Symbolic Machine Learning

Lecture Slides

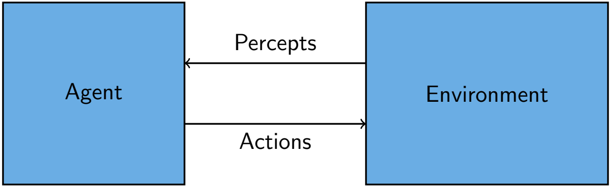
Filip Železný

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

A General Framework

Agent Interacts with Environment



- Discrete *time*

$$k = 1, 2, \ldots$$

- *Percepts*

$$\forall k : x_k \in X$$

- *Actions*

$$\forall k : a_k \in A$$

Histories, Probability, Agent's Policy

Interaction or *history* up to time m which we denote as $xa_{\leq m}$

$$xa_{\leq m} = x_1, a_1, x_2, a_2, \ldots, x_m, a_m$$

starts with a percept (arbitrary choice but stick to it) and has *probability*

$$P(xa_{\leq m}) = P(x_1)P(a_1|x_1)P(x_2|x_1, a_1)\ldots P(x_m|xa_{<m})P(a_m|x_m, xa_{<m})$$

The $P(x_1)$ and $P(x_k|.)$ factors depend on the stochastic environment while $P(a_k|.)$ factors depend on the agent. We will only assume *deterministic agents* so

$$P(a_k|xa_{<k}, x_k) = \begin{cases} 1 \text{ if } a_k = \pi(xa_{<k}, x_k) \\ 0 \quad \text{otherwise} \end{cases}$$

where $\pi(xa_{<k}, x_k)$ is the agent's *policy*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards

The environment rewards the agent depending on its past actions. Formally, *rewards* $r_k \in R \subset \mathbb{R}$ are a distinguished part of percepts

$$x_k = \langle o_k, r_k \rangle$$

while everything else in the percepts are *observations* $o_k \in O$. The set R of possible rewards must be *bounded*, i.e., for some $a, b \in \mathbb{R}$ and all $r \in R$, $a \leq r \leq b$.

Notes:

- Rewards, as part of percepts, generally depend on the entire history. A long sequence of 'good' actions may be needed for a high reward. Example: chess-game with the only 'win' reward at the end.
- r_1 is immaterial.

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Policy Value

The agent's goal is to maximize the value V^π of its policy π, defined as:

- for a *finite* time horizon $m \in \mathbb{N}$, the expected cumulative reward

$$V^\pi = \mathbb{E}\left(\sum_{k=1}^{m} r_k\right) = \sum_{x_{\leq m}} P(x_{\leq m}|a_{<m})(r_1 + r_2 + \ldots + r_m)$$

Remind that $x_k = (o_k, r_k)$ and $a_k = \pi(x_{\leq k})$.

- for the *infinite* horizon, use a *discount sequence* δ_k such that $\sum_{i=1}^{\infty} \delta_i < \infty$ (usually $\delta_k = \gamma^k$, $0 < \gamma < 1$), and maximize

$$V^\pi = \mathbb{E}\left(\sum_{k=1}^{\infty} r_k \delta_k\right) = \lim_{m\to\infty} \sum_{x_{\leq m}} P_R(r_{\leq m}|a_{<m}) \sum_{k=1}^{m} r_k \delta_k$$

Policy Value: Remarks

$P(x_{\leq m}|a_{<m})$ means the probability of sequence $x_1, \ldots, x_m$ given actions $a_1, \ldots, a_{m-1}$. If the environment did *not* depend on agent's actions, we would have just

$$P(x_{\leq m}) = P(x_1)P(x_2|x_1) \ldots P(x_m|x_{m-1}, \ldots, x_1)$$

with actions, we have

$$P(x_{\leq m}|a_{<m}) = P(x_1)P(x_2|x_1, a_1) \ldots P(x_m|x_{m-1}, a_{m-1} \ldots, x_1, a_1)$$

So, e.g., for $m = 2$, the value of policy π would be:

$$V^{\pi} = \mathbb{E}\left(\sum_{k=1}^{2} r_k\right) = \sum_{x_1, x_2} P(x_1)P(x_2|x_1, a_1)(r_1 + r_2)$$

V^{π} where the sum is over all possible sequences x_1, x_2 of percepts.

