CS230: Lecture 9
Deep Reinforcement Learning

Kian Katanforoosh
Menti code: 21 90 15
I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Learning

IV. Application of Deep Q-Learning: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics
I. Motivation

Mastering the Game of Go without Human Knowledge


DeepMind, 5 New Street Square, London EC1A 3TW.

*These authors contributed equally to this work.

A long-standing goal of artificial intelligence is an algorithm that learns, tabula rasa, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search AI AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here, we introduce an algorithm based solely on reinforcement learning, without human data, guidance, or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo’s own move selections and also the winner of AlphaGo’s games. This neural network improves the strength of tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting tabula rasa, our new program AlphaGo Zero achieved superhuman performance, winning 100-0 against the previously published, champion-defeating AlphaGo.

AlphaGo

Human Level Control through Deep Reinforcement Learning

[Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri]
I. Motivation

How would you solve Go with classic supervised learning?

Why RL?
• Delayed labels
• Making sequences of decisions

What is RL?
• Automatically learn to make good sequences of decision

Examples of RL applications

Robotics
Games
Advertisement

Issues:
- Ground truth probably wrongly defined.
- Too many states in this Game.
- We will likely not generalize.

Source: https://deepmind.com/blog/alphago-zero-learning-scratch/
I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics
II. Recycling is good: an introduction to RL

Problem statement

Goal: maximize the return (rewards)

Number of states: 5

Types of states:

Agent’s Possible actions:

Additional rule: garbage collector coming in 3min, it takes 1min to move between states

How to define the long-term return?

Discounted return

\[ R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ... \]
II. Recycling is good: an introduction to RL

**Problem statement**

Define reward "r" in every state

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2 (initial)</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
<tr>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+10</td>
</tr>
</tbody>
</table>

Assuming $\gamma = 0.9$

**What do we want to learn?**

how good is it to take action 1 in state 2

**Q-table**

$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$

**How?**

**Discounted return**

$R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
II. Recycling is good: an introduction to RL

Problem statement

What do we want to learn?

How good is it to take action 1 in state 2

\[ Q = \begin{pmatrix}
Q_{11} & Q_{12} \\
Q_{21} & Q_{22} \\
Q_{31} & Q_{32} \\
Q_{41} & Q_{42} \\
Q_{51} & Q_{52}
\end{pmatrix} \]

Q-table

How?

State 1 State 2 (initial) State 3 State 4 State 5

\[ S1 \quad S2 \quad S3 \quad S4 \quad S5 \]

How good is it to take action 1 in state 2

Define reward “r” in every state

\[ R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ... \]

Discounted return

Assuming \( \gamma = 0.9 \)

+2 0 0 +1 +10

S1 S2 S3 S4 S5

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
## II. Recycling is good: an introduction to RL

### Problem statement

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2 (initial)</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Trash can]</td>
<td>![Rubber ducky]</td>
<td>![Chocolate bar]</td>
<td>![Water drop]</td>
<td>![Recycle symbol]</td>
</tr>
</tbody>
</table>

**Define reward “r” in every state**

| +2 | 0 | 0 | +1 | +10 |
| S1 | S2 | S3 | S4 | S5 |

**Assuming** $\gamma = 0.9$

**Discounted return**

$$R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$$

### What do we want to learn?

- how good is it to take action 1 in state 2

**Q-table**

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix}$$

### How?

- State 1
- State 2 (initial) +2
- State 3 +9
- State 4 +10
- State 5

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
II. Recycling is good: an introduction to RL

Problem statement

Define reward “r” in every state

<table>
<thead>
<tr>
<th>State</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>+2</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>+1</td>
</tr>
<tr>
<td>S5</td>
<td>+10</td>
</tr>
</tbody>
</table>

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$

What do we want to learn?

how good is it to take action 1 in state 2

Q-table $Q = \begin{pmatrix}
Q_{11} & Q_{12} \\
Q_{21} & Q_{22} \\
Q_{31} & Q_{32} \\
Q_{41} & Q_{42} \\
Q_{51} & Q_{52}
\end{pmatrix}$

How?

