Notes on the MARL Book*

Zeyu Liang

Monday, November 25, 2024

Bellman Equation

$$v^\pi = r^0 + \gamma r^1 + \gamma^2 r^2 + \cdots = r^0 + \gamma v^{\pi,1}$$

Given a policy π, the state-value function $V^\pi(s)$ gives the "value" of the state s under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid S_t = s \right]$$

$$= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \left[R(s, a, s') + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right]$$

$$= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \left[R(s, a, s') + \gamma V^\pi(s') \right]$$

This is called Bellman Equation.

Why is Bellman Equation useful?
Because we can use a system of linear equations to solve for $V^\pi(\mathcal{S})$:

$$\begin{cases} V^\pi(s_1) = \sum_{a \in \mathcal{A}} \pi(a \mid s_1) \sum_{s' \in \mathcal{S}} T(s' \mid s_1, a) \left[R(s_1, a, s') + \gamma V^\pi(s') \right] \\ \vdots \\ V^\pi(s_k) = \sum_{a \in \mathcal{A}} \pi(a \mid s_k) \sum_{s' \in \mathcal{S}} T(s' \mid s_k, a) \left[R(s_k, a, s') + \gamma V^\pi(s') \right] \end{cases}$$

*Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. Authors: Stefano V. Albrecht, Filippos Christianos, Lukas Schäfer. Published by MIT Press, 2024

Similarly, we can define *action-value functions*:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[R(s,a,s') + \gamma \mathbb{E}_{\pi}\left[G_{t+1} \mid S_{t+1} = s'\right] \mid S_t = s, A_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} T(s' \mid s,a)\left[R(s,a,s') + \gamma V^{\pi}(s')\right] \end{aligned}$$

Iterative policy evaluation: Repeatedly updates sweep *all states* s:

$$V^{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} T(s' \mid s,a)\left[R(s,a,s') + \gamma V^{k}(s')\right]$$

Round function, discount factor

TD algorithms: Temporal Difference, does not rely on knowing all MDP (reward functions, transition probability, etc)
action value function updates:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(\mathcal{X} - Q(S_t, A_t)\right)$$

where $\mathcal{X} = r_t + \gamma Q(S_{t+1}, A_{t+1})$

Algorithm Sarsa for MDPs (with ε-greedy policies)

1. Initialize $Q(S,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$
2. Repeat for every episode:
3. Observe initial state S^0
4. With probability ε: choose random action $a^0 \in \mathcal{A}$
5. Otherwise: choose $a^0 \in \arg\max_a Q(S^0, a)$
6. For $t = 0, 1, 2, \ldots$ do:
7. Apply a^t, observe reward r^t and next state S^{t+1}
8. With probability ε: choose random action $a^{t+1} \in \mathcal{A}$
9. Otherwise: choose $a^{t+1} \in \arg\max_a Q(S^{t+1}, a)$
10. $Q(S^t, a^t) \leftarrow Q(S^t, a^t) + \alpha\left(r^t + \gamma Q(S^{t+1}, a^{t+1}) - Q(S^t, a^t)\right)$

Understanding: With ε probability *explore new action*, otherwise choose the *current best* action.

2.6 Difference between Q-learning and Sarsa

On page 35 bottom.

Algorithm DP Value Iteration for MDPs

1. Initialize $V(s) = 0$ for all $s \in \mathcal{S}$.
2. Repeat until V converged:
   $\forall s \in \mathcal{S} : V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T(s'|s,a) \left[R(s,a,s') + \gamma V(s') \right]$
3. Return optimal policy π^* with
   $\forall s \in \mathcal{S} : \pi^*(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T(s'|s,a) \left[R(s,a,s') + \gamma V(s') \right]$

In POSG, let $\hat{h}^t = \{s^0, a^0, s^1, a^1, \ldots, s^t, a^t\}$ denote a full history up to time t. $O(\hat{h}^{t+1})$ returns the history of joint observations.

Define the probability of joint observation: $O(o^t|\hat{h}^t, s^t) = \prod_{i \in \mathcal{I}} O_i(o_i^t|o^{t-1}, s^t)$

Define History-based expected return for agent i as:
$V_i(\pi) = \sum_{\hat{h}^t \in \mathcal{H}} \Pr(\hat{h}^t|\pi) M_i(\hat{h}^t)$
where $\Pr(\hat{h}^t|\pi)$ is the probability of full history $\hat{h}^t$ under π.