Policy Value: Remarks (cont'd)

Remind that the rewards r_k are components of the percepts $x_k = \langle o_k, r_k \rangle$ and that $a_k = \pi(xa_{<k}, x_k)$ so we can write fully explicitly (still for $m = 2$):

$$V^\pi = \sum_{\langle o_1, r_1 \rangle, \langle o_2, r_2 \rangle} P\left(\langle o_1, r_1 \rangle\right) P(\langle o_2, r_2 \rangle \,|\, \langle o_1, r_1 \rangle, \pi(\langle o_1, r_1 \rangle))(r_1 + r_2)$$

For an infinite horizon, we just need to multiply r_k by a coefficient that decays sufficiently quickly, e.g. γ^k and then

$$V^\pi = \lim_{m \to \infty} \mathbb{E}\left(\sum_{k=1}^{m} \gamma^k r_k \right)$$

Markovian Environments

A *Markovian* or *state-based* environment is one for which a *state* variable $s : \mathbb{N} \to S$ and distributions P_x,P_S exist such that S has bounded size and

- $P(x_k|xa_{<k}) = P_x(x_k|s_k)$, i.e., x_k depends only on the current state. The assumption is strong because S is bounded and cannot contain a state for each possible history $xa_{<k}$ as $k \to \infty$.
- State s_{k+1} $(k \geq 1)$ is distributed according to $P_S(s_{k+1}|s_k, a_k)$, i.e., it depends only the previous state and the action taken on it by the agent. The initial state is distributed by $P_S(s_1)$.

Note: since $P_x(x_k|s_k) = P_x(\langle o_k, r_k \rangle |s_k)$, both o_{k+1} and r_{k+1} depend solely on the state s_{k+1}. Sometimes it is more convenient to model the reward r_{k+1} as distributed by $P_r(r_{k+1}|s_k, a_k)$ instead and we will use both options.

Markovian Agents

A *Markovian* or *state-based* agent is one for which a *state* variable $t : \mathbb{N} \to T$ and *functions* π,$\mathcal{T}$ exist such that T has bounded size and

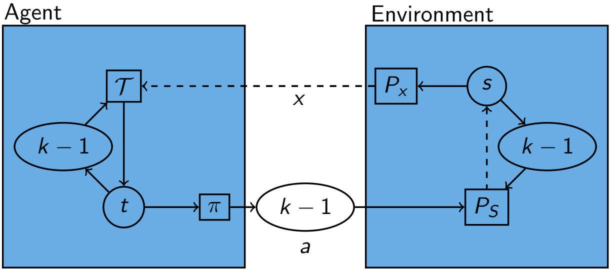
- $\pi(x_{\leq k}) = \pi(t_k, x_k)$. Since T is bounded, some different histories will result in the same action of the agent. One can view t_k as agent's flexible (learnable) decision model while π its fixed interpreter.
- For $k > 1$, $t_{k+1} = \mathcal{T}(t_k, x_{k+1})$, i.e., it depends only on the previous state and the latest percept (t_1 is some initial state). $\mathcal{T}$ is the state update function, which will be the core of learning.

Note: since $\pi(t_k, x_k) = \pi(t_k, (o_k, r_k))$, the policy depends also on r_k. Usually, it suffices to consider dependence only on o_k by $\pi(t_k, o_k)$. And since both o_k, r_k can be stored as part of t_k by the update function, one may even use $\pi(t_k)$ as done on the next page.

Markovian Interaction Model

We now have a general structure in which to study learning:

Agent		Environment	
actions:	$a = \pi(t_k)$	percepts:	$x_k \sim P_x(x_k \vert s_k)$
state update:	$t_k = \mathcal{T}(t_{k-1}, x_k)$	state update:	$s_k \sim P_S(s_k \vert s_{k-1}, a_{k-1})$



Terminal States, Proper Policies

We can distinguish some states from S as *terminal*. If s_k is terminal then s_{k+1} is sampled independently of s_k, a_k from $P_s(s_{k+1})$, i.e. just like the initial state s_1. The interaction history between a terminal (or initial) state and the next terminal state is called an *episode*.

Informally, the environment is 'restarted' after a terminal state. However, the agent is not restarted $(t_{k+1} = \mathcal{T}(t_k, x_{k+1}))$, so it can learn from one episode to another.

For a given environment, a policy π is *proper* if it is guaranteed to achieve a terminal state.

