### 03/15/17

### Article from: http://www.phrgcm.com/blog/2016/08/17/deep-q-network/

### **Deep Q Network**

구글 딥마인드 팀에서 Atari게임을 Reinforcement Learning으로 학습시키면서 [Playing Atari with Deep Reinforcement Learning](https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf)과 [Human-level control through deep reinforcement learning](http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html)이라는 두가지 논문을 냈다. 바로 DQN(Deep-Q-Network)이라는 방법이다. DQN은 간단히 말하면 Reinforcement Learning에 Deep Learning을 이용하는 방법이다. Reinforcement Learning에서 value function을 state마다 모두 표현하기 힘들 때에는 approximation function을 이용하기도 한다. 실제로 Backgammon이란 게임에서 Neural Network를 이용해 value function을 approximation한 예가 있다.([TD-gammon](http://cling.csd.uwo.ca/cs346a/extra/tdgammon.pdf) 위의 딥마인드 논문들에서는 단순한 Neural Network가 아니라 Convolutional Neural Network를 이용해 value function을 approximation하고 Atari 게임들을 학습시킨다.

### **Q Learning**

먼저 알아야 하는 부분은 Q-Learning이다. 사실 Reinforcement Learning에 대해 접하면서 다른 것 보다 먼저 접하게 된 것이 Q-Learning이다. 그만큼 Reinforcement Learning에서 유명한 방법이다. 각각의 state마다 Q table을 만들어서 [s,a,r,s’]이 주어지면 다음과 같은 식으로 Q value를 업데이트 하게 된다.

Q(s,a)←Q(s,a)+α(R+γmaxa′Q(s′,a′)−Q(s,a))

이런 방식으로 여러번의 에피소드를 돌리면 Q value가 점점 optimal Q value(Q\*)에 수렴하게 된다. 이때 각각의 state에 대한 Q value를 직접 저장하지 않고 approximation function을 이용할 수 있다. 그리고 그 approximation을 Deep Learning을 이용해서 구하자는게 DQN의 요지다.

### Experience Replay and Target Q

위의 논문들에서 단순히 CNN을 이용해 value approximation을 이용한다는 것 외에도 크게 두가지 트릭을 이용했는데 바로 Experience Replay와 Target Q라는 방법이다. 원래 Neural Network를 통한 value approximation이 상당히 unstable한 방법인데(TD-gammon은 아주 특별한 케이스라고 한다) 그것을 stable하게 만드는 트릭이다.

Experience Replay라는 방법은 모든 state와 action, reward가 사실은 이전의 state들에 상당한 영향을 받는다. 즉, state, action, reward들이 correlation이 있는 것이다. 이런 correlation이 정확히 어떤 영향을 주는지는 모르지만 approximation function이 stable하게 학습되지 못하는 이유 중 하나라고 한다. 그래서 Experience를 버퍼에 저장을 하고는 그 중에서 학습데이터를 random하게 뽑아서 correlation을 없애준 데이터들로 학습을 하게 된다.

최근에는 [Asynchronous Method](https://arxiv.org/pdf/1602.01783v2.pdf)라는 방법으로 correlation을 없애줘서 Experience Replay를 대체한다고 한다. 간단히 Asynchronous Method를 설명하면 Thread를 통해서 여러개의 agent가 동시에 [state, action, reward, state’]를 수집한다. 그렇게 여러 agent가 동시에 수집한 데이터들은 서로 correlation이 없을 것이기 때문에 Experience Replay를 대체할 수 있으면서 더 빠르고 메모리도 절약할 수 있는 방법이라고 한다.

Target Q라는 방법은 target network와 Q network를 따로 두고서는 target을 구할 때는 target network로 구하고 구한 target value로 Q network를 학습시키는 방법이다. 그러고서는 일정한 timestep마다 target network를 Q network로 업데이트를 한다. 이것은 action value와 target의 correlation을 줄여준다고 한다. 이 correlation을 줄이는게 왜 중요한지는 정확히 모르겠으나 실제로 stable하게 학습하는데 상당한 기여를 하는것 같다.

마지막으로 Network를 학습시키는 부분의 식을 보면 아래와 같다. 아래 loss function을 이용해서 Network를 학습시킬 수 있다.

