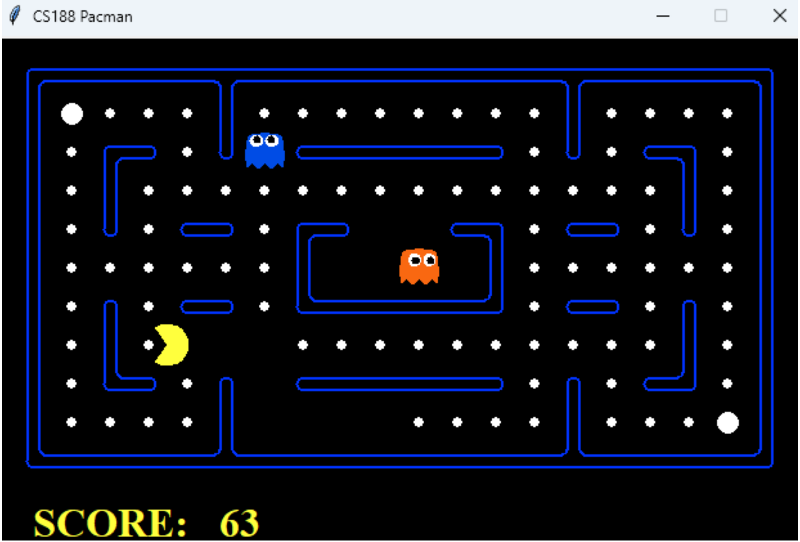
Reinforcement Learning for Business, Economics, and Social Sciences

Unit 4-3: Deep Q-Learning

Davud Rostam-Afschar (Uni Mannheim)

How to learn in very large state-action spaces?

Approximate Q-Learning



Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

State Definition

The MDP **state** is the *entire game configuration* after a full ply:

- Five binary flags:
  - Pac-Man on field?
  - Ghost on field?
  - Food on field?
  - Power-pill on field?
  - Wall on field?
- A "scared ghost" timer taking 40 integer values $0, 1, \ldots, 39$

Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

State Definition

The MDP **state** is the *entire game configuration* after a full ply:

- Five binary flags:
  - Pac-Man on field?
  - Ghost on field?
  - Food on field?
  - Power-pill on field?
  - Wall on field?
- A "scared ghost" timer taking 40 integer values $0, 1, \ldots, 39$
  $2^5 \times 40 \ = \ 32 \times 40 \ = \ 1280$ states per field

Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

State Definition

The MDP **state** is the *entire game configuration* after a full ply:

- Five binary flags:
    - Pac-Man on field?
    - Ghost on field?
    - Food on field?
    - Power-pill on field?
    - Wall on field?
- A "scared ghost" timer taking 40 integer values $0, 1, \ldots, 39$
  $2^5 \times 40 = 32 \times 40 = 1280$ states per field

Maze has $20 \times 11 = 220$ fields $\rightarrow$ a state is one out of $220 \times 1280 = 281,600$ configurations (many of which are impossible).

Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

State Definition

The MDP **state** is the *entire game configuration* after a full ply:

- Five binary flags:
  - Pac-Man on field?
  - Ghost on field?
  - Food on field?
  - Power-pill on field?
  - Wall on field?
- A "scared ghost" timer taking 40 integer values $0, 1, \ldots, 39$
  $2^5 \times 40 = 32 \times 40 = 1280$ states per field

Maze has $20 \times 11 = 220$ fields $\rightarrow$ a state is one out of $220 \times 1280 = 281,600$ configurations (many of which are impossible).

Each board configuration is a separate state with separate Q-values.

Pacman as a Markov Decision Process

Action

- Pacman's action: one of `North`, `South`, `East`, `West`

State Definition

The MDP **state** is the *entire game configuration* after a full ply:

- Five binary flags:
  - Pac-Man on field?
  - Ghost on field?
  - Food on field?
  - Power-pill on field?
  - Wall on field?
- A "scared ghost" timer taking 40 integer values $0, 1, \ldots, 39$
  $2^5 \times 40 \ = \ 32 \times 40 \ = \ 1280$ states per field

Maze has $20 \times 11 \ = \ 220$ fields $\rightarrow$ a state is one out of $220 \times 1280 \ = \ 281,600$ configurations (many of which are impossible).

Pacman has no way to generalize that running into a ghost is bad for all positions.

Deep Q-Networks

- Value or Q-Function Approximation
  - Linear approximation
  - Neural network approximation $\rightarrow$ Deep Q-network

(Goodfellow, 2016)

Q-function Approximation

- Let $s = (x_1, x_1, \ldots, x_n)$
- Linear

$$Q(s, a) \approx \sum_i w_{ai} x_i$$

- Non-linear (e.g., neural network)

$$Q(s, a) \approx g(\boldsymbol{x}; \boldsymbol{w})$$

Gradient Q-learning?

