Artificial Intelligence Course
Notes from lections

By: Alex S.

Feb 2024 - Jun 2024

Exam

Date: 11.06 12:30

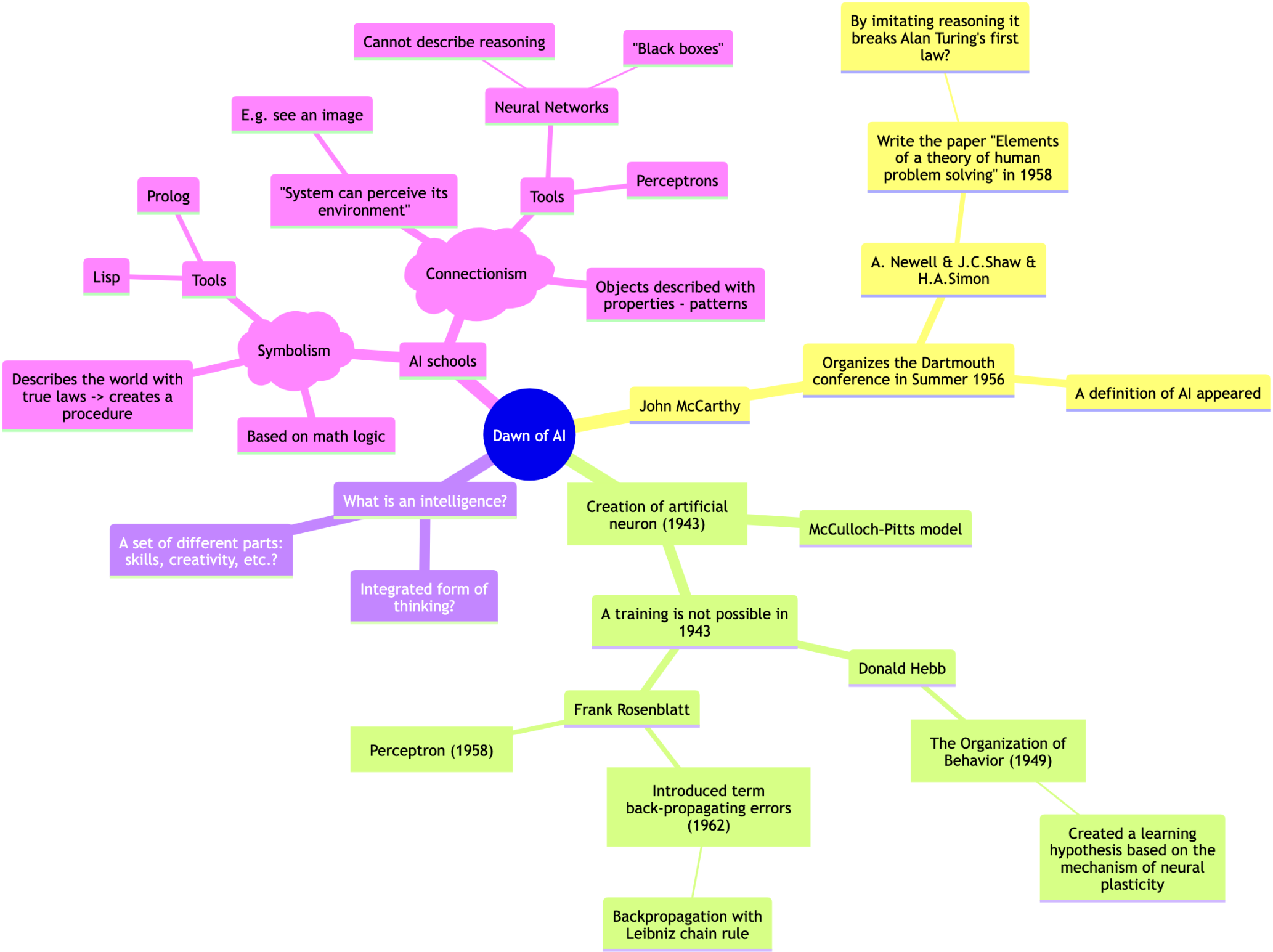
Duration: 4 hours

Theoretical part

- No use of notes, only memory
- Consists of 5 blocks(In the parentheses the number of notes is indicated)
  1. Intelligent agents and their architecture (*See 2, 3, 4*)
  2. Logical agents (*See 5, 6*)
  3. Uncertain knowledge (*See 7*)
  4. Decision systems (*See 8, 9*)
  5. Learning (*See 10, 11*)
- There will be given 2 questions to choose for the answer
- Each question gives 2 points, therefore $\Sigma = 10$
- The part is weighted with 0.3 for the whole mark

Practical part

- 4 tasks:
  1. Describe a world with the logical statements (*See 5, 6*) [2 points]
     - Given a world of 3x3 or 3x4 with empty squares, need to choose the barriers and rewards in 2 places
     - Need to specify sensors and actions in the natural language
     - Need to translate the actions to the language of logic
     - Need to create a Knowledge base of at least 5 sentences
     - The agent needs to do an inference with MODUS PONENS, preferably two times in a row
  2. Uncertain knowledge and decision making (*See 7, 8, 9*) [2 points]
     - Need to find probabilities for all states where the agent can get in 2 turns
       - The world is stochastic
       - Actions are given
     - Need to draw a stochastic action representation tree
     - Need to calculate an expected utility value
  3. Value and policy iteration algorithms (*See 9*) [3 points]
     - Need to do 2 iterations
     - Need to define a policy after each iteration
  4. Reinforcement learning (*See 11*) [3 points]
     - Need to generate training sequence by yourself
     - Use a naive learning
     - Create a transition model diagram
     - Use a temporal-difference algorithm
- Overall $\Sigma = 10$ points
- Weighted with 0.2 from the whole mark
- **Recommended sequence to do tasks: 4, 2, 3, 1**


Dawn of AI
AI schools
Symbolism
Describes the world with true laws -> creates a procedure
Tools
Lisp
Prolog
Based on math logic
Connectionism
"System can perceive its environment"
E.g. see an image
Tools
Neural Networks
Cannot describe reasoning
"Black boxes"
Perceptrons
Objects described with properties - patterns
What is an intelligence?
A set of different parts: skills, creativity, etc.?
Integrated form of thinking?
John McCarthy
Organizes the Dartmouth conference in Summer 1956
A. Newell & J.C.Shaw & H.A.Simon
Write the paper "Elements of a theory of human problem solving" in 1958
By imitating reasoning it breaks Alan Turing's first law?
A definition of AI appeared
Creation of artificial neuron (1943)
McCulloch-Pitts model
A training is not possible in 1943
Donald Hebb
The Organization of Behavior (1949)
Created a learning hypothesis based on the mechanism of neural plasticity
Frank Rosenblatt
Perceptron (1958)
Introduced term back-propagating errors (1962)
Backpropagation with Leibniz chain rule

2. Intelligent agents(IA)

By: Alex S.

Feb, 2024

1. Intelligent agents

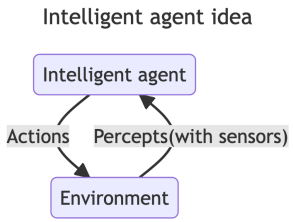


Figure 1: Simple intelligent agent representation

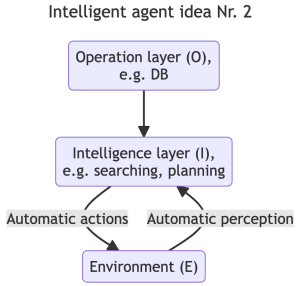


Figure 2: Extended intelligent agent representation

We can describe an intelligent agent as a mathematical function which maps perceptions to the specific action:

$$f : P \to A,$$

where P is a perception set and A is an action set.

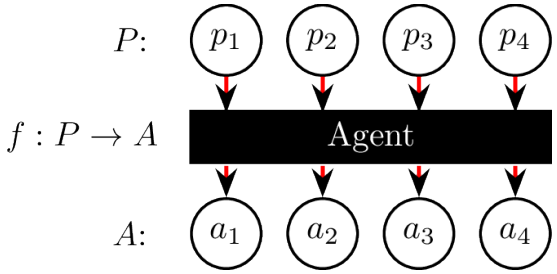
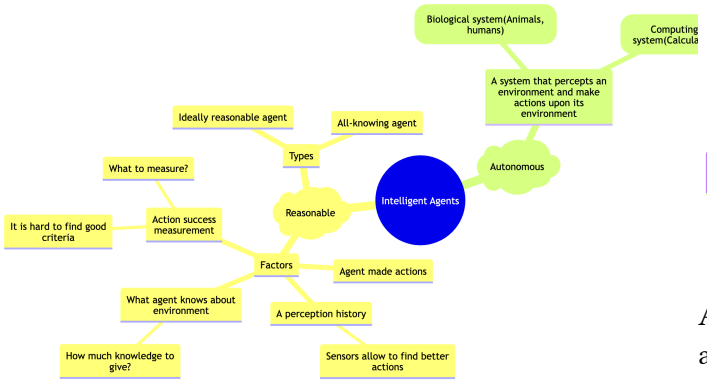


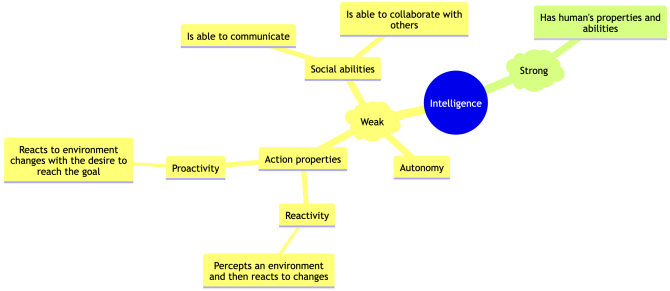
Figure 3: Agent as a function can be seen as a black box

Agent is a function that is a black box, i.e. we don't know how it produces its results. An agent receives information(perceptions) from sensors(P), processes them via function(f) and returns back an action(from set A) to execute.

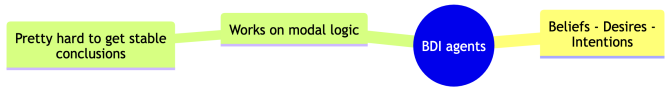
2. Intelligent agent types



3. IA intelligence



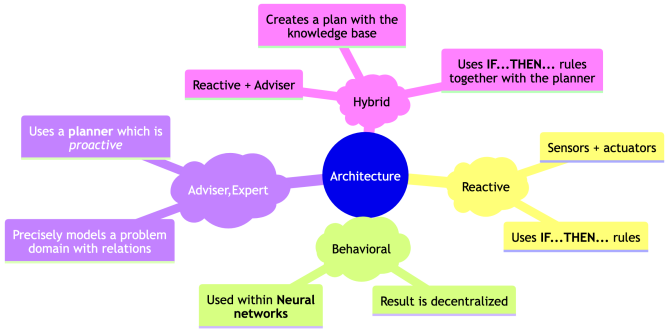
4. BDI agents



5. IA as a program

An agent can be described as a program:

agent = program + architecture(computing platform)



Architectural properties can be described with the abbreviations **PAGE** and **PEAS**.

- **PAGE** - percept, actions, goals and environment.
- **PEAS** - performance, environment, actuators and sensors.

3. Intelligent agent(IA) types

By: Alex S.

Feb, 2024

1. Intelligent agent as a function

As was described in previous lections we can describe intelligent agent as a function:

$$f : P \to A$$

where P is a perception set and A is an action set.

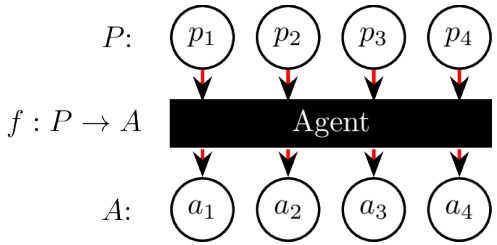


Figure 1: Agent as a function can be seen as a black box

Agent is a function that is a black box, i.e. we don't know how it produces its results. An agent receives information(perception P) from sensors, processes them via function(f) and returns back an action(A) to do.

The common algorithm of intelligent agent can be described as:

function $f : P \to a \in A$ `is`
  let `Knowledge Base : Memory(`M`)`
  `update :` $(M, P) \to M'$
  `action :` $M' \to a$
  `update :` $(M', a) \to M''$
return a

2. Intelligent agent types

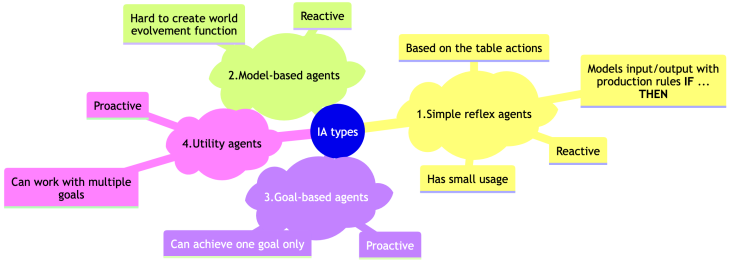


Figure 2: IA types

3. Simple reflex agents

Algorithm can be described the following way:

function $f_r : P \to a \in A$ `is`
  let R `= production rules - IF condition THEN action`
  interpret input : $P \to s_i \in S$[state]
  rule search : $(s_i, R) \to r_i \in R$
  rule burning : $r_i \to a$
return a

Illustration of the simple reflex agents working principle in Figure 3.

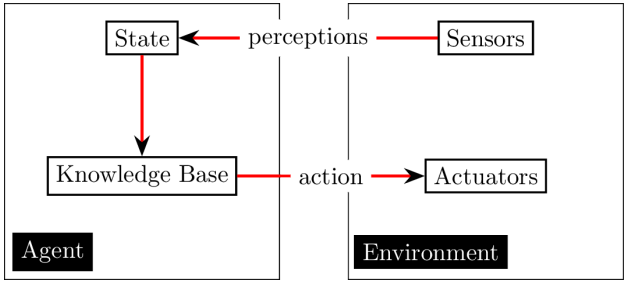


Figure 3: Simple reflex agents working principle

Simple reflex agents uses *Knowledge Base* with production rules.

Example:

R1: **IF** a driving car ahead has both stop signals turned on(*N*) **THEN** a driving car ahead is stopping (*C1*)

R2: **IF** a driving car ahead is stopping(*C1*) **THEN** a driving car ahead is slowing down (*C2*)

R3: **IF** a driving car ahead is slowing down(*C2*) **THEN** need to start to slow down (*C3*)

R4: **IF** need to start to slow down (*C3*) **THEN** press a braking pedal(*A*)

Where *R* is a production rule, *C* - conclusion, *N* - condition, *A* - action.

By using an **inference** mechanism, we can make a conclusion, that if we start with the production rule *N*, then we have the following chain: $N \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow A$ and that we should press the braking pedal.

4. Model-based agents

1. Follows how changes the environment, the state. Knows the previous state.
2. Environment changes regardless of agent.
3. Need to understand the consequences from actions.

All 3 actions describe the **inner state** of the agent(Figure 4).

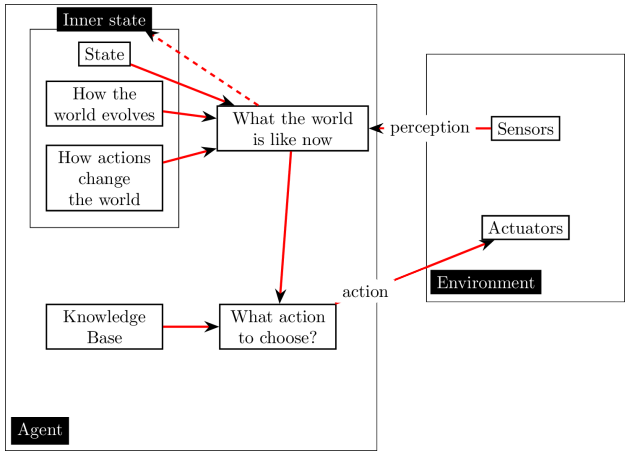


Figure 4: Model-based agents working principle

5. Goal-based agents

- Actions are dependent on what the goal is needed to be achieved by the agent.
- Can be effectively used with the search or planning(for robots) algorithms.

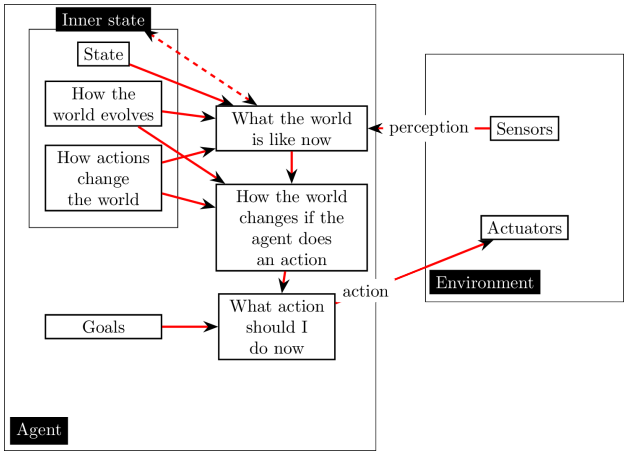


Figure 5: Goal-based agents working principle

6. Utility agents

- Can be applied when the problem cannot be solved with the goal-based agent, e.g.
  - More than 1 goal
  - And goals may be conflicting, therefore some compromise should be found
- Goals might be achieved using probabilities(e.g. games of chance)
- **Utility function**(U) is a measurement that allows to compare different states, i.e. given two states($s_1, s_2 \in S$) and give a numeric

value(v) of how these states differ: $U : (s_1, s_2) \rightarrow v$

- Agent can be based on the **utility theory**.

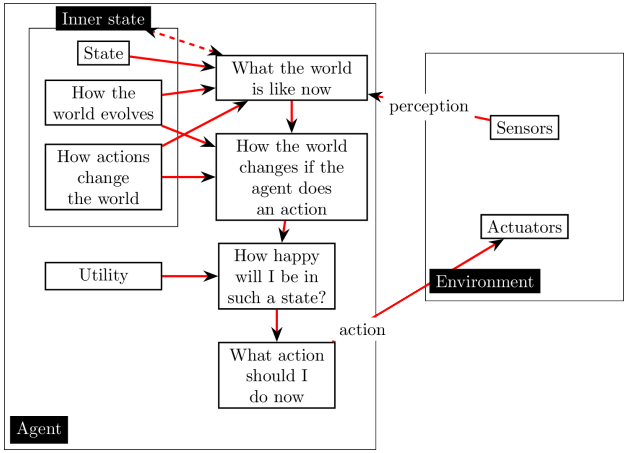


Figure 6: Utility agents working principle

7. Learning agents

- The agent has a capability to learn taking into account external performance standard

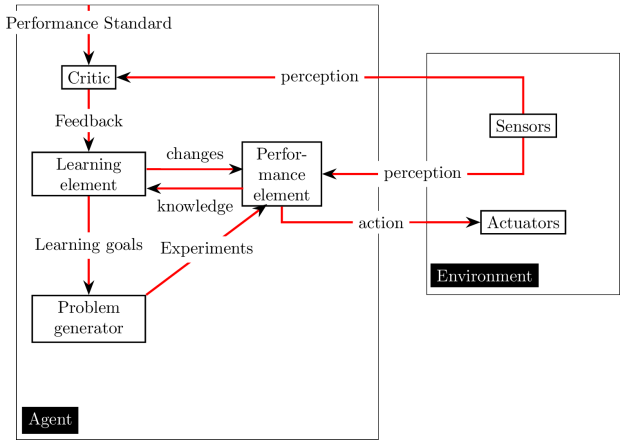


Figure 7: Learning agents working principle

8. Methodologies

- Not sure if something serious uses these tools, but somebody might use:
  - **FIPA** (Foundation for Intelligent Physical Agents)
  - **BDI** (behavior - desire - intent) agents
- Not sure if somebody will ever use some of these methodologies in practice:
  - **PROMETHEUS** - iterative waterfall model
  - **Gaia**
  - **MaSE**
  - **MASITS** - local RTU development for multi-agents that helps with tutoring

4. Intelligent agents and environments

By: Alex S.

