Deep Q-Network

Alina Vereshchaka

University at Buffalo
avereshc@buffalo.edu

June 11, 2019

*Slides are based on David Silver’s Deep RL talk
Deep Q-Network
Deep Q-Network

- Learn to master 49 different Atari games from screens
- Excel human experts in 29 games
- Uses Deep Q-network receiving only the pixels and the game score as inputs
Recap: Q-Learning Algorithm

- Q-learning learns the action-value function $Q(s, a)$: how good to take an action at a particular state.
- From the memory table, we determine the next action $a'$ to take which has the maximum $Q(s', a')$.

![Diagram showing state transitions and action selection]

select an action with highest Q
Recap: Q-Learning Algorithm

Loop for each step of episode:
Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
Take action $A$, observe $R, S'$

\[
Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_a Q(S', a) - Q(S, A) \right]
\]

$S \leftarrow S'$

until $S$ is terminal
Recap: Exploration vs Exploitation

- Exploration
  - Discover better action selections
  - Improve the knowledge about the environment
  - $\epsilon$-greedy exploration – with probability $\epsilon$ choose a random action, otherwise go with the “greedy” action with the highest Q-value.

- Exploitation
  - Maximize the reward based on what agent already knows
Exploration vs Exploitation

$\epsilon$ – greedy algorithm:

- With probability $\epsilon$ choose a random action $a$
- With probability $1 - \epsilon$ choose “greedy” action $a$ with the highest Q-value.
Reinforcement Learning (RL) defines the **objective**

Deep Learning (DL) gives the **mechanism**

\[
\text{RL + DL} = \text{General Intelligence}
\]
Deep Q-Network (DQN)

- Represent value function by deep Q-network with weights $w$

$$Q(s, a, w) \approx Q^\pi(s, a)$$
Deep Q-Network (DQN)

- Represent value function by deep Q-network with weights $w$

\[ Q(s, a, w) \approx Q^\pi(s, a) \]

- Define objective function

\[
\mathcal{L}(w) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]
\]
Deep Q-Network (DQN)

- Represent value function by deep Q-network with weights $w$
  \[ Q(s, a, w) \approx Q^\pi(s, a) \]

- Define objective function
  \[
  \mathcal{L}(w) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]
  \]

- Leading to the following Q-learning gradient
  \[
  \frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]
  \]
Deep Q-Network (DQN)

- Represent value function by deep Q-network with weights $w$
  \[ Q(s, a, w) \approx Q^\pi(s, a) \]

- Define objective function
  \[
  L(w) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]
  \]

- Leading to the following Q-learning gradient
  \[
  \frac{\partial L(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]
  \]

- Optimize objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$
Stability issues with Deep RL

Naive Q-learning oscillates or diverge with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid

2. Policy changes rapidly with slight changes to Q-values
   - Policy can oscillate
   - Distribution of data can swing from one extreme to another

3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large unstable when backpropagated
Recap: Independent and Identically Distributed (iid) data

In supervised learning, we want the input to be i.i.d. (independent and identically distributed):

- Samples are randomized among batches and therefore each batch has the same (or similar) data distribution
- Samples are independent of each other in the same batch. If not, the model may be overfitted for some class (or groups) of samples at different time and the solution will not be generalized.
Deep Q-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies

2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target

3. Clip rewards or normalize network adaptive to sensible range
   - Robust gradients
Problem: approximation of Q-values using non-linear functions is not stable

Solution:
- Take action $a_t$ according to $\epsilon$-greedy policy
**Problem:** approximation of Q-values using non-linear functions is not stable

**Solution:**
- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in a replay memory $D$
Problem: approximation of Q-values using non-linear functions is not stable
Solution:
- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in a replay memory $D$
- Sample random mini-batch of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ from $D$
Stable Deep RL (1): Experience Replay

**Problem:** approximation of Q-values using non-linear functions is not stable

**Solution:**

- Take action $a_t$ according to $\epsilon$-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in a replay memory $D$
- Sample random mini-batch of transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ from $D$
- Optimize MSE between Q-network and Q-learning targets, e.g.

$$
\mathcal{L}(w) = \mathbb{E}_{s, a, r, s' \sim D} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]
$$

This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum.
**Problem:** approximation of Q-values using non-linear functions is not stable

**Solution:**

\[ s_t, a_t, r_{t+1}, s_{t+1} \]
Create two deep networks $w^-$ and $w$

Use the first one to retrieve $Q$ values while the second one includes all updates in the training. After $C$ updates synchronize $w^- \leftarrow w$.

Motivation: Fix the Q-value targets temporarily so we don’t have a moving target.
Stable Deep RL (2): Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters $w^-$

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

- Optimize MSE between Q-network and Q-learning targets

$$ \mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim D} \left[ \left( r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right] $$

- Periodically update fixed parameters $w^- \leftarrow w$
DQN clips the reward $[-1, +1]$

This prevents Q-values from becoming too large

Ensures gradients are well-conditioned

Can’t tell difference between small and large rewards
Deep Q-Network (DQN) Architecture

Naive DQN

```
<table>
<thead>
<tr>
<th>State</th>
<th>Q-value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Optimized DQN used by DeepMind

```
<table>
<thead>
<tr>
<th>Q-value 1</th>
<th>Q-value 2</th>
<th>Q-value n</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
**Reinforcement Learning in Atari**

**Frame**: snapshot of the environment state at every point

**Action**: a set of actions that the agent can take (0, 1, 2, 3)

**Score**: evaluation metric

**Number of “lives”**: for each game (initially 5)

**Game’s level**
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step
Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ do

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ do

With probability $\epsilon$ select a random action $a_t$
otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
Every $C$ steps reset $\hat{Q} = Q$

End For

End For
Demo for Project 4, CSE 574 Machine Learning, Fall 2018 [Instructor: Dr. Sargur N. Srihari]
Authors: Alina Vereshchaka and Nathan Margaglio

Tom & Jerry
DQN Example

State

Fruit_y
Fruit_x
Player_y
Player_x

Input
4 x #samples

Hidden Layer 1
#Weights: 128x6
ReLU

Hidden Layer 2
#Weights: 128x128
ReLU

Output
4 x #samples
#Weights: 4x128
Linear

Q(s_t, a_1)
Q(s_t, a_2)
Q(s_t, a_3)
Q(s_t, a_4)
DQN Summary

- DQN: Q-Learning but with a Deep Neural Network as a function approximator
- Using a non-linear Deep Neural Network is powerful, but training is unstable if we apply it naively
- Experience Replay Trick: Store experience \((S, A, R, S_{next})\) in a replay buffer and sample minibatches from it to train the network. This decorrelates the data and leads to better data efficiency. In the beginning, the replay buffer is filled with random experience
- By using a Convolutional Neural Network as the function approximator on raw pixels of Atari games where the score is the reward we can learn to play many of those games at human-like performance.