Deep Reinforcement Learning

EMAT31530/Spring 2021/Xiaoyang Wang

A Quick Recap

- Markov Decision Process (MDP) - mathematical formulation of RL problems

  Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

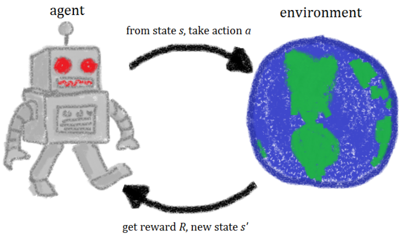
  Policy $\pi(s)$: $\mathcal{S} \rightarrow \mathcal{A}$

  Optimal policy $\pi^* = \arg\max_\pi E\left[\sum_t \gamma^t r_t\right]$

- Bellman Equation $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s') + \gamma V^*(s')]$
- Value iteration, policy iteration

A Quick Recap

- Reinforcement Learning



states \ actions	a_0	a_1	$\cdots$
s_0	$Q(s_0, a_0)$	$Q(s_0, a_1)$	$\cdots$
s_1	$Q(s_1, a_0)$	$Q(s_1, a_1)$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- A model-free, off-policy method: Q-Learning

Q-learning: estimating the action-value function

$$Q(s,a) \approx Q^*(s,a) \tag{1}$$

Update Q in Q-learning

$$Q^{new}(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)) \tag{2}$$

Why does it work?

The Bellman Equation for $Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right] \tag{3}$$

With $t \to \infty$, Q will converge to Q^*

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

The Bellman Equation for $Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a \right] \tag{3}$$

'Tabular' Q-learning - Problem?

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

- θ: function parameters
- Linear, non-linear...

Deep Reinforcement Learning

The Bellman Equation for $Q^*(s, a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right] \tag{3}$$

'Tabular' Q-learning - Problem?

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

- θ: function parameters
- Linear, non-linear...
- If $Q(s, a; \theta)$ is a deep neural network $\rightarrow$ **Deep Q-Learning**

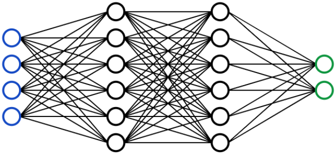


Figure: Q network

The Bellman Equation for $Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s',a') \Big| s,a \right] \tag{3}$$

To train the Q-network

Loss function in each iteration $L_i(\theta_i)$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i) \right)^2 \right] \tag{4}$$

Target y_i

$$y_i = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1} \Big| s,a) \right] \tag{5}$$

The Bellman Equation for $Q^*(s,a)$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s', a') \middle| s, a \right] \tag{3}$$

To train the Q-network

Loss function in each iteration $L_i(\theta_i)$

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2 \right] \tag{4}$$

Target y_i

$$y_i = \mathbb{E}_{s' \sim \mathcal{S}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1} \middle| s, a) \right] \tag{5}$$

The Gradient of $L_i(\theta_i)$

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{S}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i) \right) \nabla_{\theta_i} Q(s,a;\theta_i) \right] \tag{6}$$

Deep Reinforcement Learning

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{S}}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_{i-1})-Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{6}$$

Simplifications for computing Eq. (6)

- Expectation $\rightarrow$ Stochastic Gradient Descent
- Updating weights after every time step – just like in Tabular Q-learning

---

Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right] \tag{4}$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{S}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}\middle|s,a)\right] \tag{5}$$

- Target function depends on $\theta \rightarrow$ divergence problem
- $s, a, s', a', \ldots \rightarrow$ Consecutive samples, correlated
- Biased training examples generated by current Q-network

Experience Replay

- Store agent's experience (s_t, a_t, r_t, s_{t+1}) at each time-step in a fixed-size buffer $\mathcal{D}$
- Train Q-networks using minibatches sampled uniformly from $\mathcal{D}$ – break data correlation
- Behaviour distribution $\rho(\cdot)$ is averaged over previous states – smooth out learning process
- Experiences can be re-used – higher data efficiency

Any limitations?

Q-network Q and Target network $\hat{Q}$

- Using a separate neural network, $\hat{Q}$, to generate targets y_i
- For every C steps, set $\hat{Q} = Q$
- Stabilize training!

This is called the "Deep Q-network" (DQN): Deep Q-learning with the experience replay and a target Q-network.

Proposed in [1], it was successfully applied in Atari games, "was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters."

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

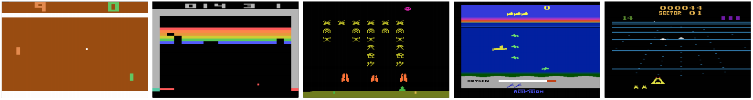


Figure: Atari 2600 games: Pong, Breakout, Space Invaders, Seaquest, Beam Rider[1]

State: Raw pixel inputs of game states

$$s_t = [x_{t-3}, x_{t-2}, x_{t-1}, x_t], \text{ then preprocess } \phi(s_t).$$
$$\phi\text{: RGB to grey, downsampling, cropping}$$

Action: Game console control: 8 directions with a button.
Reward: Game rewards (clipped).

[1] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Application: DQN for Atari Games

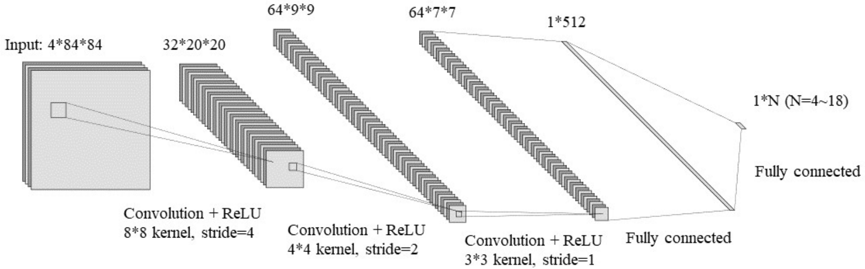


Figure: The neural network structure used in [1], i.e., $Q(s, a; \theta)$. The outputs are Q values of available actions, given the state.

Here N is the number of actions, depending on games.

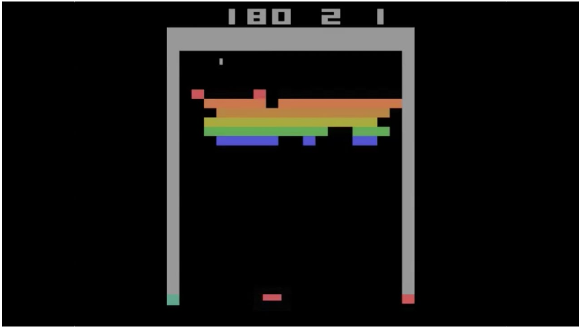
Application: DQN for Atari Games

Algorithm 1: Deep Q-learning with experience replay.

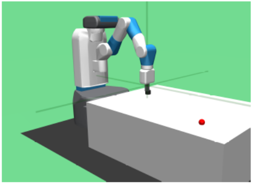
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode* $= 1, M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t; Otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Update s_{t+1} and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_i = \begin{cases} r_j, & \text{if episode terminates} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-), & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$
    end
end

Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

DQN Breakout-DeepMind



Deep RL: Applications




Bellemare, Marc G., et al. "Autonomous navigation of stratospheric balloons using reinforcement learning." Nature 588.7836 (2020): 77-82.

Summary

- Deep Q-learning
- Experience replay, Target Q network - DQN

Next: Implementation of DQN, testing on OpenAI Gym environment (e.g, Atari games)