CS 461: Machine Learning Principles

Class 26: Dec. 9

Reinforcement Learning

(Model-Free Methods and Deep Reinforcement Learning)

Instructor: Diana Kim

Outline

1. Review of RL Basic Elements

2. Review Model-Based RL (MDP environment, Value and Policy Iteration)

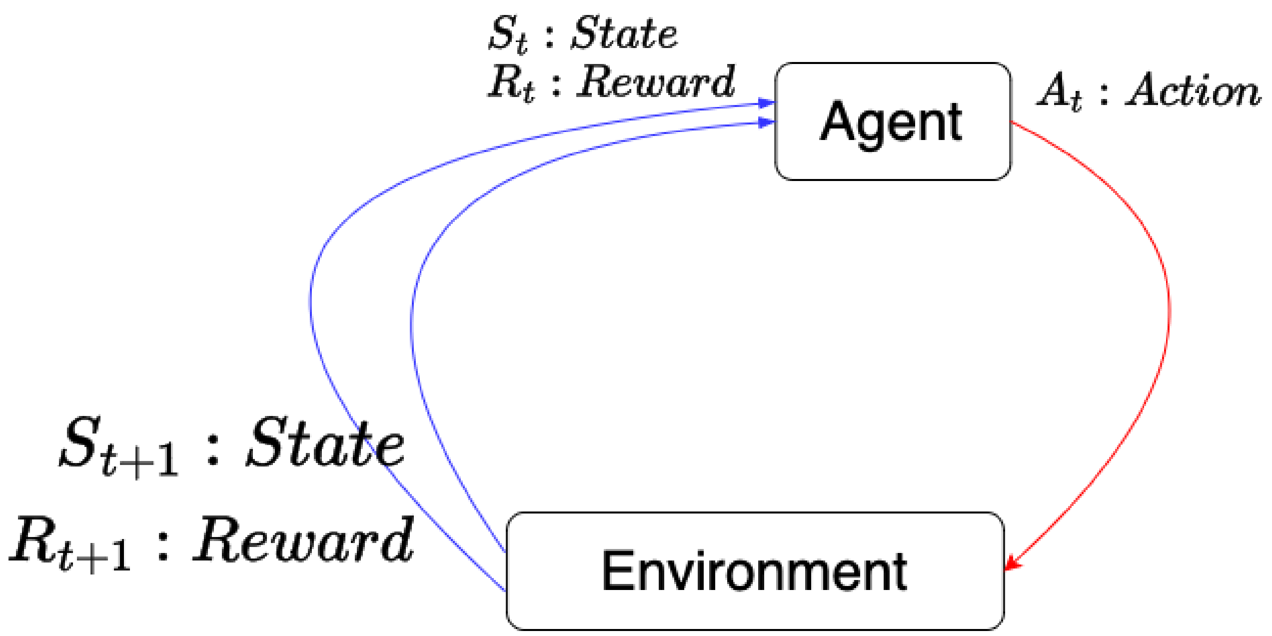
3. Model-Free Methods
    - Monte Carlo Learning
    - Temporal Difference (TD) Learning
    - Q Learning

4. Example of Deep Q Learning with Atari Games.

5. Deep RL: Actor- Critic Networks

Reinforcement Learning (RL) is
To learn control **polices** of agent
to **interact** with a complex environment through **experience**.

Agent & Environment Interactions (at every t)



- **Agent**: The learner and Decision Maker.
- **Environment**: the things the agent interacts with, comprising outside the agent, is called environment
- They together generates a sequence / (trajectory): $S_0, A_0, R_1, S_1, A_1, R_2$ ...

Reference: http://incompleteideas.net/book/the-book-2nd.html

The Elements of RL

- Policy $\pi(s) = \mathrm{P(a|s)} \approx P(a|s,\theta)$
  as |s| is complex, then direct learning is too expensive.
  so approximate the policy as a neural net / a function.

- Reward Signal (s, a)

- Value function: $$\begin{aligned} v_\pi(s) &= E_\pi[G_t|S_t = s] \\ &= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \end{aligned}$$

- A model (optional):
  Markov Decision Process (MDP) defines

One of the central challenge of RL is learning from delayed rewards.
where the optimal solution involves multiple/sequential steps.
One example is chess.

