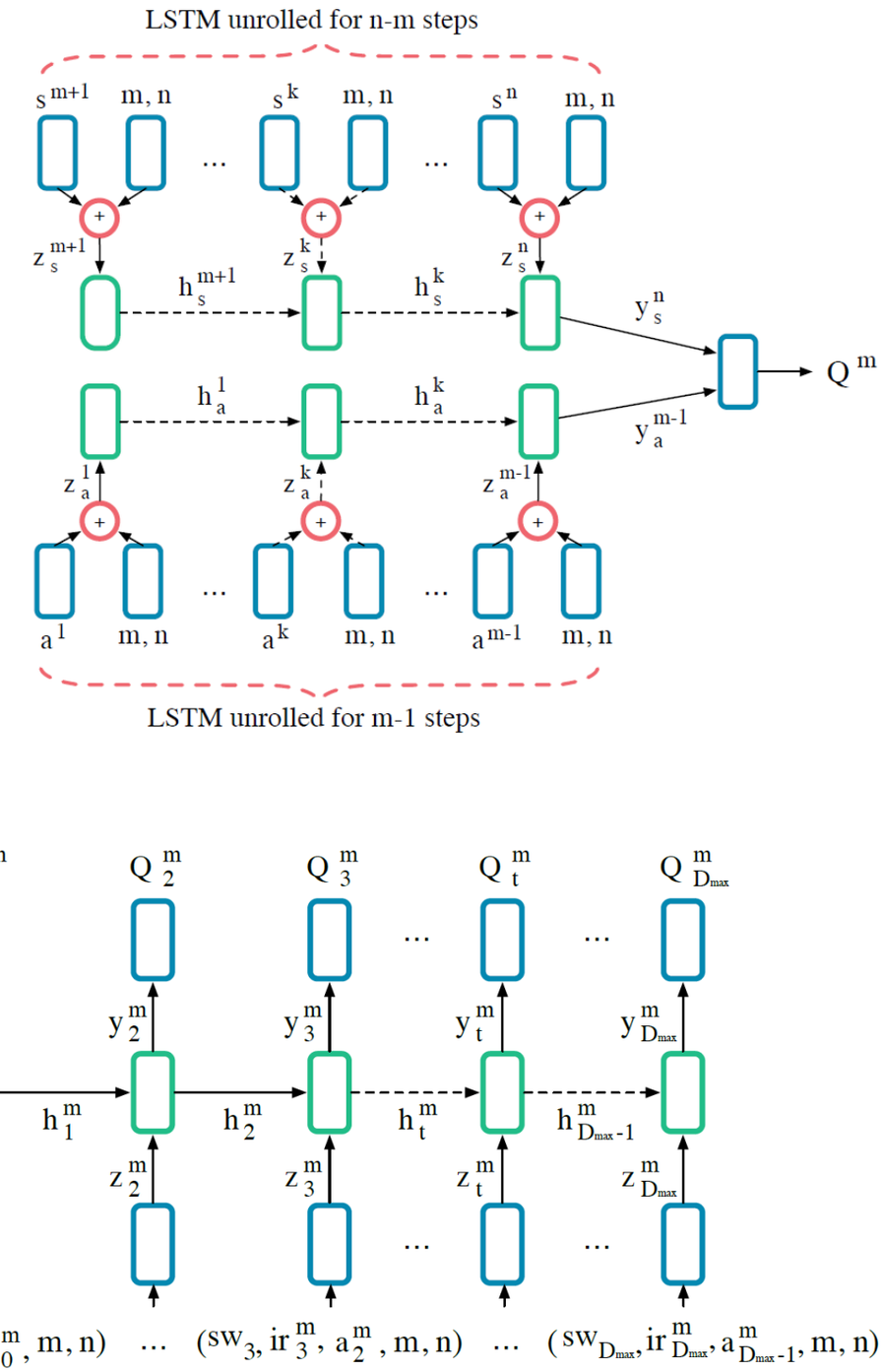
Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks

김예찬(Paul Kim)

Index



Abstract

에이전트들로 구성된 팀이 Communication을 기반으로 합동 작업을 해결할 수 있는

DDRQN(Deep Distributed Recurrent Q-Network)

을 제시함. task에서는 사전에 설계된 communication protocol을 사용하지 않음

성공적인 communication을 위해 에이전트들은 먼저 자기 자신들의 communication protocol을 자동으로 개발해야 함

이미 잘 알려진 riddle(수수께끼)에 기반한 두 가지의 다중 에이전트 학습 문제에 대한 결과를 제시하고 DDRQN으로 성공적으로 문제를 해결했으며, 학습으로 만들어진 communication protocol을 발견했다는 것을 강조(당시를 기준으로 DRL이 communication protocol을 학습한 최초의 사례)

그리고 마지막으로 DDRQN의 아키텍처의 각 주요 구성요소가 성공에 중요하다는 것을 실험 결과를 제시함

Introduction

(그 당시를 기준으로!!) DRL의 발전이 고차원 로봇제어나 Visual Attention 그리고 ALE를 포함한 여러 가지 RL문제를 해결하는데 도움이 되었음. 그렇지만 대부분 single learning agent에 국한된 경우가 대부분임.

Competitive & Cooperative

Competitive설정에서 **바둑(AlphaGo)**에 대한 DRL은 성공적인 결과물을 보여주었고

Cooperative설정에서 Tampu(리뷰한 이전 논문)는 DQN을 변형하여 두 개의 Player가 ALE환경에서 멀티 에이전트 setting이 가능함을 보여줌.

Tampu의 접근방식은 Independent Q-Learning에 기초하며 에이전트들은 각자의 Q-function을 기반으로 독립적으로 학습하는 방식임. 그렇지만 이러한 접근방식은 모든 에이전트가 환경의 상태를 **fully observe**한다는 것을 가정함

반면에 한 편으로 DQN방식으로 **partial observable**한 경우를 해결하기 위해 연구들이 진행되었으나 Single Agent에 국한된 연구가 많았다고 함

=> so 논문의 저자들은 **partial observable**하며
다중 에이전트인 경우를 상정하고 이를 해결하고자 함

Introduction

고려사항

그래서 앞서 언급한 문제를 해결하기 위해 3가지를 고려하여 DDRQN이라는 방법으로 제안

1. last action input

각각의 에이전트에게 다음 step의 input으로 이전 시점의 action을 제공함

2. inter-agent weight sharing

모든 에이전트는 단일 네트워크의 가중치를 사용하지만 에이전트의 고유 ID를 네트워크의 조건으로 사용함.
이는 결과적으로 빠른 학습을 가능하게 한다고 함

3. disabling experience replay

여러 에이전트들이 동시에 학습하는 경우에는 non-stationary하기 때문에 experience replay를
사용하는 것은 적합하지 않다는 것을 이야기 함

Introduction

DDRQN실험

DDRQN을 실험하기 위해 잘 알려진 두 가지의 수수께끼 문제를 다중 에이전트 강화학습으로 해결함

1. **Hats Riddle** : 한 줄에 있는 여러 죄수들이 자신의 모자 색이 무엇인지 맞추는 문제

2. **Switch Riddle** : 죄수들이 모두 switch가 있는 방을 언제 방문했는지를 결정하는 문제

이러한 환경은 perception으로 convolution이 필요하지 않지만 partial observability의 존재로 Recurrent Neural Network으로 복잡한 sequence를 처리해야 함

Partial observability는 다중 에이전트와 결합되기 때문에 최적의 정책은 에이전트 간의 통신에 의존인 특징이 있게 됨. communication protocol은 사전에 정의하지 않기 때문에 RL로 조정되는 protocol을 자동으로 개발해야 함

결과는 baseline에 해당하는 방법을 능가!!

RL로 communication protocol 학습을 성공한 첫번째 논문이고 DDRQN의 구성요소들이 성공에 중요하다는 것을 실험으로 제시

Background : DQN

$$s_t \in \mathcal{S}$$
$$a_t \in \mathcal{A}$$
$$\pi \quad r_t \quad s_{t+1}$$
$$\gamma \in [0, 1)$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right]$$

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s',a';\theta_i^-)$$

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots,$$

$$Q^\pi(s,a) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right]$$

Experience Replay

$$\mathcal{D}_t = \{e_1, e_2, \ldots, e_t\}$$
$$e_t = (s_t, a_t, r_t, s_{t+1})$$

Background : Independent DQN

Independent Q-Learning

DQN은 cooperative한 다중 에이전트 설정으로 확장되었고, 각각의 에이전트 m은 global state인 s_t를 관찰하고, 각각의 개별적인 행동인 a_t^m를 선택하고, 에이전트 간에 공유되는 팀단위의 보상인 r_t를 받음.

