1 Introduction

1.1 The Story So Far

So far, you have seen some simple NLP (natural language processing) in the context of neural networks and predicting words. You have seen learning, in the sense of “learn-by-example” - training over many input-output pairs. You have also learned how to install ROS and run a small system of connected nodes, publishing and subscribing information to ROS topics.

1.2 This Project

In this project, we will be looking at NLP and learning from another angle, and combining it with ROS. The type of system we will be building is sometimes called an “advice-taker”, because of the way it learns by being told explicit facts rather than by training on data.

2 Your Task

Building on the Input-Reasoner-Output structure from miniproject 4, design a system that is able to have a dialogue with the user. The system should be able to learn facts about simple relationships between objects and classes, and then answer questions about what it knows.

2.1 Setting up the ROS Infrastructure

If you have completed project 4 successfully, you can just start from your project 4 code. On Friday, we will release a canonical implementation of project 4, which you can use as a starting point if you don’t want to use your own.

This project will follow the same design pattern of the input, output, and reasoning all being handled in separate nodes, with the information flow passing from input to reasoner to output. Most of the work will be handled in the reasoner. The motivation for this modular design is that, in the future, each
individual component can be upgraded without impacting the behavior of the other components. For example, we could make language input interface with a microphone, and run the sound through a speech-to-text program before passing it off to the reasoner. (For this project, we’ll just stick with text I/O though.)

2.2 Learning

The first type of fact your system should be able to learn is an object-class relationship. An input sentence from the user for this type of fact will have this form:

“<Name> is <a/an> <class>.”

An example might be “Bob is a person.” Your system should associate the individual Bob with the class person. If the system hasn’t learned any facts about Bob yet, you’ll of course have to create some means of representing him, since you can’t pre-load all possible individuals.

The second type of fact your system should be able to learn is a class-class relationship. An input sentence from the user for this type of fact will have this form:

“<A/An> <class> is <a/an> <class>.”

An example might be “A dog is a mammal.” These sentences are distinguished from the object-class sentences because of the presence of the article a/an - this should be the key your system picks up on to determine how to store the fact.

For both of these types of facts, the names of the objects and classes may consist of more than one word. For example, “New York City is a city.”, or “A painting is a work of art.”. Names will not contain the words ‘a’, ‘an’, or ‘is’, but anything else is fair game. Be sure to handle this appropriately.

Multi-Classing

Sometimes it is convenient to think about classes as being in a strict hierarchy: all dogs are mammals, all mammals are vertebrates, all vertebrates are living things, etc.. However, in the real world, this is not the case - objects often belong to two or more intersecting classes, such that neither contains the other. For example, Bob may be both a teacher and a researcher, but there are teachers who don’t do research and researchers who don’t teach. This distinction even extends to class-class relationships: a dog is a mammal, and a dog is also a pet.

Your system will need to be able to handle this feature, of an object or class being more than one class. So, you can’t just use a simple lookup table, where each object/class is matched to exactly one class it is contained in (although you may be able to just slightly modify that approach, and match each object/class to a set or list of classes).
2.3 Question Answering

Once your system has some facts, it should be able to answer questions based on those facts. All questions will come in the same two forms as the assertions above, but they will start with the word ‘Is’ and end in a ‘?’.

For example, the user might ask “Is a dog a mammal?”, to which your system should respond either “Yes” or “I don’t know”, depending on what it knows about dogs and mammals. The user might also ask about an individual, e.g. “Is Alice a robot?”, to which your system should again respond with either “Yes” or “I don’t know”.

Transitivity

‘is a’ is a transitive relation - “x is a y” and “a y is a z” implies that “x is a z”. Your system must also be able to handle this. For example, if your system knows that Bob is a person, and a person is a living thing, it should be able to conclude that Bob is a living thing. You don’t need to store all these derived facts explicitly - just make sure you are able to produce them when asked. To accomplish this, you may find it useful to store your information in a tree-like structure (you can’t just say “keep moving up in the hierarchy”, because of multi-classing), and perform some sort of traversal when a question comes in.

2.4 What you can ignore

- You may assume that the user input will adhere to the specified formats.

- You do not have to take into account plurals (that’s why we use “A dog is a mammal” rather than “Dogs are mammals”, because then you have to know how to pluralize words; note that you could in theory have a system learn these pluralization rules as well, although we won’t be doing that for this project).

- You do not have to handle negative info (e.g. “A dog is not a fish.”).

- You do not have to deal with adjectives. For example, “A computer is an electrical machine.” should just relate “computer” to a class “electrical machine”, treated the same way as you would treat any multi-word object or class - it shouldn’t connect this to the class “machine”, unless it was explicitly told that “An electrical machine is a machine.” (or could infer this fact through transitivity).
3 A Sample Run

USER: Rover is a dog.
USER: Is Rover a dog?
ROBOT: Yes
USER: Is Rover a cat?
ROBOT: I don't know
USER: A dog is a mammal.
USER: Is a dog a mammal?
ROBOT: Yes
USER: Is Rover a mammal?
ROBOT: Yes
USER: Is a cat a mammal?
ROBOT: I don't know
USER: Are dogs mammals?
ROBOT: I don't know
USER: New York City is a city.
USER: Is New York City a city?
ROBOT: Yes
USER: Is New York a city?
ROBOT: I don't know
USER: New York City is a big thing.
USER: Is New York City a big thing?
ROBOT: Yes
USER: Is New York City a big city?
ROBOT: I don't know
USER: A computer is an electrical machine.
USER: Is a computer an electrical machine?
ROBOT: Yes
USER: Is a computer a machine?
ROBOT: I don't know

4 What we have provided

After the deadline for project 4 has passed, we will provide you with a canonical implementation of that project to serve as a starting point, if you are having trouble.

5 What to Turn In

Submit your three ROS nodes (the three .py files) on ELMS.