

Foundations of AI

Acting under Uncertainty

Maximising the expected utility

Luc De Raedt

1

Overview

- Introduction to Utility Theory
- Selecting single Actions
- Sequential Decision Problems
- Markov Decision Processes
- Value iteration

2

Utility Theory

The (*Utility function*) assigns values to states. It therefore formalizes the agent's preferences among states.

$U(S)$ Utility of state S for the agent.

A non-deterministic action A can result in any of the states $Result_t(A)$. The probability that the successor state $Result_t(A)$ is reached when A is executed given evidence E is

$$\rightsquigarrow P(Result_t(A) | Do(A), E)$$

Expected Utility

$$EU(A | E) = \sum_i P(Result_t(A) | Do(A), E) \times U(Result_t(A))$$

The *principle of maximisation of expected utility* *MEU* states that a rational agent selects the action that maximizes $EU(A | E)$.

3

Problems with the MEU-principle

$$P(Result_t(A) | Do(A), E)$$

requires a complete causal model of the world.

↪ should be modified for each change

↪ NP-complete for Bayesian nets.

$$U(Result_t(A))$$

requires one to search or plan to estimate the utility of a state (looking ahead).

4

Axioms of Utility Theory (1)

Justification of the *MEU*-principle.

Scenario = *Lottery L*

- possible outcomes = possible wins
- the result is determined at random
- $L[1, A] = \text{State } A$

Example :

Lottery L with 2 outcomes A and B :

$$L = [p, A ; 1 - p, B]$$

Preference among lotteries :

$L_1 \succ L_2$ Agent prefers L_1 over L_2

$L_1 \sim L_2$ Agent is indifferent between L_1 and L_2

$L_1 \succsim L_2$ L_1 is preferred over or indifferent to L_2

5

Axioms of Utility Theory (2)

Given states A, B, C

- **Orderability**

$$(A \succ B) \vee (B \succ A) \vee (A \sim B)$$

An agent must know what he wants.

- **Transitivity**

$$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$$

Not being transitivity causes irrational behaviour. $A \succ B \succ C \succ A$. Agent has A and would pay to change to C . C could then be changed to A again.

\rightsquigarrow Agent loses money.

6

Axioms of Utility Theory (3)

- **Continuity**

$$A \succ B \succ C \Rightarrow \exists p [p, A ; 1 - p, C] \sim B$$

With fixed preferences among A and B , one can construct a lottery so that the agent is indifferent.

- **Substitutability**

$$A \sim B \Rightarrow [p, A ; 1 - p, C] \sim [p, B ; 1 - p, C]$$

Simple lotteries can be replaced by more complicated ones without changing indifference.

7

Axioms of Utility Theory (4)

- **Monotonicity**

$$A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A ; 1 - p, B] \succsim [q, A ; 1 - q, B])$$

If the Agent prefers A , then he must also prefer the lottery with higher probability assigned to A .

- **Decomposability**

$$[p, A ; 1 - p, [q, B ; 1 - q, C]] \sim$$

$$[p, A ; (1 - p)q, B ; (1 - p)(1 - q), C]$$

There is no fun in gambling.

8

Utility function and axioms

The axioms are concerned with preference only.

However, they imply that there exists a utility function.

1. Utility principle

If the preferences by the agent satisfy the axioms, then there exists a function $U : S \rightarrow R$ such that

$$U(A) > U(B) \Leftrightarrow A \succ B$$

$$U(A) = U(B) \Leftrightarrow A \sim B$$

2. Maximum expected utility principle

$$U([p_1, S_1; \dots; p_n, S_n]) = \sum_i p_i \times U(S_i)$$

How to design utility functions that lead to the desired behaviour of the agent?

9

Sequential Decision Problems

3				$+1$
2				-1
1	START			
	1	2	3	4

The agent should get into state (4, 3) (Reward +1) and avoid state (4, 2) (Reward/Punishment -1). Actions : north, south, west, east.

deterministic variant: all actions always lead to the next box in the choosen direction. Hitting the wall has no effect.

stochastic variant: intended effect has probability of 0.8, with probability 0.2 the agent moves at right angles.