Tammpu는 DQN을 Independent Q-Learning과 결합한 프레임워크로 위의 설정을 고려. 결과적으로 아래와 같은 형태의 Q-functio을 설정하게 됨

$$Q^m(s, a^m; \theta_i^m)$$

단점

Independent Q-Learning은 수렴문제를 야기할 수 있음(한 에이전트의 학습이 환경을 다른 에이전트에게 non-stationary하게 보이기 때문)

Background : DRQN

DRQN

DQN과 IQN은 모두 fully observable한 가능성을 가정함. 에이전트의 입력으로 s_t를 사용함. 대조적으로, Partial observable한 환경에서는 s_t가 숨겨지고 에이전트는 s_t와 관계가 있는 observation를 사용하게 되지만 근본적으로 아주큰 차이를 보이는 것은 아님

기존의 연구에서 Matthew Hauknecht는 single에이전트를 처리하기 위해

Deep Recurrent Q-Network를 제안하여 partial한 경우를 부분적으로 해결

Feed forward네트워크를 통해 Q;a를 approximate하는 대신에 **internal state**를 유지하면서 시간이 흐름에 따라서 observation을 종합할 수 있는 Recurrent Neural Network로 Q(o;a)를 approximate함

$$Q(o_t, h_{t-1}, a; \theta_i)$$

DRQN은 각 timestep에서 Q_t와 h_t를 output으로 출력함

Background : Partially Observable Multi-Agent RL

Partially Observable Multi-Agent RL

다중 에이전트와 부분 관찰 가능성이 모두 있는 설정을 고려함. 각 에이전트는 각 time step마다 자체적인 /o_{t}^{m} 를 받고 내부 상태 /h_{t}^{m} 를 유지함

저자들은 학습이 centralize방식으로 일어날 수 있다고 가정함. 에이전트는 자기 자신만의 history에서 조건을 학습할 수 있다면 학습 과정에서 parameter를 공유할 수 있다고 가정함.

=> **Centralized Learning and Decentralized policy**

에이전트들이 communication할 동기를 얻는 것은 다중 에이전트와 partial observable한 가능성이 공존할 때만이 가능하기 때문에 이러한 경우에 해당하는 환경만을 고려함

DDRQN

DDRQN

Partially observable한 Multiagent설정에서 DRL에 대한 가장 간단한 방법으로 **DRQN**과 **Independent Q-Learning**을 결합. 이 방식을 논문에서는 **Naïve method**로 부름

$$Q^m(s, a^m; \theta_i^m)$$

$$Q(o_t, h_{t-1}, a; \theta_i) \Rightarrow Q^m(o_t^m, h_{t-1}^m, a^m; \theta_i^m)$$

이러한 naïve method의 **3가지**를 수정해서 DDRQN을 제시

DDRQN

1. last-action을 input으로 활용하는 방법

다음 time-step에 input으로 이전 time-step의 last-action을 제공하는 방안. 이유는 exploration을 위해서 stochastic한 정책을 사용하기 때문에 observation뿐만 아니라 action관찰의 history에 대한 행동을 조절해야 하기 때문이라고 함! 결과적으로 RNN이 행동 관찰에 대한 history를 조금 더 정확하게 파악할 수 있게 됨

2. inter-agent weight sharing

모든 에이전트 네트워크의 weight를 연결하는 것. 오직 하나의 네트워크만 학습되고 사용됨. 그렇지만 에이전트들은 다른 observation을 받고, 각각 숨겨진 상태를 학습으로 진화시키기 때문에 다르게 행동할 수 있음. 또한 각각의 에이전트들은 자체 인덱스 m을 입력값으로 받기 때문에 전문화하기가 쉬움. Weight sharing은 학습해야 할 파라미터의 숫자를 감소시켜서 학습 속도 향상에 도움을 줌

3. disabling experience replay

단일 에이전트에서는 experience replay가 유용하지만 다중 에이전트 세팅에서 에이전트들을 독립적으로 학습하는 경우 환경이 각각의 에이전트들에게 non-stationary하게 나타나기 때문에 사용하지 않는 것이 좋음

DDRQN

Algorithm 1 DDRQN

Initialise θ_1 and θ_1^-
for each episode e **do**
  $h_1^m = \mathbf{0}$ for each agent m
  $s_1 =$ initial state, $t = 1$
  while $s_t \neq$ terminal **and** $t < T$ **do**
    for each agent m **do**
      With probability ϵ pick random a_t^m
      else $a_t^m = \arg\max_a Q(o_t^m, h_{t-1}^m, m, a_{t-1}^m, a; \theta_i)$
    Get reward r_t and next state s_{t+1}, $t = t + 1$
  $\nabla\theta = 0$ ▷ reset gradient
  for $j = t - 1$ **to** $1, -1$ **do**
    for each agent m **do**
      $y_j^m = \begin{cases} r_j, \text{ if } s_j \text{ terminal, else} \\ r_j + \gamma \max_a Q(o_{j+1}^m, h_j^m, m, a_j^m, a; \theta_i^-) \end{cases}$
      Accumulate gradients for:
      $(y_j^m - Q(o_j^m, h_{j-1}^m, m, a_{j-1}^m, a_j^m; \theta_i))^2$
  $\theta_{i+1} = \theta_i + \alpha\nabla\theta$ ▷ update parameters
  $\theta_{i+1}^- = \theta_i^- + \alpha^-(\theta_{i+1} - \theta_i^-)$ ▷ update target network

DDRQN은

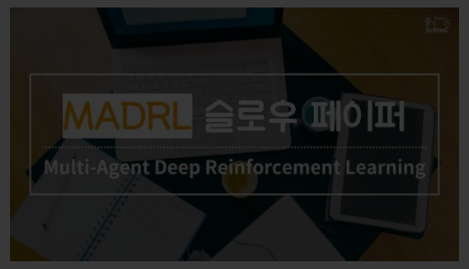
$$Q(o_t^m, h_{t-1}^m, m, a_{t-1}^m, a_t^m; \theta_i)$$

형태의 Q-function을 학습함.

θ_i **는 weight sharing 때문에 m에 대해서 condition을 걸지 않음**

a_{t-1}^m **은 history의 일부분이라는 것을 기억**

a_t^m **은 Q-Network가 estimate하는 action것을 기억**



DDRQN

Algorithm 1 DDRQN

Initialise θ_1 and θ_1^-
for each episode e **do**
  $h_1^m = \mathbf{0}$ for each agent m
  $s_1 =$ initial state, $t = 1$
  while $s_t \neq$ terminal **and** $t < T$ **do**
    for each agent m **do**
      With probability ϵ pick random a_t^m
      else $a_t^m = \arg\max_a Q(o_t^m, h_{t-1}^m, m, a_{t-1}^m, a; \theta_i)$
    Get reward r_t and next state s_{t+1}, $t = t + 1$
  $\nabla\theta = 0$ ▷ reset gradient
  for $j = t - 1$ **to** $1, -1$ **do**
    for each agent m **do**
      $y_j^m = \begin{cases} r_j, \text{ if } s_j \text{ terminal, else} \\ r_j + \gamma \max_a Q(o_{j+1}^m, h_j^m, m, a_j^m, a; \theta_i^-) \end{cases}$
      Accumulate gradients for:
      $(y_j^m - Q(o_j^m, h_{j-1}^m, m, a_{j-1}^m, a_j^m; \theta_i))^2$
  $\theta_{i+1} = \theta_i + \alpha\nabla\theta$ ▷ update parameters
  $\theta_{i+1}^- = \theta_i^- + \alpha^-(\theta_{i+1} - \theta_i^-)$ ▷ update target network

DDRQN은

$$Q^m(o_t^m, h_{t-1}^m, a^m; \theta_i^m)$$

형태의 Q-function을 학습함.