- “Credit Assignment Problem”:

Knowing what action sequence was responsible for reward ultimately received

Bellman's principle optimality (Important properties of the **Value Function)**

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_{t+1} = S', S_t = s]] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_t = S']] \\
&= E_\pi[R_{t+1} + E_\pi[\gamma G_{t+1}|S_{t+1} = S']|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

+ it tells us what is the value function for a certain policy.

$$v(s) = \max_{\pi *} E[R_0 + \gamma v(S')]$$

$$\pi = \arg\max_{\pi} E[R_0 + \gamma v(S')]$$

**Given optimal value function,
It is possible to extract the optimal policy!!**

Bellman equation defines
the linear relations between $\boldsymbol{\pi}$ $(\boldsymbol{s})$and v(s).

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]$$

- Once we have a policy $\boldsymbol{\pi}$ $(\boldsymbol{s})$,
 then we can compute state value functions **v(s)**.
- Once we have a value functions **v(s)** given a policy, we can derive a better policy **$\boldsymbol{\pi}$ (s)**.
 by choosing an action transitioning to **the next state s'** with the larger **v(s')**

A major RL dichotomy

- Model Based RL (MDP)
  (1) value iteration
  (2) policy iteration
  using dynamic programing using Bellman equation.

- Model Free RL
  (3) Q learning

[1] Policy Iteration

$$
\begin{aligned}
v_\pi(s) &= E_\pi[G_t|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_{t+1} = S', S_t = s]] \\
&= E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma G_{t+1}|S_t = S']] \\
&= E_\pi[R_{t+1} + E_\pi[\gamma G_{t+1}|S_{t+1} = S']|S_t = s] \\
&= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'r} P(s', r|s, a)[r + \gamma v_\pi(s')]
\end{aligned}
$$

Repeat two step optimization

(1) given a policy, update state value functions through brute force calculation

(2) given state value functions, update policy

$$\pi = \arg\max_\pi E[R_0 + \gamma v(S')]$$

[2] Value Iteration

Repeat two step optimization

(1) given a policy, update the state value functions.

$$V(s) = \max_a E[R_0 + \gamma v(S')]$$

(2) given state value functions, update the policy

$$\pi = \arg\max_\pi E[R_0 + \gamma v(S')]$$

- Q function: $Q(s,a)$

  the state value function when taking action $\boldsymbol{a}$ $(with\ prob\ 1)$

$$v(s) = \sum_a \pi(a|s) \sum_{s',r} P(r,s'|a,s)(r + \gamma \cdot v(s'))$$

$$Q(s,a) = 1 \cdot \sum_{s',r} P(r,s'|a,s)(r + \gamma \cdot v(s'))$$

Right Q function measures the quality of action and state (like expected reward). Once we learn an optimal Q-function, we can derive an optimal policy.

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

$$\pi * (a|s) = \frac{\exp Q(a, s)}{\sum_{a'} \exp Q(a', s)}$$

Model-Free Reinforcement Learning

(learning Q/V function from experience)

[1] Monte Carlo Learning (Simplest, Q Learning, must be episodic)

(1)Given an arbitrary policy, generate samples $S_0, A_0, R_1, S_1, A_1, R_2 \ldots R_n$

(2) Compute discounted rewards R = $\sum_{n=0}^{N} \gamma^n r_n$ (N: length of sequence)

(3) V(s) ← V(s) + 1/N (R – V(s))

+ R is the estimate for the current policy
+ the error amount needs to be updated!

(4) Q(s,a) ← Q(s,a) + 1/N(R – Q(s,a))

[2] Temporal Difference Learning (no need to be episodic (∞) but similar to Monte Carlo Learning)

V(s) ← V(s) + 1/n (R – V(s))

These are the estimate from the samples,
but now we estimate R = $\sum_{n=0} \gamma^n r_n$ by using n-step a head future rewards.

$$R \approx (r0 + \gamma r1 \ + \gamma^2 r2 \ + \ \ldots \gamma^n r(n) + \gamma^{k+1} v\,(s(n+1)))$$

This is a sample/ realization

- TD [0] Algorithm : One Step Look Ahead

Use one-step ahead future reward is used to update the current value function.

(1) At a current policy, we just generates action and next state s;

(2) Then we can update R $\approx$ r0 + γV(s'), update the policy.

$$V(s) \leftarrow V(s) + \alpha\ (r0 + \gamma V(s') - V(s))$$

SARSA(State-Action-Reward-State-Action Learning)
Is a popular TD algorithm

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r_k + \gamma Q(s', a) - Q(s, a))$$

Q-Learning (off-policy TD algorithm)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_k + \boxed{\gamma \max_a Q(s',a)} - Q(s,a))$$

+ this action is not the real sample action.
The policy used for update value function is different from the policy the current Q function represents.

