Reinforcement Learning Notes

Alexander Powers

01

May 11, 2019

1 What is Reinforcement Learning & Intro to Markov Processes

Topics Covered

- What is RL
- Markov Process
- Markov Reward Process
- Markov Decision Process

1.1 What is RL?

Reinforcement Learning is:

- A sub-field of machine learning that focuses on how a software agent takes actions in an environment to maximize a reward function
- An approach that incorporates the time dimension of ML problems
- A method that falls somewhere between supervised and unsupervised learning in terms

1.2 Markov Process

In a Markov Process we have a system that contains a set space **S** that is finite, and a transition matrix **T**. In order for a system to be a Markov Process, it must statisfy the Markov Property. The Markov Property states that the future system state depends only on the current state, and not any sequence of states. This type of system can be thought of as memoryless. I like to think of these as finite state machines (FSM) where each transition between two states has an associated probability of occuring.

Example Transition Probability Matrix:

Current State	sunny next	rainy next
sunny	0.8	0.2
rainy	0.1	0.9

1.3 Markov Reward Process

A Markov Reward Process builds off of a Markov Process by adding a reward R to every state transition.

The **reward** is just a scalar, that could be positive or negative.

Example Reward Matrix:

Current State	sunny next	rainy next
sunny	0.5	-5
rainy	5	-2.2

The **return**, G, at time t is the sum of subsequent rewards, where subsequent rewards are multiplied by a discount factor γ raised to the power of the index of the summation k.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+(k+1)}$$
$$0 \leq \gamma \leq 1$$

Intuition for gamma:
If gamma is 1, the return is the sum of all future rewards. This would correspond to perfect visibility of all future rewards.
if gamma is 0, the return is only the current reward. This would correspond to no insight into furture rewards.
Conventionally gamma is in between, like 0.9

The **value of state** is the mathematical expectation, or average of lots of possible chains. This provides a less volatile metric for reward at any given state

$$\mathbf{V}(s) = \mathbb{E}[G|S_t = s]$$

1.4 Markov Decision Process

A Markov Decision Process adds an `Action Dimension` to the transition matrix, which allows the agent to modify the probability of transition based on the action selected. The index of the `Action Dimension` is selected by the **policy**, which is the porbabilty of an action being selected given a curent state.A Markov Reward process can be thought of as a special case of and MDP, where the policy function is **fixed**.

The probability of an action occuring given a current state.
Policy Function:

$$\pi(a|s) = P[A_t = a|S_t = s]$$

02

May 11, 2019

1 OpenAI Gym API

Topics Covered

- Components of RL
  - Agent
  - Environment
  - Actions
  - Observations
  - Policy

```
In [1]: import random
```

1.1 The Agent

The **agent** is a person or a thing that takes an active role. The agent is the implementor of the **policy** which decides what action to take at each time step. The agent decides what action to take based on the observation that it recieves from the environment.

```
In [2]: # a niave agent that makes a random choice regardless of observations
        class Agent:
            def __init__(self):
                self.total_reward = 0.0

            def step(self, env):
                current_obs = env.get_observation()
                actions = env.get_actions()
                reward = env.action(random.choice(actions))
                self.total_reward += reward
```

1.2 The Environment

The **environment** is a model of the world external to the **agent**. The environment is responsible for providing the agent with **observations** and **rewards**. The environment state will change depending on the agents actions.
OpenAI Environment class has 2 main attributes `action_space` and `observation_space` as well as two main methods `reset()` and `step()`. Each of the attributes represent their respective spaces. The reset method returns the environment to it's initial state. The step method is the central method of the Environment class and does the following things:

- Takes an input that is the step to be taken and executes it
- Gets new observations after this action
- Gets the reward gained by this step
- Provides and indication that the step is complete

```
In [6]: class Environment:
            def __init__(self):
                self.steps_left = 10

            # observations will change based on agent behavior
            # this informs the agents decisions
            def get_observation(self):
                return [0.0, 0.0, 0.0]

            # action set should likely change based on the agents actions
            def get_actions(self):
                return [0, 1]

            # likely some 'win condition'
            def is_done(self):
                return self.steps_left == 0

            def action(self, action):
                if self.is_done():
                    raise Exception("Game is over")
                self.steps_left -= 1
                return random.random()
```

```
In [9]: # object instantiation
        env = Environment()
        agent = Agent()

        # the NIAVE agent will make random choices for 10 steps
        while not env.is_done():
            agent.step(env)

        print("Total reward: %.4f" % agent.total_reward)
```

```
Total reward: 3.4977
```

OpenAI ships with tons of pre-build environments to test on. A list can be found here.

1.3 Agent Actions

The action space can be either discrete or continous, or a combination of both. `Discrete Action Space` (pushing a button, moving in a grid) only one option is possible at time. `Continous Action Space` (run 9 degrees left, turn a nob 0-1). The environment could also have multiple actions that can be performed simultaneously.

1.4 Observations

Observations are information that the `Environment` provides to the `Agent`. Observations can be as simple as a couple of numbers, or as complex as multiple videos or images.
OpenAI Observation types are as follows: Discrete, Box, Tuple. `Discrete` is a set of mutually exclusive possibilities. `Box` is an n-dimensional tensor. `Tuple` allows us to group together multiple space classes.

1.5 Gym Wrappers & Monitors

Just know that these exist and help extend OpenAI functionality in generic ways. readthedocs.io

03

May 11, 2019

1 Deep Learning w/ PyTorch

```
In [ ]:
```

04

May 11, 2019

1 The Cross-Entropy Method

```
In [ ]:
```

05

May 11, 2019

1 Tabular Learning & the Bellman Equation

In [ ]:

06

May 11, 2019

1 Deep Q Networks (DQN)

07

May 11, 2019

1 DQN Extensions

```
In [ ]:
```

08

May 11, 2019

1 Stock Trading Example

```
In [ ]:
```

09

May 11, 2019

1 Policy Gradients

```
In [ ]:
```

10

May 11, 2019

1 The Actor-Critic Method

```
In [ ]:
```

11

May 11, 2019

1 Asynchronus Advantage Actor-Critic

```
In [ ]:
```

12

May 11, 2019

1 Chat-Bot Example

```
In [ ]:
```

13

May 11, 2019

1 Web Navigation

```
In [ ]:
```

14

May 11, 2019

1 Continuous Action Space

```
In [ ]:
```

15

May 11, 2019

1 Trust Regions

```
In [ ]:
```

16

May 11, 2019

1 Black Box Optimization

```
In [ ]:
```

17

May 11, 2019

1 Beyond Model-Free

```
In [ ]:
```

18

May 11, 2019

1 AlphaGo Zero

```
In [ ]:
```