조건 1, 2로 인해서 변경

θ_i **는 weight sharing 때문에 m에 대해서**

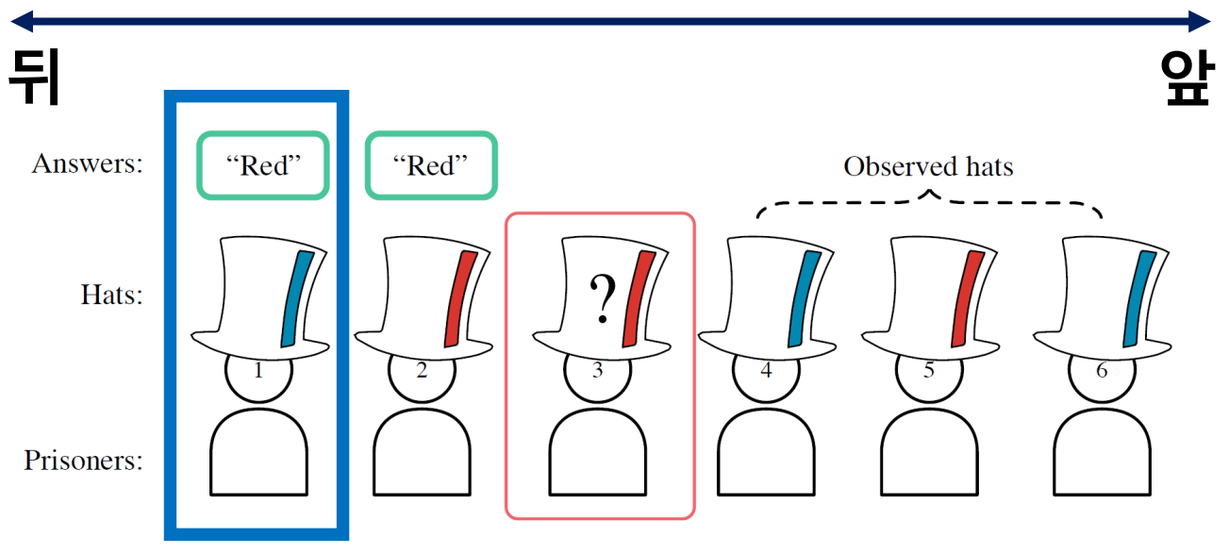
$$Q(o_t^m, h_{t-1}^m, m, a_{t-1}^m, a_t^m; \theta_i)$$

a_{t-1}^m **은 history의 일부분이라는 것을 기억**

a_t^m **은 Q-Network가 estimate하는 action것을 기억**

Multi-Agent Riddles : Hats Riddle

Hats Riddle 문제

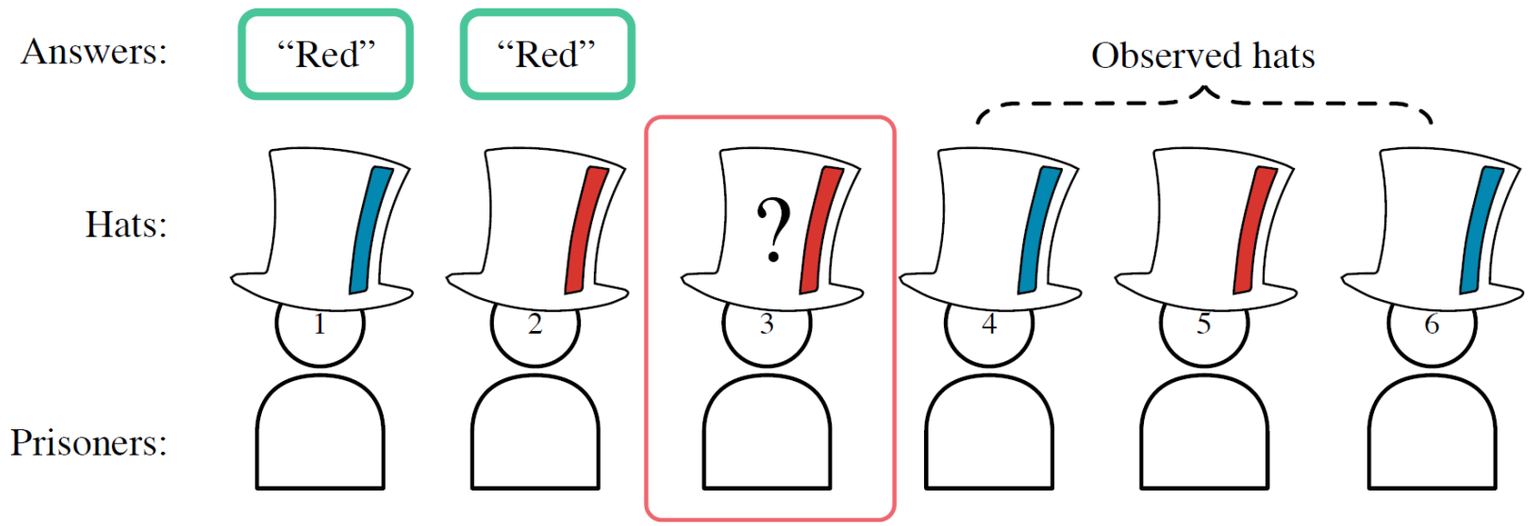


"한 사형 집행관이 100명의 죄수들을 한 줄로 묶고 각각의 죄수들의 머리에 붉은색 또는 파란색의 모자를 쓰게함. 죄수들은 자신의 앞에 있는 사람들의 모자를 줄에서 볼 수 있지만 자신과 자신의 뒤에 있는 죄수들의 모자를 볼 수 없음. 사형 집행관은 가장 뒤에 있는 죄수를 시작으로 가장 앞에 있는 죄수들에게 죄수 자신의 모자색을 물어봄. 죄수는 붉은색 혹은 파란색이라고 대답해야 하며 정답을 맞히면 살 수 있고 틀린 대답을 하면 사형에 처하게 됨. 참고로 모든 사람이 답을 듣는 동안 아무도 답이 옳은지 알 수 없음) 전날 밤에 죄수들은 자신들을 위한 전략을 세워야 함"

죄수들은 가장 뒤에 있는 죄수가 파란 모자의 숫자가 짝수라면 "청색"이라고 말하고, 그렇지 않으면 "붉은색"이라고 말하는 communication protocol에 동의하는 것이 최적의 전략(**이미 검증된 방식이니 고민하지 말고 받아드리자**). 남은 죄수들은 앞에서 본 모자와 그들 뒤에서 들은 대답을 바탕으로 자신들의 모자 색을 추론할 수 있음