With a proper policy, we can modify the agent's goal as to maximize $\mathbb{E}\left(\sum_{k=1}^{m} r_k\right)$ where s_m is the first terminal state in the interaction (no need for a discount factor).

Reinforcement Learning

Reinforcement Learning

Reinforcement learning is a collection of techniques by which the agent achieves high rewards in the state-based (Markovian) setting under two assumptions:

- Environment *fully observable*, i.e., $O = S$ and for $\forall k$: $o_k = s_k$.
- Reward is a *function* of current state: $\forall k : r_k = r(s_k)$.

There is no P_X anymore because percepts are here a function of states

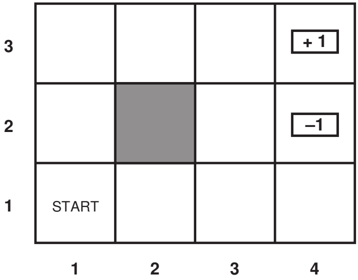
$$x_k = (o_k, r_k) = (s_k, r(s_k))$$

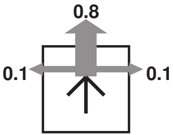
The environment is described by the update (transition) distribution P_S and the reward function r, both of which are unknown to the agent.

Reinforcement Learning: Example

Agent should learn to get from START to the $+1$ goal in the grid world. The -1 goal should be avoided.

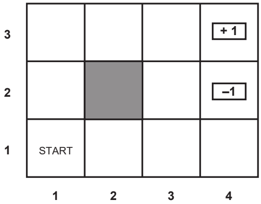
Effects of actions (moves) are uncertain.





Images in the RL part: AIMA book (Russel, Norvig), RL Book (Sutton, Barto)

Formalizing the Example in the Markovian Setting



Env. states $S = \{\,1,2,3,4\,\} \times \{\,1,2,3\,\} \setminus \{\,(2,2)\,\}$ correspond to agent's positions on the grid. States $(4,3)$ and $(4,2)$ are terminal, with respective rewards 1 and -1.

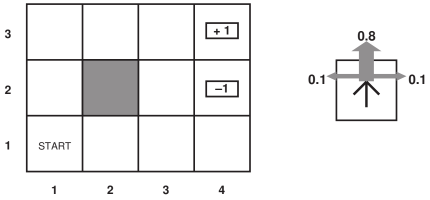
Percepts $X = S \times R$

Agent states T encode possible agent's decision models. Depend on the chosen implementation of an agent (we will study that).

Actions $A = \{\,\mathrm{left}, \mathrm{right}, \mathrm{up}, \mathrm{down}\,\}$.

Formalizing the Example (cont'd)

Actions have uncertain effects on the environment state.



Prescribed by distribution $P_S(s_{k+1}|s_k, a_k)$. Here e.g.

$$P_S(\langle 3,2\rangle \,|\, \langle 3,1\rangle\,, \mathrm{up}) = 0.8$$
$$P_S(\langle 2,1\rangle \,|\, \langle 3,1\rangle\,, \mathrm{up}) = 0.1$$
$$P_S(\langle 4,1\rangle \,|\, \langle 3,1\rangle\,, \mathrm{up}) = 0.1$$

Bouncing: if outcome s_{k+1} out of grid, then $s_{k+1} := s_k$.

Agent State, Fixed Policy

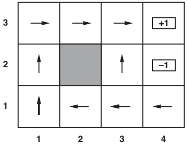
Agent state t_k contains the agent's model at time k, prescribing the action (decision) policy

$$a_k = \pi(t_k, s_k)$$

In the simplest case, t_k may be a static lookup table Π (see on right)

$$\pi(\Pi, s_k) = \Pi[s_k]$$

When the agent state does not change with k as here, we call the policy *fixed*. Of course, fixed policy means no learning. In this case we can omit the first argument, writing just $\pi(s_k)$.



s	$\Pi[s]$
$\langle 1, 1 \rangle$	up
$\langle 1, 2 \rangle$	up
$\langle 1, 3 \rangle$	right
$\langle 2, 1 \rangle$	left
$\langle 2, 3 \rangle$	right
$\langle 3, 1 \rangle$	left
$\langle 3, 2 \rangle$	up
$\langle 3, 3 \rangle$	right
$\langle 4, 1 \rangle$	left

State Utility Under a Fixed Policy

Given a fixed policy π, how good is it to be in state s_k at time k? The better, the higher the *expected utility* of the state (under that policy):

$$U^{\pi}(s_k) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r(s_{k+i})\right] = r(s_k) + \gamma U^{\pi}(s_{k+1})$$

where $0 < \gamma < 1$ is the discount factor. If π is proper, we can have $\gamma = 1$, summing only up to the first terminal state s_{k+i}.

