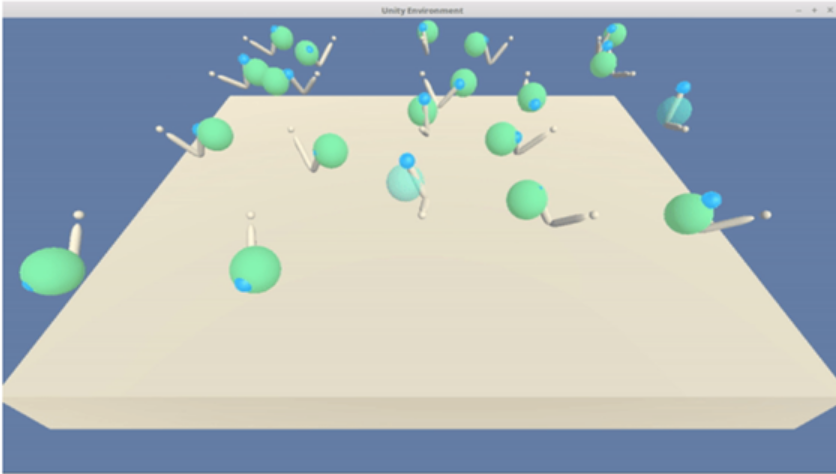
Deep Reinforcement Learning. Deep Deterministic Policy Gradient (DDPG) algorithm.

1. Mục tiêu

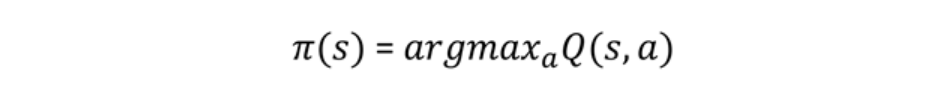
Trong project này, mục tiêu trong việc train 20 agents (mỗi agent điều khiển một cặp cánh tay robot) để giữ vị trí tới địa điểm đích với tất cả các bước chuyển động có thể thực hiện được.

Không gian quan sát bao gồm 33 biến tươn ứng với các vị trí, vòng xoay, vận tốc, và vận tốc góc của cánh tay. Mỗi action là một vector với 4 tham số, tương ứng với momen xoắn của hai khớp. Mỗi mục trong vector action là một số trong khoảng từ -1 tới 1. Yêu cầu của project là đạt điểm trung bình từ +30 tới 100 episodes liên tiếp.



1. Giới thiệu

Trong project trước chúng ta dùng Q function để tìm policy tối ưu ( bằng Q function) gần đúng (sử dụng Q tables và deep neural network) lựa chọn action tốt nhất để biểu diễn từng state một. Thông thường, điều này có thể được công thức hóa là:



có nghĩa là kết quả của policy π tại từng state s là action với Q lớn nhất (value function).

Tuy nhiên, trong tất cả trường hợp, dù chúng ta dùng bảng cho không gian trạng thái nhỏ hay mạng thần kinh cho không gian trạng thái lớn, thì trước tiên chúng ta phải ước tính value function cho hành động tối ưu (optimal) trước khi chúng ta có thể giải quyết optimal policy. Như chúng ta mô tả (trong công thức trên), khi ta dùng Q-learning, ta sẽ cố tối ưu policy bằng cách tập trung vào value function lớn nhất. Ý tưởng gần đây sẽ tìm optimal policy trực tiếp mà không cần lo về value function ở toàn cục.

Có hai lý do vì sao mà policy lại là phương pháp tiếp cận đáng để khai thác:

Đầu tiên, ta chú ý nhiều hơn về tổng reward nhưng không thường xuyên về value function cao nhất (như Deep Q Network) ở từng trạng thái. Để thu được các giá trị này (action và state), ta phải sử dụng đẳng thức Bellman, đẳng thức này biểu diễn giá trị hiện tại nhờ vào các giá trị ở bước kế tiếp. Thông thường, khi chúng ta giải quyết các vấn đê thực tế, phức tạp , các agent có được sự quan sát từ environment và cần quyết định xem nên làm gì giếp theo., ta cần policy, chứ không phải là giá trị của state hoặc hành động cụ thể. Ta cần biết xem nên làm gì tiếp theo tại mỗi bước (action được thực thi dựa trên giá trị cao nhất được tối ưu hóa thường xuyên).