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
II. Recycling is good: an introduction to RL

Problem statement

Define reward “r” in every state

What do we want to learn?

how good is it to take action 1 in state 2

Q-table

\[ Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \\ Q_{31} & Q_{32} \\ Q_{41} & Q_{42} \\ Q_{51} & Q_{52} \end{pmatrix} \]

How?

Assuming \( \gamma = 0.9 \)

Discounted return

\[ R = \sum_{t=0} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \]
What do we want to learn?

- Define reward \( r \) in every state
- Discounted return
  \[
  R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots
  \]

Problem statement

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2 (initial)</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td></td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
</tbody>
</table>

Start:

- +2 for S1
- 0 for S2
- 0 for S3
- +1 for S4
- +10 for S5

Assuming \( \gamma = 0.9 \)

Discounted return

\[
R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots
\]

How?

Q-table

\[
Q = \begin{pmatrix}
    Q_{11} & Q_{12} \\
    Q_{21} & Q_{22} \\
    Q_{31} & Q_{32} \\
    Q_{41} & Q_{42} \\
    Q_{51} & Q_{52}
\end{pmatrix}
\]

Q-table values

- \( Q_{11} = 0 + 0.9 \times 10 = 9 + 8.1 = 17.1 \)
- \( Q_{21} = 0 + 0.9 \times 10 = 10 \)
- \( Q_{31} = 0 + 0.9 \times 9 = 8.1 \)
- \( Q_{41} = 0 + 0.9 \times 9 = 8.1 \)
- \( Q_{51} = 0 + 0.9 \times 9 = 8.1 \)

## II. Recycling is good: an introduction to RL
Problem statement

Define reward “r” in every state

+2 0 0 +1 +10
S1 S2 S3 S4 S5

Assuming $\gamma = 0.9$

Discounted return

$R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$

What do we want to learn?

How good is it to take action 1 in state 2

Q-table

$Q = \begin{pmatrix}
0 & 0 & 2 & 9 & 8.1 & 10 & 9 & 10 & 0 & 0 \\
\end{pmatrix}$

How?

II. Recycling is good: an introduction to RL

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
II. Recycling is good: an introduction to RL

Problem statement

![Diagram showing states and rewards](image)

Define reward “r” in every state

- +2
- 0
- 0
- +1
- +10

Best strategy to follow if $\gamma = 0.9$

![Diagram showing Q-table](image)

What do we want to learn?

- How good is it to take action 1 in state 2

Q-table

$$Q = \begin{pmatrix}
0 & 0 \\
2 & 9 \\
8.1 & 10 \\
9 & 10 \\
0 & 0
\end{pmatrix}$$

Bellman equation (optimality equation)

$$Q^*(s, a) = r + \gamma \max_{a'} (Q^*(s', a'))$$

Policy

$$\pi(s) = \arg \max_a (Q^*(s, a))$$
What we’ve learned so far:

- Vocabulary: environment, agent, state, action, reward, total return, discount factor.

- Q-table: matrix of entries representing “how good is it to take action $a$ in state $s$”

- Policy: function telling us what’s the best strategy to adopt

- Bellman equation satisfied by the optimal Q-table
I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Learning

IV. Application of Deep Q-Learning: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics
III. Deep Q-Learning

Main idea: find a Q-function to replace the Q-table

Problem statement

State 1  State 2 (initial)  State 3  State 4  State 5

Q-table

\[
Q = \begin{pmatrix}
0 & 0 \\
2 & 9 \\
8.1 & 10 \\
9 & 10 \\
0 & 0 \\
\end{pmatrix}
\]

Neural Network

Then compute loss, backpropagate.

