(Deep) Reinforcement Learning

Martin Matyášek

Artificial Intelligence Center
Czech Technical University in Prague

October 27, 2016
Reinforcement Learning in a picture

- learning what to do to maximize future reward
- general-purpose framework extending sequential decision making when the model of the environment is unknown
RL background

- let’s assume MDP $\langle S, A, P, R, s_0 \rangle$
  - RL deals with situation where the environment model $P$ and $R$ is unknown
  - can be generalized to Stochastic Games $\langle S, N, A, P, R \rangle$

- RL agent includes:
  - **policy** $a = \pi(s)$ (deterministic), $\pi(a \mid s) = \mathbb{P}(a \mid s)$ (stochastic)
  - **value function** $Q^\pi(a \mid s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s, a]$
    - Bellman Eq.: $Q^\pi(a \mid s) = \mathbb{E}_{s', a'} [r + \gamma Q^\pi(a' \mid s') \mid s, a]$
    - opt. value functions: $Q^*(s, a) = \mathbb{E}_{s', a'} [r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$
    - opt. policy: $\pi^*(s) = \arg\max_a Q^*(s, a)$
  - **model** - learned proxy for environment
RL Types

1. **value-based** RL
   - estimate the opt. value function $Q^*(s, a)$
   - max. value achievable under *any* policy

2. **policy-based** RL
   - search directly for the opt. policy $\pi^*$
   - i.e. policy yielding max. future reward

3. **model-based** RL
   - build a model of the environment
   - plan using this model
Algorithm 1: Q-learning

1: initialize the Q-function and V values (arbitrarily)
2: repeat
3: observe the current state $s_t$
4: select action $a_t$ and take it
5: observe the reward $R(s_t, a_t, s_{t+1})$
6: $Q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha_t)Q_t(s_t, a_t) + \alpha_t (R(s_t, a_t, s_{t+1}) + \gamma V_t(s_{t+1}))$
7: $V_{t+1}(s) \leftarrow \max_a Q_t(s, a)$
8: until convergence
Q-learning

- model-free method
- *temporal-difference* version:
  \[ Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \]
  value based on next state

converges to \( Q^* \), \( V^* \) iff \( 0 \leq \alpha_t < \infty \), \( \sum_{t=0}^{\infty} \alpha_t = \infty \) and \( \sum_{t=0}^{\infty} \alpha_t^2 < \infty \)

- *zero-sum* Stochastic Games:
  - cannot simply use \( Q_{i}^{\pi} : S \times A_i \rightarrow \mathbb{R} \) but rather \( Q_{i}^{\pi} : S \times A \rightarrow \mathbb{R} \)
  - *minimax-Q* converges to NE
  - *R-max*: converge to \( \epsilon \) – Nash with prob. \( (1 - \delta) \) in poly. # steps
    ((PAC learn))
Q-Networks

- $Q^*(s, a) \approx Q(s, a, w)$
- treat right hand side $r + \gamma \max_{a'} Q(s', a', w)$ of Bellman’s Eq. as target
- minimize **MSE loss** $l = (r + \max_{a'} Q(s', a', w) - Q(s, a, w))^2$ by stochastic gradient descent

David Silver, Google DeepMind
Q-learning summary

+ converges to $Q^*$ using table lookup representation
- diverges using NN:
  - correlations between samples
  - non-stationary targets

! go deep
Deep Q-Networks (DQN)

- basic approach is called **experience replay**

  - *idea*: remove correlations by building data-set from agent’s experience \( e = (s, a, r, s') \)

  - sample experiences from \( D_t = \{e_1, e_2, \ldots e_t\} \) and apply update

- deal with non-stationarity by fixing \( w^- \) in

\[
l = (r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w))^2
\]
Algorithm 2 Deep Q-learning algorithm

1: init. replay memory $D$, init. $Q$ with random weights
2: observe initial state $s$
3: repeat
4: with prob. $\epsilon$ select random $a$, select $a = \text{argmax}_{a'} Q(s, a')$
5: carry out $a$, observe $(r, s')$ and store $(s, a, r, s')$ in $D$
6: sample random transition $(ss, aa, rr, ss')$ from $D$
7: calculate target for each minibatch transition:
8: if $ss'$ is terminal state then
9:     $tt \leftarrow rr$
10: else
11:     $tt \leftarrow rr + \gamma \max_{a'} Q(ss', aa')$
12: end if
13: train the Q-network using $(tt - Q(ss, aa))^2$ as loss
14: $s \leftarrow s'$
15: until convergence
DQN in Atari

David Silver, Google DeepMind
(Deep) Reinforcement Learning

DQN in Atari - setting

- **state**: stack of raw pixels from last 4 frames
- **actions**: 18 joystick/button positions
- **reward**: delta in score
- **learn**: $Q(s, a)$

David Silver, Google DeepMind
(Deep) Reinforcement Learning

DQN in Atari - results

Martin Matyášek VPD, 2016

13 / 17
DQN improvements

- **Double DQN** removes bias caused by $\max_a Q(\cdot)$
  - current QN - $w$ used to select actions
  - old QN - $w^-$ used to evaluate actions

- **Prioritized Replay** weight experience according to DQN error (stored in PQ)

- **Duelling Network** split Q-Network into:
  - action-independent *value* function
  - action-dependent *advantage* function
General Reinforcement Learning Architecture (GORILA)
(Deep) Reinforcement Learning

Deep Policy Network

- parametrize the policy $\pi$ by a DNN and use SGD to optimize weights $u$
  - $\pi(a \mid s, u)$ or $\pi(s, u)$
  - $\max_u L(u) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi(\cdot, u)]$

- policy gradients:
  - $\frac{\partial L(u)}{\partial u} = \mathbb{E}\left[ \frac{\partial \log \pi(a \mid s, u)}{\partial u} Q^\pi(s, a) \right]$ for stochastic policy $\pi(a \mid s, u)$
  - $\frac{\partial L(u)}{\partial u} = \mathbb{E}\left[ \frac{\partial Q^\pi(s, a)}{\partial a} \frac{\partial a}{\partial u} \right]$ for deterministic policy $a = \pi(s)$ where $a$ is cont. and $Q$ diff.
Next time..

- cont. policy-based deep RL: *Actor-Critic alg.*, *A3C*
- *Fictitious Self-Play*
- model-based deep RL