CS230: Lecture 9
Deep Reinforcement Learning
Kian Katanforoosh
Menti code: 80 24 08
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
I. Motivation

Human Level Control through Deep Reinforcement Learning

Silver et al. (2017): Mastering the game of Go without human knowledge
Mnih et al. (2015): Human Level Control through Deep Reinforcement Learning

AlphaGo

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

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

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

<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></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Goal: maximize the return (rewards)

Number of states: 5

Types of states:

- initial
- normal
- terminal

Agent’s Possible actions:

Define reward “r” in every state

- State 1: +2
- State 2 (initial): 0
- State 3: 0
- State 4: +1
- State 5: +10

How to define the long-term return?

Discounted return

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

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

Best strategy to follow if \( \gamma = 1 \)
**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>![Chocolate Bar]</td>
<td>![Recycle Bin]</td>
<td>![Item]</td>
<td>![Item]</td>
</tr>
</tbody>
</table>

**What do we want to learn?**

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

**Q-table**

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

**Define reward “r” in every state**

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

**Assuming \( \gamma = 0.9 \)**

**Discounted return**

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

**How?**

- S1 → S2 (gain +2)
- S2 → S3 (gain +0)
- S3 → S4 (gain +1)
- S4 → S5 (gain +10)
II. Recycling is good: an introduction to RL

Problem statement

State transition diagram:

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

Define reward "r" in every state:

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

Assuming \( \gamma = 0.9 \)

Discounted return:

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

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?

- States
- Actions
- Discount factor \( \gamma \)

States: S1, S2, S3, S4, S5

Actions: 1, 2

Discounted return:

\[
R = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots
\]

\[
= r_0 + \gamma (r_1 + \gamma r_2 + \ldots)
\]

\[
= r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots
\]

\[
= 0 + 10 \times 0.9
\]

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

Problem statement

Define reward “r” in every state

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?

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

What do we want to learn?

Define reward “r” in every state

Assuming $\gamma = 0.9$

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

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

**How?**

Assuming \( \gamma = 0.9 \).
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

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

How?

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

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

#states #actions

How?

S1

S2

S3

S4

S5

S1

S2

S3

S4

S5

+2

+8.1

+9

+10

+10

+9

+10

+8.1

+8.1

+9

+9

+9

+9

+9

+9
II. Recycling is good: an introduction to RL

Problem statement

What do we want to learn?

Define reward “r” in every state

Best strategy to follow if $\gamma = 0.9$

When state and actions space are too big, this method has huge memory cost

Function telling us our best strategy

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
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-Networks
IV. Application of Deep Q-Network: Breakout (Atari)
V. Tips to train Deep Q-Network
VI. Advanced topics
III. Deep Q-Networks

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

Problem statement

Neural Network

Q-table

Then compute loss, backpropagate.

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

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

Loss function

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

Target value

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

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

Immediate reward for taking action $$\leftarrow$$ in state $$s$$

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

Hold fixed for backprop

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

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

Immediate Reward for taking action $$\rightarrow$$ in state $$s$$

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

Hold fixed for backprop
III. Deep Q-Networks

\[ s = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \]

\[ Q(s, \leftarrow) \]

\[ Q(s, \rightarrow) \]

**Loss function (regression)**

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

**Target value**

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

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

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

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

**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 rewards** $r$ **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
IV. Deep Q-Networks application: Breakout (Atari)

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

Demo

Input of Q-network

Output of Q-network

Would that work?

https://www.youtube.com/watch?v=V1eYniJ0Rnk
IV. Deep Q-Networks application: Breakout (Atari)

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

Demo

What is done in preprocessing?

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

Preprocessing

\[ \phi(s) \]

Input of Q-network

Output of Q-network

\[ \begin{pmatrix} Q(s, \leftarrow) \\ Q(s, \rightarrow) \\ Q(s, -) \end{pmatrix} \]
IV. Deep Q-Networks application: Breakout (Atari)

input of Q-network

\[ \phi(s) = \]

Deep Q-network architecture?

\[ \phi(s) \rightarrow \text{CONV, ReLU} \rightarrow \text{CONV, ReLU} \rightarrow \text{CONV, ReLU} \rightarrow \text{FC (RELU), FC (LINEAR)} \rightarrow \begin{pmatrix} Q(s,\leftarrow) \\ Q(s,\rightarrow) \\ Q(s,\rightarrow) \end{pmatrix} \]
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 rewards $r$ 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)
**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 rewards $r$ 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)
Recap’ (+ preprocessing + terminal state)

**DQN Implementation:**
- Initialize your Q-network parameters
- Loop over episodes:
  - Start from initial state \(\phi(s)\)
  - 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 rewards \(r\) and next state \(s'\)
    - Use \(s'\) to create \(\phi(s')\)
    - **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)

\[
\begin{align*}
\text{if } terminal = False \quad & y = r + \gamma \max_a (Q(s', a')) \\
\text{if } terminal = True \quad & y = r \quad \text{(break)}
\end{align*}
\]
Experience replay

Current method is to start from initial state \( s \) and follow:

\[
\phi(s) \rightarrow a \rightarrow r \rightarrow \phi(s')
\]

\[
\phi(s') \rightarrow a' \rightarrow r' \rightarrow \phi(s'')
\]

\[
\phi(s'') \rightarrow a'' \rightarrow r'' \rightarrow \phi(s''')
\]

\[
\ldots
\]

Training: \( E1 \rightarrow E2 \rightarrow E3 \)

Can be used with mini batch gradient descent

Experience Replay

Training:

\[
E1 \rightarrow \text{sample}(E1, E2) \rightarrow \text{sample}(E1, E2, E3) \rightarrow \text{sample}(E1, E2, E3, E4) \rightarrow \ldots
\]

Advantages of experience replay?
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: \(\text{terminal} = \text{False}\)
  - 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 rewards \(r\) and next state \(s'\)
    - Use \(s'\) to create \(\phi(s')\)
    - **Add experience** \((\phi(s), a, r, \phi(s'))\) to replay memory \((D)\)

**Some training challenges:**

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

The transition resulting from this is added to \(D\), and will not always be used in this iteration’s update!

**Update using sampled transitions**

- 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
Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri

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

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

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

Terminal state

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: $\text{terminal} = \text{False}$
  - 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
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
Results

[https://www.youtube.com/watch?v=TmPfTpjtdgg]
Other Atari games

Pong

SeaQuest

Space Invaders

[https://www.youtube.com/watch?v=p88R2_3yWPA]
[https://www.youtube.com/watch?v=W2CAghUiofY&t=2s]
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
VI - Advanced topics

Alpha Go

DeepMind Blog
(Silver et al. (2017): Mastering the game of Go without human knowledge)
VI - Advanced topics

Competitive self-play

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

Meta learning

VI - Advanced topics

Imitation learning

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

[Source: Bellemare et al. (2016): Unifying Count-Based Exploration and Intrinsic Motivation]
For Tuesday 06/05, 9am:

This Friday:
  • TA Sections:
    • How to have a great final project write-up.
      • Advices on: How to write a great report.
      • Advices on: How to build a super poster.
      • Advices on: Final project grading criteria.
    • Going through examples of great projects and why they were great.
  • Small competitive quiz in section.