10 - Reinforcement Learning

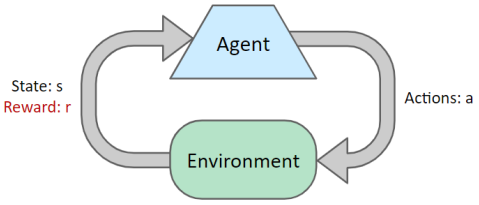
Khái niệm

Important ideas

- Khám phá (exploration): trải nghiệm những hàm động mới để lấy thông tin
- Khai thác (exploitation): sử dụng những trải nghiệm sẵn có → đưa ra hành động tiếp theo
- Hối hận (regret): thử sai
- Lấy mẫu (sampling): vì yếu tố ngẫu nhiên nên phải thực hiện hành động nhiều lần
- Sự khó khăn (difficulty): RL rất khó

Basic ideas

- MDP - offline, RL-online (có sự tương tác với môi trường)
- Tiếp nhận feedback từ môi trường dưới dạng phần thưởng ngay lập tức và trạng thái tiếp theo

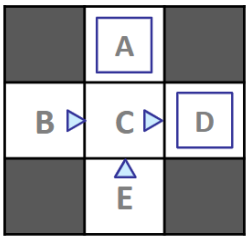


- Agent sẽ quyết định ở trạng thái tiếp theo sẽ làm gì
- Mục tiêu ở mỗi bước sao cho cực đại hóa được tổng điểm thưởng kì vọng khi xuất phát từ trạng thái s trong quá khứ

⇒ có yếu tố Markov nhưng không có P và R

Model-based Learning

- Trong học tăng cường, mục tiêu vẫn là tìm kiếm chiến lược tối ưu, tuy nhiên ta lại không có P (phân phối xác suất) và R (điểm thưởng) → bắt buộc phải tương tác với môi trường, tìm cách xấp xỉ P và R, học ra P và R từ trải nghiệm
- Gỉa sử có 1 chiến lược π, để xấp xỉ P và R, sẽ dùng chiến lược chơi nhiều ván chơi → dựa trên nhiều ván chơi, xấp xỉ ra P và R → chạy lại policy evaluation dựa trên mô hình vừa được xấp xỉ ⇒ chơi càng nhiều, đánh giá chiến lược càng chính xác
- Ví dụ:



Assume: γ = 1

giả sử có chiến lược sau

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

thực hiện 4 lần chơi
vì cả 2 lần B → C đều được nên 100%
4 lần C →D nhưng lần 4 thất bại (C→A thay vì D) = 75%

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

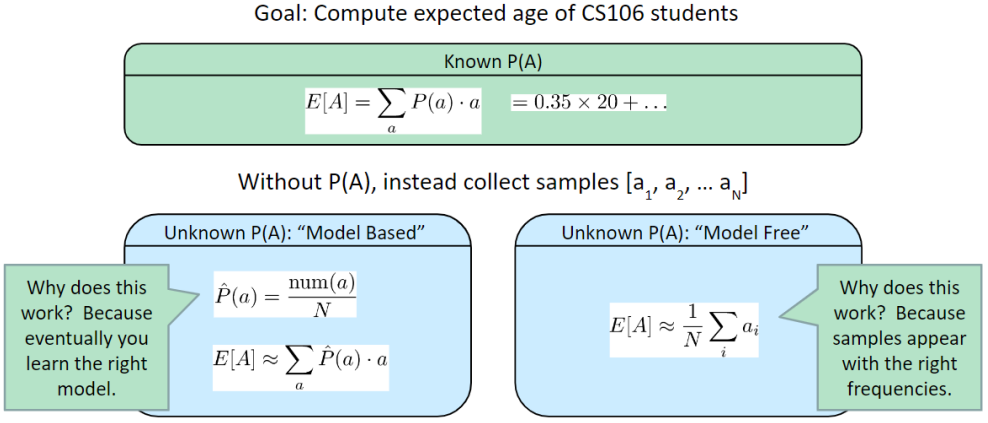
$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

học được P và R sau 4 lần chơi

Model-free Learning

-
- Tính giá trị kì vọng trực tiếp không cần thông qua phân bố xs
  - Ví dụ: ước lượng độ tuổi kì vọng của sinh viên



không cần xấp xỉ P như model-based, model-free lấy tổng những mẫu sau đó chia cho N