http://news.chosun.com/site/data/html_dir/2016/02/24/2016022402442.html

Hats Riddle : Formalization



$s = (s^1, ..., s^n, a^1, ..., a^n)$ - state of the environment

$s^m \in \{blue, red\}$ – *m*-th agent's hat colour

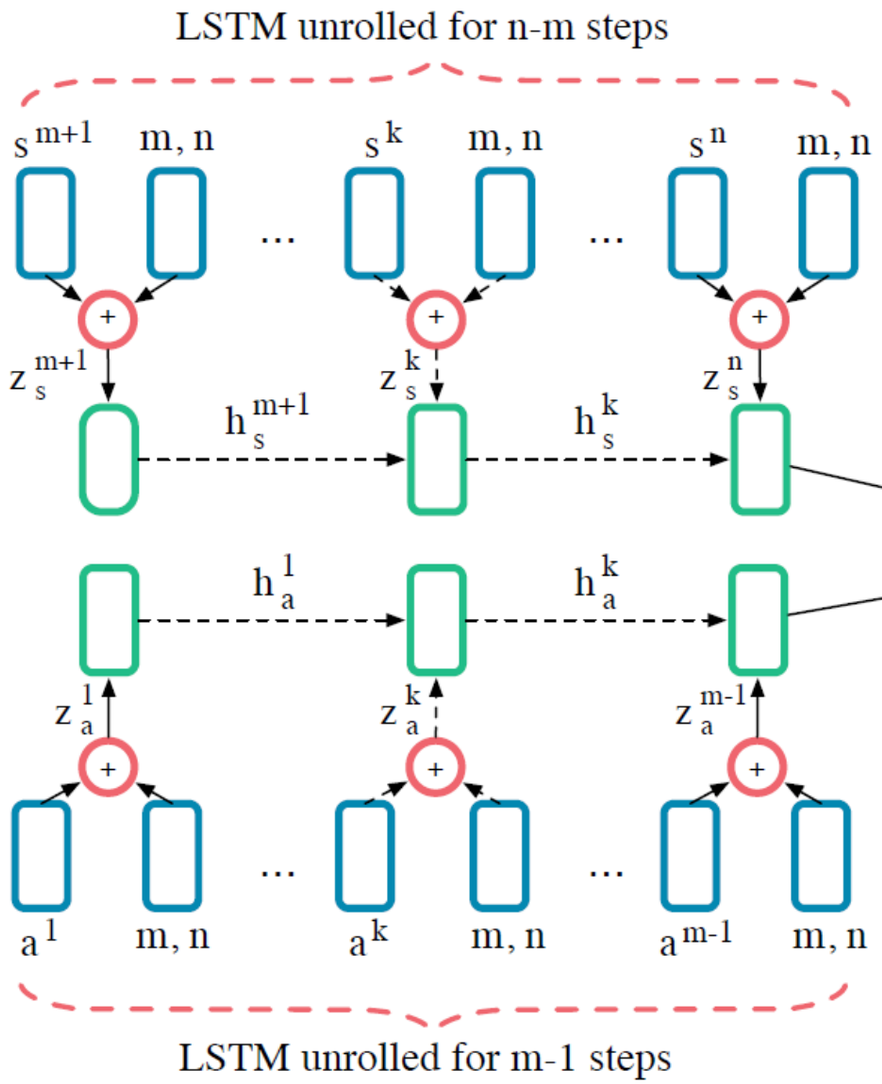
$a^m \in \{blue, red\}$ – *m*-th agent's action at *m*-th step

$o^m = (a^1, ..., a^{m-1}, s^{m+1}, ..., s^n)$ – *m*-th agent's observation

$r_n = \Sigma_m I(a^m = s^m)$ – number of prisoners alive at the end

Hats Riddle : Network Architecture

Hats Riddle : Network Architecture



$$z_s^k = MLP[1\times 64](s^k) \oplus MLP[2\times 64](m,n)$$

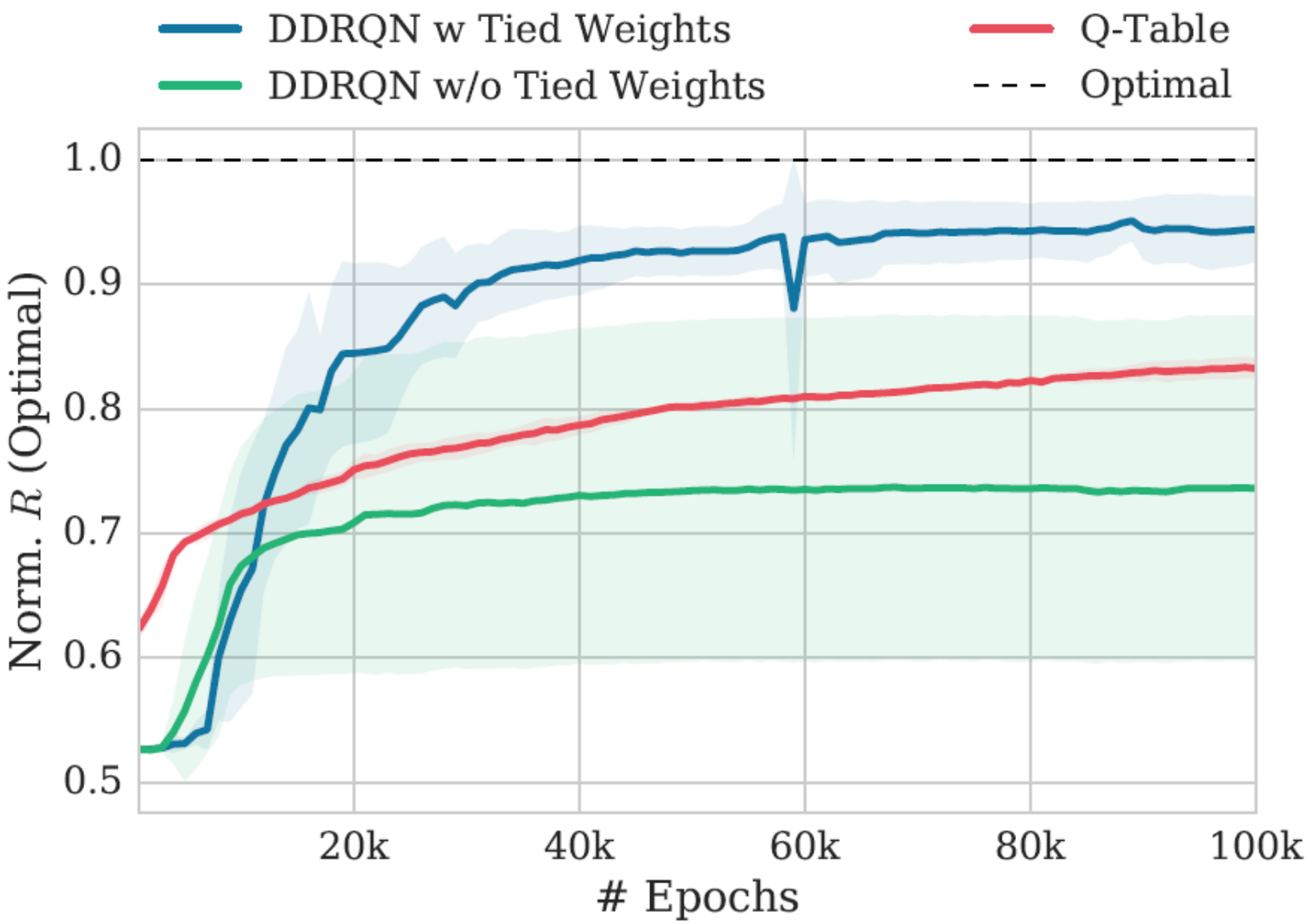
$$y_s^k, h_s^k = LSTM_s[64](z_s^k, h_s^{k-1})$$

