Künstliche Intelligenz 2 - Zusammenfassung

The agent view in AI2

Definition (Actual environment for the agent).
We are now in an environment which is only partially observable and where the agents actions are non-deterministic. Therefore, we have to optimize *expected utility* instead of *actual utility.*
A stateful reflex agent has a world model consisting of

- a **belief state** with information about possible world states.
- a **transition model** that updates the belief state based on sensors and actions.

Definition (Utility-based agent).
A **utility-based agent** uses a world model with a utility function that influences its preferences among the states of that world. It chooses the action that leads to the best expected utility. This is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Definition (Probabilistic agents).
In a partially observable world, the belief model $\hat{=}$ Bayesian network and the inference $\hat{=}$ probabilistic inference.

Definition (Decision-Theoretic agent).
In a partially observable, stochastic world, the belief model + transition model $\hat{=}$ Decision networks and the inference $\hat{=}$ MEU. This agents is a particular kind of utility-based agent.

Probability Theory

Definition (Random variable).
A **random variable** is a variable quantity whose value depends on possible outcomes of unknown variables and processes. Two of the most considered types are *finite-domain* random variables and *boolean* random variables.

Definition (Unconditional probability).
Given a random variable X, $P(X = x)$ denotes the **unconditional probability**, that X has value x in absence of any other information.

Definition (Probability distribution).
The **probability distribution** $P(X)$ for a random variable X is the vector of probabilities for the domain of X.

Example: Weather has the domains sunny, rain, cloudy and snow.
$\Rightarrow P(weather) = \langle 0.7, 0.2, 0.08, 0.02\rangle$

Definition (Event).
A set of outcomes $X = x$ is called **event**.
Given random variables $\{X_1, \ldots, X_n\}$, an **atomic event** is an assignment of values to all variables.

Example of atomic events:
Be A, B boolean random variables, then we have four atomic events: $a \wedge b,\ a \wedge \neg b,\ \neg a \wedge b,\ \neg a \wedge \neg b$

Definition (Joint probability distribution).
Given a subset $Z \subseteq \{X_1, \ldots, X_n\}$ of random variables, an event is an assignment of values to the variables in Z. The **joint probability distribution** $P(Z)$ lists the probability of all events. The **full joint probability distribution** $P(X_1, \ldots, X_n)$ lists the probabilities of all atomic events.

Definition (Conditional probability).
Given propositions A, B where $P(b) \neq 0$, the **conditional probability** is defined as:

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}$$

Definition (Independent).
Two events a, b are **independent** if

$$P(a \wedge b) = P(a, b) = P(a) \cdot P(b)$$

Note (Bayesian Rules).

1. **Product rule:**

$$P(A \wedge B) = P(A \mid B) \cdot P(B)$$

2. **Chain rule:**

$$P(X_1, \ldots, X_n) = P(X_n \mid X_{n-1}, \ldots, X_1) \cdot \ldots \cdot P(X_2 | X_1) \cdot P(X_1)$$

3. **Marginalization** (for all possible value combinations of Y):

$$P(X) = \sum_{y \in Y} P(X, y)$$

Example:

$$P(e) = P(e \mid H) \cdot P(H) + P(e \mid \neg H) \cdot P(\neg H)$$

4. **Normalization:**

$$P(X \mid e) = \alpha P(X, e) \quad \text{mit } \alpha \cdot P(e) = 1$$

Example:

$$1 = \alpha \cdot P(e) = \alpha \cdot \langle 0.4, 0.2\rangle \Leftrightarrow 1 = \alpha(0.4 + 0.2) \Leftrightarrow \alpha = {}^{1}\!/_{0.6}$$

Definition (Bayes' rule).
Given two propositions a, b, we have

$$P(a \mid b) = \frac{P(b \mid a) \cdot P(a)}{P(b)}$$

Definition (Conditional independence).
Given the random variables $Z_1,\ Z_2,\ Z$, we say that Z_1 and Z_2 are **conditionally independent** given Z if

$$P(Z_1, Z_2 \mid Z) = P(Z_1 \mid Z) \cdot P(Z_2 \mid Z)$$

It also holds:

$$P(Z_1 \mid Z_2, Z) = P(Z_1 \mid Z)$$

Further important formula:

- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$
- $P(A \mid B) + P(\neg A \mid B) = 1$