Feb, 2024

1. Environments and their properties

1. **Fully observable** vs. **Partially observable** - if the agent with sensors can percept full, correct and actual information about environment's state. If the agent has no sensors at all, then the environment is **unobservable**.
2. **Deterministic** vs. **Stochastic** - every action has a guaranteed result that the next state can be predicted: $(s_i, a_i) \to s_i'$. If the next state s_i' cannot be clearly determined, then the environment is stochastic.
3. **Dynamic** vs. **Static** - if the environment changes independently from the agent actions, e.g. during calculation times, then it is dynamic. The environment is static if only the agent changes the environment. If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is **semidynamic**, e.g. chess with the clock.
4. **Discrete** vs. **Continuous** - with discrete environment there is an amount of states that can be counted in a period of time - with continuous there are infinitely many states. E.g. chess is discrete, but moving the steering wheel is analog therefore continuous.
5. **Episodic** vs. **Sequential** - in episodic the agent's experience is divided into atomic episodes, where their actions do not affect future decisions, e.g. production line. In sequential every action affects the future ones, e.g. chess.
6. **Single agent** vs. **Multiagent** - where one agent works only with objects. Multiagent environment works with other agents where happens interaction on each other. E.g. in chess one agent's move affects decision of the second one.

2. Examples of environments & their characteristics

Agent	Observable?	Deterministic?	Static?	Discrete?	Episodic?	Single-agent?
Mail sorter	Yes	No	Yes	Yes	Yes	Yes
Detail transferer	Yes	No(Yes, if the same detail)	No(Yes, if conveyor)	Yes(No, if real situation)	Yes	Yes
Chess	Yes	Yes	Yes	Yes	No	No
Autonomous vehicle	Partially	No	No	No(Yes, if sensors)	No	No

3. Environment imitation

There is pseudocode how environment can be imitated for agent actions:

function $f : (s_0, \text{update}, A, c)$ `is`
  $s_i \leftarrow s_0$
  loop until c `is not reached`
    foreach $a_i \in A$ **do**
      generate perception : $(a_i, s_i) \to P$
      compute action : $P \to N$
      update : $(N, s_i) \to s_i'$
      evaluate : $(a_i, s_i') \to U = \text{const}$
    end foreach
    $s_i \leftarrow s_i'$
  end loop
end

where f - environment imitation function, s_0 - starting state,

update - state update function, A - set of agents,

c - condition or predicate when process should end, P - set of perceptions,

N - set of actions, s' - updated state, U - evaluation score - evaluation score.

5. Logical agents

By: Alex S.

Feb, 2024

1. Knowledge-based agents

This type of agents has a representation of knowledge. It is mostly defined as a knowledge base(KB). KB is described with the set of **sentences** or **axioms**.

There must be a way to add new sentences to the KB and a way to query what is known. The standard names for these operations are **TELL** and **ASK**, respectively. Both operations may involve **inference** - that is, deriving new sentences from old.

Each time the agent program is called, it does three things:

1. It **TELLs** the KB what it perceives.
2. It **ASKs** the KB what action it should perform.
3. The agent program **TELLs** the knowledge base which action was chosen, and the agent executes the action.

The procedure is shown in the diagram(Figure 1):

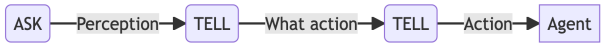


Figure 1: Inference mechanism

A generic knowledge-based agent can be described with the following pseudocode:

```
function KB-agent : P → a is
  let KB be persistent
  let t ← 0
    tell : (KB, do-percept-sentence : (P, t))
    ask : (KB, do-action-query : t → a) → a
    tell : (KB, do-action-sentence : (a, t)) → a
    t ← t + 1
  return a
```

where P - perception set; t - counter, time; a - action; KB - knowledge base

Agent's engineering can happen on different levels:

1. **Knowledge level** - where agent is being described with natural language - what it knows and what its goals are.
2. **Knowledge representation level** - How we represent the knowledge, e.g. semantic network, conceptual maps, object-oriented frames, logic language
3. **Implementation level** - how the agent is implemented and what tools are used for this goal.

2. The Wumpus World (Practical example)

Here we define the game where agent should achieve its given goal.

For the game we define the **PAGE** properties - *perceptions*, *actions*, *goals* and *environment*.

- **Goal** - to reach the gold ingot, take it and then come back to the starting position $(1, 1)$.
- **Performance measure** - is measured as a sum of points.
  - 100 points if the agent reaches the goal.
  - -100 points if the agent dies.
  - -1 points for every taken action.
  - -10 points for shooting an arrow into Wumpus.
- **Environment** - a 4x4 grid of rooms. The agent always starts at the square $(1, 1)$ facing to the right.

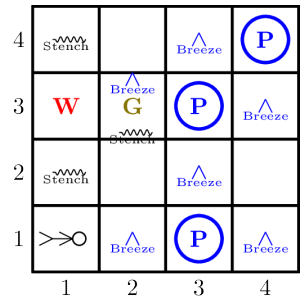


Figure 2 represents an example environment of generated randomly Wumpus World. Each initial state has *1 gold*, *1 Wumpus* and *3 pits*.

Figure 2: Wumpus World example

The initial state of the board is generated in a random manner, i.e. Wumpus and gold are chosen randomly with uniform distribution, excluding the starting square. Each other square can be a pit with the probability of 0.2. Probability number can be whatever, it can be a throw of a dice with the probability $\frac{1}{6}$. The gold can be in the same spot where *Wumpus* is, but cannot be in the same place where the pit is.

- **Sensors** - the agent has 5 sensors, each of which gives a single bit of information. The agent cannot percept diagonally.
  - The squares adjacent to the square that contains *Wumpus* has *Stench*. In Figure 2 square $(1, 3)$ emits stench to squares $(1, 2), (1, 4), (2, 3)$

- In the squares next to *pit*, the agent percepts *breeze*. In Figure 2, pit $(3, 1)$ emits breeze in $(2, 1), (3, 2), (4, 1)$
- In the square with the gold, it will perceive *Glitter*
- When the agent walks into a wall, it will perceive a *Bump*
- When the *Wumpus* is killed, it emits *Scream* that can be perceived anywhere in the cave.

- **Actuators** - The agent can:
  - Move *forward* where he looks
  - *Turn left* 90°
  - *Turn right* 90°
  - *Grab* the gold in the same square where the agent is
  - *Shoot* the arrow that is fired in a straight line, in the direction the agent is facing.

Let's explore the world.

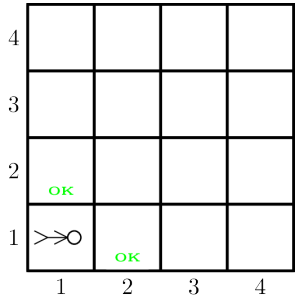


Figure 3: First move in the Wumpus World

The initial knowledge says that square $(1, 1)$ is safe, so it is marked as *OK*. The first percept is *[None, None, None, None, None]*(i.e. *[Stench, Breeze, Glitter, Bump, Scream]*). Therefore squares $(1, 2)$ and $(2, 1)$ are safe as shown in Figure 3.

Since both $(1, 2)$ and $(2, 1)$ are equally safe, it does not matter which to choose to go first. In the computer world, the first equal solution would be chosen. Let's imagine that the agent has chosen the square $(2, 1)$ and it moves forward$(100 - 1 = 99$ points, assuming it will bring the gold).

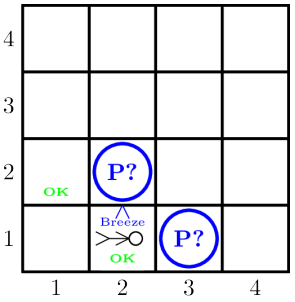


Figure 4: First move in the Wumpus World

The next percept is *[None, Breeze, None, None, None]*. The agent feels the breeze. The pit cannot be in square $(1, 1)$ by the rules. Therefore the pit or both can be on squares $(3, 1)$ or/and $(2, 2)$ as in Figure 4.

The notation **P?** indicates a possible pit in the square. The only safe square the agent knows is $(1, 1)$. So, it turns left 2 times, moves forward, turns right and moves forward to square $(1, 2)(99 - 5 = 94p)$.

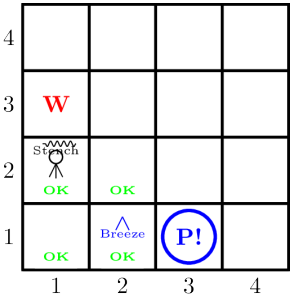


Figure 5: First move in the Wumpus World

In this state the agent percepts *[Stench, None, None, None]*. It means the *Wumpus* is nearby. But it cannot be in $(1, 1)$ square, and it cannot be in $(2, 2)$, otherwise it would have felt the stench in the square $(2, 1)$.

Therefore the agent can conclude that *Wumpus* is in $(1, 3)$. Moreover, the lack of breeze also indicates that there is no pit in $(2, 2)$ and it must be in $(3, 1)$ indicated by **P!** in Figure 5. The agent can infer that it is safe to move to the square $(2, 2)$, by rotating right and going forward$(94 - 2 = 92p)$. In $(2, 2)$ agent does not percept anything *[None, None, None, None, None]*. So, it is safe to move to $(2, 3)$ or $(3, 2)$. Let's consider the agent moves to $(2, 3)$ by rotating left and moving forwards$(92 - 2 = 90p)$.

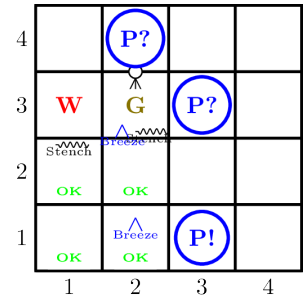


Figure 6: First move in the Wumpus World

In $(2, 3)$ the agent percepts *[Stench, Breeze, Glitter, None, None]*. So, it infers that possible pits are in $(3, 3)$ and $(2, 4)$, however, it grabs the gold as it feels the glitter$(90 - 1 = 89p)$ and starts creating a path that returns him back to $(1, 1)$.

For this purpose it needs to know the history of actions and use Dijkstra's algorithm or BFS for the shortest path. The actions to return home:

- Turn left 2 times$(89 - 2 = 87p)$
- Move to $(2, 2)(87 - 1 = 86p)$
- Turn right and move to $(1, 2)(86 - 2 = 84p)$
- Turn left and move to $(1, 1)(84 - 2 = 82p)$

In the end the agent returns back with the gold and 82 points.

3. Requirements for implementation of knowledge representation language

Languages can be split into 2 categories:

Artificially created languages

- E.g. programming, logic, mathematical languages

Natural languages

- E.g. languages we speak in
- Can be used for a communication between humans

- Can be used for a precise algorithm description,
- They are unambiguous, i.e. no hidden description
- They are not dependent on the context
- They are unclear and ambiguous
- They are dependent on the context

Characteristics of knowledge representation language:

- Expressive
- Concise and concentrated
- Unambiguous
- Context independent
- Effective, i.e. use inference mechanisms

Any language is defined by

1. **Syntax** - symbol configuration that is used to build sentences, e.g.
   - $x + y = 8$ is syntactically *correct*
   - $x8y =$ is syntactically *incorrect*
2. **Semantics** - defines the truth of each sentence with respect to each **possible world**. Semantics can be applied with the **interpretation** - that defines a relationship between sentences and facts in the real world. E.g. $x + y = 8$ can be true or false. The idea of this statement depends on the world state.

   If $x = 4$ and $y = 4$ the sentence is true.

   If $x = 7$ and $y = 4$ the sentence is false.

The amount of world states can be calculated with the formula:

$$\text{Number of worlds}(N) = 2^N$$

where N is the number of sentences.

4. Logical reasoning

Model is representation of a "possible world". Possible worlds might be thought as real environments that the agent might or might not be in.

Model is a mathematical abstraction, that simply fixes the truth or falsehood of every relevant sentence.

If sentence α is true in model m, we say that m **satisfies** α or m **is a model of** α.

Notation: $M(\alpha)$ - set of all models of α.

Logical entailment between sentences - the idea that a sentence *follows logically* from another sentence:

$$\alpha \vDash \beta$$

where α and β are sentences.

Entailment means that in every model in which α is true, β is also true.

$$\alpha \vDash \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

where $M(\alpha)$ is a subset of $M(\beta)$.

If $\alpha \vDash \beta$, then α is *stronger* assertion than β: it rules out more possible worlds. E.g. the sentence $x = 0$ entails the sentence $xy = 0$. In any model where x is zero, it is the case the xy is zero too.

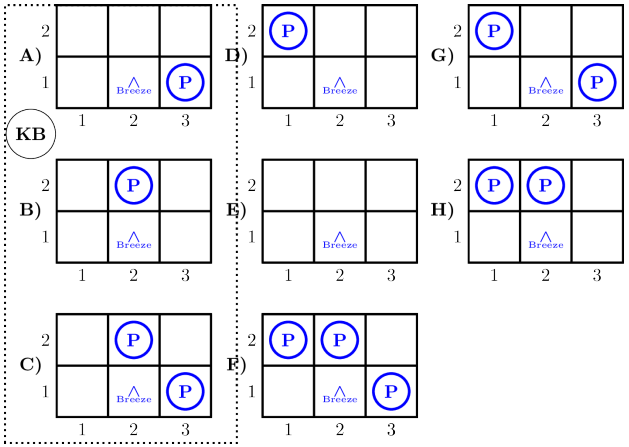


Figure 7: Possible models for the presence if pits in squares $(1, 2)$, $(2, 2)$ and $(3, 1)$. The KB corresponding to the observations of nothing in $(1, 1)$ and a breeze in $(2, 1)$ is shown by the dotted line.

In Figure 7 shown an example situation when the agent percepts a breeze in $(1, 2)$. The agent is interested in whether the adjacent squares $(1, 2)$, $(2, 2)$, and $(3, 1)$ contain pits. Each of the three squares might or might not contain a pit, so there are $2^3 = 8$ possible models.

The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences. The KB is false in models that contradict what the agent knows — for example, the KB is false in any model in which $(1, 2)$ contains a pit(D, F, G, H), because there is no breeze in $(1, 1)$. There are in fact just three models in which the KB is true, and these are shown in the dotted line in Figure 7.

Now let us consider 2 possible conclusions:

- α_1 = "There is no pit in $(1,2)$" which corresponds to states A, B, C and E in Figure 7
- α_2 = "There is no pit in $(2,2)$" which corresponds to states A, D, E and G in Figure 7

By inspection, we see the following: in every model in which *KB* is true, α_1 is also true. Hence, $\text{KB} \models \alpha_1$: there is not pit in $(1,2)$.

We can also see that in some models in which *KB* is true, α_2 is false. Hence, $\text{KB} \not\models \alpha_2$: the agent cannot conclude that there is no or there is a pit in $(2,2)$.

The example has shown that entailment can be applied to derive conclusions, i.e. carry out **logical inference**.

Model checking is the inference algorithm, shown in Figure 7 that enumerates all possible models to check that α is true in all models in which *KB* is true, so that $M(\text{KB}) \subseteq M(\alpha)$.

If an inference algorithm i can derive α from *KB*, we write:

$$\text{KB} \underset{i}{\vdash} \alpha$$

which is pronounced "α is derived from *KB* by i".

An inference algorithm that derives only entailed(true) sentences is called **sound** or **truth-preserving**, can also be called **deduction**.

The property of **completeness** is also desirable: an inference algorithm is complete if it can derive any sentence that is entailed.

Every **complete** inference can be **sound**, but not every **sound** inference can be **complete**.

The sentence is **valid** only if it is true in all possible interpretations and in all models, e.g.
- "Pit exists or pit doesn't exist".

The sentence is **executable**, if there is at least one interpretation in some model where the sentence is true, e.g.
- "Wumpus is in square $(1,4)$": might be true or false: $1 \vee 0 = 1$
- "Forward there is a wall and there is no wall": not **executable** as it is false in all models: $1 \wedge 0 = 0$

The correspondence between world and representation is shown in Figure 8.

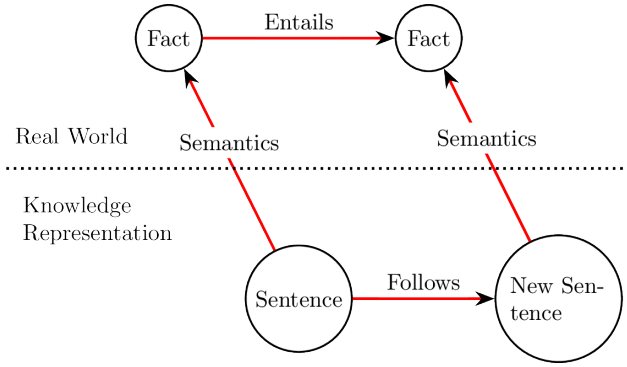


Figure 8: Sentences are physical configurations(aspects of the real world) of the agent, and reasoning is a process of constructing new physical configurations from old ones

6. Propositional logic: a very simple logic

By: Alex S.

Mar, 2024

1. First-order logic

- Weakly fitted for uncertain knowledge
- Hard to describe any type of heuristics
- Due to amount of sentences, it is hard to describe complex problems

Relations between knowledge

Ontology	Epistemology
Related to the real nature, assumptions about the real world	Related to probable knowledge and states via representation languages
• Propositional logic assumes that there are facts. • Predicate logic assumes that the world is made of **objects** that have *true* or *false* relations. • Special(temporal) logic assumes that the world is ordered by time moments or intervals.	Full confidence that sentence is either *true* or *false*
• Probability theory with stochastic environments.	Order of confidence in interval $[0; 1]$
• Theory of confidence? [*Haven't found any reference to the correct term*] - facts exists with the probability that the expert have entailed. • Fuzzy logic - facts exists with some order of truthfulness	

2. Propositional logic syntax

Syntax is made of **alphabet**(defines symbols) + **grammar**(defines how to construct sentences correctly).

1. **Proposition symbols**: big latin alphabet letters, most often starting from P: $P, Q, R, ...$, but any other letter can be used as well, e.g.

   $P_{1,3}$ - pit is in the square $(1, 3)$

2. **Boolean symbols**: *T* for true and *F* for false.
3. **Logical connectives**:
   - negation(NOT): $\neg$, e.g.

     $\neg P_{1,3}$ - pit is not in the square $(1, 3)$
   - conjunction(AND): $\wedge$, $\&$
   - disjunction(OR): $\vee$
   - implication(IF...,THEN...): $\Rightarrow$, $\rightarrow$ (implies)
   - equivalence(IF ..., AND ONLY IF ...): $\equiv$, $\Leftrightarrow$
4. **Aid symbols**: different types of parentheses $[()]$

Every proposition/boolean symbol is an **atomic** sentence, e.g.

$P_{1,3}$ - pit is in the square $(1, 3)$ - is **atomic**

Complex sentences are constructed from simpler sentences(atomics), using parentheses and **logical connectives**, e.g.

$P_{1,3} \vee \neg P_{1,3}$ - pit is or is not in the square $(1, 3)$ - is a **complex sentence**

Example of logic sentence notation:

Cyclone(P) *consequences are* ***strong wind***(Q) *and* ***big waves***(R)*, but* ***anticyclone***(S) *consequences are* ***no wind***(T) *at all.*

We can separate certain parts of the sentence into atomic sentences described by one letter in parenthesis and get:

$$(P \rightarrow (Q \wedge R)) \wedge (S \rightarrow T)$$

We can decrease number of parenthesis with **logical connectives relation power**:

1. The biggest relation is for **negation**, exactly by the symbol.
   - $\neg P \wedge Q$ - not P, then and Q
   - $\neg(P \wedge Q)$ - first $P \wedge Q$, then $\neg$
2. **Conjunction and disjunction**:
   - $P \vee Q \vee R$ - is incorrect, order of operations is not defined, need to specify either
   - $(P \vee Q) \vee R$ or $P \vee (Q \vee R)$
3. **Implication** works on the whole expression on the right or left side:
   - $P \rightarrow Q \wedge R$ - is ok, as well as $Q \wedge R \rightarrow P$ - implication works after **AND** operation.
4. **Equivalence** waits until both sides are resolved:
   - $P \wedge Q \rightarrow R \equiv S$ - waits until $P \wedge Q \rightarrow R$ and S are resolved.

