Deep Multiagent Reinforcement Learning for Partially Observable Parameterized Environments

Peter Stone*

Department of Computer Science
The University of Texas at Austin

*Cogitai

Joint work with Matthew Hausknecht
Motivation
Outline

1. Background

2. Recurrent Q-Learning for partially observable MDPs

3. Deep Multiagent RL in Half-Field-Offense

4. Future Work
Markov Decision Process

Markov Property ensures $s_{t+1}$ depends only on $s_t$

Learning an optimal policy $\pi^*$ requires no memory
Partially Observable MDP (POMDP)

Observations provide noisy or incomplete information

Memory may help to learn a better policy
Reinforcement Learning provides a general framework for sequential decision making.

**Objective**: Learn a policy that maximizes discounted sum of future rewards.

Deterministic policy $\pi$ is a mapping from states/observations to actions.

For each encountered state/observation, what is the best action to perform.
Q-Value Function

Estimates the expected return from a given state-action:

$$Q^\pi(s, a) = \mathbb{E} [r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s, a]$$

Answers the question: “How good is action $a$ from state $s$.”

Optimal Q-Value function yields an optimal policy.
Deep Neural Network

Parametric model with stacked layers of representation.

Powerful, general purpose function approximator.

Parameters $\theta$ optimized via backpropagation.
Outline

1. Background

2. Recurrent Q-Learning for partially observable MDPs

3. Deep Multiagent RL in Half-Field-Offense

4. Future Work
Atari Environment

Observation $o_t$

Reward is change in game score

Resolution 160x210x3

18 discrete actions
Atari: MDP or POMDP?

Depends on the number game screens used in the state representation.

Many games PO with a single frame.
Deep Q-Network (DQN)

Neural network estimates Q-Values $Q(s,a)$ for all 18 actions:

$$Q(s|\theta) = (Q_{s,a_1} \ldots Q_{s,a_n})$$

Learns via temporal difference:

$$L_i(\theta_i) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1}) \sim \mathcal{D}} \left[ (y_i - Q(s_t|\theta_i))^2 \right]$$

$$y_i = r_t + \gamma \text{max}(Q(s_{t+1}|\theta))$$

Accepts the last 4 screens as input.
Flickering Atari

How well does DQN perform on POMDPs?

Induce partial observability by stochastically obscuring the game screen

\[ o_t = \begin{cases} 
  s_t & \text{with } p = \frac{1}{2} \\
  < 0, \ldots, 0 > & \text{otherwise}
\end{cases} \]

Game state must be inferred from past observations
DQN Pong

True Game Screen

Observed Game Screen
DQN Flickering Pong

True Game Screen  Observed Game Screen
Deep Recurrent Q-Network

Uses a Long Short Term Memory (LSTM) to selectively remember past game screens.

Architecture identical to DQN except:
1. Replaces FC layer with LSTM
2. Single frame as input each timestep

Trained end-to-end using BPTT for last 10 timesteps.
DRQN Flickering Pong

True Game Screen  Observed Game Screen
LSTM infers velocity
DRQN Frostbite
Extensions

DRQN has been extended in several ways:

• **Addressable Memory**: *Control of Memory, Active Perception, and Action in Minecraft*; Oh et al. in ICML ’16

• **Continuous Action Space**: *Memory Based Control with Recurrent Neural Networks*; Heess et al., 2016

[Deep Recurrent Q-Learning for Partially Observable MDPs, Hausknecht et al, 2015; ArXiv]
Outline

1. Background
2. Recurrent Q-Learning for partially observable MDPs
3. Deep Multiagent RL in Half-Field-Offense
4. Future Work
Half Field Offense

Cooperative multiagent soccer domain built on the libraries used by the RoboCup competition

Objective: Learn a goal scoring policy for the offense agents

Features continuous actions, partial observability, and opportunities for multiagent coordination
Half Field Offense 25
State Action Spaces

58 continuous state features encoding distances and angles to points of interest

Parameterized-Continuous Action Space:
Dash(direction, power)
Turn(direction)
Tackle(direction)
Kick(direction, power)

Choose one discrete action + parameters every timestep
Exploration is Hard
With only goal-scoring reward, agent never learns to approach the ball or dribble.
Deep Deterministic Policy Gradients

Model-free Deep Actor Critic architecture [Lillicrap '15]

Actor learns a policy $\pi$, Critic learns to estimate Q-values

Actor outputs all 6 possible parameters.

