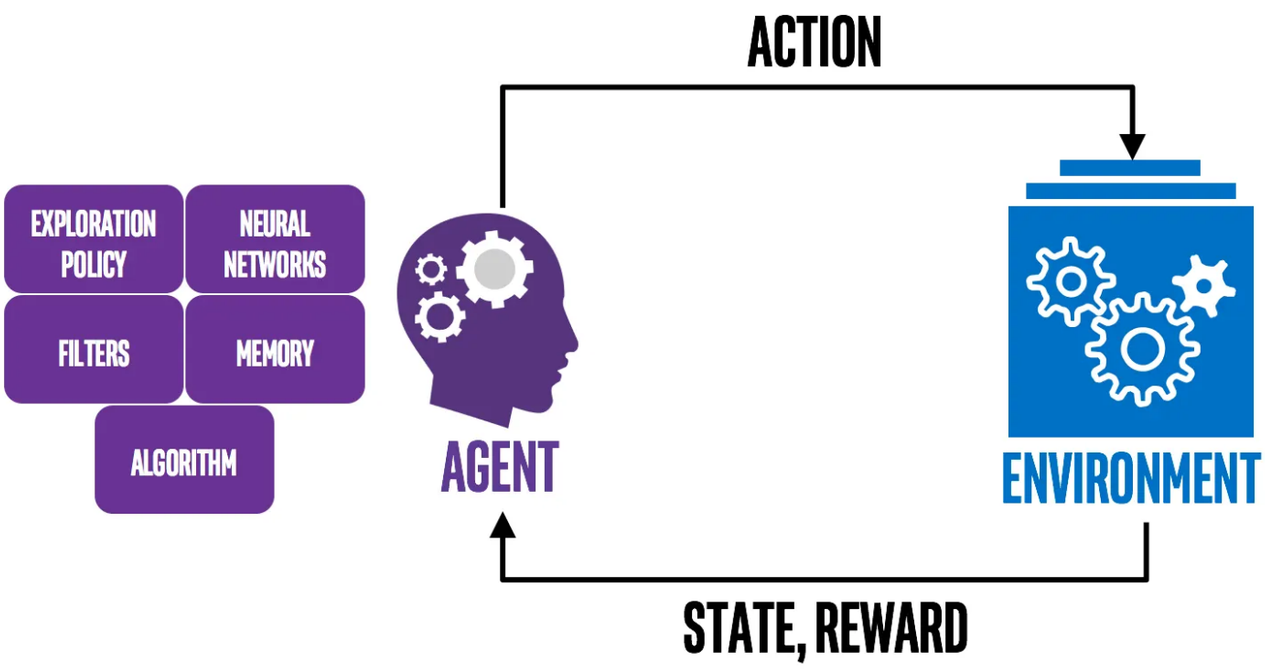
Reinforcement Learning, Part 1: A Brief Introduction

What is Reinforcement Learning and how is it used? Find out in 5 minutes!

dan lee Follow 6 min read · Oct 16, 2019



https://nervanasystems.github.io/coach

Welcome back to my blog for engineers who want to learn AI! In my last post, I talked about how to level up your machine learning career path. Today, I'll help you continue your journey by introducing Reinforcement Learning.

In addition to defining Reinforcement Learning (RL), this article will give

you a simple but essential picture of what it looks like in practical application.

By the end, you will have a basic knowledge of:

- What reinforcement learning is;
- How to frame your task into an RL problem;
- The relationship between RL and supervised/unsupervised learning;
- Using OpenAI Gym to run an RL demo with a simple policy.

How Do We Define Reinforcement Learning?

Reinforcement Learning is a subfield of machine learning that teaches an agent how to choose an action from its action space, within a particular environment, in order to maximize rewards over time.

Reinforcement Learning has four essential elements:

1. **Agent.** The program you train, with the aim of doing a job you specify.
2. **Environment.** The world, real or virtual, in which the agent performs actions.
3. **Action.** A move made by the agent, which causes a status change in the environment.
4. **Rewards.** The evaluation of an action, which can be positive or negative.

Real-World Examples Of Modeling A Reinforcement Learning Task

The first step in modeling a Reinforcement Learning task is determining what the 4 elements are, as defined above. Once each element is defined, you're ready to map your task to them.

Here are some examples to help you develop your RL intuition.

Determining the Placement of Ads on a Web Page

Agent: The program making decisions on how many ads are appropriate for a page.

Environment: The web page.

Action: One of three: (1) putting another ad on the page; (2) dropping an ad from the page; (3) neither adding nor removing.

Reward: Positive when revenue increases; negative when revenue drops.

In this scenario, the agent observes the environment and gets its current status. The status can be how many ads there are on the web page and whether or not there is room for more.

The agent then chooses which of the three actions to take at each step. if programmed to get positive rewards whenever the revenue increase, and negative rewards whenever revenue falls, it can develop its effective policy.

Creating A Personalized Learning System

Agent: The program that decides what to show next in an online learning catalog.

