Artificial Intelligence

Tutorial week 8 – Knowledge representation and uncertainty

COMP3411/9814

1 Background

Knowledge representation and uncertain reasoning are fundamental areas of AI that focus on how knowledge about the world can be formally represented and used by a machine to solve complex tasks. This discipline intertwines the cognitive aspects of how humans understand and process information with the computational methods necessary for enabling machines to exhibit intelligent behaviour.

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 Belief networks

A Bayesian or belief network is a probabilistic graphical model used to represent and analyze the dependencies between variables. In this network, nodes represent random variables, and directed edges between nodes indicate conditional dependencies. The strength of these dependencies is captured by conditional probability tables associated with each node.

Bayes Nets are widely used in various fields, including artificial intelligence, machine learning, statistics, and expert systems. They enable reasoning and inference by calculating probabilities based on evidence and updating beliefs in a systematic way using Bayesian probability principles.

1.3 Fuzzy logic

Fuzzy logic is a form of logic that allows for reasoning and decision-making in situations that involve ambiguity, imprecision, or uncertainty. Unlike classical binary logic, which deals with crisp true/false values, fuzzy logic allows for partial truths, represented by values between 0 and 1, which are referred to as degrees of membership.

The framework operates based on fuzzy rules, which use fuzzy information to model relationships between inputs and outputs. These rules are expressed in the form of if-then statements. Fuzzy logic provides a flexible and intuitive method for dealing with uncertainty and imprecision, allowing machines to emulate human-like decision-making and problem-solving processes.

2 Rule-based system

For the first part of the 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.

