CPS 제어 및 분석 알고리즘 특론

- Reinforcement Learning -

부산대학교

김호원

부산대
정보보호 및 IoT 연구실,
사물인터넷연구센터

2018.10

I. Basics & Features of RL

Supervised/Unsupervised/RL

Machine Learning Definition

- Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data, such as from sensor data or databases.

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data .

Supervised/Unsupervised/RL

- **Machine Learning Type:**

With respect to the feedback type to learner:

- **Supervised learning** : Task Driven (Classification)
- **Unsupervised learning** : Data Driven (Clustering)
- **Reinforcement learning** —
  - Close to human learning.
  - Algorithm learns a policy of how to act in a given environment.
  - Every action has some impact in the environment, and the environment provides rewards that guides the learning algorithm.

참고: Reinforcement Learning by Chandra Prakash

Supervised/Unsupervised/RL

- **Reinforcement(강화)의 의미:**
  - **Occurrence of an event, in the proper relation to a response, that tends to increase the probability that the response will occur again in the same situation.**
  - **Response**에 적절한 형태로 **event**를 발생시킬 경우, 해당 **Response**는 동일한 상황에서 다시 발생할 확률이 높아짐

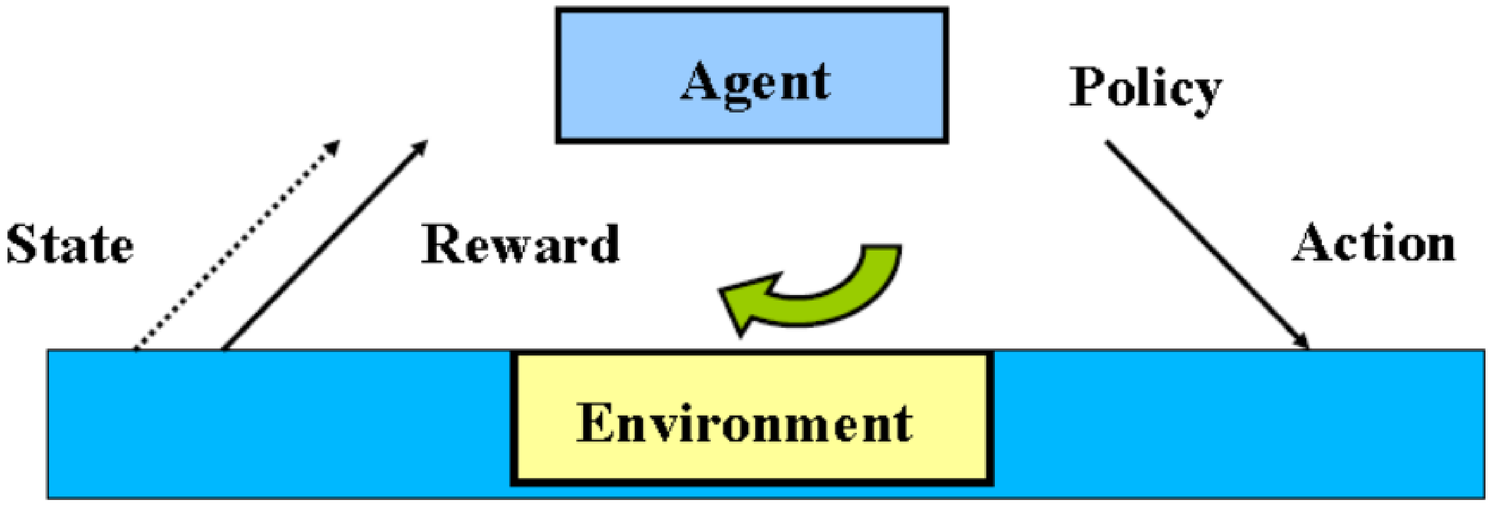
- **Reinforcement learning** is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment.
  - ❑ Reinforcement Learning is learning how to act in order to maximize a numerical reward.

  - 즉, **강화학습은 agent**가 동적 환경에서, **trial-and-error**를 통해 행동을 학습하는 문제임
  - 강화학습은 **"correctness**의 표준을 제시하지 않으면서" 학습자의 성과를 평가하는 **"learning feedback"** 특성을 가짐

참고: Reinforcement Learning by Chandra Prakash

강화학습의 주요 구성 요소

Reinforcement Learning 구성 요소



- **Agent**: Intelligent programs
- **Environment**: External condition
- **Policy**:
  - Defines the agent's behavior at a given time
  - A mapping from states to actions
  - Lookup tables or simple function

