

Integrated Modeling and Control Based on Reinforcement Learning and Dynamic Programming

Title: Week 1 Dyna-Q paper review

Presenter: Lee jung woo

Today's Art

- Pierrot and Halequin (1972)
- Picasso



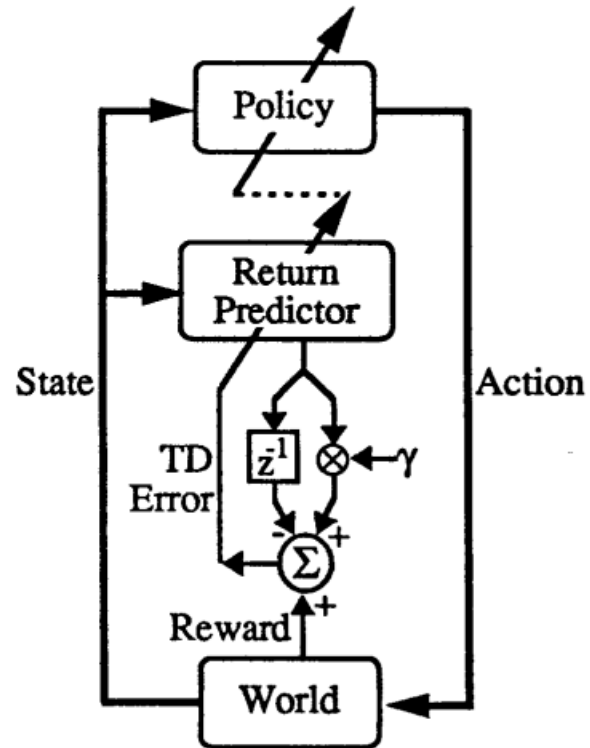
0. Abstract

- Dyna architectures integrate trial-and-error (reinforcement) learning and execution-time planning into a single process operating **alternately on the world and on a learned forward model of the world.**
- Dyna 는 실제 world와 learned forward model of the world를 번갈아가며, 단일 처리 과정에서 시도-실패 학습(강화학습)과 실행시간에서의 계획을 결합한 구조이다.
- We describe and show results for two Dyna architectures, **Dyna-AHC and Dyna-Q.**
- 우리는 2가지 Dyna-AHC와 Dyna-Q라는 Dyna 구조를 통해 결과를 보여줬다.

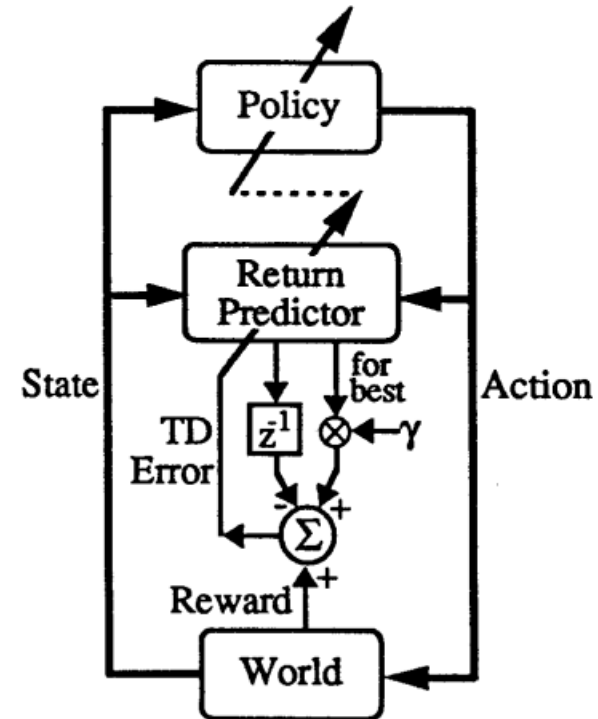
0. Abstract

- Using a navigation task, results are shown for a simple Dyna-AHC system which simultaneously learns by trial and error, learns a world model, and plans optimal routes using the evolving world model.
- navigation task(미로찾기)를 이용해 간단한 Dyna-AHC system을 실험했다. Dyna-AHC는 동시에 시도-실패에 따라 학습하며, world model을 학습하고 결합된 world model을 사용해 최적의 경로를 계획한다.
- We show that Dyna-Q architectures (based on Watkins's Q-learning) are easy to adapt for use in changing environments.
- 또한, Watkin의 Q-learning에 기반한 Dyna-Q 구조를 보여줬다. Dyna-Q 는 변화하는 환경에 쉽게 적용 가능하다.

0. Abstract (AHC & Q-Learning)



C) Adaptive Heuristic Critic



D) Q-Learning

1. Introduction

- Dyna architectures (Sutton, 1990) use learning algorithms to approximate the conventional optimal control technique known as dynamic programming (DP) (Bellman, 1957; Bertsekas, 1987).
- Dyna 구조는 DP로 알려진 전통적인 최적 제어 기법을 근사하기 위한 학습 알고리즘을 사용한다.
- DP itself is not a learning method, but rather a computational method for determining optimal behavior given a complete model of the task to be solved.
- DP 자체는 학습 방법이 아니지만, 문제를 해결하기 위해 주어진 완전한 모델이 있을 때, 최적의 행동을 측정하기 위한 계산적인 방법보다 낫다.

1. Introduction

- It is very similar to state-space search, but differs in that it is more incremental and never considers actual action sequences explicitly, only single actions at a time.
- state-space를 탐색하는 것과 유사하지만, 더 많은 incremental(확장?)과 실제 action sequences를 명확하게 고려할 필요가 없다. 시간당 single actions만 고려하면 된다.

