12 - Deep Reinforcement Learning

RL Recap

- Thay vì tạo ra đặc trưng thủ công như Approximate Q Learning, thì Deep Q Learning sẽ tự học ra được đặc trưng
- Input: trạng thái (chứ không phải vector đặc trưng)
- Bài toán Video Summarization: 1 trạng thái xem như 1 frame truyền vô → tạo ra agent quyết định frame này có được đưa vô bản tóm tắt hay khong. Nhưng thực ra 1 frame là không đủ bởi vì 1 trạng thái phải được tạo ra từ frame trước đó (và cả sau đó) → sử dụng RNN hoặc LSTM để lưu thông tin frame trước đó, nếu lưu frame sau đó nữa thì dùng bidirectional LSTM để đưa ra quyết định
- 1 agent phải có ít nhất 1 trong các thành phần sau: chiến lược giúp agent chọn lựa hành động, 1 hàm giá trị giúp đánh giá trạng thái, hành động, 1 mô hình học được môi trường
  - policy: ánh xạ từ trạng thái đến hành động
    - deterministic policy (đơn định): không có yếu tố ngẫu nhiên (input: trạng thái ⇒ output: hành động)
    - stochastic policy: có yếu tố ngẫu nhiên (input: trạng thái ⇒ output: phân bố xác suất)
      - phát sinh 1 số ngẫu nhiên
        - VD: P(left) = 0.8, P(right) = 0.2 ⇒ phát sinh r ~ uniform(0,1)
          - nếu r < 0.8 ⇒ thực hiện sang trái
          - ngược lại ⇒ sang phải
  - value function: dự đoán điểm thưởng tương lai nếu thực hiện hành động a trong trạng thái s
    - Q value: điểm thưởng kì vọng có điều kiện

$$Q^{\pi}(s,a) = E[r_{t+1} + \gamma_r t + 2 + \gamma^2 r_{t+3} + ...|s,a]$$

    - Bellman equation:

$$Q^{\pi}(s,a) = E_{s',a'}[r + \gamma Q^{\pi}(s',a')|s,a]$$

    - Nếu có hàm giá trị tối ưu:
      - Tìm trong tất cả chiến lược có thể có, đâu là chiến lược có hàm giá trị hành động chiến lược π lớn nhất → hàm giá trị hành động tối ưu

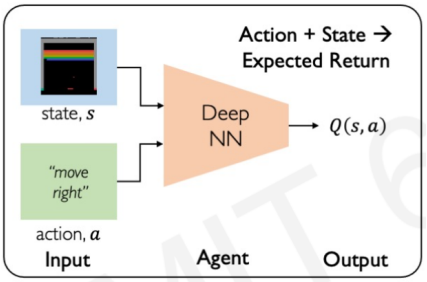
$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a) \rightarrow \pi^* = \arg\max$$
$$Q^*(s,a) = r_{r+1} + \gamma max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$
$$\rightarrow Q^*(s,a) = E_{s'}[r + \gamma max_{a'} Q^*(s',a')|s,a]$$

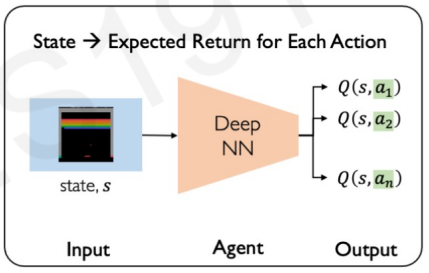
  - model: môi trường

Value-based Reinforcement Learning - DQN

- Học mạng neuron (Q Networks) → học ra hàm giá trị hành động $Q^*(s,a)$

- Có 2 loại thiết kế mạng neuron:
  - Sau khi đã có mạng neuron (Q*(s,a)), mạng neuron nhận vào trạng thái s và hành động a → output là giá trị $Q(s,a)$ → cuối cùng là lấy argmax các giá trị Q(s,a) tương ứng mỗi hành động để ra được chiến lược ⇒ nếu có n hành động thì cần inference n lần
    - Không cần dùng ϵ để lựa chọn hành động vì đây là đã học được rồi, còn random dùng để học trải nghiệm huấn luyện mô hình
  - Kiểu thiết kế khác **phổ biến hơn** với input chỉ là 1 trạng thái, output là nhiều $Q(s,a_i)$ tương ứng các hành động khác nhau → lấy argmax các output ⇒ inference 1 lần