Bayesian Network

Definition (Naive Bayes model)**.**
A Bayesian network in which a single cause directly influences a number of effects, all conditionally independent, given the cause is called a **naive Bayes model**. Here the full joint probability distribution can be written as

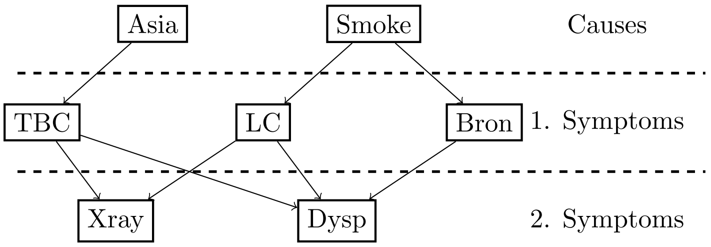
$$P(cause \mid eff_1, \ldots, eff_n) = P(cause) \cdot \prod_i P(eff_i \mid cause)$$

Definition (Bayesian network)**.**
A **Bayesian network** represents the structure of a given domain. Probabilistic inference exploits that structure to compute probabilities and update belief states. Algorithmic examples are Inference by Enumeration and Variable Elimination.
For computing the probability of $P(X \mid e)$ the following variable names are used:

- query variable: X
- evidence variable: e
- hidden variable: all other variables in the network

Drawing a Network:

- Given is a variable order
- For every variable: Insert variable and define its parents $(P(X_i|X_1, ..., X_{i-1}) = P(X_i|Parents(X_i)))$
- Network is optimal, if causes are inserted before symptoms
- Example: Asia, Smoke, TBC, LC, Bron, Xray, Dysp
- Hint: Child nodes are conditionally independent given their parents



Calculating probability (Inference by Enumeration):

- Given: Bayesian Network!
- Evaluates the tree with depth first. Space Complexity is linear. Time Complexity is exponential.
- Build: $P(query \mid evidence) = P(q \mid e)$
  (query has to be 'above' evidence, if not: Bayes Rule)
- Normalize: $P(q \mid e) = \alpha \cdot P(q, e)$
- Marginalize with all hidden variables:

$$P(q \mid e) = \alpha \cdot \sum_{a \in A, b \in B, \ldots} \ldots$$

- Insert product of all variables under their parents:

$$P(q \mid e) = \alpha \cdot \sum_{a \in A, b \in B, \ldots} P(q \mid partents_i) \cdot P(a \mid parents_i) \cdot \ldots$$

Definition (CPT)**.**
Each node X_i is associated with a **conditional probability table (CPT)**, specifying $P(X_i \mid Parents(X_i))$.

Definition (Deterministic nodes)**.**
A node X in a Bayesian network is called **deterministic**, if its value is completely determined by the values of $Parents(X)$..

Definition (Noisy nodes)**.**
Noisy node means that a node is nearly deterministic, so there is not a high probability for other causes.

Definition (Diagnostic and Causal arcs)**.**
In a Bayesian network there are causal and symptom nodes. A causal edge goes from a cause to a symptom. A diagnostic edge goes from a symptom to a cause.
The Bayes rule can be used to compute causal edges instead of diagnostic edges.
Example:
Causal: $P(Smoke \mid Bron)$
Diagnostic: $P(Bron \mid Smoke)$

Definition (Polytree)**.**
A directed acyclic graph is called **polytree**, or singly connected, if the underlying undirected graph is a tree.

Definition (Variable elimination)**.**
Variable elimination is a Bayesian network inference algorithm that avoids repeated and irrelevant computation. In some special cases, this can run in polynomial time (e.g. polytree)
Sketch of ideas:

1. Avoiding repeated computation: Evaluate expressions from right to left, storing all intermediate results.
2. Avoiding irrelevant computation: Repeatedly remove hidden variables that are leaf nodes in the Bayesian network.

Decision Theory

Definition (Decision Theory)**.**
Decision Theory investigates how an agent deals with choosing among actions based on the desirability of their outcomes.
Problem: Because our environment is just partially observable, we do not know the current state.
Idea: Rational decisions equals to choose actions that maximize expected utility.
→ Treat result of an action a as a random variable $R(a)$ whose variables are the possible outcome states.
→ Preferences of the agent are captured in a utility function U.

