COMP9414 Artificial Intelligence

Tutorial week 1 – Rule-based Systems

1 Background

1.1 Rule-based Systems

A rule-based system is a type of computer system that leverages domain-specific knowledge in the form of predefined rules. In any field or domain, there exists a body of specialised knowledge that experts use to solve problems or make decisions. This knowledge encompasses facts, relationships, constraints, and patterns relevant to that domain. In a rule-based system, this domain-specific knowledge is captured and represented in the form of rules. For example, in the medical domain, knowledge about symptoms, diseases, and their relationships might be represented.

Rules form the backbone of a rule-based system. These rules are expressed in the form of "if-then" statements, also known as production rules. Each rule consists of two parts: the antecedent (the "if" part) and the consequent (the "then" part). The antecedent specifies the conditions or criteria that must be met for the rule to be applied, while the consequent specifies the action or conclusion to be taken if the conditions are met. Rules encode the logical relationships, decision criteria, or problem-solving strategies relevant to the domain.

Rule-based systems can effectively solve problems, make decisions, or provide recommendations within their designated domain. These systems are particularly useful in domains where expertise can be codified into explicit rules, such as expert systems, diagnostic systems, decision support systems, and natural language processing applications. For instance, a rule-based system might assist a doctor in choosing a diagnosis based on symptoms, or select tactical moves to play a game.

1.2 Mountain Car Environment

For this tutorial, we will use the gymnasium library [1]. This is a well-known library regularly used for reinforcement learning environments. In particular, we will use the Mountain Car [2] environment (see Fig. 1) and create a simple rule-based system to control car movement.

[1] `https://gymnasium.farama.org/`

[2] `https://gymnasium.farama.org/environments/classic_control/mountain_car/`

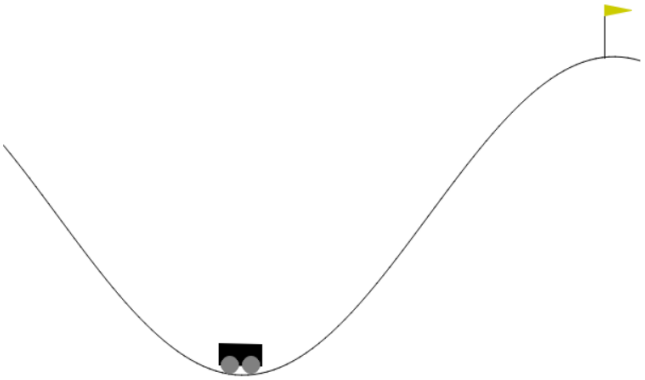

Figure 1: Mountain Car environment

The Mountain Car is a control problem in which a car is located on a unidimensional track between two steep hills. The car starts at a random position at the bottom of the valley ($-0.6 < x < -0.4$) with no velocity ($v = 0$). The aim is to reach the top of the right hill. However, the car engine does not have enough power to claim to the top directly and, therefore, needs to build momentum moving toward the left hill first. An agent controlling the car movements observes two state variables, namely, the position x and the velocity v. The position x varies between -1.2 and 0.6 in the x-axis (with $x \approx -0.53$ the lowest height) and the velocity v between -0.07 and 0.07. The agent can take three different actions: accelerate the car to the left (0), do not accelerate (1), and accelerate the car to the right (2). The task is completed in case the top of the right hill is climbed ($x \geq 0.5$) or if the length of the episode is 200 iterations in which case the episode is forcibly terminated.

Gymnasium environments provide access to the environment action and observation spaces. The following code initialises an environment while visually rendering the output (be aware that rendering the output might not be available on platforms such as Colab). Next, a random action is selected and performed in the environment for a predetermined number of iterations.

```
import gymnasium as gym
env = gym.make("MountainCar-v0", render_mode="human")
observation, info = env.reset()

for _ in range(1000):
    action = env.action_space.sample()
    observation, reward, terminated, truncated, info = env.step(action)
```

```
    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

2 Setup

(a) Using Python, create the Mountain Car environment using Gymnasium, then start resetting the environment. Gymnasium needs `swig` and `box2d` packages, therefore, before creating the environment you should run the following:

```
pip install swig
pip install gymnasium[box2d]
```

(b) Create a loop with an adequate number of iterations to run the simulation.

(c) Select a random action and execute it in the environment. Observe how the car position varies over time.

3 Experiments

(a) Instead of selecting a random action, select accelerating the car to the right at each time step.

(b) As the car needs to build momentum, let's create two simple rules as follows:

i Accelerate the car to the right if it is climbing the hill to the right while increasing the velocity.

ii. Accelerate the car to the left if it is climbing the hill to the left while decreasing the velocity.

(c) Let's expand our knowledge base by adding two more rules, as follows:

i Accelerate the car to the right if it is climbing the hill to the right while increasing the velocity.

ii. Accelerate the car to the left if it is climbing the hill to the left while decreasing the velocity.

iii. Accelerate the car to the right if it is descending the hill to the left while increasing the velocity.

iv. Accelerate the car to the left if it is descending the hill to the right while decreasing the velocity.

(d) For the previous setups, i.e., always accelerating to the right, two rules knowledge base, and four rules knowledge base, run the experiment 100 times and show the results using boxplots.

(e) **Challenge:** is there any better set of rules you can define for the Mountain Car environment?