MARL

Monday, November 25, 2024 13:48

$u^t = r^t + \gamma r^{t+1} + \gamma^2 r^{t+2} + \cdots$

$= r^t + \gamma u^{t+1}$

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

$V^\pi(s) = E_\pi[u^t \mid s^t = s]$

$= E_\pi[r^t + \gamma u^{t+1} \mid s^t = s]$

$= \sum_{a\in A}\pi(a|s)\sum T(s'|a,s)[R(s,a,s') + \gamma E_\pi[u^{t+1}|s^{t+1}=s']]$

$= \sum_{a\in A}\pi(a|s)\sum T(s'|a,s)[R(s,a,s') + \gamma V^\pi(s')]$

This is call Bellman Equation.

Why is Bellman Equation useful?
Because we can use a system of linear equations to solve for $V^\pi(s)$.

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

Similarly, we can define action-value functions
$Q^\pi(s,a) = \sum T(s'|s,a)[R(s,a,s') + \gamma\sum\pi(a'|s')Q^\pi(s',a')]$

Iterative policy evaluation: Repeatedly updates sweeps for all state s
$V^{k+1}(s) \leftarrow \sum_{a\in A}\pi(a|s)\sum_{s'\in S}T(s'|s,a)[\underset{\text{Reward function}}{R(s,a,s')} + \underset{\text{discount factor}}{\gamma} V^k(s')]$

TD algorithms: Temporal Difference, does not rely on knowing all MDP (reward functions, transition probility e.g)

action value function updates:
$Q(s^t,a^t) \leftarrow Q(s^t,a^t) + \alpha(\mathcal{X} - Q(s^t,a^t))$
where $\mathcal{X} = r^t + \gamma Q(s^{t+1}, a^{t+1})$

Algorithm Sarsa for MDPs (with ϵ-greedy policies)

```
Initialize $Q(s,a)=0$ for all $s\in S$ and $a\in A$
Repeat for every episode:
    Observe initial state $s^0$
    With probability $\epsilon$: choose random action $a^0\in A$
    Otherwise: choose $a^0 \in \arg\max_a Q(s^0,a)$
    for $t=0,1,2\cdots$ do.
        Apply $a^t$, observe reward $r^t$ and next state $s^{t+1}$
        With probability $\epsilon$: choose random action $a^{t+1}\in A$
        Otherwise: choose $a^{t+1} \in \arg\max_a Q(s^{t+1},a)$
        $Q(s^t,a^t) \leftarrow Q(s^t,a^t) + \alpha(r^t + \gamma Q(s^{t+1},a^{t+1}) - Q(s^t,a^t))$
```

Understanding: With ϵ probility 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

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

In POSG, let $\hat{h}^t = \{(s^\tau, o^\tau, a^\tau)_0^{t-1}, s^t, o^t\}$ denote a full history up to time t. $\sigma(\hat{h}^t)$ returns the history of joint observations.
Define the probability of joint observation, $O(o^t|a^{t-1},s^t) = \prod_{i\in I} O_i(o_i^t|a^{t-1},s^t)$

- Define History-based expected return for agent i as:
  $U_i(\pi) = \sum Pr(\hat{h}^t|\pi) M_i(\hat{h}^t) \quad \hat{h}^t \in H$

where $Pr(\hat{h}^t|\pi)$ is the probability of full history $\hat{h}^t$ under π.
$Pr(\hat{h}^t|\pi) = \mu(s^0)O(o^0|\emptyset,s^0)\prod_{\tau=0}^{t-1}\pi(a^\tau|h^\tau)T(s^{\tau+1}|s^\tau,a^\tau)O(o^{\tau+1}|a^\tau,s^{\tau+1})$
and $M_i(\hat{h}^t)$ is the discounted return for agent i in $\hat{h}^t$
$M_i(\hat{h}^t) = \sum_{\tau=0}^{t-1}\gamma^\tau R_i(s^\tau,a^\tau,s^{\tau+1})$

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

Recursive Expected Return.