Definition (Expected utility)**.**
The **expected utility** $EU(a|e)$ of an action a given evidence e can be calculated as

$$EU(a \mid e) = \sum_{s'} P(R(a) = s' \mid a, e) \cdot U(s')$$

Definition (Preferences)**.**
Preferences can be expressed in form of

- $A \succ B$: A is preferred over B
- $A \sim B$: Indifference between A and B
- $A \succeq B$: B is not preferred over A

Preferences are called **rational** if and only if the following constraints hold:

- Orderability: $(A \prec B) \vee (B \prec A) \vee (A \sim B)$
- Transitivity: $(A \prec B) \wedge (B \prec C) \Rightarrow (A \prec C)$
- Continuity: $A \prec B \prec C \;\Rightarrow\; \exists p([p, A; (1-p), C] \sim B)$
- Substitutability: $(A \sim B) \Rightarrow ([p, A; (1-p), C] \sim [p, B; (1-p), C])$
- Monotonicity:
  $(A \preceq B) \Rightarrow [(p \leq q) \Leftrightarrow ([p, A; (1-p), B] \preceq [q, A; (1-q), B])]$

Relation to the utility functions: According to **Ramsey's theorem**, if a given set of preferences obey the constraints above, there is a utility function U such that
$U(A) \geq U(B) \Leftrightarrow A \succeq B$ and $U([p_1, S_1, \ldots, p_n, S_n]) = \sum p_i U(S_i)$

Definition (Value function)**.**
We call a total ordering on states a **value function** or **ordinal utility function**.
⇒ An observer can construct a value function V by observing the agents preferences.

Definition (MEU principle)**.**
The **MEU principle** is to choose the action that maximizes expected utility.

Definition (Utilities)**.**
→ Best possible prize $u_\top$ with probability p.
→ Worst possible catastrophe $u_\bot$ with probability $1-p$.
→ **Normalized utilities**: $u_\top = 1,\ u_\bot = 0$.
→ **Micromorts**: One-millionth chance of death
⇒ Example: Driving a car for 370km incurs a risk of one micromort.

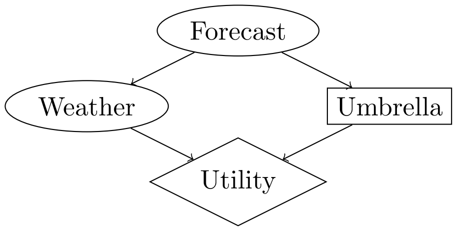
Definition (Decision Networks)**.**
For calculating the utility of an action. First build up a decision network, which contains three types of nodes:

- **Utility nodes:** The resulting utility of a network is represented as a utility node. These are rhombuses (Rauten)
- **Decision nodes:** The action we can decide about. These are rectangles
- **Random nodes:** all the other variables we can not influence directly. These are ellipses

Calculating the utility of the decision node:

- Construct multiple cases for the different values of the decision node (e.g. true, false)
- The utility of the decision node is equal to the sum of the different possible utility states with their probability
- The probability of a utility state is determined with inference by enumeration: Product of all variables under their parents (only random nodes). If those variables can be true or false, a sum for all cases is needed.
- Special Case: decision node goes directly into utility node: Only the utility states, that are fixed by the decision node case have to be considered

Examples:



Utility of Umbrella:

$$U(umb) = P(w|f) \cdot U(umb, w) + P(\neg w|f) \cdot U(umb, \neg w)$$

$$U(\neg umb) = P(w|f) \cdot U(\neg umb, w) + P(\neg w|f) \cdot U(\neg umb, \neg w)$$

Temporal Probability Models

Definition (Temporal probability model)**.**
A **temporal probability model** is a probability model, where possible worlds are indexed by a time structure.
→ X_t = set of unobservable state variables at time t
→ E_t = set of observable evidence variables at time t