3. Semantics

Atomic sentences are easy:

- *True* is true in every model and *False* is false in every model.
- The truth value of every proposition symbol must be specified directly in the model. For example, in the model m, P must be false or true.

For complex sentences, we have five rules, which hold for any subsentences P and Q in any model m (here "iff" means "if and only if"):

- $\neg P$ is true iff P is false in m
- $P \wedge Q$ is true iff both P and Q are true in m
- $P \vee Q$ is true iff either P or Q is true in m
- $P \rightarrow Q$ is true unless P is true and Q is false in m
- $P \equiv Q$ is true iff P and Q are both true or both false in m

The rules can also be expressed with **truth tables**. True is represented by 1 and False by 0:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \equiv Q$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Table 1: Truth table for the five logical connectives

Equivalences:

- $\neg\neg P \equiv P$
- $\neg P \rightarrow Q \equiv P \vee Q$
- $P \rightarrow Q \equiv \neg P \vee Q$ - resolution law
- $P \rightarrow Q \equiv \neg Q \rightarrow \neg P$
  - We can prove that the equivalence is correct by using **truth tables**:

P	Q	$P \rightarrow Q$
0	0	1
0	1	1
1	0	0
1	1	1

$\neg P$	$\neg Q$	$\neg Q \rightarrow \neg P$
1	1	1
1	0	1
0	1	0
0	0	1

- De Morgan's law:
  - $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
  - $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
- Commutative property:
  - $P \wedge Q \equiv Q \wedge P$
  - $P \vee Q \equiv Q \vee P$
- Associative property:
  - $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
- Distributive property:
  - $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
  - $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

4. Inference procedure

If sentence α is true in model m, we say that m **satisfies** α or m **is a model of** α.

Notation: $M(\alpha)$ - set of all models of α.

Logical entailment between sentences - the idea that a sentence *follows logically* from another sentence:

$$\alpha \vDash \beta$$

where α and β are sentences.

Entailment means that in every model in which α is true, β is also true.

$$\alpha \vDash \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

where $M(\alpha)$ is a subset of $M(\beta)$.

If $\alpha \vDash \beta$, then α is *stronger* assertion than β: it rules out more possible worlds. E.g. the sentence $x = 0$ entails the sentence $xy = 0$. In any model where x is zero, it is the case the xy is zero too.

Inference must be **sound** and it is preferable for it to be **complete**.

Meta-form of the entailment:

$$\frac{\alpha}{\beta} \equiv \alpha \vDash \beta$$

where α is a **premise** - a proposition — a true or false declarative statement, used in an argument to prove the truth of another proposition called the conclusion, β is the **conclusion**.

4.1. MODUS PONENS

$$\frac{\alpha \rightarrow \beta, \alpha}{\beta} \equiv \alpha \rightarrow \beta, \alpha \vDash \beta$$

where α and $\alpha \rightarrow \beta$ are premises, and β is a conclusion.

- In the KB $\alpha \rightarrow \beta$ represents sentences IF... THEN...
- α is the parameter that comes from sensors and goes into working memory
- After working memory is loaded a sentence is being searched, so that $\alpha \equiv T$ and $\alpha \rightarrow \beta \equiv T$

MODUS PONENS can be proved to be **sound** by using **truth table**. Whenever premises α and $\alpha \rightarrow \beta$ are *True*, and the sentence β is also *True* in every

case, then it can be inferred and procedure is **sound**.

α	β	$\alpha \to \beta$
0	0	1
0	1	1
1	0	0
1	**1**	**1**

Table 2: Truth table for the MODUS PONENS

Since premises $\alpha \equiv T$ and $\alpha \to \beta \equiv T$ holds *True*, and conclusion $\beta \equiv T$, then it concludes that the law is **sound**.

Example:

- $\alpha \to \beta$: (WumpusAhead $\wedge$ WumpusAlive $\to$ Shoot)
- α: (WumpusAhead $\wedge$ WumpusAlive)
- Therefore with MODUS PONENS we can infer that $\beta \equiv$ Shoot

AND-elimination

- Allows to infer separate conjuncts, that are guaranteed to be *True*:

$$\frac{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n}{\alpha_1, \alpha_2, ..., \alpha_n}$$

Example:

- $\alpha_1 \wedge \alpha_2$: (WumpusAhead $\wedge$ WumpusAlive)
- We can infer that $\alpha_1 \equiv$ WumpusAhead and $\alpha_2 \equiv$ WumpusAlive

AND-inclusion

$$\frac{\alpha_1, \alpha_2, ..., \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n}$$

OR-inclusion

$$\frac{\alpha}{\alpha \vee \beta \vee \gamma \vee ...}$$

Double negation

$$\frac{\neg\neg\alpha}{\alpha}$$

4.2. Unit resolution

$$\frac{\alpha \vee \beta, \neg\beta}{\alpha}$$

- can also be called as a **proof by contradiction**

Example:

- If there's a pit(P) in one of $(1,1), (2,2), (3,1)$ and it's not in $(2,2)$, then it's in $(1,1)$ or $(3,1)$.
- Symbolically:

$$\frac{P_{1,1} \vee P_{2,2} \vee P_{3,1}, \neg P_{2,2}}{P_{1,1} \vee P_{3,1}}$$

4.3. Full resolution

- Comes from the unit resolution as

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma} \equiv \frac{\neg\alpha \to \beta, \beta \to \gamma}{\neg\alpha \to \gamma}$$

Example:

$$\frac{P_{1,1} \vee P_{3,1}, \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

α	β	γ	$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	**1**	**1**	**1**
1	0	0	**1**	**1**	**1**
1	0	1	**1**	**1**	**1**
1	1	0	1	0	1
1	1	1	**1**	**1**	**1**

Table 3: Full resolution soundness proof

In Table 3 there are 4 cases where premises $\alpha \vee \beta$ and $\neg\beta \vee \gamma$ are true. Since all conclusions $\alpha \vee \gamma$ are also true in all those 4 cases, we can conclude that **full resolution** is **sound**.

4.4. MODUS TOLLENS

$$\frac{\alpha \to \beta, \neg\beta}{\neg\alpha}$$

- E.g. there is a breeze in the square(α sentence), then the pit is in the nearest squares(β).
- **MT**: The pit is not in the nearest squares($\neg\alpha$), then the breeze was not perceived($\neg\alpha$).

4.5. Abduction rule

$$\frac{\alpha \to \beta, \beta}{\alpha}$$

As seen in Table 4, the rule is not **sound**, since the second row does not produce a truthful conclusion α.

α	β	$\alpha \to \beta$
0	0	1
0	**1**	**1**
1	0	0
1	**1**	**1**

Table 4: Truth table for the abduction rule

4.6. Equivalence rule

$$\frac{\alpha \equiv \beta}{(\alpha \to \beta) \land (\beta \to \alpha)}$$

$$\frac{(\alpha \to \beta), (\beta \to \alpha)}{\alpha \equiv \beta}$$

5. Conjunctive normal form

Every sentence of propositional logic is logically equivalent to a conjunction of clauses.

A sentence expressed as a conjunction of clauses is said to be in **conjunctive normal form** or **CNF**.

E.g. convert $S_{1,1} \equiv W_{1,2} \lor W_{2,1}$ into **CNF**:

$$S_{1,1} \equiv W_{1,2} \lor W_{2,1}$$
$$(S_{1,1} \to W_{1,2} \lor W_{2,1}) \land (W_{1,2} \lor W_{2,1} \to S_{1,1})$$
$$(\neg S_{1,1} \lor W_{1,2} \lor W_{2,1}) \land (\neg(W_{1,2} \lor W_{2,1}) \lor S_{1,1})$$
$$(\neg S_{1,1} \lor W_{1,2} \lor W_{2,1}) \land ((\neg W_{1,2} \land \neg W_{2,1}) \lor S_{1,1})$$
$$(\neg S_{1,1} \lor W_{1,2} \lor W_{2,1}) \land ((\neg W_{1,2} \lor S_{1,1}) \land (\neg W_{2,1} \lor S_{1,1}))$$

The last line represents **CNF**.

6. Propositional logic weaknesses

- Requires to process large amounts of sentences
- No way to create a generic sentence with variables
- It doesn't allow to efficiently deal with perceptions, with no ability to apply time constraint. E.g. bumping into the wall - cannot limit the agent to do a different action on next time iteration
- Can use only sentences that produce *True* or *False*

7. Predicate logic

- Uses idea that the world is made of objects with relations between them. Relations are either *True* or *False*.
- Has 2 parts:
  1. predicates that define relations
  2. arguments to the predicates, also known as **terms**. Argument types:
     - objects, e.g. **John**
     - relations, e.g. **is a part of**
     - properties, e.g. **is red**
     - functions, e.g. **father(John)**

Predicate logic uses:

- Symbols for defining **constants**($P, A, B, ...$)
- Variables ($x, y, ...$) for defining constant types
- Functions($f(x, y, ...) \to z$) that maps every variable onto function value set. f is called the function constant, $x, y, ...$ are the function's term.
- Predicates make statements about objects that are *True* or *False*($p(x, y, ...) \to \{T, F\}$), where p is the predicate constant, but $x, y, ...$ are the arguments.

With **term** we describe variables, functions and constants.

Complex sentences are mode of atomic sentences:

S is an atomic sentence, as well as

- $\neg S$
- $S \land P$
- $S \lor P$
- $S \to P$
- $S \equiv P$

E.g. complex sentence from atomic ones:
$\text{like}(x, y) \land \text{like}(z, y) \to \neg\ \text{like}(x, z)$

7.1. Quantifiers

$\forall$ - universal quantifier("for all")

$\exists$ - existential quantifier("there exists at least one")

$\exists!$ - there exists only one entity

$\forall x$ - quantifier complex

$\forall x\ p(x)$ - predicate works for all x, e.g.
$\forall x\ \text{like}(x, \text{Mary})$ - everyone likes Mary.

Quantifier domain limits all the variables, e.g.

$\forall x((p(x) \land q(y)) \lor r(x, y))$, where y is a free term.

Examples:

$\forall x(\text{cat}(x) \to \text{animal}(x))$ - every cat is the animal

$\forall x(\text{footballer}(x) \to \text{fast}(x))$ - every footballer is fast

Equivalences:

$\forall x\ p(x) \equiv p(x_1) \land p(x_2) \land ... \land p(x_n)$

$\exists x\ p(x) \equiv p(x_1) \lor p(x_2) \lor ... \lor p(x_n)$

- Quantifier mutual expressibility

$$\neg\exists x\ p(x) \equiv \forall x\ \neg p(x)$$
$$\neg\forall x\ p(x) \equiv \exists x\ \neg p(x)$$

- Free exchange of variables

$$\exists x\ p(x) \equiv \exists y\ p(y)$$
$$\forall x\ p(x) \equiv \forall y\ p(y)$$

- Quantifiers carry-out from parentheses

$$\forall(p(x) \wedge q(x)) \equiv \forall x\ p(x) \wedge \forall x\ q(x)$$
$$\exists(p(x) \wedge q(x)) \equiv \exists x\ p(x) \vee \exists x\ q(x)$$

7.2. Semantics

- **Sentences** are the same as **statements**
- **Interpretation domain** D - prescribes an object from the D domain to each constant:

$$D = \{\text{John}, \text{Paul}, \text{Ringo}, \text{George}\}$$
$$\text{John} = \text{const}$$

- Each **variable** has a non-empty subset of D domain:

$$x - \text{guitarist } \{\text{John}, \text{Paul}, \text{George}\} \subseteq D$$

- Each **function** with the volume N is defined with N objects(arguments) from D domain and the function defines a mapping:

$$f : D^N \to D$$

- Each **predicate** with the volume M defines M arguments from D domain and defines the mapping:

$$p : D^M \to \{T, F\}$$

where T - *True*, F - *False*

7.3. Inference

Inference must be **sound**, i.e. sentences must be truthful in every interpretation, e.g.

$$\forall x(p(x) \vee \neg p(x))$$

is truthful.

7.4. Substitution

In every truthful sentence it is possible to substitute a term, and in the result gather a truthful sentence:

$$\forall x\ p(x) : p(K)$$

where x is variable, K - constant, and $p(K)$ is truthful.

Example:

$$\text{TELL}(\text{KB}, \forall x(\text{student}(x) \to \text{human}(x)))$$
$$\text{TELL}(\text{KB}, \text{student}(\text{John}))$$

With substitution and *MODUS PONENS* we can infer that John is human:

$$\frac{(\text{student}(x) \to \text{human}(x)), \text{student}(\text{John})}{\text{human}(\text{John})}$$

Skolem normal form

It is easy to substitute $\forall$, for $\exists$ we should use **Skolem normal form**.

The simplest Skolem form is when $\exists$ stands alone. It is replaced with the constant(c):

$$\exists x\ p(x) \equiv p(c)$$

With universal quantifier we should apply a function $y = f(x)$ that maps every x onto y. E.g. every child has a father, later can be expressed as a function that defines a relation between father and each child:

Example 1:

$$\forall x \exists y\ \text{father}(x, y)$$
$$\forall x\ \text{father}(x, f(x))$$

Example 2:

Every human has brains:

$$\forall x(\text{human}(x) \to \exists y\ \text{brains}(y) \wedge \text{belongs}(x, y))$$
$$\forall x(\text{human}(x) \to \text{brains}(f(x)) \wedge \text{belongs}(x, f(x)))$$

7.5. Predicates for Wumpus World

- Predicate that defines if one square coordinates are close to the other square:

$$\forall x, y, a, b\ \ \text{close}((x, y), (a, b)) \equiv$$
$$(a, b) \in \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

- Predicate that defines that there is only one Wumpus existing on the map:

$$\exists!\ \text{wumpus}(x)$$
$$\exists\ \text{wumpus}(x) \wedge \forall y\ \text{wumpus}(y) \to (x = y)$$
$$- \text{ if } \exists! \text{ is not allowed}$$

- **Diagnostic laws** describe how observed facts are related to the consequences:

  If breeze is detected, then pit is in the closest squares:

$$\forall x(\text{breeze}(x) \to \exists r(\text{close}(r, x) \wedge \text{pit}(r)))$$

  where x, r are square coordinate tuples.

  If breeze is not detected:

$$\forall s(\neg\ \text{breeze}(s) \to \forall r(\text{close}(r, s) \wedge \neg\ \text{pit}(r))) \equiv$$
$$\forall s(\neg\ \text{breeze}(s) \to \neg\exists r(\text{close}(r, s) \wedge \text{pit}(r)))$$

where s, r are square coordinate tuples.

- **Reason-consequence** laws:

  If square r is a pit, then the closest squares s has a breeze:

  $$\forall r(\text{pit}(r) \rightarrow (\forall s\ \text{close}(r, s) \rightarrow \text{breeze}(s)))$$

- The changes in the world can be described with **result** function:

$$\forall x, s(\text{is}(x, s) \wedge \text{movable}(x) \rightarrow \text{hold}(x, \text{result}(\text{grab}(x, s))))$$

  where x is a movable item, s is the square from which the item was grabbed

- Example of **the axiom of the effect of the action**:

  $$\forall x, s(\neg\ \text{hold}(x, \text{result}(\text{release}(x, s))))$$

  where x is a movable item, s is the starting square where it is allowed to release

- Example of **frame axioms**:

  $$\forall a, x, s(\neg\ \text{hold}(x, s) \wedge (a \neq \text{grab}(x)) \rightarrow \neg\ \text{hold}(x, \text{result}(a, s))$$

  where a is the action, x is a movable item, s is the starting square where it is allowed to release. This predicate describes what happens when *hold* action is not active, i.e. if we do not grab then we do not hold anything.

  $$\forall a, x, s(\text{hold}(x, s) \wedge (a \neq \text{release}(x)) \rightarrow \text{hold}(x, \text{result}(a, s))$$

  This predicate describes that if we do not release, then we certainly should be holding some item.

If we want to describe that some action will be true in later times we can describe it with the following pseudocode:

True later $\equiv$ truthful action that changes state $\vee$ (predicate that already holds the truth $\wedge$ action that does not make an action false)

E.g.

$$\forall a, x, s(\text{hold}(x, \text{result}(a, s)) \equiv \equiv ((a = \text{grab}(x) \wedge \text{is}(x, s) \wedge \text{movable}(x)) \vee \vee (\text{hold}(x, s) \wedge (a \neq \text{release}(x))))$$

where predicate describes that the result of holding an item is that item x is movable, it is in the square s and we have grabbed it. Or we already hold this item and we did not release it yet.

8. Knowledge building for logical agents

Knowledge Base is created by business specialist and knowledge engineer.

The biggest bottleneck is the knowledge acquisition(elicitation).

An example of this problem: **The bear has very small brains(Winny Puhh), therefore it is stupid**. System cannot inference this sentence unless there is a KB that can sequence to this conclusion.

Let's create KB:

1. Puhh is a bear:

   $$\text{bear}(P)$$

2. All bears are animals:

   $$\forall b\ (\text{bear}(b) \rightarrow \text{animal}(b))$$

3. All animals are physical objects:

   $$\forall a\ (\text{animal}(a) \rightarrow \text{object}(a))$$

4. All animals have brains:

   $$\forall a\ (\text{animal}(a) \rightarrow \text{brains}(a))$$

5. Brains are the part of the animal:

   $$\forall a\ (\text{belongs}(\text{brains}(a), a))$$

6. If one physical object belongs to other, then it is also an object:

   $$\forall x, y\ (\text{belongs}(x, y) \wedge \text{object}(y) \rightarrow \text{object}(x))$$

7. Every object has a size K:

   $$\forall x\ (\text{object}(x) \rightarrow \exists K\ (\text{size}(x) = K))$$

8. A relative size of different objects:

   $$\forall v, w\left(\text{relative size}(v, w) \rightarrow \frac{\text{size}(v)}{\text{size}(w)}\right)$$

9. Puhh's brains relative size:

   $$\text{relative size}(\text{brains}(P), \text{brains}(B)) = \text{very small}$$

   where B is a typical bear

From this KB we can now infer that Puhh has very small brains, he is a bear, an animal and he has brains.

7. Uncertain knowledge

By: Alex S.

Apr, 2024

1. Acting under uncertainty

Using propositional logic does not always work as intended. Consider the example:

$$\forall p \quad \text{symptoms}(p, \text{toothache}) \rightarrow \text{disease}(p, \text{cavity})$$

However, it is not always the case that toothache is received from a cavity, let's describe other problems as well:

$$\forall p \quad \text{symptoms}(p, \text{toothache}) \rightarrow \text{disease}(p, \text{cavity}) \\ \vee \text{disease}(p, \text{gum problems}) \vee \ldots$$

There are several problems:

1. There can be lots of different problems, and we might miss some condition.
2. We cannot use disjunction for inference. We need conjugate form.
3. We are lazy to describe everything.

Therefore we use **decision theory** to describe rational decisions for the agent:

Decision theory = probability theory + utility theory

With this theory, the agent is rational when it chooses the action that gives the best result which is the arithmetic average of all available actions.

2. Probability theory

- Described by **random variables**, e.g. X
- Discrete random variable: $P(\text{Weather}) = (0.7; 0.3)$ or $P(\text{Weather} = \text{sunny}) = 0.7$ and $P(\text{Weather} = \text{rainy}) = 0.3$. These types always have a **domain** - the set of possible values, in this case Weather $\in \{\text{sunny}, \text{rainy}\}$ or Dice $\in \{1 \ldots 6\}$
- Boolean random variable: $P(\text{cavity})$ or $P(\neg \text{ cavity})$. It has a domain of $\{\text{true}, \text{false}\}$
- Continuous random variables cannot be defined with the vector of values. Instead it uses a **probability density function**:

$$P(\text{Temp} = x) = \text{Uniform}_{[18,26]}(x) \\ = \begin{cases} \frac{1}{8} \text{ if } 18 \leq x \leq 26 \\ 0 \text{ otherwise} \end{cases}$$

It means that in 100% the temperature is in the interval of 18 - 26 Celsius.