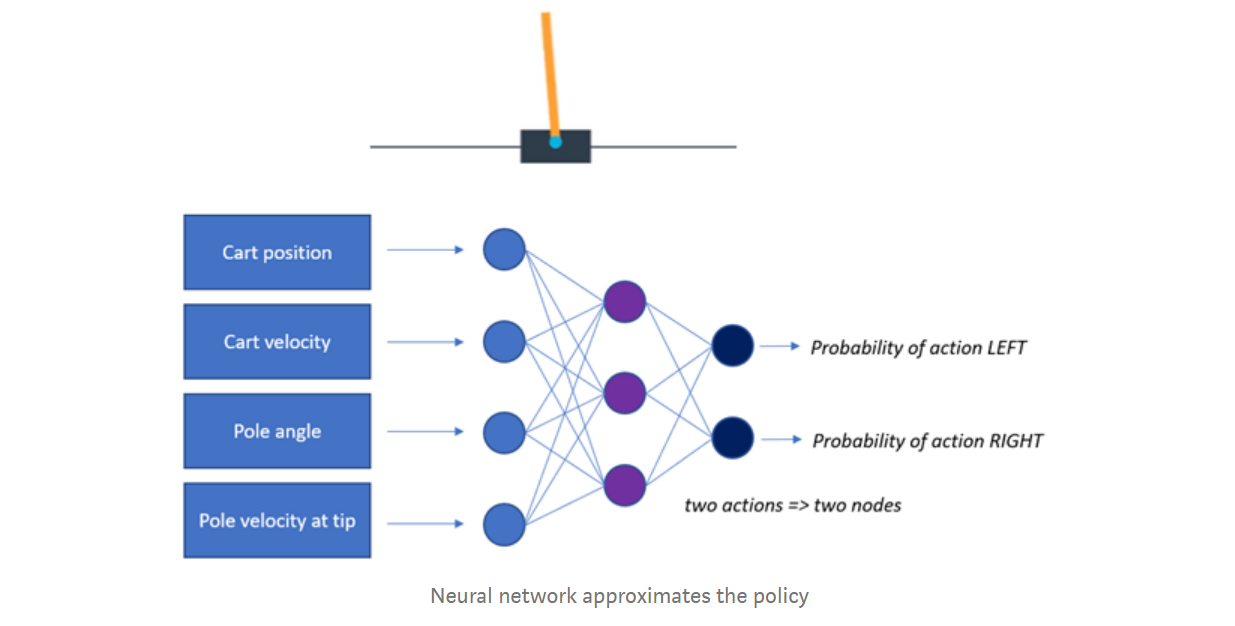
Lý do tiếp theo, khiển các policy có thể thu hút hơn các value đó là do environment có số lượng hành động cực lớn hoặc với space hành động liên tục.

Để có thể quyết định xem đâu là hành động tốt nhất, ta cần phải giải được vấn đề tối ưu hóa việc tìm kiếm action a, đó là làm cực đại hàm Q(s,a). Trong trường hợp game Atari với các hành động rời rạc, mỗi attitude không phải là vấn đề: ta cần xấp xỉ hóa giá trị tại tất cả action và lấy action với Q cao nhất (hay còn gọi là argmax). Nếu mà không gian action không phải là một tập hợp rời rạc, nhưng lại có một giá trị vô hướng được đính vào nó, ví dụ như là góc tay lái (giá trị tức thời được đặt trước), thì vấn đề trở nên khó hơn, với Q thường được biểu diễn bằng một mạng lưới thần kinh phi tuyến tinh, do đó việc tìm kiếm đối số tối đa hóa tất cả các giá trị của function có thể trở nên rất nguy hiểm. Trong từng trường hợp, nó khả thi hơn khi tránh values function và dành nhiều sự tập trung vào các tiếp cận policy.

Câu hỏi liên tiếp hướng chúng ta tiến thẳng. Ta cần phải trả lời được xem kỹ thuật nào giúp chúng ta tiếp cận ý tưởng về việc ước tính các optimal policy .

1. Policy Gradient

Về vấn đề lý do học, chúng ta xét về môi trường CartPole (một trò chơi). Trong trường hợp này, Agent dựa trên phản hồi (feedback) từ môi trường (vị trí của xe hàng, vận tốc xe hàng, góc cực và tốc độ cực) để ra quyết định về các hành động có thể được thực thi. Agent có thể, tại mỗi bước đẩy cái xe hàng qua trái hoặc qua phải. Để làm xấp xỉ (aproximate) policy, tương tự như DQN ta xây dựng mạng thần kinh, nhận đầu vào là các state.

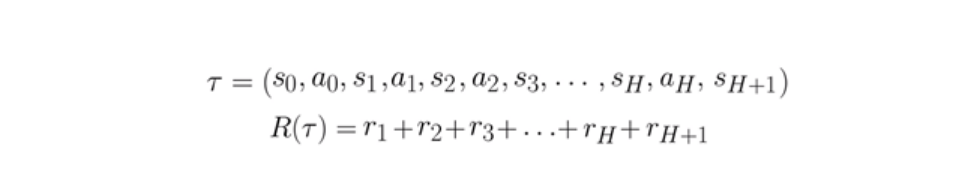


Tại thời điểm này, với đầu ra, thì mạng thần kinh trả về xác suất mà các agent có thể chọn phải. Agent sẽ theo các policy và giao tiếp với environment bằng cách chuyển các state gần nhất vào mạng lưới. Mạng sẽ tạo ra các hành động có thể (phân phối xác suất) và sau đó agent lấy mẫu một vài xác suất để chọn hành động phản hồi lại (trái hoặc phải).