How to compute the loss?
III. Deep Q-Learning

\[ Q^*(s,a) = r + \gamma \max_{a'} (Q^*(s',a')) \]

**Loss function**

\[ L = (y - Q(s, \leftarrow))^2 \]

Target value

Case: \( Q(s, \leftarrow) > Q(s, \rightarrow) \)

\[ y = r_{\leftarrow} + \gamma \max_{a'} Q(s_{\leftarrow}^{next}, a') \]

Immediate reward for taking action \( \leftarrow \) in state \( s \)

Discounted maximum future reward when you are in state \( s_{\leftarrow}^{next} \)

Hold fixed for backprop

Case: \( Q(s, \leftarrow) < Q(s, \rightarrow) \)

\[ y = r_{\rightarrow} + \gamma \max_{a'} Q(s_{\rightarrow}^{next}, a') \]

Immediate Reward for taking action \( \rightarrow \) in state \( s \)

Discounted maximum future reward when you are in state \( s_{\rightarrow}^{next} \)

Hold fixed for backprop

[Francisco S. Melo: Convergence of Q-learning: a simple proof]
III. Deep Q-Learning

Loss function (regression)
\[ L = (y - Q(s, \rightarrow))^2 \]

Target value
Case: \[ Q(s, \leftarrow) > Q(s, \rightarrow) \]
\[ y = r_\leftarrow + \gamma \max_{a'}(Q(s^{next}_\leftarrow, a')) \]

Case: \[ Q(s, \leftarrow) < Q(s, \rightarrow) \]
\[ y = r_\rightarrow + \gamma \max_{a'}(Q(s^{next}_\rightarrow, a')) \]

Backpropagation
Compute \( \frac{\partial L}{\partial W} \) and update W using stochastic gradient descent
**Recap’**

**DQN Implementation:**

- **Initialize your Q-network parameters**

- **Loop over episodes:**
  - **Start from initial state** $s$

- **Loop over time-steps:**
  - **Forward propagate** $s$ **in the Q-network**
  - **Execute action** $a$ (that has the maximum $Q(s,a)$ output of Q-network)
  - **Observe reward** $r$ and next state $s'$
  - **Compute targets** $y$ **by forward propagating state** $s'$ **in the Q-network, then compute loss.**
  - **Update parameters with gradient descent**
I. Motivation

II. Recycling is good: an introduction to RL

III. Deep Q-Networks

IV. Application of Deep Q-Network: Breakout (Atari)

V. Tips to train Deep Q-Network

VI. Advanced topics
Goal: play breakout, i.e. destroy all the bricks.

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
IV. Deep Q-Learning application: Breakout (Atari)

Goal: play breakout, i.e. destroy all the bricks.

Demo

input of Q-network

Output of Q-network

What is done in preprocessing?

- Convert to grayscale
- Reduce dimensions (h,w)
- History (4 frames)

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]

[Video credits to Two minute papers: Google DeepMind’s Deep Q-learning playing Atari Breakout https://www.youtube.com/watch?v=V1eYniJ0Rnk]

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
IV. Deep Q-Learning application: Breakout (Atari)

input of Q-network

\[ \phi(s) = \]

Deep Q-network architecture?

\[ \phi(s) \rightarrow \text{CONV} \rightarrow \text{ReLU} \rightarrow \text{CONV} \rightarrow \text{ReLU} \rightarrow \text{CONV} \rightarrow \text{ReLU} \rightarrow \text{FC (RELU)} \rightarrow \text{FC (LINEAR)} \rightarrow \begin{pmatrix} Q(s,\leftarrow) \\ Q(s,\rightarrow) \\ Q(s,-) \end{pmatrix} \]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
Recap’ (+ preprocessing + terminal state)

DQN Implementation:

- Initialize your Q-network parameters
- Loop over episodes:
  - Start from initial state $s$
  - Loop over time-steps:
    - Forward propagate $s$ in the Q-network $\phi(s)$
    - Execute action $a$ (that has the maximum $Q(s,a)$ output of Q-network) $\phi(s)$
    - Observe reward $r$ and next state $s'$
    - Use $s'$ to create $\phi(s')$
    - Compute targets $y$ by forward propagating state $s'$ in the Q-network, then compute loss $\phi(s')$
    - Update parameters with gradient descent

Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
Recap' (+ preprocessing + terminal state)

DQN Implementation:
- Initialize your Q-network parameters
- Loop over episodes:
  - Start from initial state $s$
  - Loop over time-steps:
    - Forward propagate $s$ in the Q-network
    - Execute action $a$ (that has the maximum $Q(s,a)$ output of Q-network)
    - Observe reward $r$ and next state $s'$
    - **Use $s'$ to create** $\phi(s')$
    - Compute targets $y$ by forward propagating state $s'$ in the Q-network, then compute loss.
    - Update parameters with gradient descent

