Notes for AI

Peter Heilbo Ratgen

8. februar 2021

1 1st February - Introduction

1.1 Basics

The course is an introduction to the basics of Artificial Intelligence. We will get an overview of the base of the artificial intelligence methods. We will use python as a programming language. Labs and support will be done in python. Prerequisite to the exam is to complete the homework of the lectures.

You should help each other, but coding should be done individually.

- Thinking humanly
- Acting humanly

  The Turing test is used to test this. The longer a human can be fooled into thing that the human is talking to a human, and not a bot. This could be chatbots acting humanly. You have to test for:

  - Natural language processing
  - Knowledge representation
  - Automaed reasoning
  - Machine learning

  Success depends on deception. Chatbot can use cheap tricks. Mitzuku has recently won for the 5th time. Computers have a had time with multiple choice questions. Eg the "The large ball crashed right through the table because it was made of styrofoam". If you replace "styrofoam"with "steel", then the answer is totally different.

  A better test? A better Turing test, would be one that can be administed and graded by a machine, and more objectivity in the it would not depend of subjectity of humans.

- Thinking rationally

  It is about the idealized or "right"way of thinking. It is hard to describe the world using logical notation. The procedure of applying these local statements and deducing them. We also have a though time dealing with uncertainty, representing the gray areas.

- Acting rationally

  Acting rationally is acting with the goal of achieving the goal, that has been set. Utility is about the goal that has been set, whether it is about, shortest route, least time or fewest changes in a public transit system.

1.2 Successes of AI

- IBM Watson is an AI created by IBM. IBM is one of the companies that has been investing in AI for the longest times.
- Self driving cars is one of the successes of AI. This is an example of a rational acting.
- Natural language processing is also a great improvement, with speech technologies and machine translation.
- Vision, OCR, handwriting recognition and face detection and recognition.

- Mathematics, program solved unsolved conjecture. Also wolfram alpha.
- Games

  Chess(champion beaten in 1997), checkers(solved in 2007), Go (beaten for the first time by a Google AI), Google AI beat top StarCraft players.
- Logistics, scheduling, planning.

  A lot of the advancement are done by the military. In the 1991 Gulf War an AI planned and scheduled for 50,000 vehicles and such.
- Robotics

  Mars rovers, self driving cars, drones, robot soccer, personal robotics.

1.3 History

First model of a neuron was in the 1940's. In the 1950's the turing test was created, computer chess and machine translation and theorem provers. In 1956 the 'Artificial Intelligence' term was adopted. Herbert Simon said in 1957 that a computer would beat a machine and a computer would prove a theorem, this happened 40 years later instead of 10 years. They realized the problem of machine translation and chatbots was harder to solve than initially thought. In the late 1960's machine translation was deemed a failure. Intractability is recognized as fundamental problem. When the complexity or size of the problem grows exponentially. There was a boom in expert system in the 1980's and subsequently a bust. Deep learning, big data and probabilistic learning boomed in the 1990's till now.

The successes now is due to better computers, dominance of statistical approaches and machine learning, big data and crowd sourcing.

2 8th of February - Introduction to Python

Exercises will start at 10:20.

Basics Python code is fairly simple and readable. Beginning and ending of blocks is done purely by indentation. We will use the Python Console for trying out examples. Variables can change types throughout the program. A variable can start as a string end as float. We can use `+` for string concatenation. We can use triples quotes for strings containing both `'` and `"`.

Variables in Python do not have intrinsic types. But assignment does not create copies, but references. References are deleted by the garbage collector, when the reference has passed out of scope. Names cannot start with numbers.

We can have multiple assignments. And swapping vars is easy.

```
>>> x, y = 2, 3
>>> y
3
>>> x, y = y, x
>>> y
2
>>> x
3
```

Sequence types Sequence types are tuples, strings and lists. In a tuple we can have multiple types of variables. Tuples are immutable, such that they cannot be changed after it has been created. Strings are also immutable. Lists are mutable, they can also have mixed types. These sequence types have much syntax in common. If we have to change elements in an immutable tuple or string a new copy has to be created. Lists can be shrunk or expanded as you go. We assign some different variables:

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu
(23, 'abc', 4.56, (2, 3), 'def')
>>> li = ['abc', 34, 4.34, 23]
```

```
>>> st = "Hello␣World"
>>> st
'Hello␣World'
>>> st = Helllo wordl'
␣␣File␣"<stdin>",␣line␣1
␣␣␣␣st␣=␣Helllo␣wordl'
                  ^
SyntaxError: invalid syntax
>>> st = 'Helllo␣wordl'
>>> st
'Helllo␣wordl'
>>> st = """THis is a multiple line
... string that uses triple quotes"""
>>> st
'THis␣is␣a␣multiple␣line\nstring␣that␣uses␣triple␣quotes'
```

We can also have negative indexes, such that -1 is the last character:

```
>>> st[-1]
's'
>>> st[-2]
'e'
```

If want to get the 3 middle elements of the tuple we defined:

```
>>> tu[1:4]
('abc', 4.56, (2, 3))
```

From this we get a copy of the selected part of the tuple. If we do not specify where in the tuple to start, we start from the beginning.

```
>>> tu[:3]
(23, 'abc', 4.56)
```

We can do a copy like this:

```
>>> tu[:]
(23, 'abc', 4.56, (2, 3), 'def')
```

In this example there is a big difference between line 3 and 4.

```
>>> l3 = ['4', '5']
>>> l4 = ['6', '7']
>>> l3 = l4
>>> l3 = l4[:]
```

In line 3 we assign `l3` we assign to the reference to `l4`. In the 4th line we assign `l3` to a copy of `l4`, such that changes in `l4` will not be reflected in `l3`.

We can use the `in` operator to check if we have a substring. We can concat tuples:

```
>>> tu[:] + tu2[:]
(23, 'abc', 4.56, (2, 3), 'def', 12, 'yeet')
>>> tu + tu2
(23, 'abc', 4.56, (2, 3), 'def', 12, 'yeet')
```

Dictionaries Dictionaries can store a mapping between a set of keys and values. We can create a dictionary with:

```
>>> d = {'user' : 'bozo', 'pswd' : 1234}
```

The values can be anything. We can get the value of a key like this:

```
>>> d['user']
'bozo'
```

We can delete a key:value pair like this:

```
>>> del d['pswd']
>>> d
{'user': 'bozo'}
```