Definition (Markov Process)**.**
A **Markov process** is a sequence of random variables with the **Markov property**. Markov property means that X_t only depends on a bounded subset of $X_{0:t-1}$.
→ **First-order Markov process**:

$$P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$$

→ **Second-order Markov process**:

$$P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-2}, X_{t-1})$$

Definition (Transition and Sensor Model)**.**
Random variables in a Markov process are dividable into a set of **state variables** X_t and a set of **evidence variables** E_t. We call $P(X_t \mid X_{t-1})$ the transition model and $P(E_t \mid E_{t-1})$ the sensor model.
A Markov process is **stationary** if the transition model is independent of time.
The sensor model predicts the influence of percepts on the belief state. We say that a sensor model has the **sensor Markov property** if and only if $P(E_t \mid X_{0:t}, E_{0:t-1}) = P(E_t \mid X_t)$.
Assumption here: Sensor Markov property and stationary
⇒ $P(E_t \mid X_t)$ fixed for all t

Definition (Computation with full joint probability)**.**
If we know the initial prior probabilities at $t = 0$, then we can compute the **full joint probability distribution** as

$$P(X_{0:t}, E_{0:t}) = P(X_0) \cdot \prod_{i=1}^{t} P(X_i \mid X_{i-1}) P(E_i \mid X_i)$$

Definition (Filtering)**.**
In **filtering**, we compute the belief state which is input to the decision process of a rational agent, in formula $P(X_t \mid e_{1:t})$.
Computing of filtering from t to t+1:

1. Calculate transition without evidence (forward recursion):
$$P(X_{t+1}) = \sum_{x_t} P(X_{t+1} \mid x_t) \cdot P(x_t)$$
→ The first probability can directly be taken from the transition model!
2. Update with evidence of day t+1:
$$P(X_{t+1} \mid E_{t+1}) = \alpha \cdot P(E_{t+1} \mid X_{t+1}) \cdot P(X_{t+1})$$

Definition (Prediction)**.**
For **prediction**, we evaluate the possible action sequences, in formula $P(X_{t+k} \mid e_{1:t}),\ k > 0$.
This is equivalent to filtering without evidence.
For calculation take just step 1 of filtering and forget the evidence update.

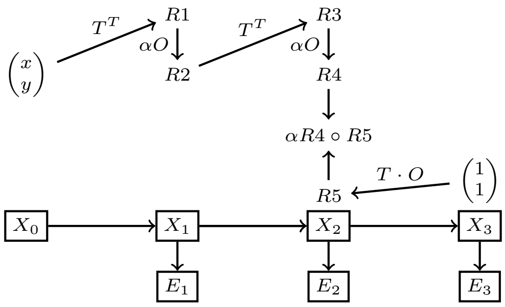
Definition (Smoothing)**.**
With the help of **smoothing**, we can better estimate the past states, which is essential for learning. In formula $P(X_k \mid e_{1:t}),\ 0 \leq k < t$.
Computing smoothing from k+1 to k:

1. Compute backwards recursion:
$$P(e_{k+1:t} \mid X_k) = \sum_{x_{k+1}} P(e_{k+1} \mid x_{k+1}) \cdot P(e_{k+2:t} \mid x_{k+1}) \cdot P(x_{k+1} \mid X_k)$$
→ First and last probability can directly obtained from the model, the second has to be calculated before!
2. Smoothing in k:
$$P(X_k \mid e_{1:t}) = \alpha \cdot P(X_k \mid e_{1:k}) \cdot P(e_{k+1:t} \mid X_k)$$
→ First probability is the result of the filtering in k!

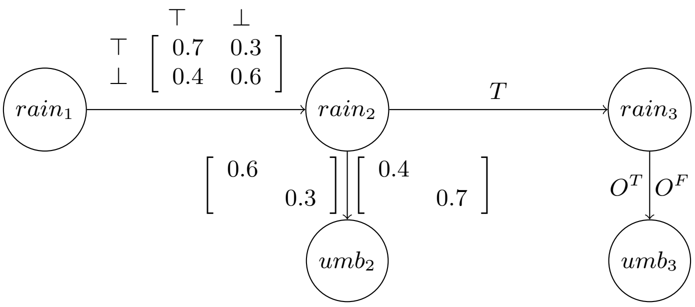
Definition (Most likely explanation)**.**
Most likely explanation is an important task for speech recognition or decoding with a noisy channel, in formula $argmax(P(x_{1:t} \mid e_{1:t}))$.