2.1. Syntax:

- $P(A)$ is a **prior** or **unconditional probability**
- $P(A|B)$ is a **conditional** or **posterior** probability where A is given by B(evidence). In the agent world B might be a measurement from the sensor.
- **Conditional probability** product rule:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}, \text{if } P(B) > 0 \\ P(A \wedge B) = P(A|B) \cdot P(B) \\ P(A \wedge B) = P(B|A) \cdot P(A)$$

2.2. Axioms

- **Probability model**:

$0 \leq P(\omega) \leq 1$ for every ω and $\sum_{\omega \in \Omega} P(\omega) = 1$, where ω is some world, but Ω is a set of all possible worlds called the **sample space**.

- $P(\text{true}) = 1, P(\text{false}) = 0$
- **Kolmogorov's axiom**: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$, we can derive $P(A \wedge \neg A) = P(A) + P(\neg A) = 1$

2.3. A full joint distribution example

	Toothache	¬ Toothache	$\sum$
Cavity	0.04	0.06	0.1
¬ Cavity	0.01	0.89	0.9
$\sum$	0.05	0.95	

- $P(\text{Cavity}) = P(\text{Toothache}) + P(\neg \text{ Toothache}) = 0.1$

It is also called a **marginal probability**. General form:

$$P(Y) = \sum_{z \in Z} P(Y, z)$$

In the example;

$$P(\text{Cavity}) = \sum_{z \in \{\text{Toothache}\}} P(\text{Cavity}, z)$$

- $P(\text{Cavity} \vee \text{Toothache}) = \\ = P(\text{Cavity}) + P(\text{Toothache}) - \\ - P(\text{Cavity} \wedge \text{Toothache}) = \\ = 0.1 + 0.05 - 0.04 = 0.11$
- $P(\text{Cavity} \mid \text{Toothache}) = \\ = \frac{P(\text{Cavity} \wedge \text{Toothache})}{P(\text{Toothache})} = \\ = \frac{0.04}{0.05} = \frac{4}{5} = 0.8$

3. Bayes' rule and its use

Previously we defined a **product rule**, from which Bayes' rule is derived:

$$P(A \wedge B) = P(A|B) \cdot P(B)$$
$$P(A \wedge B) = P(B|A) \cdot P(A)$$
$$P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$$
$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

- It is used for diagnostic laws
- It is also used for probabilistic inference mechanisms

Bayes' rule is used for **uncertain knowledge representation** - *Bayes' network*:

3.1. Bayes' network

- It is a directed acyclic graph(DAG)
- The node is a random variable
- The node has a distribution table
- Edge X to Y represents that X influences Y

An example:

- An alarm system in the house
- It can be activated if earthquake happens
- It is activated if burglary happens
- John or Mary can call if they supposedly heard the alarm

The graph can be represented as in the image Figure 1. The graph represents a knowledge base.

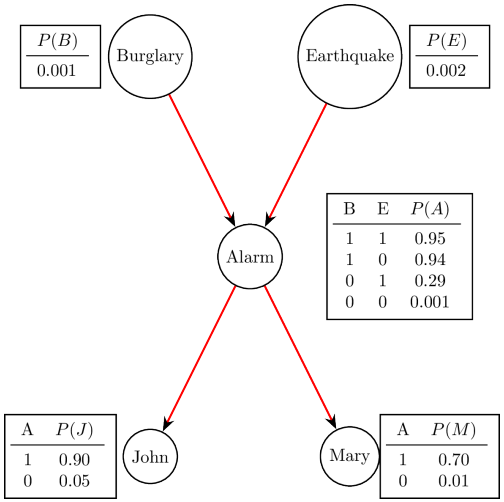


Figure 1: Bayes' network example

Now we can use it for querying probabilities:

$$P(X_i \mid \text{parents}(X_i))$$

The value of the entry can be calculated as this:

$$P(X_1 = x_1 \wedge ... \wedge X_n = x_n) = P(x_1, ..., x_n) = \Pi_{i=1}^{n} P(x_i \mid \text{parents}(x_i))$$

Examples:

$$P(A, J, M, \neg B, \neg E) = P(A \wedge J \wedge M \wedge \neg B \wedge \neg E) =$$
$$= P(J \mid A) \cdot P(M \mid A) \cdot P(A \mid \neg B \wedge \neg E) \cdot P(\neg E) \cdot P(\neg B) =$$
$$= 0.9 \cdot 0.7 \cdot 0.001 \cdot 0.999 \cdot 0.998 = 0.000628$$

- $$P(A \wedge B \wedge E) =$$
$$= P(A \mid B \wedge E) \cdot P(B) \cdot P(E) =$$
$$= 0.95 \cdot 0.001 \cdot 0.002 = 0.0000019$$

- $$P(J, \neg M, \neg A, \neg B, \neg E) =$$
$$= P(J \mid \neg A) \cdot P(\neg M \mid \neg A) \cdot P(\neg A \mid \neg B \wedge \neg E) \cdot P(\neg B) \cdot P(\neg E) =$$
$$= 0.05 \cdot 0.99 \cdot 0.999 \cdot 0.999 \cdot 0.998 = 0.0493$$

4. Chain rule

The Bayesian network is a correct representation of the domain only if each node is conditionally independent of its other predecessors in the node ordering, given its parents:

$$P(x_1, ..., x_n) = P(x_n \mid x_{n-1}, ..., x_1) P(x_{n-1}, ..., x_1)$$

$$P(x_1, ..., x_n) = P(x_n | x_{n-1}, ..., x_1) \cdot$$
$$\cdot P(x_{n-1} | x_{n-2}, ..., x_1) \cdot ... \cdot P(x_2 | x_1) \cdot P(x_1) =$$
$$= \Pi_{i=1}^{n} P(x_i | x_{i-1}, ..., x_1)$$

Every node in the network is equal to:

$$P(X_i | X_{i-1}, ..., X_i) = P(X_i | \text{Parents}(X_i))$$
$$\text{Parents}(X_i) \subseteq \{X_{i-1}, ..., X_1\}$$

5. Conditional independence relations in Bayesian networks

We have provided a "numerical" semantics for Bayesian networks in terms of the representation of the full joint distribution, as in equation above.

Now let's define topological semantics that specifies the conditional independence relationships encoded by the graph structure. From which later we can derive numerical semantics.

1. A node X is conditionally independent of its non-descendants(e.g. Z_{ij}s), given its parents (the U_is shown in the dashed area).

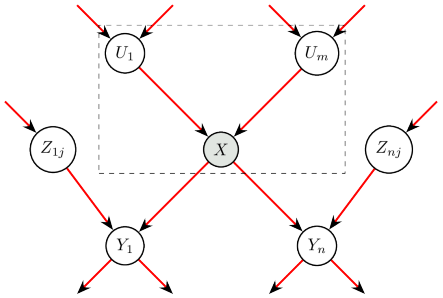


Figure 2: X - conditionally independent node, U_i - parents, $Z_{ij}s$ - non-descents, Y_i - descents

2. A node X is conditionally independent of all other nodes in the network given its **Markov blanket** (the dashed area).

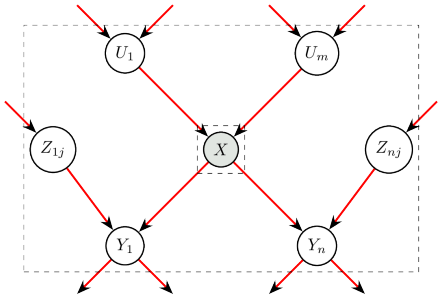


Figure 3: X - conditionally independent node, U_i - parents, $Z_{ij}s$ - non-descents, Y_i - descents

5.1. D-separation

There is also a general topological criterion called **d-separation** for deciding whether a set of nodes X is conditionally independent of another set Y, given a third set Z.

A set of nodes E(evidence) d-separate X and Y, if every non-directed path $X \rightarrow Y$ is blocked by the E set(There might be other nodes in the path as well). The path is blocked by the set E, if there is a node z, for which one out of 3 conditions is met:

1. $z \in E \wedge$ (z has one input and output)

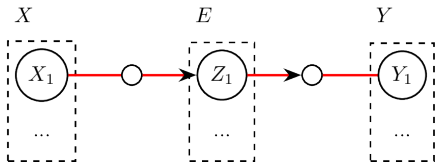


Figure 4: Z_1 blocks the path to the Y_1 for X_1

2. $z \in E \wedge$ (z has two outputs)

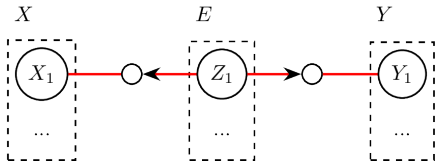


Figure 5: Z_1 blocks the path to the Y_1 for X_1

3. $z \wedge \text{children}(z) \notin E \wedge$ (z has two inputs)

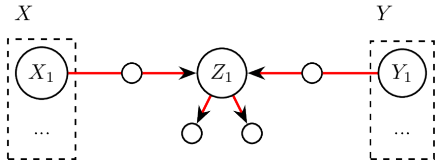


Figure 6: $Z_1 \notin E$ blocks the path to the Y_1 for X_1

5.2. D-separation example

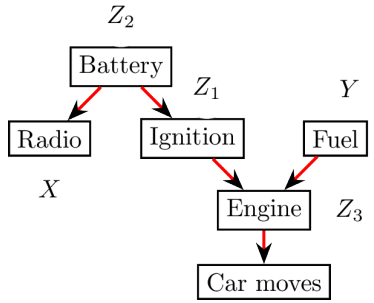


Figure 7: An example of d-separation in the Bayesian network

1. Evidence Z_1 blocks $X \rightarrow Y$ in a way described in the first D-separation rule.
2. Evidence Z_2 blocks $X \rightarrow Y$ in a way described in the second D-separation rule.
3. If there is no evidence at all, then X is independent from Y as in the third D-separation rule.
4. With evidence Z_3 it turns out that X is dependent on Y, because we have an evidence and there is no rule for a separation.

6. Inference types

- Every node in the network can be used for querying and getting a proof.

In all images Q is query, E is evidence

1. **Diagnostic inference** - from conclusions to consequences.

Figure 8: The query is $P(B|J)$, B - burglary, J - John

2. **Causal inference** - from consequences to conclusions.

Figure 9: The query is $P(J|B)$, B - burglary, J - John calls

3. **Midcausal inference** - from common conclusions get consequences.

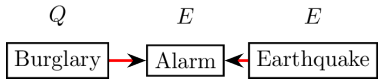


Figure 10: The query is $P(B|A \wedge K)$, B - burglary, A - Alarm, K - earthquake

4. **Mixed inference** - two or 3 previous inference combination.

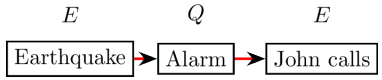


Figure 11: The query is $P(A|J \wedge \neg K)$, J - John calls, A - Alarm, K - earthquake. The combination of $1 + 2$ inference

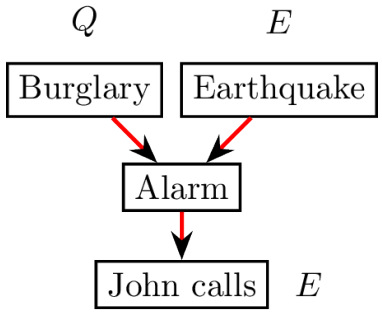


Figure 12: The query is $P(B|K \wedge J)$, J - John calls, B - Burglary, K - earthquake. The combination of $1 + 3$ inference

7. Probability system modelling steps

1. Knowledge engineer should decide what factors should be modelled and which could not directly affect the model
2. It should be decided about the random variable dictionary - is it discrete, continuous or boolean.
3. Need to decide the Bayesian network's topology. Find the qualitative or quantitative relations between random variables.
4. Encode the specific cases.
5. Test the network by querying. In some cases a **sensitivity analysis** can occur which helps to check how robust the values are.

8. Simple decision making agents

By: Alex S.

Apr, 2024

1. Making simple decisions

- The agent does not plan, but rather executes the action and sees the immediate outcome.
- Simple decisions can be done in the episodic environment.
- Complex decisions require a long chain of actions and only after them the outcome will be visible, e.g. in chess.
- The probability of the outcome s', given evidence observations e, is written:

$$P(\text{result}(a) = s' | a, e),$$

where a on the right-hand side of the conditioning bar stands for the event that action a is executed.[1] And $\text{result}(a)$ is a *random variable* whose value are the possible outcome states, since we omit the current state.

The agent's preferences are captured by a **utility function**($U(s)$) which assigns a single number to express the desirability of a state.

The **expected utility** of an action given the evidence, $\text{EU}(a|e)$, is just the average utility value of the outcomes, weighted by the probability that the outcome occurs:

$$\text{EU}(a|e) = \sum_{s'} P(\text{result}(a) = s' | a, e) \cdot U(s')$$

The principle of **maximum expected utility**(MEU) says that a rational agent should choose the action that maximizes the agent's expected utility:

$$\text{action} = \underset{a}{\text{argmax}}\ \text{EU}(a|e)$$

2. Constraints on rational preferences

Agent's preferences can be described with the following notation:

- $A \succ B$ the agent prefers A over B.
- $A \sim B$ the agent is indifferent between A and B.
- $A \succcurlyeq B$ the agent prefers A over B or is indifferent between them.

A and B can be states of the world, but can be expressed as uncertainty. Set of outcomes for each action is a **lottery**, each action as a ticket. A lottery L with possible outcomes $S_1, ..., S_n$ with probabilities $p_1, ..., p_n$:

$$L = [A, 1]$$
$$L = [p, A, (1-p), B]$$
$$L = [p_1, S_1; ...p_n, S_n]$$

3. Constraints or axioms on preferences

1. **Orderability**: the agent cannot avoid the decision making, he must choose either the first or the second option.

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

2. **Transitivity**:

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

For example, $A \succ B \succ C \succ A$ does not make sense. Since the graph creates the cycle where agent cannot decide on one preference.

3. **Continuity**: If some lottery B is between A and C, them there is some probability p for which the rational agent will be indifferent between getting B for sure and the lottery that yields A with probability p and C with probability $1-p$. (The Monty Hall problem)

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

4. **Substitutability**: If an agent is indifferent between two lotteries A and B, then the agent is indifferent between two more complex lotteries that are the same except that B is substituted for A in one of them. Long story short: it does not matter what lottery to choose. This also holds for $\succ$:

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

5. **Monotonicity**: Suppose two lotteries have the same two possible outcomes, A and B. If an agent prefers A to B, then the agent must prefer the lottery that has a higher probability for A:

$$A \succ B \rightarrow (p > q \equiv [p, A; 1-p, B] \succ [q, A; 1-q, B])$$

6. **Decomposability**: Compound lotteries can be reduced to simpler ones using the laws of probability. This has been called the *"no fun in gambling"* rule because it says that two

[1] Classical decision theory leaves the current state S_0 implicit, but we could make it explicit by writing: $P(\text{result}(a) = s' | a, e) = \sum_s P(\text{result}(s, a) = s | a) \cdot P(S_0 = s | e)$

consecutive lotteries can be compressed into a single equivalent lottery:

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

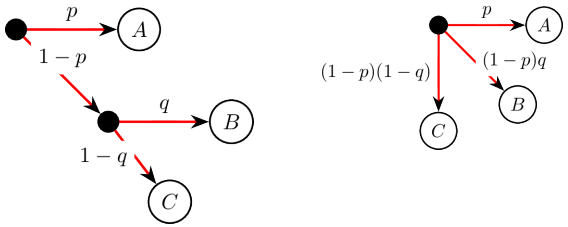


Figure 1: **On the left**: A compound lottery. **On the right**: A decomposed lottery

4. Preferences lead to utility

- **Existence of Utility Function**: If an agent's preferences obey the axioms of utility, then there exists a function U such that:

$$U(A) > U(B) \equiv A \succ B$$
$$U(A) = U(B) \equiv A \sim B$$

- **Expected Utility of a Lottery**: The utility of a lottery is the sum of the probability of each outcome times the utility of that outcome:

$$U([p_1, S_1; ...; p_n, S_n]) = \sum_i p_i \cdot U(S_i) = \text{EU}(a|e)$$

The preceding theorems establish that a utility function exists for *any rational agent*, but they do not establish that it is *unique*.

5. Agent's utility function

- The best possible prize: $U(S) = u_\top$
- The worst possible prize: $U(S) = u_\bot$
- Normalized utilities: $u_\top = 1, u_\bot = 0$
- Standard lottery: $[p, u_\top; (1-p), u_\bot]$

6. Some paradoxes

It will usually be the case that an agent prefers more money to less, all other things being equal. We say that the agent exhibits a **monotonic preference** for more money.

6.1. The utility of money

Suppose a lottery: you can take a \$1 million or flip a coin to get \$2.5 million. Assuming the coin is fair we can write the **expected monetary value(EMV)**:

$$\text{EU(Accept)} = \frac{1}{2}U(S_k) + \frac{1}{2}U(S_{k+2.5})$$
$$\text{EU(Decline)} = U(S_{k+1})$$

where S_k is the current wealth in \$ millions. Most people will choose the guaranteed option, even though it is not the most optimal(the first option EU is \$1.25 million, the second one is \$1 million). However, for a billionaire the gamble can be an option to do.

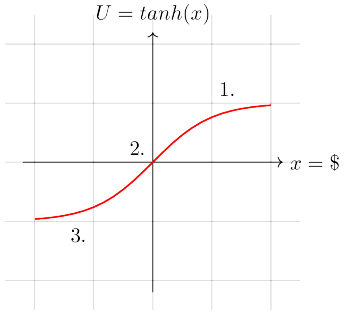


Figure 2: Not sure where this curve comes from and it is empirical curve, but it resembles $\tanh(x)$ function, but not exactly. Each number describes risk assessment.

In Figure 2 Grayson[2] or someone has created an empirical view on people behaviors depending on their monetary status:

1. **Risk-averse** - they prefer a sure thing than a gamble.
2. **Risk-neutral** - the agent gambles small sums to assess the probabilities.
3. **Risk-seeking** - the agent rather to play the gamble.

The difference between the Expected Monetary Value(EMV) of a lottery and its certainty equivalent is called the **insurance premium**, e.g. in the lottery above insurance premium $= 2.5 - 1 =$ \$1.5 million.

6.2. Human judgment and irrationality

The evidence suggests that humans are "predictably irrational".

The best-known problem is the Allais paradox (Allais, 1953). People are given a choice between lotteries A and B and then between C and D, which have the following configuration:

Lottery	Conditions	EMV	Max choice	Human choice
A	80% \$4k	\$3200	$A \succ B$	$B \succ A$
B	100% \$3k	\$3000		
C	20% \$4k	\$800	$C \succ D$	$C \succ D$
D	25% \$3k	\$750		

[2] Grayson, C. J. (1960). Decisions under uncer- tainty: Drilling decisions by oil and gas operators. Tech. rep., Division of Research, Harvard Business School.

As seen people do not always choose the biggest EMV value. One explanation for the apparently irrational preferences is the **certainty effect** (Kahneman and Tversky, 1979): people are strongly attracted to gains that are certain.

6.3. Ellsberg paradox

Here the prizes are fixed, but the probabilities are underconstrained. Your payoff will depend on the color of a ball chosen from an urn. You are told that the urn contains 1/3 red balls, and 2/3 either black or yellow balls, but you don't know how many black and how many yellow. Again, you are asked whether you prefer lottery A or B; and then C or D:

Lottery	Conditions	Probability	Human choice
A	\$100 for a red ball	$\frac{1}{3}$	$A \succ B$
B	\$100 for a black ball	$\left[0; \frac{2}{3}\right]$	
C	\$100 for a red or yellow ball	$\left[\frac{1}{3}; \frac{3}{3}\right]$	$D \succ C$
D	\$100 for a black or yellow ball	$\frac{2}{3}$	

Most people elect the known probability rather than the unknown unknowns.

7. Dominance

If there is an attribute set $X = (X_1, ..., X_n)$ then we can determine the dominance in Figure 3.

