Non-linear Value Function Approximation: Deep Q-Network

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Spring 2020

avereshc@buffalo.edu

March 11, 2020

Overview

1 Recap: Q-Learning Algorithm

2 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 a state, action, and reward system, with an arrow indicating the selection of an action with the highest Q-value.]
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.
<table>
<thead>
<tr>
<th></th>
<th>Recap: Q-Learning Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Deep Q-Network</td>
</tr>
</tbody>
</table>
Deep Q-Network (DQN): AI = RL + DL

- Reinforcement Learning (RL) defines the **objective**
- Deep Learning (DL) gives the **mechanism**

**RL + DL = 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 the 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 the 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]
  
Alina Vereshchaka (UB)  
CSE4/510 Reinforcement Learning, Lecture 12  
March 11, 2020 9 / 35
Deep Q-Network (DQN)

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

- Define the 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
  \[ \frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right] \]

- Optimize objective end-to-end by SGD, using $\frac{\partial \mathcal{L}(w)}{\partial w}$
Supervised SGD vs Q-Learning SGD

- SGD update assuming supervision

\[ J(\mathbf{w}) = \mathbb{E}_\pi \left[ (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right] \]

\[ \Delta \mathbf{w} = \alpha(q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))\nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) \]
Supervised SGD vs Q-Learning SGD

- SGD update assuming supervision

\[ J(\mathbf{w}) = \mathbb{E}_\pi \left[ (q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right] \]

\[ \Delta \mathbf{w} = \alpha (q_{\pi}(S, A) - \hat{q}(S, A, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S, A, \mathbf{w}) \]

\[ Q(S_1, A_1) := Q(S_1, A_1) + \alpha \left( R_2 + \gamma \max_{a_2'} Q(S_2, a_2') - Q(S_1, A_1) \right) \]
Supervised SGD vs Q-Learning SGD

- SGD update assuming supervision

\[ J(w) = \mathbb{E}_\pi [(q_\pi(S, A) - \hat{q}(S, A, w))^2] \]

\[ \Delta w = \alpha (q_\pi(S, A) - \hat{q}(S, A, w)) \nabla_w \hat{q}(S, A, w) \]

- SGD update for Q-Learning

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

\[ \Delta w = \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \]

\[ Q(S_1, A_1) := Q(S_1, A_1) + \alpha \left( R_2 + \gamma \max_{a_2'} Q(S_2, a_2') - Q(S_1, A_1) \right) \]
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
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
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
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 $D$
**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:
Create two deep networks $w^{-}$ and $w$
- 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 $[-1, +1]$
DQN clips the reward \([-1, +1]\)

This prevents Q-values from becoming too large
DQN clips the reward $[-1, +1]$

This prevents Q-values from becoming too large

Ensures gradients are well-conditioned
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

Optimized DQN used by DeepMind
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

Agent
Reinforcement Learning in Atari
Do we have all the information to start training?
Reinforcement Learning in Atari

We know the direction and the velocity of the ball. But do we know its acceleration?
We know the direction and the velocity of the ball. But do we know its acceleration?
Now we can extract all the information about the state.
To make sure we can generalize for other games as well, we keep 4 frames as an input.
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 a change in score for that step
DQN in Atari

1) Input:
4 images = current frame + 3 previous

2) Output: $Q(s,a_1)$
$Q(s,a_2)$
$Q(s,a_3)$
$\ldots$
$Q(s,a_{18})$
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
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 $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
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 $a_t = \text{argmax}_a Q(\phi(s_t),a; \theta)$

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 $\phi_{t+1} = \phi(s_{t+1})$
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 $D$
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory \( D \) to capacity \( N \)
Initialize action-value function \( \hat{Q} \) with random weights \( \theta \)
Initialize target action-value function \( Q \) 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
Tom & Jerry
DQN Example

```
State

Fruit_y → Input
Fruit_x → 4 x #samples
Player_y → Hidden Layer 1
Player_x → #Weights: 128x6
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
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.
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.
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.