CS230: Lecture 5
Deep Reinforcement Learning
Kian Katanforoosh
Today’s outline

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

Human Level Control through Deep Reinforcement Learning

AlphaGo

Mastering the Game of Go without Human Knowledge

DeepMind, 5 New Street Square, London EC4A 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 in 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 AlphaZero achieved superhuman performance, winning 100-0 against the previously published, champion-defeating AlphaGo.

AlphaStar

[Silver, Schrittwieser, Simonyan et al. (2017): Mastering the game of Go without human knowledge]
[Silver, Schrittwieser, Simonyan et al. (2017): Mastering the game of Go without human knowledge]
[Mnih, Kavukcuoglu, Silver 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 decisions
- Teaching by experience vs. Teaching by example.

**Examples of RL applications**
- Games
- Robotics
- Advertisement

---

**How would you solve Go with classic supervised learning?**

Input

- The move of the professional player

output class

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/](https://deepmind.com/blog/alphago-zero-learning-scratch/)
I. Motivation

Transition: $s_t \rightarrow a_t \rightarrow (o_t, r_t) \rightarrow s_{t+1}$
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
S1 S2 S3 S4 S5

Assuming $\gamma = 0.9$

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

What do we want to learn?

how good is it to take action 1 in state 2

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

Q-table

How?

Kian Katanforoosh
II. Recycling is good: an introduction to RL

Problem statement

What do we want to learn?

Define reward “r” in every state

State 1: S1 = +2
State 2 (initial): S2 = 0
State 3: S3 = 0
State 4: S4 = +1
State 5: S5 = +10

Assuming $\gamma = 0.9$

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

How?

Q-table

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

How good is it to take action 1 in state 2

Assuming $\gamma = 0.9$. How good is it to take action 1 in state 2?
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-table

Assuming $\gamma = 0.9$

Discounted return $R = \sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$
**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>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
<td>S5</td>
</tr>
<tr>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+10</td>
</tr>
</tbody>
</table>

Assuming $\gamma = 0.9$

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?**

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

Assuming $\gamma = 0.9$

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

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

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
II. Recycling is good: an introduction to RL

Problem statement

Define reward “r” in every state

Assuming $\gamma = 0.9$

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

Q-table

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?

Kian Katanforoosh
II. Recycling is good: an introduction to RL

Problem statement

What do we want to learn?

Define reward “r” in every state

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

Assuming \( \gamma = 0.9 \)

Q-table

States:
- State 1
- State 2 (initial)
- State 3
- State 4
- State 5

Values:
- \( Q = \begin{pmatrix}
0 & 0 & 9 & 8.1 & 10 \\
2 & 9 & 9 & 10 & 0 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix} \)

How good is it to take action 1 in state 2

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

#actions

#states

1. \( \gamma = 0.9 \)
2. \( \gamma = 0.9 \times 9 \)
3. \( \gamma = 0.9 \times 10 \)
4. \( \gamma = 1.0 \times 0.9 \)
5. \( \gamma = 1.0 \times 10 \)

Kian Katanforoosh
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 $\gamma = 0.9$
- How good is it to take action 1 in state 2

Bellman equation (optimality equation)

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

Policy

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

When state and action spaces are too large, this method has huge memory cost

Kian Katanforoosh

Function telling us our best strategy
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

<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>🗑️</td>
<td>START</td>
<td>🍬</td>
<td>📜</td>
<td>🗑️</td>
</tr>
</tbody>
</table>

Q-table

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

#actions  #states

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**

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

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

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

- Immediate reward for taking action \( \leftarrow \) in state \( s \)
- Discounted maximum future reward when you are in state \( s_{next}^{\leftarrow} \)
- Hold fixed for backprop

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

- Immediate Reward for taking action \( \rightarrow \) in state \( s \)
- Discounted maximum future reward when you are in state \( s_{next}^{\rightarrow} \)
- Hold fixed for backprop

Target value

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

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

---

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

**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

Kian Katanforoosh
**Recap’**

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

**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
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
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

Q-values

\[ Q(s, \leftarrow) \]
\[ Q(s, \rightarrow) \]
\[ Q(s, \rightarrow) \]

Would that work?

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

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

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

IV. Deep Q-Learning application: Breakout (Atari)

Demo

What is done in preprocessing?
- Convert to grayscale
- Reduce dimensions (h,w)
- History (4 frames)

input of Q-network

Output of Q-network

Q-values

\[
\begin{pmatrix}
Q(s, \leftarrow) \\
Q(s, \rightarrow) \\
Q(s, \rightarrow)
\end{pmatrix}
\]

[Video credits to Two minute papers: Google DeepMind’s Deep Q-learning playing Atari Breakout https://www.youtube.com/watch?v=V1eYniJ0Rnk]
[Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning]
IV. Deep Q-Learning application: Breakout (Atari)

input of Q-network

\[ \phi(s) = \]

Deep Q-network architecture?

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

[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)

---

\[ \phi(s) \]
\[ \phi(s') \]

---

[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)

[<span class="citation">
Mnih, Kavukcuoglu, Silver et al. (2015): Human Level Control through Deep Reinforcement Learning
</span>]
**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 reward $r$ 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)

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

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

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

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

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

$\ldots$

Training: $E_1 \rightarrow \text{sample}(E_1, E_2) \rightarrow \text{sample}(E_1, E_2, E_3) \rightarrow \text{sample}(E_1, E_2, E_3, E_4) \rightarrow \ldots$

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)

[Update using sampled transitions]

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

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

Just after initializing the Q-network, we get:

\[ Q(S_1, a_1) = 0.5 \]
\[ Q(S_1, a_2) = 0.4 \]
\[ Q(S_1, a_3) = 0.3 \]
Just after initializing the $Q$-network, we get:

$$Q(S_1, a_1) = 0.5$$
$$Q(S_1, a_2) = 0.4$$
$$Q(S_1, a_3) = 0.3$$
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

$S1$

$S2$

$R = +0$

$R = +1$

$R = +1000$

Just after initializing the $Q$-network, we get:

$Q(S1, a_1) = 0.5$

$Q(S1, a_2) = 0.4$

$Q(S1, a_3) = 0.3$

Will never be visited, because $Q(S1, a_3) < Q(S1, a_2)$
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 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

[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]
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]

Kian Katanforoosh
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]
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

Policy Gradient Methods

PPO

TRPO

[Schulman et al. (2017): Trust Region Policy Optimization]

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

Competitive self-play

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

Open AI Five

Deep Mind: Alpha Star

[OpenAI Blog Five]

AlphaStar: Mastering the Real-Time Strategy Game StarCraft
VI - Advanced topics

Alpha Go

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

What’s up next week:

Complete modules:
- C4M1: Foundations of Convolutional Neural Network (slides)
- C4M2: Deep Convolutional Models (slides)

Quizzes (due at 8:30am PST):
- The basics of ConvNets
- Deep convolutional models

Programming Assignments (due at 8:30am PST):
- Convolutional Model: step by step
- Convolutional Model: application
- Keras Tutorial: This assignment is optional.
- Residual Networks

Discussion section this Friday