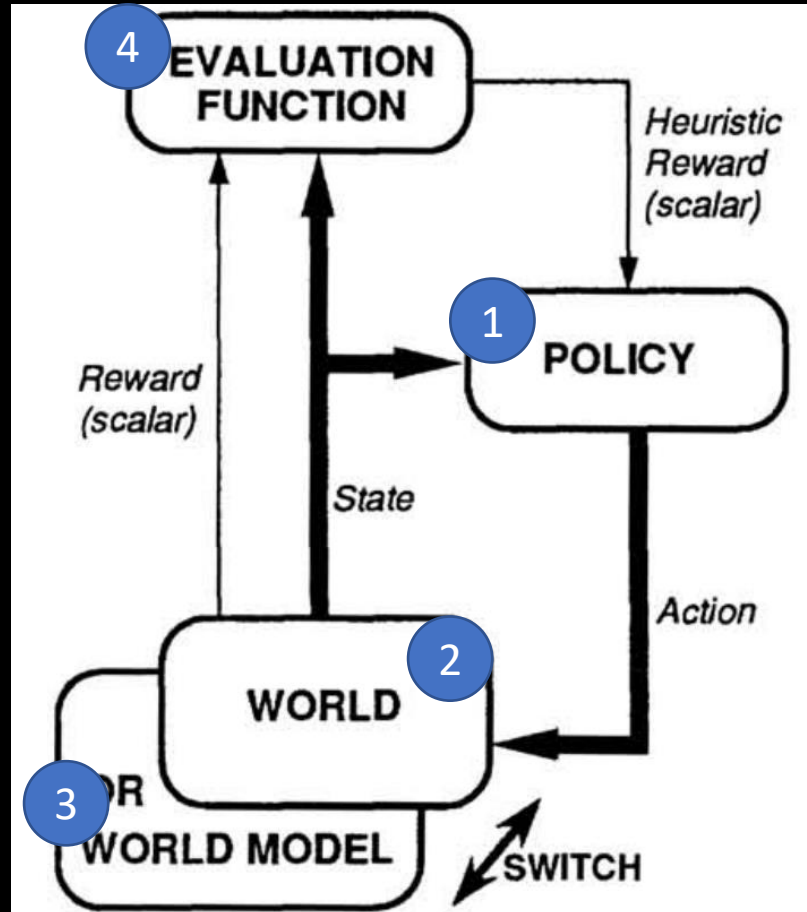
1. Introduction

- Dyna architectures are those that learn a world model online while using approximations to DP to learn and plan optimal behavior.
- Dyna 구조는 Dp로 최적 행동에 대한 학습과 계획을 근사하면서 world model을 online으로 학습한다.
- The theory of Dyna is based on the theory of DP and on DP's relationship to reinforcement learning (Watkins, 1989; Barto, Sutton & Watkins, 1989, 1990), to temporal-difference learning (Sutton, 1988), and to AI methods for planning and search (Korf, 1990).
- Dyna 이론의 기반은 DP 이론이며, 계획과 탐색을 위한 RL, TD, AI 방법론들과 DP의 관계이다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration

- The Dyna-AHC architecture is based on approximating a DP method known as policy iteration (see Bertsekas, 1987). It consists of four components interacting as shown in Figure 1.
- Dyna 구조는 policy iteration으로 알려진 DP 방법을 이용한 근사를 기반으로 한다. Dyna 구조는 상호작용하는 4개의 구성들로 이루어져 있다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration



2. Dyna-AHC: Dyna by Approximating Policy Iteration

- The policy is simply the function formed by the current set of reactions; it receives as input a description of the current state of the world and produces as output an action to be sent to the world.
- policy(정책)은 간단히 현재 반응들의 집합으로 구성된 함수이다. input으로 현재의 state의 정보를 받고 output action으로 world에 전달한다.
- The world represents the task to be solved; prototypically it is the robot's external environment. The world receives actions from the policy and produces a next state output and a reward output.
- World는 해결되어지는 task를 말한다. 일반적으로 로봇의 외부 환경을 말한다. World(environment)는 actions을 policy로부터 inputs으로 받고 next state와 reward를 output으로 반환해준다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration

- The overall task is defined as maximizing the long-term average reward per time step. The architecture also includes an explicit world model.
- 모든 task는 time step당 long-term 평균 reward를 극대화하는 것으로 정의한다. Dyna 구조에 서는 또한 분명한 world model을 포함한다.(실제 구현시 경험을 그대로 저장합니다.)
- The world model is intended to mimic the one-step input-output behavior of the real world.
- World model은 real world의 one-step input-out 행동을 따라할 의도를 가진다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration

- the Dyna-AHC architecture includes an evaluation function that rapidly maps states to values, much as the policy rapidly maps states to actions. The evaluation function, the policy, and the world model are each updated by separate learning processes.
- Dyna-AHC 구조는 빠르게 states를 values로 나타내고, 마찬가지로 policy를 states를 actions으로 빠르게 나타내는 evaluation function을 포함합니다. evaluation function, policy, world model은 각각 분리된 학습 과정에 의해 업데이트 됩니다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration

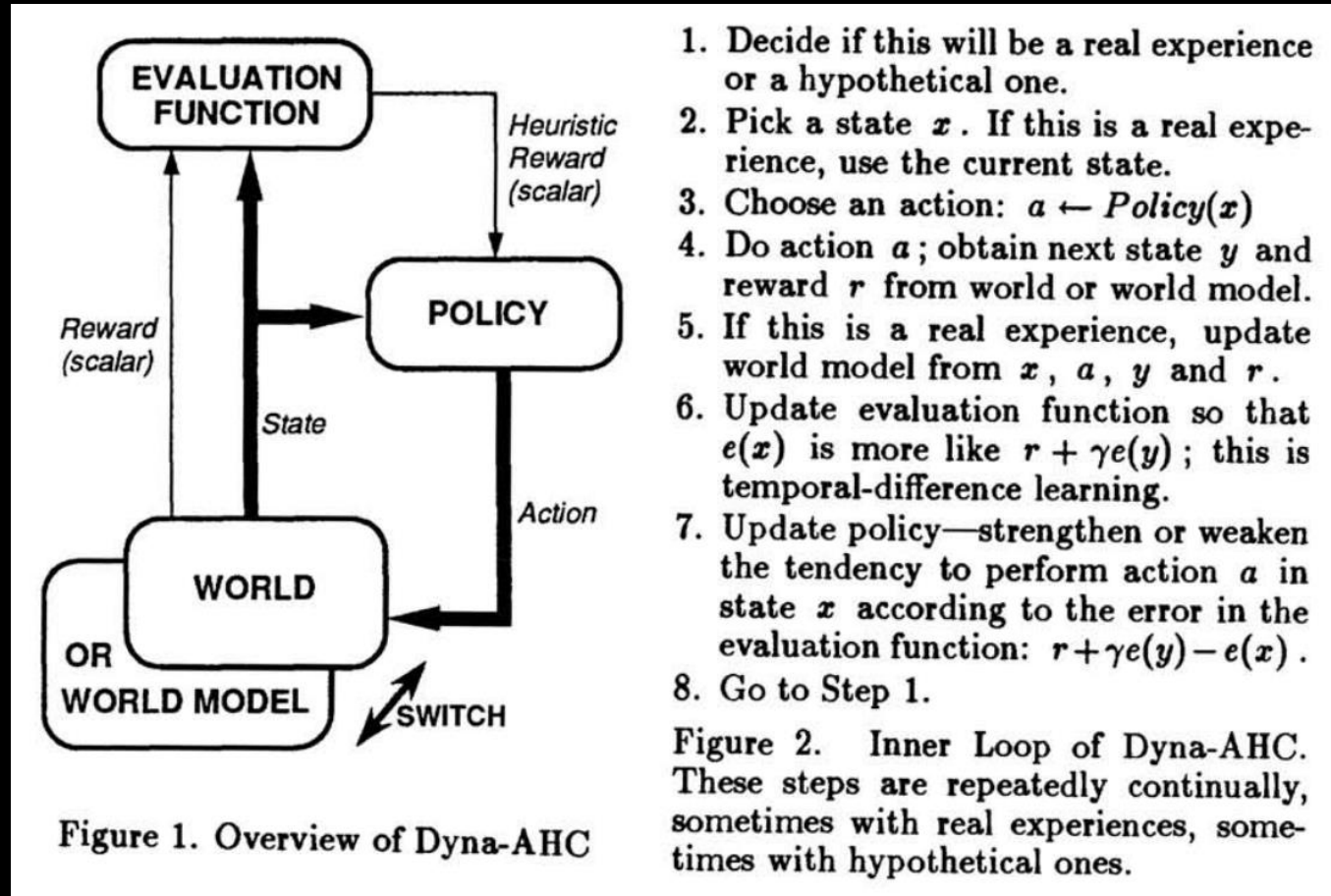
- The policy is continually modified by an integrated planning/learning process. The policy is, in a sense, a plan, but one that is completely conditioned by current input. The planning process is incremental and can be interrupted and resumed at any time. It consists of a series of shallow searches, each typically of one ply, and yet ultimately produces the same result as an arbitrarily deep conventional search. I call this relaxation planning.
- policy는 계속적으로 통합된 계획-학습 과정에 의해 수정됩니다. 그 policy는 어떤 의미에서는 계획일 수 있다. 하지만 policy는 input에 의한 완전히 조건화 된 것 입니다. planning process는 증가되고 어떤 시점이든 중단하거나 멈출 수 있습니다. 이것은 약한 탐색의 연속으로 구성되어 있습니다. 각각 1번의 동작이며, 그럼에도 궁극적으로 깊은 전통적인 탐색과 같은 결과를 만들어 낸다. 나는 이것을 relaxation planning(완화 계획? 휴식 계획?)이라고 부른다.