Since s_{k+1} $(k \geq 1)$ are distributed by $P_S(s_{k+1}|s_k, a_k)$, we can write this as

$$U^{\pi}(s_k) = r(s_k) + \gamma \sum_{s \in S} P_S(s|s_k, \pi(s_k))U^{\pi}(s)$$

Optimal Policy, State Utility

$$\pi^* = \arg\max_{\pi} U^{\pi}(s)$$

is called the *optimal policy*.

Which policy $\pi : S \to A$ is optimal depends on a state s by this definition. However, it can be shown that any $s \in S$ yields the same π^*.

Considering the definition of $U^{\pi}(s_k)$, π^* maps each s_k $(k > 1)$ to an action maximizing the expected utility of the next state

$$\pi^*(s_k) = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a)U(s)$$

$U(s) = U^{\pi^*}(s)$ is called the *state utility* (without adjectives).

Computing an Optimal Policy

So *if the agent knows P_S and r*, it can decide optimally by

$$a_k = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U(s)$$

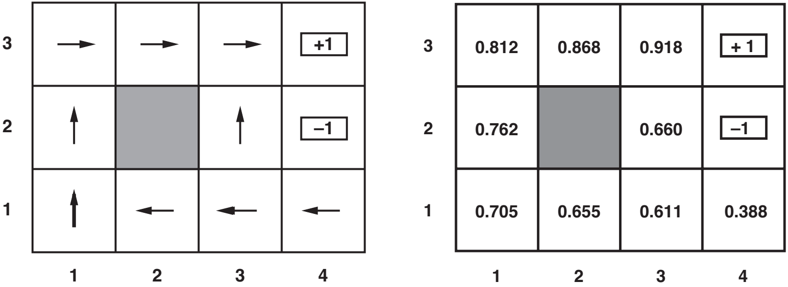
For this, it first needs to compute $U(s)$ for all $s \in S$. They are solutions of $|S|$ non-linear *Bellman* equations (one for each $s \in S$)

$$U(s) = r(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_S(s'|s, a) U(s')$$

These can be solved by the *value iteration* algorithm known from the theory of *Markov Decision Processes*.

(Note: P_S, r, S, A, γ define an MDP, π^* is its solution.)

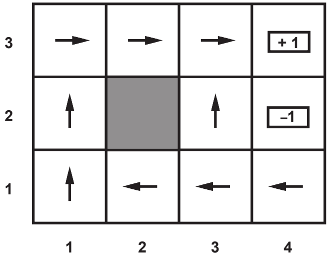
Optimal Policy and State Utilities



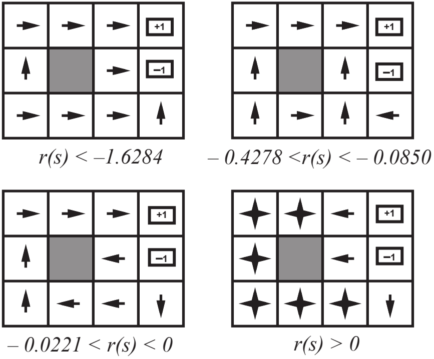
	1	2	3	4
3	0.812	0.868	0.918	+ 1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388

Optimal policy (left) and state utilities (right) for $\gamma = 1$ and $r(s) = -0.04$ for all non-terminal states s

Optimal Policies Under Different Rewards



Optimal policy for $\gamma = 1$ and $r(s) = -0.04$ for non-terminal states s.



$r(s) < -1.6284$

$-0.4278 < r(s) < -0.0850$

$-0.0221 < r(s) < 0$

$r(s) > 0$

Optimal policies for $\gamma = 1$ and different ranges of $r(s)$ for non-terminal states s.

Learning an Optimal Policy

What if P_s and r are not known?

1. If the agent knows the state utilities, it can determine an optimal policy.
2. State utilities can be estimated by interacting with the environment. But for this interaction, some policy is needed.

So 1 and 2 must be somehow interleaved.

We will first look at 2. The agent will be prescribed a fixed policy. Such an agent is called *passive*.

Then we will see how to combine 1 with 2.

