Deep Reinforcement Learning

Deep Q-network

Ing. Milan Němý
PhD candidate

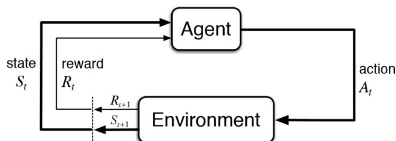
FEL ČVUT, 2. 11. 2018

Outline

1. Reinforcement Learning
   - Basic setting of RL
   - Solutions
   - Value function
2. Notable applications
   - TD-Gammon
   - Deep Q-Network

Basic elements of RL

- maximizes rewards over time through its choice of actions
- $S_0, A_0, R_1, S_1, A_1, R_2, \ldots$ - Markov Decision Process (MDP)
- probability $p(s', r|s, a)$ defines dynamics of the MDP
- **policy** $\pi(a|s)$ – mapping from states to probabilities of selecting each possible action
- **value function** of a state under a policy π: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$
- discounted return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$
- **action-value function**: $q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$



Solution

- **value function** of a state under a policy π: $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
- **action-value function**: $q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$
- optimal policy π_*:
- $v_*(s) = \max_\pi v_\pi(s)$ and $q_*(s,a) = \max_\pi q_\pi(s,a)$
- Bellman optimality equation:

$$\begin{aligned} v_*(s) = \max_{a \in \mathfrak{A}(s)} q_{\pi_*}(s,a) = \\ = \max_a \mathbb{E}_{\pi*}[G_t | S_t = s, A_t = a] = \\ = \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] = \\ = \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] = \\ = \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned} \tag{1}$$

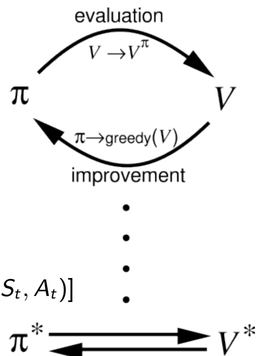
$$\begin{aligned} q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \; S_t = s, A_t = a] = \\ = \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{2}$$

Solution 2

- **Dynamic programming**: Bellman eqs into update rules
  $v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$
- **Monte Carlo**: Averaging sample returns
  - on-policy methods
  - off-policy methods
- **Temporal-Difference Learning**:
  $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1)} - V(S_t)]$
  - **Sarsa**:

    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  - **Q-learning** (off-policy):

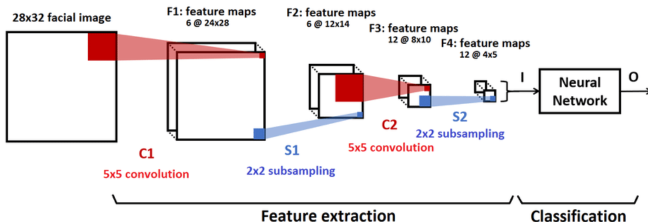
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$



Approximate value function

Value function:

- Tabular form
- Approximate value function:
  - parameterized function with weight factors $\mathbf{w}$, i.e., $\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$.
  - stochastic gradient descent
  - linear models $\hat{v}(s, \mathbf{w}) = \mathbf{w}^T.\mathbf{x}(s)$
  - non-linear models
    - artificial neural networks (ANN)
    - deep convolutional networks (CNN)

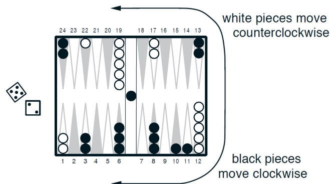


TD-Gammon

Tesauro 1992, 1994, 1995, 2002

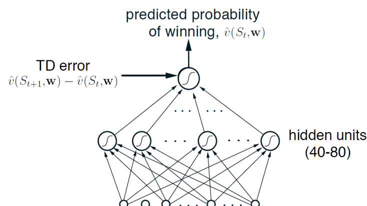
Backgammon:

- 30 pieces and 24 possible locations $\rightarrow$ enormous number of possible positions
- large number of moves
- dice rolls
- $\Rightarrow$ game tree effective branching factor $\sim$ 400



TD-Gammon:

- nonlinear form of TD
- estimated value $\hat{v}(s, \mathbf{w})$ – standard multilayer ANN – probabilities of winning from that state
- input units – representation of a backgammon position



TD-Gammon

- **TD-Gammon 0.0**
  - little backgammon knowledge
  - but clever way to present info to ANN
  - 198 input units
  - semi-gradient form of TD:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[R_t + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)].\mathbf{z_t} \\ \mathbf{z}_t &= \gamma\lambda\mathbf{z}_{t-1} + \nabla\mathbf{v}(S_t, \mathbf{w_t}) \end{aligned} \quad (3)$$

  - played against itself
  - after 300.000 games – the same performance as *Neurogammon* (based on extensive corpus)
- **TD-Gammon 1.0**
  - the same + specialized backgammon features
  - substantially better
- ...
- **TD-Gammon 3.1**
  - 160 hidden units
  - close or better than grandmasters

Human-level video game play

- problem of feature selection
- **Breakout game**
- location of the paddle?
- location/direction of the ball?
- presence/absence of each individual brick?
- move universal – *screen pixels*!