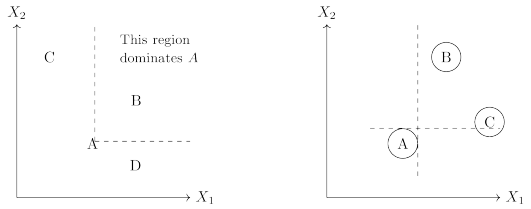


Figure 3: Right: Deterministic, option A is strictly dominated by B, but not by C or D. Left: Uncertain: A is strictly dominated by B, but not by C

8. Preference structure and multiattribute utility

Suppose we have n attributes, each of which has m distinct possible values. To specify the complete utility function $U(x_1, ..., x_n)$, we need m^n values in the worst case. Now, the worst case corresponds to a situation in which the agent's preferences have no regularity at all.

Representation theorems show that an agent with a certain kind of preference structure has a utility function:

$$U(x_1, ..., x_n) = F\left[f_1(x_1), ..., f_{n(x_n)}\right]$$

where F is, we hope, a simple function such as addition.

8.1. Preferences without uncertainty

For deterministic environments the agent has a value function $V(x_1, ..., x_n)$; the aim is to represent this function concisely.

Two attributes X_1 and X_2 are preferentially independent of a third attribute X_3 if the preference between outcomes (x_1, x_2, x_3) and (x_1', x_2', x_3) does not depend on the particular value x_3 for attribute X_3.

Consider the example of minimax in Figure 4. If node A value is changed to 40 instead of 4, it does not matter and the root value stays the same. The attributes exhibit **mutual preferential independence**.

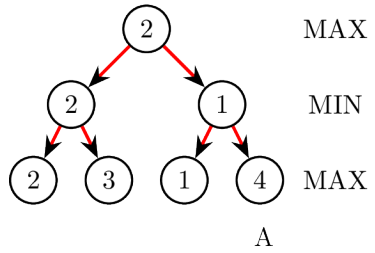


Figure 4: Minimax example

If attributes $X_1, ..., X_n$ are mutually preferentially independent, then the agent's preference behavior can be described as maximizing the **additive value function**:

$$V(x_1, ..., x_n) = \sum_i V_i(x_i)$$

The function with the state input:

$$V(S) = \sum_i V_i(x_i(S))$$

Need to remember that the function attributes must be expressed as the one unit of measurement and that the attributes most often have some sort of weight:

$$V(S) = \sum_i k_i \cdot V_i(x_i(S))$$

8.2. Preferences with uncertainty

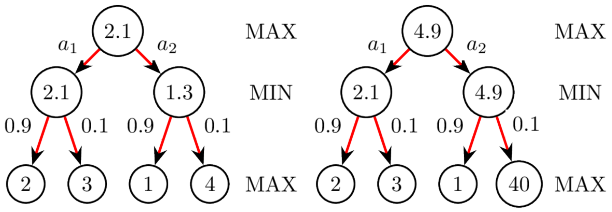
A set of attributes is **mutually utility independent** (MUI) if each of its subsets is utility-independent of the remaining attributes.

MUI implies that the agent's behavior can be described using a multiplicative utility function. Example of three attributes:

$$U = k_1U_1 + k_2U_2 + k_3U_3 + \\ +k_1k_2U_1U_2 + k_2k_3U_2U_3 + k_3k_1U_3U_1 + \\ +k_1k_2k_3U_1U_2U_3$$

For more mathematics reference Keeney K.L., Raiffa H. (1976) Decisions with multiple objectivesL preferences and value trade-off.

Example of uncertain preferences can be presented within the minimax example in Figure 5. Changing the leaf node from 4 to 40, does change the root result from 2.1 to 4.9.



,

Figure 5: Minimax with uncertainty example

9. Decision networks

Figure 6 shows a decision network for the airport siting problem. It illustrates the three types of nodes used:

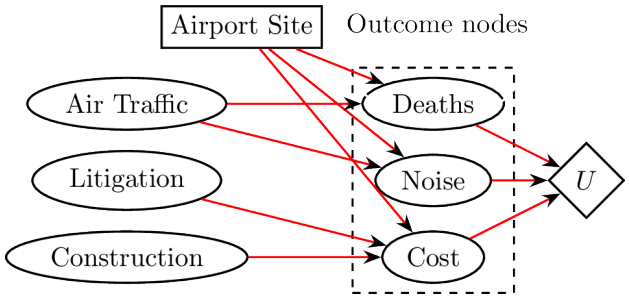


Figure 6: A simple decision network for the airport-siting problem

1. **Chance nodes**(ellipses) represents random variables. The agent could be uncertain about the construction cost, the level of air traffic and the potential for litigation, and the Deaths, Noise, and total Cost variables, each of which also depends on the site chosen.
2. **Decision node**(rectangles) represents points where the decision maker has a choice of actions.
3. **Utility node**(diamonds) represents the agent's utility function.

In Figure 6 outcome nodes can be omitted and the graph will be simplified.

Need to take into account:

- Decision maker forms the list of priorities and preferences
- Decision analyst creates the structure of priorities

Evaluating decision networks:

1. Set the evidence variables for the current state.
2. For each possible value of the decision node:
   1. Set the decision node to that value.
   2. Calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference algorithm.
   3. Calculate the resulting utility for the action.
3. Return the action with the highest utility.

10. Decision analysis methodology

1. Defines problem's scope
   1. What are the possible actions
   2. What are the possible states and outcomes
   3. What random variables affect the state
   4. What is the input data to calculate probabilities
2. Defines network's topology
3. Defines probabilities: $P(\text{effect}|\text{reason})$
4. Defines an utility function
5. Inputs an available example
6. Checks the decision network's actions
7. Tries to get a new example (cost of getting knowledge vs accuracy)
8. Sensitivity analysis is done

9. Complex decision making agents

By: Alex S.

Apr, 2024

1. Making complex decisions

- Complex decisions represent a sequence of decisions where the result is not known by doing one action.

Let's imagine a new world in Figure 1.

- Board is 4 x 3
- Square (4, 2) has a reward of –1
- Square (4, 3) has a reward of 1
- Square (2, 2) has an obstacle
- Square (1, 1) is a starting position

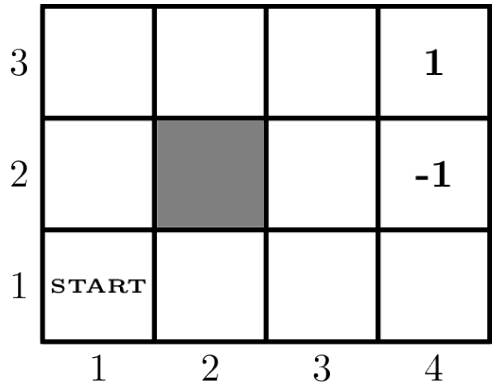


Figure 1: Environment representation

1.1. Deterministic environment

Let's imagine that the agent has 2 actions to plan ahead. The agent has complete information about the environment. For being in every state, the agent receives -0.04 points or receives points from (4, 2) and (4, 3) squares.

Let's imagine that the agent is in the square (3, 2). Then the decision tree will look like this(Figure 2).

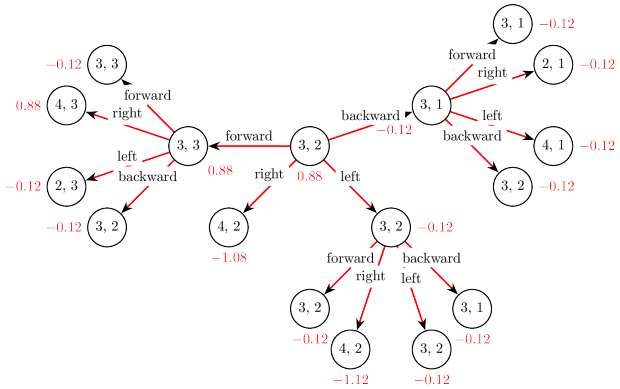


Figure 2: Deterministic environment representation

As can be seen in Figure 2, the agent can easily obtain the optimal path $(3, 2) \rightarrow (3, 3) \rightarrow (4, 3)$. However, in real life the tree is much bigger than presented here.

1.2. Stochastic environment

Stochastic environment is more complex and requires a model for the agent to obtain. For the example, let's obtain that the environment's model is the following:

1. The agent might go to the planned location with the probability of 80%
2. The agent might go to the perpendicular directions with the probability 10% each

This model can be represented as in Figure 3.

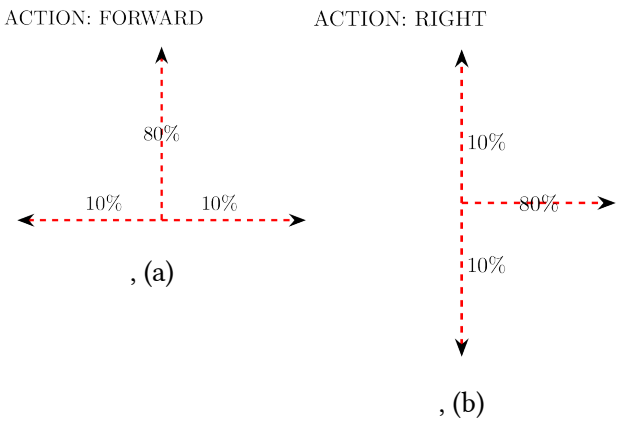


Figure 3: Move models. (a) for the forward action, (b) for the right action

If we consider that the agent can do only a pair of actions sequentially, for example, (FORWARD, RIGHT), then we can construct a tree with all consequences. There are going to be $2^4 = 16$ trees overall. After that we can maximize our utility by choosing the tree with maximum value.

In Figure 4 is shown a tree for the action pair (FORWARD, RIGHT). Note that now we have only 3 actions due to the model. We cannot move backwards.

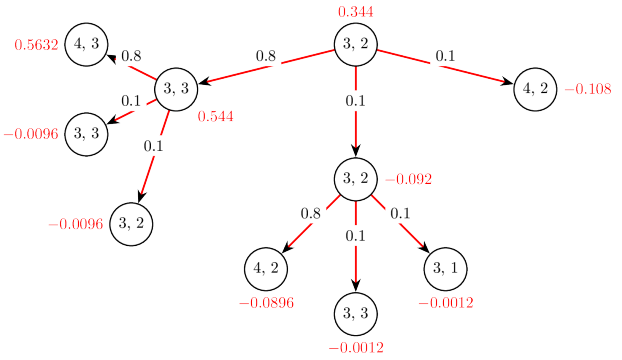


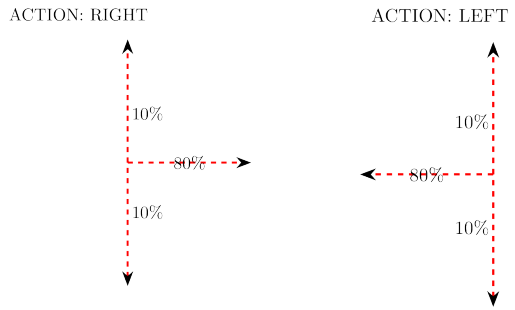
Figure 4: Stochastic environment representation

To calculate the final value of leaf nodes(v_{leaf}) we have to multiply a prior value(v) with the parents' probabilities(p):

$$v_{\text{leaf}} = \Pi_{i \in \text{ parents(leaf)}} p_i \cdot v$$

For example, the path $(3,2) \to (3,3) \to (4,3)$ value is $0.8 \cdot 0.8 \cdot (1 - 0.12) = 0.5632$. After the value for each node is calculated, it is summed to get the overall value of the root. In this case $v_{\text{root}} = 0.344$.

Other example is the model of the actions (RIGHT, LEFT) in Figure 5.



(a) (b)

Figure 5: Move models. (a) for the right action, (b) for the left action

The decision tree of this model looks like in Figure 6. In this case the sequence is not optimal. It produces negative value at the root($v_{\text{root}} = -0.888$).

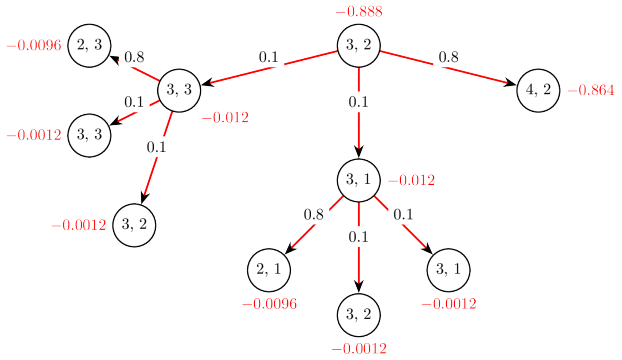


Figure 6: Second sequence stochastic environment representation

After all trees are constructed, we can make a decision on where to go(Figure 7). It turns out that the first sequence is the best one to choose.

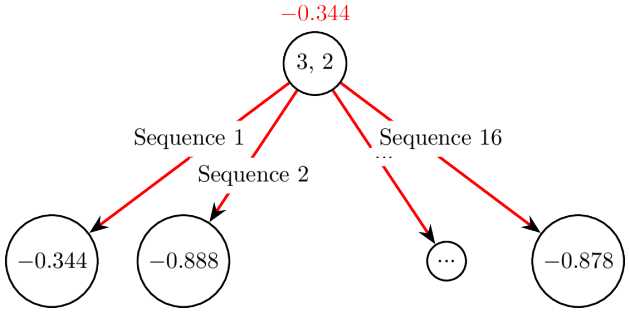


Figure 7: Final decision tree of stochastic environment where the best sequence is chosen.

2. Markovian decision process

Transition model (or just "model") describes the outcome of each action in each state. The outcome is stochastic, so we write $P(s' \mid s, a)$ to denote the probability of reaching state s' if action a is done in state s. We will assume that transitions are **Markovian**.

Because the decision problem is sequential, the utility function will depend on a sequence of states — an **environment history** — rather than on a single state.

A sequential decision problem for a fully observable, stochastic environment with a **Markovian** transition model and additive rewards is called a **Markov decision process(MDP)**. It is defined by:

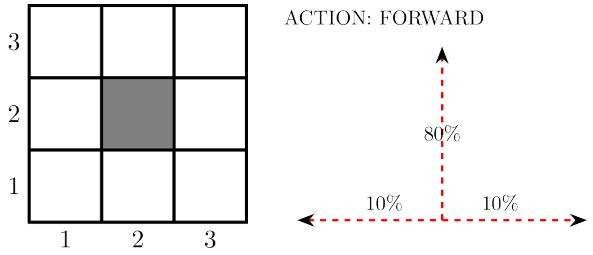
1. Initial state s_0, e.g. $s_0 = (1,1)$
2. A set of actions in each state $A(s)$
3. a reward function $R(s)$, e.g. in the example:

$$\begin{cases} \pm 1 \text{ if terminal state} \\ -0.04 \text{ else} \end{cases}$$

4. A transition model $P(s' \mid s, a)$ or M_{ij}

3. A transition model

Let's consider a simpler example of the game in the first chapter. We need to create a transition model M^{forward} that defines all probabilities transitioning from one state to another.



,

,

Figure 8: A simpler environment and a move model

Then $M^{\text{forward}} =$

	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, 3)	(3, 1)
(1, 1)	0.1	0.8	0	0.1	0	0
(1, 2)	0	0.2	0.8	0	0	0
(1, 3)	0	0	0.9	0	0.1	0
(2, 1)	0.1	0	0	0.8	0	0.1
(2, 3)	0	0	0.1	0	0.8	0.1

Note: 2 columns were cut to save space. The notation of coordinates: (column, row).

4. Policy

A solution must specify what the agent should do for any state that the agent might reach. A solution of this kind is called a **policy**. In other words a state-action mapping.

Policy is denoted by π and $\pi(s)$ is the action recommended by the policy π for state s.

Optimal policy(denoted by π^*) is a policy that yields the highest expected utility(EU):

$$\text{EU}(a|e) = \sum_i P(\text{result}(a_i) = s' \mid a_i, e) \cdot U(s')$$

Policy provides a balance between a risk and a reward.

5. Finding optimal policy

Our analysis draws on **multiattribute utility theory** and we use an environment history to measure a sum of rewards: $U_h(s_0, s_1, ..., s_n)$

There could be:

1. A **finite horizon** for decision making. It means that there is a fixed time N after which nothing matters - the game is over. Thus

   $$U_h(s_0, s_1, ..., s_{N+k}) = U_h(s_0, s_1, ..., s_N)$$

   for all $k > 0$.

   The optimal policy with the finite horizon is **non-stationary**, i.e. N value changes the behavior of the policy.

2. An **infinite horizon** for decision making - no time or step limit.

   The optimal policy with **infinite horizon** is **stationary**.

Stationarity for preferences means the following: if two state sequences $(s_0, s_1, s_2, ...)$ and $(s'_0, s'_1, s'_2,)$ begin with the same state (i.e., $s_0 = s'_0$).

Stationarity is a fairly innocuous-looking assumption with very strong consequences: it turns out that under stationarity there are just two coherent ways to assign utilities to sequences:

1. **Additive rewards**: The utility of a state sequence is

   $$U_h(s_0, s_1, s_2, ...) = R(s_0) + R(s_1) + R(s_2) + ...$$

2. **Discounted rewards**:

   $$U_h(s_0, s_1, s_2, ...) = R(s_0) + \alpha R(s_1) + \alpha^2 R(s_2) + ...$$

   where $0 < \alpha < 1$ is a discount factor. It describes the preference of an agent for current rewards over future rewards. It is equivalent to an interest rate of $\frac{1}{\alpha} - 1$

With discounted rewards, the utility of an infinite sequence is finite. In fact, if $\alpha < 1$ and rewards are bounded by $\pm R_{\max}$, we have:

$$U_h(s_0, s_1, s_2, ...) = \sum_{t=0}^{\infty} \alpha^t R(s_t) \leq \sum_{t=0}^{\infty} \alpha^t R_{\max} = \frac{R_{\max}}{1 - \alpha}$$

Using the standard formula for the sum of an infinite geometric series.

If the environment contains terminal states and if the agent is guaranteed to get to one eventually, then we will never need to compare infinite sequences.

The utility of a given state sequence is the sum of discounted rewards obtained during the sequence, we can compare policies by comparing the expected utilities obtained when executing them. The expected utility is:

$$U^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \alpha^t \cdot R(S_t)\right]$$

where $U^\pi(s)$ is the utility on the state s using a specified policy π, α - discount factor, S - state sequence, e.g. $S_0 = s$ - the current state.

Note: $R(s)$ is the "short term" reward for being in s, whereas $U(s)$ is the "long term" total reward from s onward.

Out of all policies the agent could choose one best policy:

$$\pi^*(s) = \underset{\pi}{\text{argmax}}\ U^\pi(s)$$

Remember that π^*_s is a policy, so it recommends an action for every state; its connection with s in particular is that it's an optimal policy when s is the starting state.

The utility function $U(s)$ allows the agent to select actions by using the principle of maximum expected utility:

$$\pi_s^* = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) \cdot U(s')$$

Note: From the transitional model M_{ij} we can imagine that $i = s$ and $j = s'$, therefore there is a transition $M_{i=s, j=s'}$

6. Finding optimal policies: Bellman equation

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action:

$$U(s) = R(s) + \alpha \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s')$$

Let us look at one of the Bellman equations for 4x3 world for the state (1, 1):

$$\begin{aligned} U(1,1) = -0.04 + \\ +\alpha \cdot \max[&0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \text{(Up)} \\ &0.9U(1,1) + 0.1U(1,2), \text{(Left)} \\ &0.9U(1,1) + 0.1U(2,1), \text{(Down)} \\ &0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)] \text{ (Right)} \end{aligned}$$

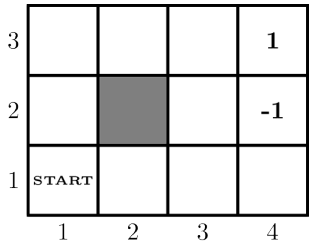


Figure 9: A reference to the board

7. The value iteration algorithm

NEED TO COME BACK TO THIS PLACE AND ADD ADDITIONAL COMMENTS FROM THE BOOK!!! page 652

$$U_{i+1}(s) = R(s) + \alpha \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U_i(s')$$

where U_{i+1} is the utility on the next iteration

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) \cdot U(s')$$

Algorithm(formal description from the book):

7.1. Example:

- $\alpha = 1$
- Every state has $\forall s \; U(s) = 0$

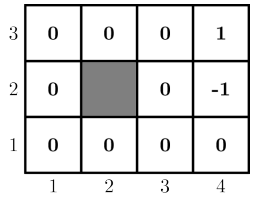


Figure 10: Starting state

FIRST ITERATION:

1. $U(3,3)$:

(Up): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 1 = 0.06$

(Right): $-0.04 + 0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 = 0.76$

(Left): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.04$

(Down): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 1 = 0.06$

$\Rightarrow \max U(3,3) = 0.76$

$\Rightarrow \pi^* = \text{Right}$

2. $U(3,2)$:

(Up): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot (-1) = -0.14$

(Right): $-0.04 + 0.8 \cdot (-1) + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.84$

(Left): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.04$

(Down): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot (-1) + 0.1 \cdot 0 = -0.14$

$\Rightarrow \max U(3,2) = -0.04$

$\Rightarrow \pi^* = \text{Left}$

3. $U(4,1)$:

(Up): $-0.04 + 0.8 \cdot (-1) + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.84$

(Right): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot (-1) + 0.1 \cdot 0 = -0.14$

(Left): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot (-1) + 0.1 \cdot 0 = -0.14$

(Down): $-0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.04$

$\Rightarrow \max U(4,1) = -0.04$

$\Rightarrow \pi^* = \text{Down}$

Other positions produce the value of $U(s) = -0.04$

The final results after the first iteration in Figure 11.