Passive Direct Utility Estimation Agent

- Follows a fixed proper policy π - see example on right. Policy formally extended with end actions to indicate terminal states (needed for later pseudo-codes).
- With $\gamma = 1$, agent's estimate of $U^{\pi}(s)$ for state s at time k is the average of all *rewards-to-go* of s until k.
- A reward-to-go of s is the sum of rewards from s till the end of the current episode. If s visited multiple times in one episode, then that episode produces multiple rewards-to-go to include in the average.

s	$\Pi[s]$
$\langle 1,1 \rangle$	up
$\langle 1,2 \rangle$	up
$\langle 1,3 \rangle$	right
$\langle 2,1 \rangle$	left
$\langle 2,3 \rangle$	right
$\langle 3,1 \rangle$	left
$\langle 3,2 \rangle$	up
$\langle 3,3 \rangle$	right
$\langle 4,1 \rangle$	left
$\langle 4,2 \rangle$	end
$\langle 4,3 \rangle$	end

Example: estimate utility of state $(1,2)$ over 3 episodes in the grid:

$(1,1)_{-.04} \to (1,2)_{-.04} \to (1,3)_{-.04} \to (1,2)_{-.04} \to (1,3)_{-.04} \to (2,3)_{-.04} \to (3,3)_{-.04} \to (4,3)_{+1} \ldots$ 0.76, 0.84
$(1,1)_{-.04} \to (1,2)_{-.04} \to (1,3)_{-.04} \to (2,3)_{-.04} \to (3,3)_{-.04} \to (3,2)_{-.04} \to (3,3)_{-.04} \to (4,3)_{+1} \ldots$ 0.76
$(1,1)_{-.04} \to (2,1)_{-.04} \to (3,1)_{-.04} \to (3,2)_{-.04} \to (4,2)_{-1} \ldots$ no occurrence

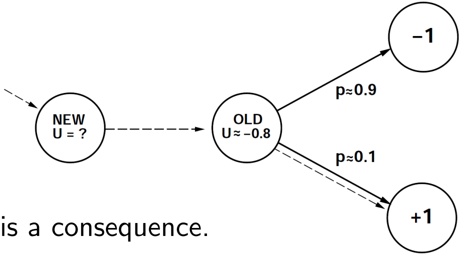
So the estimate is $(0.76 + 0.84 + 0.76)/3$.

Passive DUE Agent: Properties

The DUE Agent does not make use of the known dependence between state utilities

$$U^\pi(s_K) = r(s_K) + \gamma \sum_{s \in S} P_S(s|s_K, \pi(s_K))U^\pi(s)$$

E.g. below the newly explored state will likely have low utility due to the neighbor state (already explored). The agent does not 'know' this.



Slow convergence is a consequence.

Passive Adaptive Dynamic Programming Agent

Instead of computing $\widehat{U}$ directly from samples, learn a model of P_S and r and compute $\widehat{U}$ from them.

For r, just collect an array $\widehat{r}[s]$ of observed rewards for observed states.

For $P_S(s'|s, a)$, collect the counts $N[s', s, a]$ of observed triples of action a taken in state s and resulting in state s'. Then estimate:

$$P_S(s'|s, a) \approx \frac{N[s', s, a]}{\sum_{s'' \in S} N[s'', s, a]}$$

The *policy evaluation* algorithm (as known from MDP's) takes $\widehat{r}$ and N, and produces $\widehat{U}$.

Policy Evaluation: Reminder of Pre-Requisite Material

State utilities should satisfy

$$U^{\pi}(s) = r(s) + \gamma \sum_{s' \in S} P_S(s'|s, \pi(s)) U^{\pi}(s')$$

The policy evaluation plugs in the model $\frac{N[s',s,a]}{\sum_{s'' \in S} N[s'',s,a]}$ and $\widehat{r}$ of $P_S(s'|s,a)$ and r, and calculates $\widehat{U}$ by *value iteration* for all $s \in S$ until convergence:

$$\widehat{U}[s] \leftarrow \widehat{r}[s] + \gamma \sum_{s' \in S} \frac{N[s', s, \pi(s)]}{\sum_{s'' \in S} N[s'', s, \pi(s)]} \widehat{U}[s']$$

This is a system of *linear* assignments, so instead of iterating them, the corresponding equations can be solved through matrix algebra in $\mathcal{O}(|S|^3)$ time.