Definition (Hidden Markov Models)**.**
A **hidden Markov model** is a temporal probabilistic model in which the state of the process is described by a single discrete random variable X_t with domain $\{1, \ldots, S\}$.
Then the transition model can be translated to a **transition matrix** with dimension $S \times S$. The rows of the transition matrix sum up to 1.
The sensor matrix for each time step is a diagonal matrix. There are as many sensor matrices, as there are states for the evidence node. The sum of each row over all matrices sum up to 1.
HMM-Algorithms:

- Filtering: $f_{1:t+1} = \alpha\ (O_{t+1}\ T^{transp}\ f_{1:t})$
- Prediction: $p_{1:t+k} = \alpha\ T^{T^k}\ f_{1:t}$
- Backwards: $b_{k+1:t} = T\ O_{k+1}\ b_{k+2:t}$ with $b_{t:t} = (1, 1, ...)^T$
- Smoothing: $result_k = \alpha(f_{1:k} \circ b_{k+1:t})$



Example:
Aim: Determine whether it rains given the evidence of seeing your boss with an umbrella.
When it rains, it also rains with a probability of 70 % the next day. If it does not rain, it stays this way with 60 %. Your boss takes an umbrella on 60 % of rainy days and with 30 % on non-rainy days.



Definition (Dynamic Bayesian Networks)**.**
A Bayesian network is called **dynamic**, if and only if its random variables are indexed by a time structure.
Assumptions: time sliced and first-order Markov process
⇒ Every HMM is single-variable DBN
→ For inference, unroll the network and do rollup filtering: add slice t+1, sum out slice t using variable elimination.

Complex Decisions

Definition (Sequential decision problems)**.**
In **sequential decision problems**, the agents utility depends on a sequence of decisions which integrates utilities, uncertainty and sensing.
We are in a fully observable, stochastic environment with a Markovian transition model and an additive reward function, calling it **Markov decision process**. It consists of

- A set of states $s \in S$ with initial state $s_0 \in S$
- A set of actions $a \in A(s)$ for each state s
- A transition model $P(s' \mid s, a) \hat{=}$ prob. of a in s leads to s'
- A reward function $R : S \to \mathbb{R}$ with reward $R(s)$.

Aim is to find an optimal policy $\pi(s)$, i.e the best possible action for every possible state s.

Definition (Utility of state sequences)**.**
We need to understand preferences between sequence of states. For stationary preferences [1], there are only two ways to combine rewards over time:

- additive rewards: $U([s_0, s_1, \dots, s_n]) = \sum_{i=0}^{n} R(s_i)$
- discounted rewards: $U([s_0, s_1, \dots, s_n]) = \sum_{i=0}^{n} \gamma^i R(s_i)$

Definition (Utility of states)**.**
Utility of states is equivalent to the expected discounted sum of rewards assuming optimal actions.
The expected utility obtained by executing π starting in s is given by

$$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

with $\pi_s^* = argmax(U^\pi(s))$ independent from s. (optimal policy)
The utility $U(s)$ of a state s is $U^{\pi^*}(s)$

Definition (The Bellman equation)**.**
Definition of the utility od states leads to a simple relationship among utilities of neighboring states:

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

- γ: discount factor between 0 and 1, that tells how important future rewards are (0 = no importance, 1 = max importance)
- $R(s)$: reward of the actual state
- Rest: MEU-Principle of expected utility

Definition (Value iteration algorithm)**.**
Idea: Use a simple iteration scheme to find a fixpoint:

1. Start with random utility values
2. Update them to make them locally consistent with Bellman equation
3. If it is locally consistent everywhere, it is global optimal

Algorithm 1 Value Iteration

```
Input: States, Actions, Transition model, rewards, discount $\gamma$,
       max error $\epsilon$
repeat
    $U = U'$; $\delta = 0$
    for each state $s \in S$ do
        $U'(s) = R(s) + \gamma \max_a \sum_{s'} U(s') \cdot P(s' \mid s, a)$
        if $|U'(s) - U(s)| > \delta$ then $\delta = |U'(s) - U(s)|$
until $\delta < \epsilon(1 - \gamma)/\gamma$
```

Definition (Policy Iteration algorithm)**.**
Idea is to search for optimal policy and utility values simultaneously:

- Policy evaluation: given a policy π_i, calculate U^{π_i} for every state, if π_i is executed
- Policy improvement: calculate a new MEU policy π_{i+1} using a 1-lookahead based on U^{π_i}

⇒ Terminates, if policy change yields no further improvement for utilities.