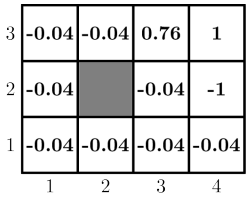


Figure 11: The values after the first iteration

SECOND ITERATION:

1. $U(3,1)$: **NEED TO CHECK RESULTS HERE!**

(Up): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-0.04) + 0.1 \cdot (-0.04) = -0.08$

(Right): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-0.04) + 0.1 \cdot (-0.04) = -0.08$

(Left): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-0.04) + 0.1 \cdot (-0.04) = -0.08$

(Down): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-0.04) + 0.1 \cdot (-0.04) = -0.08$

$\Rightarrow \max U(4,1) = -0.08$

$\Rightarrow \pi^* = ?$

2. $U(3,3)$:

(Up): $-0.04 + 0.8 \cdot 0.76 + 0.1 \cdot (-0.04) + 0.1 \cdot 1 = 0.644$

(Right): $-0.04 + 0.8 \cdot 1 + 0.1 \cdot 0.76 + 0.1 \cdot (-0.04) = 0.832$

(Left): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 0.76 + 0.1 \cdot (-0.04) = 0$

(Down): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 1 + 0.1 \cdot (-0.04) = 0.024$

$\Rightarrow \max U(3,3) = 0.832$

$\Rightarrow \pi^* =$ Right

3. $U(3,2)$:

(Up): $-0.04 + 0.8 \cdot 0.76 + 0.1 \cdot (-0.04) + 0.1 \cdot (-1) = 0.464$

(Right): $-0.04 + 0.8 \cdot (-1) + 0.1 \cdot 0.76 + 0.1 \cdot (-0.04) = 0.768$

(Left): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 0.76 + 0.1 \cdot (-0.04) = 0$

(Down): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-0.04) + 0.1 \cdot (-1) = 0.176$

$\Rightarrow \max U(3,2) = 0.464$

$\Rightarrow \pi^* =$ Up

Other positions produce the value of $U(s) = -0.08$

The final results after the first iteration in Figure 12.

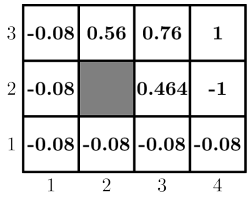


Figure 12: The values after the second iteration

The same actions are done for all states. The convergence can be received after 18 iterations in Figure 13

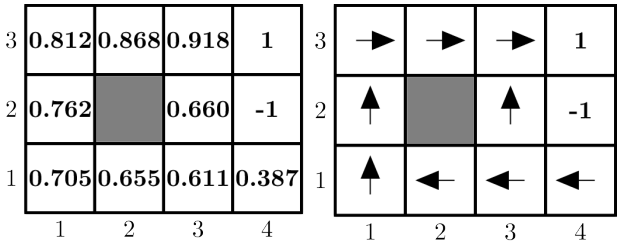


Figure 13: The values after 18 iterations. The convergence is received. On the right image we see the optimal policy π^*

7.2. Rewards

Depending on the constant reward value, the agent will behave differently.

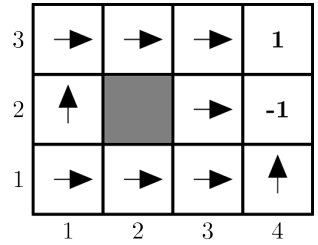


Figure 14: $R(s) < -1.684$ - everything is so bad, so the agent decides to suicide right away

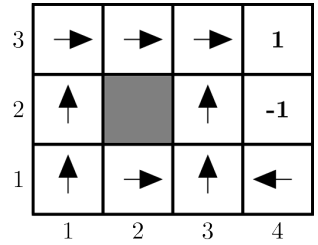


Figure 15: $-0.427 < R(s) < -0.08$ - The agent becomes quite risky

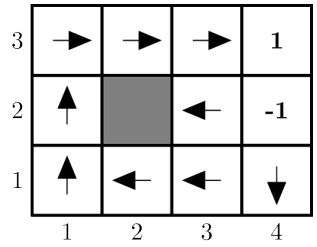


Figure 16: $-0.0221 < R(s) < 0$ - The agent becomes to play too safely

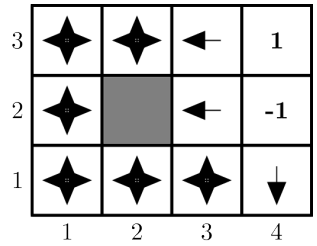


Figure 17: $R(s) > 0$ - The agent does not want to die, and it wants to leave in this world for so long it is possible.

8. Policy iteration algorithm

It consists of 2 steps:

1. Policy evaluation During the policy evaluation, we don't need max of arguments. That way we get a system of linear equations, instead of non-linear. It should be faster to calculate.

$$U_i(s) = R(s) + \alpha \sum_{s'} P(s' \mid s, \pi_i(s)) \cdot U(s')$$

2. Policy improvement(Finding better π^*)

   If this inequality holds:

$$\operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s') > \sum_{s'} P(s' \mid s, \pi(s)) \cdot U(s')$$

then:

$$\pi(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s')$$

8.1. Example

Let's imagine that we have a random policy like in Figure 18.

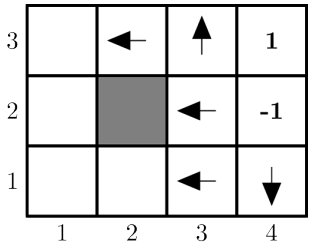


Figure 18: A part of randomly created policy

1. Let's calculate the policy evaluation

$U(3,2) = -0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.04$

$U(3,3) = -0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 1 = 0.06$

$U(3,1) = -0.04 + 0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0 = -0.04$

$U(4,1) = -0.04$

$U(2,3) = -0.04$

We get the following distribution in

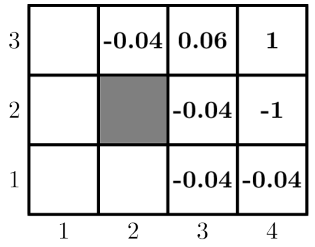


Figure 19: A random policy value distribution

2. Can we improve it?

- *Note*: We use updated values in in this iteration

$U(3,2)$:

(Up): $-0.04 + 0.8 \cdot 0.06 + 0.1 \cdot (-0.04) + 0.1 \cdot (-0.04) = -0.096$

(Right): $-0.04 + 0.8 \cdot (-1) + 0.1 \cdot 0.06 + 0.1 \cdot (-0.04) = -0.838$

(Left): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 0.06 + 0.1 \cdot (-0.04) = -0.07$

(Down): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot (-1) + 0.1 \cdot (-0.04) = -0.176$

In this case the equation does not hold, because $U(3,2) = -0.04$:

$$\operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) \cdot U(s') > \sum_{s'} P(s' \mid s, \pi(s)) \cdot U(s')$$

Therefore the policy does not change:

$\pi^*(3,2) = \leftarrow$

$U(3,3)$:

(Up): $-0.04 + 0.8 \cdot 0.06 + 0.1 \cdot 1 + 0.1 \cdot (-0.04) = 0.104$

(Right): $-0.04 + 0.8 \cdot 1 + 0.1 \cdot 0.06 + 0.1 \cdot (-0.04) = 0.762$

(Left): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 0.06 + 0.1 \cdot (-0.04) = -0.07$

(Down): $-0.04 + 0.8 \cdot (-0.04) + 0.1 \cdot 1 + 0.1 \cdot (-0.04) = 0.024$

In this case the equation does hold, because $U(3,3) = 0.06 < 0.762$.

Therefore the policy does change:

$\pi(3,3) = \uparrow (U(3,3) = 0.06)$

$\pi^*(3,3) = \rightarrow (U(3,3) = 0.762)$

For policy iteration algorithm, it is required to have only 5 iterations for the algorithm to converge to stable values. In comparison to value iteration process, where 18 were needed.

10. Learning from Examples: Machine Learning

By: Alex S.

May, 2024

1. Machine Learning

- The agent cannot be autonomous if it cannot learn
- Machine learning is the induction problem. From examples → general ideas.

1.1. Machine Learning Approaches

- *Symbolic* - formally described knowledge
- *Example based* - does not fully represent the knowledge. Examples are classified, but they are not described
- *Evolution* - using many generations, optimizing the approach

1.2. Learning approaches

- New knowledge addition to the previously available knowledge
  - Knowledge analysis
  - Error resolution
- Useful regularity findings in data
  - Data mining
  - Data Science
  - kNN, decision tree, perceptrons
  - Evolution
- Supervised/Unsupervised learning

1.3. Agent structure

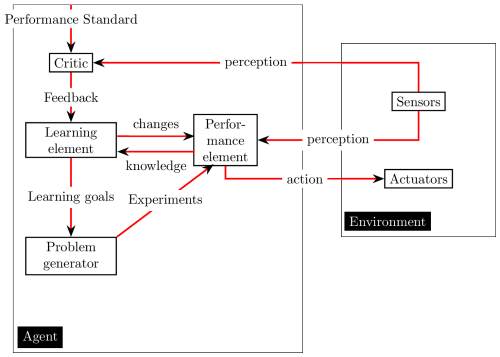


Figure 1: The learning agent general structure

In Figure 1 the problem generator is the generator of noise. The main 2 blocks are *learning* and *performance* elements.

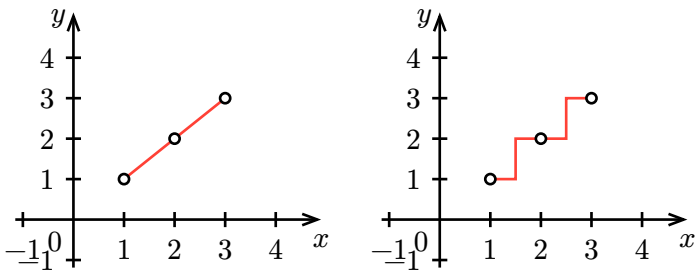
1.4. Learning element design aspects

1. Which performance element components should be modified?
   - Depends on the performance element
2. What knowledge representation type is used?
   - Depends on the project:
     - Frames, networks, logic, ...
3. What feedback is available?
   - Depends on the learning type supervised/ unsupervised
4. What primary knowledge is available?

2. Training

During training the main task of the agent is to find an approximation function by using examples: $(x, f(x))$

However, how do we know which approximation to use to get the best hypothesis?



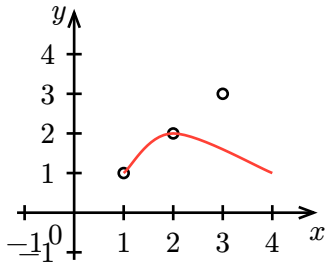


Figure 2: Several graphs that show that the function can be approximated in different ways. Example a) represents a linear relation. Example b) shows discrete relations. The c) example shows inconsistent behavior which cannot be used for hypothesis, because not all examples points are taken into account.

- We say hypothesis **generalizes** well if it correctly predicts the value of y for novel examples.
- To be compliant with hypothesis, every example must be included into it

2.1. How to choose the best hypothesis?

- *Ockham's razor* principle (1334) - there is no need to do bigger effort, if it is possible to use less effort
- Therefore we need to choose the simplest hypothesis
- There is a small possibility that the simplest hypothesis will be inconsistent or incorrect.
- The simplest hypothesis is easier to compute.
- And it better generalizes.

3. Training with decision trees

- Classified as symbolic training
- "White-box" method
- Algorithms to use:
  - ID3
    - More about the algorithm described in Russell & Norvig book in Chapter 18, page 702
  - C4.5, etc.
- Input is the set of attributes(X) and the output is the decision(y): $\{x_1, x_2, ..., x_n\} \in X \to y$
- **Classification** task trains a discrete function
- **Regression** task trains a continuous function

Decision tree is a graph where:
- Inner nodes proceeds with conditional checking
- No-way nodes defines a decision value

Decisions trees allow:
- describe one specific situation/object
- describe any boolean function

> **Note**
>
> there are 2^{2^n} boolean functions where n is the attribute amount

- hard to describe parity or majority functions

3.1. Example

Let's consider an example where we need to find a decision on waiting or not waiting a table in the restaurant. In Figure 3 the data used for the example is presented.

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 3: Data for decision trees. Taken from Russell & Norvig book.

Specific examples can be described with predicate or first-order logic:

1. $$\text{Restaurant} = \text{Full} \land \text{Waiting time} = 0\text{ - }10 \land \neg\ \text{Hungry} \to \text{Wait}$$
2. $$\forall r\ (\text{Regular guests}(r, \text{Full}) \land \text{Waiting time}(r, 0\text{ - }10) \land \neg\ \text{Hungry}(r) \to \text{Wait}(r))$$
3. $$\forall r\ (\text{Wait}(r) = P_1(r) \lor P_2(r) \lor ... \lor P_n(r))$$

   where $P_i(r)$ is one the branches of the tree

3.2. Decision tree construction

- We can create a tree, just by going through the examples one by one
  - However, this tree will be big, 12 levels deep
  - It will be overfitted, i.e. it will be able to predict only test data
  - No way it could generalize good enough
  - It breaks the Ockham's razor principle of simplicity
- Therefore, we use different approach:

Let's take a *Bar* to analyze its examples:

Bar			
No		**Yes**	
Negative	**Positive**	**Negative**	**Positive**
x_2	x_1	x_7	x_3
x_5	x_4	x_9	x_6
x_{11}	x_8	x_{10}	x_{12}

It does not fit, since there is no obvious examples for positive or negative results.

Let's take *Hungry*:

Hungry			
No		**Yes**	
Negative	**Positive**	**Negative**	**Positive**
x_5	x_3	x_2	x_1
x_7		x_{10}	x_4
x_{11}			x_6
x_9			x_8

It is closer, but still no explicit positives/negatives.

Let's take *Regular guests*:

Regular guests					
No one		**Some**		**Full**	
Neg	**Pos**	**Neg**	**Pos**	**Neg**	**Pos**
x_7			x_1	x_2	x_4
x_{11}			x_3	x_5	x_{12}
			x_6	x_9	
			x_8	x_{10}	

Here we can certainly see a nice tendency that *No one* and *Some* has explicitly only negative and positive examples. This is the best attribute, therefore it will go to the root of the tree.

The next step is to take all the data left with *Full* attribute, and do the same actions with the data left-overs.

The final tree in Figure 4

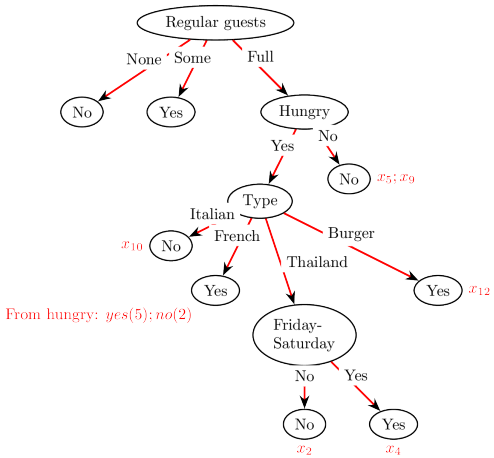


Figure 4: The final decision tree

> **Note**
>
> The value of *French* edge was not determined. It was determined by the majority of values from the parent node *Hungry*.

When doing a construction of the tree, it is needed to keep in mind following properties.

Properties:

1. If there are some positive or some negative examples, we choose the best attribute
2. If there are all positive or all negative examples, the node is fully decided and solved
3. If there are no values at the node, we return a default value. E.g. getting the majority from the parent values. As it was in the example of *French*, where majority of the parent was *Yes*.
4. If there are no attributes, but there is data, then there is no way, but to search for a new attribute

3.3. Choosing attribute tests

- To choose the attribute Shannon and Weaver method (1949) for information amount calculation is used.
- The less is known about the result, the more information is received.
- It is described with the **information** or **entropy** formula:

$$I(p(v_1), ..., p(v_n)) = \sum_{i=1}^{n} -p(v_i) \cdot \log_2 p(v_i)$$

where I is the information amount in bits and $p(v_i)$ is the probability of the random variable $v_i \in V$ to occur.

For example, the fair coin where it has a 50% probability of both results is calculated:

$$I\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 \text{ bit}$$

- We have 1 bit of information where we certainly know “heads” or “tails”
- This is called as a “perfect” attribute that equally divides the information
- Every attribute divide a dataset into subsets. E.g. attribute *Hungry* from Figure 4 divides the set into *Yes* and *No*

When working with subsets, we have the following formula:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

where p and n represent the total number of positive and negative examples.

We can also just use the information or entropy formula: $I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$ where i is the subset.

3.4. Information gain

We can calculate the information gain for each attribute in the following way:

1. Calculate the remainder entropy for the current subset:

$$R(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \cdot I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

where R is a remainder, A is the attribute, v is the number of subsets in the attribute.

2. Calculate the gain for the attribute with:

$$G(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - R(A)$$

where G is the information gain for the attribute

3.4.1. Example

Let's calculate the information gain for *Regular guests*.

Some, $\{p:4; n:0\}$ — *Full*, $\{p:2; n:4\}$

$$G(\text{Regular guests}) = 1 - \left(\frac{4}{12}\cdot I(1,0) + \frac{2}{12}\cdot I(0,1) + \frac{6}{12}\cdot I\left(\frac{2}{6},\frac{4}{6}\right)\right)$$

≈ 0.541

The full set, $\{p:6; n:6\} \to I\left(\frac{1}{2},\frac{1}{2}\right) = 1$

None, $\{p:0; n:2\}$

Let's calculate the information gain for attribute *Type*:

French, $\{p:1; n:1\}$ — *Burger*, $\{p:2; n:2\}$ — *Thailand*, $\{p:2; n:2\}$ — *Italian*, $\{p:1; n:1\}$

$$G(\text{Type}) = 1 - \left(\frac{2}{12}\cdot I\left(\frac{1}{2},\frac{1}{2}\right) + \frac{4}{12}\cdot I\left(\frac{2}{4},\frac{2}{4}\right) + \frac{4}{12}\cdot I\left(\frac{2}{4},\frac{2}{4}\right) + \frac{2}{12}\cdot I\left(\frac{1}{2},\frac{1}{2}\right)\right) = 1 - \left(\frac{1}{6}+\frac{1}{3}+\frac{1}{3}+\frac{1}{6}\right) = 0$$

It looks obvious that $G(\text{Regular guests}) > G(\text{Type})$ therefore we use the attribute with the biggest information gain as the root node.