2. Dyna-AHC: Dyna by Approximating Policy Iteration

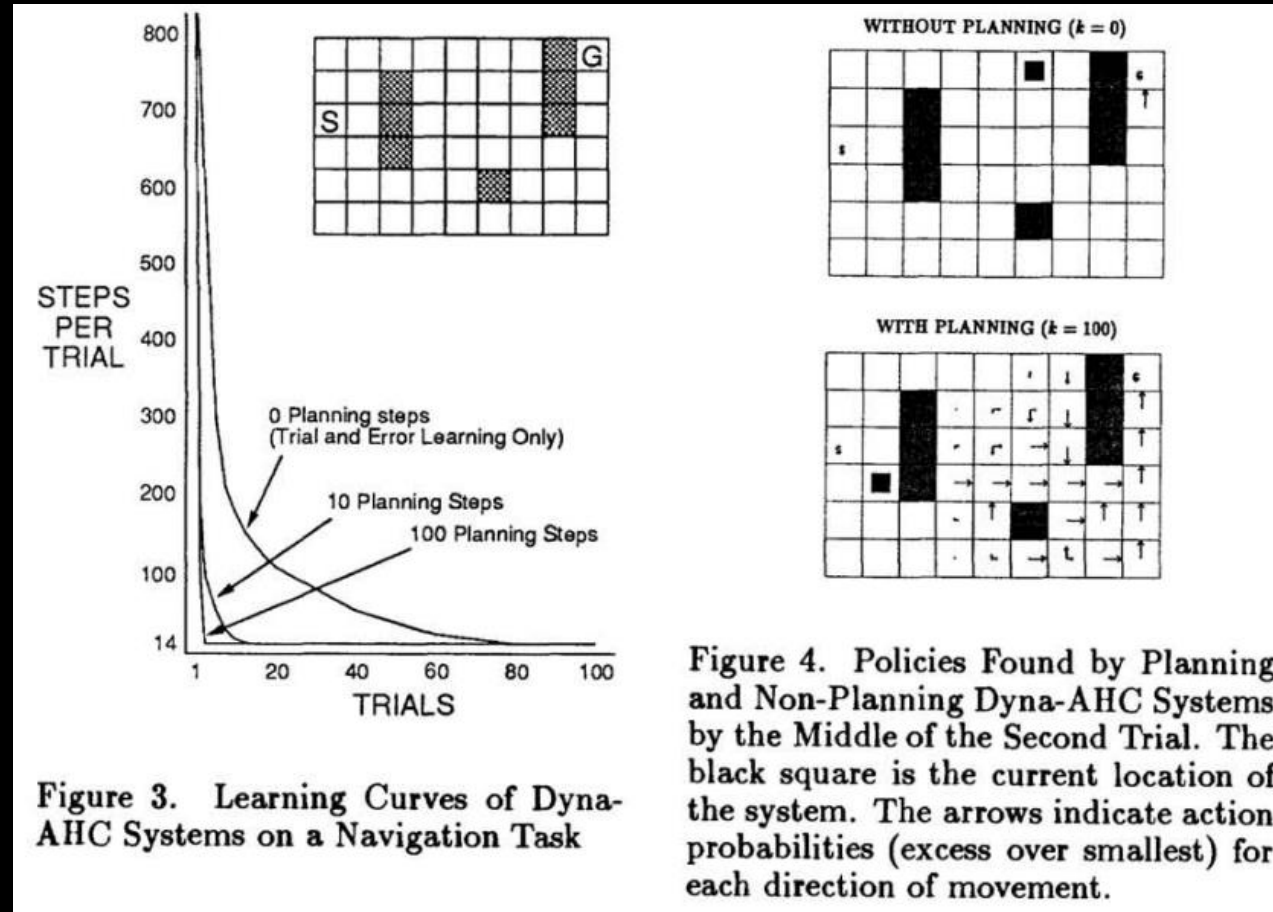
- relaxation planning.
- relaxation planning이란, 이전 장에서 말한 약한 탐색을 통해 planning하는 과정을 말합니다. evaluation function을 사용해 planning에 이용하며, policy iteration과 Dyna-AHC에서 사용된 evaluation function을 이용한 policy update과정과 완전히 같지는 않다고 말합니다.
- 즉, relaxation planning은 shallow planning을 하는 개념을 말하는 것이고, 실제 적용에 있어서는 알고리즘마다 중재안으로 적당히 shallow한 성질만 담아 적용했다고 보면 됩니다.