Gradient Q-learning

- Minimize squared error between Q-value estimate and target
  - Q-value estimate: $Q_{\mathbf{w}}(s, a)$
  - Target: $r + \gamma \max_{a'} Q_{\bar{w}}(s', a')$
- Squared error:

$$\text{Err}(\boldsymbol{w}) = 1/2 \left[Q_{w}(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a') \right]^2,$$

where $\overline{\boldsymbol{w}}$ is treated fixed.

- Gradient

$$\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s', a') \right] \frac{\partial Q_w(s, a)}{\partial \boldsymbol{w}}$$

Gradient Q-learning

Gradient Q-learning (s, Q^*)
Initialize weights $\boldsymbol{w}$ uniformly at random in [-1,1]
Observe current state s
Loop

- Select action a and execute it
- Receive immediate reward r
- Observe new state s'
- Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s', a')] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$
- Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$
- Update state: $s \leftarrow s'$

Until convergence of Q^*
Return Q^*

- Tabular Q-Learning converges to optimal Q-function under the following conditions:

$$\sum_{n=0}^{\infty} \alpha_n \to \infty \text{ and } \sum_{n=0}^{\infty} (\alpha_n)^2 < \infty$$

- Let $\alpha_n(s, a) = 1/n(s, a)$,
  - where $n(s, a)$ is $\#$ of times that (s, a) is visited
- Q-learning

$$Q(s, a) \leftarrow Q(s, a) + \alpha_n(s, a) \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s, a) \right]$$

Convergence of Linear Gradient Q-Learning

- Linear Q-Learning converges under the same conditions:

$$\sum_{n=0}^{\infty} \alpha_n \to \infty \text{ and } \sum_{n=0}^{\infty} (\alpha_n)^2 < \infty$$

- Let $\alpha_n = 1/n$
- Let $Q_w(s, a) = \sum_i w_i x_i$
- Q-learning

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_n \left[Q_{\boldsymbol{w}}(s, a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}} \left(s', a'\right) \right] \frac{\partial Q_{\boldsymbol{w}}(s, a)}{\partial \boldsymbol{w}}$$

Divergence of Non-linear Gradient Q-learning

- Even when the following conditions hold

$$\sum_{n=0}^{\infty} \alpha_n \to \infty \text{ and } \sum_{n=0}^{\infty} (\alpha_n)^2 < \infty$$

  non-linear Q-learning may diverge
- Intuition:
  - Adjusting $\boldsymbol{w}$ to increase Q at (s, a) might introduce errors

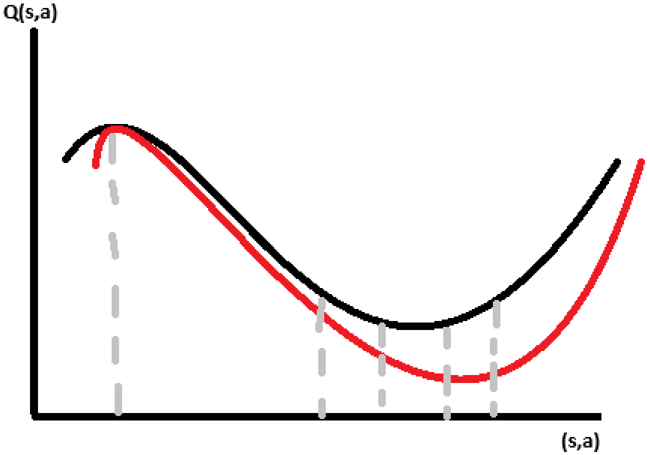
Mitigating divergence

Two tricks are often used in practice:

1. Experience replay
2. Use two networks:
   - Q-network
   - Target network

Experience Replay

Experience Replay



Experience Replay

- Idea: store previous experiences (s, a, s', r) into a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning
- Advantages
  - Break correlations between successive updates (more stable learning)
  - Fewer interactions with environment needed to converge (greater data efficiency)

Target Network

Target Network

- Idea: Use a separate target network that is updated only periodically

  repeat for each (s, a, s', r) in mini-batch:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_n \left[\underbrace{Q_{\boldsymbol{w}}(s,a)}_{update} - r - \gamma \max_{a'} \underbrace{Q_{\bar{w}}(s', a')}_{target} \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