⇒ Đa số thuật toán tăng cường là model-free, nhưng các thuật toán sota thì lại là model-based

Passive Reinforcement Learning

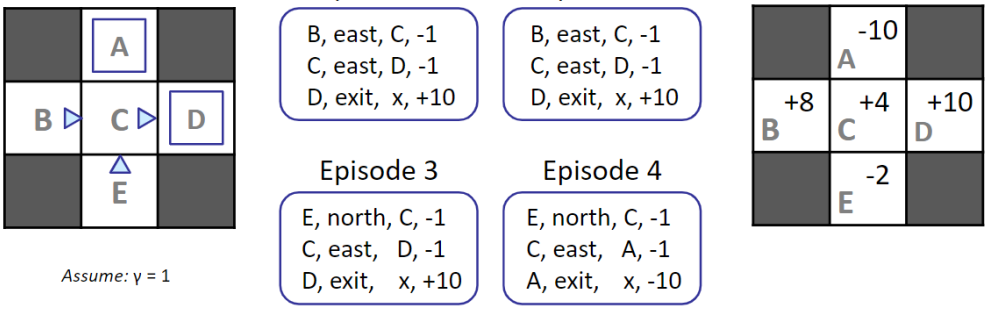
- Có chiến lược π → tính $V^{\pi}(s)$
- Nhưng không biết P và R

Direct Evaluation - Monte Carlo

- Thuật toán: Xấp xỉ trực tiếp ra V^{π} không cần thông qua P và R dùng model-free

- Áp dụng chiến lược π nhiều lần
- Với mỗi trạng thái s và mỗi lần t mà s được thăm → xác định phần thưởng ngay lập tức $r_{t+1}, r_{t+2}, ..., r_t$
- Lấy mẫu cho trạng thái s ở thời điểm t = tổng giá trị phần thưởng chiết khấu $sample = r_{t+1} + \gamma R_{t+2}(s')$
- Lấy trung bình mẫu

- Ví dụ:



Vpi(A) = -10
Vpi(B) = [(-1-1+10)+(-1-1+10)]/2 = +8 || Vpi(B) = [(-1 -1×0.9+0.9^2×10)+(-1-1×0.9+0.9^2×10)]/2
Vpi(C) = [(-1+10)+(-1+10)+(-1+10)+(-1-10)]/4 = +4 || Vpi(C) = [(-1+0.9×10)+(-1+0.9×10)+(-1+0.9×10)+(-1-0.9×10)]/4
Vpi(D) = [10 + 10 + 10]/3 = 10
Vpi(E)=[(-1-1+10)+(-1-1-10)]/2 = -2

- Nhược điểm:
  - Chơi hết 1 ván chơi mới cập nhật → lâu
  - Chơi càng nhiều ước lượng càng chính xác → tốn chi phí tính toán
  - Dù B → C và E→ C cùng đều tốn 1 bước nhưng V(B) = 8 mà V(E) = -2 (đáng lẽ phải bằng nhau vì điểm thưởng mỗi bước đi bằng nhau và hệ quả tương lai đều đến được C) → do 'thiếu kinh nghiệm'

Mô phỏng policy evaluation khi không có P và R?

- Policy evaluation:

$$V_0^\pi(s) \leftarrow 0$$
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

- Ý tưởng: lấy mẫu kết quả từ s' (bằng cách thực hiện hành động) và lấy trung bình
  - Khởi tạo hệ quả tương lai = 0
  - Lấy mẫu: làm hành động $\pi(s)$ nhiều lần (sau khi có s'_1 thì quay lại s thực hiện lại để ra được s'_n)
    - Hệ quả tương lai chưa biết thì lấy kết quả của vòng lặp trước

$$sample_1 = R(s, \pi(s), s'_1) + \gamma V_k^\pi(s'_1)$$
$$sample_2 = R(s, \pi(s), s'_2) + \gamma V_k^\pi(s'_2)$$
$$...$$
$$sample_n = R(s, \pi(s), s'_n) + \gamma V_k^\pi(s'_n)$$

- Lấy trung bình các mẫu:

$$V_{k+1}^\pi(s) \leftarrow \frac{1}{n}\sum_i sample_i$$

⇒ Xấp xỉ được mối quan hệ các trạng thái → cần ít chi phí hơn Monte Carlo

- Nhược điểm:
  - Không thực tế vì sau khi ra được s'_1, ta cần phải quay lại s để lấy mẫu lần nữa → cần can thiệp vào source code thì mới điều khiển được agent xuất hiện ở bất kì trạng thái nào của game