Li(θi)=E(s,a,r,s′)∼U(D)[(r+γmaxa′Q(s′,a′;θi−)−Q(s,a;θi))2]

여기서 r+γmaxa′Q(s′,a′;θi−)가 target value이고 Q(s,a;θi)가 Q network의 action value(Q value)이다. 보면 알겠지만 target value는 θi−를 이용하고 action value에는 θi를 이용한다. 서로 다른 네트워크의 weight를 표현하기 위해서 서로 다른 표현을 이용한다.

### **결론**

일단은 간단하게 DQN에대한 부분을 알아보았다. 다음번에는 Tensorflow를 이용한 코드를 살펴보면서 조금 더 자세하게 DQN을 살펴볼 예정이다.

### **참조**

[Playing Atari with Deep Reinforcement Learning](https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf)  
[Human-level control through deep reinforcement learning](http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html)  
[Asynchronous Method for Deep Reinforcement Learning](https://arxiv.org/pdf/1602.01783v2.pdf)

Article from: http://sanghyukchun.github.io/76/

[첫 글](http://sanghyukchun.github.io/57)에서 Machine Learning은 크게 세 가지로 구분된다는 얘기를 했었지만, 지금까지 다뤘던 주제들은 모두 supervised learning이거나 unsupervised learning이었다. Reinforcement learning은 그 둘과는 구분되는 명백히 다른 task이지만, machine learning에서 그만큼 대중적인 분야는 아니다. 아직까지 reinforcement learning을 사용한 적절한 application이 많이 제안된 것도 아니라서 practical하게 많이 사용지도 않는다. 그러나 reinforcement learning을 사용하면 supervised, unsupervised learning와는 전혀 다른 방법으로 문제를 접근하는 것이 가능하다. 최근 deep learning과 reinforcement learning을 결합한 재미있는 연구주제들도 나오고 있는 만큼 ([Atari 리뷰](http://sanghyukchun.github.io/90), [Visual Attention 리뷰](http://sanghyukchun.github.io/91)), 앞으로 더 재미있는 방향으로 연구될 수 있는 주제가 아닐까 생각한다.

이 글은 Andrew Ng. 교수가 Stanford에서 강의하는 CS229 Machine Learning 수업의 [lecture note](http://cs229.stanford.edu/notes/cs229-notes12.pdf)를 바탕으로 쓰여졌다. 이 글이 조금 부족하다고 느끼는 경우에는 해당 reference를 읽어보면 큰 도움이 될 것으로 생각된다.

### **Reinforcement Learning: Problem Definition**

Supervised learning은 주어진 데이터의 label을 mapping하는 function을 찾는 문제이다. 이 경우 알고리즘은 얼마나 label을 정확하게 분류하느냐 혹은 정해진 loss function을 minimize시킬 수 있느냐에만 초점을 맞추어 모델을 learning하게 된다. 분명 supervised learning은 상당히 많은 application들에 응용될 수 있는 방법이다. 하지만 모든 문제들이 이런 방식으로 해결할 수 있는 것은 아니다. 예를 들어 2족 보행을 하는 알고리즘을 디자인한다고 생각해보자. 우리가 알고 싶은 것은 어떻게 로봇의 관절들을 움직여야 로봇이 넘어지지 않고 잘 걸을 수 있을까이다. 이 경우 우리는 관절의 움직임을 control하는 function을 learning해야한다. 이 문제를 머신러닝으로 풀기 위해서는 어떻게 문제를 정의해야할까? [첫 글](http://sanghyukchun.github.io/57)에서 머신러닝 문제는 (1) 데이터 (2) output (3) target function (4) loss를 minimize하는 algorithm이 필요하다고 언급했었다. 먼저 데이터는 현재 관절들의 상태(각도, 위치 등)와 주변 환경(흙 위인지 아스팔트 위인지 앞에 벽이 있는지 등등)을 데이터라고 정의하자. 우리는 지금 보행을 learning하는 알고리즘을 찾고 있으므로 원하는 output은 지금 상태 다음의 관절 상태가 될 것이다. 즉, [f: 현재 관절 상태, 환경 -> 다음 관절 상태]라는 target function까지 정의할 수 있다. Loss는 특정 시간 이후 얼마나 많이 걸었는지 등으로 판단할 수 있을 것이다.