- Define $BR_i(\pi_{-i}) = \arg\max U_i(\pi_i, \pi_{-i})$ BR: Best Response

In a zero-sum two agents game, a joint policy $\pi = (\pi_i, \pi_j)$ is a minmax solution, if $U_i(\pi) = \max_{\pi_i}\min_{\pi_j} U_i(\pi_i,\pi_j) = \min_{\pi_j}\max_{\pi_i} U_i(\pi_i,\pi_j) = -U_j(\pi)$

- Minmax via Linear programming.
  $\min U_j^*$
  subject to $\sum_{a_i\in A_i} R_j(a_i,a_j)\pi_i(a_i) \le U_j^* \quad \forall a_j \in A_j$
  $x_{a_i} \ge 0 \quad \sum_{a_i\in A_i} x_{a_i} = 1$
- 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
  $\forall i, \pi_i': U_i(\pi_i', \pi_{-i}) \le U_i(\pi)$.
- ϵ-Nash Equilibrium: $\forall i, \pi_i': U_i(\pi_i',\pi_{-i}) - \epsilon \le U_i(\pi)$
- 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 $i\in I$ and every action modifier $\xi_i: A_i \to A_i$
  $\sum_{a\in A}\pi_c(a)R_i(\langle\xi_i(a_i), a_{-i}\rangle) \le \sum_{a\in A}\pi_c(a)R_i(a)$

  Correlated Equilibrium allows correlation between policies whereas Nash-Equilibrium doesn't.
- Linear programming for Correlated Equilibrium.
  maximize $\sum_{a\in A}\sum_{i\in I} x_a R_i(a)$ $\quad x_a = \pi(a)$ representing selecting action a under policy π.

  Subjected to $\sum_{a\in A} x_a R_i(a) \ge \sum x_a R_i(a_i', a_{-i})$?
- Pareto Optimality: A joint policy π is Pareto dominated by another policy π' if: $\forall i: U_i(\pi') \ge U_i(\pi)$ 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
- Walfare of π: $W(\pi) = \sum_{i\in I} U_i(\pi)$
- Fairness of π: $F(\pi) = \prod_{i\in I} U_i(\pi)$

If Walfare is fixed, fairness is maximum if all $U_i(\pi)$ are equal.
Walfare Optimality ⟶ Pareto Optimality, but not vice versa.

- Regret$_i$ $= \max_{a_i\in A_i}\sum_{e=1}^{z}[\underset{\substack{\uparrow \\ \text{If knows future,} \\ \text{chose again.}}}{R_i(a_i, a_{-i}^e)} - \underset{\substack{\uparrow \\ \text{history}}}{R_i(a^e)}]$

No Regret if $\forall i$: $\lim_{z\to\infty}\frac{1}{z}\text{Regret}_i^z \le 0$. Refer fig 4.6.

- Central Learning. CQ-Learning

```
Initialize $Q(s,a)=0$ for all $s\in S$ and $a\in A = A_1\times\cdots A_n$.
Repeat for every episode.
for $t=1,\cdots$ do
    Observe current state $s^t$.
    With probability $\epsilon$: choose random action $a^t\in A$
    Otherwise, choose $a^t \in \arg\max_a Q(s^t,a)$
    Apply joint action $a^t$, observe rewards $r_1^t,\cdots,r_n^t$ and $s^{t+1}$
    Transform $r_1^t \cdots r_n^t$ to $r$  this step is usually unclear for general-sum game.
    $Q(s^t,a^t) \leftarrow Q(s^t,a^t) + \alpha[r^t + \gamma\max_{a'}Q(s^{t+1},a') - Q(s^t,a^t)]$
```

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.

```
Initialize $V_i(s)=0$ for all $s\in S$ and $i\in I$
Repeat until all $V_i$ have converged
for all $s\in S$, $i\in I$, joint action $a\in A$ do
    $M_{s,i}(a) \leftarrow \sum_{s'\in S}T(s'|s,a)[R_i(s,a,s') + \gamma V_i(s')]$
for all $s\in S$ and $i\in I$ do
    $V_i(s) \leftarrow \text{Value}_i(M_{s,1},\cdots M_{s,n})$
```

$M_{s,i}(a)$ 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 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
  $Q_i^\pi(s,a) = \sum_{s'\in S}T(s'|s,a)[R_i(s,a,s') + \gamma\sum_{a'\in A}\pi(a'|s')Q_i^\pi(s',a')]$
- 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 viewed 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 $\Gamma_{s,i}(a)$.

