DQN - Deep Q Learning

Syllable

0. WHY DQN?
1. QNetwork
2. Experience Replay
3. Train and update QNetwork

0. WHY DQN?

Q-Learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode :

Start from S_1 :

  Loop for each step t :

    Sample actions A_t using policy derived from $Q(S_t, \cdot)$ (e.g. ϵ-greedy)

    Observe R_t, S_{t+1}

$$\boxed{\begin{aligned} &\bar{Q}(S_t, A_t) \leftarrow R_t + \gamma \max_a Q(S_{t+1}, a) \\ &Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\bar{Q}(S_t, A_t) - Q(S_t, A_t)] \end{aligned}}$$

    $S_t \leftarrow S_{t+1}$

  until S_T is terminal

- maintain Q table, size = [nS, nA]
  - might be infinite
  - searching might be slow

1. Q-learnig $\rightarrow$ Deep learning

$$\overline{Q}(S_t, A_t) \leftarrow R_t + \gamma \max_a Q(S_{t+1}, a)$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\overline{Q}(S_t, A_t) - Q(S_t, A_t)]$$

$\rightarrow$ Deep learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

$$Q^*(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s', a')|s, a]$$

θ - weigh of the neural network, to update

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

`{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com`

Abstract

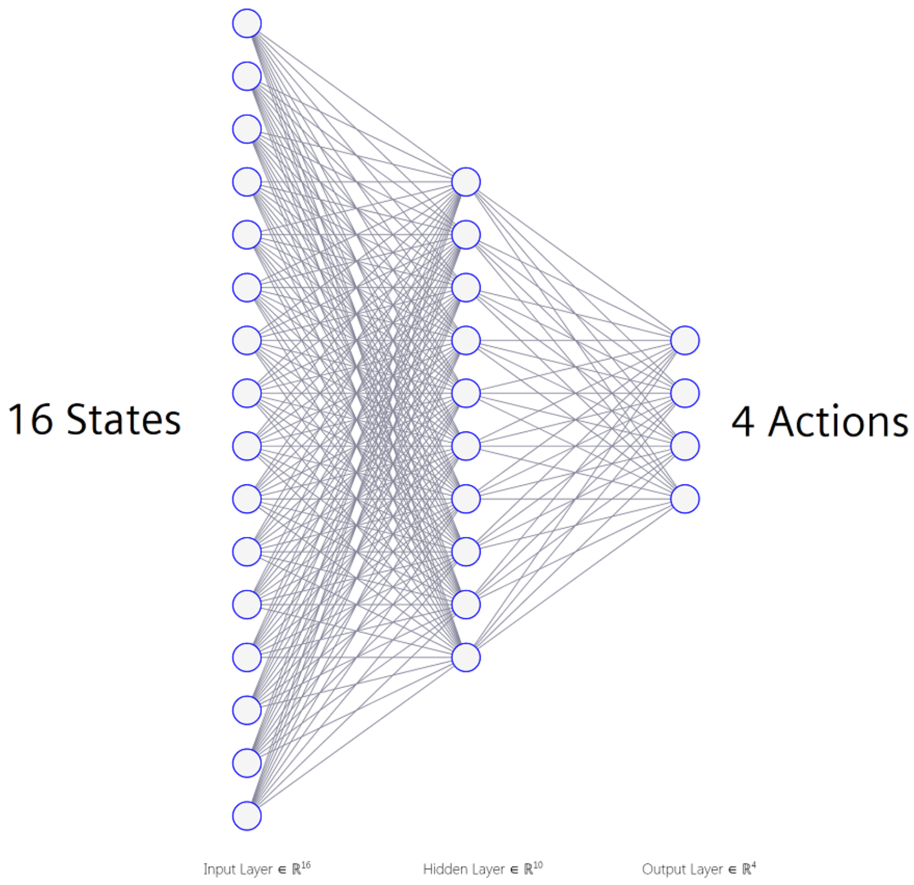
We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

1.1 Definition of Q-function / QNetwork

1. [INPUT] - state, action; [OUTPUT] - $Q(s, a)$
2. [INPUT] - state; [OUTPUT] - $Q(s)$, size = [nA,], $a = \arg\max_a Q(s)$
   e.g. frozen lake:



coding 1: Define QNetwork

1.2 Loss function

- minimising a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}[(y_i - Q(s, a; \theta_i))^2]$$