$$e(x) \text{ “=” } \max_{a \in \text{Actions}} E \{ r + e(y) \mid x, a \},$$

2. Dyna-AHC: Dyna by Approximating Policy Iteration

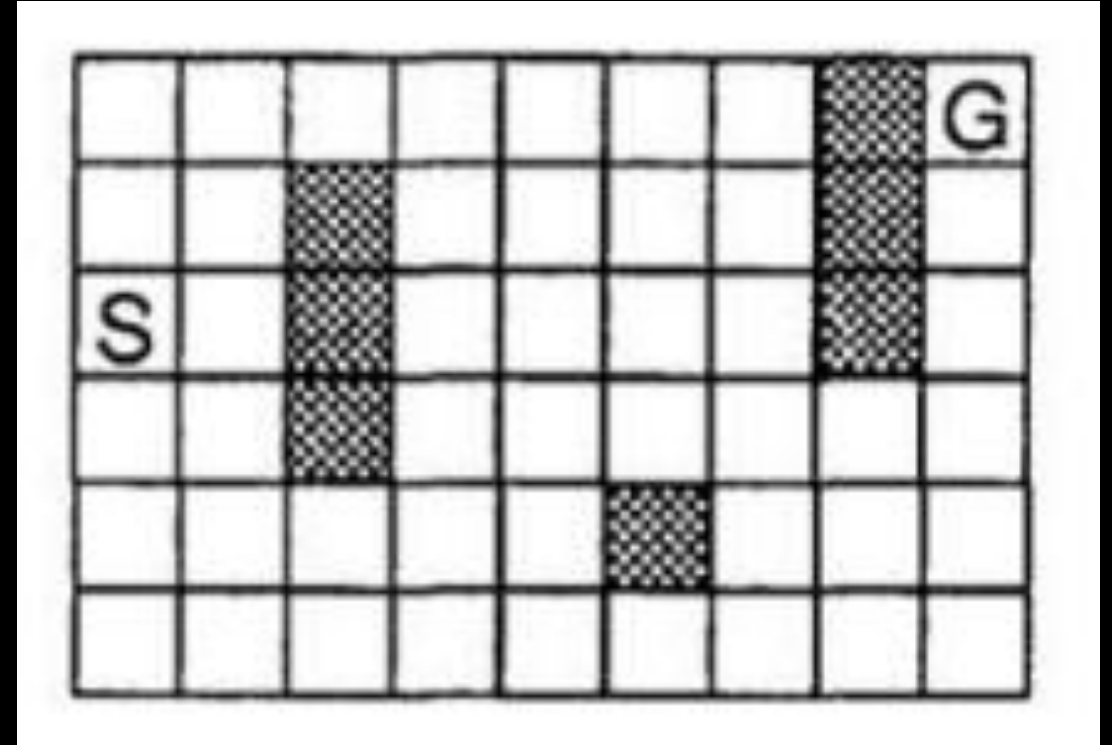


2. Dyna-AHC: Dyna by Approximating Policy Iteration



3. A Navigation Task

- Environment
- Map size: 6 by 9
- Type: grid
- Actions: up, down, right, left
- reward: only goal state +1, others 0
- Start state: S(3,1)
- Goal state: G(1,9)



3. A Navigation Task

- Without planning ($k = 0$), each trial adds only one additional step to the policy, and so only one step (the last) has been learned so far. With planning, the first trial also learned only one step, but here during the second trial an extensive policy has been developed that by the trial's end will reach almost back to the start state.
- planning을 하지 않을 때는 one-step만 진행되지만, planning을 하게 되면, second trial동안 start state로 거의 도달할 수 있는 확장된 policy를 trial의 끝 즈음에 발전 시킬 수 있다.

4. Dyna-Q: Dyna by Q-learning

- The Dyna- Q architecture is the combination of this new kind of learning with the Dyna idea of using a learned world model to generate hypothetical experience and achieve planning.
- Dyna-Q 구조는 learned world model을 사용해 가상의 경험을 생성하고 계획을 하는 결합된 새로운 종류의 학습이다.
- Whereas the AHC reinforcement learning architecture maintains two fundamental memory structures, the evaluation function and the policy, Q-learning maintains only one. That one is a cross between an evaluation function and a policy.
- AHC 강화학습 구조는 evaluation function과 policy 2개의 기본적인 메모리 구조이며. Q-learning은 evaluation function과 policy 2가지를 1개로 유지한다.

4. Dyna-Q: Dyna by Q-learning

best state-action pair: $e(x) \stackrel{\text{def}}{=} \max_a Q_{xa}$. In general, the Q-value for a state x and an action a should equal the expected value of the immediate reward r plus the discounted value of the next state y :

$$Q_{xa} \text{ " = " } E \{ r + \gamma e(y) \mid x, a \}. \quad (3)$$

To achieve this goal, the updating steps (Steps 6 and 7 of Figure 2) are implemented by

$$Q_{xa} \leftarrow Q_{xa} + \beta (r + \gamma e(y) - Q_{xa}). \quad (4)$$

This is the only update rule in Q-learning. We note that it is very similar though not identical to Holland's bucket brigade and to Sutton's (1988) temporal-difference learning.

4. Dyna-Q: Dyna by Q-learning

- An exploration bonus of $\epsilon\sqrt{n_{xa}}$ is used to make actions that have not been tried in a long time (and that therefore have uncertain consequences) appear more attractive by replacing (4) with
- $\epsilon\sqrt{n_{xa}}$ state x , action a 가 오래 선택되지 않을 수록 값이 커진다.

$$Q_{xa} \leftarrow Q_{xa} + \beta(r + \epsilon\sqrt{n_{xa}} + \gamma e(y) - Q_{xa}).$$

5. Changing-World Experiments

- Three Dyna systems were used: the Dyna-AHC, Dyna-Q+ system, Dyna-Q- system.
- Dyna-AHC: policy iteration 기반의 dyna system.
- Dyna-Q+ system: Q-learning 기반의 dyna system with exploration bonus.
- Dyna-Q- system: Q-learning 기반의 dyna system without exploration bonus.