  $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

- Advantage: mitigate divergence

Target Network

- Idea: Use a separate target network that is updated only periodically

  repeat for each (s, a, s', r) in mini-batch:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_n \left[\underbrace{Q_{\boldsymbol{w}}(s, a)}_{update} - r - \gamma \max_{a'} \underbrace{Q_{\bar{w}}(s', a')}_{target} \right] \frac{\partial Q_{\boldsymbol{w}}(s, a)}{\partial \boldsymbol{w}}$$

  $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

- Advantage: mitigate divergence
- Similar to value iteration:

  repeat for all s

$$\underbrace{V(s)}_{update} \leftarrow \max_a R(s) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, a) \underbrace{\bar{V}(s')}_{target} \forall s$$

$$\bar{V} \leftarrow V$$

- Q-learning is a sampling version of value iteration

Deep Q-network

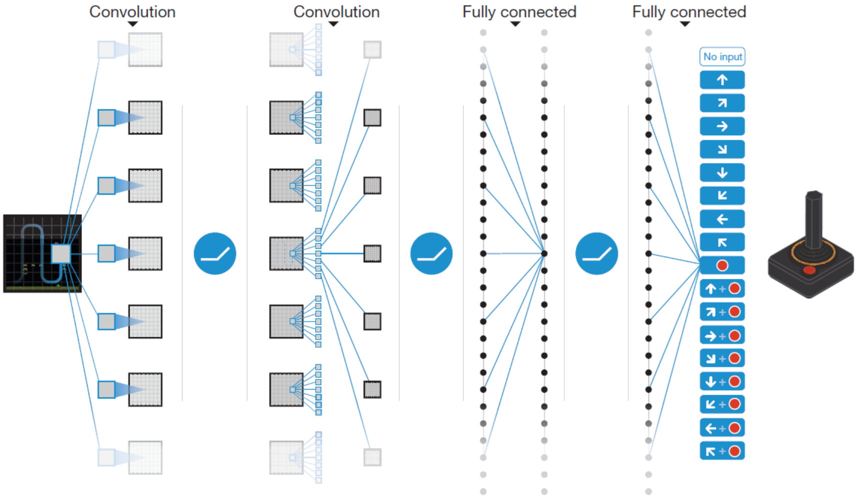
Google Deep Mind:

- Deep Q-network: Gradient Q-learning with
  - Deep neural networks
  - Experience replay
  - Target network
- Breakthrough: human-level play in many Atari video games

Deep Q-network

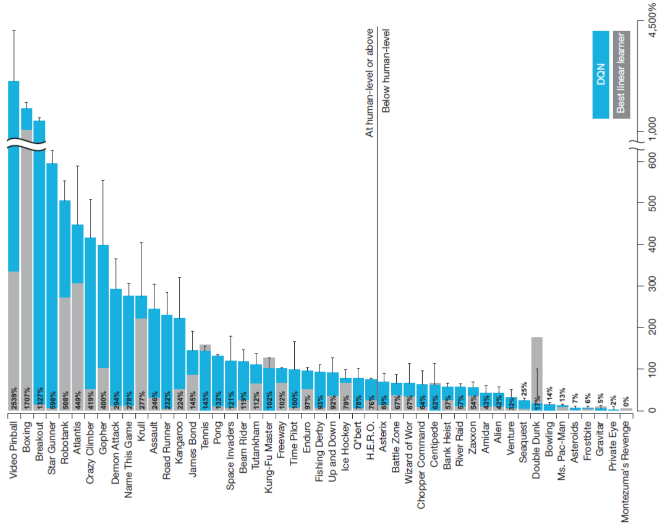
DQNetwork ($Q_{\boldsymbol{w}}(s, a)$)
Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ at random in $[-1, 1]$
Observe current state s
Loop
Select action a and execute it
Receive immediate reward r
Observe new state s'
Add (s, a, s', r) to experience buffer
Update Q-func by sampling mini-batch from buffer
For each experience $(\hat{s}, \hat{a}, \hat{s}', \hat{r})$ in mini-batch
Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = [Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} Q_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}')] \frac{\partial Q_{\boldsymbol{w}}(\hat{s}, \hat{a})}{\partial \boldsymbol{w}}$
Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$
Update state: $s \leftarrow s'$
Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-Network for Atari



Source: Mnih et al. (2015)

DQN versus Linear approx.



Note: Human: 75% of professional human games tester. *Source: Mnih et al. (2015)*

References I

Goodfellow, I. (2016): *Deep learning*, vol. 196. MIT press, Available at http://deeplearningbook.org/.

Takeaways

Deep Q-Learning (DQN)

- Combines Q-learning with deep neural networks for large state-action spaces
- It approximates Q-values by minimizing the Bellman error using gradient descent
- Experience replay and target networks stabilize training and prevent divergence
- DeepMind's DQN achieved human-level performance on Atari games with these techniques