On-Policy (SARSA) vs Off-Policy (Q-Learning)

(1) on-policy : value function is aligned well the policy we found (**Exploitation**)
(2) off –policy : the value function is not aligned well with the current policy.
(more flexible **Exploration**)

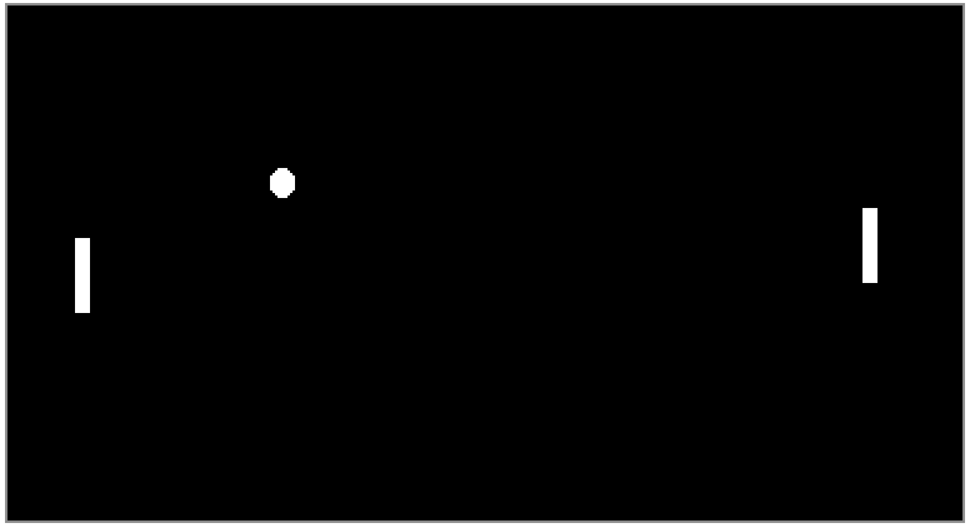
- Q Learning General Steps (Value Based)

(1) Set a Policy
(2) Perform an action
(3) Update Value / Q function
(4) Update Policy, Goes to (1)

Policy Network (Deep Q Learning)
: often state space is too complex, so we implement the
$\pi(s)$ is implemented by a function.

Can we represent the policy function for the game “**pong from pixel**” with a neural net? What will be the input and output?

http://karpathy.github.io/2016/05/31/rl/



Policy Network and Q Learning Example

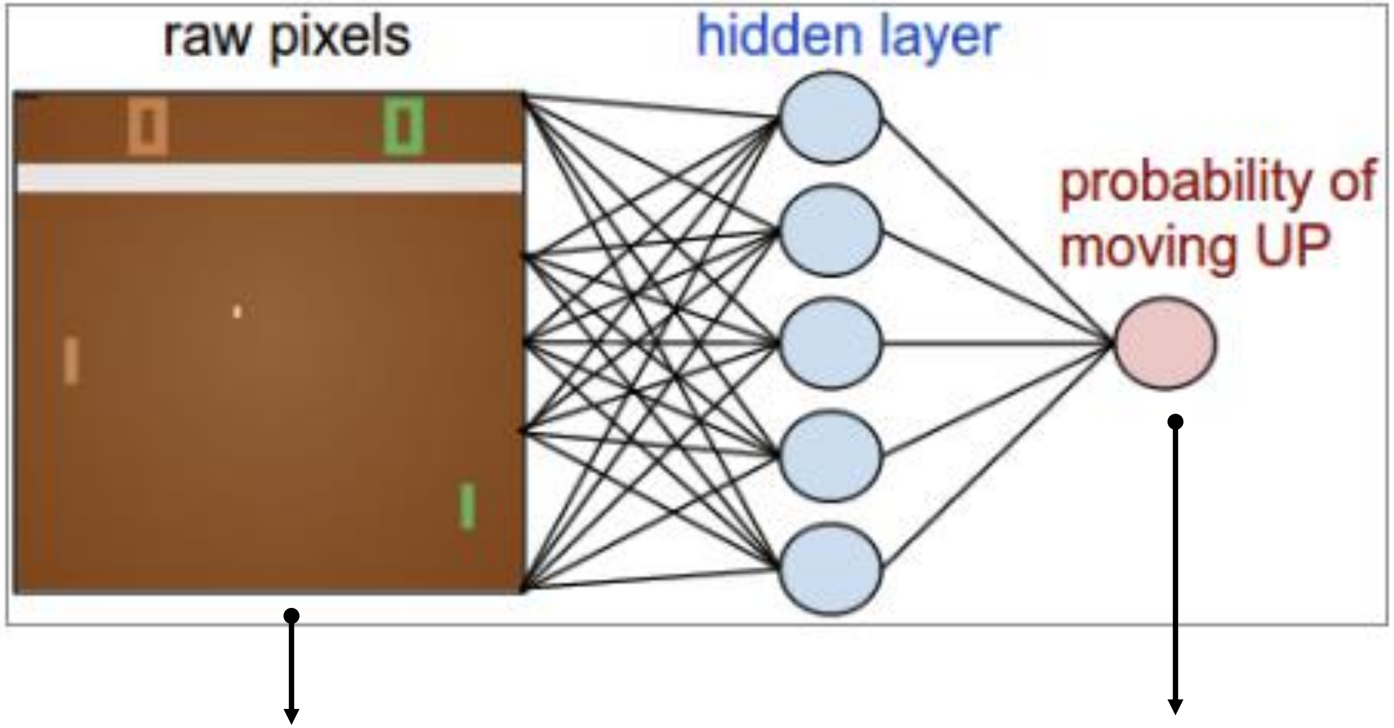
Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

