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

AlphaGo

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

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**

| +2 | 0 | 0 | +1 | +10 |

**Best strategy to follow if** \( \gamma = 1 \)

**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

+2 0 0 +1 +10

Assuming $\gamma = 0.9$

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

What do we want to learn?

How good is it to take action 1 in state 2

Q-table

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?

Define reward “r” in every state

Assuming $\gamma = 0.9$

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

How?
II. Recycling is good: an introduction to RL

Problem statement

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

START

Define reward “r” in every state

<table>
<thead>
<tr>
<th>State</th>
<th>Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>+1</td>
</tr>
<tr>
<td>5</td>
<td>+10</td>
</tr>
</tbody>
</table>

How good is it to take action 1 in state 2

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 to learn Q-table

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

\( \gamma = 0.9 \)
II. Recycling is good: an introduction to RL

**Problem statement**

State 1 | State 2 (initial) | State 3 | State 4 | State 5
---|---|---|---|---
[Trash] START | [Chocolate] | [Recycle] | | 

Define reward “r” in every state

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+10</td>
</tr>
</tbody>
</table>

**What do we want to learn?**

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

**Q-table**

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

**How?**

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

**Discounted return**

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

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>🗑️ START</td>
<td>🍫</td>
<td>🌿</td>
<td>+10</td>
<td></td>
</tr>
</tbody>
</table>

**Define reward “r” in every state**

<table>
<thead>
<tr>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
<th>State 4</th>
<th>State 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+10</td>
</tr>
</tbody>
</table>

**What do we want to learn?**

**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 good is it to take action 1 in state 2?

**How?**

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

**Discounted return**

\[R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots\]
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>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+10</td>
</tr>
</tbody>
</table>

What do we want to learn?

How good is it to take action 1 in state 2

Q-table

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

How?

Assuming \( \gamma = 0.9 \)

Discounted return

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

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

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-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

**Q-table**

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

Then compute loss, backpropagate.

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

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

\[
\begin{align*}
&\begin{cases} a_1^{[1]} \rightarrow Q(s, \leftarrow) \\
&\begin{cases} a_1^{[2]} \rightarrow Q(s, \rightarrow) \\
&\begin{cases} a_1^{[3]} \rightarrow Q(s, \leftarrow) \\
&\begin{cases} a_1^{[4]} \rightarrow Q(s, \rightarrow) \\
\end{cases}
\end{cases}
\end{cases}
\end{cases}
\end{align*}
\]

Loss function

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

Target value

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

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

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

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

Hold fixed for backprop

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

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

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

Discounted maximum future reward when you are in state \(s^{\text{next}}\)
III. Deep Q-Networks

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

\[
\begin{array}{c}
\left( a_1^{[1]} \right) \\
\left( a_1^{[2]} \right) \\
\left( a_3^{[1]} \right) \\
\left( a_3^{[2]} \right) \\
\left( a_4^{[1]} \right)
\end{array}
\xrightarrow{	ext{\hspace{1cm} \rightarrow \hspace{1cm} Q(s,\leftarrow)}}
\left( a_1^{[3]} \right) \rightarrow Q(s,\rightarrow)
\]

\text{Target value}
\hspace{1cm}
\text{Case: } Q(s,\leftarrow) > Q(s,\rightarrow)
\hspace{1cm}
\text{Case: } Q(s,\leftarrow) < Q(s,\rightarrow)

\[
y = r_\leftarrow + \gamma \max_{a'} (Q(s_{\leftarrow}^{\text{next}}, a'))
\]
\hspace{3cm}
\[
y = r_\rightarrow + \gamma \max_{a'} (Q(s_{\rightarrow}^{\text{next}}, a'))
\]

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

\text{Backpropagation}
\hspace{1cm}
\text{Compute } \frac{\partial L}{\partial W} \text{ and update } W \text{ 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
Goal: play breakout, i.e. destroy all the bricks.

Demo

Input of Q-network

Output of Q-network

$$Q(s, \leftarrow)$$

$$Q(s, \rightarrow)$$

$$Q(s, \uparrow)$$

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)

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

input of Q-network

\[ \phi(s) = \]

Deep Q-network architecture?

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

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
**Recap’ (+ preprocessing + terminal state)**

**DQN Implementation:**
- Initialize your Q-network parameters
- Loop over episodes:
  - Start from initial state $s$
  - 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)

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

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

\[
\begin{align*}
\text{E1: } & \phi(s) \rightarrow a \rightarrow r \rightarrow \phi(s') \\
\text{E2: } & \phi(s') \rightarrow a' \rightarrow r' \rightarrow \phi(s'') \\
\text{E3: } & \phi(s'') \rightarrow a'' \rightarrow r'' \rightarrow \phi(s''') \\
& \ldots
\end{align*}
\]

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

Can be used with mini batch gradient descent

Experience Replay

1 experience (leads to one iteration of gradient descent)

\[
\begin{align*}
\text{Training: } & \ E1 \rightarrow \text{sample}(E1, E2) \rightarrow \text{sample}(E1, E2, E3) \\
& \rightarrow \text{sample}(E1, E2, E3, E4) \rightarrow \ldots
\end{align*}
\]

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: 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 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

**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!
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:

\[
Q(S1, a_1) = 0.5 \\
Q(S1, a_2) = 0.4 \\
Q(S1, a_3) = 0.3
\]

Initial state

S1

Terminal state

S2

R = +0

S3

R = +1

S4

R = +1000
Exploration vs. Exploitation

Just after initializing the Q-network, we get:

\[
\begin{align*}
Q(S_1, a_1) &= 0.5 \\
Q(S_1, a_2) &= 0.4 \\
Q(S_1, a_3) &= 0.3
\end{align*}
\]
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 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
Overall recap’

**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

---

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
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=NirMkC5uvWU]
[https://www.youtube.com/watch?v=W2CAghUiofY&t=2s]
Today’s outline

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

Meta learning

VI - Advanced topics

Imitation learning

[Ho et al. (2016): Generative Adversarial Imitation Learning]
VI - Advanced topics

Auxiliary task

Kian Katanforoosh, Andrew Ng, Younes Bensouda Mourri
Announcements

Check out the project example code (cs230-stanford.github.io)

For Thursday 03/15, 9am:

C5M3
• Quiz: Sequence models & attention mechanism
• Programming Assignment: Neural Machine Translation with Attention
• Programming Assignment: Trigger word detection

Final project and posters!