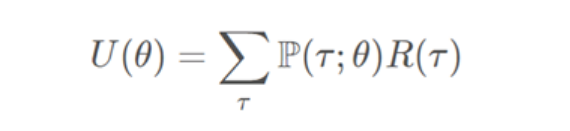
Ta cần phải nhớ là khi làm việc với DQN, đầu ra của mạng sẽ là các Q-values, vậy nên nếu một trong các action bị hội tụ về 0.3 và một cái khác là 0.5, vậy nên “chiến thuật” của chúng ta có action đạt cực đại với giá trị cao hơn được ưu tiên 100%. Thực ra, chúng ta xét về phân phối xác suất, nên nếu action có xác suất là 0.3 và hành động thứ 2 là 0.5, nên agent thực thi action đầu tiên với cơ hội là 30% và với action thứ hai là 50%.

Mục tiêu của chúng ta sau đó là thiết lập các giá trị phù hợp cho các trọng số trong mạng để khi mà mỗi state chúng ta chuyển vào trong mạng thần kinh nhân tạo, nó sẽ trả về xác suất action trong đó action tối ưu được chọn nhiều hơn (có xác suất là cao nhất). Khi Agent tương tác với môi trường và học nhiều hơn xem action nào là tốt nhất để tối ưu hóa reward, nó thay đổi trọng số trong mạng thần kinh. Thay đổi trọng số mạng thần kinh diễn ra theo nhịp độ cập nhật gradient theo cách mà action đem lại reward cao trong một state sẽ có xác suất cao và các action đem lại reward thấp sẽ có xác suất thấp (biến đổi trọng số sẽ tăng xác suất cho action trong đó episode kết thúc với thành công hoặc là giảm xác suất cho action trong đó episode kết thúc với thất bại).

Tiếp cận giải pháp trong việc tìm kiếm optimal policy thì tiếp theo ta cần phải định nghĩa trajectory(τ) (quỹ tích), có thể được biểu diễn dưới dạng một chuỗi độ dài H (theo chiều ngang) các state và action. Với R(τ) ta có thể định nghĩa về reward của trajectory mà ta đang xét.

Mục tiêu ở đây cũng tương tự, ta cần phải tìm trọng số θ của mạng thần kinh mà có thể tối ưu được tổng reward kỳ vọng. –U. Như khi thảo luận, ta điêu chỉnh trọng số.

Có thể biểu diễn biểu thức U ở hàm sau:

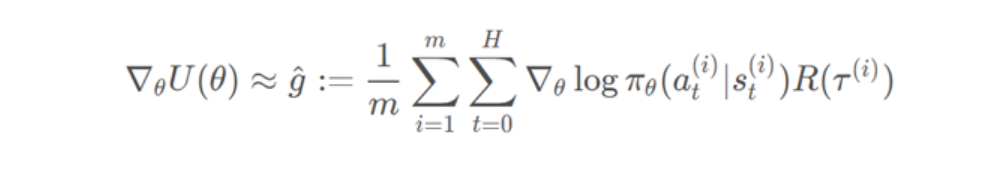


Trong đó thì  R(τ)  là lợi nhuận tương ứng với trajectory τ bất kỳ. Một thành phần khác trong đẳng thức trên P(τ**;** θ) là xác suất của mỗi trajectory bất kỳ.

Ta có thể thấy xác suất dựa trên trọng số θ trong mạng thần kinh định nghĩa policy dùng để chọn action trong trajectory. Như ta thảo luận trước đó, mục tiêu là tìm được giá trị θ mà tối ưu hóa được lợi nhuận kỳ vọng trả về. Để làm như thế, ta có thể tính Gradient Ascent (đối lập với gradient descent mà ta dùng để tính cực tiểu hàm số).

Tại đây, ta cần biết trong tính toán giá trị thực của gradient ascent quy định rằng ta cần phải đánh giá tất cả các trajectory có thể. Có vẻ là quá trình này rất tiêu tốn cho việc tính toán, nên cách tiếp cận của chúng ta ở đây sẽ là lấy mẫu một vài trajectory (m) bằng cách sử dụng policy và sau đó sử dụng m—các trajectory cho việc ước lượng gradient.

Cuối cùng thì gradient được biểu diễn như sau:



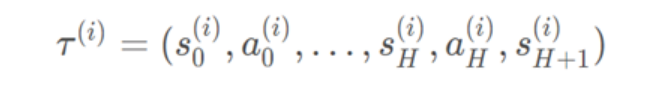
Khi ta ước tính cho gradient, ta có thể sử dụng nó để update trọng số cho policy. Sau đó, ta liên tục lặp lại bước này để hội tụ đến trọng số của optimal policy.