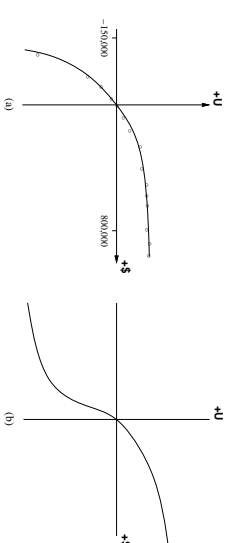
Example:

$$P((1, 2) | Do(north, (1, 1))) = 0.8 \ \& \ P((2, 1) | Do(north, (1, 1))) = P((1, 1) | Do(north, (1, 1))) = 0.1$$

11

Possible Utility Functions

The utility of money :



Scaling and normalisation

- best price $U(u_{\top}) = 1$
- worst case $U(u_{\perp}) = 0$

Intermediate values are obtained by varying the probability p in a standard lottery and comparison with the resulting state S , until the agent is indifferent between S and the lottery.

10

Markov Decision Problems (MDP)

Given :

- Set of states in an accessible stochastic environment
- Set of goal states
- Set of actions
- Transitionsmodel M_{ij}^a
- Utility function

Transitionsmodel: M_{ij}^a is the probability that state j is reached when action a is executed in state i .

Policy: Function from states to action. Specifies the action to execute in any given state.

Find: Optimal Policy, i.e. policy that maximizes the expected utility.

12

MDP-based Agent

```

function SIMPLE-POLICY-AGENT(percept) returns an action
static: M, a transition model
         U, a utility function on environment histories
         P, a policy, initially unknown
if P is unknown then P ← the optimal policy given U, M
return P[percept]
    
```

Example :

Utility of an action sequence

= Value of the final state – Number of actions / 25

With 6 actions to final state +1: $1 - \frac{6}{25} = 0.76$

13

Value iteration (2)

The utility of a state i is determined by the expected utility of the optimal policy under the given transition model M :

$$\begin{aligned}
 U(i) &= EU(H(i, policy^*) | M) \\
 &= \sum P(H(i, policy^*) | M) \cdot U_h(H(i, policy^*))
 \end{aligned}$$

Because of the additivity of the utility function, one can obtain

$$\begin{aligned}
 policy^*(i) &= \underset{a}{\operatorname{argmax}} \sum_j M_{ij}^a U(j) \\
 U(i) &= R(i) + \max_a \sum_j M_{ij}^a U(j)
 \end{aligned}$$

↪ Basis for dynamic programming

15

Value iteration (1)

An algorithm to compute an optimal policy.

Idea: compute the utility of each state. This can then be used to determine the optimal action for each state.

An action sequence generates a tree of possible successor states (*Histories*).

A utility function U_h on histories is *separable* iff there is a function f so that

$$U_h([s_0, s_1, \dots, s_n]) = f(s_0, U_h([s_1, \dots, s_n]))$$

The simplest form employs additive utility functions (*Rewards*)

$$U_h([s_0, s_1, \dots, s_n]) = R(s_0) + U_h([s_1, \dots, s_n])$$

In the example, $R((4, 3)) = +1$, $R((4, 2)) = -1$, $R(\text{other}) = -\frac{1}{25}$.

14

Value iteration (3)

If the utility of the final states is known, then an n -step decision problem can be reduced to that of computing the utility of final states of $(n - 1)$ -step decision problems.

↪ **Iterative and efficient procedure**

Problem: Typical problems contain cycles, so that Histories may be infinite long.

Solution: Application of

$$U_{t+1}(i) \leftarrow R(i) + \max_a \sum_j M_{ij}^a U_t(j)$$

where $U_t(i)$ is the utility of the i -th state after t iterations

Observation: Convergence when $t \rightarrow \infty$.

16

Value iteration (4)

```

function VALUE-ITERATION( $M, R$ ) returns a utility function
  inputs:  $M$ , a transition model
          $R$ , a reward function on states
  local variables:  $U$ , utility function, initially identical to  $R$ 
                  $U'$ , utility function, initially identical to  $R$ 
  repeat
     $U \leftarrow U'$ 
    for each state  $i$  do
       $U'[i] \leftarrow R[i] + \max_a \sum_j M_{ij}^a U[j]$ 
    end
  until CLOSE-ENOUGH( $U, U'$ )
  return  $U$ 
  
```

17

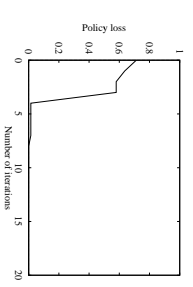
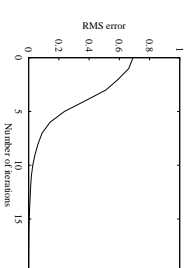
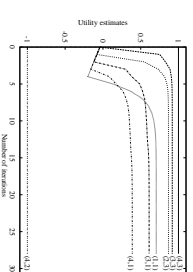
Summary

- Rational agents can be developed by combining **probability theory** and **utility theory**.
- Agents, whose preferences satisfy the axioms of utility theory have **Utility function**.
- Sequential problems in uncertain environments (MDP's) can be solved by computing a **policy**.
- **Value iteration** is one way of computing optimal policies.

18

Application of value iteration

Convergence behaviour



Utility of states and resulting policy after convergence.

3	0.812	0.888	0.912	↖ ↗
2	0.762	0.660	↖ ↗	
1	0.705	0.655	0.611	↖ ↗

3	→	→	→	↖ ↗
2	↑	↑	↑	↖ ↗
1	↑	↑	↑	↖ ↗

Foundations of AI

Reinforcement Learning

Passive and Active Learning

Luc De Raedt

19

Overview

- Naive Updating
- Adaptive Dynamic Programming
- Temporal Difference
- Exploration
- Q Learning
- Input generalisation
- Some applications

20

What are we learning ?

```
function PASSIVE-RL-AGENT( $\epsilon$ ) returns an action
  state:  $U$ , a table of utility estimates
   $N$ , a table of frequencies for states
   $M$ , a table of transition probabilities from state to state
   $percepts$ , a percept sequence (initially empty)
  add  $\epsilon$  to  $percepts$ 
  increment  $N[STATE[e]]$ 
   $U \leftarrow UPDATE(U, \epsilon, percepts, M, N)$ 
  if TERMINAL? $[e]$  then  $percepts \leftarrow$  the empty sequence
  return the action Observe
```

3 different "Update"-functions for passive agents :

1. naive updating (without a model)
2. adaptive dynamic programming (with a model)
3. temporal difference (without model)

How to learn a model ?

How to actively learn ?

22

Reinforcement Learning

The agent has *no* model of its environment $M_{\mathcal{S}}^a$ and *no* utility function $U(\mathcal{z})$ for states \mathcal{z} .

The agent is given rewards and punishments for sequences of actions. It is not informed of the "right" or better actions. \rightsquigarrow reward/reinforcement

Example: Winning or losing in games

passive learning: The agent observes the world and attempts to learn the utilities of states.

active learning: The agent performs explorative actions in order to learn about its environment. To this aim, it employs the already learned knowledge.

21

3 different learning situations :

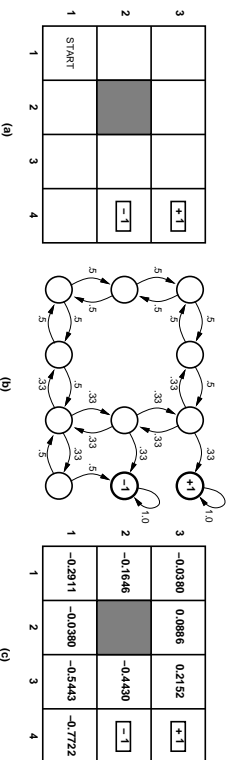
1. passive learning in a known environment
2. passive learning in an unknown environment
3. active learning in an unknown environment

23

Passive learning in a known environment

~> The agent knows the transition model M_{ij} .

State transitions/Actions are generated which are perceived by the agent = the agent determines the value of a given policy.



one possible trainingsequence : 11 → 21 → 31 → 32 → 33 → 43 : +1

Goal is to learn $U(i)$ for each non-terminal state i

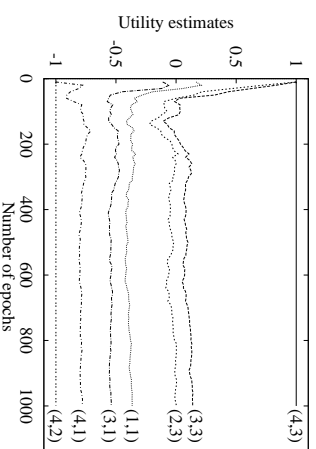
$U(i)$ = reward-to-Go for additive function

Sum of the rewards from i to terminal state

24

Learning curve for NU

The probabilities of the state transitions are ignored. The learning curve converges very slowly (more than 1000 epochs required).



