Contents

Chapter 1

Reninforcement Learning(RL)

1.1 Markov Decision Process

1.1.1 Basic Formulation and terminology

A **Markov Decision Process (MDP)** has the following four key componnents: states space $\mathbb{S}$, action space $\mathbb{A}$, transtion model $\mathbf{T}$, reward model $\mathbf{r}$. In some literature the discount factor γ may be treated as the fifth components of a MDP. The reward is commonly chosen to be one scalar for simplicity, so a MDP can be written as a 5-tuple $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$.

The relative definitions are summaried in the following subsections.A basic MDP means the state space and actions spaces are finite, time steps are discrete. Classical Dynamic Programming algorithms can be used to solve this case.

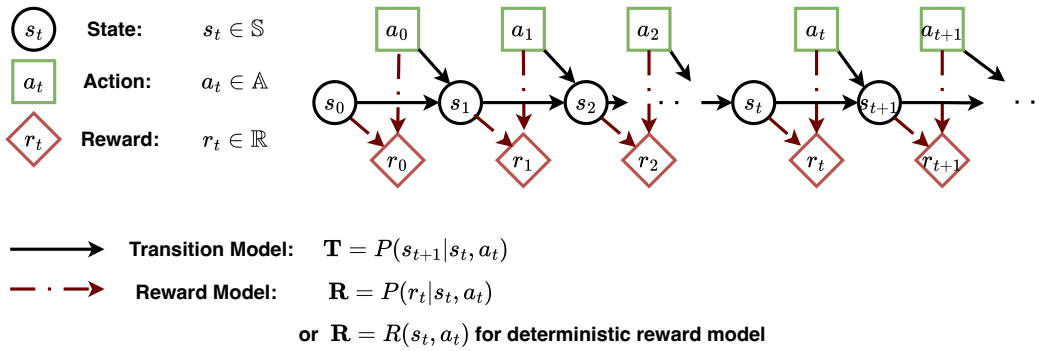

Figure 1.1: Sequential Probabilistic Graphical Model of Markovian Decision Process

State space $\mathbb{S}$

a finite set of states $\mathbb{S}$. It is very important to choose the state variables to represent the key information of the environment. In real life there are multiple state variables representing different properties of the environment. We may use the $d_{\mathbb{S}}$ to represent the number of features we choose as state variables. e.g. The healthy state of a person contain mental health and physical health two aspects, $d_{\mathbb{S}} = 2$. For each kind of state variable, its value could be discrete or continous.

- Discrete State Variable, has the number of N possible values, e.g. the mental health level could be {"Happy","Depressive","sad","angry"}, then $N = 4$
- Continuous State Variable

Action space $\mathbb{A}$

a finite set of actions $\mathbb{A}$; $a_{i \geqslant t} \in \mathbb{A}$

Policy $\pi(a|s)$

A policy $\pi(a|s)$: the state-dependent sequence of actions

- Deterministic policy $\pi(a|s) : \mathbb{S} \to \mathbb{A}$
- or Stochastic policy $\pi(a|s) : \mathbb{S} \times \mathbb{A} \to \mathbb{R} \in [0, 1]$

In the reinforcement learning, we need to let the agent and the environment interact with each other, and record all observed states, actons, rewards trajectories to learn a best policy from these experiences. Here we need to distinguish two kinds of policies: Behavior policy μ: the policy we use to collect experience; Target policy π: the finally trained policy

- If the Behavior policy μ and the target policy π are the same, which means $a_{i \geqslant t} \sim \pi$,then it is called the On-policy.
- If the Behavior policy μ and the target policy π are different, which means $a_{i>t} \sim \pi$, $a_t \sim \mu$, then it is called the Off-policy. The advantage of Off-policy is that we could use the behavior policy to collect experience , save (s_t, a_t, r_t, s_{t+1}) to an array and use such array (aka. replay buffer) repeatedly to update the target policy. This training method is called the experience replay.

  The commonly-chosen behavior policy is $\epsilon-$greedy policy:

$$a_t = \begin{cases} \arg\max_a Q^\star(s_t, a; \theta) & \text{with the prob. } 1 - \epsilon, \\ \text{randomly choose one action from } \mathbb{A} & \text{with the prob. } \epsilon. \end{cases} \tag{1.1}$$

The random behavior policy could explore more unknown states and it is good to enlarge the ϵ at the beginning (e.g. $\epsilon = 0.5$) and reduce it during the trainning process (e.g. $\epsilon = 0.01$)

State Transition Model T

- It could be a deterministic transition function: e.g. $s_{t+1} = f(s_t, a_t)$;
- or a transition probability: $P(s_{t+1}|s_t, a_t, \cdots, s_1, a_1)$ ($P(s_{t+1}|s_t, a_t)$ under the Markov Assumption), naturallly we have

$$\sum_{s_{t+1} \in \mathbb{S}} P(s_{t+1}|s_t, a_t) = 1$$

Reward model R (negative cost function)

If we consider determnistic reward model, $\mathbf{R} = P(r_{t+1}|a_t, s_t)$ can be written as $\mathbf{R} = R(a_t, \mathbf{b}_t)$ A reward model(negative cost function could be formulated as:

- A general reward function $R(s_t, a_t, s_{t+1})$
- Or an immediate reward function $R(s_t, a_t)$

The cumulative weighted total return $U^\pi(s_t, a_t, \cdots, s_T, a_T)$

The cumulative weighted total return is a function of all the states and actions from time t when taking a policy π:

$$U_t^\pi = U^\pi(s_t, a_t, \cdots, s_T, a_T) = k_t R(s_t, a_t, s_{t+1}) + \cdots + k_T R(s_T, a_T, s_{T+1}) = \sum_{i=t}^{T} k_i R(s_i, a_i, s_{i+t}) = \sum_{i=t}^{T}$$

(1.2)

where the discount factor $\gamma \in [0, 1]$: weighting the relative importance of the current reward against the future reward. In extreme cases when $\gamma = 0$, means only the current reward matters.

The Expected cumulative weighted total return J^π

The Expected cumulative weighted total return is a meassure of the policy π:

$$J^\pi = \mathbb{E}_{S_{i \geqslant t} \sim \boldsymbol{T}, A_{i \geqslant t} \sim \pi} [U^\pi(s_t, a_t, S_{t+1}, A_{t+1}, \cdots, S_T, A_T)] \tag{1.3}$$

where $\boldsymbol{T} = P(s_t|s_{t-1}, a_{t-1})$ is the transition probability distribution of state s_t and π is the probability of action (aka. the stochastic policy).

The State-Action Value function $Q^{\pi}(s_t, a_t)$

The State-Action value function $Q^{\pi}(s_t, a_t)$ describes the value of the policy π given a State-Action Pair $Q^{\pi}(s_t, a_t)$. It is a meassure of the State-Action pair at time step t and the policy π from time step $t+1$. That is why we need to reduce all the randomness from all the future state and future action.

The value of taking an action a_t at the state s_t and using the strategy π for the rest of time span until T is calculated by the expectation of all the state and actions from time t to T. s

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi}\left[U^{\pi}(s_t, a_t, S_{t+1}, A_{t+1}, \cdots, S_T, A_T)|s_t, a_t\right] \tag{1.4}$$

where $\boldsymbol{T} = P(s_t|s_{t-1}, a_{t-1})$ is the transition probability distribution of state s_t and π is the probability of action (aka. the stochastic policy).

The State Value Function $V^{\pi}(s_t)$

The State Value Function $V^{\pi}(s_t)$ is a measure of the value of current state s_t and that is why we need to further eliminate the randomness of action at time A_t.

The value of a state s_t when taking a policy from this time step t until the rest of the time span can be expressed by the expected total return, mathematically written as:

$$\begin{aligned}
V^{\pi}(s_t) &= \mathbb{E}_{A_t\sim\mu}\left[Q^{\pi}(s_t, A_t)\right] && \text{(1.5a)}\\
&= \mathbb{E}_{S_{i>t}\sim\boldsymbol{T}, A_{i>t}\sim\pi, A_t\sim\mu}\left[U^{\pi}(s_t, A_t, \cdots, S_T, A_T)|s_t, a_t\right] && \text{(1.5b)}\\
&= \mathbb{E}_{S_{i>t}\sim\boldsymbol{T}, A_{i>t}\sim\pi}\left[U^{\pi}(s_t, a_t, \cdots, S_T, A_T)\right] && \text{(1.5c)}\\
&= \mathbb{E}_{S_{i>t}\sim\boldsymbol{T}, A_{i>t}\sim\pi}\left[\sum_{i=t}^{T}\gamma^{i-t}R(s_i, a_i, s_{i+1})\middle|s_t\right], && \text{(1.5d)}
\end{aligned}$$

where μ shows the probability distribution of a_t at time t specifically, which is not necessarily equal to the policy π chosen to taken actions from time $t+1$ on.

The Optimal State Action Value Function $Q^{\star}(s_t, a_t)$

The Optimal State Action Value function $Q^{\star}(s_t, a_t)$ is a meassure of values of the current state-action pair s_t. Based on the State Action Value function $Q^{\pi}(s_t.a_t)$ we need to further eliminate the randomness of the policy π:

$$Q^{\star}(s_t, a_t) = \max_{\pi}\mathbb{E}_{S_{i>t}\sim\boldsymbol{T}}\left[\sum_{i=t}^{T}\gamma^{i-t}R(s_i, a_i, s_{i+1})\middle|s_t, a_t\right] \tag{1.6}$$

The Optimal State Value Function $V^\star(s_t)$

The Optimal State Value function $V^\star(s_t)$ is a meassure of values of the current state s_t only. Based on the State Value function $V^\pi(s_t)$ we need to further eliminate the randomness of the policy π:

$$V^\star(s_t) = \max_\pi \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}} \left[\sum_{i=t}^{T} \gamma^{i-t} R(s_i, a_i, s_{i+1}) \Big| s_t \right] \tag{1.7}$$

Bellman Equation for State-Action Value Function $Q^\pi(s_t, a_t)$

Now we will derive the Bellman equation for the state-action value function

$$Q^\pi(s_t, a_t) = \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} [U^\pi(s_t, a_t, \cdots, S_T, A_T) | s_t, a_t] \tag{1.8a}$$

$$= \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[\sum_{i=t}^{T} \gamma^{i-t} R(s_i, a_i, s_{i+1}) \Big| s_t, a_t \right] \tag{1.8b}$$

$$= \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[R(s_t, a_t, S_{t+1}) + \gamma \sum_{i=t+1}^{T} \gamma^{i-(t+1)} R(s_i, a_i, s_{i+1}) \Big| s_t, a_t \right] \tag{1.8c}$$

$$= \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[R(s_t, a_t, S_{t+1}) + \gamma U^\pi(s_{t+1}, a_{t+1}, \cdots, S_T, A_T) \big| s_t, a_t \right] \tag{1.8d}$$

$$= \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} [R(s_t, a_t, S_{t+1})] + \gamma \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[U^\pi(S_{t+1}, A_{t+1}, \cdots, S_T, A_T) \big| s_t, a_t \right] \tag{1.8e}$$

With the assumption of discrete states and discrete actions, the first expectation can be written as:

$$\mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[R(s_t, a_t, S_{t+1}) \big| s_t, a_t \right] = \sum_{s_{t+1}\in\mathbb{S}} R(s_t, a_t, s_{t+1}) P(s_{t+1}|s_t, a_t) \tag{1.9}$$

The second expecation

$$\mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[U^\pi_{t+1} | s_t, a_t \right] = \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t, a_t) \sum_{a_{t+1}\in\mathbb{A}} \pi(a_{t+1}|s_{t+1}) \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}, A_{i>t}\sim\pi} \left[U^\pi_{t+1} \big| s_{t+1}, a_{t+1} \right. \tag{1.10a}$$

$$= \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t, a_t) \sum_{a_{t+1}\in\mathbb{A}} \pi(a_{t+1}|s_{t+1}) \mathbb{E}_{S_{i>t+1}\sim \rho, A_{i>t+1}\sim\pi} \left[U^\pi_{t+1} \big| s_{t+1}, a \right. \tag{1.10b}$$

Combining the two equations we have

$$Q^{\pi}(s_t, a_t) = \sum_{s_{t+1} \in \mathbb{S}} P(s_{t+1}|s_t, a_t) \left[R(s_t, a_t, s_{t+1}) + \gamma \sum_{a_{t+1} \in \mathbb{A}} \pi(a_{t+1}|s_{t+1}) \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi} \left[U^{\pi}_{t+1} \middle| s_{t+1}, a \right. \right. \tag{1.11a}$$

$$= \sum_{s_{t+1} \in \mathbb{S}} P(s_{t+1}|s_t, a_t) \left[R(s_t, a_t, s_{t+1}) + \gamma \sum_{a_{t+1} \in \mathbb{A}} \pi(a_{t+1}|s_{t+1}) Q^{\pi}(s_{t+1}, a_{t+1}) \right] \tag{1.11b}$$