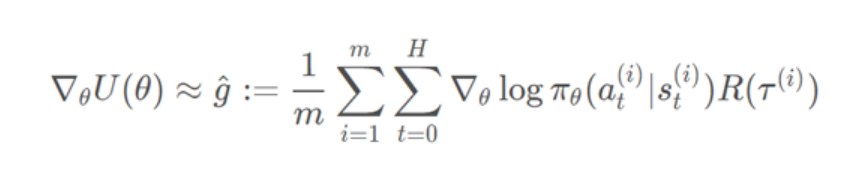
Tại đây thì ta cần hiểu/ nhắc lại một điều. . Đầu ra của mạng thần kinh đơn giản nó chỉ là một số (với 2 nodes trong ví dụ CartPole, thì ta có 2). Tính toán phân phối xác suất liên quan tới nhu cầu của hàm softmax (hàm số lấy đầu vào là một vector K gồm các số thực, và chuẩn hóa nó dưới dạng phân phối xác xuất bao gồm K xác suất).

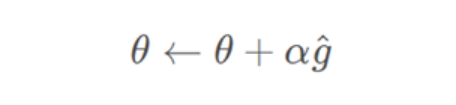
Một điều khác đó là log của xác suất π, được tham chiếu đến policy, được tham số hóa bởi theta. Phép toán này có thể được hiểu là một phép tính cross entropy, được sử dụng để định lượng sự sai khác giữa hai phân phối xác suất. Cuối cùng, ta cần ước lượng được cách ta nên thay đổi trọng số của policy –theta. Nếu ta muốn thay đổi log của xác suất (nhớ là khi ta định nghĩa trước đó. Agent thay đổi trọng số để tăng xác suất các action trong đó episode kết thúc trong thành công hoặc giảm xác suất của action khi mà episode kết thúc trong thất bại).

Pseudocode của thuật toán reinforce có thể tổng kết dưới đây.

1. Sử dụng policy πθ​ thể thu thập M—trajectory với chiều ngang H.



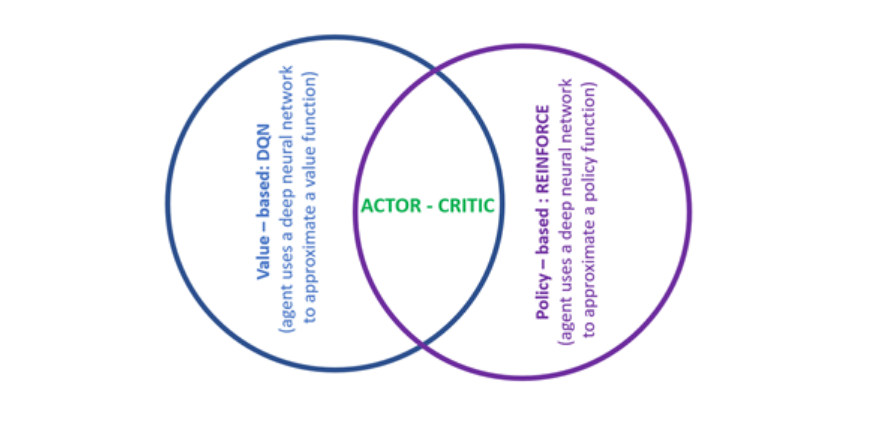
1. Sử dụng trajectory để ước lượng policy gradient.
2. Cập nhật trọng số neural network của policy (a—là một hyper parameter của neural network, step—learning rate):



1. Lặp lại bước 1 tới 3.

**4**.Phương pháp Actor-Critics

Mục tiêu của ta là dùng thuật toán DDPG nhưng vì quá phức tạp ta nên tiếp cận sự hiểu biết về định nghĩa.



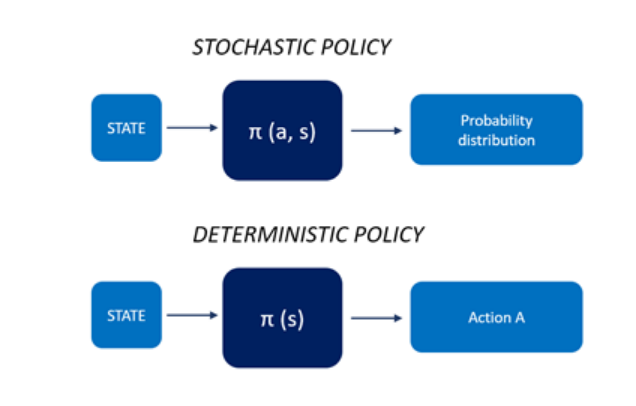
Bức ảnh trên miêu tả định nghĩa chính của Actor-Critic. Các phương pháp Actor-Critics ở đây kết hợp phương pháp values-based ví dụ như DQN và phương pháp policy-based như Reinforce.

Ở đoạn trước ta định nghĩa về DQN Agents (trong project 1) học để xấp xỉ hàm value action tối ưu. Nếu Agents học đủ tốt thì việc đưa ra một policy tốt cho Agents là rất đơn giản.

Ở một khía cạnh khác thì Reinforce Agents tham số hóa policy và học để tối ưu hóa policy đó trực tiếp.

