Paper Reading Report

Kai Yan

EECS, PKU

Aug 2nd, 2019

Contents

Contribution of This Paper

- Introduce recurrency in actor-critic model, significantly improve the ability of solving time-dependent partial-observable **cooperative** tasks
- No regression from original MADDPG (neither strong assumptions, nor performance drop)
- Adapting to any amount of communication

This paper empirically proves that recurrent architecture for **critic** is essential (for actor it is of little use)

Why Recurrent ?

- The authors : Recurrent serves as an explicit method for agents to "remember" what they have heard if the communication budget is limited.
- (Also : Recurrency serves as an important role for time-dependent opponent-awareness. E.g. keep under surveillance, or any scenario that requires collections of observation.)

Actor Recurrent Network

- An actor's replay buffer D contains experiences, where an experience at time t contains (the last four is optional according to the architecture selected) :

$$(o_{i,t}, a_{i,t}, o'_{i,t+1}, r_{i,t}, h^p_{i,t}, h^p_{i,t+1})$$

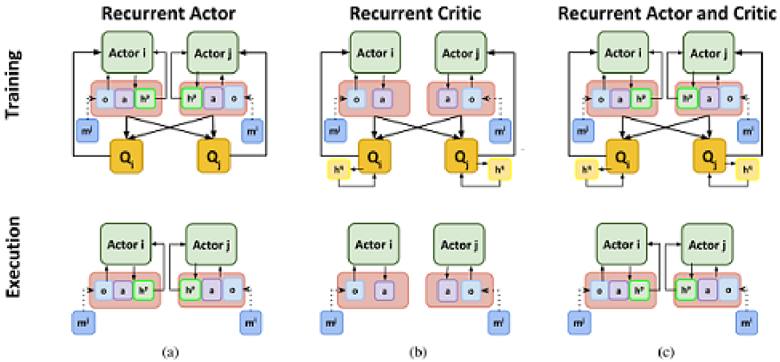
- $h^p_{i,t}$ is **the hidden state** of the agent i's critic network before selection, $h^p_{i,t+1}$ is that after selection.
- Let $\mu = \mu_{\theta_i}$ be the (continuous) policy of agent i, μ' be the target policy
- Policy gradient :

$$\nabla_{\theta_i} J(\mu) = E_{Uni(D)}[\nabla_{\theta_i}\mu(a_{i,t}|o_{i,t}, h^p_{i,t})\nabla_{a_{i,t}} Q^\mu_i(x, a)|_{a_{i,t}=\mu_i(o_{i,t},h^p_{i,t})}]$$

- Action-value function Q^μ_i :

$$L(\theta_i) = E_{Uni(D)}[((r_i + \gamma Q^{\mu'}_i(x', a'_j)|_{a'_j=\mu'_j(o_j,h^p_{j,t})} - Q^\mu_i(x, a))^2]$$

Architecture



Critic Recurrent Network

- An actor's replay buffer D contains experiences, where an experience at time t contains (the last four is optional according to the architecture selected) :

$$(o_{i,t}, a_{i,t}, o'_{i,t+1}, r_{i,t}, h^q_{i,t}, h^q_{i,t+1})$$

- $h^q_{i,t}$ is **the hidden state** of the agent i's actor network before selection, $h^q_{i,t+1}$ is that after selection.
- Let $\mu = \mu_{\theta_i}$ be the (continuous) policy of agent i, μ' be the target policy
- Policy gradient :

$$\nabla_{\theta_i} J(\mu) = E_{Uni(D)}[\nabla_{\theta_i} \mu(a_{i,t}|o_{i,t}) \nabla_{a_{i,t}} Q_i^{\mu}(x, a, h^q_t)|_{a_{i,t}=\mu_i(o_{i,t})}]$$

- Action-value function Q_i^{μ} :

$$L(\theta_i) = E_{Uni(D)}[((r_i + \gamma Q_i^{\mu'}(x', a'_j, h^q_{t+1})|_{a'_j=\mu'_j(o_j)} - Q_i^{\mu}(x, a, h^q_t))^2]$$

Actor-Critic Recurrent Network

- An actor's replay buffer D contains experiences, where an experience at time t contains (the last four is optional according to the architecture selected) :

$$(o_{i,t}, a_{i,t}, o'_{i,t+1}, r_{i,t}, h^p_{i,t}, h^p_{i,t+1}, h^q_{i,t}, h^q_{i,t+1})$$