Algorithm 2 Policy Iteration

```
Input: mdp = States, Actions, Transition model
Initialize U with zero for each state and choose policy $\pi$ randomly
repeat
    $U = Policy - evaluation(\pi, U, mdp)$
    unchanged = true
    for each state $s \in S$ do
        if $\max_a(\sum_{s'} P(s'|s,a)U(s')) > \sum_{s'} P(s'|s,\pi[s'])U(s')$ then
            $\pi[s] = argmax \sum_{s'} P(s' \mid s, a)U(s')$
            unchanged = false
until unchanged
return $\pi$
```

Implementation of Policy-evaluation: Using Bellman equation, but without max. Instead of a, we use policy $\pi_i[s]$
→ Often converges in a few iterations, but each is expensive.
→ Policy iteration has the advantage, that in the Bellman

[1] $[r, r_0, r_1, \dots] > [r, r'_0, r'_1, \dots] \Leftrightarrow [r_0, r_1, \dots] > [r'_0, r'_1, \dots]$

equation, the action is fixed by the policy. For example, in the i^{th} iteration with policy π_i, we just have to calculate the action $\pi_i[s]$ in state s.

Definition (Partially observable MDP)**.**
A **partially observable MDP** is a MDP together with an observation model O that is stationary and has the sensor Markov property: $O(s, e) = P(e|s)$.
The optimal policy in this context is a function $\pi(b)$, where b is the belief state.
Update of the belief state:

$$b'(s') = \alpha P(e|s') \sum_s P(s'|s, a) \cdot b(s)$$

Observe: Not just physical states can change the belief states, also actions.
⇒ Filtering updates the belief state for new evidence.
→ Introducing belief states representing the probability distribution over the physical state space, we can reduce partially observable MDPs to normal MDPs.
⇒ Equivalent to MDP on belief state!

Definition (Dynamic decision networks)**.**
Given transition and sensor models represented as a dynamic Bayesian network, action nodes and utility nodes have to be added to create a **dynamic decision network**.
A filtering algorithm is used to incorporate each new percept and action and to update the belief state representation. Decisions are made by projecting forward possible action sequences and choosing the best one.

Machine Learning

Definition (Inductive learning)**.**
The **inductive learning problem** $P = \langle H, f\rangle$ consists in finding a hypothesis $h \in H$ and a target function f of examples. An example is a pair (x, y) of an input sample x and an outcome y.
A set S of examples is consistent, if S is a function.
A hypothesis is consistent with target f, if it agrees with it on all examples (e.g. Curve fitting).
→ Whether we can find a consistent hypothesis depends on the chosen space
⇒ To large space leads to high computational complexity
→ Simplest form: learn a function from examples
→ Highly simplified model of real learning (no knowledge, examples given . . .)

Definition (Learning decision trees)**.**
In **attribute-based representations**, examples are described by attributes, their values, and outcomes. A **decision tree** for a given attribute-based representation is a tree, where non-leaf nodes are attributes, their arcs are corresponding attribute values and the leaf nodes are labeled by the outcomes.
→ It is preferable to find more compact decision trees.
⇒ Idea: Choose most significant attribute as root of the subtree.

Algorithm 3 Decision tree learning DTL

Input: examples, attributes, default, target

```
if no example left then return default
else if all examples have same target-value then
    return target-value
else if no attributes left then
    return most-frequent-target-value(example)
else
    best = Attribute with highest information gain
    tree = new subtree with root best
    m = most-frequent-target-value(example)
    for all values $v_i$ of best do
        $examples_i$ = examples with best=$v_i$
        subtree = DTL($examples_i, attributes \setminus best, m$)
        add branch to tree with arc label $v_i$ and subtree
    return tree
```

Definition (Information Gain)**.**
Entropy:

$$I(\langle P_1, \ldots, P_n\rangle) = \sum_{i=1}^{n} -P_i \log_2(P_i)$$

Entropy is a measurement for the information of a block of data.
Important values: $I(\langle 1, 0\rangle) = 0bit,\ I(\langle 1/2, 1/2\rangle) = 1bit$
Information Gain of an attribute:

$$Gain(A) = \underbrace{I(\langle \frac{p}{p+n}, \frac{n}{p+n}\rangle)}_{Entropy\ of\ actual\ root} - \underbrace{\sum_{i=1}^{n} \frac{p_i + n_i}{p+n} I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\rangle)}_{\substack{Expected\ number\ of\ bits \\ per\ example\ over\ all\ branches}}$$

