Deep Reinforcement Learning

Wenjun Zeng

Reading Group of Ye Lab, UM

Mar. 9, 2019

Contents

- Recap
- Q-Learning
- Deep Q-Networks (DQN)

I. Recap

Markov Decision Processes (MDP)

- State space $\mathcal{S}$, $s \in \mathcal{S}$
- Action space $\mathcal{A}$, $a \in \mathcal{A}$
- Policy π, $a = \pi(s)$ for deterministic case or $a \sim \pi(a|s)$ for stochastic case
- Reward $R(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- Transition Dynamics P: $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$

  $P_{sa}(s')$ is the probability that the agent reaches state s' from s after taking action a.

Objective of MDP

Denoting s_t, a_t, and $R_t = R(s_t, a_t)$ as the state, action, and reward of $t = 0, 1, 2, \cdots$, MDP aims at finding policy to maximize the expectation of

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_t$$

with $\gamma \in [0, 1)$ the discount factor.

- Model-free: State transition $P_{sa}(s')$ and reward function are unknown.

Q-function (state-action value function)

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a, \pi \right]$$

is the expected return for executing a particular action at a given state.

Bellman's Equation

- Bellman equation on Q-function

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_{sa}(s') \max_{a' \in \mathcal{A}} Q^*(s',a')$$

- Bellman equation in expectation form:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P_{sa}(s')} \left[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a') \right]$$

- In conventional optimization (linear/nonlinear programming...), we solve Karush–Kuhn–Tucker (KKT) condition to obtain the optimal (stationary) solution.
- In MDP (RL), we solve Bellman's equation to obtain the optimal solution. If $P_{sa}(s')$ is known (model-based), it is dynamic programming.

Q-Value Iteration for Model-Based Case

- $Q^*(s,a)$: expected return starting in s, taking action a, and (thereafter) acting optimally
- Bellman Equation:

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} P_{sa}(s')(R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a'))$$

- Q-Value Iteration:

$$Q(s,a) \leftarrow \sum_{s' \in \mathcal{S}} P_{sa}(s')(R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s',a'))$$

Model-Free Reinforcement Learning

- In realistic problems, often the state transition and reward function are not explicitly given.
- Model-free RL is to directly learn value & policy from experience.
- How to accumulate experience? $\rightarrow$ learning from episodes

For $t = 0, 1, 2, \cdots, T$

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

(Tabular) Q-Learning

Rewrite Q-value iteration as expectation:

$$Q(s,a) \leftarrow \mathbb{E}_{s' \sim P_{sa}(s')}\left[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s',a')\right]$$

(Tabular) Q-Learning: replace expectation by samples

- For a state-action pair (s,a), receive: $s' \sim P_{sa}(s')$
- Use the old estimate: $Q(s,a)$
- Consider your new sample estimate:

$$y = \mathrm{target}(s') = R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

- Incorporate the new estimate into a running average:

$$Q^{\mathrm{new}}(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \mathrm{target}(s')$$

Algorithm: (Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \left[\text{target}\right]$$

$$s \leftarrow s'$$

Learning rate α: $\sum_{t=0}^{\infty} \alpha_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$

Theorem: *Q-learning converges to the optimal action-value function*

$$Q(s, a) \to Q^*(s, a).$$

How to sample actions?

- Choose random actions?
- Greedy? That is, choose action that maximizes $Q(s, a)$.
- ϵ-Greedy Policy Exploration:
  - With probability ϵ, choose an action at random
  - With probability $1 - \epsilon$, choose the greedy action

III. DQN

Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

Approximate Q-Learning

- Solve the curse of dimensionality of too big table ($|\mathcal{S}| \times |\mathcal{A}|$) of tabular Q-learning
- Instead of a table, we use a parametrized Q-function: $Q_\theta(s, a)$ with $\boldsymbol{\theta} = [\theta_1, \cdots, \theta_d]^\top$.
- Nonlinear neural network approximation of Q-function: Deep Q-Networks (DQN)

  $\boldsymbol{\theta}$: weights of the neural network

Approximate Q-Learning Rule:

- Remember

$$\mathrm{target}(s') = R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q_\theta(s', a')$$

- Update $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \left(\frac{1}{2} (Q_\theta(s, a) - \mathrm{target}(s'))^2 \right)$$

Why the objective function $\frac{1}{2}(Q_\theta(s, a) - \mathrm{target}(s'))^2$ in gradient method?

We can check that the gradient update rule recovers the tabular Q-learning update rule if $Q_\theta(s, a) \equiv \theta_{sa}$ for $\theta \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$.

Instability of Nonlinear Approximation Using Neural Networks

- Correlations present in the sequence of observations (episodes) $e_t = (s_t, a_t, r_t, s_{t+1})$
- Small updates to Q-function may significantly change the policy and therefore change the data distribution.
- Correlations between $Q(s,a)$ and the target values $\mathrm{target}(s') = R(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$

Novelty of DQN: stabilize Q-learning for nonlinear approximation with CNN

Experience Replay

- Store agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time-step in a data set $D_t = \{e_1, \cdots, e_t\}$ pooled over many episodes into a replay memory. The end of an episode occurs when a terminal state is reached.
- During the inner loop of the algorithm, apply Q-learning updates, or minibatch updates, to samples of experience, $(s, a, r, s') \sim U(D)$, drawn randomly from the pool of stored samples.
  Online update $\rightarrow$ off-line

Advantages of experience replay

- Each step of experience is potentially used in many weight updates, which allows for greater data efficiency.
- Learning directly from consecutive samples is inefficient, owing to the strong correlations between the samples; randomizing the samples breaks these correlations and therefore reduces the variance of the updates.
- By using experience replay, the behavior distribution is averaged over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.
- When learning by experience replay, it is necessary to learn off-policy (because our current parameters are different to those used to generate the sample), which motivates the choice of Q-learning.

Target Network

Further improving the stability is to use a separate network (target network) for generating the targets $y = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')$ in the Q-learning update.

Use an older set of weights to compute the targets:

Keeps the target function from changing too quickly.

At iteration i, DQN uses minimizes the following loss function:

$$\min \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q_{\boldsymbol{\theta}_i^-}(s', a') - Q_{\boldsymbol{\theta}_i}(s, a) \right)^2 \right]$$

- $\boldsymbol{\theta}_i^-$: target network parameters at iteration i
- $\boldsymbol{\theta}_i$: parameters of the Q-network at iteration i
- The target network parameters $\boldsymbol{\theta}_i^-$ are only updated with the Q-network parameters $\boldsymbol{\theta}_i$ every C steps and are held fixed between individual updates.

Target Network

- Every C updates we clone the network Q to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the Q-learning targets y for the following C updates to Q.
- This modification makes the algorithm more stable compared to standard online Q-learning.
- Generating the targets using an older set of parameters adds a delay between the time an update to Q is made and the time the update affects the targets y, making divergence or oscillations much more unlikely.

Other Details of DQN

- Downsampling: 210×160 game images to ones of 84×84 ($s_t \rightarrow \phi(s_t)$).
- CNN: 5 layers: 3 convolution layers + 2 full connection layers

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

- Uses RMSProp instead of vanilla SGD

  Optimization in RL really matters.
- It helps to anneal the exploration rate:

  Start ϵ at 1 and anneal it to 0.1 or 0.05 over the first million frames.

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 $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ 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 $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Thank you!