Algorithm JAL-GT (controls agent i)

```
Initialize $Q_j(s,a)=0$ for all $s\in S$, $a\in A$
Repeat for every episode:
for $t=0,1,2\cdots$ do.
    Observe current state $s^t$
    With prob $\epsilon$: choose random action $a_i^t$
    Otherwise. Solve $\Gamma_{s^t}$ to get policies $(\pi_1,\cdots\pi_n)$, then sample action $a_i^t \sim \pi_i$
    Observe joint action $a^t = (a_1^t,\cdots a_n^t)$, rewards $r_1^t,\cdots r_n^t$, next state $s^{t+1}$
    for all $j\in I$ do
        $Q_j(s^t,a^t) \leftarrow Q_j(s^t,a^t) + \alpha[r_j^t + \gamma\Gamma_{s^{t+1},j} - Q_j(s^t,a^t)]$
```

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^τ, a_j^τ) 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)

```
Initialize:
    $Q_i(s,a)=0$ for all $s\in S$, $a\in A$
    Agent set $\hat{\pi}_j(a_j|s) = \frac{1}{|A_j|}$ for all $j\ne i$ and $s\in S$.
Repeat for every episode.
for $t=0,1,2\cdots$ do.
    Observe current state $s^t$.
    $\epsilon$-greedy explore
    Otherwise, choose BR $a_i^t \in \arg\max_{a_i} AV_i(s^t,a_i)$
    Observe joint action $a^t = (a_1^t\cdots a_n^t)$, $r_i^t$ and $s^{t+1}$
    for $j\ne i$ do:
        update $\hat{\pi}_j$ agent model
    $Q_i(s^t,a^t) \leftarrow Q_i(s^t,a^t) + \alpha[r_i^t + \gamma\max_{a_i'}AV_i(s^{t+1},a_i') - Q_i(s^t,a^t)]$
```

where $AV_i(s,a_i) = \sum_{a_{-i}\in A_{-i}} Q_i(s,\langle a_i,a_{-i}\rangle)\prod_{j\ne i}\hat{\pi}_j(a_j|s)$

- 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

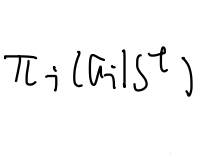
- Read 6.4 for IGA and WoLF-IGA

Algorithm Win or Learn fast with policy hill climbing. for agent i.

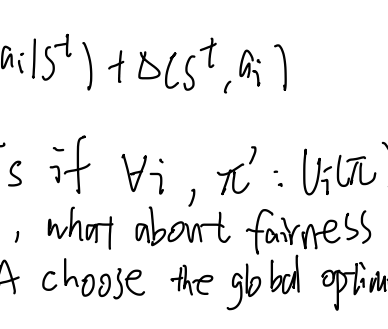
```
Initialize:
    Learning rates $\alpha\in(0,1]$ and $l_l$ and $l_w \in (0,1]$ with $l_l > l_w$
    Value function $Q(s,a_i) \leftarrow 0$ and policy $\pi_i(a_i|s) \leftarrow \frac{1}{|A_i|}$ for all $s\in S$ and $a_i\in A_i$
    State counter $C(s) \leftarrow 0$ for all $s\in S$.
    Average policy $\bar{\pi}_i \leftarrow \pi_i$
Repeat for every episode.
    for $t=0,1,2\cdots$ do
        Observe $s^t$
        $\epsilon$-greedy explore
        Otherwise: Sample action from policy. $a_i^t \sim \pi_i(\cdot|s^t)$
        Observe $r_i^t$ and $s^{t+1}$
        Update Q-value:
            $Q(s^t,a_i^t) \leftarrow Q(s^t,a_i^t) + \alpha[r_i^t + \gamma\max Q_i(s^{t+1},a_i') - Q_i(s^t,a_i^t)]$
        Update average policy $\bar{\pi}_i$:
            $C(s^t) \leftarrow C(s^t)+1$
            $\forall a_i\in A_i: \bar{\pi}_i(a_i|s^t) \leftarrow \bar{\pi}_i(a_i|s^t) + \frac{1}{C(s^t)}(\pi_i(a_i|s^t) - \bar{\pi}_i(a_i|s^t))$
        Update policy $\pi_i$:
            $\forall a_i\in A_i: \pi_i(a_i|s^t) \leftarrow \pi_i(a_i|s^t) + \Delta(s^t,a_i)$
```

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

	C	D
C	-1 -1	-5 0
D	0 -5	-3 -3



Coop



Grim.

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

Chapter 7. Neural Network
$f_k(x_{k-1};\theta^k) = g_k(W_k^T x_{k-1} + b_k)$, $W_k \in \mathbb{R}^{d_{k-1}\times d_k}$, $x_{k-1}\in\mathbb{R}^{d_{k-1}}$
$b_k\in\mathbb{R}^{d_k}$

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

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

Vanilla gradient descent is too costly,
SGD: $\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta|d) \mid d\sim\mu(D)$ d drawn from dataset D.
Mini batch: $\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta|B)$ $B = \{d_i \sim \mu(D)\}_{i=1}^{B}$

trade-off between converge fast and less variance.

Chapter 8.
$Q(s^t,a^t) \leftarrow Q(s^t,a^t) + \alpha(r^t + \gamma\max_{a'}Q(s^{t+1},a') - Q(s^t,a^t))$

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

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

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

```
Initialize value network $Q$ with random parameters $\theta$

    same for all algorithms


choose $a^t \in \arg\max_a Q(s^t,a;\theta)$
Apply action $a^t$; observe reward $r^t$ and next state $s^{t+1}$
if $s^{t+1}$ is terminal then
    Target $y^t \leftarrow r^t$
else Target $y^t \leftarrow r^t + \gamma\max_{a'}Q(s^{t+1},a';\theta)$ ← moving target problem.   (computed by bootstrapping)
Loss $L(\theta) \leftarrow (y^t - Q(s^t,a^t;\theta))^2$
Update parameters $\theta$
```

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

```
Initialize main network $Q$ with random parameters $\theta$
Initialize target network $\bar{\theta} = \theta$
    ⋮
    Same as algorithm 10
    ⋮
else: Target $y^t \leftarrow r^t + \gamma\max_{a'}Q(s^{t+1},a';\bar{\theta})$
Loss $L(\theta) \leftarrow (y^t - Q(s^t,a^t;\theta))^2$
Update $\theta$
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.

```
Initialize $Q$ with $\theta$
Target network $\bar{\theta} = \theta$
Empty replay buffer $D = \{\}$
    ⋮
    Same for all
    ⋮
Otherwise: choose $a^t \in \arg\max_a Q(s^t,a;\theta)$
Apply $a^t$ and observe $r^t$, $s^{t+1}$
Store transition $(s^t,a^t,r^t,s^{t+1})$ to $D$
Sample random mini-batch $B$ from $D$. $B = (s^k,a^k,r^k,s^{k+1})$
if $s^{k+1}$ is terminal:
    $y^k \leftarrow r^k$
else:
    $y^k \leftarrow r^k + \gamma\max_{a'}Q(s^{k+1},a';\bar{\theta})$
Loss $L(\theta) \leftarrow \frac{1}{D}\sum_{k=1}^{D}(y^k - Q(s^k,a^k;\theta))^2$
Update parameters $\theta$
In a interval, update $\bar{\theta}$
```

8.2
Why use policy gradient algorithms?
1. Able to represent probablistic equilibrium
2. Able to handle continuous action space.

To represent a probalistic policy for discrete action spaces using neural networks, the network receives a state as input and outputs a scalar $l(s,a)$ for all $a\in A$
Then put them in a Softmax function
$\pi(a|s;\phi) = \frac{e^{l(s,a)}}{\sum_{a'\in A}e^{l(s,a')}}$

- Policy gradient theorem:
  $\nabla_\phi J(\phi) \propto \sum_{s\in S}Pr(s|\pi)\sum_{a\in A}Q^\pi(s,a)\nabla_\phi\pi(a|s;\phi)$

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

  $= E_{s\sim Pr(\cdot|\pi), a\sim\pi(\cdot|s;\phi)}[Q^\pi(s,a)\nabla_\phi\log\pi(a|s;\phi)]$

  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

```
Initialize policy network $\pi$ with random parameters $\phi$
Repeat for every episode:
for time $t = 0\cdots T-1$ do:
    Observe $s^t$
    Sample action $a^t \sim \pi(\cdot|s^t;\phi)$
    Apply $a^t$ and observe $r^t$ and $s^{t+1}$
Loss $L(\phi) \leftarrow -\frac{1}{T}\sum_0^{T-1}(\sum_{\tau=t}^{T-1}\gamma^{\tau-t}r^\tau)\log\pi(a^t|s^t;\phi)$
Update $\phi$ by minimizing $L(\phi)$
```

8.2.3 baseline ???

- Actor-Critic Algorithm
  Actor-Critic is a family of algorithms that trains a parameterized policy called actor, and a value function called the critic, alongside each other, able to update the policy in a single time step rather than episode.
  Trade-off is that has higher bias.
  Single step ⟵ N-Step ⟵ Entire episode
  Actor critic MC
- Advantage Actor-Critic
  $Adv^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

  $Q^\pi(s,a)$: At state s, do action a, the follows 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

```
Initialize actor (policy) network $\pi$ with para $\phi$
Initialize critic (state-value) network $V$ with para $\theta$
Repeat for every episode
for time step $t = 0,1,2\cdots$ do
    Observe $s^t$
    Sample action $a^t \sim \pi(\cdot|s^t;\phi)$
    Apply $a^t$, observe $r^t$ and $s^{t+1}$
    if $s^{t+1}$ is terminal then
        $Adv(s^t,a^t) \leftarrow r^t - V(s^t;\theta)$
        Critic target $y^t \leftarrow r^t$
    else
        $Adv(s^t,a^t) \leftarrow r^t + \gamma V(s^{t+1};\theta) - V(s^t;\theta)$
        critic target $y^t \leftarrow r^t + \gamma V(s^{t+1};\theta)$
    actor Loss $L(\phi) \leftarrow -Adv(s^t,a^t)\log\pi(a^t|s^t;\phi)$
    Critic Loss $L(\theta) \leftarrow (y^t - V(s^t;\theta))^2$
    Update $\phi$
    Update $\theta$
```

9.1.3

- 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)

```
Initialize $n$ value networks with random $(\theta_1,\cdots\theta_n) = \theta$
Initialize $n$ target networks with $\bar{\theta} = \theta$
Initialize $n$ replay buffers $D_1,\cdots D_n$.
for time step $t = 0,1,2,\cdots$ do
    Collect observations $o_1^t\cdots o_n^t$
    for agent $i=1,\cdots n$ do
        $\epsilon$-Greedy explore
        otherwise: choose $a_i^t \in \arg\max_{a_i}Q(h_i^t,a_i;\theta_i)$
    Apply actions $(a_1^t,\cdots a_n^t)$; collect rewards $r_1^t,\cdots r_n^t$ and next observation $o_1^{t+1},\cdots o_n^{t+1}$
    for agent $=1,\cdots n$ do
        Store $(h_i^t,a_i^t,r_i^t,h_i^{t+1})$ to $D_i$
        Sample random mini-batch of $B$ transitions $(h_i^k,a_i^k,r_i^k,h_i^{k+1})$ from $D_i$
        if $s^{k+1}$ is terminal then
            Target $y_i^k \leftarrow r_i^k$
        else
            $y_i^k \leftarrow r_i^k + \gamma\max_{a_i'\in A_i}Q(h_i^{k+1},a_i';\bar{\theta}_i)$

        Loss $L(\theta_i) \leftarrow \frac{1}{B}\sum_{k=1}^{B}(y_i^k - Q(h_i^k,a_i^k;\theta_i))^2$
        Update $\theta_i$ by minimizing $L(\theta_i)$
        In a interval, update $\bar{\theta}_i$
```

- Multi-agent policy gradient theorem.

$\nabla_{\phi_i}J(\phi_i) \propto E_{\hat{h}\sim Pr(\hat{h}|\pi), a_i\sim\pi_i, a_{-i}\sim\pi_{-i}}[Q_i^\pi(\hat{h},\langle a_i,a_{-i}\rangle)\nabla_{\phi_i}\log\pi_i(a_i|h_i = \sigma_i(\hat{h});\phi_i)]$

- 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 centralized critic, becomes
  $L(\theta_i) = (y_i - V(h_i^t,z^t;\theta_i))^2$ with $y_i = r_i^t + \gamma V(h_i^{t+1},z^{t+1};\theta_i)$

9.4.3
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's conditioned on individual observation history, centralized information, and actions of all agents.

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

policy loss. $L(\phi_i) = -Q(h_i^t,z^t,a^t;\theta_i)\log\pi(a_i^t|h_i^t;\phi_i)$

$L(\theta_i)$ is value loss, $L(\phi_i)$ 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 returns. z is centralized information.

  Possibly more difficult to train, more outputs.

- No conflict game: $\arg\max U_i(\pi) = \arg\max U_j(\pi) \quad \forall i,j\in I$.
- 9.5

Value decomposition algorithms decompose centralized 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 should be equal to individual actions of all agents that maximize the respective individual utilities

Define: $A^*(h,z;\theta) = \arg\max_{a\in A} Q(h,z,a;\theta)$
$A_i^*(h_i;\theta_i) = \arg\max_{a_i\in A_i} Q(h_i,a_i;\theta_i)$

IGM: $\forall a = (a_1\cdots a_n)\in A: a\in A^*(h,z;\theta) \Longleftrightarrow \forall i\in I: a_i \in A_i^*(h_i;\theta_i)$

can adress multi-agent credit assignment problem

Simple approach: $Q(h^t,z^t,a^t;\theta) = \sum_{i\in I}Q(h_i^t,a_i^t;\theta_i)$

Algorithm 21 Value decomposition networks (VDN)

```
Initialize $n$ utility networks $\theta_1\cdots\theta_n$
Initialize $n$ target networks $\bar{\theta}_1 = \theta_1\cdots\bar{\theta}_n = \theta_n$
Initialize a shared replay buffer $D$.
for time step $t = 0,1,2\cdots$ do
    Collect observations $o_1\cdots o_n$
    for agent $i=1,\cdots n$ do
        With probility $\epsilon$: choose random
        Otherwise: choose $a_i^t \in \arg\max_{a_i} Q(h_i^t,a_i;\theta_i)$
    Apply actions. Collect shared reward $r^t$ and next observations $o_1^{t+1},\cdots o_n^{t+1}$
    Store $(h^t,a^t,r^t,h^{t+1})$ in $D$
    Sample $B$ $(h^k,a^k,r^k,h^{k+1})$ from $D$
    if $s^{k+1}$ is terminal then:
        $y^k = r^k$
    else:
        $y^k = r^k + \gamma\sum_{i\in I}\max_{a_i'\in A_i}Q(h_i^{k+1},a_i';\bar{\theta}_i)$
    Loss $L(\theta) \leftarrow \frac{1}{B}\sum_{k=1}^{B}(y^k - \sum_{i\in I}Q(h_i^k,a_i^k;\theta_i))^2$
    Update $\theta$ by minimize $L(\theta)$
    In a set interval, update $\bar{\theta}_i$ for each $i$.
```

- Need not to be linear in practice, IGM satifies as long as
  $\forall i\in I, \forall a\in A: \frac{\partial Q(h,z,a;\theta)}{\partial Q(h_i,a_i;\theta_i)} > 0$ QMIX
  for instance $Q(h,z,a;\theta) = \sum_{i\in I}\alpha_i(h)Q(h_i,a_i;\theta_i)$ for $\alpha_i(h) > 0$

Sufficient Condition for IGM if

$(\sum Q(h_i,a_i;\theta_i) - \sum Q(h_i,a_i^*;\theta_i)) + (\max_{a\in A}Q(h,z,a;\theta^q) - Q(h,z,a;\theta^q))$
$= \begin{cases} 0 & \text{if } a = a^* \\ \ge 0 & \text{else} \end{cases}$ $\quad a_i^* = \arg\max_{a_i\in A_i} Q(h_i,a_i;\theta_i)$

- 9.6 Agent Modeling
  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
  $L(\phi_j^i) = -\log\hat{\pi}_j^i(a_j^t|h_i^t;\phi_j^i)$

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:

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

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

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

⟶ 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 Self-play in Zero-Sum Games.

Algorithm 25 MCTS for MDPs

```
Repeat for every episode:
for $t=1,2,3\cdots$ do
    Observe current state $s^t$
    for $k$ simulations do
        $\tau \leftarrow t$
        $\hat{s}^\tau \leftarrow s^t$            // perform simulation.
        while $\hat{s}^\tau$ is non-terminal and $\hat{s}^\tau$-node exists in tree do
            $\hat{a}^\tau \leftarrow$ ExploreActions$(\hat{s}^\tau)$
            $\hat{s}^{\tau+1} \sim T(\cdot|\hat{s}^\tau,\hat{a}^\tau)$
            $\hat{r}^\tau \leftarrow R(\hat{s}^\tau,\hat{a}^\tau,\hat{s}^{\tau+1})$
            $\tau \leftarrow \tau+1$
        if $\hat{s}^\tau$-node does not exist in tree then:
            InitializeNode$(\hat{s}^\tau)$  ↘ set counter $N(\hat{s}^\tau,\hat{a})=0$ and $Q(\hat{s}^\tau,\hat{a})=0$ for all $\hat{a}\in A$
        while $\tau > t$ do
            $\tau \leftarrow \tau-1$
            Update$(Q,\hat{s}^\tau,\hat{a}^\tau)$
    Select $a^t$ for $s^t$:
        $\pi^t \leftarrow$ BestAction$(s^t)$
        $a^t \sim \pi^t$.
```