Neural Network Representation of the **Policy** $\boldsymbol{\pi(a|s)}$



Input: steel shot (state)

Output： $\pi(a|s)$

How can we define an objective (Loss) for the policy network ?

The ultimate goal of Reinforcement Learning is to learn a policy $\pi^*(a|s)$ that maximizes the excepted rewards (Expected Cumulative Reward)

$$E[R] = \sum_{a,s} P(s)P(a|s)R(s,a)$$

$$\begin{aligned}
\nabla_\theta E[R] &= \sum_{a,s} P(s)\nabla_\theta P_\theta(a|s)R(s,a) \\
&= \sum_{a,s} P(s)\frac{\nabla_\theta P_\theta(a|s)}{P_\theta(a|s)}P_\theta(a|s)R(s,a) \\
&= \sum_{a,s} P(s)\nabla_\theta \log P_\theta(a|s)P_\theta(a|s)R(s,a) \\
&= \boxed{E[R(s,a)\cdot\nabla_\theta \log P_\theta(a|s)]} \\
&\approx 1/N\sum_{n=1}^{N} R(s,a_n)\cdot\nabla_\theta \log P_\theta(a_n|s)]
\end{aligned}$$

[Policy Gradient Methods]

How could we compute this?

By Monte Carlo Rollout

There are several ways to compute the policy gradient

$E[R(s,a) \cdot \nabla_\theta \log P_\theta(a|s)]$ likelihood × reward for each step

[1] Immediate rewards case (independent trial and error)

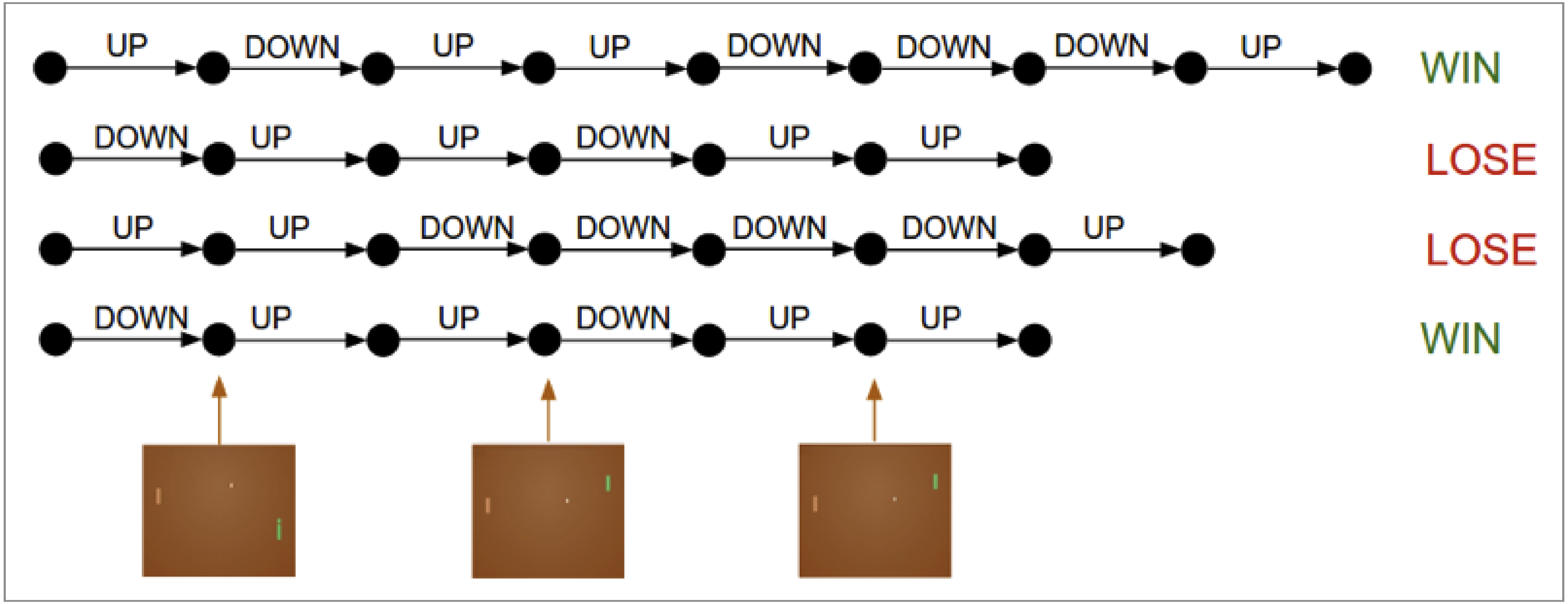
$$\approx 1/N \sum_{n=1}^{N} R(s,a_n) \cdot \nabla_\theta \log P_\theta(a_n|s)]$$ (N trials)