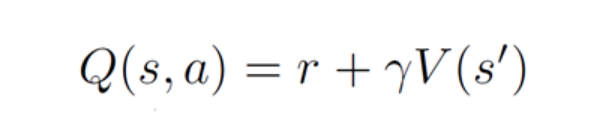
Tại đây, policy thường là ngẫu nhiên, giống như ta nhận các phân phối xác suất.

Ngay bây giờ, ta sẽ tìm hiểu về các **policy xác định (deterministic policies),** các policy này sẽ lấy trạng thái và về từng action (không ngẫu nhiên, các policy này sẽ xác định).



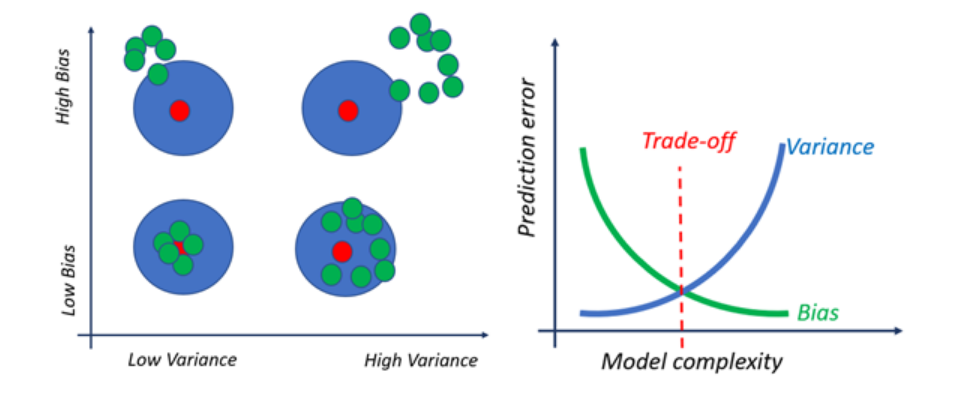
Trong phần trước ta đã trình bày thuật toán Reinforce, thuật toán phải kết thúc các episode trước khi ta có thể bắt đầu training. Đối với môi trường, nơi mà episode có thể kéo dài hàng trăm hoặc hàng ngàn frame. (giống như Atari game). Điều này có thể rất lãng phí cho việc training, nơi ta phải tương tác với môi trường trong kịch bản dài, mà chỉ thực hiện một bước training (để ước tính Q một cách chính xác nhất có thể). Trong trường hợp này thì, training batch sẽ trở nên rất lớn.

Tuy nhiên với DQN, nó có thể khả thi trong việc thay thế chính xác giá trị cho discounted reward với ước lượng sử dụng đẳng thức Bellman

Khi ta xét phương pháp Policy Gradient (như ta thảo luận ở trên), ta xem các giá trị V(s) hoặc là Q(s,a) tồn tại . Trong trường hợp này ta dùng phương pháp Actor-Critics thay thế, trong đó ta dùng mạng thần kinh để ước lượng V(s) và sử dụng sự ước lượng để lấy được giá trị

Q.Gradient được ước lượng trong phương pháp Policy Gradient tỷ lệ thuận với discounted reward từ state đã cho. Tuy nhiên, phạm vi của reward phụ thuộc nhiều vào environment (nó có thể xảy ra khi Agent chơi các short game -- Agent thua trò chơi rât nhanh (value thấp của reward) hoặc Agent thông minh đủ để chơi các game trong thời gian dài, trong lúc thu thập reward). Điểm khác biệt lớn trong việc thu thập reward có thể thực sự ảnh hưởng tới động lực training ,vì một episode may mắn sẽ chiếm ưu thế trong gradient cuối cùng, Trong những tình huống phát sinh như vậy, phương pháp policy gradient có phương sai cao, có thể ảnh hưởng tới quá trình trainning và làm cho nó không ổn định.

Trong bài toán học tăng cường, ta tìm kiếm sự đánh đổi bias – phương sai (xem xét hình dưới), khi mà Agent cố ước lượng hàm value hoặc policy từ return (ta cần nhớ là Agent chỉ lấy mẫu môi trường, vì thế ta cũng chỉ ước lượng các kỳ vọng này). Thông thường, công việc chính trong lĩnh vực này (học tăng cường) là cố gắng giảm phương sai của thuật toán trong khi giữ được bias nhỏ nhất. Công việc này khó mà làm được, nhưng ta sẽ tiếp cận một số kỹ thuật được tạo ra để thực hiện điều này.



1. Thuật toán Deep Deterministic Policy Gradient (DDPG).

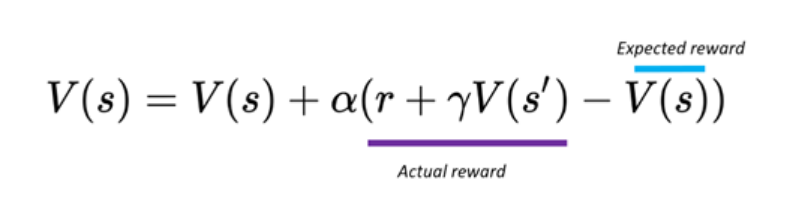
Như khi lúc thảo luận trong khóa Udacity Deep Reinforcement Learning thì có 2 cách để ước lượng return kỳ vọng.