If we can simplify the reward function as $R(s_t, a_t, s_{t+1}) = R(s_t, a_t)$, then the above recursive equation can be written as

$$Q^{\pi}(s_t, a_t) = R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathbb{S}} \sum_{a_{t+1} \in \mathbb{A}} P(s_{t+1}|s_t, a_t) \pi(a_{t+1}|s_{t+1}) Q^{\pi}(s_{t+1}, a_{t+1}) \tag{1.12}$$

Written in expectation form as

$$Q^{\pi}(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{S_{t+1} \sim \boldsymbol{T}, A_{t+1} \sim \pi}[Q^{\pi}(S_{t+1}, A_{t+1})] \tag{1.13}$$

Bellman Equation for State Value Function $V^{\pi}(s_t)$

Now we will derive the Bellman equation for the state value function

$$V^{\pi}(s_t) = \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} [U^{\pi}(s_t, A_t, \cdots, S_T, A_T)|s_t] \tag{1.14a}$$

$$= \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} \left[\sum_{i=t}^{T} \gamma^{i-t} R(s_i, a_i, s_{i+1}) \middle| s_t \right] \tag{1.14b}$$

$$= \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} \left[R(s_t, A_t, S_{t+1}) + \gamma \sum_{i=t+1}^{T} \gamma^{i-(t+1)} R(s_i, a_i, s_{i+1}) \middle| s_t \right] \tag{1.14c}$$

$$= \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} \left[R(s_t, A_t, S_{t+1}) + \gamma U^{\pi}(s_{t+1}, a_{t+1}, \cdots, s_T, a_T) \middle| s_t \right] \tag{1.14d}$$

$$= \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} \left[R(s_t, A_t, S_{t+1}) \middle| s_t \right] + \gamma \mathbb{E}_{S_{i>t} \sim \boldsymbol{T}, A_{i>t} \sim \pi, A_t \sim \mu} \left[U^{\pi}(s_{t+1}, a_{t+1}, \cdots, s_T, a_T) \right. \tag{1.14e}$$

Now we can look at the two expectations separately. Assume that the state and actions are discrete, then the first expectation can be written as:

$$\mathbb{E}_{S_{i>t}\sim \boldsymbol{T},A_{i>t}\sim\pi,A_t\sim\mu}\left[R(s_t,A_t,S_{t+1})|s_t\right] = \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t)\mathbb{E}_{S_{i>t}\sim \boldsymbol{T},A_{i>t}\sim\pi}[R(s_t,a_t,S_{t+1})|s_t,a_t] \tag{1.15a}$$

$$= \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \sum_{s_{t+1}\in\mathbb{S}} R(s_t,a_t,s_{t+1})p(s_{t+1}|s_t,a_t) \tag{1.15b}$$

The second expectation $\gamma\mathbb{E}_{S_{i>t}\sim \boldsymbol{T},A_{i>t}\sim\pi,A_t\sim\mu}\left[U^\pi(S_{t+1},A_{t+1},\cdots,S_T,A_T)\middle|s_t\right]$:

$$\mathbb{E}_{S_{i>t}\sim \boldsymbol{T},A_{i>t}\sim\pi,A_t\sim\mu}\left[U^\pi_{t+1}\middle|s_t\right] = \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t,a_t)\mathbb{E}_{s_{i>t+1}\sim\rho,a_{i>t+1}\sim\pi,a_{t+1}\sim\pi}\left[U^\pi_{t+1}\middle|s_{t+1}\right] \tag{1.16a}$$

$$= \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t,a_t)V^\pi(s_{t+1}) \tag{1.16b}$$

Combining these two expectations, we have

$$V^\pi(s_t) = \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t,a_t)\left(R(s_t,a_t,s_{t+1}) + \gamma V^\pi(s_{t+1})\right) \tag{1.17}$$

With the simplification of $R(s_t,a_t,s_{t+1}) = R(s_t,a_t)$, then the above recursive equation can be written as s

$$V^\pi(s_t) = \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t,a_t)\left(R(s_t,a_t) + \gamma V^\pi(s_{t+1})\right) \tag{1.18a}$$

$$= \sum_{a_t\in\mathbb{A}} \mu(a_t|s_t) \left(R(s_t,a_t) + \gamma \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t,a_t)V^\pi(s_{t+1})\right) \tag{1.18b}$$

$$= \mathbb{E}_{A_{i=t}\sim\mu}\left[R(s_t,A_t)\right] + \gamma\mathbb{E}_{A_t\sim\mu,S_{t+1}\sim\boldsymbol{T}}\left[V^\pi(S_{t+1})\right] \tag{1.18c}$$

Optimal Bellman Equation for State Action Value Function $Q^\star(s_t,a_t)$

Bellman Equation for Optimal State Value Function $V^\star(s_t)$ is also known as Optimal Bellman Equation.

Under standard conditions for discounted MDPs, out of all possible policies there existes at least one deterministic policy that is optimal, maximizing the value of state $V^\pi(s_t)$. For a deterministic policy, with a given transition probability

$P(s_{t+1}|s_t, a_t)$, the optimal state-action value function is denoted as $Q^\star(s_t, a_t)$ and the optimal state value function is denoted as $V^\star(s_t)$. The optimal Bellman equation of the state-action value function is:

$$Q^\star(s_t, a_t) = \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}}\left[R(s_t, a_t, S_{t+1}) + \gamma V^\star(s_{t+1})\right] \tag{1.19a}$$

$$= \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}}\left[R(s_t, a_t, S_{t+1}) + \gamma \max_{a_{t+1}\in\mathbb{A}} Q^\star(s_{t+1}, a_{t+1})\right] \tag{1.19b}$$

$$= \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t, a_t)\left[R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}\in\mathbb{A}} Q^\star(s_{t+1}, a_{t+1})\right] \tag{1.19c}$$

with the simplification of $R(s_t, a_t, s_{t+1}) = R(s_t, a_t)$, we can write the above optimal Bellman equation as

$$Q^\star(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_{S_{i>t}\sim \boldsymbol{T}}\left[\max_{a_{t+1}\in\mathbb{A}} Q^\star(s_{t+1}, a_{t+1})\right] \tag{1.20a}$$

$$= R(s_t, a_t) + \gamma \sum_{s_{t+1}\in\mathbb{S}}\left[\max_{a_{t+1}\in\mathbb{A}} P(s_{t+1}|s_t, a_t) Q^\star(s_{t+1}, a_{t+1})\right] \tag{1.20b}$$

$$= R(s_t, a_t) + \gamma \sum_{s_{t+1}\in\mathbb{S}}\left[\max_{a_{t+1}\in\mathbb{A}} P(s_{t+1}|s_t, a_t) Q^\star(s_{t+1}, a_{t+1})\right] \tag{1.20c}$$

Optimal Bellman Equation for State Value Function $V^\star(s_t)$

Bellman Equation for Optimal State Value Function $V^\star(s_t)$ is also known as Optimal Bellman Equation.

Under standard conditions for discounted MDPs, out of all possible policies there existes at least one deterministic policy that is optimal, maximizing the value of state $V^\pi(s_t)$. For a deterministic policy, with a given transition probability $P(s_{t+1}|s_t, a_t)$, the optimal state-action value function is denoted as $Q^\star(s_t, a_t)$ and the optimal state value function is denoted as $V^\star(s_t)$.

The optimal Bellman equation of the state value function is

$$V^\star(s_t) = \max_{a_t\in\mathbb{A}} Q^\star(s_t, a_t) \tag{1.21a}$$

$$= \max_{a_t\in\mathbb{A}} \mathbb{E}_{S_{t+1}\sim\rho}\left(R(s_t, a_t, S_{t+1}) + \gamma V^\star(s_{t+1})\right) \tag{1.21b}$$

$$= \max_{a_t\in\mathbb{A}} \sum_{s_{t+1}\in\mathbb{S}} P(s_{t+1}|s_t, a_t)\left(R(s_t, a_t, s_{t+1}) + \gamma V^\star(s_{t+1})\right) \tag{1.21c}$$

with the simplification of $R(s_t, a_t, s_{t+1}) = R(s_t, a_t)$, we can write the above optimal Bellman equation as

$$V^\star(s_t) = \max_{a_t \in \mathbb{A}} \{R(s_t, a_t) + \gamma \mathbb{E}_{S_{t+1} \sim \boldsymbol{T}} [V^\star(s_{t+1})]\} \tag{1.22a}$$

$$= \max_{a_t \in \mathbb{A}} \left\{ R(s_t, a_t) + \gamma \sum_{s_{t+1} \in \mathbb{S}} P(s_{t+1}|s_t, a_t) V^\star(s_{t+1}) \right\} \tag{1.22b}$$

1.1.2 Algorithm to solve a Markov Decision Process

1.1.3 Value Learning

Value Learning is a reinforcement learning methods where we want to estimate the value function ($Q(s, a)$, $V(s)$, $Q^\pi(s, a)$,$V^\pi(s)$) as accurate as possible.

The old way of estimation is to build a table to approximate the value function. This is only applicable for discrete state space and discrete action space. For example in Table 1.1:

Table 1.1: Example of a table estimation for the optimal state-action value function $Q(S, A)$

$Q(S, A)$	A_1	A_2	A_3	A_4
S_1	380	-95	20	173
S_2	-7	64	-195	210
S_3	152	72	413	-80

The more widely used estimation is to build a neural network $Q(s, a; \boldsymbol{\theta}^Q)$, $V(s; \boldsymbol{\theta}^V)$, $Q^\pi(s, a; \boldsymbol{\theta}^{Q^\pi})$,$V^\pi(s; \boldsymbol{\theta}^{V^\pi})$ to estimate the corresponding value $Q(s, a)$, $V(s)$, $Q^\pi(s, a)$,$V^\pi(s)$.

- Deep Q-Network (DQN): To learn optimal value function $Q(s, a)$ or $V(s)$. It requries 5-tuple $\{s_t, a_t, r_t, s_{t+1}\}$ generated from any policy stored in experience replay. It belongs to the off-policy learning.

- State Action Reward State Action (SARSA): To learn policy-based value function $Q^\pi(s, a)$ or $V^\pi(s)$. As suggested by the name, SARSA requires a five-tuple $\{s_t, a_t, r_t, s_{t+1}, \tilde{a}_{t+1}\}$ .It belongs to the on-policy algorithm. The policy-based value learning is usually combined with policy learning, which is usually called Actor-Critic Algorithm

To train any one of the Neural Network estimated Value function, we will make use of the Bellman equation, take $Q(s, a)$ as an example, the Bellman equation of

the optimal state action value function is given in Eq. 1.19

$$Q(s_t, a_t) = \mathbb{E}_{S_{i>t}\sim\rho}[U_t^\pi | s_t, a_t] = \mathbb{E}_{S_{i>t}\sim\rho}[R_t + \gamma V(S_{t+1})] = \mathbb{E}_{S_{i>t}\sim\rho}\left[r_t + \gamma \max_{a_{t+1}\in\mathbb{A}} Q(S_{t+1}, a_{t+1})\right] \tag{1.23}$$

where $U_t^\pi = \sum_{t=t}^{T} \gamma^i R_i$ and $R_t = R(s_t, a_t, s_{t+1})$.