Temporal Difference Learning

- Cập nhập $V^\pi(s)$ theo diễn biến trò chơi - đi 1 bước, học 1 bước
- Mỗi bước tạo 1 mẫu dữ liệu và dùng mẫu dữ liệu đó để cập nhập

- Mẫu = thông tin chính xác (R) + thông tin không chính xác (hệ quả tương lai) → mẫu là thông tin không chính xác → **Label**
- Cập nhật $V^\pi(s)$: giá trị kì vọng → **Output**

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to V(s): $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha)sample$

Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

alpha = learning rate → khi có trải nghiệm mới, muốn học trải nghiệm mới này bao nhiêu % để cập nhập

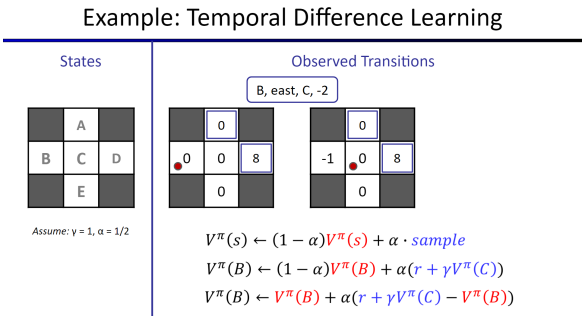
- Nếu alpha = 1 → $V^\pi(s) = sample$ → tất cả những gì trong quá khứ quên đi hết, chỉ biết trải nghiệm hiện tại
- Nếu alpha = 0 → không học được gì cả

⇒ Phát sinh dữ liệu, gán nhãn trong lúc học nhưng gán nhãn sai

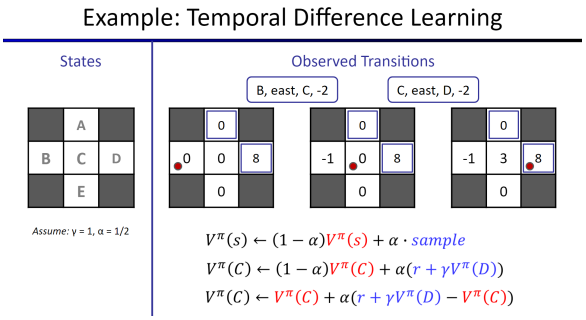
- Thuật toán:

1. Initialize each $V^\pi(s)$ with some value.
2. Observe experience (s, π (s), r, s').
3. Use observation in rough estimate of long-term reward $V^\pi(s)$
$$sample = r + \gamma V^\pi(s')$$
4. Update $V^\pi(s)$ by moving values slightly toward estimate:
$$V^\pi(s) \leftarrow V^\pi(s) + \alpha\big(sample - V^\pi(s)\big)$$

Có thể khởi tạo Vpi(s) bất kì giá trị nào vì công thức cập nhật giống với momentum (vector gradient trung bình có gắn trọng số, thông tin xa thì trọng số khiến nó nhỏ dần đi → quên dần đi)



pi(B) right → reward=-2 → update Vpi(B)
Vpi(B)=0+1/2x(-2+1×0-0)= -1
khởi tạo Vpi(D)=8 (khởi tạo bằng mấy cũng được)



Vpi(C) = 0+1/2x(-2+1×8-0)=3

- Nhận xét:
  - Có khả năng đánh giá được chiến lược bằng cách tính $V^\pi(s)$ nhưng không có khả năng tính được π^*
  - Để lấy max được V của các hành động a khác nhau trong 1 trạng thái s, gần như cần phải có source của game, vì sau khi thực hiện a1 trong s thì qua s', cần phải quay lại s để thực hiện a2
  - Dù ra được $V^*(s)$ nhưng không thể ra được π^* vì chạy policy extraction (cần thông tin P và R)

⇒ Active Learning: đi tìm Q^* thay vì V^*

Active Reinforcement Learning

Q - Value Iteration

- Chiến lược: đi tìm $Q^*(s,a)$
- Phương trình Bellman tối ưu:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V_k^\pi(s')]$$

⇒ Chuyển $V^*(s)$ thành $Q^*(s,a)$

$$Q_{k+1}(s) \leftarrow \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k^*(s',a')]$$

