Q-learning with Nearest Neighbors

Chun-Heang Tay
Institute of Computer Science and Engineering
National Chiao Tung University
Hsinchu Taiwan
petertay1996@gmail.com

1. Introduction

Q-learning is an Reinforcement Learning algorithm method that directly get the optimal Q value by learning the optimal action-value function (Q function) from the observation of system trajectories.

Nearest Neighbor Q-Learning (NNQL) is an algorithm that its purpose is to learn the optimal Q function using nearest neighbor regression method.

2. Assumption

2.1 Assumption 1

We need to analysis a continuous state space problem, so we need to apply some assumption: MDP Regularity.

Assume that
(i) Continuous state space X is a compact subset of R_d;
(ii) A is a finite set of cardinality |A|
(iii) One-stage R_t is non-negative and uniformly bounded by R_{max}
(iv) $a\epsilon\ A, r(.|a)\epsilon\ Lip(X, Mr), Mr > 0$ which

$$\mathrm{Lip}(E, M) = \{f \in C(E) \mid |f(x) - f(y)| \le M\rho(x, y),\ \forall x, y \in E\}.$$

Since it's a continuous state space, we can define transition prob. :

$$\Pr(x_{t+1} \in B | x_t = x, a_t = a) = \int_B p(y|x, a)\lambda(dy)$$

satisfies

$$|p(y|x, a) - p(y|x', a)| \le W_p(y)\rho(x, x'), \qquad \forall a \in \mathcal{A}, \forall x, x', y \in \mathcal{X},$$

Under the condition of continuous state space, Bellman optimality operator can be written as

$$(FQ)(x, a) = r(x, a) + \gamma\mathbb{E}\left[\max_{b \in \mathcal{A}} Q(x', b) \mid x, a\right] = r(x, a) + \gamma\int_{\mathcal{X}} p(y|x, a)\max_{b \in \mathcal{A}} Q(y, b)\lambda(dy)$$

The smoother the MDP will lead to a smoother optimal Q function.

2.2 Assumption 2

Since handling continouous state space problem, we need to find a series of point on this state space X, then use these point describe Q-function.

After knowing all the point on this state space:
$q = \{q(c_i, a), c_i \in \mathcal{X}, a \in \mathcal{A}\}$
we represent any state using q.
$(\Gamma_{\mathrm{NN}}q)(x, a) = \sum_{i=1}^{n} K(x, c_i)q(c_i, a), \qquad \forall x \in \mathcal{X}, a \in \mathcal{A},$

The function above project a value on X_hxA function to a XxA function. The K is a kernel function that satiesfies:

$$K(x, c_i) \ge 0, \sum_{i=1}^{n} K(x, c_i) = 1$$

$$K(x, y) = 0 \text{ if } \rho(x, y) \ge h, \qquad \forall x \in \mathcal{X}, y \in \mathcal{X}_h$$

From given set: $\mathcal{Z}_h := \mathcal{X}_h \times \mathcal{A}$
we can get value: $q = \{q(c_i, a), c_i \in \mathcal{X}, a \in \mathcal{A}\}$
then we can have the Q function:

$$\tilde{Q}(x, a) = (\Gamma_{\mathrm{NN}}q)(x, a), \quad \forall (x, a) \in \mathcal{Z}.$$

The Objective of Q-learning is to learn to find the optimal Q function:

$$(Gq)(c_i, a) \triangleq (F\Gamma_{\mathrm{NN}}q)(c_i, a) = (F\tilde{Q})(c_i, a) = r(c_i, a) + \gamma\mathbb{E}\left[\max_{b \in \mathcal{A}}(\Gamma_{\mathrm{NN}}q)(x', b) \mid c_i, a\right]$$

Assume the covering time in finite($L_h < \infty$) such that
$E[\tau_{\pi,h}(x, t)] \le L_h, \forall x \in X, t > 0$

3. Proposition

3.1 Proposition1

Proposition 1: Start at a random state, by using any policy after some step it must can arrived at every Bi, expected upper bound covering time should be stricter.

Proposition 1. *Suppose that the MDP satisfies the following: there exists a probability measure ν on $\mathcal{X}$, a number $\varphi > 0$ and an integer $m \ge 1$ such that for all $x \in \mathcal{X}$, all $t \ge 0$ and all policies μ,*

$$\Pr_\mu(x_{m+t} \in \cdot | x_t = x) \ge \varphi\nu(\cdot). \tag{6}$$

Let $\nu_{\min} \triangleq \min_{i \in [n]} \nu(\mathcal{B}_i)$, where we recall that $n \equiv N_h = |\mathcal{X}_h|$ is the cardinality of the discretized state space. Then the expected covering time of ε-greedy is upper bounded by $L_h = O\left(\frac{m|\mathcal{A}|}{\varepsilon\varphi\nu_{\min}}\log(n|\mathcal{A}|)\right)$.

3.2 Proposition2

Proposition 2: Begin at any state, there should be existing an action series, after some steps must have a probability for reaching every single Bi, upper bound for covering time should be looser.