- Định nghĩa giá trị hành động: trong trạng thái s làm hành động a và sau đó tuân theo chiến lược tối ưu, giá trị kì vọng nhận được gồm điểm thưởng lập tức + hệ quả tương lai, nên từ s_{t+1} phải làm hành động cực đại hóa được giá trị hành động

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

- Huấn luyện được trọng số w mà không có dữ liệu → phải cho thuật toán tương tác với môi trường → tạo ra trải nghiệm: điểm dữ liệu → tiến hành gán nhãn dữ liệu theo định nghĩa giá trị hành động: $r + \gamma \max_{a'} Q(s',a',w)$
- Nhưng công thức trên khi đã có Q(s,a) thực sự, nhưng ta không có Q(s,a) thực sự do ta đang tìm kiếm nó, nên ta phải dùng chính mô hình đang huấn luyện để gán nhãn nó

Nhận xét

- Nên sử dụng design 2 bởi vì trong quá trình gán nhãn dữ liệu, ta dùng chính mạng Q để gán nhãn, design 1 khiến việc gán nhãn lâu vì cần phải lặp lại nhiều lần đưa input vào Q ra output tương ứng các hành động, trong khi design 2 thì chỉ cần đưa 1 lần và có output tất cả các giá trị

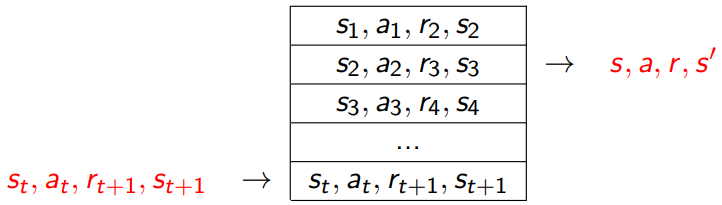
Vấn đề của Q-Networks

- Ta dùng mạng neuron tính Q(s,a) → huấn luyện cực tiểu hóa mean square error như stochastic gradient descent (cập nhật trên 1 điểm dữ liệu)

$$l = \left(r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w})\right)^2$$

1. Có phần thông tin chính xác là r nên chạy càng lâu thì càng chính xác dần → nhưng lại không huấn luyện nổi vì đã vi phạm qui tắc trong máy học: các **điểm dữ liệu** phải **độc lập** với nhau (**correlations**)

⇒ Giải quyết bằng **Replay Buffer:**
lưu lại trạng thái s1, hành động a1,
nhận được r1 và qua được s2
(s1,a1,r1,s2) mỗi lần chạy



- Lấy mạng neuron làm đầy buffer rồi mới học, thực hiện việc học như minibatch gradient descent:
  - Lấy một minibatch trong buffer → dùng minibatch tính vector gradient
    - Đi 1 bước → ra được 1 điểm dữ liệu nhưng không dùng điểm dữ liệu đó để huấn luyện như trên mà lấy minibatch ngẫu nhiên trong buffer

  ⇒ Các điểm dữ liệu cách xa → phụ thuộc yếu

2. Vì ta dùng chính mạng đang huấn luyện gán nhãn, mà mạng này được cập nhật liên tục nên trọng số thay đổi liên tục ⇒ giá trị target bị thay đổi liên tục → ảnh hưởng tới tốc độ chạy (**non-stationary**)

⇒ Giải quyết bằng **Target Network**: 1 mạng được clone từ **Q-Network**

$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

- Sử dụng Target Network để gán nhãn dữ liệu
- Chỉ cập nhật trọng số của Q Network
- Sau **x** steps/iterations, clone lại Target Network = Q Network → coi TargetNet như 1 siêu tham số

Kết luận

- Khi cập nhật trọng số mạng neuron hiện tại, ta sử dụng các điểm dữ liệu trong buffer (mà dữ liệu trong buffer được tạo ra bởi các mạng neuron trước đó- khác trọng số) → dùng các hành động được phát sinh bởi chiến lược khác để cập nhật chiến lược hiện tại