Thứ nhát là ước lượng **Monte-Carlo,** đóng vai trò trong một episode trong việc tính toán tổng số reward từ một chuỗi reward.

Trong Quy hoạch động (Dynamic Programming), **Markow Decision Process (MDP)** được giải bằng việc lặp giá trị và lặp policy. Cả hai kỹ thuật đều cần xác suất chuyển tiếp và xác suất reward để tìm ra optimal policy. Khi xác suất chuyển tiếp và xác suất reward không biết, ta sử dụng Monte-Carlo để giải MDP. Phương pháp Monte-Carlo chỉ sử dụng trên một chuỗi mẫu gồm state, action và reward. Monte-Carlo chỉ dùng cho các công việc episode.

Ta có thể tiếp cận ước lượng Monte-Carlo bằng cách xét Agent trong episode A. Ta bắt đầu với trạng thái St và lấy action At. Dựa trên quá trình Agent chuyển sang state St+1. Từ environment, Agent gửi reward Rt+1. Quá trình này tiếp diễn cho tới khi Agent tiếp cận episode cuối cùng. Agent có thể tham gia tương tự vào các episode khác như B, C hoặc D. Một vài các episode đó có trajectory đi qua cùng một state, điều này ảnh hưởng tới value function được tính như trung bình ước lượng. Ước tính cho một trạng thái có thể khác nhau giữa các episode, do đó, ước tính Monte-Carlo sẽ có phương sai cao.

Mặt khác, ta có thể dùng ước lượng **Temporal Difference.** Ở đây, TD xấp xỉ ước lượng hiện tại dựa trên ước lượng học trước đó, và điều này cũng được gọi là bootstrapping (ta cố để dự đoán state value).



Đẳng thức ở trên là sự sai khác (TD error) giữ giá trị reward thật và reward kỳ vọng nhân với learning rate alpha (learning rate, còn gọi là step size, sử dụng cho lý do hội tụ).

Ước lượng TD có phương sai thấp vì bạn chỉ trộn mỗi step bất ngẫu nhiên thay vì khai triển đầy đủ như Monte-Carlo. However, vì sử dụng boostrapping (quy hoạch động) state tiếp theo chỉ được ước lượng. Giá trị được ước lượng đưa ra bias vào trong tính toán. Agent sẽ học nhanh hơn, nhưng hội tụ có thể xảy ra.

Bắt đầu khái niệm Actor-Critic cần phải xem xét cách tiếp cận policy-based trước (thực thi bởi Agent). Như ta thảo luận trước đó Agent chơi game tăng xác suất các action dẫn tới tới chiến thắng, và giảm xác xuất action dẫn đến thất bại. Tuy nhiên, quá trình này khá nặng do có nhiều dữ liệu để tiếp cận optimal policy.

Mặt khác, ta có thể đánh giá tiếp cận value-based (thực thi bởi Critic), trong đó các dự đoán được thực thi nhanh chóng, qua tất cả các episode. Lúc đầu, các dự đoán sẽ bị sai lệch (không chính xác). Nhưng theo thời gian, khi mà ta đạt được nhiều kinh nghiệm hơn, ta cũng có thể tạo ra được dự đoán đúng. Mặc dù đây không phải là một cách tiếp cận hoàn hảo, dự đoán đưa ra bias vì đôi khi chúng có thể sai, đặc biệt là do thiếu kinh nghiệm.

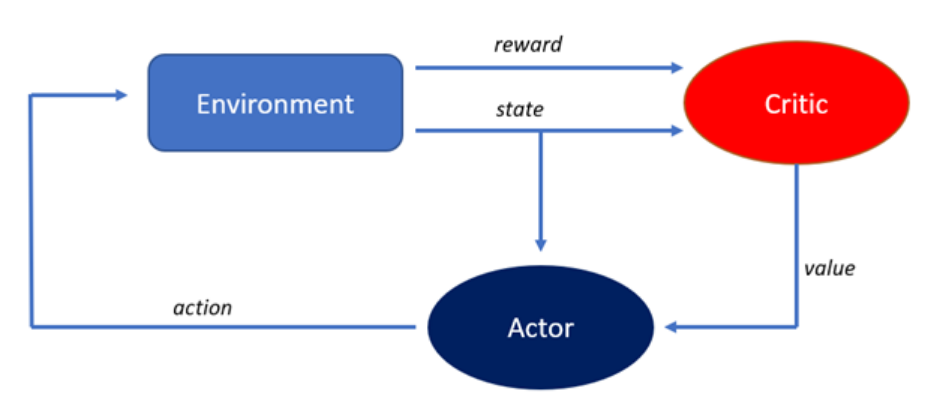
Dựa trên phân tích ngắn này ta có thể tổng kết rằng Agent dùng cách tiếp cận policy-base là học để hành động (agent học bằng cách giao tiếp với environment và điều chỉnh xác suất hành động tốt hoặc xấu, trong khi đó tiếp cận values-based, agent học để ước lượng state và actions). Ta dùng Critics song song để có thể đánh giá chất lượng hành động nhanh hơn (action đúng hay không) và tăng tốc độ học. Phương pháp Actor-Critic ổn định hơn value-based, trong khi yêu cầu ít mẫu trainning policy-based agent.