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

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

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

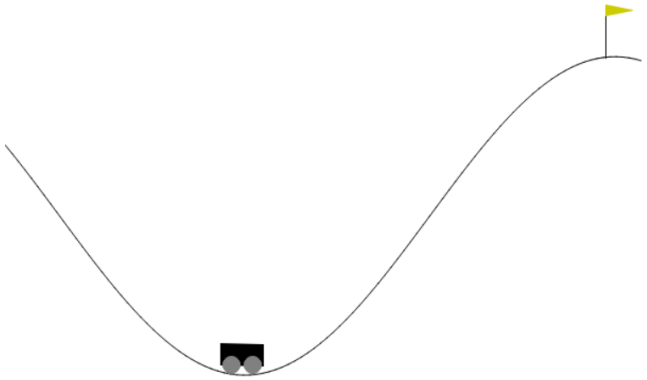

Figure 1: Mountain Car environment

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()
```

2.1 Exercise:

(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.

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

(e) 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.

(f) 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.

(g) 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.

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

3 Belief network

Consider the problem domain reviewed during the lecture when I go home, and I want to know if someone from my family is home before I go in. Let's say I know the following information:

1. When my wife leaves the house, she often (but not always) turns on the outside light.

2. When nobody is home, the dog is often left outside.

3. If the dog has bowel troubles, it is also often left outside.

4. If the dog is outside, I will probably hear it barking (though it might not bark, or I might hear a different dog barking and think it's my dog).

Given the previous information, we can consider the following five Boolean random variables:

1. family-out (***fo***): everyone is out of the house.

2. light-on (***lo***): the light is on.

3. dog-out (***do***): the dog is outside.

4. bowel-problem (***bp***): the dog has bowel troubles.

5. hear-bark (***hb***): I can hear the dog barking.

From this information, the following direct causal influences are appropriate:

1. ***hb*** is only directly influenced by ***do***. Hence ***hb*** is conditionally independent of ***lo***, ***fo*** and ***bp*** given ***do***.

2. ***do*** is only directly influenced by ***fo*** and ***bp***. Hence ***do*** is conditionally independent of ***lo*** given ***fo*** and ***bp***.

3. ***lo*** is only directly influenced by ***fo***. Hence ***lo*** is conditionally independent of ***do***, ***hb*** and ***bp*** given ***fo***.

4. ***fo*** and ***bp*** are independent.

The belief network representing these direct causal relationships (though these causal connections are not absolute, i.e., they are not implications) is shown in Fig. 2. The network includes the prior probability of the random variable for root nodes ***fo*** and ***bp*** as well as the conditional probabilities of the node's variable given all possible combinations of its immediate parent nodes need to be determined for non-root nodes.

3.1 Exercise:

1. Using `pmgpy`, create the structure of the model considering the four edges as shown in Fig. 2.

2. Create the root nodes ***fo*** and ***bp*** and assign them the respective prior probability.

3. Create the non-root nodes ***lo***, ***do***, and ***hb***. For ***lo*** and ***hb*** consider four combinations taking into account one immediate parent. For ***do*** consider eight combinations as this node has two immediate parents.

4. Create the Bayesian network model.

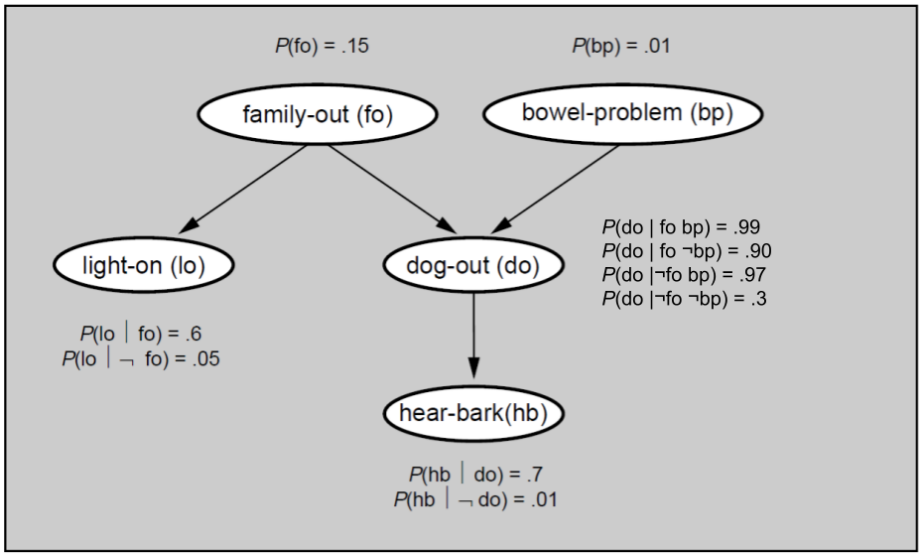

Figure 2: Belief network for the family out problem.

5. Check the conditional probability distributions for all variables.
6. Test the Bayesian network model for different situations.
   - Test 1: Compute the joint probability P(~FO,BP,~LO,DO,HB).