⇒ **Off Policy**

- Để cài đặt kiểu **on policy**, buffer chỉ được chứa những hành động được phát sinh ở mạng neuron hiện tại bằng cách lấy mạng neuron hiện tại phát sinh nhiều trải nghiệm làm đầy buffer → lấy minibatch cập nhật → xóa buffer

Một số DQN khác

Double DQN

- Lựa chọn a′ bằng Q Network $\mathbf{w}$→ đưa a′ này vào Target Network $\mathbf{w}^-$
- Target Network $\mathbf{w}^-$ dùng để đánh giá giá trị hành động

→ hành động thay đổi, target được giữ

$$I = \left(r + \gamma Q(s', \underset{a'}{\text{argmax}}\ Q(s', a', \mathbf{w}), \mathbf{w}^-) - Q(s, a, \mathbf{w})\right)^2$$

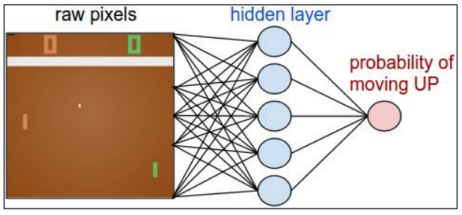
Priorities Replay

- Trải nghiệm đưa vào buffer sẽ được gán trọng số theo công thức độ lỗi giữa target và dự đoán của Q Network → lấy minibatch có sai số ít (điểm dữ liệu tốt) → priority queue

$$\left|r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, w)\right|$$

Policy-based Reinforcement Learning

Policy Network



- Input là trạng thái s → Output là hành động cho trạng thái đó hay phân bố xác suất các hành động cho trạng thái đó

→ tương tự bài toán phân lớp hình ảnh với hàm kích hoạt sigmoid

- Nếu có 1 hidden layer, bức hình 80×80, 200 neuron → tổng cộng [(80×80)x200+200]+[200×1+1]~1.3M tham số

Random Search

1. Vòng lặp:
2. Gán 1.3M giá trị ngẫu nhiên cho mạng neural
3. Chơi 1 vài ván chơi
4. Nếu tốt thì giữ lại tham số, nếu xấu thì bỏ đi

→ Sẽ tới lúc ra được bộ trọng số tốt nhưng rất tốn kém chi phí

⇒ Phải dùng thuật toán khác để tìm kiếm bộ trọng số → Policy Gradient

Policy Gradients

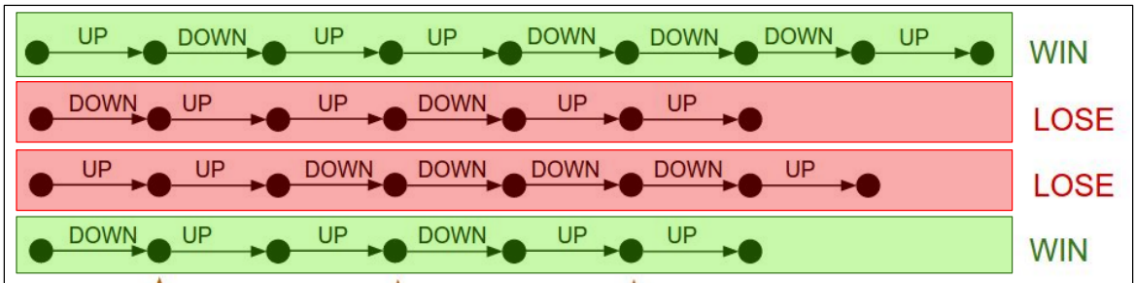
- Gỉa sử có bộ dữ liệu huấn luyện được gán nhãn [trạng thái - hành động] → tương tự bài toán phân lớp nhị phân với hàm loss là binary cross entropy loss/log likelihood

- Cập nhật trọng số mô hình sao cho khi input là tấm hình thì xác suất mô hình dự đoán lớp tấm hình là đúng nhất

$$(-1)\sum_i \log p(y_i|x_i)$$

$$\sum_i \log p(y_i|x_i)$$

- Tuy nhiên ta không có nhãn → thử ngẫu nhiên hành động và xem kết quả
  - Nếu hành động tốt → trong tương lai thử hành động đó nhiều hơn
  - Nếu hành động xấu → giảm xác suất hành động xuất hiện
- Khi lấy policy (trọng số) ra chơi → ta được 1 chuỗi hành động, làm lại nhiều lần
  - Tuy hành động đầu tiên giống nhau nhưng có lúc thắng lúc thua → kết quả ván chơi phụ thuộc vào dây chuyền hành động → ta sẽ không biết hành động nào tốt/xấu
  - Giả sử toàn bộ hành động trong ván chơi thắng là tốt và ngược lại