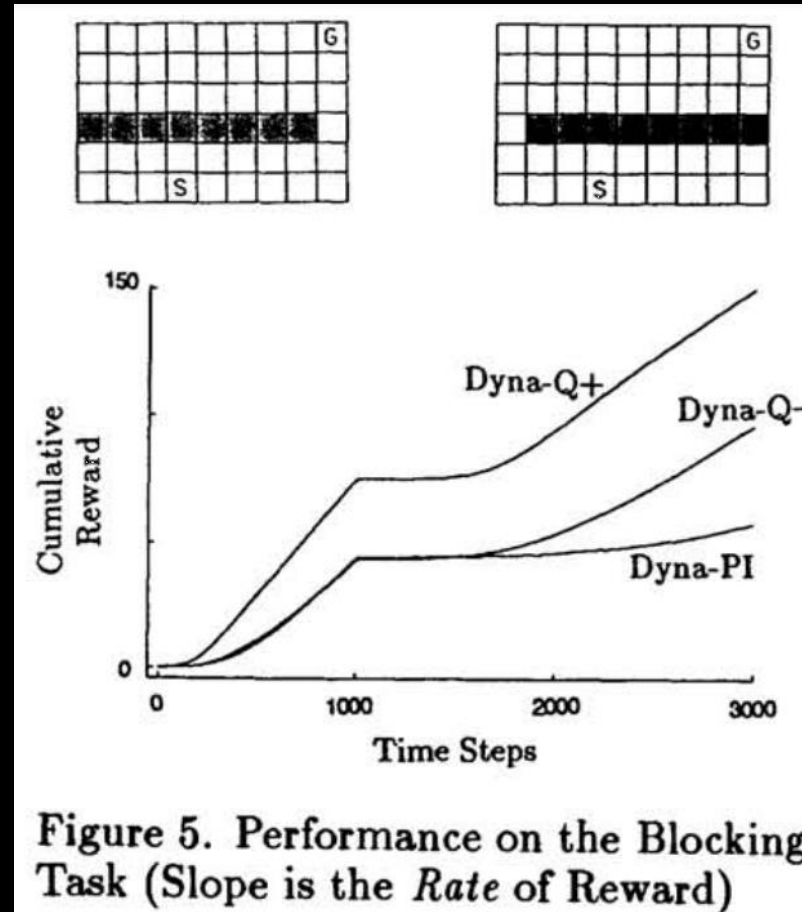
5. Changing-World Experiments

(blocking experiment)

- Initially a short path from start to goal was available (first maze). After 1000 time steps, by which time the short path was usually well learned, that path was blocked and a longer path was opened (second maze).
- 첫번째 미로는 짧은 경로로 도달할 수 있다. 1000 steps까진 first maze를 학습한다. 그 이후로 더 먼 경로를 가진 second maze를 학습한다.
- In the first 1000 trials, all three Dyna systems found a short route to the goal, though the Dyna-Q+ system did so significantly faster than the other two. After the short path was blocked at 1000 steps, the graph for the Dyna-AHC system remains almost flat, indicating that it was unable to obtain further rewards. The Dyna-Q systems, on the other hand, clearly solved the blocking problem, reliably finding the alternate path after about 800 time steps.

5. Changing-World Experiments

(blocking experiment)



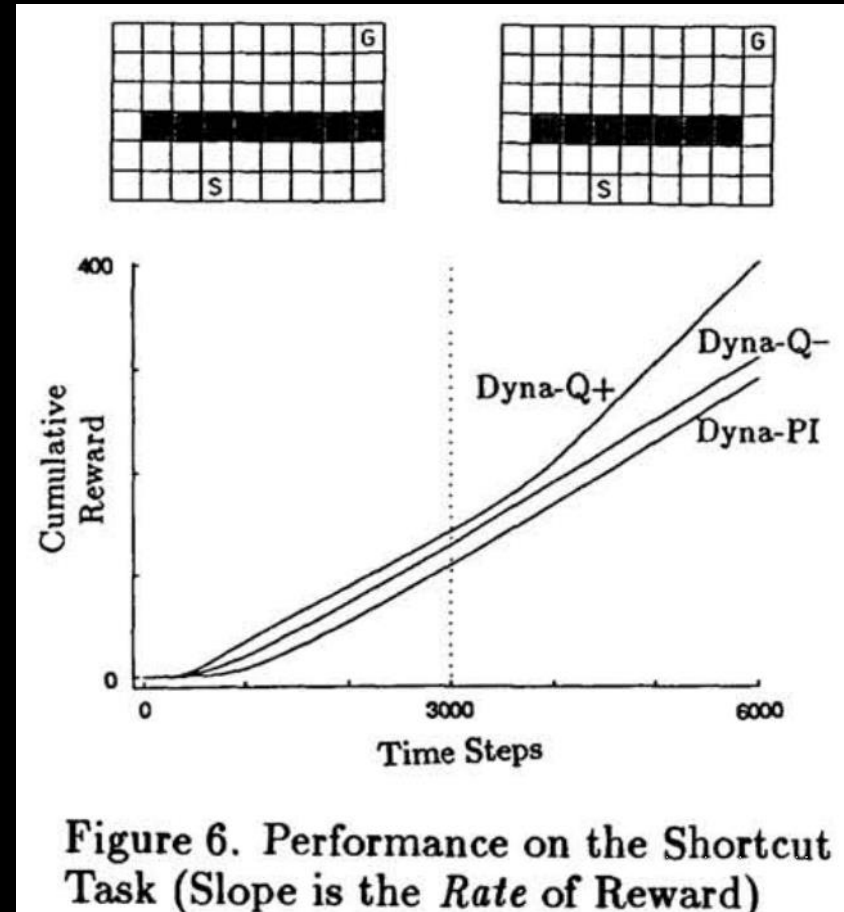
5. Changing-World Experiments

(shortcut experiment)

- After 3000 times steps all three Dyna systems had learned the long path, and then a shortcut was opened without interfering with the long path
- 3000 step 이후로 3가지 Dyna systems은 long path에 대해 학습했다. 그후 shortcut이 열렸다.
- The Dyna-Q+ system also learned the original long route faster than the Dyna-Q- system, which in turn learned it faster than the Dyna-AHC system. However, the ability of the Dyna-Q+ system to find shortcuts does not come totally for free . Continually re-exploring the world means occasionally making suboptimal actions.