- Let $\mu = \mu_{\theta_i}$ be the (continuous) policy of agent i, μ' be the target policy
- Policy gradient :

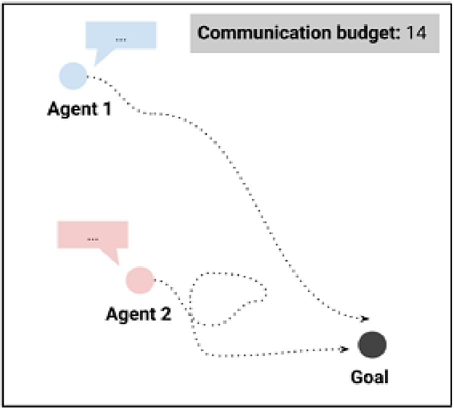
$$\nabla_{\theta_i} J(\mu) = E_{Uni(D)}[\nabla_{\theta_i}\mu(a_{i,t}|o_{i,t}, h^p_{i,t})\nabla_{a_{i,t}} Q^\mu_i(x, a, h^q_t)|_{a_{i,t}=\mu_i(o_{i,t},h^p_{i,}$$

- Action-value function Q^μ_i :

$$L(\theta_i) = E_{Uni(D)}[((r_i+\gamma Q^{\mu'}_i(x', a'_j, h^q_{t+1})|_{a'_j=\mu'_j(o_j,h^p_j)} - Q^\mu_i(x, a, h^q_t))^2$$

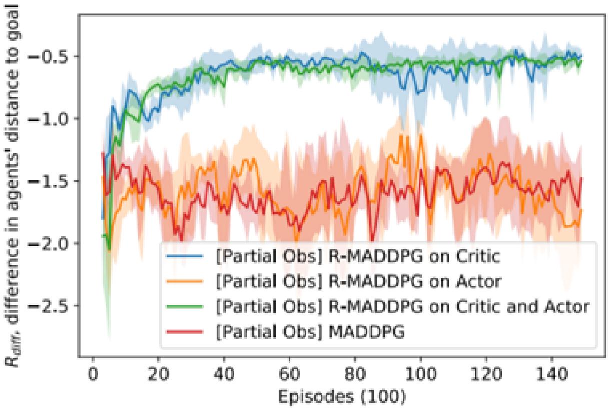
Experiment : Simultaneous Arrival

- An interesting fact is that critic with recurrency performs much better while actor with recurrency almost has no improvement.



Experiment : Simultaneous Arrival

- An interesting fact is that critic with recurrency performs much better while actor with recurrency almost has no improvement.



Normal form games

- n player select an action (usually) simultaneously, and get payoff (reward) according to a payoff matrix.

Prisoner's dilemma payoff matrix

B / A	B stays silent	B betrays
A stays silent	-1 -1	0 -3
A betrays	-3 0	-2 -2

Fictitious Play

- A statistic-based algorithm to approximate a Nash-equilibrium of a normal game
  1. Arbitrarily take a strategy as the opponent's expected strategy
  2. Play the best response of that expected strategy
  3. Update the opponent's strategy using $P(a_i) = \frac{n(a_i)}{\sum_{a_j} n(a_j)}$
  4. GOTO 2 until convergence
- e.g. Rocks, scissors and papers

(Imperfect Information) Extensive Form Games

- Players take actions in turn on a tree, going from root to leaves.
- Decision is often relying on opponent's action
- Sometimes, a set of states cannot be distinguished by a player. These states belong to one **information set** of a player. The following example shows that player 1 cannot tell if player 2 plays A or B in step 2.

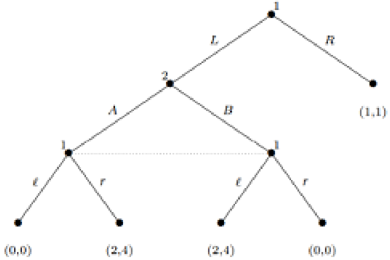


Figure 5.10: An imperfect-information game.

Kuhn's Theorem

- **mixed strategy** : convex combinations of pure strategy
- **behavioral strategy** : strategy based on opponent's behavior
- It is proved that if agents all remember everything they have done in a game (i.e. perfect recall), then for every mixed strategy, there is a equivalent behavioral strategy with equal payoff.
- That is, fictitious play can be used on extensive form games to get Nash equilibrium.