- Cập nhật trọng số để xét 1 hành động trong ván chơi thắng → xác suất thực hiện hành động đó tăng và ngược lại
  - Nếu ván thắng → maximize log-likelihood
  - Nếu ván thua → minimize log-likelihood

maximize: $\log p(y_i \mid x_i)$ | maximize: $(-1) * \log p(y_i \mid x_i)$

- Trong supervise learning, y_i là nhãn thật, trong RL y_i là hành động mà hành động đó được phát sinh từ policy và ta không thể biết được y_i là hành động tốt hay xấu nên sau khi ván chơi kết thúc, ta sẽ gán nhãn nó

$$y_i \sim p(\cdot|x_i)$$

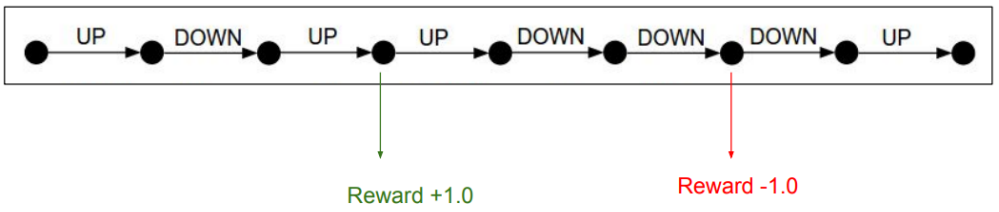
$$\sum_i A_i * \log p(y_i|x_i)$$

  - $A_i = \{-1, 1\}$ tùy thuộc vào kết quả ván chơi (1: thắng, -1:thua)

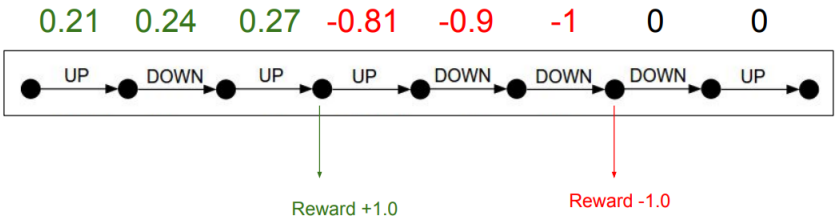
⇒ Cực đại hóa hàm log-likelihood trên → dùng gradient ascend (cập nhật trọng số cùng hướng vector gradient)

- Hành động đầu có vai trò đối với kết quả trận đấu, hành động cuối mới có vai trò lớn → vì thế đánh trọng số đều nhau là không hợp lí ⇒ discounting

Discounting



- Nếu chọn hệ số chiết khấu $\gamma = 0.9$:
  - hành động sau cùng tới thất bại: $a_k = -1 + \gamma \times 0 = -1$
    - $a_{k-1} = \gamma \times a_k = 0.9 \times -1 = -0.9$
    - $a_{k-2} = \gamma \times a_{k-1} = 0.9 \times -0.9 = -0.81$
  - hành động sau cùng tới chiến thắng: $a_{k-3} = 1 + \gamma \times a_{k-2} = 1 + 0.9 \times (-0.81) = 0.27$
    - $a_{k-4} = \gamma \times a_{k-3} = 0.9 \times 0.27 = 0.24$
    - $a_{k-5} = \gamma \times a_{k-4} = 0.9 \times 0.24 = 0.21$



- Kĩ thuật sẽ hơi nguy hiểm khi trong 1 số trò chơi, ví dụ như trong pong, khi banh vượt qua cần gạt, những frame cuối mang trọng số rất cao nhưng lại không có giá trị → đánh trọng số sai