5. Changing-World Experiments

(shortcut experiment)



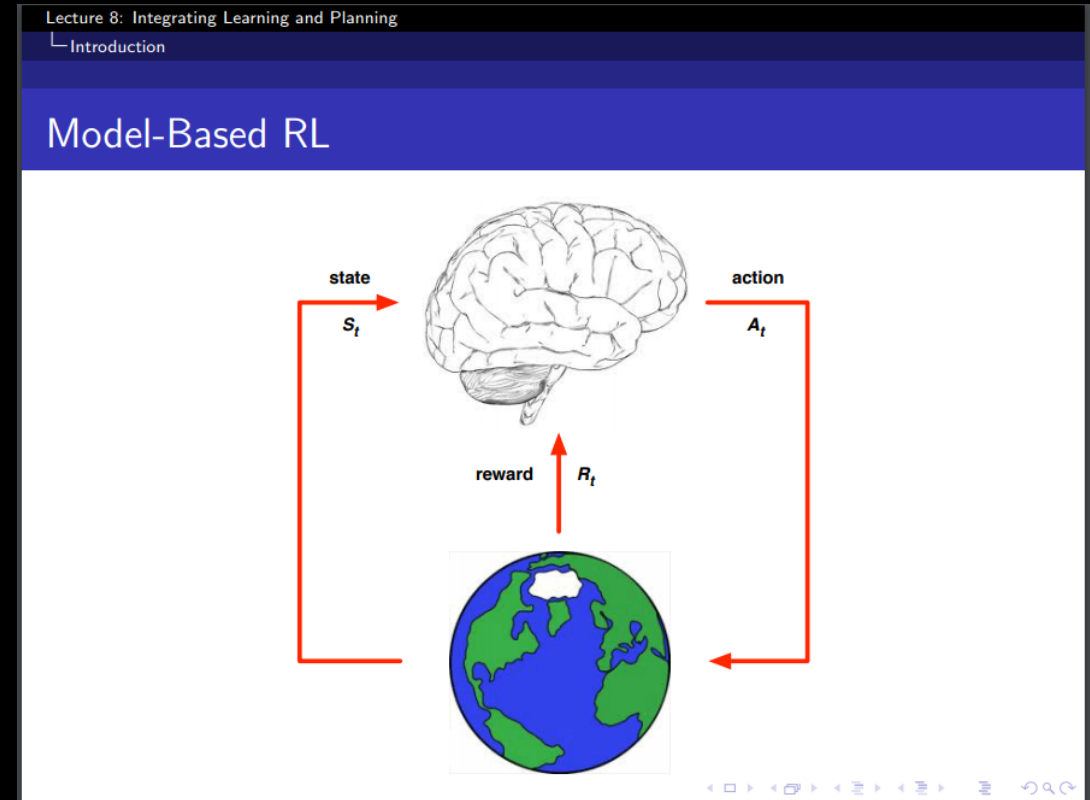
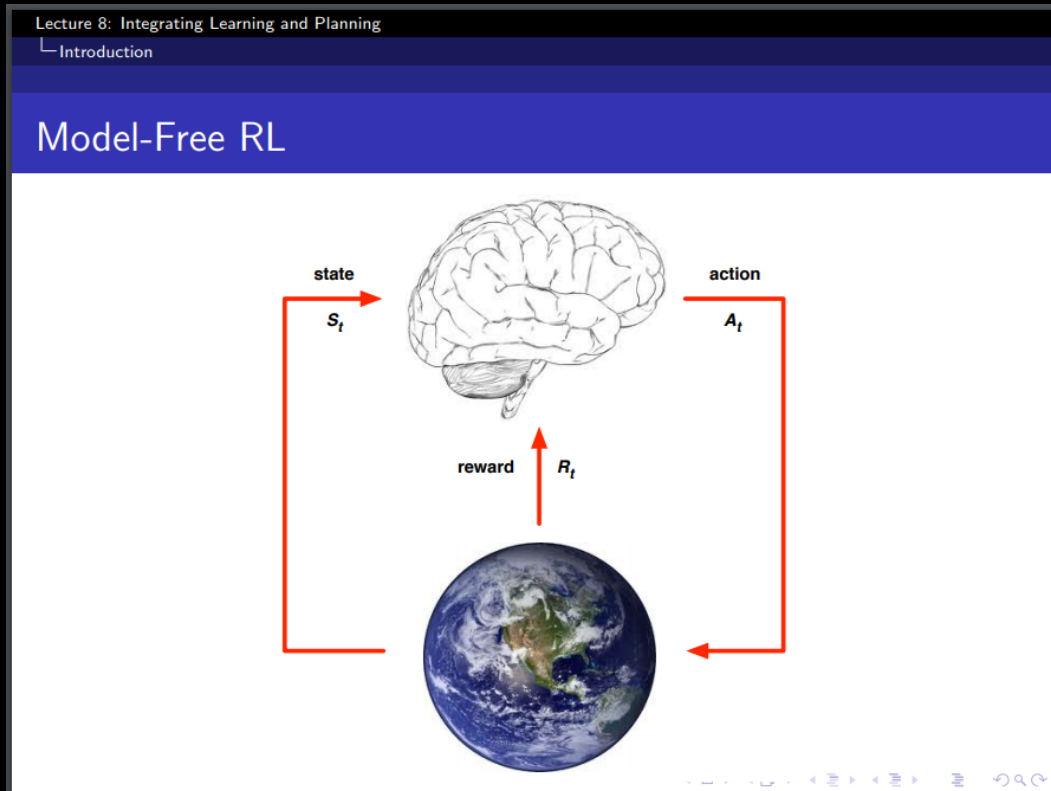
5. Changing-World Experiments

- In a static environment, Dyna-Q+ will eventually perform worse than Dyna-Q-, whereas, in a changing environment, it will be far superior, as here. One possibility is to use a meta-level learning process to adjust the exploration parameter ϵ to match the degree of variability of the environment.
- 정적인 환경에서 Dyna-Q+는 Dyna-Q보다 좋지 못하다.(exploration을 수시로 해서) 하지만, 변화하는 환경에서는 더 좋다.