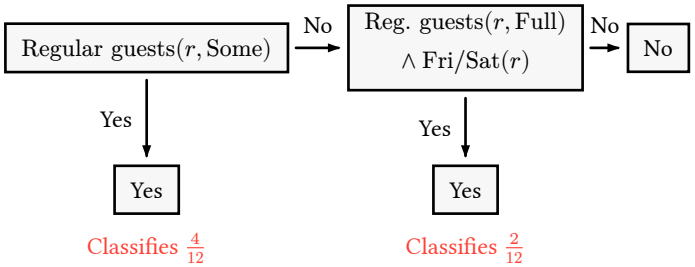
4. Evaluation of learning algorithm

1. Need to collect a big dataset with examples
2. Need to separate the dataset into 2 sets: **training** and **testing**
3. Need to use a **training** dataset to generate a hypothesis
4. Measure the amount of examples that are correctly classified
5. Amend the **training** dataset with more examples possible

5. Decision list learning

- Logical statement in the limited form that consists of the series of tests described with words and conjunctions. If the test is successful then the following tests are decided.

Example:

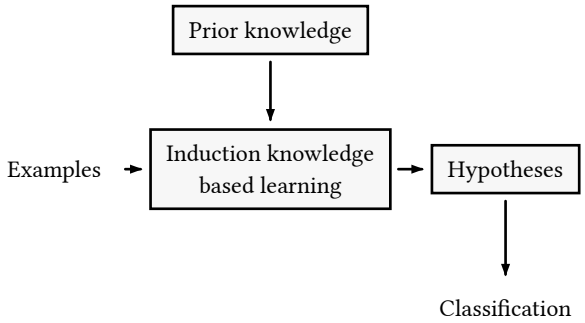


The diagram above can be described as:

$$\text{Wait} \equiv (\text{Restaurant} = \text{Some}) \vee (\text{Restaurant} = \text{Full} \wedge \text{Fri/Sat})$$

6. Usage of knowledge in learning

- Most of the methods researched previously do not use any prior domain knowledge
- The cumulative learning process can be displayed the following way:



- Prior knowledge is described with the first-order logic

6.1. Example

There is a logical description:

$$\text{Alternative}(X_1) \wedge \neg\,\text{Bar}(X_1) \wedge \wedge \neg\,\text{Fri/Sat}(X_1) \wedge \text{Hungry}(X_1)$$

The classification is:

$$\text{Wait}(X_1) \equiv Q(X_1)$$

where $Q(X_1)$ is the unary goal predicate

- The goal is to find the logical statement that will satisfy the $Q(X)$ for all examples.
- Every hypothesis describes a goal predicate candidate C_i
- Therefore every hypothesis H_i is a sentence described as:

$$\forall x\ Q(x) \equiv C_i(x)$$

- For the restaurant problem, the inducted decision tree suggests the following candidate(hypothesis):

$$\begin{aligned}\forall r \ \ \text{Wait}(r) \equiv \text{Regular guests}(r, \text{Some}) \vee \\ \vee (\text{Regular guests}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \\ \wedge \text{Type}(r, \text{French})) \vee \\ \vee (\text{Regular guests}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \\ \wedge \text{Type}(r, \text{Burger})) \vee \\ \vee (\text{Regular guests}(r, \text{Full}) \wedge \text{Hungry}(r) \wedge \\ \wedge \text{Type}(r, \text{Thailand}) \wedge \text{Fri/Sat}(r))\end{aligned}$$

- If hypothesis is **consistent** with the whole dataset, then it is **consistent** with every example.
- The table of hypothesis consistence with examples:

		Actual	
		True	*False*
Predicted	*True*	True positive(TP)	False positive(FP)
	False	False negative(FN)	True negative(TN)

7. Searching the current-best-hypothesis

- For pseudocode use Russell & Norvig book on page 770
- The idea of the method is to keep one hypothesis and adjust it to keep the consistence
- The algorithm is greedy
- The process can be visualized as in Figure 5

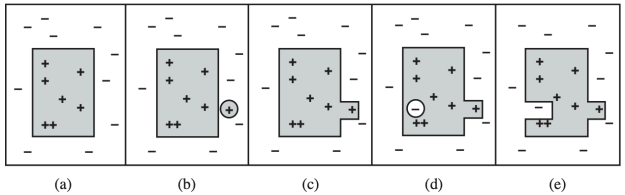


Figure 5: a) A consistent hypothesis. (b) A false negative. (c) The hypothesis is generalized. (d) A false positive. (e) The hypothesis is specialized.

- Generalization and specialization are defined as the operations which change the hypothesis extension
- If hypothesis H_1 with definition C_1 is the hypothesis' H_2 with definition C_2 generalization, then

$$\forall x \ C_2(x) \Rightarrow C_1(x)$$

I.e. C_1 is the generalization over C_2

E.g.

$$\forall r \ C_2(r) \equiv \text{Alternative}(r) \wedge \text{Regular guests}(r, \text{Some})$$
$$\forall r \ C_1(r) \equiv \text{Regular guests}(r, \text{Some})$$
$$C_2 \Rightarrow C_1$$

7.1. Example

- $H_1 : \forall r \ \ \text{Wait}(r) \equiv \text{Alternative}(r)$
  - 2nd example is false positive from Figure 3
  - Therefore we use specialization
- $H_2 : \forall r \ \ \text{Wait}(r) \equiv \text{Alternative}(r) \wedge \text{Regular guests}(r, \text{Some})$
  - 3rd example from Figure 3 is false negative
  - Therefore we generalize the hypothesis
- $H_3 : \forall r \ \ \text{Wait}(r) \equiv \text{Regular guests}(r, \text{Some})$
  - 4th example from Figure 3 is false negative
  - Therefore we generalize the hypothesis
  - The generalization removes the attribute, therefore we replace it with something else or use disjunction($\vee$)
- $H_4 : \forall r \ \ \text{Wait}(r) \equiv \text{Regular guests}(r, \text{Some}) \vee (\text{Regular guests}(r, \text{Full}) \wedge \text{Fri/Sat}(r))$
  - H_4 is consistent with all examples
  - It could also be different, e.g.
    - $H_4' : \forall r \ \ \text{Wait}(r) \equiv \neg \ \text{Waiting time}(r, 30 - 60)$
    - etc.

11. Learning from Examples: NNs and Reinforcement Learning

By: Alex S.

May, 2024

1. Artificial neural networks

Perceptron - is a single neuron neural network model (Figure 1).

It also represents a **feed-forward** network where input is fed towards the output.

The **recurrent** network can also communicate with the previous layers.

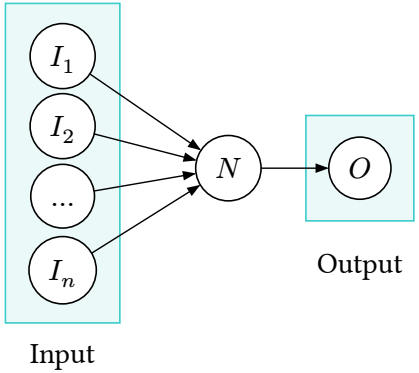


Figure 1: A perceptron

A neuron can be represented as a function in Figure 2. The model takes inputs $\{x_1, x_2, ..., x_n\} \in X$ and they are calculated in the summation function:

$$z_i = \sum_{j=1}^{n} x_j \cdot w_{ji}$$

where w_{ji} is the weight for the specific input and z_i is the summation result.

After that the result goes to the non-linear activation function φ that calculates the output($\hat{y}$) that is called a **prediction**:

$$\hat{y} = \varphi(z_i)$$

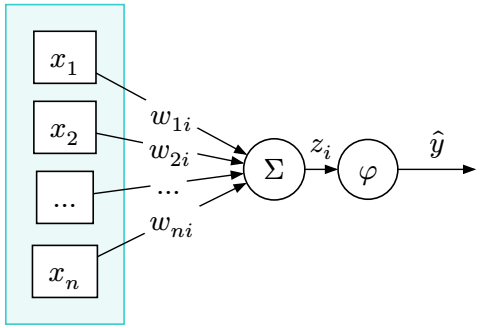


Figure 2: An i-th neuron mathematical representation

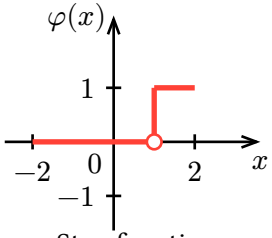
1.1. Activation functions

Activation functions are used to transform the summed weighted input from a node into an output value that is passed on to the next layer.

Step function:

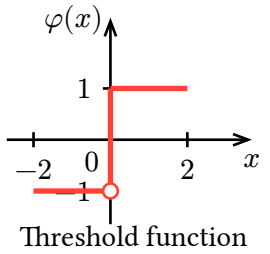
$$\varphi(x) = \begin{cases} 0 \text{ if } x < T \\ 1 \text{ if } x \geq T \end{cases}$$

In the image $T = 1$



Threshold function:

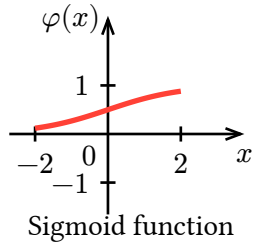
$$\varphi(x) = \begin{cases} -1 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$



Sigmoid function:

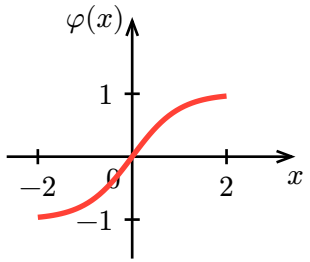
$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

The derivative is $\varphi'(x) = \varphi \cdot (1 - \varphi)$



Tanh function:

$$\varphi(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



1.2. Example

Let's say we want to classify black and white dots in Figure 3. The red dashed line represents a linear function that will do the classification job.

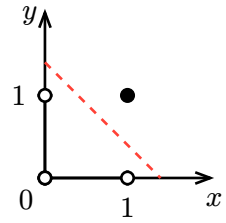


Figure 3: An **AND** classification problem

In order to solve this problem, we can use a simple perceptron with the step function(Figure 4).

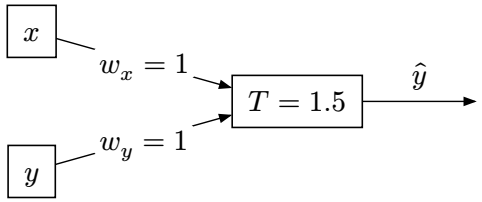


Figure 4: Logical **AND** representation with the perceptron

Step function's threshold is $T = 1.5$, in this case we can create a table to check the results:

x	y	output	$\hat{y}$	y
0	0	0	0	0
0	1	1	0	0
1	0	1	0	0
1	1	2	1	1

Figure 5: Check table for **AND** neuron

The similar perceptron model can be created for the logical "OR" function(Figure 6), the only change is in the step function's threshold.

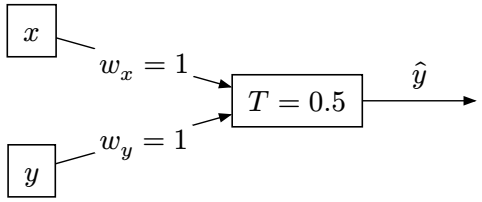


Figure 6: Logical **AND** representation with the perceptron

However, if we want to separate classes with two lines as in Figure 7, i.e. make an **XOR** operation, then it requires to add an additional neuron to the network.

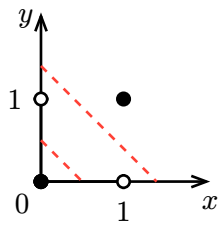


Figure 7: An **XOR** classification problem

2. Multilayer feed-forward neural networks

The generalized structure can be obtained the following way:

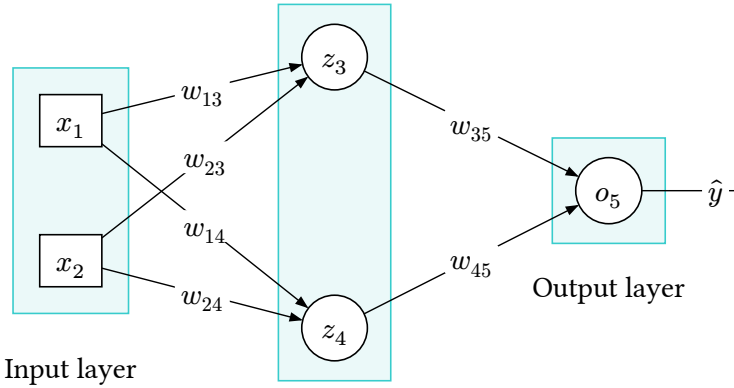


Figure 8: General multilayer neural network scheme

In Figure 8 in order to get the $\hat{y}$ value, the following calculations must be done:

$$o_5 = \varphi(w_{35} \cdot z_3 + w_{45} \cdot z_4)$$
$$z_4 = \varphi(w_{14} \cdot x_1 + w_{24} \cdot x_2)$$
$$z_3 = \varphi(w_{13} \cdot x_1 + w_{23} \cdot x_2)$$

where φ is the activation function

The **XOR** problem mentioned previously can be solved with the following network in Figure 9

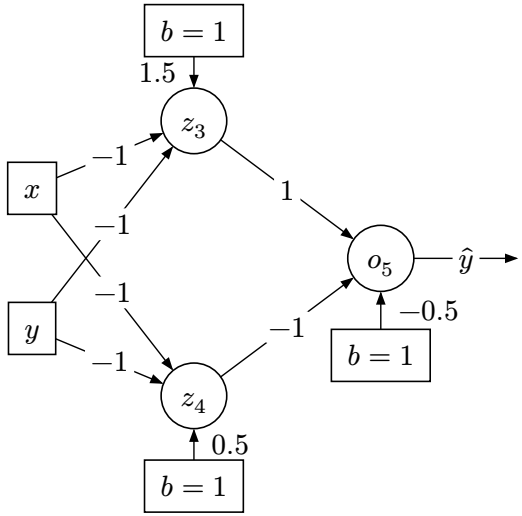


Figure 9: A multilayer network solution for **XOR** problem.

For the network all neurons have a *threshold activation function*:

$$\varphi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

x	y	z_3	$\varphi(z_3)$	z_4	$\varphi(z_4)$	o_5	$\hat{y}$	y
0	0	1.5	1	0.5	1	−0.5	0	0
0	1	0.5	1	−0.5	0	0.5	1	1
1	0	0.5	1	−0.5	0	0.5	1	1
1	1	−0.5	0	−1.5	0	−0.5	0	0

Figure 10: Check table for **XOR** operation neural network. *Note*: $\varphi(o_5) = \hat{y}$

> **Note**
>
> We added the bias to the linear summation.
> E.g. $z_3 = -1 \cdot x + -1 \cdot y + 1.5 \cdot b$

3. Learning in neural networks

- For the perceptron to learn it needs to get *enough* examples to learn a linear function.
- *Enough* examples can be calculated by using Novikov's theorem:

$$N = \frac{D^2}{d^2}$$

where D is the class set diameter, i.e. the maximum distance between examples, and d is the distance between sets, i.e. the distance between the nearest neighbors(examples) of sets

- Learning happens by sequentially providing examples to the perceptron, then calculating the error and updating the weights.

Learning process:

1. Get the output for an example, i.e. predicted value $\hat{y}$, from the network
2. Calculate the error: $E = y - \hat{y}$, where y is the actual value
3. Update the weights with $w_j{}' = w_j + \alpha \cdot x_j \cdot E$ where α is the learning rate, x_j is the input value, E an error
4. Continue the training until all examples are not seen
5. Continue until weights do not change or iteration count is exceeded

3.1. Learning in the perceptron model

Consider the example of getting predictions on the student's marks:

- x_1 - does the student do the assignments(no - 0, yes - 1)
- x_2 - does the student go to lections(no - 0, yes - 1)
- x_3 - does the student prepare for the exam(no - 0, yes - 1)
- $x_4 = 1$ - is a bias

Overall: there are $2^3 = 8$ data points available

Output: +1 or −1 for good and results

Activation function: Threshold function

x_1	x_2	x_3	x_4	y
0	0	0	1	1
1	0	0	1	1
0	1	1	1	−1
1	1	0	1	1

Figure 11: A training dataset for perceptron

Let's do some training. *1st iteration*:

#	Input				Weights				z	$\hat{y}$	y	New weights			
	x_1	x_2	x_3	x_4	w_1	w_2	w_3	w_4				$w_1{}'$	$w_2{}'$	$w_3{}'$	$w_4{}'$
1	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0
2	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0
3	0	1	1	1	0	0	0	0	0	1	−1	0	−1	−1	−1
4	1	1	0	1	0	−1	−1	−1	−2	−1	1	1	0	−1	0

Figure 12: 1st iteration of table with perceptron training. *Note*: $z = \sum_j^n w_j \cdot x_j$

Until the 3rd example, no change in weights was necessary.

3rd example weights updates:

Note: $\alpha = 0.5$

$$E = y - \hat{y} = -1 - 1 = -2$$
$$w_1{}' = w_1 + \alpha \cdot x_1 \cdot E = 0 + 0.5 \cdot 0 \cdot (-2) = 0$$
$$w_2{}' = w_2 + \alpha \cdot x_2 \cdot E = 0 + 0.5 \cdot 1 \cdot (-2) = -1$$
$$w_3{}' = w_3 + \alpha \cdot x_3 \cdot E = 0 + 0.5 \cdot 1 \cdot (-2) = -1$$
$$w_4{}' = w_4 + \alpha \cdot x_4 \cdot E = 0 + 0.5 \cdot 1 \cdot (-2) = -1$$

4th example weights updates:

$$E = y - \hat{y} = 1 - (-1) = 2$$
$$w_1{}' = w_1 + \alpha \cdot x_1 \cdot E = 0 + 0.5 \cdot 1 \cdot 2 = 1$$
$$w_2{}' = w_2 + \alpha \cdot x_2 \cdot E = -1 + 0.5 \cdot 1 \cdot 2 = 0$$
$$w_3{}' = w_3 + \alpha \cdot x_3 \cdot E = -1 + 0.5 \cdot 0 \cdot 2 = -1$$
$$w_4{}' = w_4 + \alpha \cdot x_4 \cdot E = -1 + 0.5 \cdot 1 \cdot 2 = 0$$

2nd iteration:

#	Input				Weights				z	$\hat{y}$	y	New weights			
	x_1	x_2	x_3	x_4	w_1	w_2	w_3	w_4				$w_1{}'$	$w_2{}'$	$w_3{}'$	$w_4{}'$
1	0	0	0	1	1	0	−1	0	0	1	1	1	0	−1	0
2	1	0	0	1	1	0	−1	0	1	1	1	1	0	−1	0
3	0	1	1	1	1	0	−1	0	−1	−1	−1	1	0	−1	0
4	1	1	0	1	1	0	−1	0	1	1	1	1	0	−1	0

Figure 13: 2nd iteration of table with perceptron training. *Note*: $z = \sum_j^n w_j \cdot x_j$

Weights have converged and no change was done.

The final perceptron model:

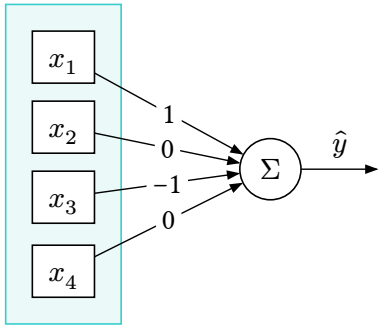


Figure 14: Final trained perceptron model

Let's see how it performs. The test data is in Figure 15.

x_1	x_2	x_3	x_4	y
1	0	1	1	1
0	0	1	1	−1
1	1	1	1	−1
0	1	0	1	−1

Figure 15: A testing dataset for perceptron

#	Input				Weights				z	$\hat{y}$	y	Ok?
	x_1	x_2	x_3	x_4	w_1	w_2	w_3	w_4				
1	1	0	1	1	1	0	−1	0	0	1	1	+
2	0	0	1	1	1	0	−1	0	−1	−1	−1	+
3	1	1	1	1	1	0	−1	0	0	1	−1	-
4	0	1	0	1	1	0	−1	0	0	1	−1	-

Figure 16: Results for perceptron testing. *Note*: $z = \sum_j^n w_j \cdot x_j$

The accuracy of the perceptron is only 50%...

3.2. Learning in multilayer model

For the learning of multilayer neural network model, it is needed to use a **backpropagation** algorithm. More on the algorithm see Russell & Norvig's book page 734.

The generalized multilayer network can be represented as in Figure 8.