Neural Network approximation is $\hat{q}_t = Q(s_t, a_t; \boldsymbol{\theta}^Q)$. The loss function is defined as $L(\boldsymbol{\theta}^Q) \overset{\text{def}}{=\!=} \mathbb{E}[\hat{q}_t - \hat{y}_t)^2]$ e.g. $L(\boldsymbol{\theta}^Q) \overset{\text{def}}{=\!=} \frac{1}{2}[\hat{q}_t - \hat{y}_t]^2$. We hope to minimize the loss function and based on the gradient descent:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}^Q} L(\boldsymbol{\theta}^\pi) = (\hat{q}_t - \hat{y}_t) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^Q} \hat{q}_t \tag{1.24}$$

To minimize the loss function, we need to update the parameter like this

$$\boldsymbol{\theta}^Q \leftarrow \boldsymbol{\theta}^Q - \alpha(\hat{q}_t - \hat{y}_t)\boldsymbol{\nabla}_{\boldsymbol{\theta}^Q} \hat{q}_t = \boldsymbol{\theta}^Q - \alpha\delta_t \boldsymbol{\nabla}_{\boldsymbol{\theta}^Q} \hat{q}_t \tag{1.25}$$

There are usually two ways of training the parameters of the neural network:

- Monte Carlo estimation: The target is the real observation of one episode (the return of one episode) starting from s_t and taking action a_t:

$$\hat{y}_t = u_t = \sum_{t=t}^{T} \gamma^i r_i \tag{1.26}$$

- Temperal Difference estimation: The m-step TD target is combined of the first m observations and the expected return of the remaining steps:

$$\hat{y}_t = \sum_{t=t}^{m} \gamma^i r_i + \gamma^m \max_{a_t\in\mathbb{A}} q(s_{t+m}, a_{t+m}; \boldsymbol{\theta}^Q) \tag{1.27}$$

1.1.4 Policy Learning

Policy learning is to find the best policy $\pi(a|s)$ directly. Policy learning is usually combined with also the value learning of this policy $q^\pi(s, a; \boldsymbol{\theta}^{Q^\pi})$ or $v(s; \boldsymbol{\theta}^{V^\pi})$. Take $Q^\pi(s, a)$ as an example, the Bellman equation of the state action value function is given in equation Eq. 1.8

$$Q^\pi(s_t, a_t) = \mathbb{E}_{S_{i>t}\sim\rho, A_{i>t}\sim\pi}[U_t^\pi | s_t, a_t] = \mathbb{E}_{S_{i>t}\sim\rho, A_{i>t}\sim\pi}[R_t + \gamma Q^\pi(S_{t+1}, A_{t+1})] \tag{1.28}$$

For discrete states and discrete action space, we could also use a table for estimation shown in Table 1.2:

Or using neural Network approximation is $\hat{Q}_t = Q(s_t, a_t; \boldsymbol{\theta}^{Q^\pi})$.

Similarly we can build the loss function and update the parameters as following

$$\boldsymbol{\theta}^{Q^\pi} \leftarrow \boldsymbol{\theta}^{Q^\pi} - \alpha(\hat{Q}_t - \hat{y}_t)\boldsymbol{\nabla}_{\boldsymbol{\theta}^{Q^\pi}} \hat{Q}_t = \boldsymbol{\theta}^{Q^\pi} - \alpha\delta_t \boldsymbol{\nabla}_{\boldsymbol{\theta}^{Q^\pi}} \hat{Q}_t \tag{1.29}$$

There are usually two ways of training the parameters of the neural network:

Table 1.2: Example of a table estimation for $Q^\pi(S, A)$

$Q^\pi(S, A)$	A_1	A_2	A_3	A_4
S_1	380	-95	20	173
S_2	-7	64	-195	210
S_3	152	72	413	-80

- Monte Carlo estimation: The target is the real observation of one episode (the return of one episode) starting from s_t and taking action a_t:

$$\hat{y}_t = u_t = \sum_{t=t}^{T} \gamma^i r_i \tag{1.30}$$

- Temperal Difference estimation: The m-step TD target is combined of the first m observations and the expected return of the remaining steps:

$$\hat{y}_t = \sum_{t=t}^{m} \gamma^i r_i + \gamma^m q(s_{t+m}, a_{t+m}; \boldsymbol{\theta}^{Q^\pi}) \tag{1.31}$$

There will be a neural network estimation of the policy function $\pi(a, s; \boldsymbol{\theta}^\pi)$. The objective function is the so-called J function:

$$J(\boldsymbol{\theta}^\pi) \overset{\text{def}}{=\!=} \mathbb{E}_{S_t \sim \rho}[V^\pi(S_t)] = \mathbb{E}_{S_t \sim \rho}[\mathbb{E}_{A_t \sim \pi}[Q^\pi(S_t, A_t)]] \tag{1.32}$$

$$\boldsymbol{g} \overset{\text{def}}{=\!=} \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} J(\boldsymbol{\theta}^\pi) \approx \mathbb{E}_{S_t \sim \rho, A_t \sim \pi}[Q^\pi(S_t, A_t) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(A_t | S_t; \boldsymbol{\theta}^\pi)] \tag{1.33}$$

Here we hope to maximize the J function, so we will use the gradient ascent to update the parameter by Eq. 1.34

$$\boldsymbol{\theta}^\pi \leftarrow \boldsymbol{\theta}^\pi + \beta \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} J(\boldsymbol{\theta}^\pi) \tag{1.34}$$

We need some more approximation and advanced techniques to represent the gradient $\boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} J(\boldsymbol{\theta}^\pi)$: Usually we will add baseline $b = V^\pi(S)$ for better training result

$$\boldsymbol{g}_b \overset{\text{def}}{=\!=} \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} J(\boldsymbol{\theta}^\pi) \approx \mathbb{E}_{S_t \sim \rho, A_t \sim \pi}[(Q^\pi(S_t, A_t) - V^\pi(S_t)) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(A_t | S_t; \boldsymbol{\theta}^\pi)] \tag{1.35}$$

The advantage function is defined as

$$A^\pi(s_t, a_t) \overset{\text{def}}{=\!=} Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{1.36}$$

We could see that in the gradients calculation for policy learning, we need to also know the value of this policyQ^π or V^π. There are many possible ways to handle this:

- REINFORCE: without base line

$$\tilde{\boldsymbol{g}}_{\text{REINFORCE}}(s_t, a_t; \boldsymbol{\theta}^\pi) = q(s_t, a_t; \boldsymbol{\theta}^{Q^\pi}) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) \tag{1.37}$$

Considering baseline, we will use the real observed return to approximate $Q^\pi(S_t, A_t)$, and use another neural network $v(s_t; \boldsymbol{\theta}^{V^\pi})$ to approximate $V^\pi(S_t)$

$$\tilde{\boldsymbol{g}}_{\text{REINFORCE}_b}(s_t, a_t; \boldsymbol{\theta}^\pi) = [u_t - v(s_t; \boldsymbol{\theta}^{V^\pi})] \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t : \boldsymbol{\theta}^\pi) \tag{1.38}$$

- Actor Critic (AC):

$$\tilde{\boldsymbol{g}}_{\text{AC}}(s_t, a_t; \boldsymbol{\theta}^\pi) = q(s_t, a_t; \boldsymbol{\theta}^{Q^\pi}) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) \tag{1.39}$$

When Actor-Critic considers the baseline, it becomes Advantage-Actor-Critic (A2C): First due to the Bellman equation of the state action value function Eq. 1.8

$$Q^\pi(s_t, a_t) = \mathbb{E}_{S_{t+1} \sim p(\cdot|s_t, a_t)}[R_t + \gamma V^\pi(S_{t+1})] \tag{1.40}$$

the expression in equation Eq. 1.35 $\mathbb{E}_{S_t \sim \rho, A_t \sim \pi}[(Q^\pi(S_t, A_t) - V^\pi(S_t)) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(A_t|S_t; \boldsymbol{\theta}^\pi)]$ could be transferred to

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} J(\boldsymbol{\theta}^\pi) \approx \mathbb{E}_{S_t \sim \rho, A_t \sim \pi, S_{t+1} \sim p(\cdot|s_t, a_t)}[R_t + \gamma V^\pi(S_{t+1}) - V^\pi(S_t)) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(A_t|S_t; \boldsymbol{\theta}^\pi)] \tag{1.41}$$

So the gradient in actor-critic algorithm is now approximated as

$$\tilde{\boldsymbol{g}}_{\text{A2C}} = (r_t + \gamma v(s_{t+1} : \boldsymbol{\theta}^{V^\pi}) - v(s_t, \boldsymbol{\theta}^{V^\pi})) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) \tag{1.42}$$

$$\boldsymbol{\theta}^\pi \leftarrow \boldsymbol{\theta}^\pi + \beta(r_t + \gamma v(s_{t+1} : \boldsymbol{\theta}^{V^\pi}) - v(s_t, \boldsymbol{\theta}^{V^\pi})) \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) = \boldsymbol{\theta}^\pi - \beta \delta_t \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) \tag{1.43}$$

The one-step TD error of value learning δ_t is defined as $\delta_t \stackrel{\text{def}}{=\!=} -(r_t + \gamma v(s_{t+1}; \boldsymbol{\theta}^{V^\pi}) - v(s_t; \boldsymbol{\theta}^{V^\pi}))$.

1.2 Deep Reinforcement Learning

Deep Reinforcement Learning is the Reinforcment Learning employed the deep neural networks.

Neural networks (like any other approximation structure like polynomials, splines, radial basis functions) can approximate any continous function within a compact set. In other words, given a continous function $f(x)$, a finite range

for the input $x \in [a, b]$ and an expected approximation accuracy ϵ, there exists a neural network that approximate $f(x)$ with an approximation error less than ϵ everywhere within $[a, b]$.

According to the universal approximation theorem, any continuous function can be arbitrarily closely approximated by a multi-layer perceptron with only one hidden layer and a finite number of neurons [**info10040122**, **cybenko1989approximation**, **hornik1989multilayer**, **yarotsky2017error**, **cuomo2022scientific**]. Deep neural networks is a type of artificial neural network with more than two layers.

The key idea of DRL is to utilize the deep neural networks as function approximators to provide arbitrarily accurate proxies of the original functions in large state spaces, such as:

$$F \simeq F(\cdot|\theta^F), \tag{1.44}$$

where F is one of the previously defined functions Q^π, V^π, Q, V, π and $\theta^F \in \Theta$ are real-vallued vectors of parameters. Thereby, the whole problem of determining values at each point of a high-dimensional space $\mathbb{A} \times \mathbb{S}$ to the lower-dimensional space Θ, $|\Theta| << |\mathbb{S} \times \mathbb{A}|$.

1.2.1 Deep Q-Network (DQN)

Deep Q-Network is based on the off-policy Q-learning scheme using the NN to approximate the optimal Q-function. The general DQN concept ist shown in Figure 1.2. State s_t is introduced as input to a deep neural network, with an appropriate number of hidden layers and nonlinear unit activations, which output an approximation of the action-value function $Q(s_t, a_t|\theta^Q) \in \mathbb{R}^{|\mathbb{A}|}$ If the state is

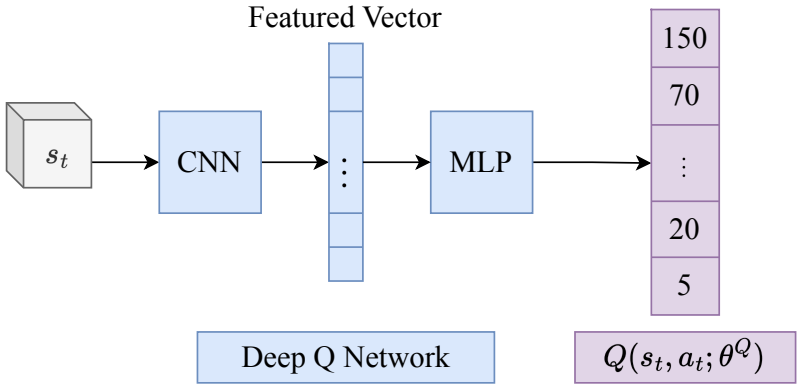

Figure 1.2: Deep Q-Network approximate the Q function for every available action using state as input $Q(s_t, a_t)$

matrix or multi-dimensional tensor, we will use Convolutional Neural Network ; if the state is a vector, then we can use Multi-layer Perceptron (MLP) directly.

The objective function that is minimized during training to determine θ^{Q} is given by the loss function

$$L_{Q}(\theta^{Q}) = \mathbb{E}\left[(Q(s_t, a_t|\theta^{Q}) - y_t)^2\right], \tag{1.45}$$

where μ is the behaviour policy of off-policy learning with $\mu \neq \pi$, ρ is the limiting distribution of states for policy μ, and $y_t \stackrel{\text{def}}{=\!=} R(s_t, a_t) + \gamma \max_{a_{t+1} \in \mathbb{A}} Q(s_{t+1}, a_{t+1})$.

To increase the training stability and robustness, we can also use two central features in the DQN algorithm [**mnih2013playing**]: the first is the experience replay (replay buffer) and the second is the use of a separate target network.