$$\Pr(\hat{h}^t|\pi) = \mu(s^0) O(o^0|a^0, s^0) \prod_{\tau=1}^{t} \pi(a^\tau|h^\tau) T(s^{\tau+1}|s^\tau, a^\tau) O(o^\tau|o^{\tau-1}, s^{\tau+1})$$

and $M_i(\hat{h}^t)$ is the discounted return for agent i in $\hat{h}^t$:

$$M_i(\hat{h}^t) = \sum_{\tau=0}^{t} \gamma^\tau R^i(s^\tau, a^\tau, s^{\tau+1})$$

$\pi(a^t|h^t)$ is the probability of joint action a^t under π after joint observation h^t

Recursive Expected Return

- Define $\text{BR}_i(\pi_{-i}) = \arg\max_{\pi_i} U_i(\pi_i, \pi_{-i})$ BR : Best Response
- In a zero-sum two agents game, a joint policy $\pi = (\pi_i, \pi_j)$ is a minimax solution,
  if $U_i(\pi) = \max_{\pi_i} \min_{\pi_j} U_i(\pi_i, \pi_j) = \min_{\pi_j} \max_{\pi_i} U_i(\pi_i, \pi_j) = -U_j(\pi)$
- **Minimax via Linear Programming**
  $\min U_j^*$
  subject to $\sum_{a_i \in A_i} R_j(a_i, a_j)\pi_i(a_i) \leq U_j^* \quad \forall a_j \in A_j$
  $\chi_{a_j} \geq 0, \sum_{a \in A_i} \chi_a = 1$
- **Define: Nash Equilibrium** In a general sum game with n agents, a joint policy $\pi = (\pi_1, \cdots, \pi_n)$ is a Nash Equilibrium if
  $\forall i, \pi_i' : U_i(\pi_i', \pi_{-i}) \leq U_i(\pi)$
- **ε-Nash Equilibrium**: $\forall i, \pi_i' : U_i(\pi_i', \pi_{-i}) - \varepsilon \leq U_i(\pi)$
- **Correlated Equilibrium**: In a general sum game with n-agents, let $\pi_C(a)$ be a joint policy that assigns probabilities to joint actions $a \in A$. Then, π_C is a correlated equilibrium if for every agent $i \in \mathcal{I}$ and every action modifier $\xi_i : A_i \to A_i$
  $\sum_{a \in A} \pi_C(a) R_i(\xi_i(a_i), a_{-i}) \leq \sum_{a \in A} \pi_C(a) R_i(a)$
  Correlated Equilibrium allows correlation between policies whereas Nash-Equilibrium doesn't.
- **Linear Programming for Correlated Equilibrium**
  maximize $\sum_{a \in A} \sum_{i \in \mathcal{I}} \chi_a R_i(a)$ $\chi_a = \pi(a)$ representing selecting action a under policy π
  subject to $\sum_{a \in A} \chi_a R_i(a) \geq \sum_{a \in A} \chi_a R_i(a_i', a_{-i})$??
- **Pareto Optimality**: A joint policy π is Pareto dominated by another policy π'
  if $\forall i : U_i(\pi') \geq U_i(\pi)$ and $\exists i : U_i(\pi') > U_i(\pi)$
  π' is better than π because everyone gets more.
  π is Pareto optimal if it's not Pareto-dominated by any other policy
- Welfare of π: $W(\pi) = \sum_{i \in \mathcal{I}} U_i(\pi)$
- Fairness of π: $F(\pi) = \prod_{i \in \mathcal{I}} U_i(\pi)$
  If Welfare is fixed, fairness is maximum if all $U_i(\pi)$ are equal.
  Welfare Optimality $\implies$ Pareto Optimality, but not vice versa.

- Regret: $= \max_{a_i \in A_i} \sum_{t=1}^{2} \left[R_i(a_i, a_{-i}^e) - R_i(a^e) \right]$
  if knows future, choose again. history
  No Regret if $\forall i : \lim_{z \to \infty} \frac{1}{z} \text{Regret}_i^z \leq 0$. Refer Fig 4.6.

Central Learning, CQ-Learning

1. Initialize $Q(S, a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A} = A_1 \times \cdots \times A_n$.
2. Repeat for every episode.
3. For $t = 1, \cdots$ do
4. Observe current state s^t.
5. With probability ε: choose random action $a^t \in \mathcal{A}$
6. Otherwise, choose $a^t \in \arg\max_a Q(s^t, a)$
7. Apply joint action a^t, observe rewards $r_1^t, \cdots, r_n^t$ and s^{t+1}
8. Transform $r_1^t \cdots r_n^t$ to r *This step is usually unclear for general-sum game.*
9. $Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \alpha \left[r^t + \gamma \max_{a'} Q(s^{t+1}, a') - Q(s^t, a^t) \right]$

Another limitation is that action space grows exponentially.
Also, Central policy to agents communication might not be feasible.