Bellman Equations Recap

- Gía trị hành động $Q^\pi(s,a)$: ở trạng thái s_t làm hành động a_t, có nhiều s′ xảy ra, với mỗi tình huống lấy xác suất x [điểm thường lập tức + hệ quả tương lai theo chiến lược π]. Phụ thuộc vào
  - trạng thái s′ sau khi làm hành động a, p(s′|s,a)
  - điểm thưởng ngay lập tức r(s,a,s′)
  - hệ quả tương lai nhận được bao nhiêu $V^\pi(s')$

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbf{E}_\pi[R_t|s_t = s, a_t = a] \\
&= \mathbf{E}[r_{t+1} + \gamma R_{t+1}|s_t = s, a_t = a] \\
&= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')]
\end{aligned}
$$

- Gía trị trạng thái $V^\pi(s)$. Phụ thuộc:
  - giá trị tất cả trạng thái khác

- chiến lược π hiện tại

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) Q^{\pi}(s,a)$$
$$= \sum_{a \in A} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$

- Bellman equation cho 1 hành động tại 1 trạng thái:

$$Q^{\pi}(s,a) = \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]$$
$$= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma \sum_{a \in A} \pi(a'|s') Q^{\pi}(s',a')]$$

Policy Search

- Nếu π là chiến lược có yếu tố ngẫu nhiên → trả ra phân bố xác suất thực hiện các hành động a
- Học ra trực tiếp π, thường tham số (trọng số) hóa bằng mạng neuron, để khi thay đổi tham số mạng neuron tức là thay đổi policy
- Mục tiêu là cực đại hóa reward $R(\tau)$

$$J(\theta) = \mathbf{E}_{\tau \sim \rho_{\theta}}[R(\tau)] = \mathbf{E}_{\tau \sim \rho_{\theta}}[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1})]$$
$$J(\theta) = \int_{\tau} \rho_{\theta}(\tau) R(\tau) d\tau$$

⇒ Hiệu năng của chiến lược có trọng số θ bằng tổng điểm thưởng kì vọng tính trên những đường đi τ mà ta có thể phát sinh bởi chiến lược π_{θ}

- Một chiến lược tốt tức là có bộ θ tốt → có thể phát sinh được nhiều τ có $R(\tau)$ lớn
- Likelihood của đường đi τ: xác suất đường đi τ xuất hiện là bao nhiêu
  - xác suất trạng thái bắt đầu (do policy không kiểm soát được)
  - s_t có nhiều lựa chọn, xs làm mỗi lựa chọn là bao nhiêu, ứng với mỗi lựa chọn khi làm hành động đó có nhiều khả năng xảy ra nên x tiếp với s_{t+1}

$$\rho_{\theta}(\tau) = p_{\theta}(s_0, a_0, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_{\theta}(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

- Ta không thể xét tất cả τ có thể xảy ra → xấp xỉ bằng Monte Carlo: từ π_{θ} phát sinh nhiều ván chơi → tính kì vọng xấp xỉ J

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

  - N càng lớn → xấp xỉ càng chính xác
  - N càng nhỏ → không ổn định

- Vấn đề Monte Carlo:
  - số ván chơi ít → high variance
  - trò chơi vô hạn → không xấp xỉ được do không thể tính tổng điểm thưởng
- Muốn sử dụng gradient ascent cần có vector gradient nhưng ta không có công thức gradient → tìm kiếm công thức

Thuật toán REINFORCE

- Ta có: vector gradient của hàm tổng điểm thưởng kì vọng theo θ

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau = \int_\tau (\nabla_\theta \rho_\theta(\tau)) R(\tau) d\tau$$

- τ phát sinh bởi θ nên nó phụ thuộc vào θ nhưng tổng reward phát sinh không phụ thuộc $\tau \leftrightarrow$ tổng điểm thưởng phụ thuộc vô két quả môi trường trả về ứng với hành động chứ không phụ thuộc vào mạng neuron
- **log-trick:**

$$\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$$

Áp dụng log-trick ta có grad-log-prob:

$$\nabla_\theta \rho_\theta(\tau) = \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau)$$

Policy gradient trở thành:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) R(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim \rho_\theta} [\nabla_\theta \log \rho_\theta(\tau) R(\tau)] \end{aligned}$$

⇒ Gradient của hàm tổng điểm thưởng kì vọng = tổng điểm thưởng ván chơi x grad-log-prob ván chơi đó → lấy kì vọng