Algorithm 1: Deep Q Network(DQN) Training

Input: The decision process model MDP $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$, or POMDP $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{r}, \mathbf{O}, \gamma\}$

Output: The optimal state-action value function $Q(s, a)$

1 Collect the training data: Use any policy (e.g. use the $\epsilon-$greedy policy) to interact with the environment and get many trajectories(where each trajectory consists of the following elements $s_1, a_1, r_1, \cdots, s_T, a_T, r_T$):

$$a_t = \begin{cases} \text{argmax}_{a_t} Q(s_t, a_t; \boldsymbol{\theta}^{Q}), & \text{with prob. } 1-\epsilon \\ \text{randomly select from the action space } \mathbb{A}, & \text{with prob. } \epsilon. \end{cases} \tag{1.46}$$

Store the most recent 10^4 most recent (s, a, r, s_{t+1})tuples in the replay buffer;

2 Update the parameters $\boldsymbol{\theta}^{Q}$:

3 **repeat**

4 Select one four-tuple from the replay buffer based on priortized experience replay;

5 TD target $\hat{y}_t = r_t + \gamma \max_{a_{t+1} \in \mathbb{A}} Q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}^{Q})$;

6 TD error $\delta_t = \hat{q}_t = Q(s_t, a_t, \boldsymbol{\zeta}; \boldsymbol{\mu}^{Q}, \boldsymbol{\sigma}^{Q}) - \hat{y}_t$;

7 $\boldsymbol{\theta}^{Q} \leftarrow \boldsymbol{\theta}^{Q} - \alpha\delta_t \cdot \nabla_{\boldsymbol{\theta}^{Q}} Q(s_t, a_t; \boldsymbol{\theta}^{Q})$

8 **until** $\boldsymbol{\theta}^{Q}$ *converges*;

Disadvantages of DQN

The Deep Q Network has two problems: Overestimating and bootstrapping:

- Overestimating due to maximization : Wiithout detailed proofs, it could be shown that: Assume that $x_1, \cdots, x_d$ are d random real numbers. After

adding randome noise with zero expectation, we could get random variables $\mathsf{Z}_1, \cdots, \mathsf{Z}_\mathrm{d}$, it can be proved that

$$\begin{cases} \mathbb{E}[\mathsf{mean}(\mathsf{Z}_1, \mathsf{Z}_2, \cdots, \mathsf{Z}_\mathrm{d})] & = \mathsf{mean}(\mathsf{x}_1, \cdots, \mathsf{x}_\mathrm{d}) \\ \mathbb{E}\left[\max(\mathsf{Z}_1, \mathsf{Z}_2, \cdots, \mathsf{Z}_\mathrm{d})\right] & \geqslant \max(\mathsf{x}_1, \cdots, \mathsf{x}_\mathrm{d}) \end{cases} \tag{1.47}$$

Which means that the random noise will not affect the expectation value, but will increase the maximization value.

Look back at the Depp Q Network (DQN) $\mathsf{q}(\mathsf{s}, \mathsf{a}; \boldsymbol{\theta}^\mathrm{Q})$. Assume for all actions $\mathsf{a} \in \mathbb{A}$ and states $\mathsf{s} \in \mathbb{S}$, the output of DQN is the real value plus random noise with zero expectation ϵ

$$\mathsf{q}(\mathsf{s}, \mathsf{a}; \boldsymbol{\theta}^\mathrm{Q}) = \mathrm{Q}(\mathsf{s}, \mathsf{a}) + \epsilon \tag{1.48}$$

So obviously, $\mathsf{q}(\mathsf{s}, \mathsf{a}; \boldsymbol{\theta}^\mathrm{Q})$ is the unbiased estimation of $\mathrm{Q}(\mathsf{s}, \mathsf{a})$, and the following inequality holds:

$$\mathbb{E}_\epsilon \left[\max_{\mathsf{a} \in \mathbb{A}} \mathsf{q}(\mathsf{s}, \mathsf{a}; \boldsymbol{\theta}^\mathrm{Q}) \right] \geqslant \max_{\mathsf{a} \in \mathbb{A}} \mathrm{Q}(\mathsf{s}, \mathsf{a}) \tag{1.49}$$

Recall that our TD target is $\hat{\mathsf{y}}_\mathsf{t} = \mathsf{r}_\mathsf{t} + \gamma \cdot \max_{\mathsf{a}_{\mathsf{t}+1} \in \mathbb{A}} \mathsf{q}(\mathsf{s}_{\mathsf{t}+1}, \mathsf{a}_{\mathsf{t}+1}; \boldsymbol{\theta}^\mathrm{Q})$. And Temperal difference algorithm encourages the estimation $\mathsf{q}(\mathsf{s}_\mathsf{t}, \mathsf{a}_\mathsf{t}; \boldsymbol{\theta}^\mathrm{Q})$ to come close to the TD target. That is why the estimation $\mathsf{q}(\mathsf{s}, \mathsf{a}; \boldsymbol{\theta}^\mathrm{Q})$would overestimate the true value $\mathrm{Q}(\mathsf{s}, \mathsf{a})$.

- Bootstrapping: bootstrapping means the problem when we evaluate ourselves based on our value. For the Temperal Difference algorithm, the overestimating at time step $\mathsf{t}+1$ will propagate to the time step t.

Improve the training algorithm

To overcome the above overestimating problem of Temporal Difference Training algorithm, we could use another target network or double Q learning Review that for the basic Q learning algorithm: the TD target is calculated as

$$\hat{\mathsf{y}}_\mathsf{t} = \mathsf{r}_\mathsf{t} + \gamma \max_{\mathsf{a}_{\mathsf{t}+1} \in \mathbb{A}} \mathsf{q}(\mathsf{s}_{\mathsf{t}+1}, \mathsf{a}_{\mathsf{t}+1}; \boldsymbol{\theta}^\mathrm{Q}) = \mathsf{r}_\mathsf{t} + \gamma \mathsf{q}\left(\mathsf{s}_{\mathsf{t}+1}, \operatorname*{argmax}_{\mathsf{a}_{\mathsf{t}+1} \in \mathbb{A}} \mathsf{q}(\mathsf{s}_{\mathsf{t}+1}, \mathsf{a}_{\mathsf{t}+1}; \boldsymbol{\theta}^\mathrm{Q}); \boldsymbol{\theta}^\mathrm{Q}\right) \tag{1.50}$$

- Q-learning with target network could alleviate bootstrapping TD target is

$$\hat{\mathsf{y}}_\mathsf{t} = \mathsf{r}_\mathsf{t} + \gamma \mathsf{q}\left(\mathsf{s}_{\mathsf{t}+1}, \operatorname*{argmax}_{\mathsf{a}_{\mathsf{t}+1} \in \mathbb{A}} \mathsf{q}(\mathsf{s}_{\mathsf{t}+1}, \mathsf{a}_{\mathsf{t}+1}; \boldsymbol{\theta}^{\mathrm{Q}-}); \boldsymbol{\theta}^{\mathrm{Q}-}\right) \tag{1.51}$$

- Double Q-learning: could alleviate bootstrapping and overestimating TD target is

$$\hat{y}_t = r_t + \gamma q\left(s_{t+1}, \underset{a_{t+1}\in\mathbb{A}}{\operatorname{argmax}}\, q(s_{t+1}, a_{t+1}; \boldsymbol{\theta}^Q); \boldsymbol{\theta}^{Q-}\right) \tag{1.52}$$

Improved the Neural Network Architecture

Now we will focus on the techniques to improve the architecture design and the training algorithm is kept the same as the prrevious DQN:

- Dueling Network: Divide the state-action value to the state value and the advantage function shown in Figure 1.3:

$$Q(s_t, a_t) = V(s_t) + D(s_t, a_t) \tag{1.53}$$

  To assure the uniqueness, the dueling network is designed as

$$Q(s_t, a_t; \boldsymbol{\theta}^Q) \overset{\text{def}}{=\!=} V(s_t; \boldsymbol{\theta}^V) + V(s_t, a_t; \boldsymbol{\theta}^D) - \max_{a_t\in\mathbb{A}} D(s_t, a_t; \boldsymbol{\theta}^D) \tag{1.54}$$

  or

$$Q(s_t, a_t; \boldsymbol{\theta}^Q) \overset{\text{def}}{=\!=} V(s_t; \boldsymbol{\theta}^V) + V(s_t, a_t; \boldsymbol{\theta}^D) - \mathit{mean}_{a_t\in\mathbb{A}} D(s_t, a_t; \boldsymbol{\theta}^D) \tag{1.55}$$

  And $\boldsymbol{\theta}^Q = (\boldsymbol{\theta}^D, \boldsymbol{\theta}^V)$

- Noisy net: Replace the parameters of the Neural Network with the mean and standard deviation

$$\theta = \mu + \sigma \odot \boldsymbol{\zeta}, \tag{1.56}$$

  where $\odot$ is the element-wise product, $\boldsymbol{\zeta}$ is random noise following standard normal distribution $\mathcal{N}(0, 1)$; μ and σ are parameters to be learned. Thus the parameters of noisy network is doubled. The adding noise will contribute to the exploration and robustness of the result. Based on the previous algorithm of DQN 1, the modifications are the following:

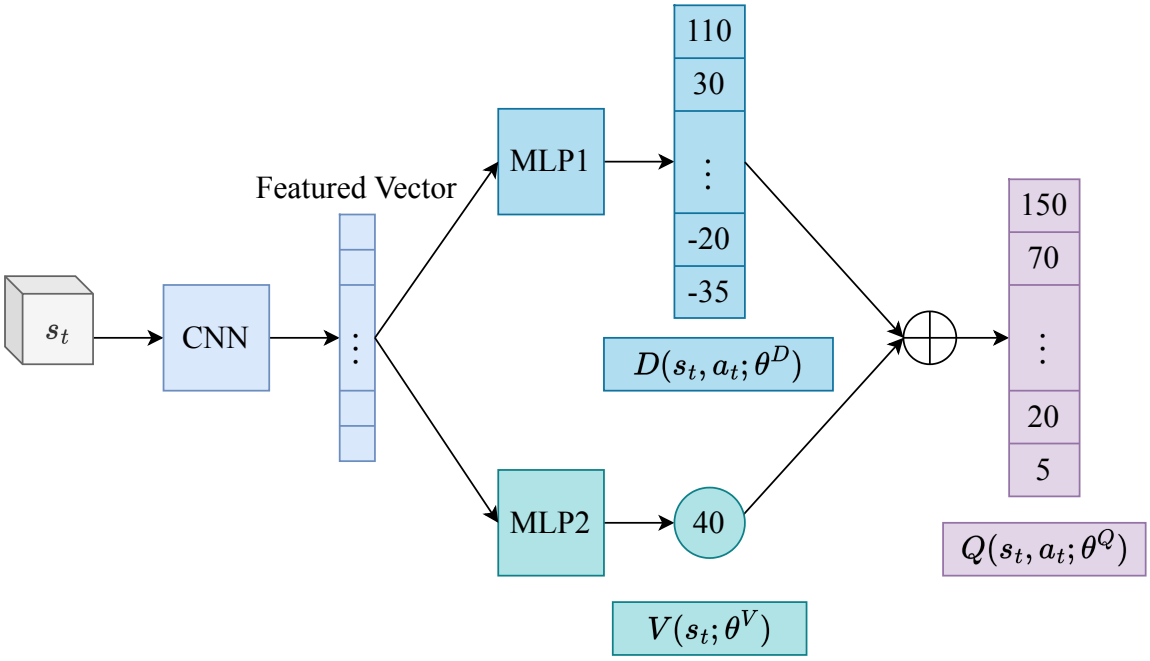

Figure 1.3: Dueling Network architecture

Algorithm 2: Deep Q Network(DQN) with Noisy Net Training

Input: The decision process model MDP $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$, or POMDP $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{r}, \mathbf{O}, \gamma\}$

Output: The optimal state-action value function $Q(s, a)$

Collect the training data: Use any policy (e.g. here we can directly use the noisy DQN) to interact with the environment and get many trajectories(where each trajectory consists of the following elements $s_1, a_1, r_1, \cdots, s_T, a_T, r_T$):

$$a_t = \underset{a_t}{\operatorname{argmax}}\, Q(s_t, a_t, \boldsymbol{\zeta}; \boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q) \tag{1.57}$$

Store the most recent 10^4 most recent (s, a, r, s_{t+1})tuples in the replay buffer;

Update the parameters $\boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q$:

repeat

  Select one four-tuple from the replay buffer based on priortized experience replay;

  TD target $\hat{y}_t = r_t + \gamma \max_{a_{t+1} \in \mathbb{A}} Q(s_{t+1}, a_{t+1}, \boldsymbol{\zeta}; \boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q)$;

  TD error $\delta_t = Q(s_t, a_t, \boldsymbol{\zeta}; \boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q) - \hat{y}_t$;

$$\begin{cases} \boldsymbol{\mu}^Q & \leftarrow \boldsymbol{\mu}^Q - \alpha_\mu \delta_t \cdot \nabla_{\boldsymbol{\mu}^Q} Q(s_t, a_t, \boldsymbol{\zeta}; \boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q) \\ \boldsymbol{\sigma}^Q & \leftarrow \boldsymbol{\sigma}^Q - \alpha_\sigma \delta_t \cdot \nabla_{\boldsymbol{\sigma}^Q} Q(s_t, a_t, \boldsymbol{\zeta}; \boldsymbol{\mu}^Q, \boldsymbol{\sigma}^Q) \end{cases} \tag{1.58}$$

until $\boldsymbol{\theta}^Q$ *converges*;