Learning Motor Primitives for Robotics

There are several ways to compute

$E[R(s,a) \cdot \nabla_\theta \log P_\theta(a|s)]$ likelihood × reward for each step

[2] Delayed rewards case (sequential and dependent actions, the rewards are sparse.)

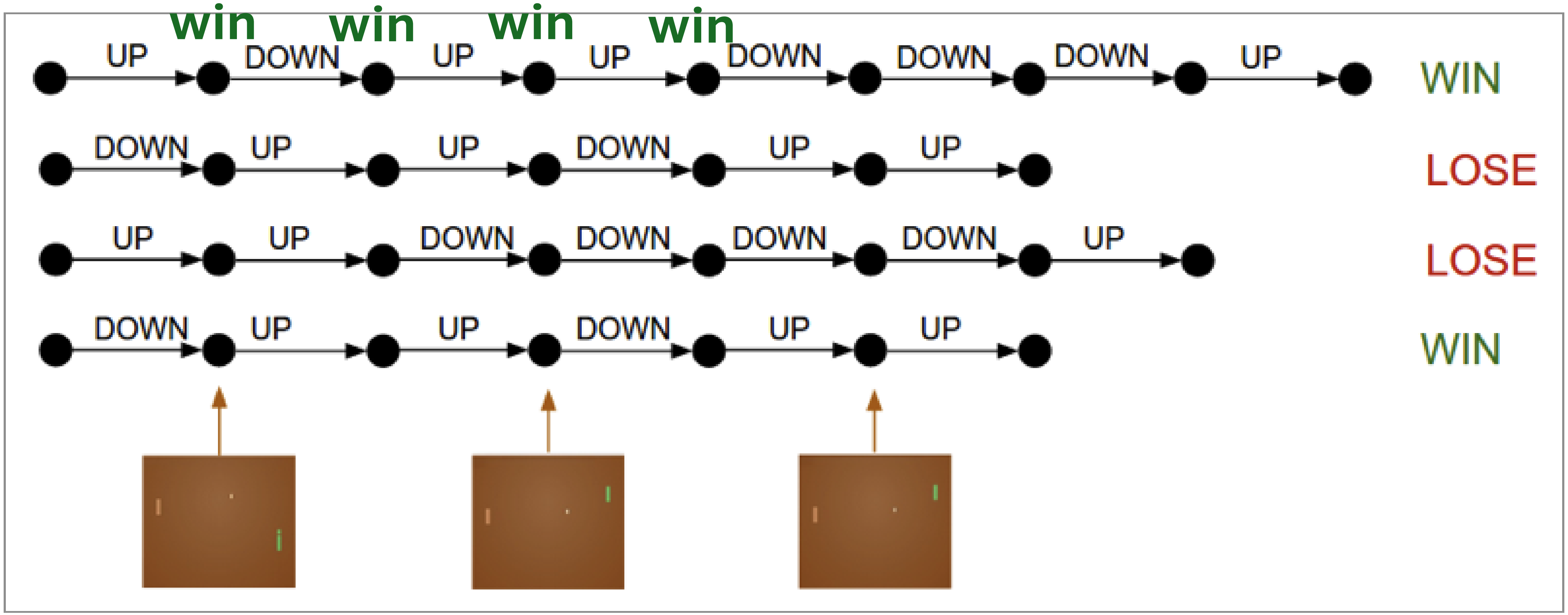


There are several ways to represent

$E[R(s,a) \cdot \nabla_\theta \log P_\theta(a|s)]$ likelihood × reward for each step