Kết quả của việc kết hợp Actor-Critic, ta sử dụng hai mạng thần kinh riêng biệt. Vai trò của Actor là xác định actions tốt nhất (từ phân phối xác suất) trong state bằng cách điều chỉnh tham số θ (trọng số). Critic bằng cách tính toán temporal difference error TD (ước lượng return kỳ vọng), đánh giá action được tạo ra bằng Actor.

Trong project trước (DQN) ta thảo luận về environment rời rạc trong đó số lượng action có thể được thực thi bởi Agent là giới hạn. Tuy nhiên, ta thường hoạt động với môi trường tiếp diễn, đảm bảo chuyển động liên tục. Số lượng actions có thể là không giới hạn (lớn). Một vấn đê DDPG giải quyết. Thuật toán DDPG sử dụng khái niệm Actor-Critic, trong đó chúng ta sử dụng hai mạng thần kinh sâu.

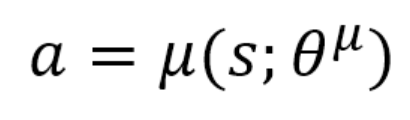
Trong DDPG, Actor được sử dụng để xấp xỉ hóa optimal policy xác định. Điều này có nghĩa là ta thường xuyên muốn tạo ra action đáng tin cậy tốt nhất cho bất kỳ trạng thái nào.

Actor sẽ theo cách tiếp cận policy-based, và học cách làm thế nào để hành động bằng cách ước lượng trực tiếp optimal policy và tối ưu hóa reward suốt gradient ascent. Tuy nhiên Critic sử dụng cách tiếp cận values-based và học cách làm thế nào để ước lượng sự sai khác của cặp giá trị state-action.

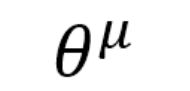


Thuật toán DDPG có thể biểu diễn theo cách sau (Reinforcement Learning with Python by Sudharsan Ravichandiran):

1. Actor và Critic sử dụng hai mạng thần kinh riêng biệt.
2. Mạng của Actor với :

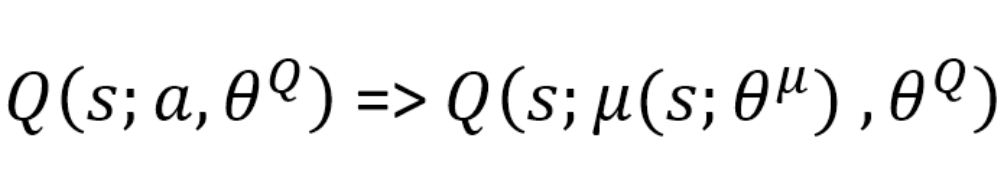


Lấy đầu vào là **state s** và kết quả trong action a tại đó

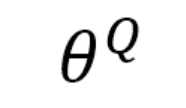


Là trọng số của mạng Actor. Ở đây Actor được sử dụng để xấp xỉ hóa optimal policy xác định. Có nghĩa là đầu ra là các actions có thể tin cậy được tốt nhất cho bất kỳ trạng thái nào. Điều này không giống với stochastic policy (phân phối xác xuất) mà ta muốn policy tìm hiểu phân phối xác suất qua các action. Trong DDPG, ta muốn hành động được tin cậy tốt nhất mỗi thời điểm ta truy vấn mạng Actor. Về cơ bản Actor chỉ học argmax a Q(S,a), là action tốt nhất.

1. Mạng Critic

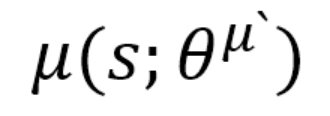


Lấy đầu vào là state s và action a và trả về giá trị Q trong đó mạng Critic

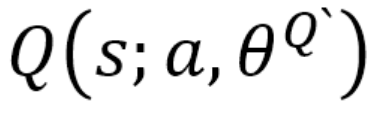


Là trọng số. Critics học để đánh giá hàm giá trị action tối ưu bằng cách dùng action đáng tin cậy tốt nhất của Actors.

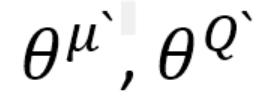
1. Ta định nghĩa mạng mục tiêu cho cả mạng Actor



và mạng Critics tương ứng,

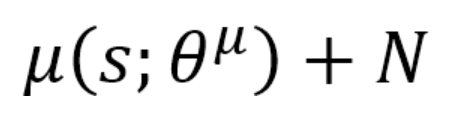


Trong đó



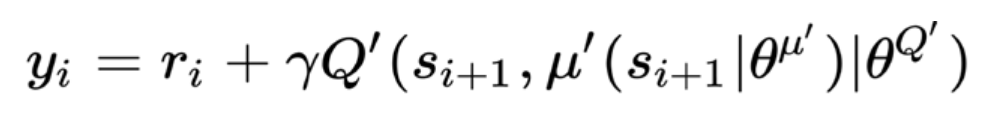
là trọng số của mạng Actor và Critic.