After training, we will use the trained network to make decisions. At this stage, no noise is necessary and the noisy network becomes standard DQN

$$Q(s, a, \zeta; \mu^Q, \sigma^Q = 0) = Q(s, a; \mu^Q) \tag{1.59}$$

1.2.2 Deep Policy Gradients

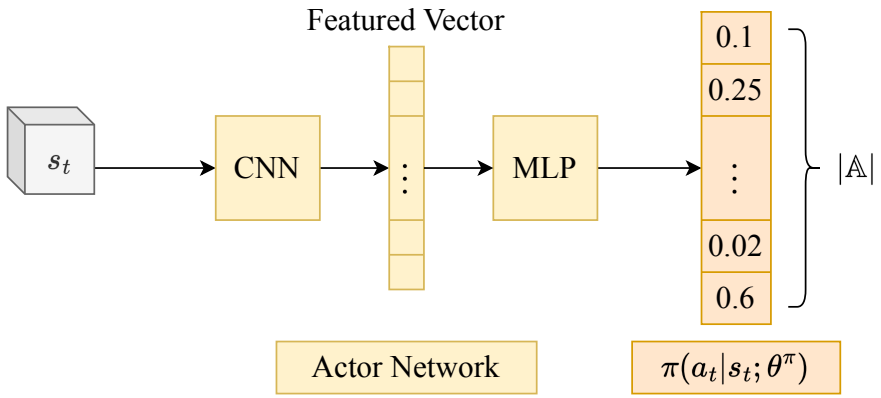

Figure 1.4: NN approximates $\pi(a_t|s_t)$

Commonly used policy gradients algorithms are REINFORCE and Actor Critic. Actor-Critic method shown in the Figure 1.5 and its advanced version like Advantage Actor-Critic (A2C). When it comes to multi-agent actor critic training algorithm, the so-called Asynchronous Advantage Actor-Critic (A3C) is used very often.

Actor-Critic Algorithm

Here is the basic training process of Actor-Critic is the following:

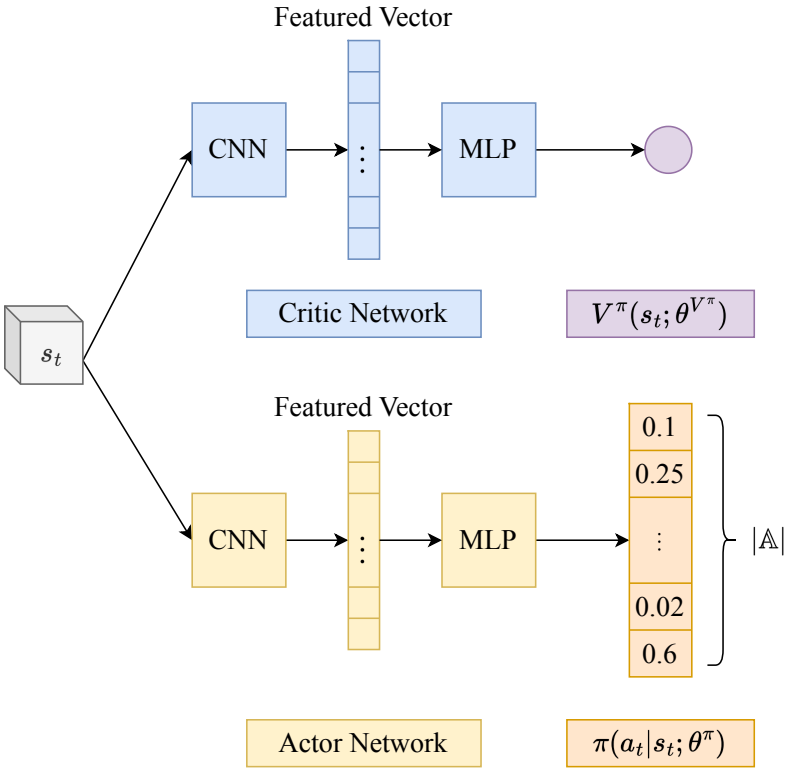

Figure 1.5: Actor Critic Neural Network architecture

Algorithm 3: Actor-Critic(AC)

Input: The decision process model MDP $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$, or POMDP $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{r}, \mathbf{O}, \gamma\}$

Output: The optimal state-action-policy value function $Q^{\pi}(s, a)$ and the policy function $\pi(a|s)$

Initialize the policy network and the value network randomly as $\pi(s, a, \boldsymbol{\theta}^{\pi})$ and $q(s, a; \boldsymbol{\theta}^{Q^{\pi}})$; **repeat**

  Take action according to policy network $a_t = \pi(\cdot|s_t; \boldsymbol{\theta}^{\pi})$ and perform action a_t;

  Observe the reward $r_t = r(s_t, a_t)$ and the new state s_{t+1} from the environment;

  Take action according to policy network $\tilde{a}_{t+1} = \pi(\cdot|s_{t+1}; \boldsymbol{\theta}^{\pi})$ but not perform action $\tilde{a}_{t+!}$;

  Use the value network to evaluate the two time steps:

$$\begin{cases} \hat{q}_t & = q(s_t, a_t; \boldsymbol{\theta}^{Q^{\pi}}) \\ \hat{q}_{t+1} & = q(s_{t+1}, \tilde{a}_{t+1}; \boldsymbol{\theta}^{Q^{\pi}}) \end{cases} \tag{1.60}$$

  Calculate TD target and TD error

$$\begin{cases} \hat{y}_t & = r_t + \gamma \hat{q}_{t+1} \\ \delta_t & = \hat{q}_t - \hat{y}_t \end{cases} \tag{1.61}$$

  Update the parameters

To alleviate the bootstrapping problem of time difference algorithm, we can also introduce a target value network separately to calculate TD target $q(s, a; \boldsymbol{\theta}^{Q^\pi -})$, then the Actor Critic C algorithm 3 becomes:

Algorithm 4: Actor-Critic(AC) with target network

Input: The decision process model MDP $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$, or POMDP $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{r}, \mathbf{O}, \gamma\}$

Output: The optimal state-action-policy value function $Q^\pi(s, a)$ and the policy function $\pi(a|s)$

Initialize the policy network and the value network randomly as $\pi(s, a, \boldsymbol{\theta}^\pi)$ $q(s, a; \boldsymbol{\theta}^{Q^\pi})$, and the target value network $q(s, a; \boldsymbol{\theta}^{Q^\pi -})$;

repeat

Take action according to policy network $a_t = \pi(\cdot|s_t; \boldsymbol{\theta}^\pi)$ and perform action a_t;

Observe the reward $r_t = r(s_t, a_t)$ and the new state s_{t+1} from the environment;

Take action according to policy network $\tilde{a}_{t+1} = \pi(\cdot|s_{t+1}; \boldsymbol{\theta}^\pi)$ but not perform action $\tilde{a}_{t+!}$;

Use the value network to evaluate the two time steps:

$$\begin{cases} \hat{q}_t & = q(s_t, a_t; \boldsymbol{\theta}^{Q^\pi}) \\ \hat{q}_{t+1} & = q(s_{t+1}, \tilde{a}_{t+1}; \boldsymbol{\theta}^{Q^\pi -}) \end{cases} \tag{1.63}$$

Calculate TD target and TD error

$$\begin{cases} \hat{y}_t & = r_t + \gamma \hat{q}_{t+1} \\ \delta_t & = \hat{q}_t - \hat{y}_t \end{cases} \tag{1.64}$$

Update the parameters

$$\begin{cases} \boldsymbol{\theta}^\pi & \leftarrow \boldsymbol{\theta}^\pi - \beta \cdot \hat{q}_t \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^\pi} \ln \pi(a_t|s_t; \boldsymbol{\theta}^\pi) \\ \boldsymbol{\theta}^{Q^\pi} & \leftarrow \boldsymbol{\theta}^{Q^\pi} - \alpha \cdot \delta_t \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^Q} Q(s_t, a_t; \boldsymbol{\theta}^{Q^\pi}) \\ \boldsymbol{\theta}^{Q^\pi -} & \leftarrow \tau \cdot \boldsymbol{\theta}^{Q^\pi} + (1 - \tau) \boldsymbol{\theta}^{Q^\pi -}, \end{cases} \tag{1.65}$$

where $\tau \in (0, 1)$

until $Q(s, a; \boldsymbol{\theta}^{Q^\pi})$ *and* $\pi(a|s; \boldsymbol{\theta}^\pi)$ *converges*;

Advantage Actor-Critic (A2C) Algorithm

Algorithm 5: Advantage Actor-Critic(A2C) with target network

Input: The decision process model MDP $\{\mathbb{S}, \mathbb{A}, \mathbf{T}, \mathbf{r}, \gamma\}$, or POMDP $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{r}, \mathbf{O}, \gamma\}$

Output: The optimal state-action-policy value function $V^{\pi}(s, a)$ and the policy function $\pi(a|s)$

Initialize the policy network and the value network randomly as $\pi(s, a, \boldsymbol{\theta}^{\pi})$ $v(s; \boldsymbol{\theta}^{V^{\pi}})$, and the target value network $v(s; \boldsymbol{\theta}^{V^{\pi}-})$;

repeat

  Take action according to policy network $a_t = \pi(\cdot|s_t; \boldsymbol{\theta}^{\pi})$ and perform action a_t;

  Observe the reward $r_t = r(s_t, a_t)$ and the new state s_{t+1} from the environment;

  Take action according to policy network $\tilde{a}_{t+1} = \pi(\cdot|s_{t+1}; \boldsymbol{\theta}^{\pi})$ but not perform action $\tilde{a}_{t+!}$;

  Use the value network to evaluate the two time steps:

$$\begin{cases} \hat{v}_t & = v(s_t, a_t; \boldsymbol{\theta}^{V^{\pi}}) \\ \hat{v}_{t+1} & = v(s_{t+1}, \tilde{a}_{t+1}; \boldsymbol{\theta}^{V^{\pi}-}) \end{cases} \tag{1.66}$$

  Calculate TD target and TD error

$$\begin{cases} \hat{y}_t & = r_t + \gamma \hat{v}_{t+1} \\ \delta_t & = \hat{q}_t - \hat{y}_t \end{cases} \tag{1.67}$$

  Update the parameters

$$\begin{cases} \boldsymbol{\theta}^{V^{\pi}} & \leftarrow \boldsymbol{\theta}^{V^{\pi}} - \alpha \cdot \delta_t \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^{V^{\pi}}} v(s_t; \boldsymbol{\theta}^{V^{\pi}}) \\ \boldsymbol{\theta}^{\pi} & \leftarrow \boldsymbol{\theta}^{\pi} - \beta \cdot \delta_t \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}^{\pi}} \ln \pi(a_t|s_t; \boldsymbol{\theta}^{\pi}) \\ \boldsymbol{\theta}^{V^{\pi}-} & \leftarrow \tau \cdot \boldsymbol{\theta}^{V^{\pi}} + (1-\tau)\boldsymbol{\theta}^{V^{\pi}-}, \end{cases} \tag{1.68}$$

  where $\tau \in (0, 1)$

until $Q(s, a; \boldsymbol{\theta}^{Q})$ *and* $\pi(a|s; \boldsymbol{\theta}^{\pi})$ *converges*;

Asynchronous Advantage Actor Critic (A3C) Algorithm

A3C is the parallel computing based on A2C. Parallel gradient descent

1.2.3 Multi-agent Actor Critic(MAC)

When there are multiple agents in the environment e.g. there are $\mathfrak{m}$ components in the system and $\mathfrak{n}$ control units, the system state is determined by components state $\boldsymbol{s}_t = \{s_t^j\}_{j=1}^{\mathfrak{m}}$. And the final action is also a combination of all control units $\boldsymbol{a}_t = \{a_t^i\}_{i=1}^{\mathfrak{n}}$.

Depends on the relationship of all components, there are typically four assumptions: fully cooperative mode, fully competitive mode, mixed cooperative & competitive and self-interested.