[2] Delayed rewards case (rewards are sparse)

- copy the final reward for every step and train like supervised learning

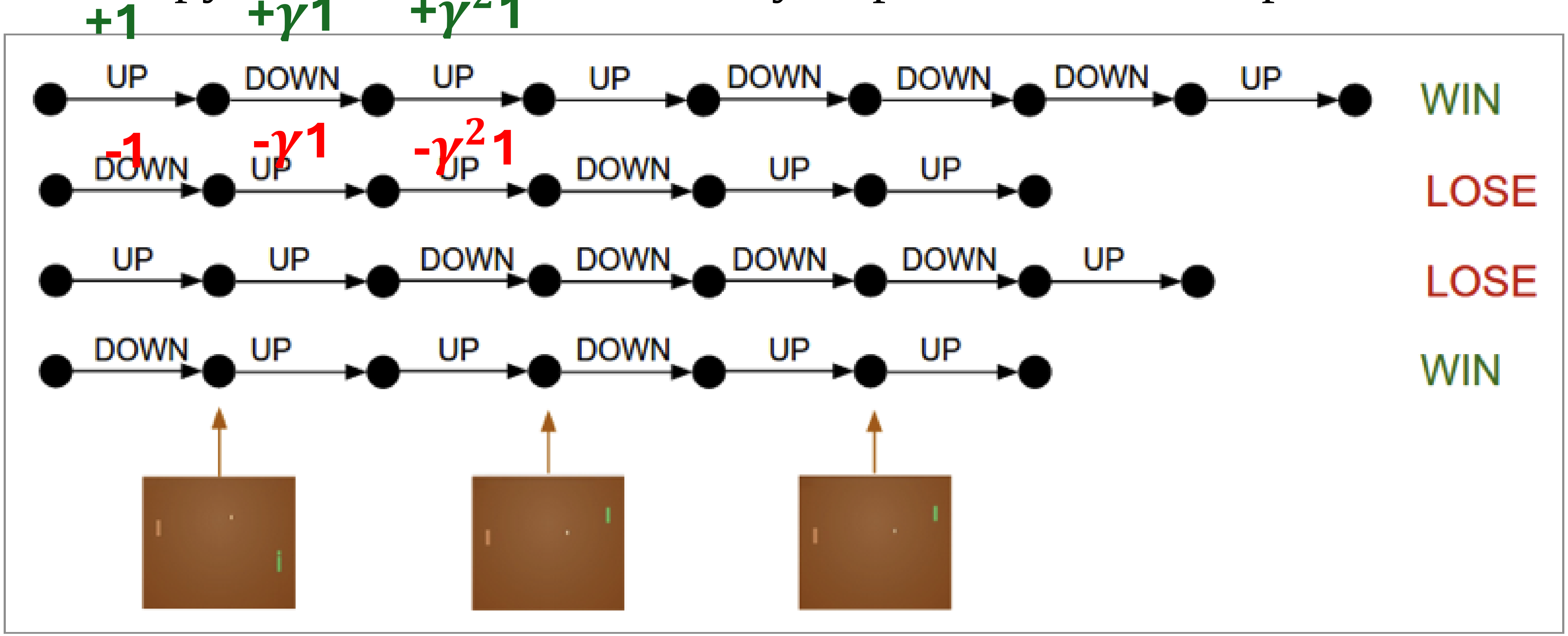


There are several ways to compute

$E[R(s,a) \cdot \nabla_\theta \log P_\theta(a|s)]$ likelihood × reward for each step

[2] Delayed rewards case (rewards are sparse)

- copy the final reward for every step and train like supervised learning



When State Value Function is available (through estimation/ learning),
We can update the policy network based on **Temporal Difference Function**.

- **Temporal Difference Function** $\delta(t)$

$$\delta(t) = \boxed{R(t+1) \ + \ \gamma \, v(s'(t+1))} \ - \ \boxed{v(s(t))}$$

- rewards

as using the current policy

- average rewards

- $\delta(t) \geq 0$: current action's reward is better average rewards
- $\delta(t) < 0$: current action's rewards is worse average rewards
- Hence , if we update the policy $network \ parameters$

$$\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} + \alpha \boldsymbol{\delta(t) \nabla \pi} \ (\boldsymbol{a_t} | \boldsymbol{s})$$

It updates parameters to maximize the rewards.

