MARL

Lectures 6

15-03-22

Multi-Agent-~~Linear-Regression~~-Reinforcement-Learning is used to model MAS's, which means Multi-Agent Systems. MAS are very complex, highly dynamic, non-deterministic; mostly because agents need to cooperate and agents influence each others worlds.

Reinforcement Learning

Given

Set of states S
Set of actions A
(Sub)Tranition function $\delta: S \times A \to S$
(Sub)Reward function $r: S \times A \to R$

S, A, δ, R, γ form a Markov Decision Process

Find

Policy π

such that $V^{\pi}(s_t) \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is maximized

(V^{π}: value; s_t: start; γ: discount factor (given); r_{t+i}: reward)

Which is updated as:

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} \delta(s, \pi(s), s') V^{\pi}(s')$$

intxt: value of state is immediate reward + discountfactor times the weighted sum of all ~~per~~ possible states we can end up in their value times prob. of ending up there.

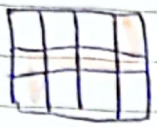
Calculating optimal policy

There are 3 ways to tackle this

- ~~Policy Iteration~~
  - ~~For all states keep using the update function until convergence is reached, then extract optimal policy. Convergence is determined by the bellman error.~~
- ~~Value Iteration~~
  - ~~Uses a different update function where instead of prob x value sum, we simply take the max possible answer.~~

- ~~Partial Evaluation~~
  - ~~Instead of sum~~

bellman error: Δ for convergence
full backup: updating states once based on current V

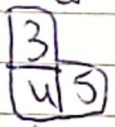


- Policy Iteration **guaranteed convergence**
  1. Choose random $v(s)$ $\forall s \in S$, random π
  2. Update until convergence based on action by policy chosen
  3. ~~If~~ get greedy policy from states
  4. ~~If~~ repeat from 2 till policy converges

- Partial Evaluation
  1. Choose random $v(s)$ $\forall s \in S$. random π
  2. Update states once (full backup)
  3. get greedy policy from states
  4. repeat from 2 till π converges

- Value Iteration **guaranteed convergence**
  1. Choose random
  2. update once based on max neighbour value → so not based on policy
  3. ~~get greedy π~~
  4. repeat from ~~2~~ till ~~π~~ V converges

Q-learning



But what if we don't know the transition function between states? (Next step in puddle with 0.4 and 0.2 if dry not known) We use Q-values. We will "walk" through the states and observe what values we'll get. This, not knowing, is model-free. Q-learning is also off-policy, so exploration independent. Update rule: ↳ don't learn during policy.

- $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \max(Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

α: learning param
r_t: reward
γ: discount

Execute a at state s, observe s' and reward r,
update Q value: current + α(reward + γ · max next) − current

Q-learning is off-policy because it does not always follow the optimal policy (ε-greedy)

Action choosing Q-learning

Various options, including:

- Epsilon Greedy:
  - ↳ roll die if value ≤ exploration rate ε, random action, else action with max Q. Note that ε can be set to decay over time.
- Boltzmann exploration:
  - ↳ select action with probability.

$$P(a|s) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$$

with T decaying exponentially. With a high T, the value of Q(s,a) is not very important (cause both very big, however with the temperature low, exploitation >>> exploration. And so each action is taken with probability p.

MARL

Cooperative System Rewards

If agents are working together, how to give rewards?

- Full System Reward (general reward to all agents)
- Local Reward (individual reward)
- Difference Reward (each agent info about their impact)

(increasing ↓)

One of the main problems is, which agents' their actions led to this award? Credit Assignment Problem.

Competing Agents

Well, here the reward is for a single agent, easy, but we will want to base our actions on mixed policies, based on mine and the agents actions. Where will we end up?

- Nash Equilibrium: everyone has best response based on others
- Pareto optimality: globally best outcome, highest reward

Goal:

- Converge (behave predictably)
- Converge to best response (behave rationally)

Some Game Lingo

Symmetric game: one agents payout (↗ dif person same policy should give same) and actions are exactly the same as others

Zero-Sum game: one agents gain is exactly the others loss.

Mini-Max Q-learning

Let x describe policies of player 1, $[x_1, x_2, \ldots x_n]$ where x_1, x_2.. are probability distributions, and let y be the same for player 2. Let A denote the set of all actions for player 1 (I think).

$$\max_x \min_y (y^T A) x$$ → estimate with Q-values

So player 2 will minimize the payout for p1 by chosing policy y_i, where x will then maximize this by chosing policy x_i.