Fully cooperative mode

In fully cooperative assumption, the reward , return, value are the same for all agents. All agent uses one same value network

Deep Centralized Multi-agent Actor Critic (DCMAC)

Deep Centralized Multi-agent Actor Critic Architecture [**andriotis2019managing**] is shown in Figure 1.6:

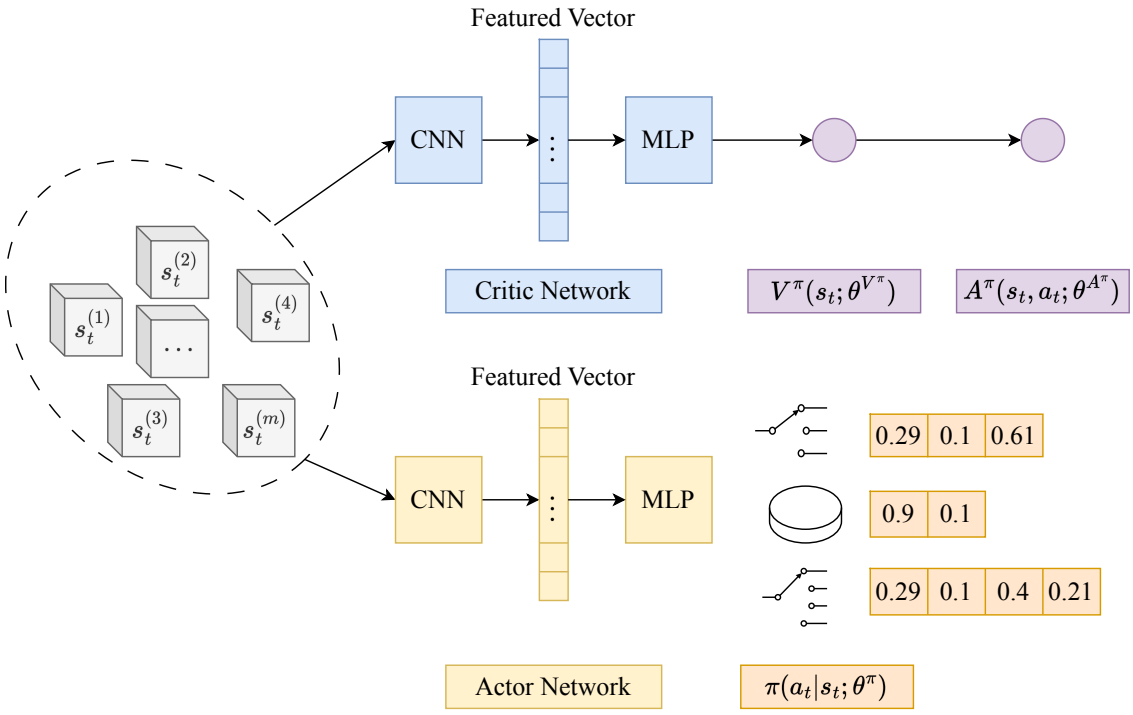

Figure 1.6: Deep Centralized Multi-agent Actor Critic Neural Network architecture

Deep Decentralized Multi-agent Actor Critic (DDMAC)

Deep Decentralized Multi-agent Actor Critic Architecture [**andriotis2021deep**] is shown in Figure 1.7:

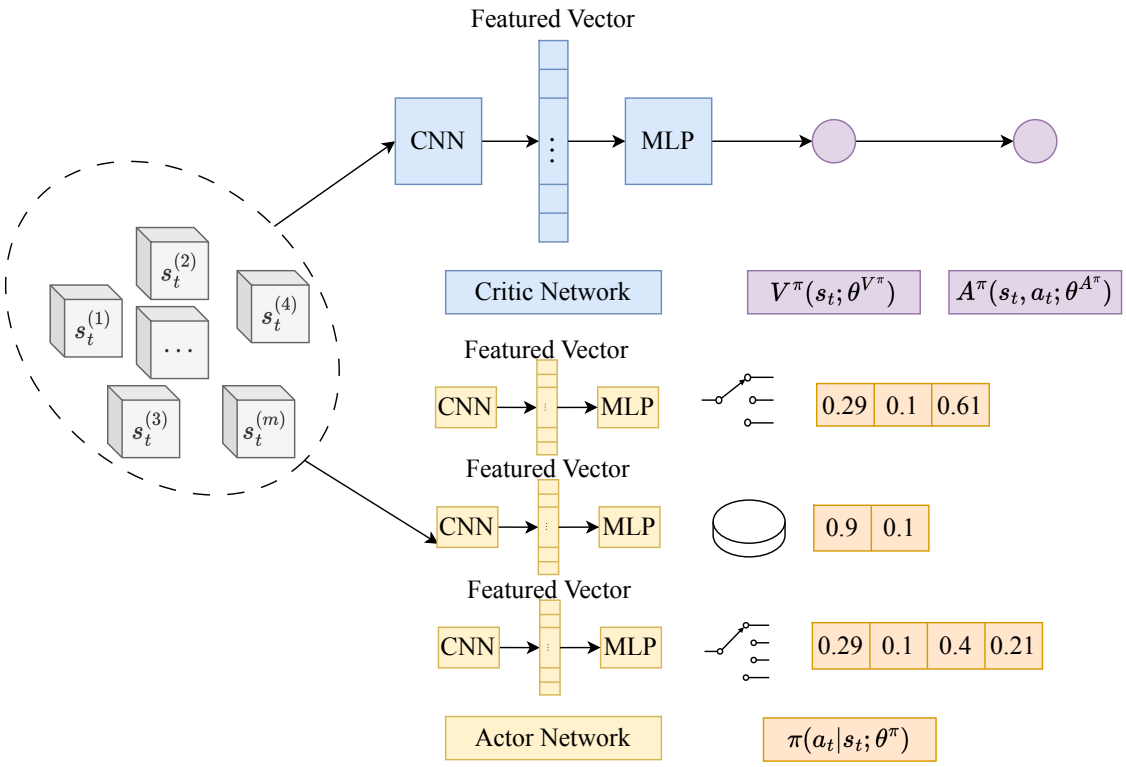

Figure 1.7: Deep Decentralized Multi-agent Actor Critic Neural Network architecture

Hierarchical Resource Allocation and Continuous-control Reinforcement Learning

Hierarchical Resource Allocation and continuous-control reinforcemnt learning [**andriotis2023struc**

1.3 Partially Observable Markov Decision Process

1.3.1 Basic Formulation and terminology

Based on the Markov Decision Process (MDP), the Partially Observable MDP can not fully observe the state, but instead, obtain some observation of the state. Thus we need to add observation space $\mathbb{O}$ and observation model $\mathsf{P}(\mathsf{o}_t|\mathsf{s}_t, \mathsf{a}_{t-1})$ in the original MDP. It can be written a 7-tuple $\{\mathbb{S}, \mathbb{A}, \mathbb{O}, \mathbf{T}, \mathbf{R}, \mathbf{O}, \gamma\}$. The sequential Probabilistic Grphical Model of POMDP is shown in Figure 1.8. The shaded round node denoting the state variable is not directly observable, but a hidden variable.

A Markov Decision Process shows the markovian property, where the next state s_{t+1} can be predicted just based on the current state s_t and action a_t, regardless of the whole past history $\mathsf{s}_{0:t}, \mathsf{a}_{0:t}$[1]. In other words, $\mathsf{s}_t, \mathsf{a}_t$ already contains all the information needed to predict the next state s_{t+1}.

But for Partially observable MDP, just the current observation o_t and the current action a_t could not provide all of the necessary information needed to make decisions. e.g. the Tiger problem,, we could not confidently choose the side when hearing the sound once behind another door. The whole history $\mathsf{o}_{0:t}$ are necessary for making decisions.

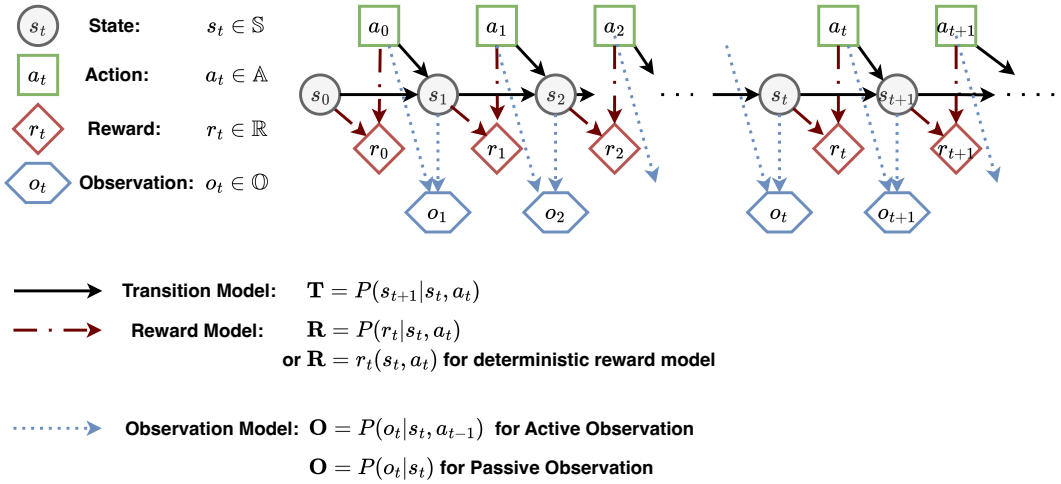

Figure 1.8: Probabilistic Graphical Model of Partially Observable Markovian Decision Process defined in State Space

[1] $\mathsf{s}_{0:t}$ is a short notation of $\mathsf{s}_0, \mathsf{s}_1, \cdots, \mathsf{s}_t$ and $\mathsf{a}_{0:t}$ a short notation of $\mathsf{a}_0, \mathsf{a}_1, \cdots, \mathsf{a}_t$

Belief Space $\mathbb{B}$

To implement those methods for solving MDP in solving POMDP, we can introduce the belief of the state variable, which is the probability distribution over the whole state space. to transfer the POMDP as a belief-MDP.

$$b_t(s) \overset{\text{def}}{=\!=} P(S_t = s | o_{1:t}, a_{0:t-1}) \tag{1.69}$$

$\mathbf{b_t} \in \mathbb{B}$ is the our believed-disctribution of all possible state variables based on all history observations and actions until time t. The dimension of the belief state space is dependent on the original state space $\mathbb{S}$.

- For discrete state space, the belief $\mathbf{b}$ is the Probability Mass Function (PMF). If the discrete state has N possible values, then the belief state $\mathbf{b_t}$ is a point in the $(N-1)$-simplex Δ^{N-1}. The definition of a simplex is

$$\Delta^{N-1} = \left\{ (p_1, p_2, \cdot, p_N) \in \mathbb{R}^N \middle| p_i > 0 \ \forall i \text{ and } \sum_{i=1}^{N} p_i = 1 \right\} \tag{1.70}$$

  - if $N = 2$, where the original state space is a 2D space, the belief state $\mathbf{b_t}$ is a point on a line between $(1, 0)$ and $(0, 1)$, e.g. $\mathbf{b_t} = (0.4, 0.6)$;
  - if $N = 3$, where the original state space is a 3D space, the belief state $\mathbf{b_t}$ is a point on a triangle between $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$, e.g.$\mathbf{b_t} = (0.3, 0.2, 0.5)$

- For continuous state space, the belief $\mathbf{b}$ is the Probability Density Function (PDF). If there are more than one chosen state variables are continuous, the belief of those states are also the joint PDF. In real life application, Under some assumptions, if we could parameterize the PDF, then the number of parameters determine the dimension of the belief space.

  - If we assume the Multi-gaussian Distribution for these $d_{\mathbb{S}}$ continuous state variables, for every continuous state variables, we have the mean and standard deviation to describe them. Considering also the correlation between those variables. The total number of parameters is calculated as:

$$d_{\mathbb{B}} = d_{\mathbb{S}} + \frac{d_{\mathbb{S}}(d_{\mathbb{S}} + 1)}{2} \tag{1.71}$$

  - If we assuem the Gaussian Mixture Distribution for these $d_{\mathbb{S}}$ continuous state variables, the number of parameters is calculated as

$$d_{\mathbb{B}} = K(d_{\mathbb{S}} + \frac{d_{\mathbb{S}}(d_{\mathbb{S}} + 1)}{2} + 1) - 1 \tag{1.72}$$

$\mathbf{b_t}$ contains the probability of the whole state space.

Belief Update Rule

When we connecting with the Bayesian updating, the belief $\mathbf{b}_t$ is actually the posterior distribution.