$$y_a^k, h_a^k = LSTM_a[64](z_a^k, h_a^{k-1})$$

$$z_a^k = MLP[1\times 64](a^k) \oplus MLP[2\times 64](m,n)$$

$$Q^m = MLP[128\times 64, 64\times 64, 64\times 1](y_a^{m-1} \,\|\, y_s^n)$$

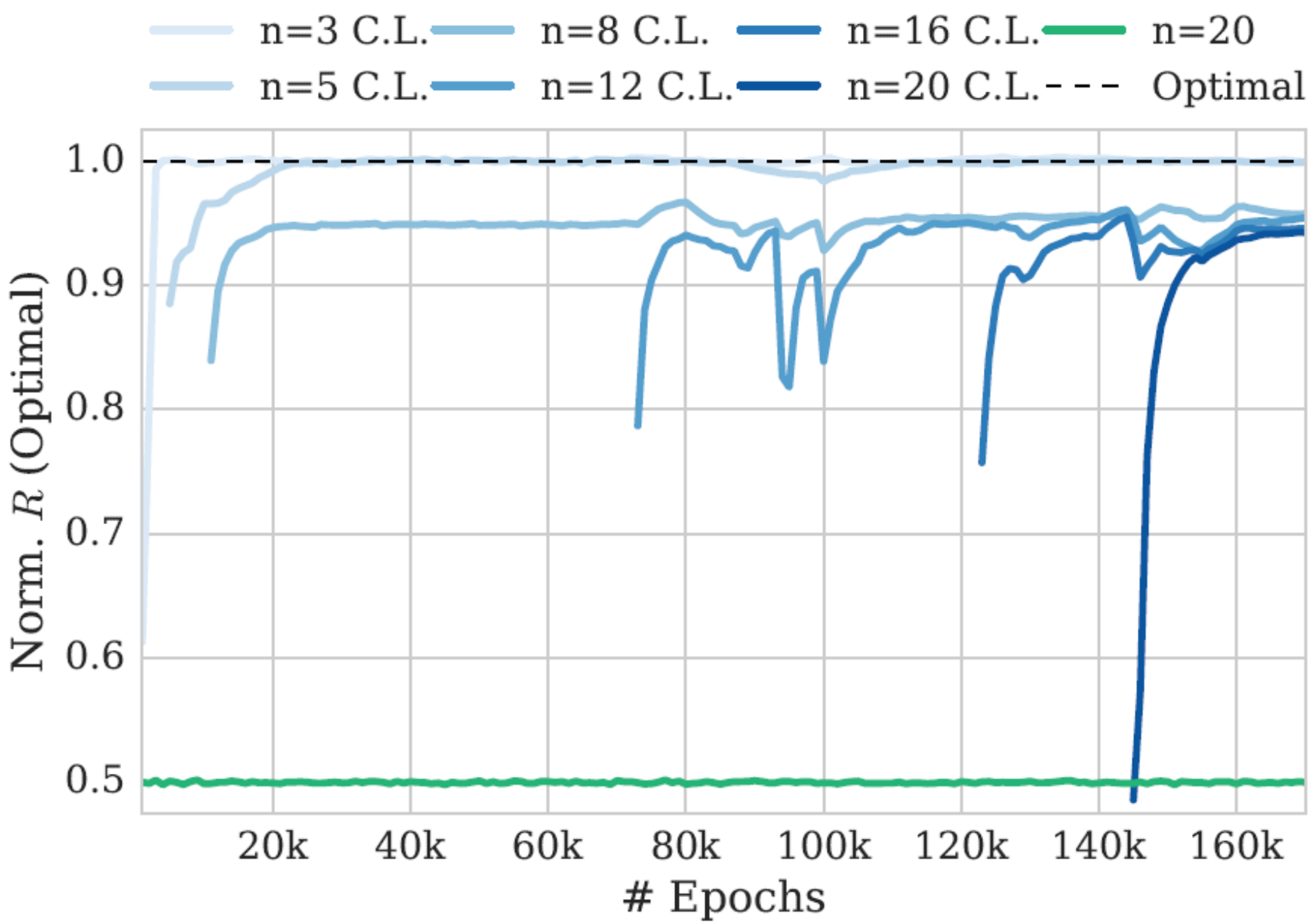
Hats Riddle : Results



Hats Riddle : Emergent Strategy

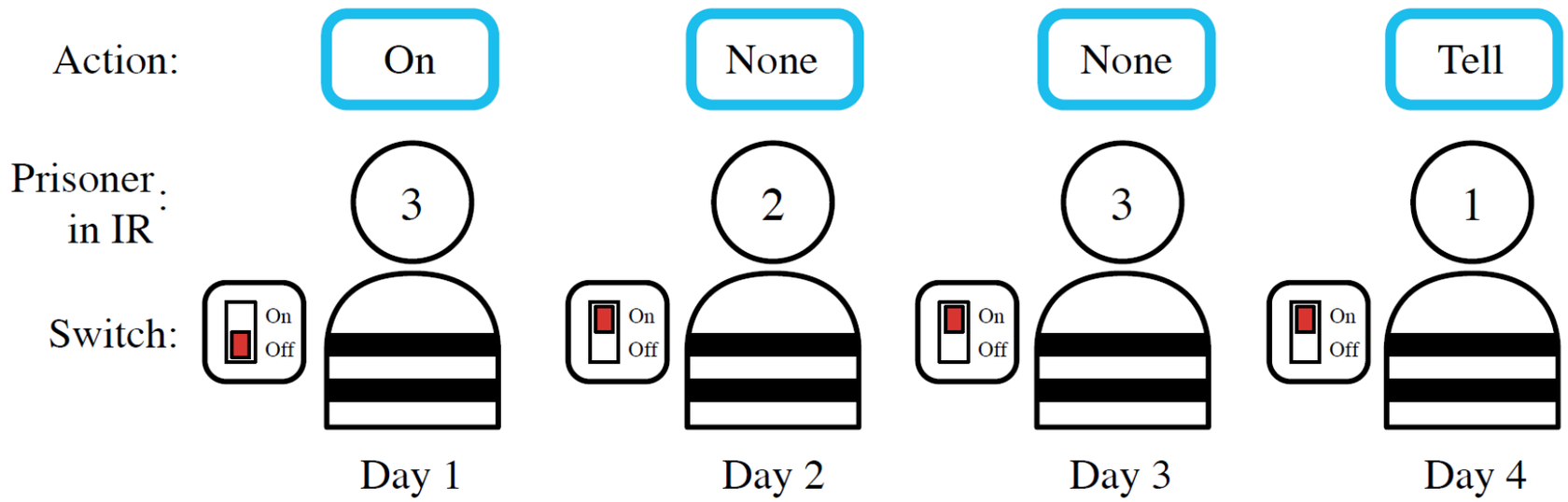
N	% AGREEMENT
3	100.0%
5	100.0%
8	79.6%
12	52.6%
16	50.8%
20	52.5%

Hats Riddle : Curriculum Learning



Multi-Agent Riddles : Switch Riddle

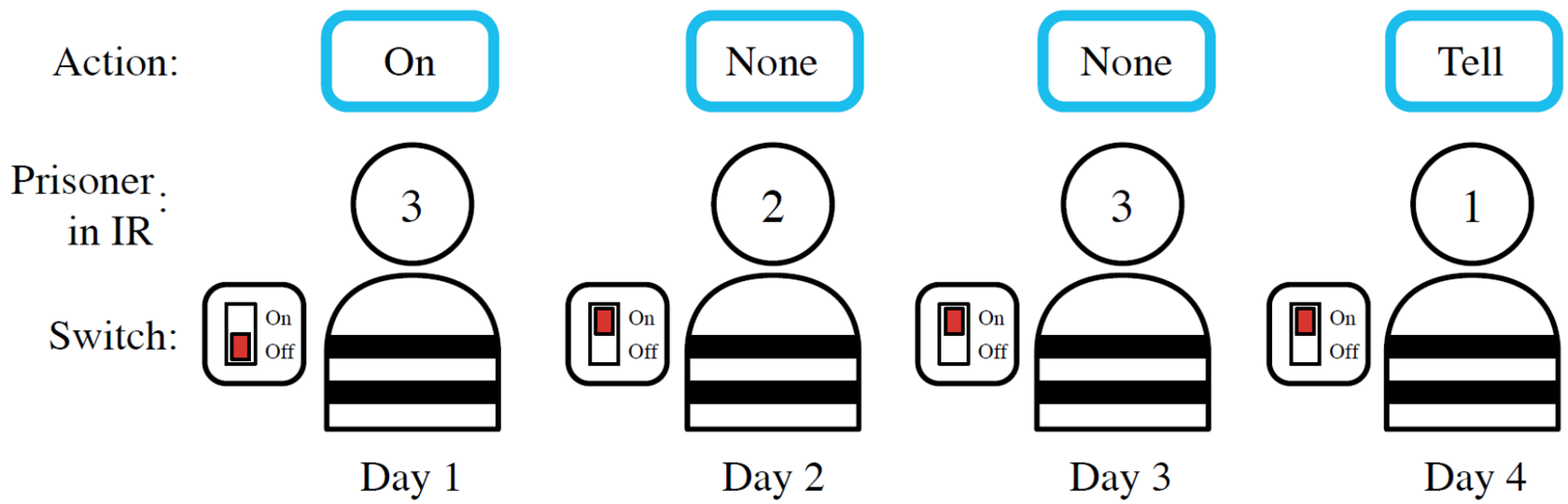
Switch Riddle 문제



“100명의 죄수가 감옥에 들어가 있음. 관리자는 죄수들에게 내일부터 서로 간에 소통을 할 수 없는 격리된 감옥에 갇히게 될 것이라고 말함. 매일 교도장은 죄수 중 한 명을 교체와 함께 무작위로 교체하고 토글 스위치가 달린 전구가 있는 수사실에 배치함. 죄수는 전구의 현재 상태를 관찰할 수 있음. 죄수가 원할 경우 스위치를 토글할 수 있고 모든 죄수들이 어느 시점에 수사실에 방문했다고 믿는다는 것을 발표할 수 있음. 만약 이 발표가 사실이면, 모든 죄수들은 석방되지만 거짓일 경우에는 모든 죄수들이 처형됨. 관리자가 떠나고 죄수들은 서로 모여서 그들의 운명을 의논함”

여러가지 전략들이 연구되었지만 그 중에 한가지 잘 알려진 전략은 죄수 한 명이 카운터에 지명되는 것임. 죄수들이 스위치를 한 번만 켜는 동안 카운터 지명된 사람만이 스위치를 끌 수 있음. 그렇기 때문에 카운터가 스위치는 n-1번 끄면, 살아서 나갈 수 있음