Convolution Structure of deep Q network used to play Atari Game

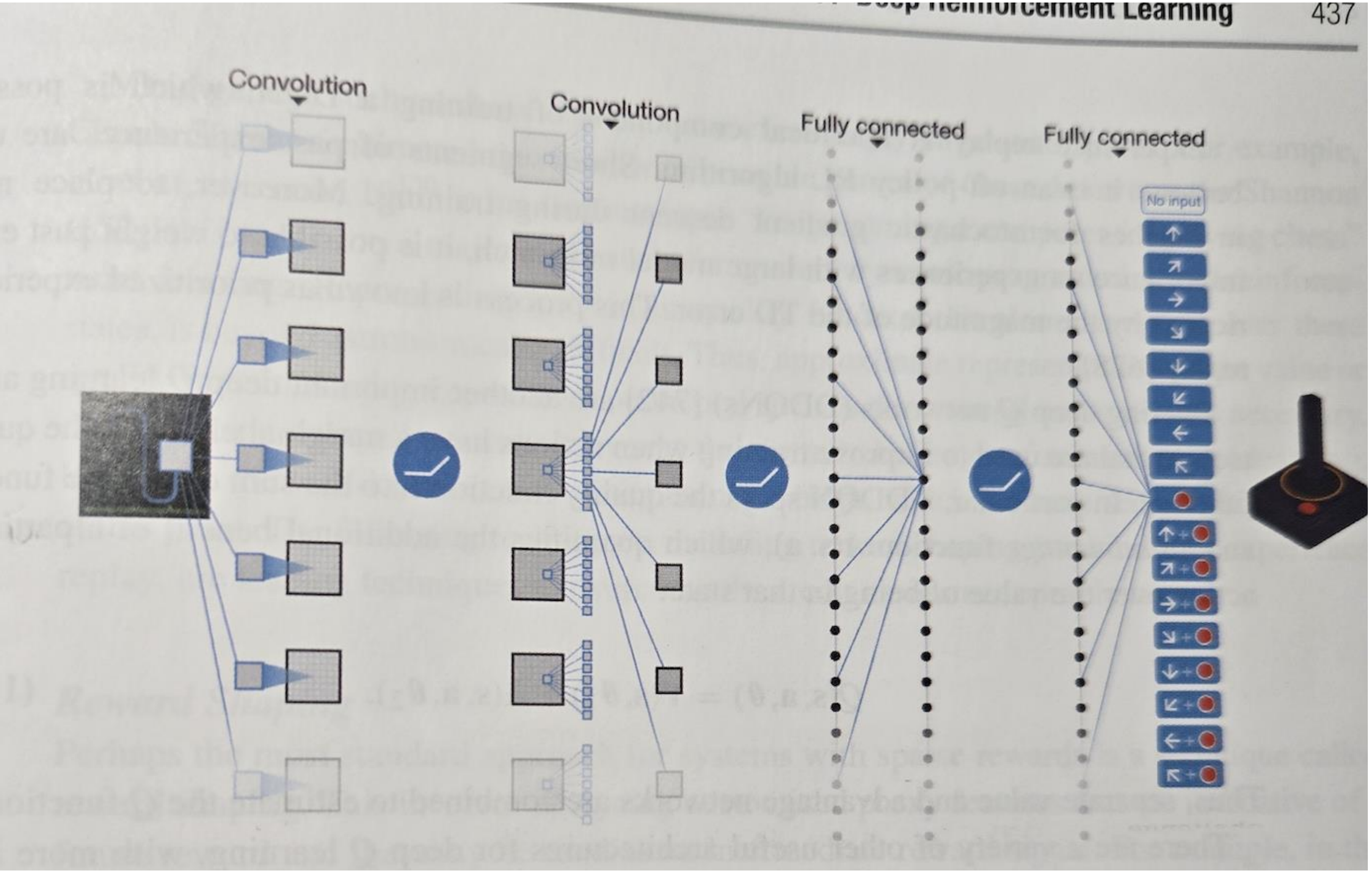


Fig 11.5: https://databookuw.com

Deep Q Learning Loss

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\underbrace{\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)}_{\text{Q function (error estimate)}} \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

Just need to know next state is

- success +1) : I just sent the ball to opponent
- fail (-1) :missed the ball
- o.w (0) : just opponent hit the ball.