The weight update process for the one hidden layer NN is the following:

1. Calculate the error:

$$E = y - \hat{y}$$

where y is the actual value, $\hat{y}$ - predicted

2. Update the weights for the hidden layer:
   a. Get the partial error:

$$\Delta_i = E \cdot \varphi'(z_j)$$

   where $\varphi'(z_j)$ is the derivative for the activation function, e.g. sigmoid will be calculated as $\varphi' = \varphi(1-\varphi)$, z_j is the weighted sum with of the neuron
   b. Update the weight:

$$w_{ji}{}' = w_{ji} + \alpha \cdot a_j \cdot \Delta_i$$

   where w_{ji} is the weight of the j-th neuron of the previous layer to the i-th neuron of the next layer. α is the learning rate.

3. Update the weights for the input layer:
   a. Get the partial error:

$$\Delta_j = \varphi'(z_j) \cdot \sum_i w_{ji} \cdot \Delta_i$$

   b. Update the weight:

$$w_{kj}{}' = w_{kj} + \alpha \cdot x_k \cdot \Delta_j$$

   where x_k is the input value

4. Repeat the process for the number of iterations

3.3. Multilayer learning example

Consider the problem of **XOR** operation, where one weight was modified to create an error:

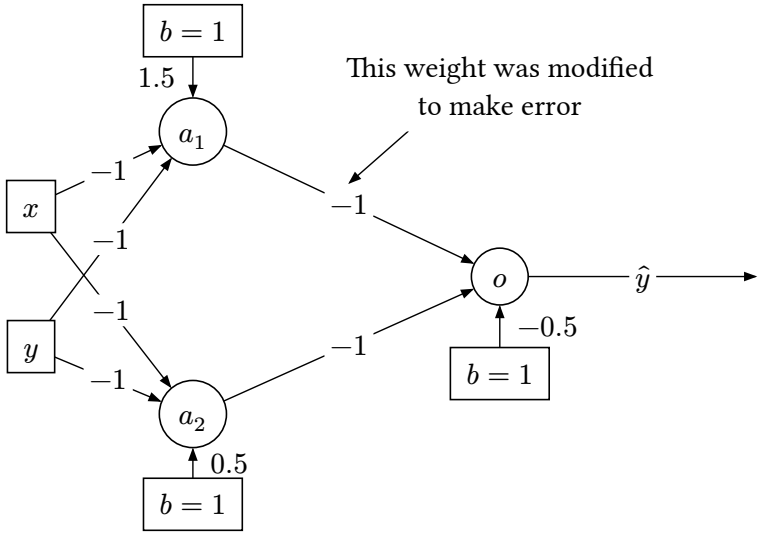


Figure 17: A multilayer network solution for **XOR** problem with one modified weight.

Note: Using a sigmoid function

1. Calculate the predicted value:

$$z_1 = 1 \cdot 1.5 + 1 \cdot (-1) + 1 \cdot (-1) = -0.5$$

$$a_1 = \frac{1}{1 + e^{0.5}} = 0.38$$

$$z_2 = 1 \cdot 0.5 + 1 \cdot (-1) + 1 \cdot (-1) = -1.5$$

$$a_2 = \frac{1}{1 + e^{1.5}} = 0.18$$

$$z_3 = (-0.5) \cdot 1 + (-1) \cdot 0.38 + (-1) \cdot 0.18 = -1.06$$

$$o = \frac{1}{1 + e^{1.06}} = 0.26$$

2. There is an error, since $y = 0$, but we predicted $\hat{y} = 0.26$, so we need to update the weights. Calculate the error:

$$E = y - \hat{y} = 0 - 0.26 = -0.26$$

3. Get the partial error for the output layer:

$$\Delta_o = E \cdot g'(o) = -0.26 \cdot 0.26(1 - 0.26) = -0.05$$

4. Update weights for the hidden layer:
   a. $w_{a_1,o}{}' = w_{a_1,o} + \alpha \cdot a_1 \cdot \Delta_o = -1 + 1 \cdot 0.38 \cdot (-0.05) = -1.02$
   b. $w_{a_2,o}{}' = w_{a_2,o} + \alpha \cdot a_2 \cdot \Delta_o = -1 + 1 \cdot 0.18 \cdot (-0.05) = -1.01$
   c. $w_{b,o}{}' = w_{b,o} + \alpha \cdot b \cdot \Delta_o = -0.5 + 1 \cdot 1 \cdot (-0.05) = -0.55$

5. Get partial errors for the hidden layer:
   a. $\Delta_{a_1} = g'(a_1) \cdot w_{a_1,o} \cdot \Delta_o = a_1(1 - a_1) \cdot w_{a_1,o} \cdot \Delta_o = 0.38(1 - 0.38) \cdot -1 \cdot (-0.05) = 0.012$
      - We have only one edge to the output, therefore no summation!
   b. $\Delta_{a_2} = g'(a_2) \cdot w_{a_2,o} \cdot \Delta_o = a_2(1 - a_2) \cdot w_{a_2,o} \cdot \Delta_o = 0.18(1 - 0.18) \cdot -1 \cdot (-0.05) = 0.007$

6. Update the input to hidden layer weights:
   a. $w_{x,a_1}{}' = w_{x,a_1} + \alpha \cdot x \cdot \Delta_{a_1} = -1 + 1 \cdot 1 \cdot 0.012 = -0.988$
   b. $w_{x,a_2}{}' = w_{x,a_2} + \alpha \cdot x \cdot \Delta_{a_2} = -1 + 1 \cdot 1 \cdot 0.007 = -0.993$
   c. $w_{y,a_1}{}' = w_{y,a_1} + \alpha \cdot y \cdot \Delta_{a_1} = -1 + 1 \cdot 1 \cdot 0.012 = -0.988$
   d. $w_{y,a_2}{}' = w_{y,a_2} + \alpha \cdot y \cdot \Delta_{a_2} = -1 + 1 \cdot 1 \cdot 0.007 = -0.993$
   e. $w_{b,a_1}{}' = w_{b,a_1} + \alpha \cdot b \cdot \Delta_{a_1} = 1.5 + 1 \cdot 1 \cdot 0.012 = 1.512$
   f. $w_{b,a_2}{}' = w_{b,a_2} + \alpha \cdot b \cdot \Delta_{a_2} = 0.5 + 1 \cdot 1 \cdot 0.007 = 0.507$

The network after weight updates from the 1st example:

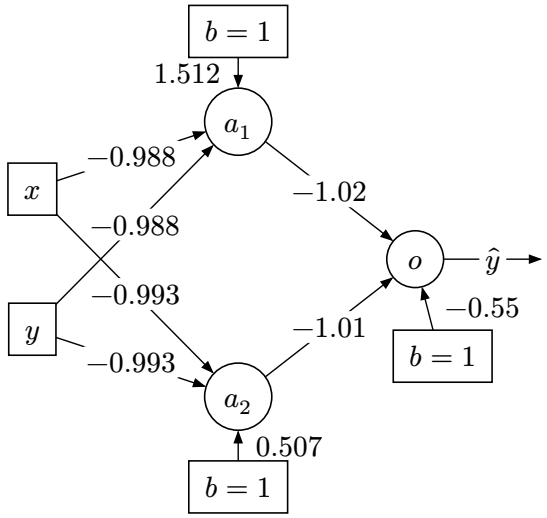


Figure 18: A multilayer network solution for **XOR** problem after weight update with 1 example

- The speed of learning is very slow, since weight corrections are minimal.
- The next step is to take the second example and repeat the weight update process once again until all 4 examples are met, after that the first iteration will be done.

4. Reinforcement Learning

Learning tasks:

1. Environment can be accessible or inaccessible(the agent needs to store the inner state)
2. The agent can start with the knowledge about the environment and actions, or this model should learned the same way as the utilities
3. Rewards can be given in the end or any other states
4. Rewards can be a part of the utility component(e.g. money, points in the game) that is tried to be maximized, or it could be a direction to the real utility, e.g. a “good move”.
5. The agent can be a passive learner, which follows the world and learns the utilities. Or it can be an active learner, which uses learned information and can create new problems to explore the environment

Reinforcement learning directions:

1. The agent learns the **utility-based function** which allows to choose the maximum result action. That is in utility-based agent.
2. The agent learns **action-utility-based function** which gives an expected utility, if the action chosen in the current state. That is **Q-learning**.
3. The reflex agent learns **policy** that maps states to actions.

5. Passive learning in accessible environment

5.1. Conditions

- We use a **naive** approach to update the utilities
- The game is the same 4x3 stochastic environment as used before(Figure 19).
- $R(s) = -0.04$

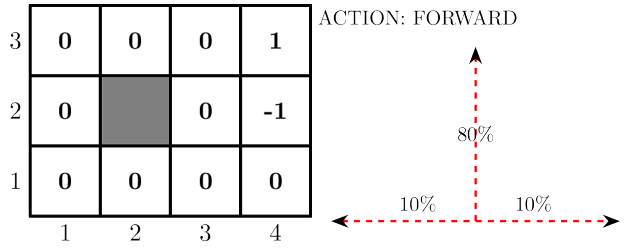


Figure 19: The environment starting state and the transition model

5.2. Starting policy

Let's imagine that the agent uses the following policy π from the start to learn utility function $U^{\pi}(s)$:

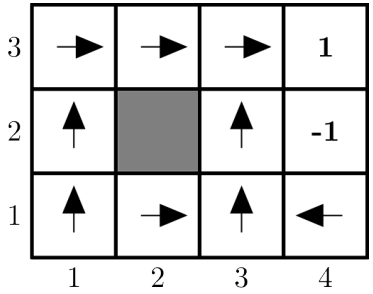


Figure 20: Starting policy

5.3. Training examples

To learn the new policy, the agent needs some examples. We will use the following 5 sequences of actions in the table as the learning examples. These sequences were generated by using the generated policy π before.

#	Sequence
1	$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3)$
2	$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,3)$
3	$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2)$
4	$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (4,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (4,3)$
5	$(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (4,1) \rightarrow (4,2)$

5.4. Naive approach

- We will go backwards through the sequence, each time adding the $R(s)$ to the final score.
- If the action was met several times, we take an average of it
- We go through each training example one by one

1. $(1,1)_{U=0.72} \rightarrow (1,2)_{U=0.76} \rightarrow (1,3)_{U=0.80} \rightarrow (1,2)_{U=0.84} \rightarrow (1,3)_{U=0.88} \rightarrow (2,3)_{U=0.92} \rightarrow (3,3)_{U=0.96} \rightarrow (4,3)_{U=1}$
   - Two states are repeated, therefore we use an average value:
     - $U(1,2) = \frac{0.76+0.84}{2} = 0.80$
     - $U(1,3) = \frac{0.88+0.80}{2} = 0.84$

3	**0.84**	**0.92**	**0.96**	**1**
2	**0.80**		**0**	**-1**
1	**0.72**	**0**	**0**	**0**
	1	2	3	4

Figure 21: First example results

2. $(1,1)_{U=0.72} \rightarrow (1,2)_{U=0.76} \rightarrow (1,3)_{U=0.80} \rightarrow (2,3)_{U=0.84} \rightarrow (3,3)_{U=0.88} \rightarrow (3,2)_{U=0.92} \rightarrow (3,3)_{U=0.96} \rightarrow (4,3)_{U=1}$
   - Utilities from previous calculation are also used to get averaged:
     - $U(1,1) = 0.72$
     - $U(1,2) = \frac{0.76+0.80}{2} = 0.78$
     - $U(1,3) = \frac{0.84+0.80}{2} = 0.82$
     - $U(2,3) = \frac{0.84+0.92}{2} = 0.88$
     - $U(3,3) = \frac{(\frac{0.88+0.96}{2})+0.96}{2} = 0.94$

3	**0.82**	**0.88**	**0.94**	**1**
2	**0.78**		**0.92**	**-1**
1	**0.72**	**0**	**0**	**0**
	1	2	3	4

Figure 22: Second example results

3. $(1,1)_{U=-1.16} \rightarrow (2_1)_{U=-1.12} \rightarrow (3,1)_{U=-1.08} \rightarrow (3,2)_{U=-1.04} \rightarrow (4,2)_{U=-1}$
   - We will omit average calculations from previous examples

3	**0.82**	**0.88**	**0.94**	**1**
2	**0.78**		**-0.06**	**-1**
1	**-0.22**	**-1.12**	**-1.08**	**0**
	1	2	3	4

Figure 23: Third example results

4. $(1,1)_{U=0.72} \rightarrow (2,1)_{U=0.76} \rightarrow (3,1)_{U=0.80} \rightarrow (4,1)_{U=0.84} \rightarrow (3,1)_{U=0.88} \rightarrow (3,2)_{U=0.92} \rightarrow (3,3)_{U=0.96} \rightarrow (4,3)_{U=1}$

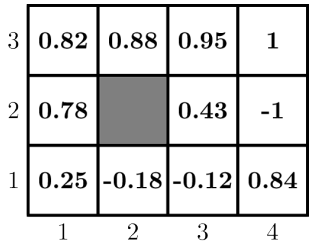


Figure 24: Fourth example results

5. $(1,1)_{U=-1.24} \rightarrow (2,1)_{U=-1.20} \rightarrow (3,1)_{U=-1.16} \rightarrow (2,1)_{U=-1.12} \rightarrow (3,1)_{U=-1.08} \rightarrow (4,1)_{U=-1.04} \rightarrow (4,2)_{U=-1}$

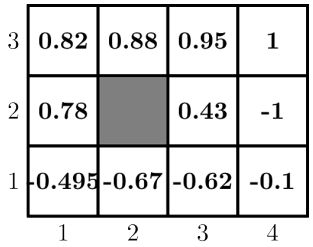


Figure 25: Fifth example results

After 5 examples, we get the end policy that is very close to the optimal policy that was found in the complex decision works.

Take into account that (4, 1) is not completely solved and it is indifferent of choosing either of directions. More examples can make it more decisive.

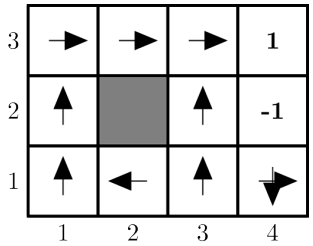


Figure 26: The policy after 5 examples

> **Note**
>
> The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed as in equation:
>
> $$U^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \alpha^t R(S_t)\right]$$

5.5. Direct utility estimation

Direct utility estimation succeeds in reducing the reinforcement learning problem to an inductive learning problem, about which much is known.

The utility values obey the Bellman equations for a fixed policy:

$$U^{\pi}(s) = R(s) + \alpha \cdot \sum_{s'} P(s' \mid s, \pi(s)) \cdot U^{\pi}(s')$$

where $P(s' \mid s, \pi(s))$ can also be represented as the transition matrix $M_{s,s'}$

The transition model can also be represented as a graph:

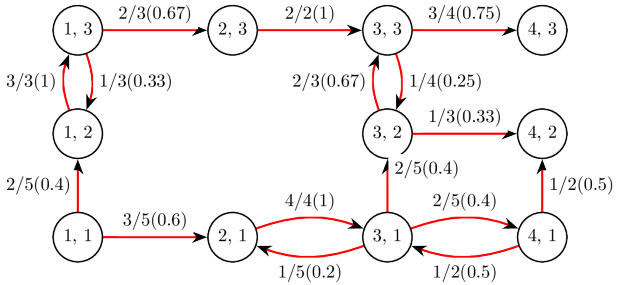


Figure 27: A transition model for the 4x3 world problem

It allows to get the following system of equations:

$$\begin{cases} U(1,1) = -0.04 + 0.4 \cdot U(1,2) + 0.6 \cdot U(2,1) \\ U(1,2) = -0.04 + U(1,3) \\ U(1,3) = -0.04 + 0.33 \cdot U(1,2) + 0.67 \cdot U(2,3) \\ U(2,1) = -0.04 + U(3,1) \\ U(2,3) = -0.04 + U(3,3) \\ U(3,1) = -0.04 + 0.2 \cdot U(2,1) + 0.4 \cdot U(3,2) \\ +0.4 \cdot U(4,1) \\ U(3,2) = -0.04 + 0.67 \cdot U(3,3) + 0.33 \cdot U(4,2) \\ U(3,3) = -0.04 + 0.25 \cdot U(3,2) + 0.75 \cdot U(4,3) \\ U(4,1) = -0.04 + 0.5 \cdot U(3,1) + 0.5 \cdot U(4,2) \\ U(4,2) = -1 \\ U(4,3) = 1 \end{cases}$$

The system can be solved, for example, with the *Gauss elimination method.*

5.6. Temporal difference learning

- The main idea is to limit local limits of each transition
- It is described with:

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + U^{\pi}(s') - U^{\pi}(s))$$

where α is the learning rate

Let's try this approach with 5 examples and $\alpha = 0.5$

1. $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3)$

$$U(3,3) = 0 + 0.5 \cdot (-0.04 + 1 - 0) = 0.48$$

$$U(2,3) = 0 + 0.5 \cdot (-0.04 + 0.48 - 0) = 0.22$$

$$U(1,3) = 0 + 0.5 \cdot (-0.04 + 0.22 - 0) = 0.09$$

Previous value of $U(1,3)$

$$U(1,2) = 0 + 0.5 \cdot (-0.04 + 0.09 - 0) = 0.025$$

$$U(1,3) = 0.09 + 0.5 \cdot (-0.04 + 0.025 - 0.09) = 0.0375$$

$$U(1,2) = 0.025 + 0.5 \cdot (-0.04 + 0.0375 - 0.025) = 0.0112$$

$$U(1,1) = 0 + 0.5 \cdot (-0.04 + 0.0112 - 0) = -0.0144$$

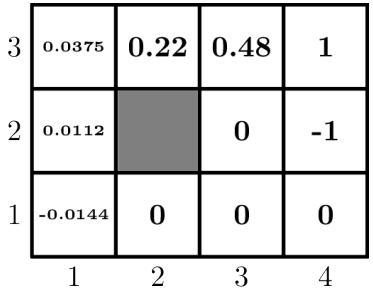


Figure 28: Results after 1st example

2. $(1,1) \to (1,2) \to (1,3) \to (2,3) \to (3,3) \to (3,2) \to (3,3) \to (4,3)$

$$U(3,3) = 0.48 + 0.5 \cdot (-0.04 + 1 - 0.48) = 0.72$$

$$U(3,2) = 0 + 0.5 \cdot (-0.04 + 0.72 - 0) = 0.34$$

$$U(3,3) = 0.72 + 0.5 \cdot (-0.04 + 0.34 - 0.72) = 0.51$$

$$U(2,3) = 0.22 + 0.5 \cdot (-0.04 + 0.51 - 0.22) = 0.345$$

$$U(1,3) = 0.0375 + \\ +0.5 \cdot (-0.04 + 0.345 - 0.0375) = 0.1713$$

$$U(1,2) = 0.0112 + \\ +0.5 \cdot (-0.04 + 0.1713 - 0.0112) = 0.0713$$

$$U(1,1) = -0.0144 + \\ +0.5 \cdot (-0.04 + 0.0713 - (-0.0144)) = 0.0084$$

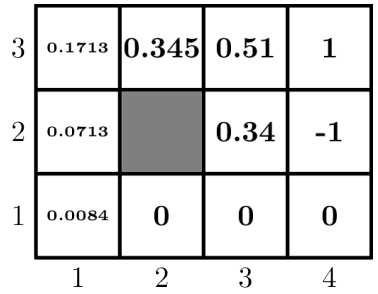


Figure 29: Results after 2nd example

Doing these operations for all five examples we receive the final utilities and the policy π:

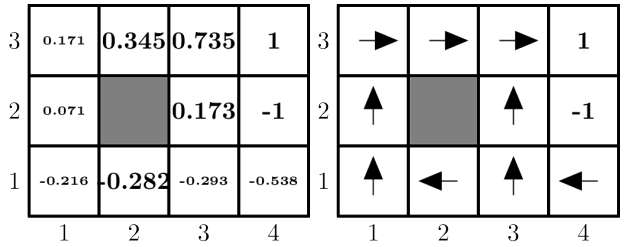


Figure 30: Utilities and policy after 5 examples

The training gave a good result in comparison to the optimal policy.

> **Info**
>
> This is the end of the course