그렇다면 이 문제는 supervised learning이나 unsupervised learning으로 풀 수 있을까? 데이터만 무한하게 있다면 가능할지도 모르지만 현실적으로는 그럴 수 없다는 것을 알 수 있다. 왜냐하면 (data, output)의 조합이 너무 많기 때문에 의미있는 learning을 할 수 있을 정도로 많은 데이터를 모을 수 없기 때문이다. 즉, 어떤 action이 ‘correct’ action인가 판단하는 것이 불가능하다. 대신에 이렇게 생각해보면 어떨까? 우리가 알고 싶은 것은 관절 상황과 환경이 주어졌을 때 로봇이 어떻게 action을 취해야하는가라는 policy이다. 매 action을 주어진 policy를 통해 시행하고 나면 다음 state가 정의된다. 만약 성공적으로 걸었다면 +1 점을 주고 넘어졌다면 -1점을 주는 방식을 통해 매 action의 reward를 정의할 수 있을 것이다. 그렇다면 static한 데이터 셋에서 거의 무한하게 많이 필요한 (data, output)를 사용해 learning하는 방법 대신에, 직접 매 순간 action을 실행해 reward를 받으면서 최종적으로 맨 마지막에 모든 reward의 합이 가장 좋게 만드는 policy를 learning하는 것이다.

이런 식으로 reinforcement learning을 high level로 설명할 수 있다. 그렇다면 RL을 조금 더 formal하게 정의해보자.

### **Markov Decision Process (MDP): Problem definition**

앞에서 설명한 방식대로 RL을 정의하면 RL problem은 정말 여러가지 형태로 정의할 수 있지만, 보통 RL문제를 푼다고 하면 Markov Decision Process (MDP)를 의미한다. MDP는 (S,A,{Psa},γ,R) 이라는 것들의 튜플이다. 각각에 대해 알아보도록 하자.

* S - state들의 set을 의미한다. 앞에서 예를 든 2족 보행 로봇의 경우 모든 가능한 관절의 상태와 환경 등이 state가 된다. 참고로 state와 다음에 기술한 action의 개수가 유한하다면 |S|<inf,|A|<inf, 주어진 MDP를 finite MDP라고 부른다.
* A - action들의 set을 의미한다. 2족 보행 로봇의 경우 어떻게 관절을 control할 것인가를 의미한다.
* Psa:(st,at)→sat - State의 transition probability로, 특정 state에서 특정 action을 취했을 떄 다음 state는 어떤 state가 될지에 대한 확률 값이다.
* R:S×A→R - 주어진 state에 action을 수행했을 때 얻게 되는 reward를 function으로 표현한 것이다. Reinforcement learning의 목표는 시간이 T만큼 흘렀을 때 최종적으로 얻게 되는 모든 reward들의 합을 (정확하게는 그것의 expectation을) maximization하는 policy를 learning하는 것이다.
* γ∈[0,1) - 앞에서 설명한 reward의 discount factor로, 시간이 지날수록 reward의 가치를 떨어뜨리고, 처음 받은 reward의 가치를 더 키워주는 역할을 한다. 즉, time t에서 얻은 reward를 rt라고 했을 때, 전체 reward Rtot는 ∑Tt=0γtrt가 된다.

MDP의 dynamics는 다음과 같은 식이다. 먼저 initial state s0에서 어떤 초기 action a0를 수행하게 된다. 이 행동으로 인하여 주어진 probability Ps0a0에 따라 다음 state s1이 확률적으로 결정된다. 그리고 그 결과로 reward R(s0,a0)를 얻게 된다. 이를 다시 s1에 대해 반복하면서 state가 terminal state에 도달할 때 까지 이 과정을 반복하게 된다. 이때, MDP의 Markov property 때문에 다음 step의 reward와 transition probability는 오직 지금 state와 지금 action에 의해서만 결정된다.

s0−→a0s1−→a1s2−→a2….

이때, 앞에서 설명한 바와 같이 매 action을 취할 때 마다 reward가 결정된다. 이때 최대한 빠르게 좋은 reward를 받을수록 좋기 때문에 나중에 얻은 reward보다 일찍 얻은 reward의 값이 같더라도, 일찍 얻은 reward가 더 valuable하다고 가정한다. 이것을 우리는 discount factor를 통해 조절하게 되며, 그 결과 우리가 maximization하고 싶은 최종 reward는 discount factor γ에 의해 다음과 같이 결정된다. (참고로 이 값을 sum of discounted reward라고 부른다.)

R(s0,a0)+γR(s1,a1)+γ2R(s2,a2)+….