- $V^*(s') \rightarrow Q_k^*(s', \pi^*(s')) \rightarrow \max_{a'} Q_k^*(s',a')$
  - $Q_k^*(s',a')$ đã xác định ở vòng lặp thứ k, nếu vòng lặp đầu thì khởi tạo bất kì giá trị nào
- TD learning: đi 1 bước, cập nhập Vpi(s)

  Q learning: đi 1 bước, cập nhập Q*(s,a)

Q - Learning

- Học ra Q*(s,a)
- Q-table: mỗi hàng là 1 trạng thái, mỗi cột là 1 hành động → mỗi ô là giá trị Q*(s,a)

1. Initialize $Q(s,a) = 0$ for each s,a pair
2. Select an action an observe an experience (s,a,r,s')
3. Consider your old estimate: $Q(s,a)$
4. Use observation in rough estimate of $Q(s,a)$
$$sample = r + \gamma \max_{a'} Q(s', a')$$
5. Update $Q(s,a)$ by moving values slightly toward estimate
$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big)$$
$$= (1-\alpha)Q(s,a) + \alpha \cdot sample$$

thường sẽ không có dấu * vì chưa là tối ưu

- Bắt đầu, trạng thái s0 → **chọn hành động** → quan sát (môi trường trả về s' và R) → gán nhãn = điểm thưởng lập tức + hệ quả (hệ quả không có nhưng đã được khởi tạo ở Q table, một lúc sau thì giá trị này khác 0 vì Q table đã được điền) → cập nhật Q*(s,a) → đang ở s' → quay lại bước 2
- Nếu chạy đủ lâu thì sẽ điền được Q table → Q*(s,a) → policy extraction lấy chiến lược tối ưu

ϵ-Greedy Policies

- Chiến lược lựa chọn hành động
- Dựa vào tỉ lệ thắng của những hành động ở những ván chơi trước mà sẽ ưu tiên thực hiện hành động
- Có bảng Q table → lấy argmax Q(s,a) sẽ ra được hành động có tỉ lệ thắng cao nhất
- Nhưng nếu chỉ argmax thì sẽ không thể khám phá được map → phải chọn hành động ngẫu nhiên để điền được hết Q table
- Sau khi học được Q*(s,a) thì không cần ngẫu nhiên nữa mà lấy argmax, ý nghĩa của hành động ngẫu nhiên là để học

⇒ Cân bằng giữa exploration (khám phá) và khai phá (exploitation)

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a), & \text{with prob. } 1 - \varepsilon + \frac{\varepsilon}{|A|} \\ a' \neq \arg\max_a Q(s,a), & \text{with prob. } \frac{\varepsilon}{|A|} \end{cases}$$

trong đó |A| là số lượng hành động

- Phải chia |A| vì có khả năng rơi trúng trường hợp argmax

Q - Learning với ϵ-greedy exploration

Với mỗi step trong episode:

1. Initialize $Q(s,a) = 0$ for each s, a pair, $t = 0$
2. Set $\pi_b(s) = \arg\max_{a'} Q(s,a)$ with prob. $1-\varepsilon$, else random.
3. Loop
    1. Sample action a from policy π_b an observe an experience (s,a,r,s')
    2. Consider your old estimate: $Q(s,a)$
    3. Use observation in rough estimate of $Q(s,a)$

$$sample = r + \gamma \max_{a'} Q(s',a')$$

    4. Update $Q(s,a)$ by moving values slightly toward estimate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big) = Q(s,a) + \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$$

$$\pi_b(s) = \arg\max_{a'} Q(s,a) \text{ with prob. } 1-\varepsilon, \text{else random.}$$

    5. $s \leftarrow s'$

```
q_table = np.zeros((env.observation_space.n, env.action_space.n))
rewards_all = []
for episode in range(num_episodes):
    state, _ = env.reset()
    reward_episode = 0.0
    done = False
    epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-epsilon_decay_rate*episode)
    for step in range(num_steps_per_episode):
        exploration = random.uniform(0,1)
        if exploration < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(q_table[state, :])
        next_state, reward, terminated, truncated, _ = env.step(action)
        q_table[state, action] = q_table[state, action] * (1 - learning_rate)
                                 + learning_rate * (reward + gamma * np.max(q_table[next_state,:]))
        reward_episode += reward
        state = next_state
        if terminated or truncated:
            done = True
        if done:
            break
    rewards_all.append(reward_episode)
```