1. Tiếp theo ta thực thi việc update trọng số của mạng Actor với policy gradient và trọng số của mạng Critics với gradients được tính toán bằng TD error
2. Để chọn action chính xác, trước tiên ta phải thêm các nhiễu exploration (định nghĩa exploration và exploitation trong RL) N vào các action được tạo ra bới Actor:

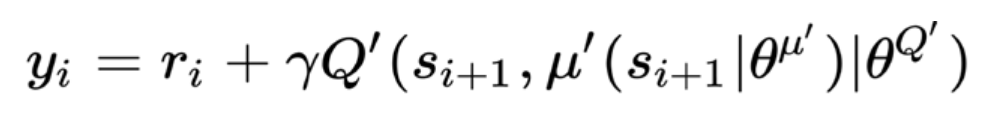


(thêm nhiễu nhằm tăng cường expoloration vì policy là xác định)

1. Chọn acton từ trạng thái s, nhận reward, r và chuyển sang reward mới, s’
2. Ta lưu thông tin chuyển tiếp này vào trong replay buffer experience
3. Giống như thực hiện khi ta sử dụng thuật toán DQN, ta lấy mẫu chuyển tiếp từ replay buffer và train mạng, sau đó ta tính toán Q value mục tiêu:



10.Sau đó, ta tính toán hàm TD error với:

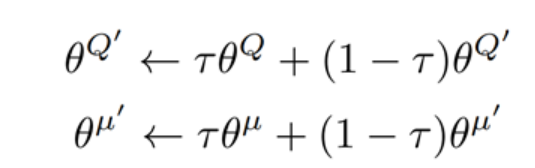


11 Sau đó, ta thực hiện việc update các trọng số của mạng Critic với Gradient được tính từ Loss L

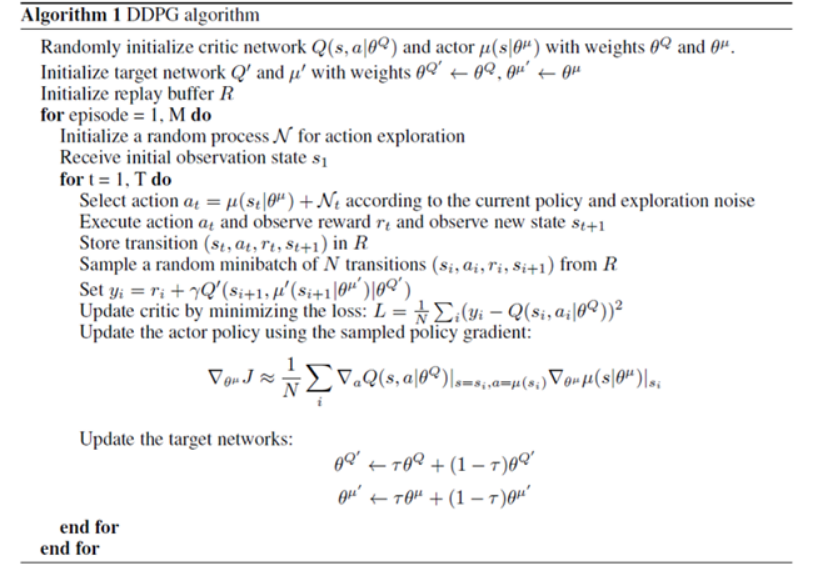
12. Tiếp, ta update trọng số policy network sử dụng policy gradient.

13. Sau đó, ta update trọng số của mạng Actor và Critic trong mạng đích. Trong cấu trúc liên kết thuật toán DDPG bao gồm hai bản sao trọng số mạng của từng mạng, (Actor: thông thường và mục tiêu) và (Critics: thông thường và mục tiêu). In DDPG, mạng mục tiêu được update sử dụng chiến thuật soft update. Chiến thuật soft update bao gồm việc trộn trọng số mạng thông thường với trọng số mạng mục tiêu. Có nghĩa là mỗi bước ta làm cho mạng mục tiêu của mình là 99.99% trọng số mạng mục tiêu và chỉ 0.01% trọng số mạng thông thường (trộn chậm trọng số mạng thông thường vào trong trọng số mạng mục tiêu).

14. Ta update trọng số của mạng mục tiêu (Agent, Critic) chậm rãi, nhằm thúc đẩy mức ổn định mạnh hơn (chiến thuật soft update)



Thuật toán DDPG có thể biểu diễn dưới dạng giả mã như sau:



Mô tả chung của thuật toán nằm tại biểu đồ phía dưới:

