# Notes for AI

Peter Heilbo Ratgen

8. april 2021

## Indhold

# 1 Week 5 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.

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'}
```

# 2 Week 7 - Intelligent Agents

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. We can say that an agents behaviour is described by its **agent function**. This maps any sequence to an action. This is a mathematical abstraction, the *actual program* is called the **agent program**

We can have many types of agents, these can perceive in many ways: here are some examples.

- Human agent

  Eyes, ears, nose

- Robotic agent

  Cameras, and infrared range finders

- Software agent

  Keystrokes, file contents & network packets.

## 2.1 Vaccum-cleaner world

A vecuum-cleaner has two locations to take care of: A and B. It has four actions: left, right, such and NoOp. We can have a simple programme for cleaning with this vaccum:

```
if status = Dirty then return Suck
else if Location = A then return Right
else if Location = B then return Left
```

The vacuum-cleaner is a rational agent, it tries to optimize on a performance measure. The choice of performance measure is a critical one. "Doing the right thing"is to always act according to the performance measure. We also call the performace measure the utility function, it is an objetive criteria for the success of an agent's behaviour.

An example of potential performance measures for the vacuum agent.

- amount of dirt cleaned up
- amount of time taken,
- amount of electricity consumed
- amount of noise generated

It would be desirable to have the agent take as little time as possible, but if we only consider this, then it would stand still. But if combined, with eg the amount of dirt cleaned up, then we could have beter operation.

## 2.2 Autonomy

An autonomous agnet always has the ability to learn and adapt, it can always say "no", it also needs enough built-in knowledge to survive.

**Task Environment Specification** Problem specification:

- Performace meassure
- Environment
- Actuators
- Sensors

For a autonomous taxi

- Performace meassure

  Safe, fastm, legal

- Environment

  Roads

- Actuators

  Steering wheel

- Sensors

  Cameras, speedometer

For an email spam filter:

- Performace meassure

  Manimizing false positives

- Environment

  User email account

- Actuators

  Mark as spam, let mail through

- Sensors

  Incoming messages

## 2.3 Environment types

**Fully observable vs partially observable** In a games, such as FIFA or any game, it is possible to oberserve everything, and perceive all options. If a robot is playing football in the real world, then the perception of the environment is constrained by the input of the sensors eg eyes (you do not know what happens behind your back).

**Deterministic vs stochastic** A deterministic environment is when the coming events in time is only defined by the current state, and the agent's future actions. Versus when the game is dependent on some type of randomness, whether it be a deck of cards or dice. This is very hard to predict.

**Episodic vs sequential** Is every step of the agent independent? Or does it depend on a previous sequence? A sequence could be a game, where current decisions depend on a previous sequence of decisions. In a spam filter the sequence of spam mail does not matter. This is a matter of defintion, it is how you see your environment in terms of your decisions. A game could be episodic, if you choose to look upon it as a snapshot.

**Static vs dynamic** Is the world changing while the agent is thinking. Solving a rubrics cube or chess is static. However self-driving is very dynamic, here things are changing all the time. This a very challenging

**Discrete vs continuous** Does the environment provid a fixed number of distinct number of states(can they be enumerated)? Where time is a factor, then time is always continuous, if you do not choose to look at time in buckets of seconds or minutes.

**Single-agent vs multiagent** Is the agent operating by itself?

## 2.4 Hierarchy of agent types

- Table-driven agents, it's just like a big lookup table.

  The table sizes can become huge. Designing such a table is challenging. We need to think about every single case.

- Simple reflex agent

  This is the vacuum cleaner. For every state given the rules, we find the rule applying to the current state. It selects its action from the current percept only. Implemented through condition-action rules. Example of if-then algorithms. For this to work, then the environment must be fully observable. This wont then work for a self-driving car.

- Agents with memory, internal state to keep track of past states of the world

  We can call this a model-based reflex agent. Internal state: aspects of the environment that cannot be currently observed. This is useful for partially observable environments. In addition to the rule we have a model, a desripton of how the next state depends on current state and action. We also have a state, a description of the current world state the action, is the most recent action.

  We update the state, according to the state, action, percept and the model. When matching a rule, the we take into account the state and the rules.

- Agents with goals

- Utility

# 3 Week 8 - Solving Problems Through Search

Many problems can be solved through search. Which sequence of actions can get us to the goal state? This can be solved through search. We might also have a performance measure of minimizing time. For an example a GPS is a search problem, which sequence of places gets us to the goal?

A search algorithm will tell us the exact sequence to get us from the start to the goal. Search strategies are important methods for many approaches to problem-solving. Search algorithms is a basis for many optimization and planning methods.

- We need to formulate appropriate problem as search taske

  We need to think about states, initial state, goal state, successor functions (operators), cost. It is about creating a microworld, where the computer understands what is going on.

- Know the fundamental seach strategies and algorithms.
  - Uninformed search

    breadth-fist, depth-first, uniform-cost, iterative deepening
  - Informed search

    best-first (greedy, A*)

- Evaluate the suitabilitu of a search strategy for a problem

  Compoleteness, optimality

We will cosider the poroblem of desinging goal-based agen in observale, deterministic, discrete, known environments.

How we solve a comic-book maze.

- solution fixed sequence of actions
- Search: process of looking for the sequence of of actions that reaches the goal
- Agent can ignore percepts during execution.

**Formalizing the maze problem**

- Initial state

  Entry
- Successor Function
- Goal state
- Path cost

  Assume that it is a sum of nonnegative *step costs*.
- Optimal solution: sequence of actions with the lowest path cost.

**Vacation in Romania:**

- Initial state

  Arad
- Successor function

**State Space** Is the inital state and the succesor function defines the state space of the problem.

- The set of all states reachable from initial state by any sequence of actions.
- Can represented as a directed graph (nodes are states and linked between nodes are actions).

The state space for vaccum world.

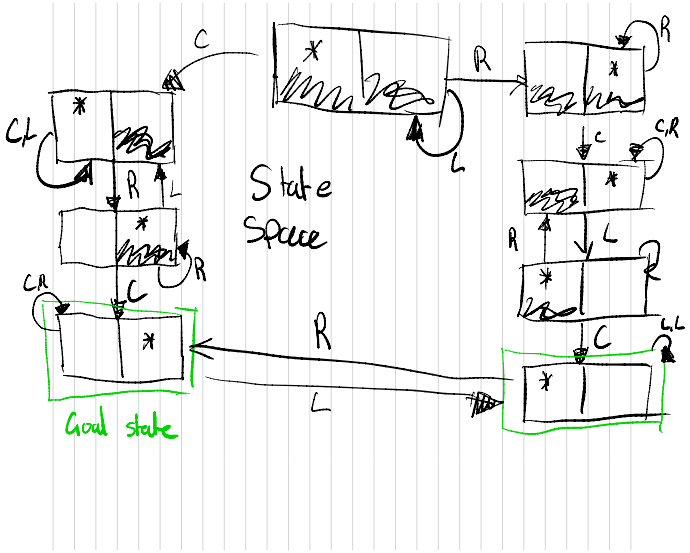
- Initial state

  Dirty

- Goal state

  All clean

- State space.




- Path cost

  Could be the sum of the amouth of electricity used

**Example: The 8-puzzle** The state space of the 8-puzzle has a size of 181440 states. The succesor function is the actions of move blank left, right, up, down.

To build a goal-based agent, we must answer the questions:

- How to we represent the state of the world.
- What is the goal and how caan we recongize it?

How to we define the **Remove 5 sticks** problem?

## 3.1 Search Strategies

**Breadth-first** The fringe is explored in a first in first out fashion.

# 4 Week 9 - Informed Search

# 5 Week 10 - Local Search

We can formulate problems in terms of optimization. We do not care about the path to a solution. We have an objective function that us about the quailt of a possible solution, and we want to find a good solution by minimizing or maximizing the value of the function.

There is three approaches to local search.

## 5.1 Hill-climbing (greedy) search

Here the crux is: keep a single "current state"and try to locally improve it. You can like it to "climbing mount Everest in thick fog with amnesia".

If we look at a puzzle, it should be 1, 2, 3, 4, 5, 6, 7, 8 going in a clockwise circle our function is

$$f(n) = -(\text{number of tiles out of place})$$

. We want to find the best solution, the one closest to what we think we want.

In each iteration we probe, for which move increases our evaluation function. It can get stuck in a local optimum.

## 5.2 Simulated annealing search

The hill-climbing will be never go down the hill of global and local maxima. The idea is to escape local maxiam by allowing some "bad"moved but gradually decrease the frequency of these bad moves.

- Probability of takeing downhill move decreases with number of iterations, steepness of downhill move
- Controlled by annealing schedule

Inspired by annealing process, as part of the tempering of glass, metal.

# 6 Week 11 - Adversarial Search

# 7 Week 12 - Constraint Satisfaction Problems

# 8 Week 15 - Probability

# 9 Week 16 - Bayesian Networks

# 10 Week 17 - Hidden Markov Models

# 11 Week 18 - Intro to Machine Learning

# 12 Lab - Week 7

## 12.1 TABEL-DRIVEN-AGENT

**Run the module** The program prints:

**The percept**

# 13 Lab - Week 8

1. Successor nodes are inserted at front of the fringe (successor list) as a node is expanded. Is this a breadth (FIFO) or depth-first search (LIFO)?

   This is depth-first search. We put the successor node in front, and examine these first.

2. For goal J, give the fringe (successor list) after expanding each node.

   The fringe list, should be empty, when we have found the goal J. However if we just expand the tree, going depth-first with LIFO then we would have:

   J I H G F C E D B A

3. What is the effect of inserting successor nodes at the end of the fringe as a node is expanded? A depth or breadth-first search?

   The effect of this is a breadth-first search.

4. For goal J, give the fringe (successor list) after expanding each node

   A, B, C, D, E, F , G, H, I, J

# 14 Lab - Week 9