- Muốn tính kì vọng → có thể sử dụng Monte Carlo phát sinh nhiều ván chơi
- Nhưng ta không thể tính được xác suất xuất hiện đường đi $\rho(\tau)$ nhưng có thể tính gradient của nó thông qua log-likelihood

$$\begin{aligned} \log \rho_\theta(\tau) &= \log \left(p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right) \\ &= \log p_0(s_0) + \sum_{t=0}^{H} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{H} \log p(s_{t+1}|s_t, a_t) \end{aligned}$$

  - bởi vì vế đầu không có θ và vế cuối chỉ phụ thuộc vào môi trường (dù có cập nhật trọng số model, nó cũng không ảnh hưởng đến môi trường), nên:

$$\nabla_\theta \log \rho_\theta(\tau) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

⇒ chỉ phụ thuộc vào output mạng neuron do mình điều khiển (chỉ output mạng neuron phụ thuộc vào mạng neuron)

- Model-free learning:

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t=0}^{H} \gamma^t r_{t+1}\right)\right]\end{aligned}$$

⇒ Có thể sử dụng Monte Carlo

- Ý nghĩa vector gradient: hướng của log prob (xs thực hiện những hành động trong ván chơi τ tăng lên) → **score function**
- 1 ván chơi chiến thắng → tổng điểm thưởng lớn (số dương) → scale grad-log-prob lớn (dương) → cập nhật trọng số neuron khiến xác suất thực hiện diễn biến đó lớn
- Thuật toán:

While not converged:

1. Sample N trajectories $\{\tau_i\}$ using the current policy π_θ and observe the returns $\{R(\tau_i)\}$
2. Estimate the policy gradient as an average over the trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i)$$

3. Update the policy using gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

- **REINFORCE - baseline**

While not converged:

1. Sample N trajectories $\{\tau_i\}$ using the current policy π_θ and observe the returns $\{R(\tau_i)\}$
2. Compute the mean return:

$$\hat{R} = \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

3. Estimate the policy gradient as an average over the trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau_i) - \hat{R})$$

4. Update the policy using gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

  - Reducing variance
  - Baseline có thể tự thiết kế

- Việc thêm trừ baseline → vế phải thay đổi thì xấp xỉ gradient còn đúng không? ĐÚNG, việc thêm baseline không làm thay đổi vế trái, mình chỉ quan tâm → phép ước lượng không thiên lệch (unbias)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)(R(\tau) - b)]$$

▶ We have:

$$\begin{aligned}\mathbb{E}_{\tau\sim\rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) b] &= \int_\tau \rho_\theta(\tau)\nabla_\theta \log \rho_\theta(\tau) b d\tau \\ &= \int_\tau \nabla_\theta \rho_\theta(\tau) b d\tau \\ &= b\nabla_\theta \int_\tau \rho_\theta(\tau) d\tau = b\nabla_\theta 1 = 0\end{aligned}$$

▶ If b does not depend on θ, the estimator is unbiased.

- Thuật toán REINFORCE xấp xỉ policy gradient sau khi gán nhãn:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=0}^{H} \gamma^k r(s_k, a_k, s_{k+1})\right)\end{aligned}$$

→ **Causality principle** (quan hệ nhân quả): điểm thưởng nhận được ở bước thứ t không chịu ảnh hưởng bởi hành động mình làm ở tương lai

Ta cập nhật trọng số mạng neuron ở bước thứ t để gia tăng khả năng thực hiện hành động a_t ở bước thứ t → xs việc mình làm hành động a_t bước thứ t không ảnh hưởng tới quá khứ ↔ trong quá khứ mình nhận điểm thưởng là gì, việc thay đổi mạng neuron hiện tại không thể thay đổi quá khứ

⇒ loại bỏ điểm thưởng từ 0 tới t-1

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k, s_{k+1})\right)$$

- Màu đỏ là $Q^\pi(s_t, a_t)$→ không tính được (do mình chỉ có π không có Q) → xấp xỉ bằng mạng neuron → **actor-critic**
  - actor: là θ, policy network thực hiện hành động → policy gradient
  - critic: là $Q^\phi(s, a)$, đánh giá giá trị hành động a trong trạng thái s → mean square error loss → DQN (dùng thêm target critic)