The belief $\mathbf{b}_t$ is our current guess of the state distributions based on the information we have until time t, it is not stationary, and it will update dynamically when we perform new actions a_t and the new observation o_{t+1}. If we have performed action a_t and obtained the new observation o_{t+1}, how to update our belief $\mathbf{b}_{t+1}$?

$$b_{t+1}(s_{t+1}) \stackrel{\text{def}}{=\!=} P(S_{t+1} = s_{t+1} \big| o_{t+1}, a_t, \mathbf{b}_t) = \frac{P(O_{t+1} = o_{t+1} \big| s_{t+1}, a_t, \mathbf{b}_t) P(S_{t+1} = s_{t+1} \big| a_t, \mathbf{b}_t)}{P(O_{t+1} = o_{t+1} \big| a_t, \mathbf{b}_t)} \tag{1.73}$$

where

- $P(O_{t+1} = o_{t+1} \big| s_{t+1}, a_t, \mathbf{b}_t) = P(O_{t+1} = o_{t+1} \big| s_{t+1}, a_t)$, which is the observation model defined in POMDP
- $P(S_{t+1} = s_{t+1} \big| a_t, \mathbf{b}_t) = \int_{s_t \in \mathbb{S}} P(S_{t+1} = s_{t+1} \big| s_t, a_t) \mathbf{b}_t(s_t) ds_t$
- $P(O_{t+1} = o_{t+1} \big| a_t, \mathbf{b}_t) = \int_{s_{t+1} \in \mathbb{S}} P(O_{t+1} = o_{t+1} \big| s_{t+1}, a_t) \int_{s_t \in \mathbb{S}} P(S_{t+1} = s_{t+1} \big| s_t, a_t) \mathbf{b}_t(s_t) ds_t ds_{t+1}$

Then the simplified Belief Update equation becomes

$$b_{t+1}(s_{t+1}) \stackrel{\text{def}}{=\!=} P(s_{t+1} \big| o_{t+1}, a_t, \mathbf{b}_t) = \frac{P(o_{t+1} \big| s_{t+1}, a_t) P(s_{t+1} \big| a_t, \mathbf{b}_t)}{P(o_{t+1} \big| a_t, \mathbf{b}_t)} \tag{1.74}$$

As in the real case, multi-dimensional integral are difficult to calculate. We will use the sequential Monte Carlo Simulation to update the belief (the posterior distribution) at each time step.

Use the particle filtering method to approximate the quantitty of interest like value function, State action value function etc.

Belief-MDP

After introducing the belief, the POMDP becomes the Belief-MDP shown in Figure 1.9, since the next belief state $\mathbf{b}_{t+1}$ is only dependent on the curretn belief state $\mathbf{b}_t$ and current action a_t. The belief state transition model can be derived from the above belief update formular:

$$P(\mathbf{b}_{t+1} \big| \mathbf{b}_t, a_t) = \int_{o_{t+1} \in \mathbb{O}} P(\mathbf{b}_{t+1} \big| \mathbf{b}_t, a_t, o_{t+1}) P(o_{t+1} \big| \mathbf{b}_t, a_t) do_{t+1} \tag{1.75}$$

$$P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t) = \sum_{o_{t+1}\in\mathbb{O}} P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t, o_{t+1})P(o_{t+1}|\mathbf{b}_t, a_t) \tag{1.76}$$

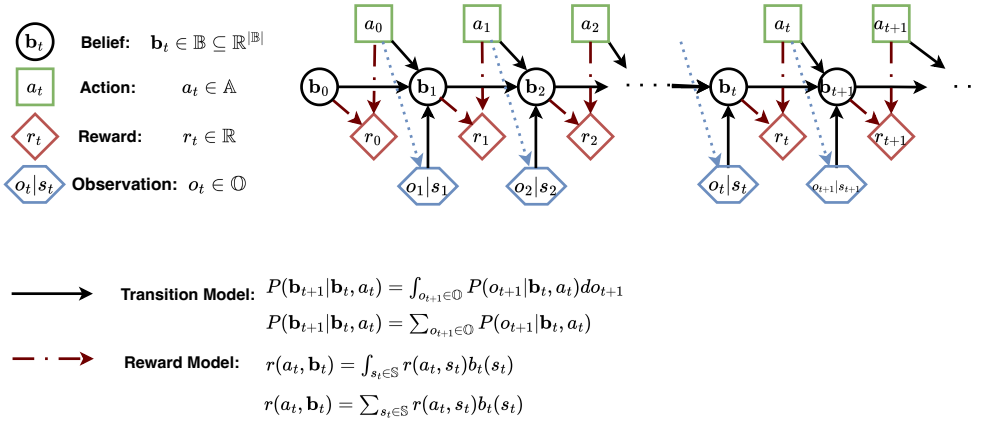

Figure 1.9: Probabilistic Graphical Model of Partially Observable Markovian Decision Process defined in Belief Space

Observation Model of Belief-MDP $P(o_{t+1}|\mathbf{b}_t, a_t)$

- For continuous State space

$$P(o_{t+1}|\mathbf{b}_t, a_t) = \int_{s_{t+1}\in\mathbb{S}} P(o_{t+1}|s_{t+1}, a_t) \int_{s_t\in\mathbb{S}} P(s_{t+1}|s_t, a_t)b_t(s_t)ds_t ds_{t+1} \tag{1.77}$$

- For discrete State space

$$P(o_{t+1}|\mathbf{b}_t, a_t) = \sum_{s_{t+1}\in\mathbb{S}} P(o_{t+1}|s_{t+1}, a_t) \sum_{s_t\in\mathbb{S}} P(s_{t+1}|s_t, a_t)b_t(s_t) \tag{1.78}$$

State Transition Model of Belief-MDP $\mathbf{T}_{belief}$

- For continuous Observation space

$$P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t) = \int_{o_{t+1}\in\mathbb{O}} P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t, o_{t+1})P(o_{t+1}|\mathbf{b}_t, a_t)do_{t+1} = \int_{o_{t+1}\in\mathbb{O}} P(o_{t+1}|\mathbf{b}_t, a_t)do_{t+1} \tag{1.79}$$

- For discrete Observation space

$$P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t) = \sum_{o_{t+1}\in\mathbb{O}} P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t, o_{t+1})P(o_{t+1}|\mathbf{b}_t, a_t) = \sum_{o_{t+1}\in\mathbb{O}} P(o_{t+1}|\mathbf{b}_t, a_t) \tag{1.80}$$

Reward Model of Belief-MDP $\mathbf{R}_{belief}$

If we consider determnistic reward model, $\mathbf{R}_{belief} = P(r_{t+1}|a_t, \mathbf{b}_t)$ can be written as $\mathbf{R}_{belief} = r_{belief}(a_t, \mathbf{b}_t)$

- For continuous Observation space

$$r_{belief}(a_t, \mathbf{b}_t) = \int_{s_t \in \mathbb{S}} r(a_t, s_t) b_t(s_t) \tag{1.81}$$

- For discrete Observation space

$$r_{belief}(a_t, \mathbf{b}_t) = \sum_{s_t \in \mathbb{S}} r(a_t, s_t) b_t(s_t) \tag{1.82}$$

The cumulative weighted total return $U^{\pi}(\mathbf{b}_t, a_t, \cdots, \mathbf{b}_T, a_T)$

The cumulative weighted total reward is now a function of all the belief states and actions from time t when taking a policy π:

$$U_t^{\pi} = U^{\pi}(\mathbf{b}_t, a_t, \cdots, \mathbf{b}_T, a_T) = \sum_{i=t}^{T} \gamma^{i-t} r_{belief}(a_i, \mathbf{b}_i) \tag{1.83}$$

where the discount factor $\gamma \in [0, 1]$: weighting the relative importance of the current reward against the future reward. In extreme cases when $\gamma = 0$, means only the current reward matters.

The State-Action Value Function $Q^{\pi}(\mathbf{b}_t, a_t)$

The State-Action value function $Q^{\pi}(\mathbf{b}_t, a_t)$ describes the value of the policy π given a State-Action Pair $Q^{\pi}(\mathbf{b}_t, a_t)$. It is a meassure of the State-Action pair at time step t and the policy π from time step $t + 1$. That is why we need to reduce all the randomness from all the future state and future action.

The value of taking an action a_t at the state $\mathbf{b}_t$ and using the strategy π for the rest of time span until T is calculated by the expectation of all the state and actions from time t to T. s

$$Q^{\pi}(\mathbf{b}_t, a_t) = \mathbb{E}_{\boldsymbol{B}_{i>t} \sim \boldsymbol{T}_{belief}, A_{i>t} \sim \pi} \left[U^{\pi}(\mathbf{b}_t, a_t, \boldsymbol{B}_{t+1}, A_{t+1}, \cdots, \boldsymbol{B}_T, A_T) | \mathbf{b}_t, a_t \right] \tag{1.84}$$

where $\boldsymbol{T}_{belief} = P(\mathbf{b}_t | \mathbf{b}_{t-1}, a_{t-1})$ is the transition probability distribution of state $\mathbf{b}_t$ and π is the probability of action (aka. the stochastic policy).

The State Value Function $V^{\pi}(\mathbf{b}_t)$

The State Value Function $V^{\pi}(\mathbf{b}_t)$ is a measure of the value of current state $\mathbf{b}_t$ and that is why we need to further eliminate the randomness of action at time A_t.

The value of a state $\mathbf{b}_t$ when taking a policy from this time step t until the rest of the time span can be expressed by the expected total return, mathematically written as:

$$
\begin{aligned}
V^{\pi}(\mathbf{b}_t) &= \mathbb{E}_{A_t \sim \mu}\left[Q^{\pi}(\mathbf{b}_t, A_t)\right] && (1.85a)\\
&= \mathbb{E}_{\boldsymbol{B}_{i>t}\sim \boldsymbol{T}_{belief}, A_{i>t}\sim\pi, A_t\sim\mu}\left[U^{\pi}(\mathbf{b}_t, A_t, \cdots, \mathbf{b}_T, A_T)|\mathbf{b}_t, a_t\right] && (1.85b)\\
&= \mathbb{E}_{\boldsymbol{B}_{i>t}\sim \boldsymbol{T}_{belief}, A_{i>t}\sim\pi}\left[U^{\pi}(\mathbf{b}_t, a_t, \cdots, \boldsymbol{B}_T, A_T)\right] && (1.85c)\\
&= \mathbb{E}_{\boldsymbol{B}_{i>t}\sim \boldsymbol{T}_{belief}, A_{i>t}\sim\pi}\left[\sum_{i=t}^{T}\gamma^{i-t} r_{belief}(a_i, \mathbf{b}_i)\middle|\mathbf{b}_t\right], && (1.85d)
\end{aligned}
$$

where μ shows the probability distribution of a_t at time t specifically, which is not necessarily equal to the policy π chosen to taken actions from time $t+1$ on.

The Optimal State Action Value Function $Q^{\star}(\mathbf{b}_t, a_t)$

The Optimal State Action Value function $Q^{\star}(\mathbf{b}_t, a_t)$ is a meassure of values of the current state-action pair $\mathbf{b}_t$. Based on the State Action Value function $Q^{\pi}(\mathbf{b}_t.a_t)$ we need to further eliminate the randomness of the policy π:

$$
Q^{\star}(\mathbf{b}_t, a_t) = \max_{\pi} \mathbb{E}_{\boldsymbol{B}_{i>t}\sim \boldsymbol{T}_{belief}}\left[\sum_{i=t}^{T}\gamma^{i-t} r_{belief}(a_i, \mathbf{b}_i)\middle|\mathbf{b}_t, a_t\right] \tag{1.86}
$$

The Optimal State Value Function $V^{\star}(\mathbf{b}_t)$

The Optimal State Value function $V^{\star}(\mathbf{b}_t)$ is a meassure of values of the current state $\mathbf{b}_t$ only. Based on the State Value function $V^{\pi}(\mathbf{b}_t)$ we need to further eliminate the randomness of the policy π:

$$
V^{\star}(\mathbf{b}_t) = \max_{\pi} \mathbb{E}_{\boldsymbol{B}_{i>t}\sim \boldsymbol{T}_{belief}}\left[\sum_{i=t}^{T}\gamma^{i-t} r_{belief}(a_i, \mathbf{b}_i)\middle|\mathbf{b}_t\right] \tag{1.87}
$$

The Optimal State Action Value Function $V^{\star}(\mathbf{b}_t)$

Bellman Equation for Optimal State Value Function $V^{\star}(\mathbf{b}_t)$ is also known as Optimal Bellman Equation.

Under standard conditions for discounted MDPs, out of all possible policies there existes at least one deterministic policy that is optimal, maximizing the value

of state $V^{\pi}(\mathbf{b}_t)$. For a deterministic policy, with a given transition probability $P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t)$, the optimal state-action value function is denoted as $Q^{\star}(\mathbf{b}_t, a_t)$ and the optimal state value function is denoted as $V^{\star}(\mathbf{b}_t)$.