하지만 아무리 같은 state와 action으로 시작했다고 하더라도 이 과정은 전부 Psa에 의해 확률적으로 결정되는 값이기 때문에, 실제로 maximization하기 위한 target은 그 값의 expectation으로 주어진다.

E[R(s0,a0)+γR(s1,a1)+γ2R(s2,a2)+…].

우리가 위 expectation을 maximization하기 위해 learning해야하는 것이 바로 policy π:S→A 이다. Policy는 state에서 action으로 mapping되는 function이다 (즉, at=π(st). 앞에서 설명했던 transition probability는 현재 state와 action을 다음 state와 mapping해주는 function이었고, policy는 지금 state에서 내가 어떤 action을 취해야하는지 mapping해주는 function인 것이다. 즉, policy가 어떻게 변화하느냐에 따라 최종 reward가 크게 바뀌게 된다.

### **Value function and Bellman Equation**

그럼 어떻게 reward를 maximize하는 policy를 learning할 수 있을까? 그것을 설명하기에 앞서, 먼저 policy π의 value function Vπ(s)이라는 것을 정의해보자. 이때 앞으로 reward R(s,a)는 state에 의해서만 결정된다고 가정하고, notation을 R(s)로 바꾸도록 하겠다. 만약 action과 state 둘 다에 의해 reward가 결정되는 경우는 앞으로 설명하게 될 value function V가 아니라 [action-value function Q](http://sanghyukchun.github.io/76/76-qfunction)라는 것을 정의하고 그것에 대한 Bellman Equation을 구해 아래와 같은 방식을 그대로 적용하는 것이 가능하다.

olicy π의 value function Vπ(s)은 다음과 같이 정의된다.

Vπ(s)=E[R(s0)+γR(s1)+γ2R(s2)+…|s0=s,π].

이 값은 즉, 간단하게 이야기 하여 주어진 state s를 initial state로 두고 action을 policy π를 사용하여 고르게 되었을 때 우리가 얻게되는 reward의 expectation 값이 된다. 이렇게 정의했을 경우 fixed policy π에 대해 value function은 다음과 같은 관계식을 가진다. 증명은 크게 어렵지 않으므로 생략하겠다.

Vπ(s)=R(s)+γ∑s′∈SPsπ(s)(s′)Vπ(s′).

이 관계식을 Bellman Equation이라고 부르며, 이 관계식을 통해 우리는 Vπ(s)과 다음과 같은 두 가지 성분으로 표현된다는 것을 알 수 있다. 첫째로 immediate reward R(s,π(s))이다. 이 값은 우리가 처음 state s에서 바로 얻을 수 있는 reward를 의미한다. 다음으로 future reward의 expectation이다. 이 값에 discount factor를 곱하고 immediate reward를 더하게 되면 우리가 원하는 Vπ(s)를 구하는 것이 가능하다. 이때, future reward term은 사실 Es′∼Psπ(s)[Vπ(s′)] 으로 표현할 수 있는데, 다시 말해 future reward term은 처음 state s에서 policy π로 정해진 다음 state s′의 distribution에 대한 sum of discounted reward의 expectation 값과 같다는 것을 알 수 있다. 그러므로 두 번째 term은 MDP의 한 step이 지나고 난 이후에 발생하는 모든 sum of discounted reward들의 expectation이라는 것을 알 수 있는 것이다.

Bellman Equation을 사용하면 finite MDP에 대해 value function Vπ(s)를 효율적으로 계산할 수 있다. 만약 finite MDP에 대해 문제를 풀고 있다고 가정해보자. 그렇다면 주어진 모든 state s에 대해 Vπ(s)의 Bellman Equation을 적는 것이 가능한데, 이렇게 되면 우리는 |S|개의 linear equation과 |S|개의 variable들 (이 경우는 각 state에 대한 Vπ(s)들)이 존재하기 때문에 간단한 연립방적식으로 value function의 값을 찾는 것이 가능하다.

하지만 우리가 처음부터 원했던 것은 optimal policy π∗이지, 주어진 π에 대한 expectation of sum of discounted reward가 아니다. 하지만 이 optimal policy 역시 Bellman Equation을 사용하면 계산할 수 있다. 이것을 어떻게 하는지 설명하기 전에 앞서, 먼저 optimal value function V∗(s)를 다음과 같이 정의해보자.