Some training challenges:
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

[φ($s$)]
[φ($s$)]
[φ($s'$)]
[φ($s'$)]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
**DQN Implementation:**

- Initialize your Q-network parameters
- Loop over episodes:
  - Start from initial state
- Create a boolean to detect terminal states: `terminal = False`
  - Loop over time-steps:
    - Forward propagate `s` in the Q-network
    - Execute action `a` (that has the maximum Q(s,a) output of Q-network)
    - Observe reward `r` and next state `s'`
    - Use `s'` to create `φ(s')`
    - Check if `s'` is a terminal state. Compute targets `y` by forward propagating state `s'` in the Q-network, then compute loss.
- Update parameters with gradient descent

**Some training challenges:**
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

\[
\begin{align*}
\text{if } & \text{terminal} = \text{False} : \quad y = r + \gamma \max_{\hat{a'}} (Q(s', \hat{a'})) \\
\text{if } & \text{terminal} = \text{True} : \quad y = r \quad (\text{break})
\end{align*}
\]
IV - DQN training challenges

Experience replay

Current method is to start from initial state $s$ and follow:

- $E_1$: $\phi(s) \rightarrow a \rightarrow r \rightarrow \phi(s')$
- $E_2$: $\phi(s') \rightarrow a' \rightarrow r' \rightarrow \phi(s'')$
- $E_3$: $\phi(s'') \rightarrow a'' \rightarrow r'' \rightarrow \phi(s''')$

$\ldots$

Training: $E_1 \rightarrow E_2 \rightarrow E_3$

Can be used with mini batch gradient descent

$\phi(s) \rightarrow a \rightarrow r \rightarrow \phi(s')$

Advantages of experience replay?

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
Recap’ (+ experience replay)

**DQN Implementation:**
- Initialize your Q-network parameters
- Initialize replay memory D
- Loop over episodes:
  - Start from initial state $\phi(s)$
  - Create a boolean to detect terminal states: terminal = False
- Loop over time-steps:
  - Forward propagate $\phi(s)$ in the Q-network
  - Execute action $a$ (that has the maximum $Q(\phi(s), a)$ output of Q-network)
  - Observe reward $r$ and next state $s'$
  - Use $s'$ to create $\phi(s')$
  - **Add experience** $(\phi(s), a, r, \phi(s'))$ to replay memory (D)
- Sample random mini-batch of transitions from D
- Check if $s'$ is a terminal state. Compute targets $y$ by forward propagating state $\phi(s')$ in the Q-network, then compute loss.
- Update parameters with gradient descent

**Some training challenges:**
- Keep track of terminal step
- Experience replay
- Epsilon greedy action choice (Exploration / Exploitation tradeoff)

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
Exploration vs. Exploitation

Just after initializing the Q-network, we get:

- $Q(S1, a_1) = 0.5$
- $Q(S1, a_2) = 0.4$
- $Q(S1, a_3) = 0.3$
Exploration vs. Exploitation

Initial state: $S_1$
- $a_1$ leads to $S_2$ with $R = +0$
- $a_2$ leads to $S_3$ with $R = +1$
- $a_3$ leads to $S_4$ with $R = +1000$

Just after initializing the Q-network, we get:

- $Q(S_1, a_1) = 0.5$ (red cross)
- $Q(S_1, a_2) = 0.4$
- $Q(S_1, a_3) = 0.3$
Exploration vs. Exploitation

Just after initializing the Q-network, we get:

- $Q(S1, a_1) = 0$
- $Q(S1, a_2) = 0.4$
- $Q(S1, a_3) = 0.3$
Exploration vs. Exploitation

Just after initializing the Q-network, we get:

\[ Q(S_1, a_1) = 0.5 \]
\[ Q(S_1, a_2) = 0.4 \]
\[ Q(S_1, a_3) = 0.3 \]

Will never be visited, because \( Q(S_1, a_3) < Q(S_1, a_2) \)
Recap’ (+ epsilon greedy action)

**DQN Implementation:**

- Initialize your Q-network parameters
- Initialize replay memory D

- Loop over episodes:
  
  - Start from initial state \( \phi(s) \)
  
  - Create a boolean to detect terminal states: terminal = False
  
  - Loop over time-steps:

    - **With probability epsilon, take random action a.**
    
    - **Otherwise:**
      
      - Forward propagate \( \phi(s) \) in the Q-network
      
      - Execute action a (that has the maximum \( Q(\phi(s),a) \) output of Q-network).