Passive ADP Agent: Design

Agent's state is a tuple

$$t_k = \left\langle \Pi, s_k^{\mathrm{old}}, N_k, \widehat{r}_k, \widehat{U}_k \right\rangle$$

- Π: fixed decision array (as in the DUE agent)
- s_k^{old}: last seen state
- N_k: 3-way contingency array indexed by $[s' \in S, s \in S, a \in A]$.
- $\widehat{r}_k$: reward array indexed by $s \in S$.
- $\widehat{U}_k$: state utility estimate array indexed by $s \in S$.

The last four variables are initially $(k = 1)$ filled with the none value.

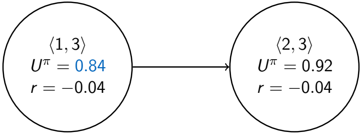
Passive ADP Agent: Design (cont'd)

Update step $t_{k+1} = \mathcal{T}(t_k, x_k)$ where $t_k = \left\langle \Pi, s_k^{\mathrm{old}}, N_k, \widehat{r}_k, \widehat{U}_k \right\rangle$ and $x_k = (r_k, s_k)$:

$s_{k+1}^{\mathrm{old}} = s_k$	Current state is stored for use at next update.
$N_{k+1}[s_k, s_k^{\mathrm{old}}, \Pi[s_k^{\mathrm{old}}]]$ $= N_k[s_k, s_k^{\mathrm{old}}, \Pi[s_k^{\mathrm{old}}]] + 1$	Contingency array is incremented.
$\widehat{r}_{k+1}[s_k] = r_k$	Reward observed for the current state is stored.
$\widehat{U}_{k+1} =$ $\mathrm{policy\_eval}(N_{k+1}, \widehat{r}_{k+1})$	Utilities are estimated by the policy evaluation algorithm.

Passive Temporal Difference Agent

In the passive *temporal difference agent*, the expensive policy evaluation of the ADP agent is replaced by only local changes.



If there was no other transition from $(1,3)$, then with $\gamma = 1$, $U^\pi((1,3))$ should be changed to

$$\widehat{U}[(1,3)] \leftarrow -0.04 + \gamma \widehat{U}[(2,3)] = 0.88$$

In the general case, we make a small iteration for each executed transition:

$$\widehat{U}_{k+1}[s_k] = \widehat{U}_k[s_k] + \alpha \left(r_k + \gamma \widehat{U}_k[s_{k+1}] - \widehat{U}_k[s_k] \right)$$

where α decreases with the number of times s_k has been visited.

Passive TD Agent: Design

Agent's state is a tuple

$$t_k = \left\langle \Pi, s_k^{\mathrm{old}}, r_k^{\mathrm{old}}, N_k, \widehat{U}_k, \alpha \right\rangle$$

- Π: fixed decision array (as in the DUE and ADP agents)
- s_k^{old}: last seen state, $s_1^{\mathrm{old}} = \mathrm{none}$
- r_k^{old}: last reward, $r_1^{\mathrm{old}} = 0$
- N_k: state frequency array addressed by s, N_1 filled with zeros.
- $\widehat{U}_k$: state utility estimate array addressed by $s \in S$. $\widehat{U}_1$ filled with none.
- $\alpha : \mathbb{N} \to \mathbb{R}$: a positive, monotone decreasing function

Note: no model of P_S or r! r_k^{old} just remembers a single (last state) reward.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Passive TD Agent: Design (cont'd)

Update step $t_{k+1} = \mathcal{T}(\left\langle \Pi, s_k^{\mathrm{old}}, r_k^{\mathrm{old}}, N_k, \widehat{U}_k, \alpha \right\rangle, (r_k, s_k))$

$s_{k+1}^{\mathrm{old}} = s_k,\ r_{k+1}^{\mathrm{old}} = r_k$ Current observation and reward are stored for use at next update.

