge state space)

Solution: Use a **function approximator** for $Q^*(s, a) \approx Q(s, a; \theta)$

If the function approximator is a *deep neural network*, this is referred to as **deep Q-learning**.

Case Example: Atari Games

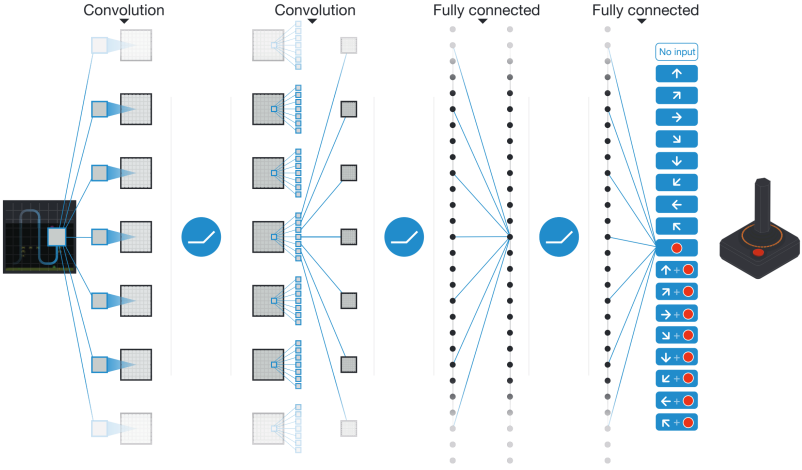
Objective: Complete the game with the highest score

Action: Game controls e.g. Left, Right, Up, Down

State: Raw pixel inputs of the game state

Reward: Score increase/decrease at each time step

Atari Games Case: Q-Network Architecture



Deep Neural Network $Q(s, a; \theta)$. Current state s is the $84 \times 84 \times 4$ stack of last 4 frames. The output is the Q-values $Q(s, a, \theta)$ for all the actions $a \in \{a_1, \ldots, a_2\}$.

From. Mnih et al., Nature 2015

Atari Games Case: Training the Q-network

Loss function

$$L_i(\theta_i) = \mathrm{E}_{s',a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{4}$$

where $y_i = \mathrm{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q_i(s',a';\theta_{i-1})|s,a\right]$.

Gradient update

$$\nabla_{\theta_i} L_i(\theta_i) = \mathrm{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q_i(s',a';\theta_{i-1}) - Q_i(s',a';\theta_i)\right)\right]. \tag{5}$$

Atari Games Case: Experience Replay

Learning from batches of **consecutive** samples is problematic

- Correlated samples (inefficient learning)
- Current Q-network determine next training samples (bad feedback loops)

Solution

- Keep memory of table of transitions (s_t, a_t, r_t, s_{t+1}) as games is played
- Train the Q-network on random minibatches of transitions from the replay memory instead.

Atari Games Case: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Summary

- RL can be seen as the third major branch of machine learning
  - No supervisor, only a reward
  - Sequential learning
- Based on a Markov System
- Different approaches based on
  - Model
  - Policy
  - Value
- Various applications from autonomous vehicles and control systems and investment portfolio management to computer games.

References I


Alpaydin, E. (2014). *Introduction to Machine Learning*. Third Edition. The MIT Press.