참고: Reinforcement Learning by Chandra Prakash

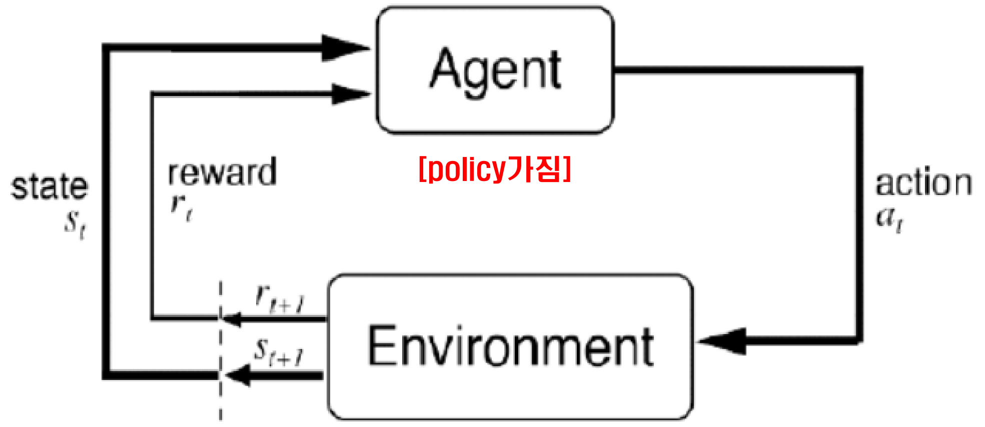
강화학습의 주요 구성 요소

- **Reward function**
  - **RL problem**에서 목표를 정의함
  - **Policy**는 이 목표를 성취하기 위해 변경됨
- **Value function**
  - **Reward function indicates what is good in an immediate sense while a value function specifies what is good in the long run.**
    - **Reward function:** 즉각적인 보상, **value function:** 장기적 관점에서 유익한 것이 무엇인지 알려줌
  - **Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.**
    - **State**의 **value: agent**가 기대하는 총 **reward**의 합
- **Model of the environment**
  - **Environment**에 대한 **mimic behavior**를 예측함
  - **Used for planning**
  - **If we know current state and action, then predict the resultant next state and next reward**

참고: Reinforcement Learning by Chandra Prakash

강화학습의 주요 구성 요소

Agent/Environment Interface, Steps for RL



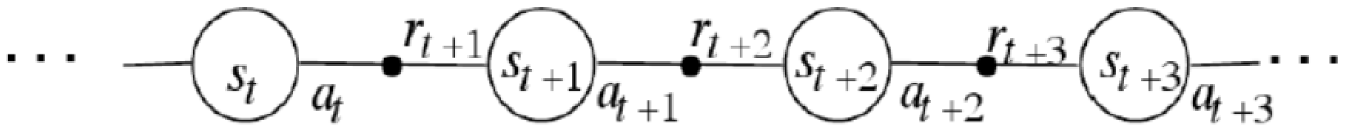
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \ldots$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



1. The agent observes an input state
2. An action is determined by a decision making function (policy)
3. The action is performed
4. The agent receives a scalar reward or reinforcement from the environment
5. Information about the reward given for that state / action pair is recorded

참고: Reinforcement Learning by Chandra Prakash

강화학습의 주요 구성 요소

Silent Features of RL

- **Set of problems** rather than a set of techniques
- Without specifying how the task is to be achieved.

- **"RL as a tool"** point of view:
  - RL is training by rewards and punishments.
  - Train tool for the computer learning.

- **The learning agent's point of view:**
  - RL is learning from trial and error with the world.
  - Eg. how much reward I much get if I get this.

강화학습의 주요 구성 요소

Silent Features of RL

- Reinforcement Learning uses **Evaluative Feedback**
- **Purely Evaluative Feedback**
  - Indicates how good the action taken is.
  - Not tell if it is the best or the worst action possible.
  - Basis of methods for function optimization.
- **Purely Instructive Feedback**
  - Indicates the correct action to take, independently of the action actually taken.
  - Eg: Supervised Learning

강화학습의 주요 구성 요소

(정확한 최적 데이터 기반) 지도학습 vs. 강화학습의 평가 피드백

Supervised vs. evaluative feedback is concerned with **how the learner is informed of right and wrong answers.**

A requirement for applying supervised learning methods is the availability of examples with known optimal decisions. (**지도 학습에서는 알려진 최적 결정을 내릴 수 있는 예제, 즉 데이터가 있어야 함**)
In the patron-assignment problem, a host-in-training could work as an apprentice to a much more experienced supervisor to learn how to handle a range of situations. If the apprentice can only learn from supervised feedback, however, she would have no opportunities to improve after the apprenticeship ends. (**경험이 많은** superviser**의 지도가 없으면 향상이 없을 것임** O_O)

By contrast, **evaluative feedback** provides the learner with an assessment of the effectiveness of the decisions that she made; no information is available on the appropriateness of alternatives.
(**강화학습은 평가 피드백** ! → **학습자 자신이 내린 결정의 효율성에 대한 평가를 제공함**)

For example, a host might learn about the ability of a server to handle unruly patrons by trial and error**: when the host makes an assignment of a difficult customer,** it is possible to tell whether things went smoothly with the selected server, **but no direct information is available as to whether one of the other servers might have been a better choice.** The central problem in the field of reinforcement learning is addressing the challenge of evaluative feedback.