Switch Riddle : Formalization



$s = (SW_t, IR_t, s^1, ..., s^n)$ – state of the environment

$SW_t \in \{on, off\}$ – the position of the switch

$IR_t \in \{1, ..., n\}$ – current visitor in the room

$s^1, ..., s^n \in \{0,1\}$ – has the prisoner already been in the room

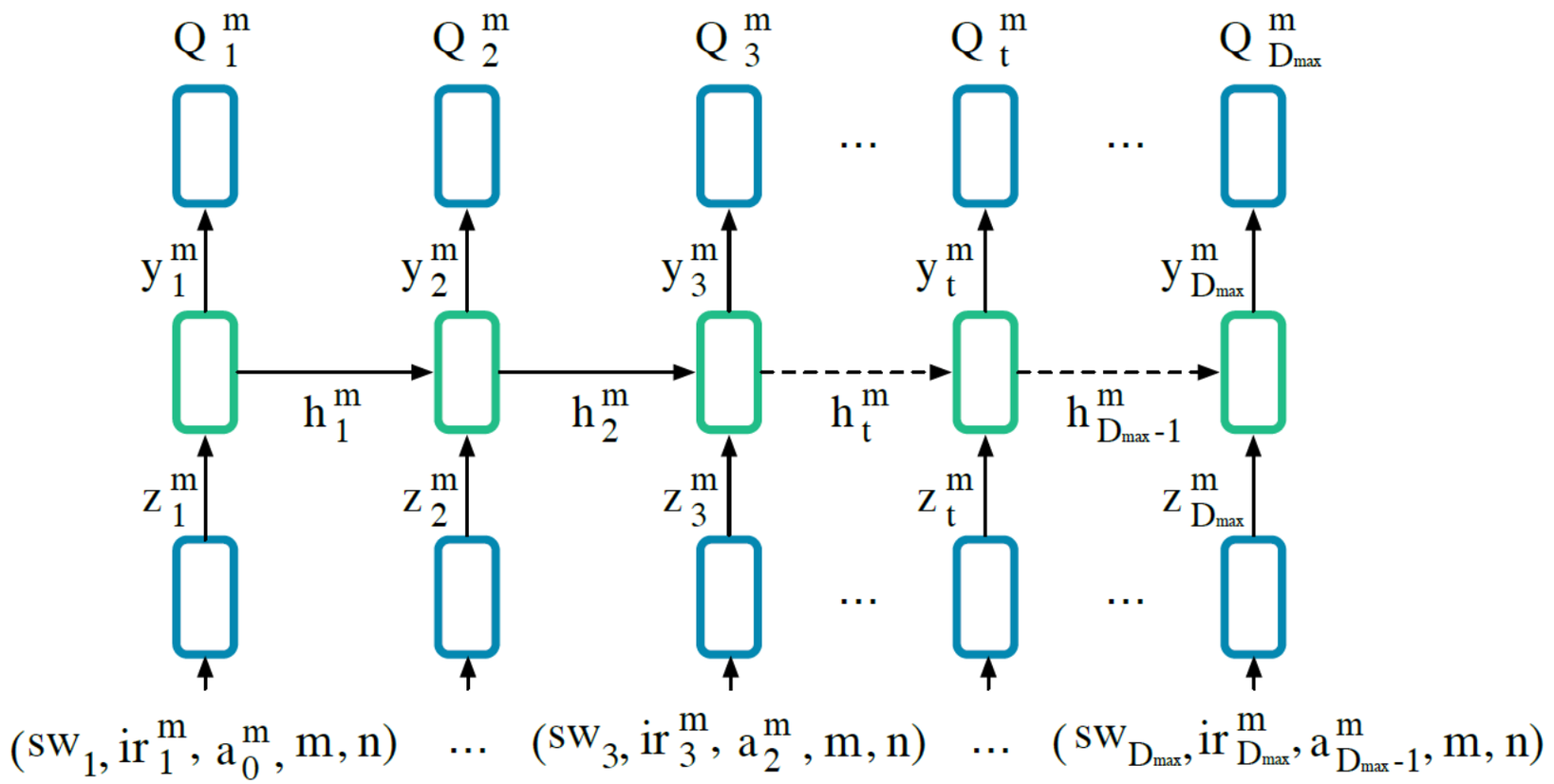
$o_t^m = (ir_t, sw_t)$ – observation, where $ir_t = \boldsymbol{I}(IR_t=m)$, $sw_t = SW_t$

$a_t^m \in \{\text{“On”}, \text{“Off”}, \text{“Tell”}, \text{“None”}\}$ – action of agent *m*

$R_t \in \{0, 1, -1\}$ – reward after max days or when action is Tell.

Switch Riddle : Network Architecture

Switch Riddle : Network Architecture

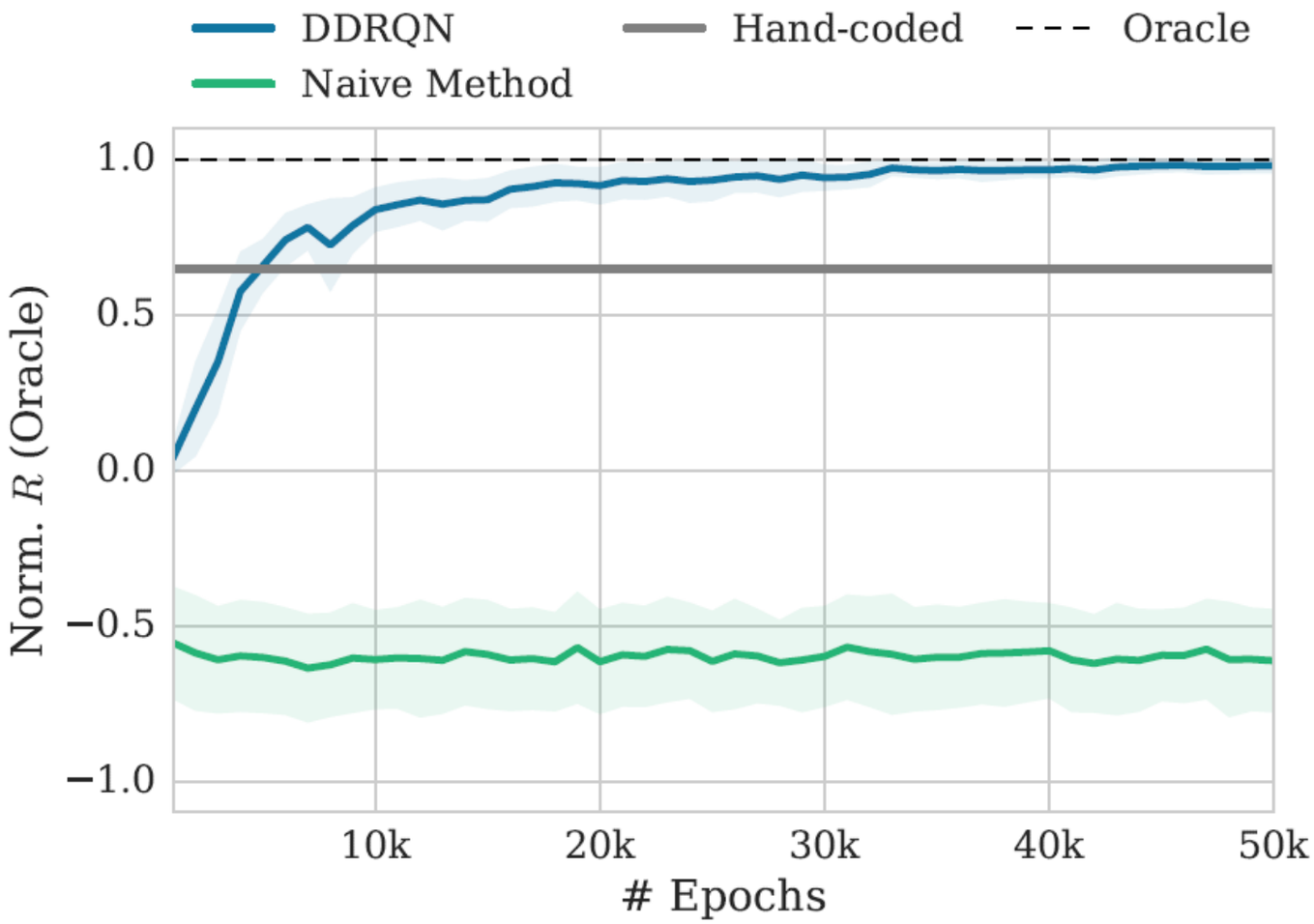


$$z_t^m = MLP[(7+n)\times 128, 128\times 128](o_t^m, OneHot(a_{t-1}^m), OneHot(m), n)$$

$$y_t^m, h_t^m = LSTM[128](z_t^m, h_{t-1}^m)$$

$$Q_t^m = MLP[128\times 128, 128\times 128, 128\times 4](y_t^m) \qquad D_{\max} = 4n-6$$