$a_t = \max(4 \text{ actions}) + \text{associated parameter(s)}$
Training

Critic trained using temporal difference:

\[ L = \left\| Q(s_t, \mu(s_t)|\theta_Q) - y \right\|^2_2 \]
\[ y = r_t + \gamma(Q(s_{t+1}, \mu(s_{t+1})|\theta_Q)) \]

Actor trained via Critic gradients:

\[ \nabla_{\theta \mu} \mu(s) = \nabla_a Q(s, a|\theta_Q) \nabla_{\theta \mu} \mu(s|\theta_\mu) \]
Bounded Action Space

HFO’s continuous parameters are bounded

Dash(direction, power)
Turn(direction)
Tackle(direction)
Kick(direction, power)

Direction in [-180, 180], Power in [0, 100]

Exceeding these ranges results in no action

If DDPG is unaware of the bounds, it will invariably exceed them
Bounded DDPG

We examine 3 approaches for bounding the DDPG’s action space:

1. Squash Gradients
2. Zero Gradients
3. Invert Gradients
Squashing Gradients

1. Use Tanh non-linearity to bound parameter output

2. Rescale into desired range
Squashing Gradients

![Graphs showing reward and average critic Q-value over episodes and critic iterations.](image-url)
Zeroing Gradients

Each continuous parameter has a range: \([p_{\text{min}}, p_{\text{max}}]\)

Let \(p\) denote current value of parameter, and \(\nabla_p\) the suggested gradient.

Then:

\[
\nabla_p = \begin{cases} 
\nabla_p & \text{if } p_{\text{min}} < p < p_{\text{max}} \\
0 & \text{otherwise}
\end{cases}
\]
Zeroing Gradients

![Graphs showing reward and average critic Q-value over episode and critic iteration.](image-url)
Inverting Gradients

For each parameter:

\[
\nabla_p = \nabla_p \cdot \begin{cases} 
\frac{(p_{\text{max}} - p)}{(p_{\text{max}} - p_{\text{min}})} & \text{if } \nabla_p \text{ suggests increasing } p \\
\frac{(p - p_{\text{min}})}{(p_{\text{max}} - p_{\text{min}})} & \text{otherwise}
\end{cases}
\]

Allows parameters to approach the bounds of the ranges without exceeding them

Parameters don’t get “stuck” or saturate
Inverting Gradients

![Graph showing reward and average critic Q-value over episode and critic iteration.](image)
## Results

<table>
<thead>
<tr>
<th></th>
<th>Scoring Percent</th>
<th>Avg. Steps to Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDPG&lt;sub&gt;1&lt;/sub&gt;</td>
<td>1.0</td>
<td>108.0</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;2&lt;/sub&gt;</td>
<td>.99</td>
<td>107.1</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;3&lt;/sub&gt;</td>
<td>.98</td>
<td>104.8</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;4&lt;/sub&gt;</td>
<td>.96</td>
<td>112.3</td>
</tr>
<tr>
<td>Helios’ Champion</td>
<td>.96</td>
<td>72.0</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;5&lt;/sub&gt;</td>
<td>.94</td>
<td>119.1</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;6&lt;/sub&gt;</td>
<td>.84</td>
<td>113.2</td>
</tr>
<tr>
<td>SARSA</td>
<td>.81</td>
<td>70.7</td>
</tr>
<tr>
<td>DDPG&lt;sub&gt;7&lt;/sub&gt;</td>
<td>.80</td>
<td>118.2</td>
</tr>
</tbody>
</table>

[Deep Reinforcement Learning in Parameterized Action Space, Hausknecht and Stone, in ICLR ‘16]
Deep Multiagent RL

Can multiple Deep RL agents cooperate to achieve a shared goal?

Examine several baseline architectures:

Decentralized: Independent agents

Centralized: Single controller for multiple agents

Parameter Sharing: Layers shared between agents
Centralized

Both agents are controlled by a single DDPG

State & Action spaces are concatenated

Learning is more challenging for this reason
Parameter Sharing

Shared weights between layers in Actor networks. Separate sharing between Critic networks.

Reduces total number of parameters!

Encourages both agents to participate even though 2v0 is solvable by a single agent.
Related Work

• *Multiagent Cooperation and Competition with Deep Reinforcement Learning*; Tampuu et. al, 2015

• *Learning to Communicate to Solve Riddles with Deep Distributed Recurrent Q-Networks*; Foerster et al., 2016

• *Learning to Communicate with Deep Multi-Agent Reinforcement Learning*; Foerster et al., 2016
Thanks!