6. Limitations and Conclusions

- The state and action spaces are small and denumerable, permitting tables to be used for all learning processes and making it feasible for the entire state space to be explicitly explored. In addition, these results have assumed knowledge of the world state, have used a trivial form of search control (random exploration), and have used terminal goal states.
- They show that the use of a forward model can dramatically speed trial-and-error (reinforcement) learning processes even on simple problems. Moreover, they show how planning can be done with the incomplete, changing, and of times incorrect world models that are constructed through learning.

Supplements




Supplements

Lecture 8: Integrating Learning and Planning

- └ Integrated Architectures
 - └ Dyna

Integrating Learning and Planning

- Model-Free RL
 - No model
 - **Learn** value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
 - Learn a model from real experience
 - **Plan** value function (and/or policy) from simulated experience
- Dyna
 - Learn a model from real experience
 - **Learn and plan** value function (and/or policy) from real and simulated experience



Supplements

Model-Based vs. Model-Free Algorithms

Models:

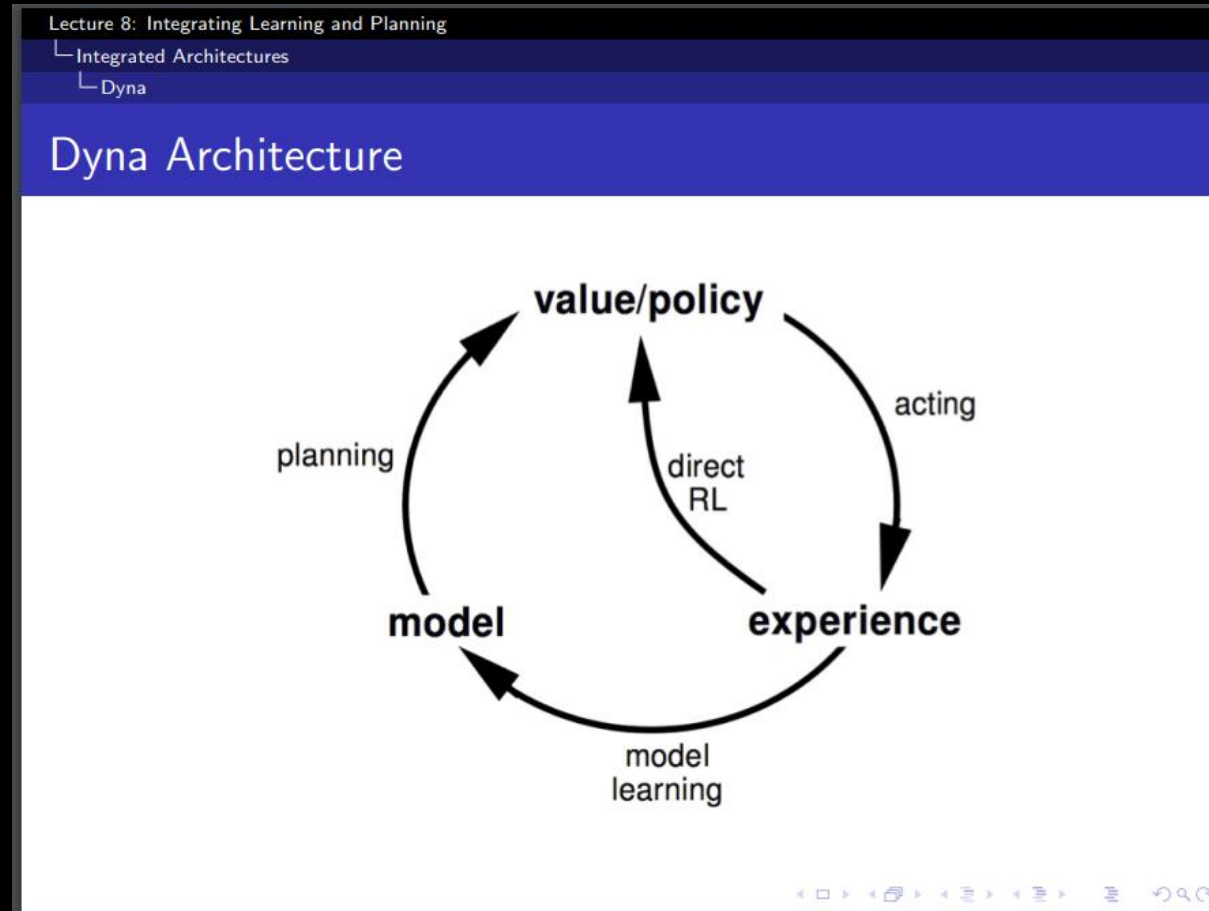
- + Easy to collect data in a scalable way (self-supervised)
- + Possibility to transfer across tasks
- + Typically require a smaller quantity of supervised data
- Models don't optimize for task performance
- Sometimes harder to learn than a policy
- Often need assumptions to learn complex skills (continuity, resets)

Model-Free:

- + Makes little assumptions beyond a reward function
- + Effective for learning complex policies
- Require a lot of experience (slower)
- Not transferable across tasks

Ultimately we will want both!

Supplements



Supplements

Lecture 8: Integrating Learning and Planning

- └ Integrated Architectures
 - └ Dyna

Dyna-Q Algorithm

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \epsilon\text{-greedy}(S, Q)$
- (c) Execute action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Repeat n times:
 - $S \leftarrow$ random previously observed state
 - $A \leftarrow$ random action previously taken in S
 - $R, S' \leftarrow Model(S, A)$
 - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