Environment: The learning system.

Action: Playing a new class video and an advertisement.

Reward: Positive if the user chooses to click the class video presented; greater positive reward if the user chooses to click the advertisement; negative if the user goes away.

This program can make a personalized class system more valuable. The user can benefit from more effective learning and the system can benefit through more effective advertising.

Controlling A Walking Robot

Agent: The program controlling a walking robot.

Environment: The real world.

Action: One out of four moves (1) forward; (2) backward; (3) left; and (4) right.

Reward: Positive when it approaches the target destination; negative when it wastes time, goes in the wrong direction or falls down.

In this final example, a robot can teach itself to move more effectively by adapting its policy based on the rewards it receives.

Supervised, Unsupervised, and Reinforcement Learning: What are the Differences?

Difference #1: Static Vs.Dynamic

The goal of supervised and unsupervised learning is to search for and learn about patterns in training data, which is quite static. RL, on the other hand, is about developing a policy that tells an agent which action to choose at each step — making it more dynamic.

Difference #2: No Explicit Right Answer

In supervised learning, the right answer is given by the training data. In Reinforcement Learning, the right answer is not explicitly given: instead, the agent needs to learn by trial and error. The only reference is the reward it gets after taking an action, which tells the agent when it is making progress or when it has failed.

Difference #3: RL Requires Exploration

A Reinforcement Learning agent needs to find the right balance between exploring the environment, looking for new ways to get rewards, and exploiting the reward sources it has already discovered. In contrast, supervised and unsupervised learning systems take the answer directly from training data without having to explore other answers.

Difference #4: RL is a Multiple-Decision Process

Reinforcement Learning is a multiple-decision process: it forms a decision-making chain through the time required to finish a specific job. Conversely, supervised learning is a single-decision process: one instance, one prediction.

An Introduction to OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents in everything from walking

to playing games like Pong or Pinball.

OpenAI Gym gives us game environments in which our programs can take actions. Each environment has an initial status. After your agent takes an action, the status is updated.

When your agent observes the change, it uses the new status together with its policy to decide what move to make next. The policy is key: it is the essential element for your program to keep working on. The better the policy your agent learns, the better the performance you get out of it.

Here is a demo of the game CartPole from OpenAI. We can see the policy on the sixth line: the agent may take a *random* action from its action space.

```
import gym

env = gym.make("CartPole-v1")
observation = env.reset()
for _ in range(1000):
    env.render()
action = env.action_space.sample()  # your agent here (this takes
random actions)
observation, reward, done, info = env.step(action)
if done:
    observation = env.reset()
env.close()
```

The 2nd line creates a CartPole environment.

The 3rd line initializes the status parameters.

The 5th line shows the game.

The 6th line prompts an action with a policy that is ‘random’.

On the 7th line, the action is taken and the environment gives four returns:

- **Observation:** Parameters of the game status. Different games return different parameters. In CartPole, there are four in total. The second parameter is the angle of the pole.
- **Reward:** The score you get after taking this action.
- **Done:** Game over or not over.

- **Info:** Extra debug information. It could be cheating.

The 8th line means if the game is done; restart it.

Reinforcement Learning Demo

The policy can be programmed in any way you like. It can be based on if-else rules or a neural network. Here is a little simple demo, with the simplest policy for the CartPole game.

```
import gym

def policy(observation):
    angle = observation[2]
    if angle < 0:
        return 0
    else:
        return 1

env = gym.make("CartPole-v1")
observation = env.reset()
for _ in range(1000):
    env.render()
    action = policy(observation)
    observation, reward, done, info = env.step(action)
    if done:
        observation = env.reset()
env.close()
```

Of course, the policy function content can be replaced by a neural network, with observation parameters as input and an action as output.

Conclusion:

- Reinforcement Learning is a subfield of machine learning, parallel to but differing from supervised/unsupervised learning in several ways.
- Because it requires simulated data and environment, it is harder to apply to practical business situations.
- However, its learning process is natural to sequential decision-making scenarios, making RL technology undeniably promising.

There are currently two principal methods often used in RL: probability-based Policy Gradients and value-based Q-learning.

I’ve covered many of these topics already, in the following posts:

Part 1: A Brief Introduction to RL

Part 2: Introducing the Markov Process

Part 3: Markov Decision Process (MDP)

Part 4: Optimal Policy Search with MDP

Part 5: Monte-Carlo and Temporal-Difference Learning

Part 6: TD(λ) & Q-learning

Part 7: A Brief Introduction to Deep Q Networks

so stay tuned!

Machine Learning | Reinforcement Learning | AI

Published in AI³ | Theory, Practice, Business

8.1K followers · Last published Nov 17, 2020

The AI revolution is here! Navigate the ever changing industry with our thoughtfully written articles whether your a researcher, engineer, or entrepreneur

Follow

Written by dan lee

436 followers · 5 following

NLP Engineer, Google Developer Expert, AI Specialist in Yodo1

Follow

Responses (2)