Fictitious Play for Extensive Form Play

Algorithm 1 Full-width extensive-form fictitious play

```
function FictitiousPlay(Γ)
    Initialize π_1 arbitrarily
    j ← 1
    while within computational budget do
        β_{j+1} ← ComputeBRs(π_j)
        π_{j+1} ← UpdateAvgStrategies(π_j, β_{j+1})
        j ← j + 1
    end while
    return π_j
end function

function ComputeBRs(π)
    Recursively parse the game's state tree to compute a
    best response strategy profile, β ∈ b(π).
    return β
end function

function UpdateAvgStrategies(π_j, β_{j+1})
    Compute an updated strategy profile π_{j+1} according
    to Theorem 7.
    return π_{j+1}
end function
```

Problems of This Algorithm ?

- Given π, how to compute best response strategy ?
  Reinforcement learning ! (Opponent's average policy is fixed in training)
- Given trajectories of self plays, how to model current opponent's policy ?
  Behavioral cloning with loss $E_{(s,a)\sim M_{SL}}[-\log \Pi(s,a|\pi^{\Pi})]$
  (p.s. IRL is not recommended for its underdefineness and lots of assumptions)

Problems of This Algorithm ?

- How to mix two policies (mix a behavioral policy) ?

$$\sigma(s,a) \propto \lambda_1 x_{\pi_1}(s)\pi_1(s,a) + \lambda_2 x_{\pi_2}(s)\pi_2(s,a) \forall s,a,$$

  where $x_{\pi_1}(s)$ and $x_{\pi_2}(s)$ is the probability of reaching this state. (That is, mix them at every node/state of the game tree)
- Note that the convex combination of **policy** levies the burden to combine **possible rollouts**(by translating extensive form game to normal form game with brute force), which is exponential.

Neural Fictitious Self Play

Algorithm 1 Neural Fictitious Self-Play (NFSP) with fitted Q-learning

Initialize game Γ and execute an agent via RUNAGENT for each player in the game
function RUNAGENT(Γ)
  Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
  Initialize average-policy network $\Pi(s, a \mid \theta^{\Pi})$ with random parameters θ^{Π}
  Initialize action-value network $Q(s, a \mid \theta^{Q})$ with random parameters θ^{Q}
  Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
  Initialize anticipatory parameter η
  for each episode **do**
    Set policy $\sigma \leftarrow \begin{cases} \epsilon\text{-greedy}\,(Q)\,, & \text{with probability } \eta \\ \Pi, & \text{with probability } 1-\eta \end{cases}$
    Observe initial information state s_1 and reward r_1
    for $t = 1, T$ **do**
      Sample action a_t from policy σ
      Execute action a_t in game and observe reward r_{t+1} and next information state s_{t+1}
      Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $\mathcal{M}_{RL}$
      if agent follows best response policy $\sigma = \epsilon$-greedy (Q) **then**
        Store behaviour tuple (s_t, a_t) in supervised learning memory $\mathcal{M}_{SL}$
      end if
      Update θ^{Π} with stochastic gradient descent on loss
        $\mathcal{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log \Pi(s, a \mid \theta^{\Pi})\right]$
      Update θ^{Q} with stochastic gradient descent on loss
        $$\mathcal{L}\left(\theta^{Q}\right) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s', a' \mid \theta^{Q'}) - Q(s, a \mid \theta^{Q})\right)^2\right]$$
      Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
    end for
  end for
end function

Monte-Carlo Neural Fictitious Self Play : For Scalability

- Use MCTS to approximate best-response strategy instead of DQN. Agent choose action to maximize $U(s, a)$

$$U(s,a) = Q(s,a) + cP(s,a)\frac{\sqrt{N(s)}}{1 + N(s,a)}$$

  where Q is the expect payoff value, N is a visit counter, and P is the (approximated) best response policy.
- $Q(s, a)$ is updated as

$$Q(s,a) = \frac{Q(s,a) * N(s,a) + v(s)}{N(s,a) + 1}$$

- $v(s)$ is evaluated using a policy-value network(which calculates both $v(s)$ and $P(s, a)$)