26

Naive Updating (NU)

Assumption: observed Reward-to-Go determines actual Reward-to-Go

```

function LMS-UPDATE( $U, e, \text{percepts}, M, N$ ) returns an updated  $U$ 
  if TERMINAL? $[e]$  then  $\text{reward-to-go} \leftarrow 0$ 
  for each  $e_i$  in  $\text{percepts}$  (starting at end) do
     $\text{reward-to-go} \leftarrow \text{reward-to-go} + \text{REWARD}[e_i]$ 
   $U[\text{STATE}[e_i]] \leftarrow \text{RUNNING-AVERAGE}[U[\text{STATE}[e_i]], \text{reward-to-go}, N[\text{STATE}[e_i]]]$ 
end
  
```

~> Learning the utility function from examples (state, Reward-to-Go)

~> Reinforcement Learning reduced to inductive learning

Problem:

NU ignores the probabilities with which the state transitions occur.

25

The real utility of a state is

= weighted utility of successor states
+ own utility

27

Reinforcement learning using Adaptive Dynamic Programming

$$U(i) = R(i) + \sum_j M_{ij} U(j)$$

Note: actions are not important to a passive agent !

$R(i)$ Reward for reaching state i

has to be learned, i.e. observed once for each state !

M_{ij} probabilities of state transitions

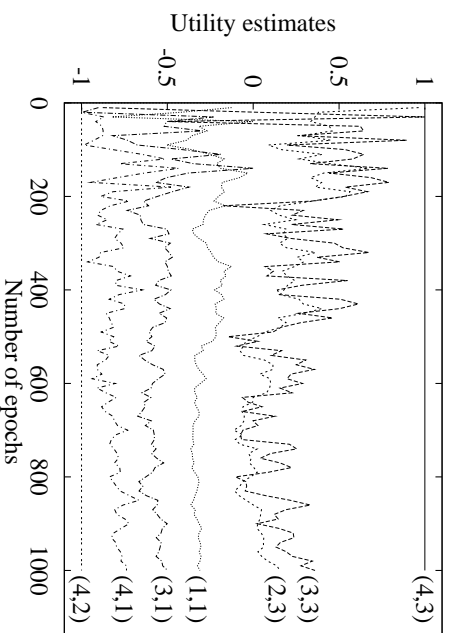
supposed to be known !

ADP solves set of equations !

ADP sets a standard for comparison (with other reinforcement learning techniques).

Not applicable to large state spaces: for Backgammon 10^{50} equations with 10^{50} variables.

28



30

Temporal Difference Learning

$$U(i) \leftarrow U(i) + \alpha (R(i) + U(j) - U(i))$$

When a state transition from i to j is observed, the value of i is updated using the value of j with learning rate α .

```
function TD-UPDATE( $U, e, \text{percepts}, M, N$ ) returns the utility table  $U$ 
  if TERMINAL[ $e$ ] then
     $U[\text{STATE}[ $e$ ]] \leftarrow \text{RUNNING-AVERAGE}(U[\text{STATE}[ $e$ ]], \text{REWARD}[ $e$ ], N[\text{STATE}[ $e$ ]])$ 
  else if  $\text{percepts}$  contains more than one element then
     $e' \leftarrow$  the penultimate element of  $\text{percepts}$ 
     $i, j \leftarrow \text{STATE}[ $e'$ ], \text{STATE}[ $e$ ]$ 
     $U[i] \leftarrow U[i] + \alpha(N[i])(\text{REWARD}[ $e'$ ] + U[j] - U[i])$ 
```

29

Passive Learning in an unknown environment

So far, we have assumed that the agent knows M_{ij}

- NU and TD employ only information about successors / final states, not about model.
- ~> can be applied unchanged

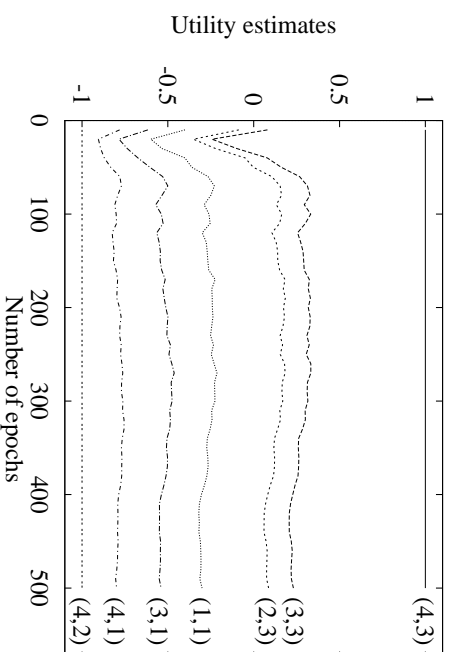
- ADP employs a model

~> has to learn a model of the environment
Learning of M_{ij}

- by directly observing state transitions
- each percept is an input/output pair of the transition function.