In practise this means these steps:

1. init policies with uniform probabilities and set (comm) Q values ~~#~~ to 1, V to 1.
2. Choose action (ε-greedy, boltzmann etc)
3. learn reward, update Q-value
4. Use LP to find max min Q-value

Basically $Q_n = Q + rew + \gamma V(s')$ (next state)
with value next state is min(~~$\pi \cdot Q$~~ by y, over ~~the sum of each~~ all of a's ~~actions with~~ their sum of $\pi \times Q_s$

With properties

- for competitive zero-sum games
- LP makes it slow
- Assumes opponent minimizes always (best worst opp)
- If opp not minimizing, can't exploit that

Independent learner: Q-learner ignoring others (treated as noise) in MARL. Convergence guarantees are lost.

question: NashQ for both competitive & cooperation

Look at AlphaGo

question: why $\frac{1}{|A|-1} - \pi_{a_l}$?

question: wtf happened when filling in cross learning

Nash Q-learning **no guaranteed converge (multiple NE)**

For general sum games, both competitive as cooperative. We're going to define optimal Q-values as the ones received in a Nash Equilibrium, thus creating NashQ-values. For this we maintain our and their Q-values. A NashQ is going to be expected sum of discounted (γ) reward when all agents follow their NE policies from next timestep onwards.

$$Q_j^{t+1}(s, \underbrace{a_1, \dots a_m}_{\text{joint actions}}) = (1-\alpha_t) Q_j^t(s, a_1, \dots, a_m) + \alpha_t [r_j^t + \gamma \, \text{NashQ}_j^t(s')]$$

With $\text{NashQ}_j^t(s')$ is the estimated value for agent j from state s' onward when all agents play the Nash Equilibrium based on the current Q-values. So compute NE $\pi_1, \pi_2, \pi_3, \dots$ using $Q_j^t(s', a_1, \dots, a_m)$ as payoffs

↑ j refers to specific agents Q-value that state.

Learning Automata

Each agent has a policy vector with for every possible action the preference, so probability of choosing.

$$\pi_a \leftarrow \pi_a + \begin{cases} \underbrace{\alpha r_{t+1}(1-\pi_a)}_{\text{reward} \times \text{inverse prob}} - \underbrace{\beta(1-r_{t+1})\pi_a}_{\text{penalty} \times \text{prob}} & \text{if } a = a_t \\ \underbrace{-\alpha r_{t+1}\pi_a}_{\text{substract gotten reward}} + \underbrace{\beta(1-r_{t+1})\left(\frac{1}{|A|-1} - \pi_a\right)}_{\text{add penalty}\times} & \text{else} \end{cases}$$

So α = reward β = penalty

$\alpha = \beta$ means linear reward penalty

$\beta = 0$ li linear reward in-action

$\alpha \gg \beta$ means linear reward ε-penalty (so very small penalty)

if $\alpha = 1$ $\beta = 0$ we have cross-learning.

If you make this about states, you just give each state a learning automata and use the average reward obtained since last update.

Cross Learning

It's learning automata but then $\alpha = 1$ $\beta = 0$

this gives:

$$\pi_a \leftarrow \pi_a + \begin{cases} r_{t+1}(1 - \pi_a) & \text{if } a = a_t \\ -r_{t+1}(\pi_a) & \text{else} \end{cases}$$

which becomes

$$x_i(t+1) \leftarrow x_i(t) + \begin{cases} R_j - R_j x_i(t) \\ -R_j x_i(t) \end{cases}$$

Then we minimize the loss in a new approach:

$l_j = R^* - R_j$ best reward - current reward (normalized)

$$w_i(t+1) \leftarrow \begin{cases} w_i(t)(1 - \alpha l_i(t)) \\ w_i(t) \end{cases}$$

then

$$x_i(t) = \frac{w_i(t)}{\sum_j w_j(t)}$$

x_i is the normalized weight.

Joint Action Learning

Action-Selection-count is $E(\pi^j(a)) = \frac{C_a^j}{\sum_{b \in A^j} C_b^j}$

so that is percentage of that action being chosen.

Then expected value of a joint action:

$$E[V(a^i)] = \sum_{a' \in A'} Q(a') \prod_{j \neq i} E[\pi^j(a^j)]$$

- $E[V(a^i)]$: expected value of doing action i as agent
- $\sum_{a' \in A'}$: all a''s that include a^i
- $\prod_{j \neq i} E[\pi^j(a^j)]$: all probabilities of doing this joint action by other agent

State vs Stateless

State has delayed rewards

Stateless only immediate rewards.

Friend-or-Foe Q-learning

- Cooperative or zero-sum
- Much stronger convergence than Nash-Q
- The algorithm uses minmax for competitive and cooperative it uses the max Q for both agents combined.

(Idealized) Cross learning

We have

$$\Delta x(t) = x(t+1) - x(t)$$

Then we want to know $E[\Delta x_i]$

$$E[\Delta x_i] = x_i[f_i(1-x_i)] + \sum_{j \neq i} x_j[-f_j x_i]$$

(x_i: update this action (see cross); x_j: other actions (see cross))

$f_i = E(R_i)$

expected value so times probability of this happening (if chances to 0 of happening it might not change at all)

$$= x_i[f_i - \sum_j x_j f_j]$$

↑ factor out x_i, plug $f_i x_i$ in second part.

Now if we want to take 'infinitesimal steps we look at vectors:

$e_i R y^T$ = f_i for row player with chosen action i

$x C e_j^T$ = f_j for a col player playing action j

(x: prob of x's for each action; C: col; e_j^T: chosen action)

Now if we fill that in:

$$\frac{\delta x_i}{\delta t} = x_i[e_i R y^T - x R y^T]$$

this was $\sum x_j e_j R y^T$ but e_j is chosen action each time, cause we're summing it's just the totality of x, in this case its a sparse vector with one 1 so basically x times Identity.