   - Test 2: Compute a causal inference (top-down) using the conditional probability P(DO|BP).
   - Test 3: Compute a diagnostic inference (bottom-up) using the conditional probability P(BP| DO).

4 Fuzzy logic controller

In the last experience, we aim to build a fuzzy logic controller to regulate the engine's fuel injection using two antecedent variables: speed and temperature. The domain of each variable is as follows:

- **Speed:** from 0 to 150.
- **Temperature:** from 0 to 150.
- **Injection:** from 0 to 100.

Both input variables use fuzzy values for three ranges, as shown next (see Fig. 3):

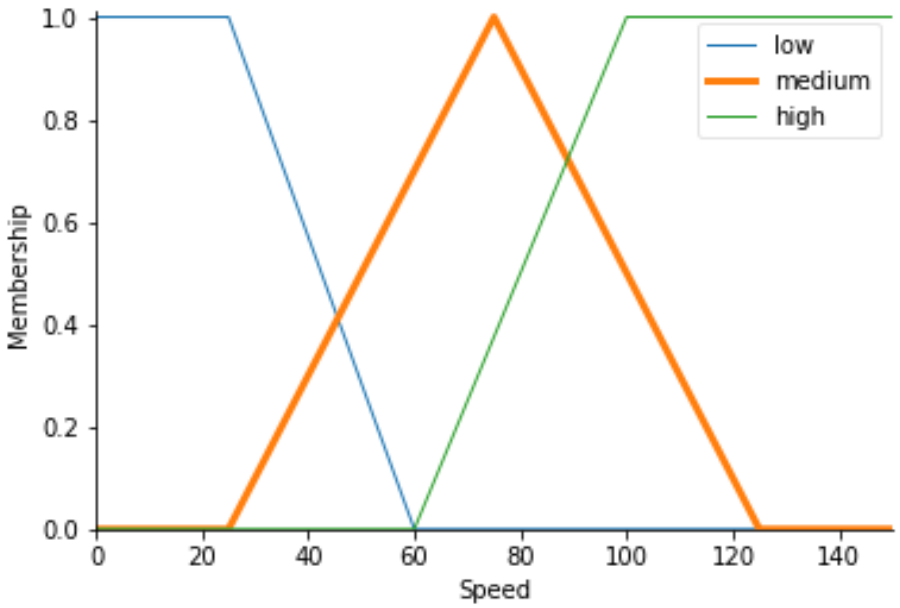

Figure 3: Membership function for speed (temperature variable uses the same membership function).

- **Low speed:** A trapezoid membership function equal to one between 0 and 25, and decreasing to zero between 25 and 60.
- **Medium speed:** A triangular membership function increasing from zero to one between 25 and 75, and decreasing from one to zero between 75 and 125.
- **High speed:** A trapezoid membership function increasing from zero to one between 60 and 100, and equal to one between 100 and 150.

The temperature variable has the same values for its membership function, i.e., low temperature, medium temperature, and high temperature. The output variable uses fuzzy values for three ranges as well, this is shown in Fig. 4 and described next:

- **Low injection:** A triangular membership function decreasing from one to zero between 0 and 30.
- **Medium injection:** A triangular membership function increasing from zero to one between 10 and 50, and decreasing from one to zero between 50 and 90.
- **High injection:** A triangular membership function increasing from zero to one between 70 and 100.

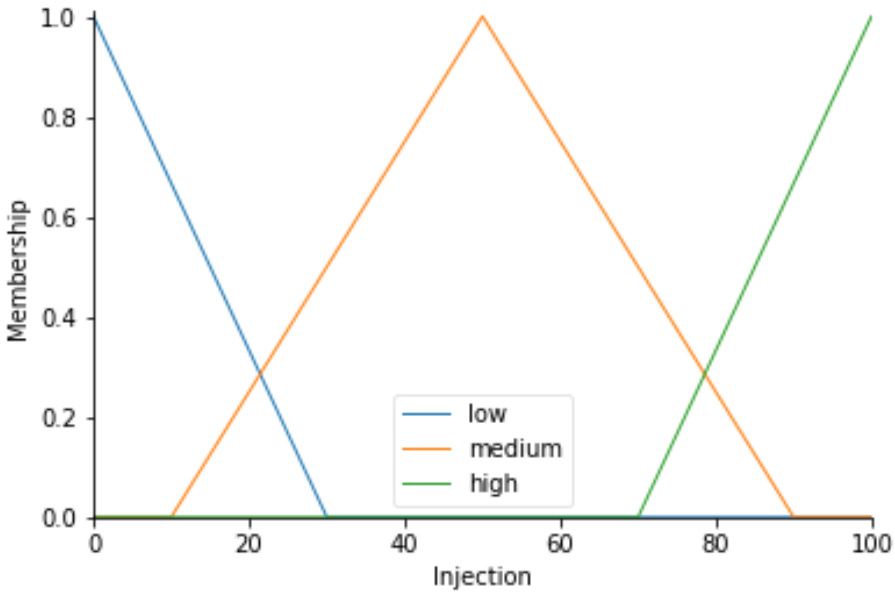

Figure 4: Membership function for output variable injection.

The fuzzy logic controller will implement the following rules:

- IF speed = *medium* AND temperature = *high* THEN injection = *low*.
- IF speed = *low* AND temperature = *low* THEN injection = *high*.
- IF speed = *high* AND temperature = *medium* THEN injection = *low*.

4.1 Exercise:

1. Using `skfuzzy` choose and create the antecedents and consequent for the fuzzy logic controller. For the consequent, use the defuzzification method centroid.
2. Build the membership function for each variable, i.e., speed, temperature, and injection.
3. Visualise the fuzzy sets for each variable.
4. Create the rules implemented by the controller.
5. Using the rules, create the controller and a control simulator.
6. Test the controller with a couple of speeds and temperatures and visualise the output.

- Test 1: speed = 50, temperature = 10.
- Test 2: speed = 70, temperature = 100.

7. Add the following rules to have a full definition of the controller:
   - IF speed = *low* AND temperature = *medium* THEN injection = *high*.
   - IF speed = *low* AND temperature = *high* THEN injection = *medium*.
   - IF speed = *medium* AND temperature = *low* THEN injection = *high*.
   - IF speed = *medium* AND temperature = *medium* THEN injection = *medium*.
   - IF speed = *high* AND temperature = *low* THEN injection = *medium*.
   - IF speed = *high* AND temperature = *high* THEN injection = *low*.
8. Plot the surface of the controller for both input variables: speed and temperature.