V∗(s)=maxπVπ(s).

즉, optimal value function은 모든 policy π에 대한 value function Vπ(s) 중에서 가장 reward를 maximize시키는 policy를 통해 얻게 된 reward가 된다. Optimal value function 역시 Bellman Equation을 증명할 수 있는데, 그 식은 다음과 같다.

V∗(s)=R(s)+maxa∈Aγ∑s′∈SPsa(s′)V∗(s′).

앞의 term은 위와 동일하고, 두 번째 term은 모든 expected future sum of discounted reward를 action a에 대해 maximize한 결과이다. 즉, 모든 action 중에서 reward를 가장 maximize하는 action을 선택하였을 때 얻게되는 reward의 값이다. 그런데 그런 action을 고르는 방법이 바로 optimal policy π∗이므로, optimal policy는 다음과 같이 구할 수 있다.

π∗(s)=argmaxa∈Aγ∑s′∈SPsa(s′)V∗(s′).

모든 state s와 모든 policy π에 대해 우리는 다음과 같은 관계식을 얻을 수 있다.

V∗(s)=Vπ∗(s)≥Vπ(s).

첫번째 관계식은 optimal policy π에 대한 value function Vπ∗(s)와 optimal value function V∗(s)가 모든 state s에 대해 equivalent하다는 것을 보여준다. 이 내용이 중요한 이유는, 초기 state가 무엇인지와 관계없이 항상 같은 optimal policy π∗를 사용해 optimal value function을 구할 수 있다는 의미가 되기 때문이다. 즉, 만약 optimal policy를 구할 수 있는 algorithm이 있다면 그 알고리즘의 initial state로 어느 state를 고르더라도 우리는 항상 같은 policy를 얻게된다는 사실을 암시한다. 그리고 두 번째 equation은 모든 policy π에 대한 value function보다 Vπ∗(s)의 값이 더 크거나 같다는 것을 의미한다. 만약 optimal policy를 구하는 algorithm이 value function을 monotonically increase 시키는 방향으로 learning한다고 했을때, update되는 값의 upper bound가 존재하므로 항상 converge하게 된다는 것을 알 수 있다.

Finite MDP의 optimal policy를 구하는 대표적인 두 알고리즘으로는 value iteration과 policy iteration이라는 알고리즘이 존재한다. 두 알고리즘은 이름에서 알 수 있듯 모두 iterative algorithm이며, 위에서 언급한 intuition이 그대로 적용되는 알고리즘들이다. 즉, initial state에 invariant하며 iteration 동안 value function이 monotonically increase한다. 그리고 그 값이 converge하게 되면 우리가 원하는 optimal policy를 구할 수 있다.

### **Value Iteration**

Value iteration 알고리즘은 다음과 같다.

1. Initialize V(s)=0,  for all s.
2. Repeat until converge

V(s)=R(s)+maxa∈Aγ∑a′Psa(s′)V(s′),  for all s.

이 알고리즘은 앞서 설명했던 Bellman Equation에서 optimal value function의 관계식을 iterative하게 반복하면서 찾아나가는 알고리즘이다. 이 알고리즘의 두 번째 state는 synchronous update와 asynchronous update 총 두 가지 방법으로 접근이 가능하다. 먼저 synchronous update는 모든 s에 대해 value function V(s) 값 을 계산하고 그 값들을 한 번에 update하는 방법이고, asynchronous update는 한 state s에 대해 V(s)를 구하고 바로 update를 하는 방법이다. 쉽게 생각하면 asynchronous update는 stochastic gradient descent같은 방법이라 생각하면 된다. 이 두 가지 방법 모두 finite하고 polynomial time 안에 V가 optimal value function V∗으로 수렴한다는 것을 증명할 수 있다. 이렇게 구해진 V∗를 사용하면 앞에서 구했던 다음 관계식을 통해 optimal policy를 구할 수 있다.

π∗(s)=argmaxa∈Aγ∑s′∈SPsa(s′)V∗(s′).

### **Policy Iteration**

이번에는 policy iteration 알고리즘에 대해 살펴보자. 알고리즘은 다음과 같다.

1. Initialize π randomly
2. Repeat until converge
   * Let V=Vπ.
   * For each state s, let π(s)=argmaxa∈Aγ∑s′∈SPsa(s′)V∗(s′).