Independent Learning, IQL
The algorithm is essentially the same with Q-learning. Algorithm control agent i.

6.1 Value iteration for stochastic games

1. Initialize $V_i(s) = 0$ for all $s \in \mathcal{S}$ and $i \in \mathcal{I}$
2. Repeat until all V_i have converged
3. For all $s \in \mathcal{S}, i \in \mathcal{I}$, joint action $a \in \mathcal{A}$ do
   $M_{s,i}(a) \leftarrow \sum_{s' \in \mathcal{S}} T(s'|s, a) \left[R_i(s, a, s') + \gamma V_i(s') \right]$

4. For all $s \in \mathcal{S}$ and $i \in \mathcal{I}$
   $V_i(s) \leftarrow \text{Value}(M_{s,1}, \cdots, M_{s,n})$

$M_{s,i}(a)$ is the entry for matrix contains states and agents for rows and columns.
$M_{s,i}(a)$ represents an approximation of return for agent i choosing joint action a after state s.

Joint-action learning

It is a family of MARL algorithms based on temporal-difference learning. follows from Bellman Equation, the expected return for agent i when selecting joint action $a = (a_1, \cdots, a_n)$ in state s and subsequently follow joint policy π is

$$Q_i^{\pi}(s,a) = \sum_{s' \in S} T(s'|s,a) \left[R_i(s,a,s') + \gamma \sum_{a'' \in A} \pi(a''|s') Q_i^{\pi}(s',a'') \right]$$

The underlying idea in JAL-GT (joint action learning game theory) is that the set of joint-action
joint-action values $Q_1(s,\cdot), \cdots, Q_n(s,\cdot)$ can be treated as a non-repeated normal-form game.
Γ_s for state s, in which the reward function for agent i is given by
$R_i(a_1, \cdots, a_n) = Q_i(s, a_1, \cdots, a_n)$, (sometime noted as $\Gamma_{s,i}(\cdot)$)

Algorithm JAL-GT (controls agent i)

1. Initialize $Q_j(s,a) = 0$ for all $s \in S$, $a \in A$
2. Repeat for every episode:
3. for $t = 0, 1, 2 \cdots$ do:
4. observe current state s^t
5. With prob ε: choose random action a_i^t
6. Otherwise: solve Γ_{s^t} to get policies $(\pi_1, \cdots, \pi_n)$, then sample action $a_i^t \sim \pi_i$

7. Observe joint action $a^t = (a_1^t, \cdots, a_n^t)$, rewards $r_1^t, \cdots, r_n^t$, next state s^{t+1}

8. for all $j \in \mathcal{I}$ do

9. $Q_j(s^t, a^t) \leftarrow Q_j(s^t, a^t) + \alpha \left[r_j^t + \gamma Q_j(s^{t+1}, a^{t+1}) - Q_j(s^t, a^t) \right]$

For example, a two agent i and j and three possible actions for each agent. $\Gamma_s = [\Gamma_{s,i}, \Gamma_{s,j}]$

$$\Gamma_{s,i} = \begin{bmatrix} Q_i(s, a_{i,1}, a_{j,1}), & Q_i(s, a_{i,1}, a_{j,2}), & Q_i(s, a_{i,1}, a_{j,3}) \\ Q_i(s, a_{i,2}, a_{j,1}), & Q_i(s, a_{i,2}, a_{j,2}), & Q_i(s, a_{i,2}, a_{j,3}) \\ Q_i(s, a_{i,3}, a_{j,1}), & Q_i(s, a_{i,3}, a_{j,2}), & Q_i(s, a_{i,3}, a_{j,3}) \end{bmatrix}$$

Difference between Minmax-Q-learning and Q-learning (Littman 1994)
Minmax-Q is robust to optimal opponent, but can not analysis opponent weakness. Q-learning vice versa.

Agent modeling, based on history, belief, goal.
Policy reconstruction of j: learning parameters of $\hat{\pi}_j$ can be framed as a supervised learning based on (s_j^t, a_j^t) data, history of state and action

Let $C(s, a_j)$ denote the number of times that agent j selected action a_j in state s. Then, the agent $\hat{\pi}_j$ is defined as $\hat{\pi}_j(a_j|s) = \frac{C(s,a_j)}{\sum_{a'_j \in A_j} C(s,a'_j)}$
See Fig 6.6 for example.

Algorithm JAL-AM (controls agent i)

1. Initialize:

2. $Q_j(s, a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$

3. Agent set $\hat{\pi}_j(a_j|s) = \frac{1}{|\mathcal{A}_j|}$ for all $j \neq i$ and $s \in \mathcal{S}$.

4. Repeat for every episode:

5. For $t = 0, 1, 2, \cdots$ do:

6. Observe current state s^t.

7. ε-greedy explore
8. Otherwise, choose BR $a_i^t \in \arg\max_{a_i} AV_i(s^t, a_i)$
9. Observe joint action $a^t = (a_1^t, \ldots, a_n^t)$, r_i^t and s^{t+1}
10. For $j \neq i$ do:
11. update $\hat{\pi}_j$ agent model
12. $Q_j(s^t, a^t) \leftarrow Q_j(s^t, a^t) + \alpha\big[r_i^t + \gamma \max_{a_i'} AV_i(s^{t+1}, a_i') - Q_i(s^t, a^t)\big]$

where $AV_i(s, a_i) = \sum_{a_{-i} \in \mathcal{A}_{-i}} Q_i(s, (a_i, a_{-i})) \prod_{j \neq i} \hat{\pi}_j(a_j|s)$

- After observing agent j's action a_j^t at state s^t, agent i updates its belief by computing Bayesian posterior distribution:

$$\Pr(\hat{\pi}_j'|h^{t+1}) = \frac{\hat{\pi}_j(a_j^t|h^t)\Pr(\hat{\pi}_j'|h^t)}{\sum_{\hat{\pi}_j'' \in \hat{\Pi}_j} \hat{\pi}_j''(a_j^t|h^t)\Pr(\hat{\pi}_j''|h^t)}$$

where h^t is the history up to time t and $\hat{\pi}_j$ is the model or policy

- Read 6.4 for 2GA and WoLF-2GA

Algorithm Win or Learn Fast with policy hill climbing

For agent i

1. Initialize:
2. Learning rates $\alpha \in (0, 1]$ and $l_h, l_w \in [0, 1]$ with $l_h > l_w$
3. Value function $Q(s, a_i) \leftarrow 0$ and policy $\pi_i(a_i|s) \leftarrow \frac{1}{|\mathcal{A}_i|}$ for all $s \in \mathcal{S}$ and $a_i \in \mathcal{A}_i$
4. State counter $C(s) \leftarrow 0$ for all $s \in \mathcal{S}$
5. Average policy $\bar{\pi}_i \leftarrow \pi_i$
6. Repeat for every episode:

7. For $t = 0, 1, 2, \cdots$ do:
8. Observe s^t
9. ε-greedily explore
10. Otherwise: Sample action from policy, $a_i^t \sim \pi_i(\cdot|s^t)$
11. Observe r_i^t and s^{t+1}
12. Update Q-value:

$$Q(s^t, a_i^t) \leftarrow Q(s^t, a_i^t) + \alpha \left[r_i^t + \gamma \max_{a_i'} Q(s^{t+1}, a_i') - Q(s^t, a_i^t) \right]$$

13. Update average policy $\bar{\pi}_i$:

$$C(s^t) \leftarrow C(s^t) + 1$$

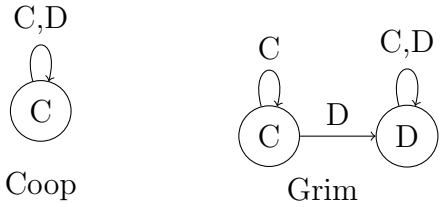
$$\forall a_i \in \mathcal{A}_i : \bar{\pi}_i(a_i|s^t) \leftarrow \bar{\pi}_i(a_i|s^t) + \frac{1}{C(s^t)} \left(\pi_i(a_i|s^t) - \bar{\pi}_i(a_i|s^t) \right)$$

14. Update policy π_i:

$$\forall a_i \in \mathcal{A}_i : \pi_i(a_i|s^t) \leftarrow \pi_i(a_i|s^t) + \Delta(s^t, a_i)$$

- A joint policy π is a global optimum in Γ if $\forall i, \pi' : U_i(\pi) \geq U_i(\pi')$
But even if both agents can be better off, what about fairness?
$[A : -1, B : -1] \rightarrow [A : 1, B : 100]$, will A choose the global optimum.

	C	D
C	$-1, -1$	$-5, 0$
D	$0, -5$	$-1, -1$



Suppose Agent 1 knows Agent 2 has either Coop or Grim model.

Chapter 7. Neural Network

$f_k(x_{k-1};\theta^k) = g_k(W_k^T x_{k-1} + b_k), \quad W_k \in \mathbb{R}^{d_{k-1}\times d_k},\ x_{k-1} \in \mathbb{R}^{d_{k-1}},\ b_k \in \mathbb{R}^{d_k}$

Goal: $\theta^* \in \arg\min L(\theta)$, Compute parameters θ^* to minimize $L(\theta)$

$$\text{MSE}: \ L(\theta) = \frac{1}{B}\sum_{k=1}^{B}\left(V^\pi(s_k) - V^\wedge(s_k;\theta)\right)^2$$

Vanilla gradient descent is too costly,
SGD: $\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta|d)\big|_{d\sim\mathcal{D}}$, d drawn from dataset $\mathcal{D}$.
Mini-batch: $\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta|B)$, $B = \{d_i \sim \mathcal{D}\}_{i=1}^B$
trade-off between converge fast and less variance.

Chapter 8. Deep Reinforcement Learning

$$Q(s^t,a^t) \leftarrow Q(s^t,a^t) + \alpha\left(r^t + \gamma\max_{a'} Q(s^{t+1},a') - Q(s^t,a^t)\right)$$

Define a neural network that receives state s as input, and outputs its estimated action value for each possible action.

$$s \rightarrow \begin{bmatrix} Q(s,a_1;\theta) \\ \vdots \\ Q(s,a_{|A|};\theta) \end{bmatrix}$$

Loss function $L(\theta) = (y^t - Q(s^t,a^t;\theta))^2$
To ensure state values for terminal states are always 0,
we can set $y^t = \begin{cases} r^t & \text{if } s^{t+1} \text{ is terminal} \\ r^t + \gamma\max_{a'} Q(s^{t+1},a';\theta) & \text{otherwise} \end{cases}$

$Q(s^{t+1},a';\theta) = 0$ Why setting $y^t = r^t$ can ensure this.

Algorithm 10

1. Initialize value network Q with random parameters θ
2. Same for all algorithms
3. Choose $a^t \in \arg\max_a Q(s^t, a; \theta)$
4. Apply action a^t; observe reward r^t and next state s^{t+1}
5. If s^{t+1} is terminal then
6. Target $y^t \leftarrow r^t$ computed by bootstrapping
7. Else Target $y^t \leftarrow r^t + \gamma \max_{a'} Q(s^{t+1}, a'; \theta)$ $\leftarrow$ moving target problem
8. Loss $L(\theta) \leftarrow (y^t - Q(s^t, a^t; \theta))^2$
9. Update parameters θ

A solution to this problem is by using another network, target network which periodically updates the parameters. Not too far from the estimates, and stays unchanged for a while.

Algorithm 11 Deep Q-Learning with target network

1. Initialize main network Q with random parameters θ
2. Initialize target network $\bar{\theta} = \theta$
3. *Same as algorithm 10*
4. Else: Target $y^t \leftarrow r^t + \gamma \max_{a'} Q(s^{t+1}, a'; \bar{\theta})$
5. Loss $L(\theta) \leftarrow (y^t - Q(s^t, a^t; \theta))^2$
6. Update θ
7. In a given interval, update $\bar{\theta}$

A phenomenon in Deep Learning is called catastrophic-forgetting, which is that updating the value function with the experience samples from the most recent episodes might cause the agent to forget how it arrived here (former experience).
To handle this problem, we might use a replay buffer D which stores experiences and train the model by randomly sample experiences from D.

Algorithm 12. Deep-Q-Network

1. Initialize Q with θ
2. Target network $\bar{\theta} = \theta$
3. Empty replay buffer $D = \{\}$
4. *Same for all*
5. Otherwise: choose $a^t \in \arg\max_a Q(s^t, a; \theta)$
6. Apply a^t and observe r^t, s^{t+1}
7. Store transition (s^t, a^t, r^t, s^{t+1}) to D
8. Sample random mini-batch B from D. $B = (s^k, a^k, r^k, s^{k+1})$
9. If s^{k+1} is terminal:
10. $y^k \leftarrow r^k$
11. else:
12. $y^k \leftarrow r^k + \gamma \max_{a'} Q(s^{k+1}, a'; \bar{\theta})$
13. Loss $L(\theta) \leftarrow \frac{1}{D} \sum_{k=1}^{D} (y^k - Q(s^k, a^k; \theta))^2$
14. Update parameters θ
15. In a interval, update $\bar{\theta}$

8.2 Why use policy gradient algorithms?

1. Able to represent probabilistic equilibrium
2. Able to handle continuous action space.

To represent a probabilistic policy for discrete action spaces using neural networks, the network receives a state as input and outputs a scalar (s, a) for all $a \in A$ then put them in a softmax function:

$$\pi(a|s;\phi) = \frac{e^{U(s,a)}}{\sum_{a' \in A} e^{U(s,a')}}$$

Policy Gradient Theorem:

$$\nabla_\phi J(\phi) \propto \sum_{s \in S} P_r(s|\pi) \sum_{a \in A} Q^\pi(s,a) \nabla_\phi \pi(a|s;\phi)$$

J represents the quality of a policy π with parameters ϕ, therefore, we want to maximize it.

$$= \mathbb{E}_{s \sim P_r(s|\pi), a \sim \pi(\cdot|s;\phi)} \left[Q^\pi(s,a) \nabla_\phi \log \pi(a|s;\phi) \right]$$

This expectation also clearly illustrates the restriction that optimization of parametrized policy is limited to on-policy data, that is, the data used to optimize π is generated by policy π itself.
We need to either approximate the derived expectation or obtain samples from it.

Algorithm REINFORCE

1. Initialize policy network π with random parameters ϕ
2. Repeat for every episode:
3. For time $t = 0 \cdots T-1$ do:
4. Observe s_t
5. Sample action $a_t \sim \pi(\cdot|s_t;\phi)$
6. Apply a_t and observe r_t and s_{t+1}
7. Loss $\mathcal{L}(\phi) \leftarrow -\frac{1}{T} \sum_{t=0}^{T-1} \left(\sum_{\tau=t}^{T-1} \gamma^{\tau-t} r_\tau \right) \log \pi(a_t|s_t;\phi)$
8. Update ϕ by minimizing $\mathcal{L}(\phi)$

8.2.3 REINFORCE: Monte Carlo Policy Gradient baseline???

Actor-Critic Algorithm
Actor-Critic is a family of algorithms that trains a parametrized policy called actor, and a value function called critic, alongside each other, able to update the policy in a single time step rather than episode.
Trade-off is that has higher variance.
Single step ⟵ N-step ⟵ Entire episode
Actor-Critic MC

Advantage Actor-Critic
$\text{Adv}^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$
$Q^{\pi}(s,a)$: At state s, do action a, then follow π
$V^{\pi}(s)$: At state s, directly follow π.

$$Q(s^t, a^t) = \begin{cases} r^t & \text{if } s^{t+1} \text{ is terminal} \\ r^t + \gamma V(s^{t+1}) & \text{otherwise} \end{cases}$$

Algorithm 14 Simplified advantage actor critic

1. Initialize actor (policy) network π with para ϕ
2. Initialize critic (state-value) network V with para θ
3. Repeat for every episode
4. For time step $t = 0, 1, 2 \cdots$ do
5. Observe s_t
6. Sample action $a_t \sim \pi(\cdot|s_t; \phi)$
7. Apply a_t, observe r_t and s_{t+1}
8. If s_{t+1} is terminal then
9. $\text{Adv}(s_t, a_t) \leftarrow r_t - V(s_t; \theta)$
10. Critic target $y_t \leftarrow r_t$

11. else
12. $\text{Adv}(s_t, a_t) \leftarrow r_t + \gamma V(s_{t+1}; \theta) - V(s_t; \theta)$
13. Critic target $y_t \leftarrow r_t + \gamma V(s_{t+1}; \theta)$
14. actor loss $\mathcal{L}(\phi) \leftarrow -\text{Adv}(s_t, a_t) \log \pi(a_t | s_t; \phi)$
15. critic loss $\mathcal{L}(\theta) \leftarrow (y_t - V(s_t; \theta))^2$
16. Update ϕ
17. Update θ

9.1.3 Centralized Training with Decentralized Execution (CTDE)

- Centralized training and decentralized execution (CTDE): used (centralized training to train agent policies, while each agent's policy itself only requires the agent's local observation.

Algorithm 17 Independent deep Q-networks. (Similar to DQN)

1. Initialize n value networks with random $\theta_1, \cdots, \theta_n; \theta$
2. Initialize n target networks with $\bar{\theta}_1 = \theta_1, \cdots, \bar{\theta}_n = \theta_n$
3. Initialize n replay buffers $D_1, \cdots, D_n$
4. For time step $t = 0, 1, 2, \cdots$ do
5. Collect observations $o_1^t, \cdots, o_n^t$
6. For agent $i = 1, \cdots, n$ do
7. ε-greedy explore
8. Otherwise: choose $a_i^t \in \arg\max_{a_i} Q_i(o_i^t, a_i; \theta_i)$

9. Apply actions $(a_1^t, \cdots, a_n^t)$; collect rewards $r_1^t, \cdots, r_n^t$ and next observations $o_1^{t+1}, \cdots, o_n^{t+1}$
10. For agent $i = 1, \cdots, n$ do
11. Store $(o_i^t, a_i^t, r_i^t, o_i^{t+1})$ to D_i
12. Sample random mini-batch of B-transitions $(o_i^k, a_i^k, r_i^k, o_i^{k+1})$ from D_i
13. If o_i^{k+1} is terminal then
14. Target $y_i^k \leftarrow r_i^k$
15. else
16. $y_i^k \leftarrow r_i^k + \gamma \max_{a_i'} Q_i(o_i^{k+1}, a_i'; \bar{\theta}_i)$
17. Loss $\mathcal{L}(\theta_i) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y_i^k - Q_i(o_i^k, a_i^k; \theta_i)\right)^2$
18. Update θ_i by minimizing $\mathcal{L}(\theta_i)$
19. In a interval, update $\bar{\theta}_i$

Multi-agent policy gradient theorem

$$\nabla_{\phi_i} J(\phi_i) \propto \mathbb{E}_{h \sim P_h(\cdot|\pi), a_i \sim \pi_i, a_{-i} \sim \pi_{-i}} \left[Q_i^{\pi}(h, \langle a_i, a_{-i} \rangle) \nabla_{\phi_i} \log \pi_i(a_i | h_i = o_i^t(h); \phi_i) \right]$$

Once the training is completed, the Critic network is no longer utilized. Therefore, there's no requirement for a decentralized critic network. So we can define critic as $V(h_1^t, \cdots, h_n^t; \theta_i)$ for agent i, value loss of estimated critic, becomes $\mathcal{L}(\theta_i) = (y_i - V(h_i^t, z^t; \theta_i))^2$ with $y_i = r_i^t + \gamma V(h_i^{t+1}, z^{t+1}; \theta_i)$

9.4.3 Centralized Action-Value Critics

For each agent i, trains a policy π_i which is Conditioned on agent i's observation history, For the critic, agent i trains an action-value function Q that is conditioned on individual observation history, Centralized information, and actions of all agents.

$$\mathcal{L}(\theta_i) = \left(y_i - Q(h_i^t, z^t, a^t; \theta_i)\right)^2 \text{ with } y_i = r_i^t + \gamma Q(h_i^{t+1}, z^{t+1}, a^{t+1}; \theta_i)$$

$$\text{policy loss } \mathcal{L}(\phi_i) = -Q(h_i^t, z^t, a^t; \theta_i) \log \pi_i(a^t | h_i^t; \phi_i)$$

$\mathcal{L}(\theta_i)$ is value loss, $\mathcal{L}(\phi_i)$ is policy loss.

Why training action-value critic $Q(h_i, z, a; \theta_i)$ instead of centralized value function $V(h_i, z; \theta_i)$? To directly estimate the impact of the action selection on expected return. z is Centralized information.
Possibly more difficult to train, more outputs.

No conflict game: $\arg\max_i U_i(\pi) = \arg\max_j U_j(\pi) \quad \forall i, j \in \mathcal{I}$

9.5 Value Decomposition in Common-Reward Games

Value decomposition algorithms decompose optimized action-value function into simpler functions that can be learned more efficiently and enable decentralized execution. Called, "utility function", "jointly optimized".
IGM (Individual-global-max) property: States that the greedy joint actions with respect to (centralized action-value function) equal to individual actions of all agents that maximize the respective individual utilities.
Define:

$$A^*(h, z; \theta) = \arg\max_{a \in A} Q(h, z, a; \theta)$$

$$A_i^*(h_i, z; \theta_i) = \arg\max_{a_i \in A_i} Q_i(h_i, z, a_i; \theta_i)$$

IGM: $\forall a = (a_1 \cdots a_n) \in A, a \in A^*(h, z; \theta) \iff \forall i \in \mathcal{I}, a_i \in A_i^*(h_i; z; \theta_i)$
Can address multi-agent credit assignment problem.
Simple approach: $Q(h^t, z^t, a^t; \theta) = \sum_{i \in \mathcal{I}} Q_i(h^t, a^t; \theta_i)$

Algorithm 21 Value decomposition networks (VDN)

1. Initialize n utility networks $\theta_1 \cdots \theta_n$
2. Initialize n target networks $\bar{\theta}_1 = \theta_1 \cdots \bar{\theta}_n = \theta_n$

3. Initialize a shared replay buffer D
4. For time step $t = 0, 1, 2 \cdots$ do
5. Collect observations $o_1 \cdots o_n$
6. For agent $i = 1 \cdots n$ do
7. With probability ε: choose random a_i^t
8. Otherwise: choose $a_i^t \in \arg\max_{a_i} Q_i(h_i^t, a_i; \theta_i)$
9. Apply actions, collect timed reward r^t and next observations $o_1^{t+1} \cdots o_n^{t+1}$
10. Store (h^t, a^t, r^t, h^{t+1}) in D
11. Sample $B = (h^k, a^k, r^k, h^{k+1})$ from D
12. If s^{k+1} is terminal then
13. $y^k \leftarrow r^k$
14. else
15. $y^k \leftarrow r^k + \gamma \sum_{i \in \mathcal{I}} \max_{a_i'} Q_i(h_i^{k+1}, a_i'; \bar{\theta}_i)$
16. Loss $\mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{k=1}^{B} \left(y^k - \sum_{i \in \mathcal{I}} Q_i(h^k, a^k; \theta_i)\right)^2$
17. Update θ by minimizing $\mathcal{L}(\theta)$
18. In a set interval, update $\bar{\theta}_i$ for each i

Need not to be linear in practice, IGM satisfies as long as

$$\forall i \in \mathcal{I}, \forall a \in A : \frac{\partial Q(h, z, a; \theta)}{\partial Q(h, z, a_i; \theta_i)} > 0 \quad Q\text{-MAX}$$

For instance $Q(h, z, a; \theta) = \sum_{i \in \mathcal{I}} Q_i(h, z, a_i; \theta_i)$ (for $a = (a_1 \cdots a_n)$)

Sufficient condition for IGM if

$$\left(\sum_{i \in \mathcal{I}} Q_i(h, a_i; \theta_i) - \sum_{i \in \mathcal{I}} Q_i(h, a_i^{*'}; \theta_i)\right) + \left(\max_{a''} Q(h, z, a''; \theta) - Q(h, z, a; \theta)\right)$$

$$= \begin{cases} = 0 & \text{if } a = a^* \\ \geq 0 & \text{else} \end{cases}$$

where $a_i^* = \arg\max_{a_i \in A_i} Q_i(h, z, a_i; \theta_i)$

9.6 Agent Modeling with Neural Networks

Agent i model agent j, $\hat{\pi}_j^i$ by minimizing the cross-entropy loss between the predicted value $\hat{\pi}_j^i$ and true action j

$$\mathcal{L}(\phi_j^i) = -\log \hat{\pi}_j^i(a_j^t | h_j^t; \phi_j^i)$$

Each agent also trains a centralized action-value function Q parametrized by θ_i
Takes in h_i^t and a_{-i} as input, and outputs the estimated centralized action value for each action of i. Loss of this function training:

$$\mathcal{L}(\theta_i) = \frac{1}{B} \sum_{(h_i^t, a^t, r_i^t, h_i^{t+1}) \in B} \left(r_i^t + \gamma \max_{a_i' \in A_i} AV(h_i^{t+1}, a_i'; \bar{\theta}_i) - Q(h_i^t, a^t, a_i^t; \theta_i) \right)^2$$

AV = Action Value

$$AV(h_i, a_i; \theta_i) = \sum_{a_{-i} \in A_{-i}} Q(h_i, \langle a_i, a_{-i} \rangle; \theta_i) \hat{\pi}_{-i}^i(a_{-i} | h_i; \phi_{-i}^i)$$

$$= \sum_{j \neq i} Q(h_i, \langle a_i, a_{-i} \rangle; \theta_i) \prod_{j \neq i} \hat{\pi}_j^i(a_j | h_j; \phi_j^i)$$

$\rightarrow$ Actual Value
We can also estimate $AV(h_i, a_i; \theta_i)$ by

$$= \frac{1}{K} \sum_{k=1}^{K} Q(h_i, \langle a_i, a_{-i}^k \rangle; \theta_i)$$

9.8 Policy Self-Play in Zero-Sum Games

Algorithm 25 MCTS for MDPs

1. Repeat for every episode:
2. For $t = 1, 2, 3 \cdots$ do
3. Observe current state s^t
4. For k simulations do
5. $\tau \leftarrow t$
6. $s^\tau \leftarrow s^t$ *//perform simulation*
7. while s^τ is non-terminal and s^τ-node exists in tree do
8. $a^\tau \leftarrow$ Explore Actions(s^τ)
9. $s^{\tau+1} \sim T(\cdot|s^\tau, a^\tau)$
10. $r^\tau \leftarrow R(s^\tau, a^\tau, s^{\tau+1})$
11. $\tau \leftarrow \tau + 1$
12. if s^τ-node does not exist in tree then:
13. Initialize Node (s^τ): set counter $N(s^\tau, a) = 0$ and $Q(s^\tau, a) = 0$ for all $a \in A$
14. while $\tau > t$ do
15. $\tau \leftarrow \tau - 1$
16. Update (s^τ, a^τ)
17. Select action for s^t
18. $\pi^t \leftarrow$ best Action(s^t)
19. $a^t \sim \pi^t$