Human-level video game play

- problem of feature selection
- Google DeepMind – deep multi-layer ANN can automate the feature design process
- *Mnih et al.* – reinforcement learning agent Deep Q-Network (DQN) $=$ Q-learning $+$ deep convolutional ANN
- 49 different Atari 2600 video games
- different policies for different games BUT the same:
  - raw input
  - network architecture
  - hyperparameters
- Generating next states for each possible action? No
- Q-learning: model-free and off-policy

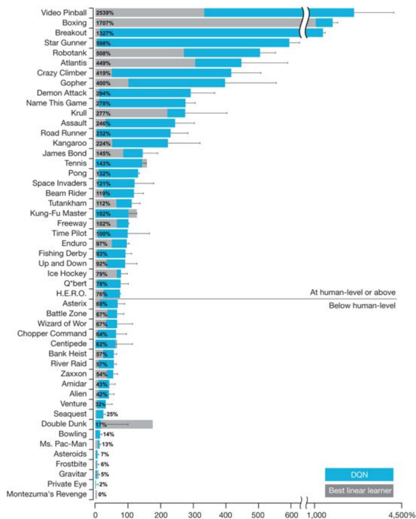
DQN Skill levels

compared to

- professional human tester – 2 h of practice then average reward over the next 20 games
- random agent
- best from literature (linear function approximation w/ designed features)

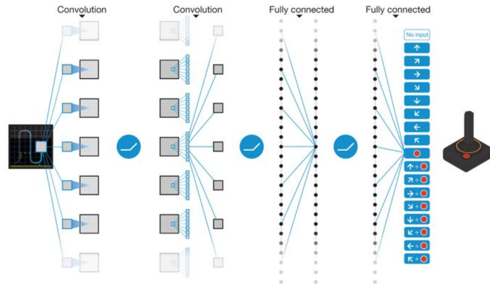
training:

- 50M frames $\sim$ 38 days of experience
- better than other RL systems on all but 6 games
- better than human on 22 of the games
- $\geq$ 75% of the human score – 29 out of 46 games



Q-network architecture

- original input: 210x160 px frame - 128 colors at 60 Hz
- preprocessing: 84x84 array of luminance values + 4 images stacked (observability)
- input: 84x84x4 input vector (preprocessing map Φ)
- architecture:
  - 3 hidden convolutional layers
  - 1 FC hidden layer (512)
  - output layer (18 possible actions)

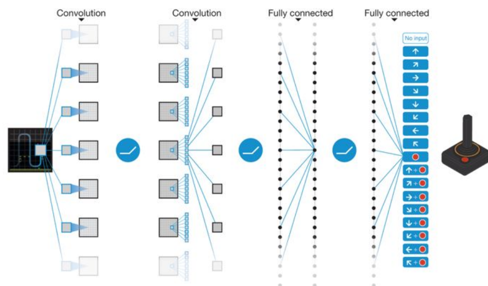


Q-network architecture

architecture:

- 32 filters of 8x8 with stride 4 + ReLU
- 64 filters of 4x4 with stride 2 + ReLU
- 64 filters of 3x3 with stride 1 + ReLU
- FC with 512 units
- output layer (18 possible actions)
- + reward signal (+1/-1/0) for all games

State → [Q-network] → 18 different Q values



Training

- take action according to ϵ-greedy policy $\rightarrow$ executed $\rightarrow$ reward + next video frame
- optimize MSE between Q-network and Q-learning targets

$$L_t(\mathbf{w_t}) = \mathbb{E}_{s,a,r,s'}[(\underbrace{r + \gamma \max_{a'} Q(s', a, \mathbf{w_i})}_{\text{target}} - Q(s, a, \mathbf{w}_t))^2]$$

- gradient descent:

$$\nabla_{\mathbf{w}_t} L(\mathbf{w_t}) = \mathbb{E}_{s,a,r,s'}[(r + \gamma \max_{a'} Q(s', a, \mathbf{w_i}) - Q(s, a, \mathbf{w}_t))\nabla_{\mathbf{w}_t} Q(s, a, \mathbf{w}_t)]$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla \hat{q}(S_t, A_t, \mathbf{w})$$

- gradient by backpropagation
- mini-batch method
- gradient-ascent algorithm *RMSProp*
- model-free and off-policy