All inductive learners for stochastic functions are adequate. Easy: how frequently does a state transition occur ?

31



32

Performance Element

```

function ACTIVE-ADP-AGENT( $e$ ) returns an action
state:  $U$ , a table of utility estimates
 $M$ , a table of transition probabilities from state to state for each action
 $R$ , a table of rewards for states
 $percepts$ , a percept sequence (initially empty)
 $last-action$ , the action just executed

add  $e$  to  $percepts$ 
 $R[STATE][e] \leftarrow REWARD[e]$ 
 $M \leftarrow UPDATE-ACTIVE-MODEL(M, percepts, last-action)$ 
 $U \leftarrow VALUE-ITERATION(U, M, R)$ 
if TERMINAL?[ $e$ ] then
     $percepts \leftarrow$  the empty sequence
     $last-action \leftarrow$  PERFORMANCE-ELEMENT( $e$ )
return  $last-action$ 

```

34

Active Learning in an unknown environment

The transition model has to take into account all possible actions of the agent, we have to use M_{ij}^a .

The original equation of the value iteration algorithm has to be employed:

$$U(i) = R(i) + \max_a \sum_j M_{ij}^a \times U(j)$$

The agent applies its *Performance Element*, to select the actions.

33

Exploration (1)

Is there an optimal learning strategy ?

Maximizing exploration selecting actions at random in the hope of learning as much as possible about the environment \rightsquigarrow does not optimize utility; does not act goal oriented.

Maximizing exploitation maximizes utility on the basis of the present estimates of the utility function.

\rightsquigarrow the agent does not improve itself much after finding first solutions. \rightsquigarrow Agents do not find optimal solutions

\rightsquigarrow statistical decision problem: the n Bandit problem

\rightsquigarrow simple solution to assign (state,action) pairs a higher utility

35

Exploration (2)

$U^+(i)$ expected reward-to-go
 $N(a, i)$ number of times that action a was applied in state i
for value iteration, we obtain

$$U^+(i) \leftarrow R(i) + \max_a f \left(\sum_j M_{ij}^a U^+(j), N(a, i) \right)$$

$f(a, n)$ is the exploration function and determines the proportion between exploration and exploitation (increases with a and decreases in n).

36

Q Learning

so far, learning of utility function $U(i)$

now: learning the utility of executing an action a in a state i , i.e. learning the Q-function $Q(a, i)$

$$U(i) = \max_a Q(a, i)$$

- Q values are used directly for decision making
- are learned directly from rewards

$$Q(a, i) = R(i) + \sum_j M_{ij}^a \max_{a'} Q(a', j)$$

Q Learning with TD approach :

$$Q(a, i) \leftarrow Q(a, i) + \alpha (R(i) + \max_{a'} Q(a', j) - Q(a, i))$$

~> no explicit model M_{ij}^a needed !!!

37

Discussion

What is better ?

- A model and a utility function for learning ?
- The value of an action without a model ?

How can agents best be represented ?

knowledge based approach :

The explicit representation of the agent and its environment in a model is advantageous when situations become complex.

counter argument :

The existence of successful learning algorithms that do not require a model proves that knowledge based methods are not needed.

~> Discussion is still open

38

Generalisation (1)

The states space of chess and backgammon have ca. 10^{120} resp. 10^{50} states. It is impossible to visit all of these states while learning (as required by the previous methods).

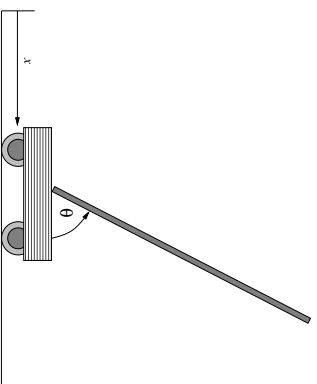
We need an implicit representation of the utility function. For instance, in chess, we can describe positions using attributes. ~> ca. 10 attributes instead of 10^{120} states

Generalisation of visited states to non-visited ones ! ~> often employ neural nets

39

Successful applications

- in Games: Checkers, TD-gammon, Chess.
- **Control**
E.g. elevators, cart-and-pole



40

Summary(1)

We distinguish **active** from **passive** Learning.

1. with a world model : **Utility function** over states is learned
2. without a model : **Action value function** is learned

The **utility** of a state is the sum of the rewards that the agents expects between the present state and the final one.

41

Summary(2)

3 approaches to learning the utility function

1. Naive Updating
2. Adaptive Dynamic Programming
3. Temporal Difference.

Q Learning Learning of Action Value function

42