Loss function

- The loss function of the policy-value network is

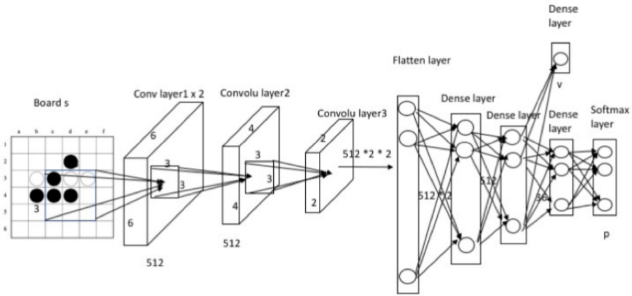
$$l_1 = -\sum_t (\pi_t \log p_t - (v(s_t) - z_t)^2)$$

which is a combination of KL-divergence of policy and MSELoss of value function (t stands for timesteps)

- The loss function of the average policy network is

$$l_2 = -\sum_t \pi_t \log p_t$$

Architecture



Algorithm

Algorithm 1 MC-NFSP algorithm

InitializeΓ,execute function $InitGame()$, $RunAgent(\Pi, B)$;
function InitGame()
    Initialize policy-value network $B(s|\theta^B)$ randomly
    Initialize policy network $\Pi(s|\Theta^\Pi)$ randomly
    Initialize experience replay M_{RL} and M_{SL}
    (players share networks B and Π)
end function
function RunAgent()
    for each iteration **do**
        its := its + 1
        policy $\sigma \leftarrow \begin{cases} B, & \textit{with probability } \eta \\ \pi, & \textit{with probability } 1-\eta \end{cases}$
        observe initial state s and reward r
        while not terminal **do**:
            If policy comes from π,choose action a in state s_t according to π
            If policy comes from B, choose action according to adapted MCTS
            Execute action a, observe next state s_{t+1}
            if *terminal* **then**:
                store $(s_t, \boldsymbol{\pi_t}, Z_t)$ in M_{RL}, store $(s_t, \boldsymbol{\pi_t})$ in M_{SL} if policy comes from B
            end if
        end while
        if $its\%update == 0$ **then**:
            update best response network with $l = -\sum_t \left(\boldsymbol{\pi_t} \log p_t - (\mathrm{v}(s_t) - z_t)^2\right)$
            update average network with $l = -\sum_t \boldsymbol{\pi_t} \log \boldsymbol{p_t}$
        end if
    end for
end function

Asynchronous NFSP

Algorithm 2 Asynchronous-Neural-Fictitious-Self-Play

InitGame(), Init game Γ, execute multiple thread *RunAgent*()
function InitGame()
  Init average strategy network $\Pi(s,a|\theta^{\Pi})$
  Init Q-value network $Q(s,a|\theta^{Q})$
  Init target network $\theta^{Q'} \leftarrow \theta^{Q}$
  Init global anticipatory parameter η
  Init global count T = 0
  Init global iteration count iterations = 0
  return
end function
function RunAgent()
  Init thread count $t \leftarrow 0$
  repeatFor each iteration
    $\text{policy}\sigma \leftarrow \begin{cases} \epsilon - greedy(Q), & with\ probability\ \eta \\ \Pi, & with\ probability\ 1-\eta \end{cases}$
    observe state s and reward r
    determine action a, observe reward r_{t+1}, next state s_{t+1}
    $y = \begin{cases} r \\ r + \gamma max_{a'} Q\left(s', a'; \theta^{-}\right), & if\ s_{t+1}\ is\ not\ terminal \end{cases}$
    accumulate gradient $d\theta^{Q} \leftarrow d\theta^{Q} + \frac{\partial\left(y - Q\left(s,a;\theta^{Q}\right)\right)^2}{d\theta^{Q}}$
    If policy σ comes from $\epsilon - greedy(Q)$, store pair (s_t, a_t) in
    $s_t \leftarrow s_{t+1}$
    $T \leftarrow T+1$
    $t \leftarrow t+1$
    if $T\ mod\ I_{target} == 0$ **then**
      update target network $\theta^{Q'} \leftarrow \theta^{Q}$
    end if
    if s is terminal **then**
      iterations += 1
      if iterations $mod\ I_{\text{Asyncupdate}}$ == 0 **then**
        update θ^{Q} with $d\theta^{Q}$ asynchronously
        update $\theta\Pi$ with $L\left(\theta^{\Pi}\right) = \mathrm{E}_{(s,a)\sim M_{SL}}\left[-\log \Pi(s,a|\theta^{\Pi})\right]$
        $d\theta^{Q} \leftarrow 0, d\theta^{\Pi} \leftarrow 0$
      end if
    end if
  until $T > T_{\max}$
  return
end function