- Sử dụng π_b để chọn hành động a → quan sát môi trường trả về s′ và r → tiến hành gán nhãn dựa và Q table và s′ → cập nhật Q(s,a) → ⇒ xong bước 4 là đang ở s′ (gán s ← s′)
- a′ chỉ xuất hiện ở argmax gán nhãn, π_b không đụng gì vào a′ hết

SARSA với ϵ-greedy exploration

Với mỗi step trong episode:

1. Initialize $Q(s,a) = 0$ for each s, a pair, $t = 0$
2. Set $\pi_b(s) = \arg\max_{a'} Q(s,a)$ with prob. $1-\varepsilon$, else random.
3. Sample action a from policy π_b an observe an experience (s,a,r,s')
4. Loop
    1. Sample action a' from policy π_b an observe an experience (s',a',r'',s'')
    2. Consider your old estimate: $Q(s,a)$
    3. Use observation in rough estimate of $Q(s,a)$

$$sample = r + \gamma Q(s',a')$$

    4. Update $Q(s,a)$ by moving values slightly toward estimate

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big) = Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$

$$\pi_b(s) = \arg\max_{a'} Q(s,a) \text{ with prob. } 1-\varepsilon, \text{else random.}$$

    5. $s \leftarrow s', a \leftarrow a'$

```
q_table = np.zeros((env.observation_space.n, env.action_space.n))
rewards_all = []
for episode in range(num_episodes):
    state, _ = env.reset()

    reward_episode = 0.0
    done = False
    epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-epsilon_decay_rate*episode)
    exploration = random.uniform(0,1)
    if exploration < epsilon: action = env.action_space.sample()
    else: action = np.argmax(q_table[state, :])

    for step in range(num_steps_per_episode):
        next_state, reward, terminated, truncated, _ = env.step(action)

        exploration = random.uniform(0,1)
        if exploration < epsilon: next_action = env.action_space.sample()
        else: next_action = np.argmax(q_table[next_state, :]) # a'

        q_table[state, action] = q_table[state, action] +
                    learning_rate * (reward + gamma * q_table[next_state, next_action] - q_table[state, action])
        reward_episode += reward
        state, action = next_state, next_action
        if terminated or truncated: break

    rewards_all.append(reward_episode)
```

- a′ ở Q learning không phụ thuộc π_b vì nó chỉ argmax Q table nhưng a′ ở trong vòng lặp 1 Sarsa phụ thuộc π_b vì chọn a′ từ π_b
- gán nhãn bằng r của hành động a và hệ quả tương lai (hệ quả này không có max vì a′ phụ thuộc π_b chứ không phải lấy argmax) → cập nhật Q(s,a)
- Ở vòng lặp tiếp theo, Sarsa sẽ cập nhật Q(s′,a′)
- công thức gán nhãn này không còn là Q* vì a′ tạo ra từ $\pi_b(s')$, tức trong trạng thái s làm hành động a, từ s′ trở đi tuân theo π_b → $Q^{\pi_b}(s,a)$

⇒ bản chất: đi tìm $Q^{\pi_b}(s,a)$

- Nhưng Sarsa khởi tạo π_b bằng argmax Q(s,a) với xác suất lớn nên Sarsa không học ra π^* mà học π_b nhưng học đủ lâu thì π_b hội tụ về π^* (để được điều này thì ϵ phải giảm dần để càng về sau nó sẽ ít khi chọn hành động ngẫu nhiên nữa)

Nhận xét

- **Off Policy - Q Learning:** dùng chiến lược π_b phát sinh ra hành động để từ đó tính ra π^* (tính Q* (s,a)) là gì

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

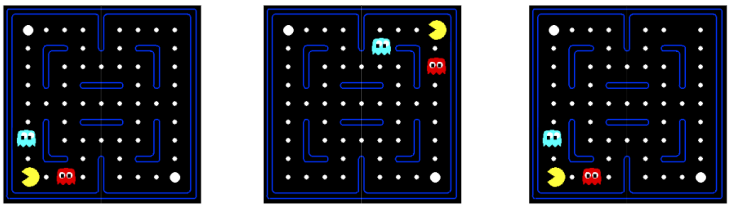
- **On Policy - SARSA:** dùng chiến lược π_b tạo ra hành động để đánh giá π_b (tính $Q^{\pi_b}(s,a)$)