Value iteration이 value function의 값을 update하는 알고리즘이라면 policy iteration은 policy를 udpate하는 iterative algorithm이다. 두 알고리즘 모두 Bellman equation을 통해 얻어지는 알고리즘이다. Policy iteration에서 π(s)를 업데이트하는 방식을 주어진 value function V에 대해 greedy한 policy update rule이라고 부른다. Policy iteration 역시 polynomial time 안에 optimal policy로 수렴하게 된다.

Value iteration과 policy iteration은 모두 MDP를 해결하는 알고리즘이며, 둘 중 무엇이 더 좋냐를 비교할 수는 없다. 그러나 일반적으로 small MDP에 대해서는 policy iteration이 빠른 시간 안에 효과적으로 수렴하고, large MDP의 경우에는 policy iteration에서 greedy policy rule update가 비효율적일 수 있기 때문에 value iteration으로 문제를 푸는 것이 computationally 좀 더 efficient하다고 한다.

다만 value iteration과 policy iteration은 이론적으로 optimal value function을 계산할 수 있도록 보장하는 알고리즘이기는 하지만, 실제 세상의 large MDP에 적용하기에는 모든 state와 action에 대한 경우 수를 계산하는 이런 알고리즘들은 다소 비효율적이다. 대신 다른 방법으로 value function을 update할 수 있는 알고리즘을 제안하기도 하는데, 예를 들면 지난 번에 리뷰했던 [Playing Atari With Deep Reinforcement Learning (NIPS 2013)](http://sanghyukchun.github.io/90) 논문을 예로 들 수 있을 것 같다.

### **Action Value Function**

Reward가 state, action 둘 다에 의해 결정될 경우, value function Vπ(s)가 아니라 action value function Qπ(s,a)를 사용하여야한다. Q function은 다음과 같이 정의할 수 있다.

Qπ(s,a)=E[R(s0,a0)+γR(s1,a1)+γ2R(s2,a2)+…|s0=s,a0=a,π].

이 값을 사용하게 되면 initial state와 action에 대해 앞에서 value function에 대해 계산했던 것들을 그대로 반복할 수 있다. 먼저 Q∗(s,a)=maxπQπ(s,a)이고, optimal action value function의 Bellman Equation은 다음과 같이 주어진다.

Q∗(s,a)=R(s,a)+γ∑s′∈SPsa(s′)maxa∈AQ∗(s′,a).

남은 부분은 value function으로 진행했던 내용과 동일하게 진행하면 된다.

### **Learning Model for MDP**

지금까지 앞에서 살펴봤던 내용은 모두 MDP의 state transition probability와 reward function이 전부 주어진 상태라고 가정하고 문제를 푸는 방법이었다. 하지만 실제로는 transition probability와 reward가 직접적으로 알려져있지않고, 실제 action을 수행하기 전까지 알 수 없는 것들이 훨씬 많다. 이 경우 data를 통해 transition probability와 reward function을 estimate해야한다. 이 경우 각각의 state에 대해 모든 action을 반복적으로 수행하면서 probability의 approximation 값을 구하고, reward 역시 같은 방법으로 계산해야한다.

### **정리**

이 글에서는 reinforcement learning의 가장 기본적인 모델인 MDP에 대해 다루었다. MDP는 state, action, reward function, transition matrix와 discount factor로 구성된 튜플이며, optimal policy를 구하기 위해서 value function이라는 개념을 도입하고, 이 optimal value function을 계산할 수 있다면 optimal policy를 구할 수 있다. Optimal value function을 update하기 위해서 Bellman Equation이라는 관계식을 사용해 value iteration과 policy iteration이라는 알고리즘까지 살펴보았다. 이 경우 모든 reward와 transition matrix는 이미 알려져있다고 가정하였고, 만약 모르는 경우 finite MDP에서는 실제 estimation을 통해 model을 leanring해야한다는 얘기까지 하였다. 실제로 MDP 문제를 다루게 될 경우 이것보다 훨씬 복잡한 문제를 다뤄야할 일이 많지만, 근본적으로는 value iteration 등을 사용하여 optimal value function 혹은 optimal action value function의 값을 구해 optimal policy를 구한다는 기본적인 아이디어는 같다.

### **Reference**

* [Reinforcement Learning Lecture Note - Stanford CS229 Lecture Note by Andrew Ng](http://cs229.stanford.edu/notes/cs229-notes12.pdf)