Proposition 2. *Suppose that the MDP satisfies the following: there exists a probability measure ν on $\mathcal{X}$, a number $\varphi > 0$ and an integer $m \ge 1$ such that for all $x \in \mathcal{X}$, all $t \ge 0$, there exists a sequence of actions $\hat{\mathbf{a}}(x) = (\hat{a}_1, \ldots, \hat{a}_m) \in \mathcal{A}^m$,*

$$\Pr(x_{m+t} \in \cdot | x_t = x, a_t = \hat{a}_1, \ldots, a_{t+m-1} = \hat{a}_m) \ge \varphi\nu(\cdot). \tag{7}$$

Let $\nu_{\min} \triangleq \min_{i \in [n]} \nu(\mathcal{B}_i)$, where we recall that $n \equiv N_h = |\mathcal{X}_h|$ is the cardinality of the discretized state space. Then the expected covering time of ε-greedy is upper bounded by $L_h = O\left(\frac{m|\mathcal{A}|^{m+1}}{\varepsilon^{m+1}\varphi\nu_{\min}}\log(n|\mathcal{A}|)\right)$.

4. Proposed approaches

After obtaining every single sample point, the information is update and store in the Bellman backup $G_q(c_i, a)$. After every element in Z_h has been visited, $q(c_i, a)$ will be softly update closer toward $G_q(c_i, a)$.

Policy 1 Nearest-Neighbor Q-learning

Input: Exploration policy π, discount factor γ, number of steps T, bandwidth parameter h, and initial state Y_0.
Construct discretized state space $\mathcal{X}_h$; initialize $t = k = 0,\ \alpha_0 = 1,\ q^0 \equiv 0$;
Foreach $(c_i, a) \in \mathcal{Z}_h$**, set** $N_0(c_i, a) = 0$; **end**
repeat
  Draw action $a_t \sim \pi(\cdot|Y_t)$ and observe reward R_t; generate the next state $Y_{t+1} \sim p(\cdot|Y_t, a_t)$;
  Foreach i **such that** $Y_t \in \mathcal{B}_i$ **do**
    $\eta_N = \frac{1}{N_k(c_i,a_t)+1}$;
    if $N_k(c_i, a_t) > 0$ **then**
      $(G^k q^k)(c_i, a_t) = (1 - \eta_N)(G^k q^k)(c_i, a_t) + \eta_N \left(R_t + \gamma \max_{b \in \mathcal{A}}(\Gamma_{\mathrm{NN}} q^k)(Y_{t+1}, b)\right)$;
    else $(G^k q^k)(c_i, a_t) = R_t + \gamma \max_{b \in \mathcal{A}}(\Gamma_{\mathrm{NN}} q^k)(Y_{t+1}, b)$;
    end
    $N_k(c_i, a_t) = N_k(c_i, a_t) + 1$
  end
  if $\min_{(c_i,a) \in \mathcal{Z}_h} N_k(c_i, a) > 0$ **then**
    Foreach $(c_i, a) \in \mathcal{Z}_h$ **do**
      $q^{k+1}(c_i, a) = (1 - \alpha_k) q^k(c_i, a) + \alpha_k (G^k q^k)(c_i, a)$;
    end
    $k = k + 1; \alpha_k = \frac{\beta}{\beta + k}$;
    Foreach $(c_i, a) \in \mathcal{Z}_h$ **do** $N_k(c_i, a) = 0$; **end**
  end
  $t = t + 1$;
until $t \geq T$;
return $\hat{q} = q^k$

5. Experiment Result

We reproduce the work using [1] repository then we tried to modify it to suit the NNQL algorithm. From the figure shown below, the y-axis is the reward and the x-axis is the epoch. From the beginning we can see that the flappy bird keeps on failing causing the reward keep on dropping at the training stage, then after a few epochs later we can see that the frequency of failing has decrease a lot, this show that NNQL algorithm work pretty well even at a limited time.
Code Link:
https://github.com/petertay1996/RL_TheoryProject

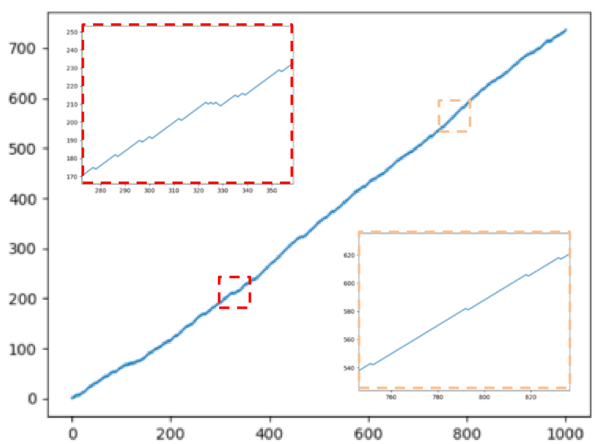
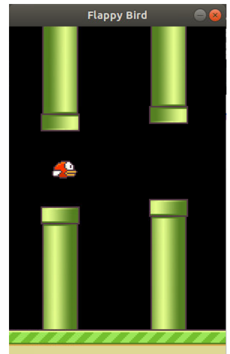


Figure 1: Relationship between reward and epoch of the Flappy Bird



REFERENCES

[1] yenchenlin. Using Deep Q-Network to Learn How To Play Flappy Bird
https://github.com/yenchenlin/DeepLearningFlappyBird

[2] Shah, Devavrat, and Qiaomin Xie. Q-learning with nearest neighbors. Advances in Neural Information Processing Systems. 2018.