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

- Vì giải quyết bài toán dựa trên Q table nên chỉ giải quyết được các bài toán nhỏ, rời rạc

Approximate Q - Learning

- Có những tình huống giống nhau nhưng trạng thái lại khác nhau → mất chi phí để học dù kết quả như nhau



Feature-based Representation

- Biểu diễn trạng thái thông qua các đặc trưng theo mình là quan trọng nhất, như: khoảng cách giữa closest ghost tới pacman, ...
- Có hàm truyền vào trạng thái → rút trích đặc trưng

⇒ dùng feature vector

- Biểu diễn giá trị trạng thái và giá trị hành động thành các **hàm tuyến tính**

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$$
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + ... + w_n f_n(s,a)$$

- Không phải học tăng cường chính xác mà chỉ là xấp xỉ học tăng cường vì ta chỉ dùng những đặc trưng quan trọng nhất chứ không phải toàn bộ đặc trưng → approximate → đi tìm trọng số w → tương tự hồi quy tuyến tính
- Nếu xấp xỉ bằng hàm tuyến tích, rút trích đặc trưng của trạng thái sau khi thực hiện hành động a trong trạng thái s
- Không có nhãn thật → không thể làm hồi quy tuyến tính
- Thuật toán:

$$\text{transition} = (s,a,r,s')$$
$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}] \quad \text{Exact Q's}$$
$$w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(s,a) \quad \text{Approximate Q's}$$

- Ở Q learning, tìm max Q(s',a') ta dựa vào Q table → không có Q table, lấy hàm Q(s,a) thế từng hành động a' xong lấy max → lấy sample - Q(s,a) (sự khác biệt giữa label và output của mô

hình, label chỉ có 1 phần thông tin chính xác là điểm thưởng lập tức, còn phần hệ quả không là thông tin chính xác vì lấy ra từ mô hình đang học) → vì không có Q table nên thay vì cập nhật Q(s,a), ta sẽ cập nhật trọng số w

- Khi đã hội tụ (tìm được Q*(s,a)) → difference = 0 nhưng nếu đang học thì không thể bằng 0 và do đang sử dụng những đặc trưng quan trọng nhất chứ không phải toàn bộ đặc trưng

⇒ tìm w sao cho difference càng nhỏ càng tốt

- Tại sao phải nhân với $f_i(s,a)$ vì liên quan đến đạo hàm hàm lỗi bình phương

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s',a') - Q(s,a)\right] f_m(s,a)$$

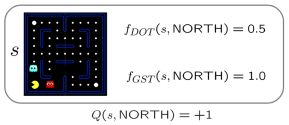
"target" "prediction"

- Nếu chạy đủ lâu thì nhãn sẽ được gán chính xác dần

- Ví dụ:

Example: Q-Pacman

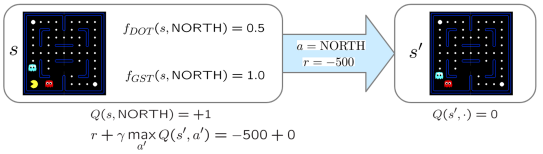
$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



Kết quả mô hình dự đoán Q(s, NORTH) = 4×0.5-1×1=1

Example: Q-Pacman

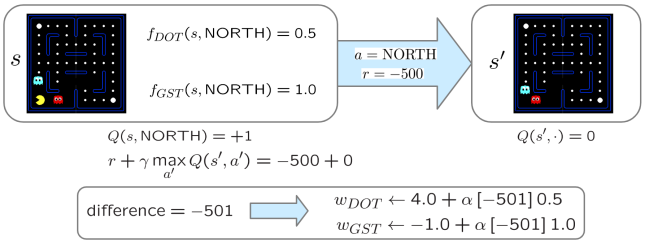
$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



Gán nhãn dữ liệu → vì trò chơi kết thúc (pacman bị ăn)
→ hệ quả = 0
→ Nhãn = r + hệ quả = -500 + 0 = -500

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

tính difference = -500-1 = -501 → update trọng số

Từng ra thi

- Nếu chọn số đặc trưng đúng bằng số ô trong Q table thì khi đó approximate Q learning sẽ thành exact Q learning → chạy approximate sẽ hội tụ về đúng Q*. Chuyển Q Learning từ dạng bảng về dạng tuyến tính → chạy đủ lâu thì sẽ hội tụ về Q*