    - Observe reward \( r \) and next state \( s' \)

    - Use \( s' \) to create \( \phi(s') \)

    - Add experience \( (\phi(s), a, r, \phi(s')) \) to replay memory \( (D) \)

    - Sample random mini-batch of transitions from \( D \)

    - Check if \( s' \) is a terminal state. Compute targets \( y \) by forward propagating state \( \phi(s') \) in the Q-network, then compute loss.

    - Update parameters with gradient descent

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
**DQN Implementation:**

- Initialize your Q-network parameters
- **Initialize replay memory** $D$
- Loop over episodes:
  - Start from initial state $\phi(s)$
  - Create a boolean to detect terminal states: $\text{terminal} = \text{False}$
  - Loop over time-steps:
    - **With probability epsilon, take random action** $a$.
    - Otherwise:
      - Forward propagate $\phi(s)$ in the Q-network
      - Execute action $a$ (that has the maximum $Q(\phi(s), a)$ output of Q-network).
    - Observe rewards $r$ and next state $s'$
    - **Use $s'$ to create** $\phi(s')$
    - **Add experience** $(\phi(s), a, r, \phi(s'))$ to replay memory $(D)$
    - Sample random mini-batch of transitions from $D$
    - **Check if $s'$ is a terminal state.** Compute targets $y$ by forward propagating state $\phi(s')$ in the Q-network, then compute loss.
    - Update parameters with gradient descent

**Preprocessing**

- Detect terminal state
- Experience replay
- Epsilon greedy action

---

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
Results

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
[Credits: DeepMind, DQN Breakout - https://www.youtube.com/watch?v=TmPfTpjtdgg]
Difference between with and without human knowledge

Imitation learning

[Source: Bellemare et al. (2016): Unifying Count-Based Exploration and Intrinsic Motivation]

[Ho et al. (2016): Generative Adversarial Imitation Learning]
Other Atari games

Pong

SeaQuest

Space Invaders

[Chia-Hsuan Lee, Atari Seaquest Double DQN Agent - https://www.youtube.com/watch?v=NirMkC5uvWU]

[mooopan, Deep Q-Network Plays Atari 2600 Pong - https://www.youtube.com/watch?v=p88R2_3yWPA]

[DeepMind: DQN SPACE INVADERS - https://www.youtube.com/watch?v=W2CAghUiofY&t=2s]

[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
I. Motivation
II. Recycling is good: an introduction to RL
III. Deep Q-Networks
IV. Application of Deep Q-Network: Breakout (Atari)
V. Tips to train Deep Q-Network
VI. Advanced topics
Alpha Go

[Silver, Schrittwieser, Simonyan et al. (2017): Mastering the game of Go without human knowledge]
VI - Advanced topics

Policy Gradient Methods

PPO

[Open AI Blog]

TRPO

[TRPO]

[Schulman et al. (2017): Proximal Policy Optimization]
[Schulman et al. (2017): Trust Region Policy Optimization]
VI - Advanced topics

Competitive self-play

[Bansal et al. (2017): Emergent Complexity via multi-agent competition]
[OpenAI Blog: Competitive self-play]
Meta learning

VI - Advanced topics

Exploration in Reinforcement Learning

[Bellemare et al. (2017): Unifying Count-Based Exploration and Intrinsic Motivation]
VI - Advanced topics

Imitation learning

[Ho et al. (2016): Generative Adversarial Imitation Learning]
Announcements

For Wednesday 12/05, 11am:
  • No assignment this week, but work on your projects and meet your mentors!

This Friday:
  • TA Sections: reading research papers.
VI - Advanced topics

Imitation learning

[Source: Bellemare et al. (2016): Unifying Count-Based Exploration and Intrinsic Motivation]

[Ho et al. (2016): Generative Adversarial Imitation Learning]