The optimal Bellman equation of the state value function is

$$V^{\star}(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} Q^{\star}(\mathbf{b}_t, a_t) \tag{1.88a}$$

$$= \max_{a_t \in \mathbb{A}} \mathbb{E}_{\boldsymbol{B}_{t+1} \sim \boldsymbol{T}_{\mathtt{belief}}} \left(r_{\mathtt{belief}}(\mathbf{b}_t, a_t, \boldsymbol{B}_{t+1}) + \gamma V^{\star}(\mathbf{b}_{t+1}) \right) \tag{1.88b}$$

$$= \max_{a_t \in \mathbb{A}} \int_{\mathbf{b}_{t+1} \in \mathbb{B}} P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t) \left(r_{\mathtt{belief}}(\mathbf{b}_t, a_t, \boldsymbol{B}_{t+1}) + \gamma V^{\star}(\mathbf{b}_{t+1}) \right) d\mathbf{b}_{t+1} \tag{1.88c}$$

with the simplification of $r_{\mathtt{belief}}(\mathbf{b}_t, a_t, \mathbf{b}_{t+1}) = r_{\mathtt{belief}}(\mathbf{b}_t, a_t)$, we can write the above optimal Bellman equation as

$$V^{\star}(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \{ r_{\mathtt{belief}}(\mathbf{b}_t, a_t) + \gamma \mathbb{E}_{\boldsymbol{B}_{t+1} \sim \boldsymbol{T}_{\mathtt{belief}}} [V^{\star}(\mathbf{b}_{t+1})] \} \tag{1.89a}$$

$$= \max_{a_t \in \mathbb{A}} \left\{ r_{\mathtt{belief}}(\mathbf{b}_t, a_t) + \gamma \int_{\mathbf{b}_{t+1} \in \mathbb{B}} P(\mathbf{b}_{t+1}|\mathbf{b}_t, a_t) V^{\star}(\mathbf{b}_{t+1}) d\mathbf{b}_{t+1} \right\} \tag{1.89b}$$

$$= \max_{a_t \in \mathbb{A}} \left\{ \sum_{s_t \in \mathbb{S}} r(a_t, s_t) b_t(s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} P(o_{t+1}|\mathbf{b}_t, a_t) V^{\star}(\mathbf{b}_{t+1}) \right\} \tag{1.89c}$$

The optimal state-value function is defined over the continuous space of the belief simplex $\mathbb{B}$, which essentially consists of an infinite number of beliefs. However, it has been proven that the optimal value function is piece-wise linear and convex, and can thus be described by a finite number of affine hyperplanes. This important result reduces the decision problem to determining a finite set of vectors, also known as the $\hat{\alpha}$—vectors:

$$V^{\star}(\mathbf{b}) = \max_{\hat{\alpha} \in \Gamma} \sum_{s \in \mathbb{S}} b(s) \hat{\alpha}(s), \tag{1.90}$$

where Γ is the set comprising all α—vectors. After substituting the Eq. 1.90 for $V^{\star}(\mathbf{b}_{t+1})$into Eq. 1.89, we could get:

$$V^{\star}(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \left[\sum_{s_t \in \mathbb{S}} b_t(s_t) r(a_t, s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} P(o_{t+1}|\mathbf{b}_t, a_t) \max_{\hat{a} \in \Gamma} \sum_{s_{t+1} \in \mathbb{S}} b_{t+1}(s_{t+1}) \hat{\alpha}(s_{t+15}) \right] \tag{1.91}$$

Considering the belief update formular in Eq. 1.73 $b_{t+1}(s_{t+1}) \stackrel{\text{def}}{=\!=} P(s_{t+1}|o_{t+1}, a_t, \mathbf{b}_t) = \frac{P(o_{t+1}|s_{t+1},a_t)P(s_{t+1}|a_t,\mathbf{b}_t)}{P(o_{t+1}|a_t,\mathbf{b}_t)}$

$$V^\star(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \left[\sum_{s_t \in \mathbb{S}} b_t(s_t) r(a_t, s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} P(o_{t+1}|\mathbf{b}_t, a_t) \max_{\hat{a} \in \Gamma} \sum_{s_{t+1} \in \mathbb{S}} \frac{P(o_{t+1}\big|s_{t+1}, a_t)P(s_{t+1}\big|a_t, \mathbf{b}_t}{P(o_{t+1}\big|a_t, \mathbf{b}_t)} \right. \tag{1.92}$$

After rearanging the terms, we could get

$$V^\star(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \left[\sum_{s_t \in \mathbb{S}} b_t(s_t) r(a_t, s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} \max_{\hat{a} \in \Gamma} \sum_{s_{t+1} \in \mathbb{S}} P(o_{t+1}\big|s_{t+1}, a_t)P(s_{t+1}\big|a_t, \mathbf{b}_t)\hat{\alpha}(s_{t+1}) \right] \tag{1.93}$$

Since $P(s_{t+1}\big|a_t, \mathbf{b}_t) = \sum_{s_t \in \mathbb{S}} P(s_{t+1}\big|s_t, a_t)\mathbf{b}_t(s_t)ds_t$ we could get

$$V^\star(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \left[\sum_{s_t \in \mathbb{S}} b_t(s_t) r(a_t, s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} \max_{\hat{a} \in \Gamma} \sum_{s_{t+1} \in \mathbb{S}} P(o_{t+1}\big|s_{t+1}, a_t) \sum_{s_t \in \mathbb{S}} P(s_{t+1}\big|s_t, a_t)\mathbf{b}_t(s_t)\hat{\alpha}(s \right. \tag{1.94}$$

After arranging the sum order we could get

$$V^\star(\mathbf{b}_t) = \max_{a_t \in \mathbb{A}} \left[\sum_{s_t \in \mathbb{S}} b_t(s_t) r(a_t, s_t) + \gamma \sum_{o_{t+1} \in \mathbb{O}} \max_{\hat{a} \in \Gamma} \sum_{s_t \in \mathbb{S}} \mathbf{b}_t(s_t) \sum_{s_{t+1} \in \mathbb{S}} P(o_{t+1}\big|s_{t+1}, a_t)P(s_{t+1}\big|s_t, a_t)\hat{\alpha}(s \right. \tag{1.95}$$

1.4 How solve a Partially Observable Markov Decision Process

Although have mentioned above, we will here emphasize the way to solve the Partially Observable Markov Decision Process based on the previous section about the Reinforcement Learning to solve the Markov Decision Process 1.2.

Partially Observable Markov Decision Process can be treated as the Belelif-MDP. Where the belief is actually the posterior distribution of the state Eq. 1.69.

All of the algorithm and neural network when solving MDP problem used to have the states as input. But here we will first use sequential Monte Carlo Simulation (particle filtering) to get the updated belief (which is actually a number of particles with its corresponding weights) as the input. Which means that, the figures 1.2 become now 1.10

Figure 1.3 becomes now figure 1.11

The theoretical foundations of this research are rooted in several key areas. [**johnson2018foundations**] provided a comprehensive overview of the fundamental principles.

1.5 Common Implementations

Traditional methods have primarily focused on...

Method	Accuracy	Complexity	Scalability
Method A	0.85	O(n)	High
Method B	0.92	$O(n^2)$	Medium
Method C	0.78	$O(n \log n)$	High

Table 1.3: Comparison of traditional methods

1.6 Recent Advances

Recent developments in the field have introduced novel approaches...

1.6.1 Machine Learning Techniques

Deep learning methods have shown remarkable performance in various applications [**brown2021deeplearning**]. The key architectures include:

- **Convolutional Neural Networks (CNNs)**: Particularly effective for image processing...
- **Recurrent Neural Networks (RNNs)**: Suitable for sequential data analysis...
- **Transformer Networks**: Revolutionized natural language processing...

1.7 Research Gaps

Despite the significant progress, several research gaps remain:

1. Limited studies addressing the scalability issues...

2. Inadequate consideration of real-world constraints...

3. Lack of comprehensive benchmarking across diverse datasets...

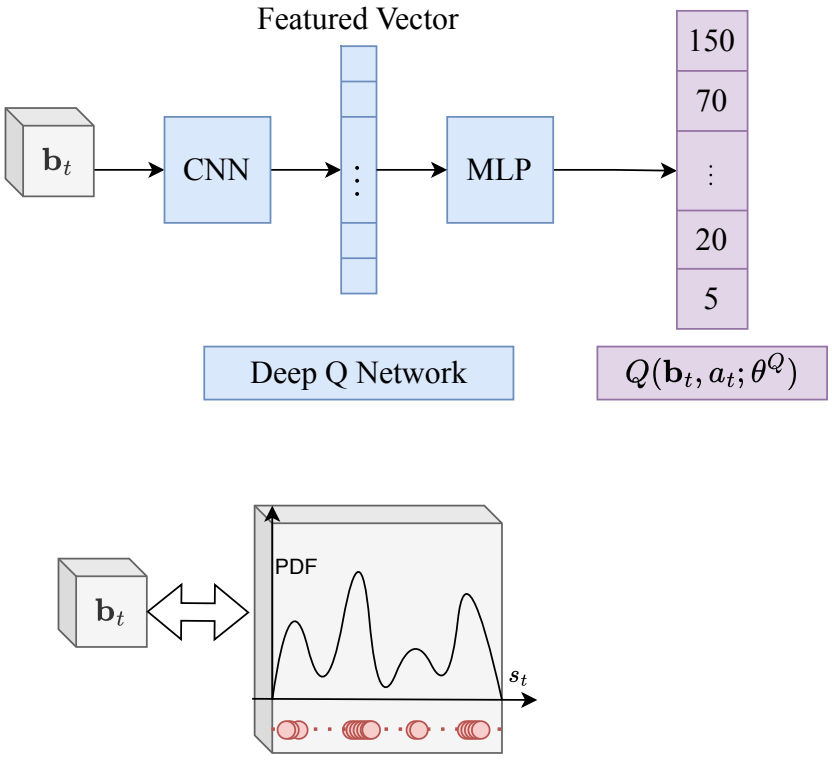

Figure 1.10: Deep Q-Network approximate the Q function for every available action using belief as input $\mathrm{Q}(\mathbf{b_t}, \mathbf{a_t})$

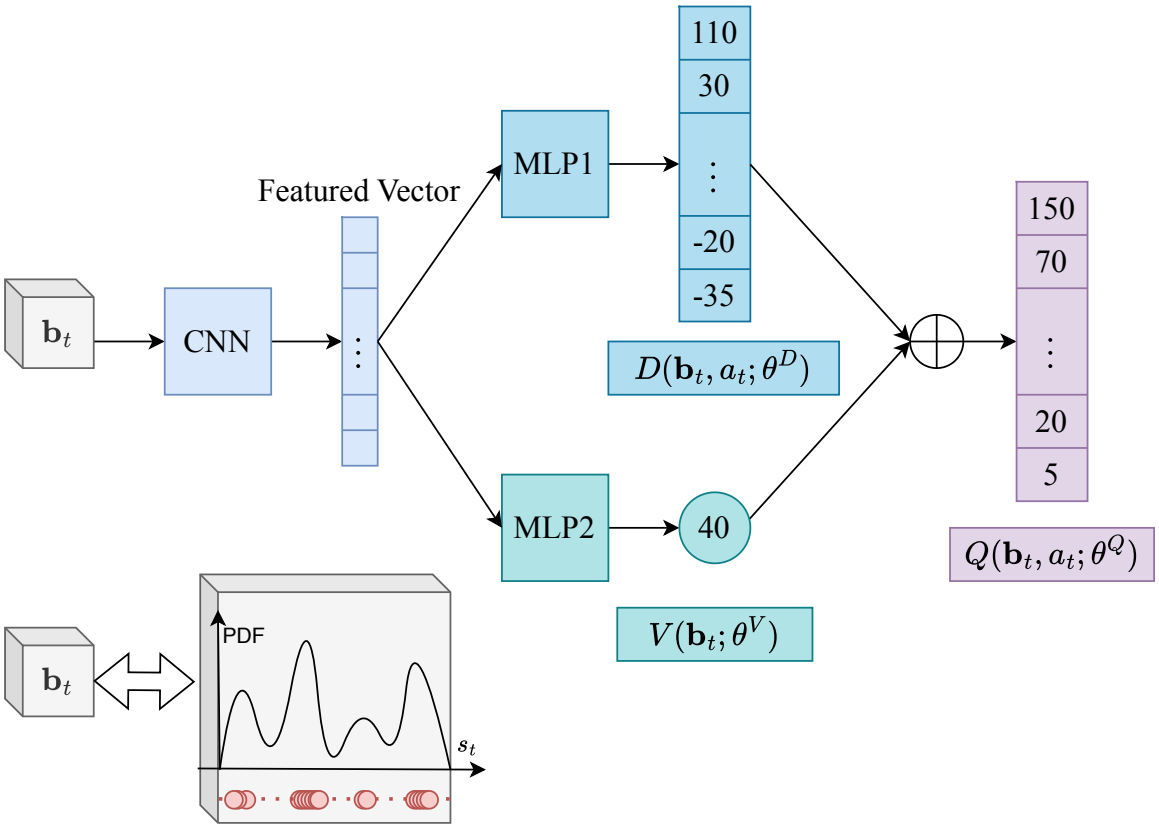

Figure 1.11: Dueling Network architecture