$N_{k+1}[s_k] = N_k[s_k] + 1$ Frequency array is is incremented for $s = s_k$

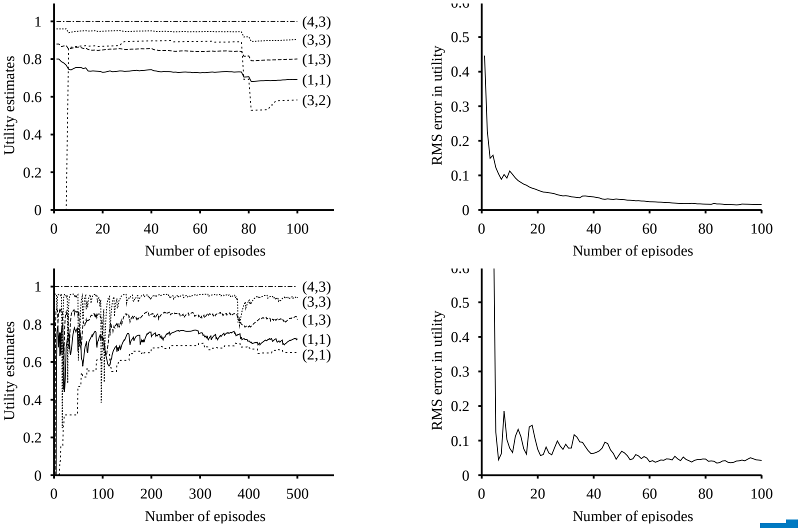
$\widehat{U}_{k+1}[s_k] = r_k$ iff $\widehat{U}_k[s_k] = \mathrm{none}$ Utility set to reward for a newly visited state.

and iff $s_k^{\mathrm{old}} \neq \mathrm{none}$:

$$\widehat{U}_{k+1}[s_k^{\mathrm{old}}] = \widehat{U}_k[s_k^{\mathrm{old}}] + \alpha(N_k[s_k]) \left(r_k + \gamma \widehat{U}_k[s_k] - \widehat{U}_k[s_k^{\mathrm{old}}] \right)$$

Π and α do not change. N and $\widehat{U}$ retain values for all non-indicated arguments.

ADP (top) vs TD (bottom)



Passive Agents – Discussion

- Direct utility estimation
    - simple to implement, model-free,
    - each update is fast,
    - does not exploit state dependence and thus converges slowly,
- Adaptive dynamic programming
    - harder to implement, model-based,
    - each update is a full policy evaluation (expensive),
    - fully exploits state dependence, fastest convergence in terms of episodes,
- Temporal difference learning
    - similar to DUE: model-free, update speed and implementation
    - partially exploits state dependence but does not adjust to *all* possible successors,
    - convergence in between DUE and ADP.

Active ADP Agent

Change the passive ADP agent into an *active* one following the optimal policy principle:

$$a_k = \pi^*(t_k, s_k) = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U(s)$$

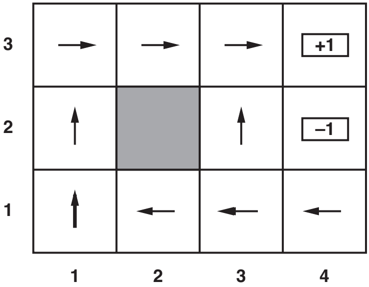
With models $N_k, \widehat{U}_k$ of P_S, U stored in $t_k = \left\langle \Pi, s'_k, N_k, \widehat{r}_k, \widehat{U}_k \right\rangle$, this gives:

$$a_k = \pi(t_k, s_k) = \arg\max_{a \in A} \sum_{s \in S} \frac{N_k[s, s_k, a]}{\sum_{s' \in S} N_k[s', s_k, a]} \widehat{U}_k[s]$$

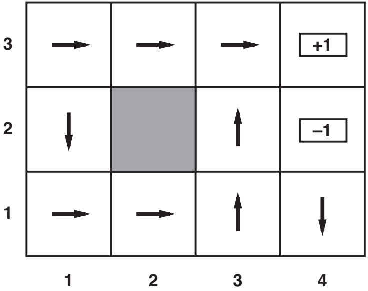
Note that $N_k, \widehat{U}_k$ evolve by following the agent's policy π, which in turn depends on them.

This agent is *greedy*: never chooses a sub-optimal (w.r.t. $\widehat{U}_k$) state just to explore it. If π were optimal that would be OK but ...

Greedy ADP Agent: Properties



Optimal policy



Policy learned by Greedy ADP

- Agent did not learn an optimal policy because it followed an inaccurate model of P_S, U.
- Agent did not learn an accurate model of P_S, U because it did not follow an optimal policy.

N-Armed Bandit

Converging to an optimal strategy requires *exploration*, i.e. actions suboptimal w.r.t. the current utility model.

Easily demonstrated in a setting even simpler than reinforcement learning.