in which,

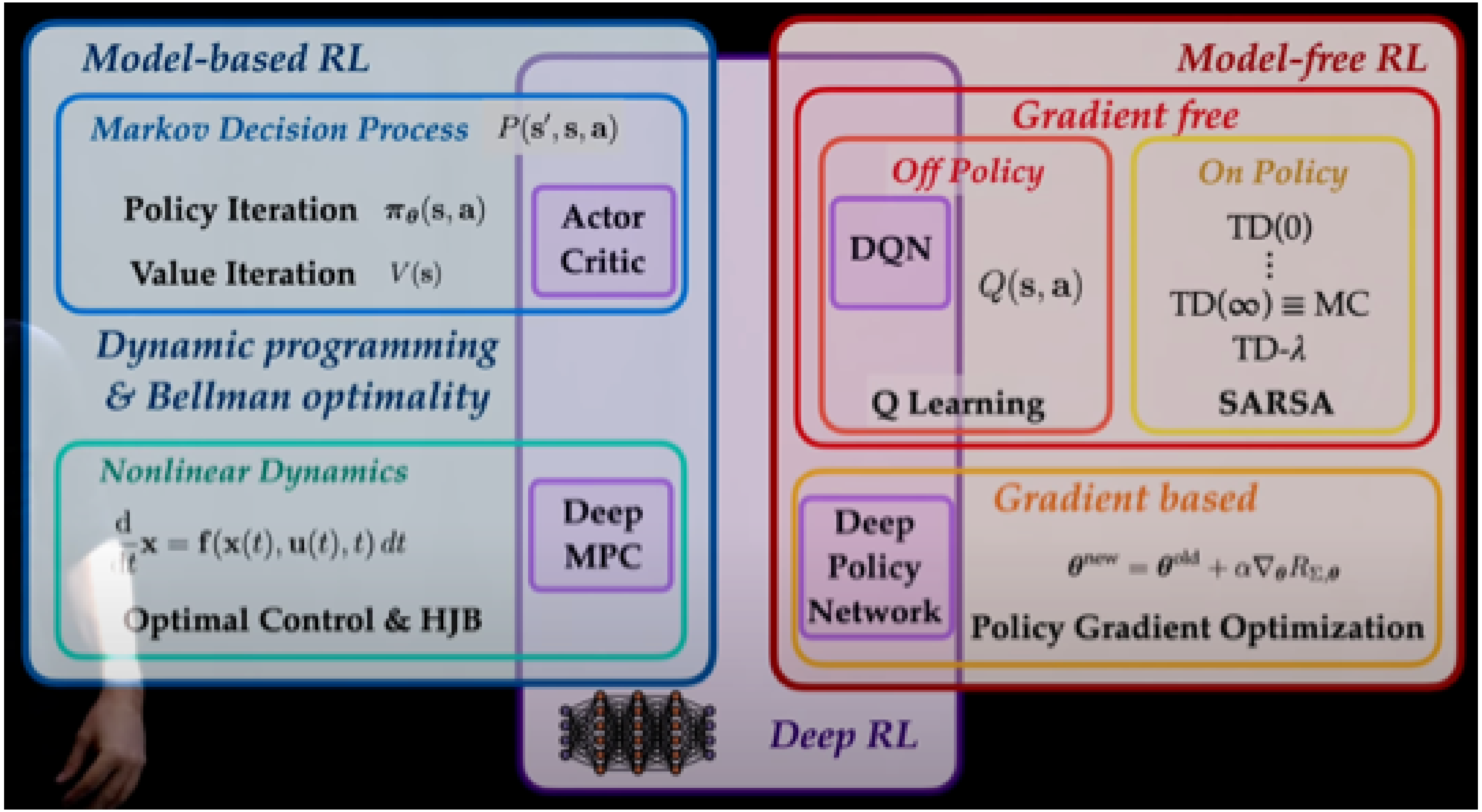
- $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a]$
  approximate target value for iteration i
- $\rho(s, a)$ - probability distribution over sequences s and actions a that we refer to as the behaviour distribution

1.2.1 Relation with other algor

Note that this algorithm is model-free: it solves the reinforcement learning task directly using samples from the emulator E, without explicitly constructing an estimate of E.

It is also off-policy: it learns about the greedy strategy $a = \max_a Q(s, a; \theta)$,

while following a behaviour distribution that ensures adequate exploration of the state space. ε-greedy



Model-based RL
Markov Decision Process $P(s', s, a)$
Policy Iteration $\pi_\theta(s, a)$
Value Iteration $V(s)$
Actor Critic
Dynamic programming & Bellman optimality
Nonlinear Dynamics
$\frac{d}{dt}x = f(x(t), u(t), t)\,dt$
Optimal Control & HJB
Deep MPC
Model-free RL
Gradient free
Off Policy
DQN
$Q(s, a)$
Q Learning
On Policy
TD(0)
$TD(\infty) \equiv MC$
TD-λ
SARSA
Gradient based
Deep Policy Network
$\theta^{new} = \theta^{old} + \alpha \nabla_\theta R_{\Sigma,\theta}$
Policy Gradient Optimization
Deep RL

1.3 Problems, Q-learning Unstable with non-linear approximator

1. small updates to Q may significantly change the policy and the data distribution
2. correlations between the action-values (Q) and the target values $r + \gamma \max_{a'} Q(s', a')$
3. targets depend on the network weights; this is in contrast with the targets used for supervised learning, which are fixed before learning begins

Solve

1. Experience replay, randomizes over the data, removing correlations in the observation sequence and smoothing over changes in the data distribution
2. periodically updated, θ_i^- : At each stage of optimization, we hold the parameters from the previous iteration θ_i^- fixed when optimizing the ith loss function $L_i(\theta_i)$

2. Experience Replay, get [input] for QNetwork

1. Sample several steps! before updating θ
   store the agent's experiences, $e_t = (s_t, a_t, r_t, s_{t+1})$
   at each time-step t in a data set $D_t = \{e_1, ..., e_t\}$
2. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s, a, r, s') \sim U(D)$
   drawn uniformly at random from the pool of stored samples

coding 2: sample & store agent's experiences

create minibatches for input

Train and update

Loss Function

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}[(y_i - Q(s,a;\theta_i^-))^2]$$

in which,

- $y_i = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}^-)|s,a]$

Algorithm: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

for episode = 1, M do
    Initialize sequence s_1 and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for t = 1, T do
        sample actions a_t derived from $Q^*(\phi(s_t), a; \theta)$, with ϵ-greedy
        Execute a_t, observe r_t and s_{t+1}; Proprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $e_t = (s_t, a_t, r_t, s_{t+1})$ in D
        sample minibatches of transitions (s_t, a_t, r_t, s_{t+1}) from D
        Set $y = r_j + \gamma \max_{a'} Q(s_{j-1}, a_{j-1}; \theta^{-})$ if non-terminal s_{j+1}; else r_j & break
        Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    end for
end for

coding 3: Train and update

倒立摆

https://blog.csdn.net/qq_32892383/article/details/89576003