Example: Decision tree of renting an apartment

Rooms	Kitchen	Acceptable
3	yes	yes
3	no	no
4	no	yes
3	no	no
4	no	yes

Rooms and Kitchen are attributes, Acceptable is the target. The entries of the columns are values. A row is called an example.
Entropy for the whole set regarding the target:

$$I(\langle \frac{3}{5}, \frac{2}{5}\rangle) = -\frac{3}{5} \cdot \log_2(\frac{3}{5}) - \frac{2}{5} \log_2(\frac{2}{5}) \approx 0.97bit$$

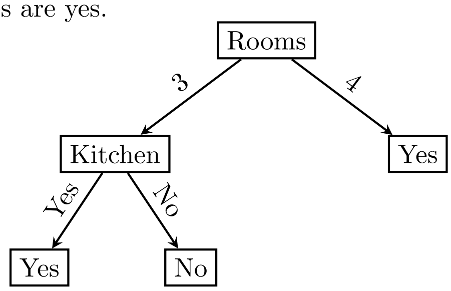
Information gain for rooms is bigger than kitchen:

$$Gain(Rooms) = \underbrace{I(\langle \frac{3}{5}, \frac{2}{5}\rangle)}_{Entropy\ root} - \underbrace{\frac{2}{5} I(\langle 1, 0\rangle)}_{Rooms=4} - \underbrace{\frac{3}{5} I(\langle \frac{1}{3}, \frac{2}{3}\rangle)}_{Rooms=3} = 0.42bit$$

Information gain for the case rooms = 3, kitchen:

$$Gain(Kitchen) = I(\langle \frac{1}{3}, \frac{2}{3}\rangle) - \frac{1}{3} I(\langle 1, 0\rangle) - \frac{2}{3} I(\langle 0, 1\rangle) = 0.92bit$$

Information gain for the case room = 4 is not computed, because all goalvalues are yes.



Definition (Decision tree pruning)**.**
For **decision tree pruning** repeat the following on a learned decision tree:

- Find terminal test node (only ancestor of leaves)

- If information gain was low, prune it by replacing it by a leaf node

A result has **statistical significance**, if the probability they could arise from the null hypothesis is very low (usually 5%).
⇒ For decision tree pruning, the null hypothesis is that the attribute is irrelevant

Definition (PAC learning)**.**
Basic idea of Computational Learning Theory:

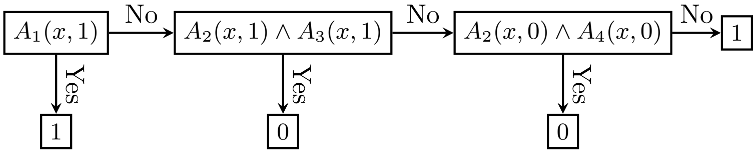
- Any hypothesis h that is seriously wrong, will almost certainly be revealed after a small number of examples, because it will make an incorrect prediction
- Thus, if h is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong.
  ⇒ h is probably approximately correct

Any learning algorithm that returns hypotheses that are probably approximately correct is called a **PAC learning algorithm**.
→ Problem: PAC learning for Boolean functions needs to see (nearly) all examples.
⇒ Ways out: prior knowledge, simple hypothesis (e.g decision tree pruning) or focus on learnable subsets.

Definition (Decision lists)**.**
Idea: Apply PAC learning to a 'learnable hypothesis space'.
A **decision list** consists of a sequence of tests, each of which is a conjunction of literals. The set of decision lists where tests are a conjunction of at most k literals is called k-DL.
→ Test succeed: Stop with return value
→ Test fails: Continue with next test in list
Decision list learning algorithm scheme
Greedy algorithm that repeats the following steps:

1. find test that agrees exactly with some subset E of the target
2. add it to the decision list under construction and remove E
3. construct the remainder of the DL using just the remaining examples

Example:



⇒ Like decision trees, but restricted branching and more complex tests.

