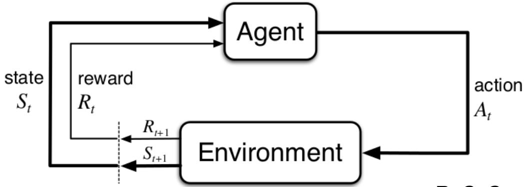
(Deep) Reinforcement Learning

Martin Matyášek

Artificial Intelligence Center
Czech Technical University in Prague

October 27, 2016

Reinforcement Learning in a picture



R. S. Sutton and A. G. Barto 2015

- 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 funcions: $Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a') \mid s, a]$
    - opt. policy: $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$
  - **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 π^*
   - i.e. policy yielding max. future reward
3. **model-based** RL
   - build a model of the environment
   - plan using this model

Q-learning

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

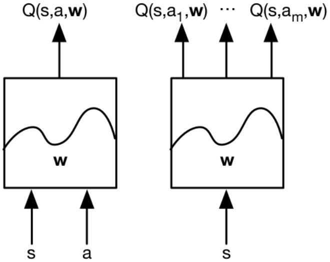
CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

Q-learning

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

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

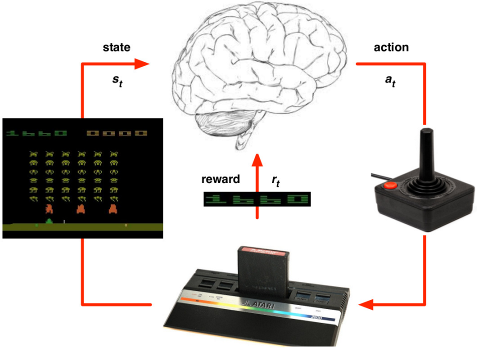
Deep Q-Networks (DQN)

- 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 $D_t = \{e_1, e_2, \dots e_t\}$ and apply update
- deal with non-stationarity by fixing $\mathbf{w}^-$ in $l = (r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}))^2$

Algorithm 2 Deep Q-learning algorithm

init. replay memory D, init. Q with random weights
observe initial state s
repeat
  with prob. ϵ select random a, select $a = \text{argmax}_{a'} Q(s, a')$
  carry out a, observe (r, s') and store (s, a, r, s') in D
  sample random transition (ss, aa, rr, ss') from D
  calculate target for each minibatch transition:
  if ss' is terminal state **then**
    $tt \leftarrow rr$
  else
    $tt \leftarrow rr + \gamma \max_{a'} Q(ss', aa')$
  end if
  train the Q-network using $(tt - Q(ss, aa))^2$ as loss
  $s \leftarrow s'$
until convergence

DQN in Atari



David Silver, Google DeepMind

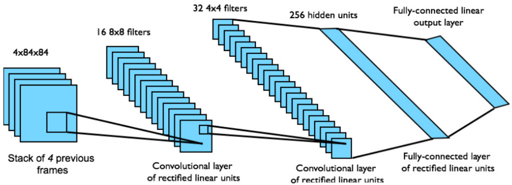
DQN in Atari - setting

state stack of raw pixels from last 4 frames

actions 18 joystick/button positions

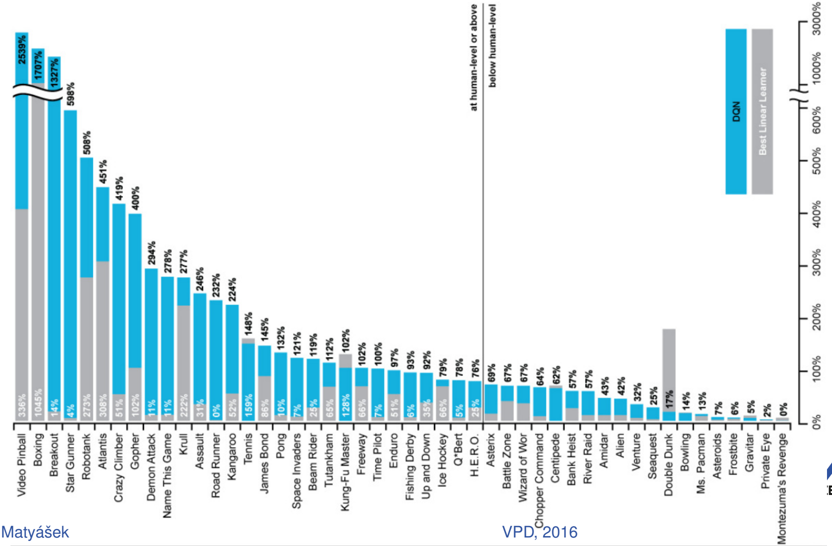
reward delta in score

learn $Q(s, a)$



David Silver, Google DeepMind

DQN in Atari - results

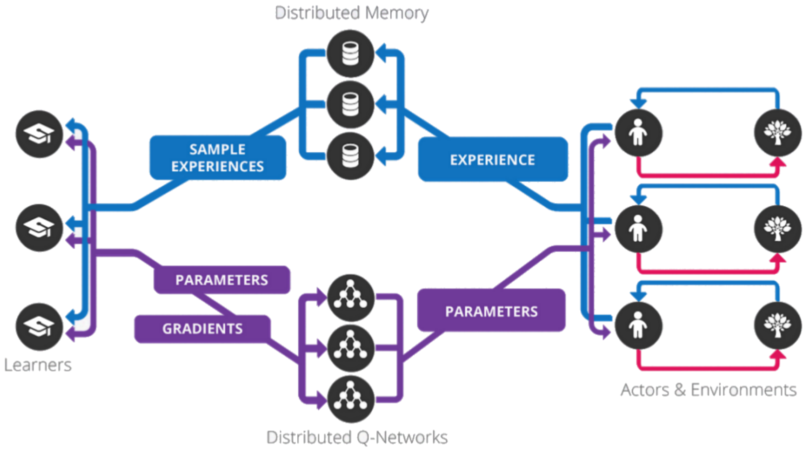


AI CENTER

DQN improvements

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

- parametrize the policy π by a DNN and use SGD to optimize weights $\mathbf{u}$
  - $\pi(a \mid s, \mathbf{u})$ or $\pi(s, \mathbf{u})$
  - $\max_{\mathbf{u}} L(\mathbf{u}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi(\cdot, \mathbf{u})]$
- policy gradients:
  - $\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}[\frac{\partial \log \pi(a|s,\mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)]$ for *stochastic* policy $\pi(a \mid s, \mathbf{u})$
  - $\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}[\frac{\partial Q^{\pi}(s,a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}}]$ for *deterministic* policy $a = \pi(s)$ where a is cont. and Q diff.

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

Next time..

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