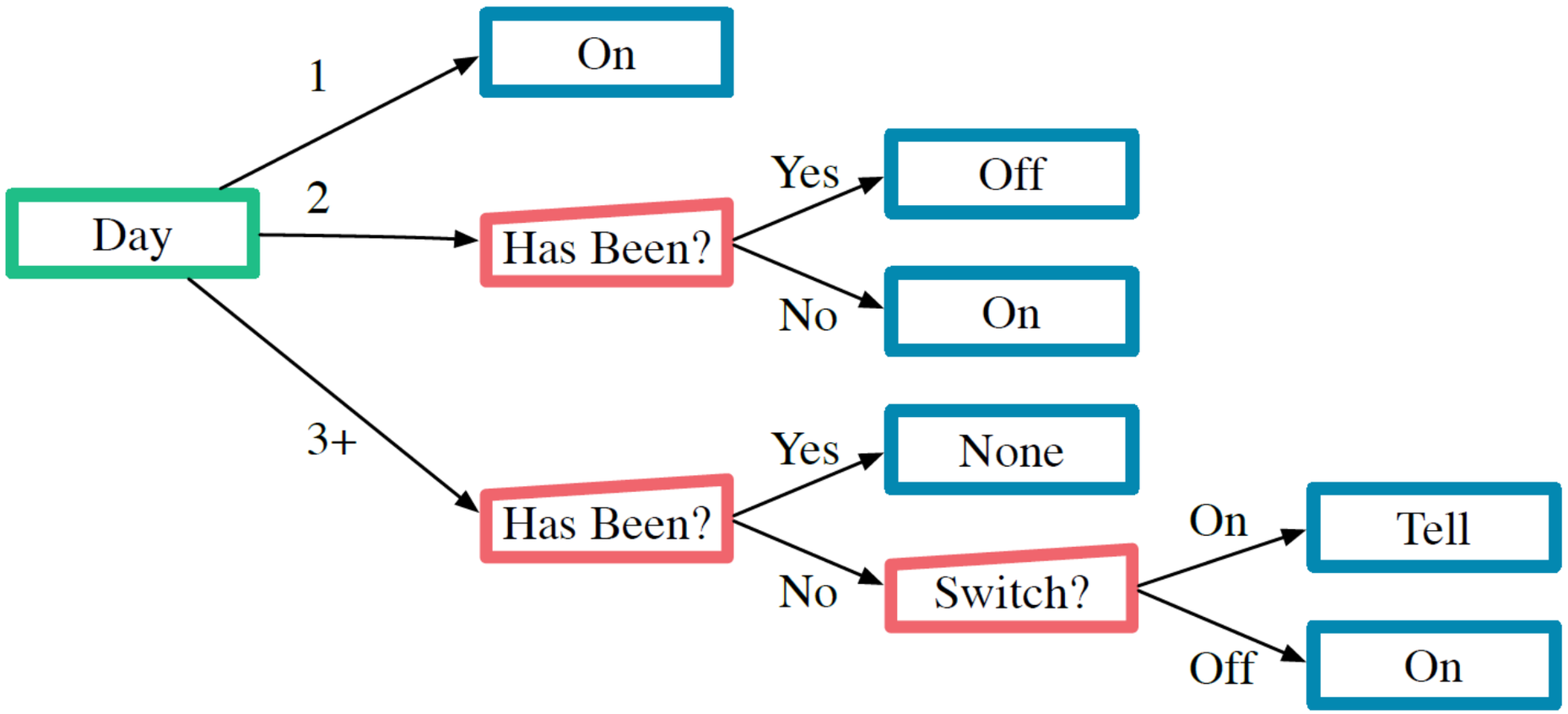
Switch Riddle : Results n=3



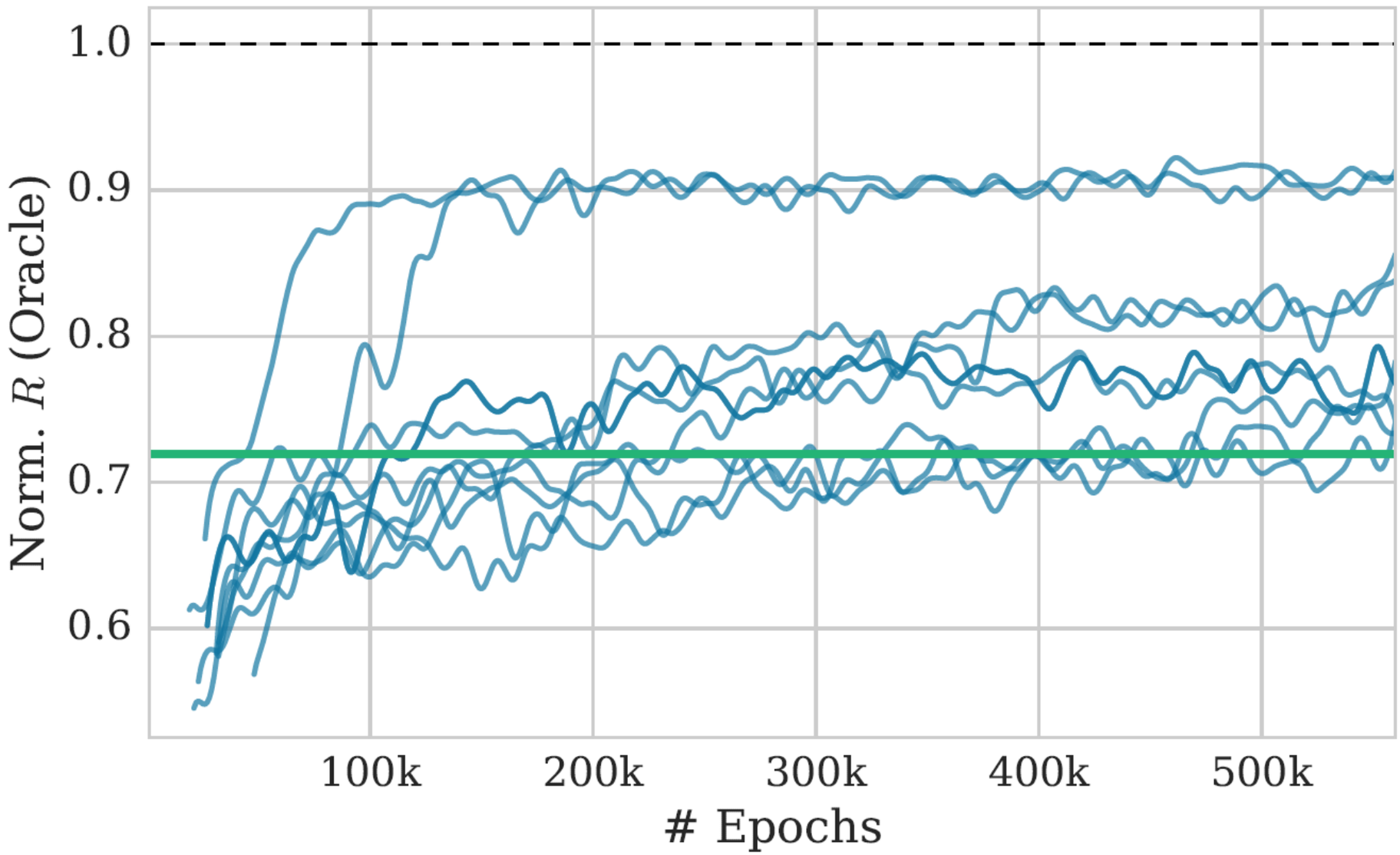
N=3에 대한 결과를 확인하면
DDRQN과 naïve approach, hand-coded 전략의 “tell on last day” 전략의 최적 정책을 비교한 내용임