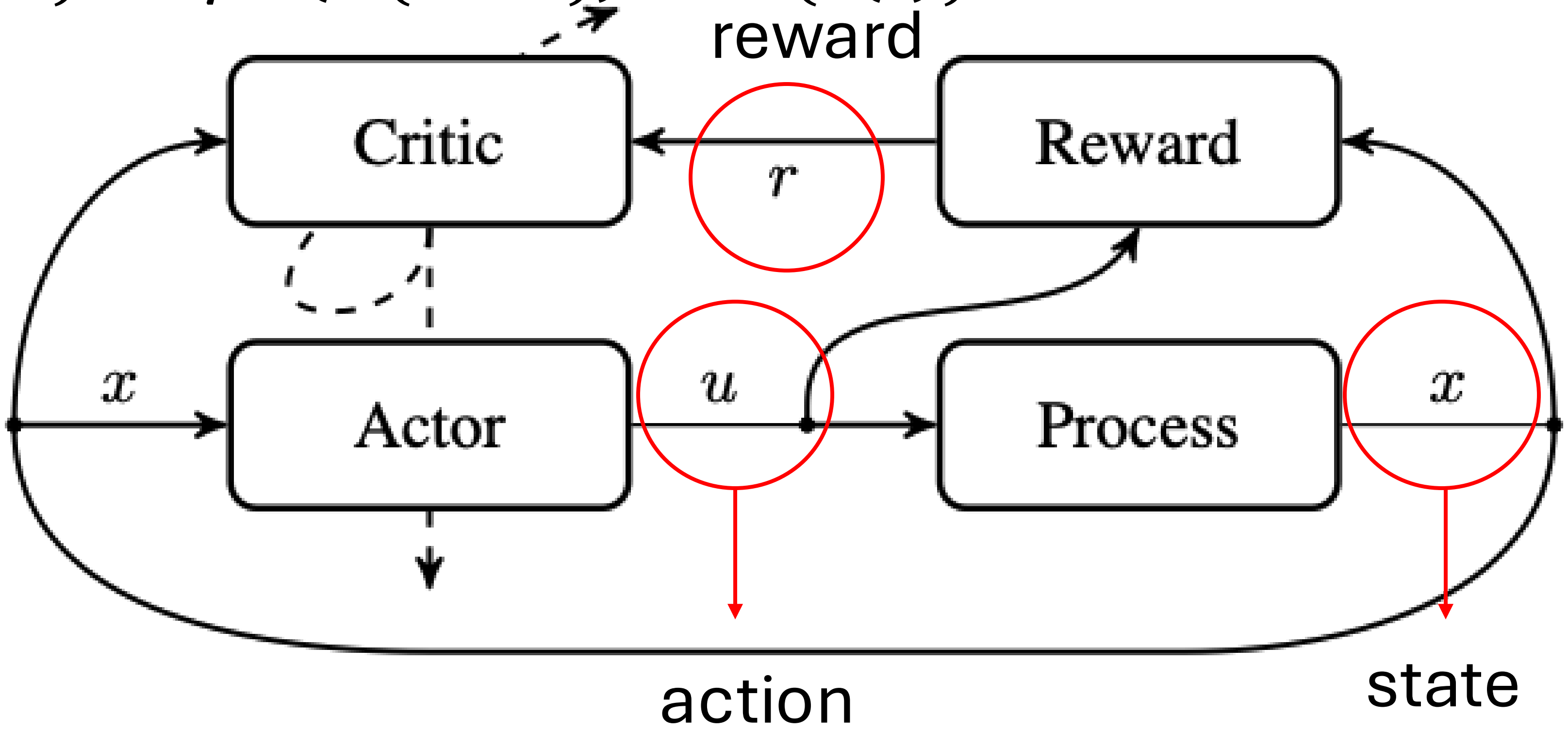
Actor and Critic Methods
(Two networks Learning Policy and Value Function)

Schematic Overview of an Actor Critic Algorithm

(paper: https://busoniu.net/files/papers/ivo_smcc12_survey.pdf)

by the temporal difference, both of the critic and actor are updated at every t

$$\delta(t) = R(t+1) \;+\; \gamma\, v(s'(t+1)) \;-\; v(s(t))$$



- Critic : Value Network $\theta_{t+1} \leftarrow \theta_t + \;\alpha\delta(t)\nabla\pi\,(a_t|s)$ (updated first)
- Actor: Policy Network $v_{t+1} \leftarrow v_t + \beta\delta(t)$