"Reinforcement learning improves behaviour from evaluative feedback from https://www.nature.com/articles/nature14540

Associative & Non Associative Tasks

- **Associative :**
  - Situation Dependent
  - Mapping from situation to the actions that are best in that situation
- **Non Associative:**
  - Situation independent
  - No need for associating different action with different situations.
  - Learner either tries to find a single best action when the task is stationary, or tries to track the best action as it changes over time when the task is non stationary.

II. N-armed Bandit Problem

Bandits Problem

N-armed bandit problem

We have to choose from n different options or actions.
We will choose the **one with maximum reward**.

One-Bandit
"arms"

Pull arms sequentially so as to maximize the total expected reward

It is non associative and evaluative

Bandits Problem

- **Two-armed bandit problem**
  - This tutorial contains a simple example of how to build a policy-gradient based agent that can solve the multi-armed bandit problem

- **The Bandits**
  - Here we define our bandits. For this example we are using a four-armed bandit. The pullBandit function generates a random number from a normal distribution with a mean of 0.
  - The lower the bandit number, the more likely a positive reward will be returned. We want our agent to learn to always choose the bandit that will give that positive reward.

Bandits Problem

The Bandits

- Here we define our bandits. For this example we are using a four-armed bandit. The pullBandit function generates a random number from a normal distribution with a mean of 0.
- The lower the bandit number, the more likely a positive reward will be returned. We want our agent to learn to always choose the bandit that will give that positive reward.

```
#List out our bandits. Currently bandit 4 (index#3) is set to most often provide a positive reward.
bandits = [0.2,0,-0.2,-5]
num_bandits = len(bandits)
def pullBandit(bandit):
  #Get a random number.
  result = np.random.randn(1)
  if result > bandit:
     #return a positive reward.
     return 1
  else:
     #return a negative reward.
     return -1
```

Bandits Problem

Agent modeling

- **The code below established our simple neural agent. It consists of a set of values for each of the bandits. Each value is an estimate of the value of the return from choosing the bandit. We use a policy gradient method to update the agent by moving the value for the selected action toward the received reward.**

```
tf.reset_default_graph()

#These two lines established the feed-forward part of the network. This does the actual choosing.

weights = tf.Variable(tf.ones([num_bandits]))
chosen_action = tf.argmax(weights,0)

// weights가 총 4개의 항을 가지므로, argmax는 4개의 항 중에서 가장 큰 값을 갖는 index를 리턴함
```

Bandits Problem

Agent modeling

#The next six lines establish the training procedure.

# We feed the reward and chosen action into the network to compute the loss, and use it to update the network.

```
reward_holder = tf.placeholder(shape=[1],dtype=tf.float32)
action_holder = tf.placeholder(shape=[1],dtype=tf.int32)
responsible_weight = tf.slice(weights,  action_holder, [1])
```

// weights **값에서** action_holder**에 해당하는 값** 1**개를 추출함** (**아래의** π**에 해당함**. **정책임**)

```
loss = -(tf.log(responsible_weight)*reward_holder)  // Loss = -log(π)*A
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
update = optimizer.minimize(loss)
```

Bandits Problem

Training Agent

We will train our agent by taking actions in our environment, and receiving rewards.

Using the rewards and actions, we can know how to properly update our network in order to more often choose actions that will yield the highest rewards over time.

```
total_episodes = 1000      #Set total number of episodes to train agent on.
total_reward = np.zeros(num_bandits)       #Set scoreboard for bandits to 0.
e = 0.1             #Set the chance of taking a random action.

init = tf.initialize_all_variables()

# Launch the tensorflow graph
with tf.Session() as sess:
   sess.run(init)
   i = 0
```

Training Agent

```
while i < total_episodes:
  # Choose either a random action or one from our network.
  # 이를 보면, 원래는 chose_action(즉, weights 중에서 가장 큰 값을 갖는 것을
  # 선택하는데, 변칙으로 e (0.1) 값보다 작을 경우, 아무거나 다른 것을 선택한다.
  if np.random.rand(1) < e:
     action = np.random.randint(num_bandits)
  else:
     action = sess.run(chosen_action)
     reward = pullBandit(bandits[action]) #Get our reward from picking one of the bandits.

  #Update the network.
   _,resp,ww = sess.run([update,responsible_weight,weights],
                       feed_dict= {reward_holder: [reward], action_holder:[action]})

  #Update our running tally of scores.
  total_reward[action] += reward
      if i % 50 == 0:
     print "Running reward for the " + str(num_bandits) + " bandits: " + str(total_reward)
  i+=1
```

Bandits Problem

Training Agent

```
Running reward for the 4 bandits: [ 1. 0. 0. 0.] Running
reward for the 4 bandits: [ 0. -2. -1. 38.]

…

Running reward for the 4 bandits: [ -4. -7. 2. 720.]
Running reward for the 4 bandits: [ -4. -7. 3. 769.]
Running reward for the 4 bandits: [ -6. -8. 3. 814.]
Running reward for the 4 bandits: [ -7. -7. 3. 858.]


The agent thinks bandit 4 is the most promising.... ...and
it was right!
```

감사합니다

Q & A

부산대학교
PUSAN NATIONAL UNIVERSITY