Instability Issues

Naive Deep-RL oscillates and diverge

- Data is sequential - successive samples are correlated (non i.i.d)
- Policy changes rapidly with slight changes to Q-values
- Naive Q-learning gradients can be large and unstable when backpropagated

Instability Issues - Solution

Experience replay *Lin (1992)*

- break correlations in data, bring us back to i.i.d. setting
- learn from all past policies
- Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory D
- Q-learning updates based on experiences sampled uniformly at random from replay memory

$$L_t(\mathbf{w_t}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}[(\underbrace{r + \gamma \max_{a'} Q(s', a, \mathbf{w_i})}_{\text{target}} - Q(s, a, \mathbf{w}_t))^2]$$

Instability Issues - Solution

Fixed Target Q-Network

- $\gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$ depends on the parameters ($\mathbf{w}_t$) being updated
- $\rightarrow$ oscillations and/or divergence
- compute Q-learning targets w.r.t. old, fixed parameters $\mathbf{w}_t^-$
- optimize MSE between Q-network and Q-learning targets

$$L_t(\mathbf{w_t}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}[(\underbrace{r + \gamma \max_{a'} Q(s', a, \mathbf{w_i^-})}_{\text{target}} - Q(s, a, \mathbf{w}_t))^2]$$

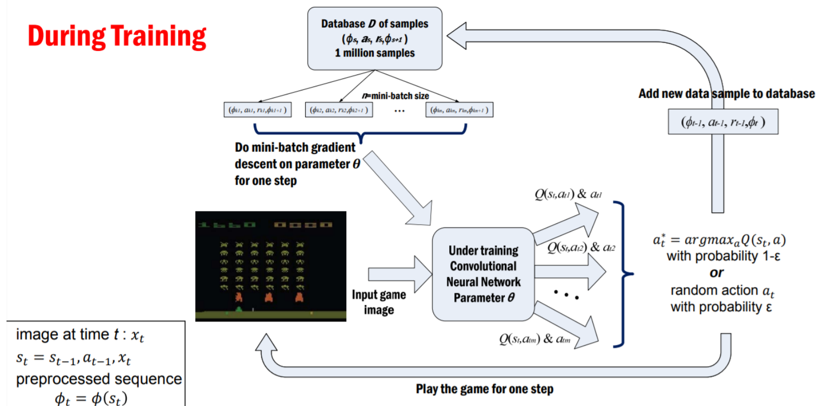
- periodically update fixed parameters

Instability Issues - Solution

Clipping

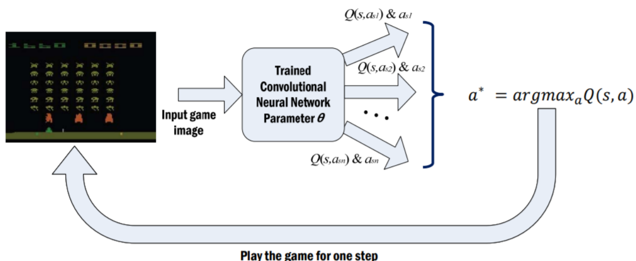
- clip $r + \gamma \max_{a'} Q(s', a', \mathbf{w}_t^-) - Q(s, a, \mathbf{w}_i)$ to remain in the interval $[-1, 1]$.
- further improves stability
- ensures gradients are well-conditioned

How to Train the Q-network



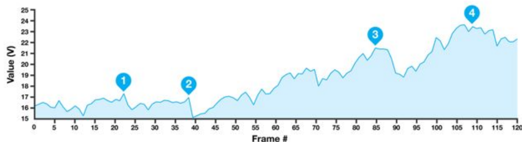
Slide by Bowen Xu (https://www.teach.cs.toronto.edu//~csc2542h/fall/material/csc2542f16_dqn.pdf)

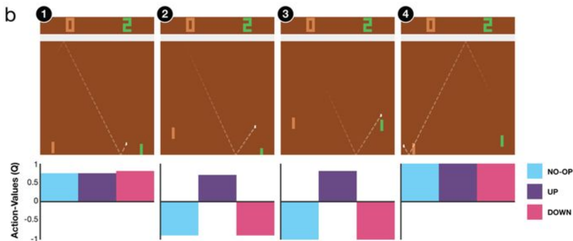
How to Test the Q-network



Slide by Bowen Xu (https://www.teach.cs.toronto.edu//~csc2542h/fall/material/csc2542f16_dqn.pdf)

Visualization of Value Functions





Replay and Target Q

Extended Data Table 3 | The effects of replay and separating the target Q-network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Conclusion

- pro: no need for problem-specific design and tuning
- pro: single agent can solve many challenging tasks
- con: not a complete solution, poor on some games
- recommended reading: *Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.*