Definition (Gradient Descent Method)**.**
The **gradient descent algorithm** for finding a minimum of a continuous function f is hill-climbing in the direction of the steepest descent, which can be computed by partial derivatives of f.
→ Used for continuous target functions f!
Explanation of the algorithm:
It inputs a differentiable function and initial weights. Until w converges, it takes steps into the direction of the greatest descend, restricted by a parameter α, which is also called learning rate.

Definition (Linearly separable)**.**
A linear decision boundary is called a linear separator and data that admits one are called **linearly separable**. A **decision boundary** is a line that separates two classes of points.

Definition (Logistic regression)**.**
The process of weight-fitting in $h_w(x) = \frac{1}{1+e^{-wx}}$ called logistic regression.
→ Learning via uncontinuous functions is often unpredictable
⇒ Approximate with a differentiable function

Definition (Over- and underfitting)**.**
We speak of **overfitting**, if a hypothesis h describes random error rather than the underlying relationship. **Underfitting** occurs when h cannot capture the underlying trend of the data.
⇒ Overfitting increases with the size of hypothesis space and the number of attributes, but decreases with number of examples.
→ Disadvantage of overfitting: Has to learned to much by the examples, it is harder to have general learning.
→ Use overfitting to generalize decision trees → prune nodes

Neuronal Networks

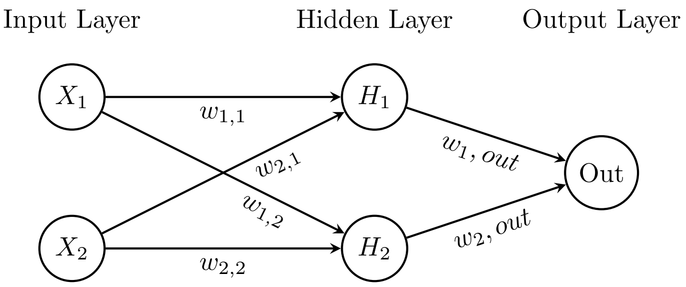
Definition (Neuronal networks)**.**
The AI sub-field of **neural networks** studies computing systems inspired by the biological neural networks that constitute brains.
An **artificial neural network** is a directed graph of units and links. A link from unit i to unit j propagates the activation a_i from unit i to unit j, it has a weight w_i, j associated with it.
A **McCulloch-Pitts unit** first computes a weighted sum of all inputs and then applies an activation function g to it. If g is a threshold function, we call the unit a perceptron unit, if g is a logistic function a sigmoid perceptron unit.

Definition (Feed-forward networks)**.**
A neural network is called a **feed-forward network**, if it is acyclic. Feed-forward networks are usually organized in layers, where edges only connect nodes from subsequent layer. The first layer is called **input layer**, the last **output layer** and every other layer is called **hidden layer**.
→ Opposite are recurrent networks, which have directed cycles.

Definition (Perceptrons)**.**
A perceptron network is a feed-forward network of perceptron units. A single-layer perceptron network is called a **perceptron**.
→ All input units are directly connected to output units
→ Output units all operate separately, no shared weights

Definition (Perceptron learning)**.**
Idea: Learn by adjusting weights in w to reduce generalization loss on training set.
→ Compute with the squared error loss of a weight vector w for an example (x,y)
→ Perform optimization search by gradient descent for any weight w_i
→ Use a simple weight update rule
⇒ Perceptron learning rule converges to a consistent function for any linearly separable data set

Example of a 2-layer feed-forward network: XOR-network



→ A perceptron with $g = step\_function$ can express AND, OR, NOT, but not XOR
⇒ Input space is linearly separable for the perceptrons!

Definition (Multilayer perceptrons an learning)**.**
A Perceptron with at least one additional hidden layer is called **multilayer perceptron**.
Idea for learning: Learn by adjusting weights to reduce error on training set.
Problem: Neuronal networks have multiple outputs, but we can compute the squared error loss of a weight matrix for an example (x,y)
⇒ Output layer is analogous to that for single-layer perceptron,

but multiple output units

Problem: For the hidden layers examples do not say anything about them!

Idea: back-propagate the error from the output layer!

Definition (Back-propagation process)**.**
The **back-propagation process** can be summarized as follows:

1. Compute the Δ values for the output units, using the observed error
2. Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
   (a) Propagate the Δ values back to the previous (hidden) layer
   (b) Update the weights between the two layers

⇒ Kind of gradient descent procedure

→ Applications: speech, driving, handwriting