DDRQN이 성능을 향상시키는 것을 알 수 있음

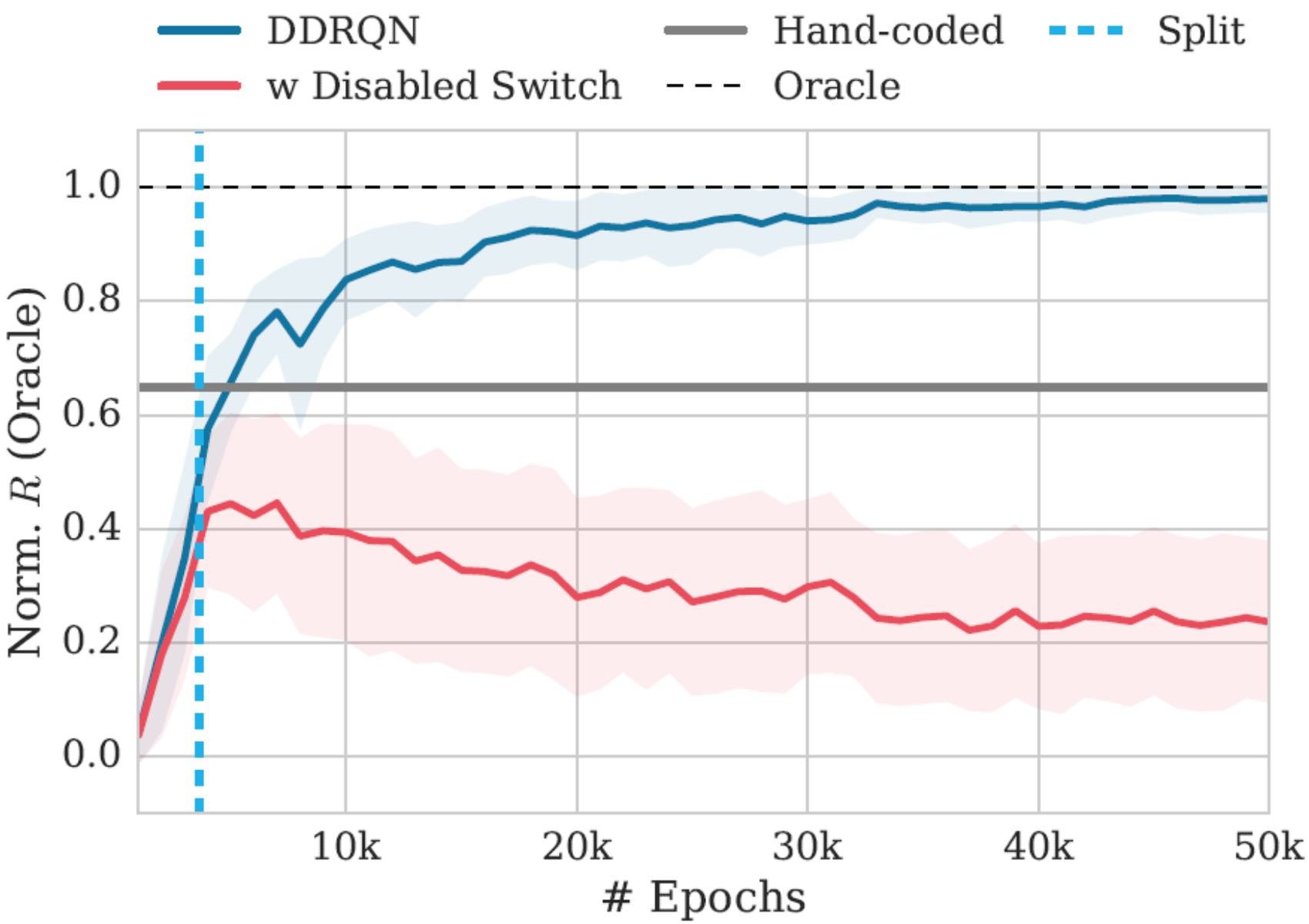
Switch Riddle : Strategy n=3



Switch Riddle : Results n=4

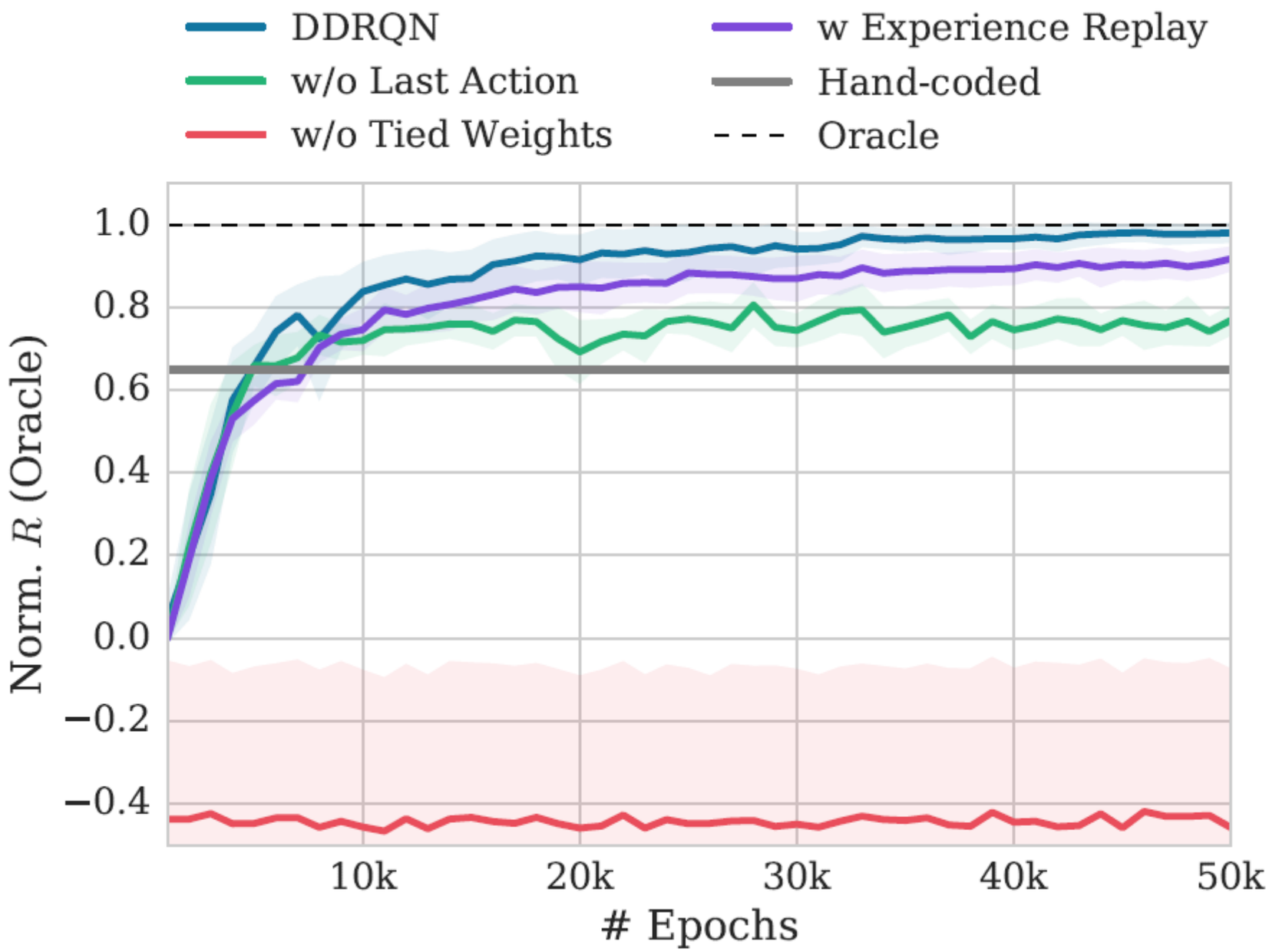


Switch Riddle : No Switch



Switch Riddle : Ablation Experiments

Switch Riddle : Ablation Experiments



Finish!!

Thank you