The *n-armed bandit* problem.:

- set of actions A and rewards R
- Agent repeatedly picks $a \in A$ and gets $r \in R$ according to $P_{r|a}(r|a)$
- No states, just a series of independent trials
- Agent's goal: without knowing $P_{r|a}$, maximize mean of received rewards.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

N-Armed Bandit: Greedy vs. Explorative

Optimal strategy: $a = \arg\max_{a \in A} \mathbb{E}\left(r|a\right) = \arg\max_{a \in A} \sum_{r \in R} P_{r|a}(r|a)r$

Without knowing $P_{r|a}$, the agent first tries each action $a \in A$ exactly once, storing the received rewards $\widehat{r}[a] = r$ and then iterate one of:

Greedy approach: $a = \arg\max_{a \in A} \widehat{r}[a]$ (would be optimal if $\widehat{r}[a] = \mathbb{E}\left(r|a\right)$)

Explorative approach: with some $0 < \epsilon < 1$:

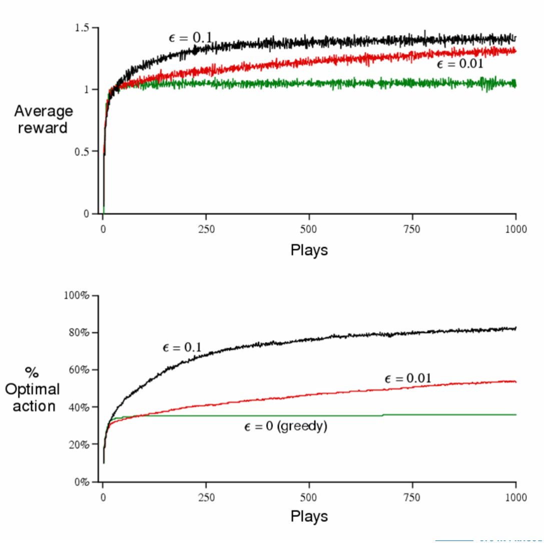
$$a = \begin{cases} \arg\max_{a \in A} \widehat{r}[a] \text{ with probability } 1 - \epsilon \\ \text{random action with probability } \epsilon \end{cases} \tag{1}$$

updating $\widehat{r}[a]$ to the mean of all rewards seen for a.

Exploration vs. Exploitation

- Greedy (green) performs poorly: not enough *exploration* to estimate $\widehat{r}$.
- But once $\widehat{r}$ is accurate, it is better to *exploit* it by the greedy strategy.
- When to switch from exploration to exploitation? An essential AI dilemma.
- Instead of switching, decaying $\epsilon \to 0$ also possible.

Example with $P_{r|a}$ Gaussian



Greedy in the limit of infinite exploration (GLIE)

A GLIE strategy makes sure that with $k \to \infty$, in each state, each action is tried an infinite number of times. This way, no good action is missed.

In reinforcement learning, take a *random* action with a decaying probability $\epsilon > 0, \epsilon \xrightarrow{k\to\infty} 0$. For example, instead of taking action

$$\arg\max_a Q(s,a) \text{ where } Q(s,a) = \sum_{s'} P_S(s'|s,a)U^{\pi}(s')$$

choose a *random* action a with *softmax* probability.

$$\frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}$$

where the *temperature* $\tau \to 0$ with $k \to \infty$. GLIE strategies tend to converge slow.

Exploration Function

Faster convergence is achieved if unexplored nodes are deliberately promoted, e.g. by tweaking the utility function

$$U^{\pi}(s_k) = r(s_k) + \gamma \sum_{s \in S} P_S(s|s_k, a_k) U^{\pi}(s_k)$$

where $a_k = \pi(s_k) = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U^{\pi}(s)$, into

$$U_e^{\pi}(s_k) = r(s_k) + \gamma \max_a f \left(\sum_{s \in S} P_S(s|s_k, a_k) U_e^{\pi}(s_k), N_k(s_k, a_k) \right)$$

where $a_k = \pi(s_k) = \arg\max_{a \in A} \sum_{s \in S} P_S(s|s_k, a) U_e^{\pi}(s)$ and the *exploration function* f trades off between

- the estimated expected utility of the next state
- $N_k(s, a)$: the number of times action a was taken in state s until time k.

Optimistic Utilities

f should not

- decrease with $\sum_{s \in S} P_S(s|s_k, a_k) U_e(s_k)$
- increase with $N_k(s, a)$

A simple option is to assign an *optimistic utility* value ($\max R$ - highest reward value) to states explored less then m times:

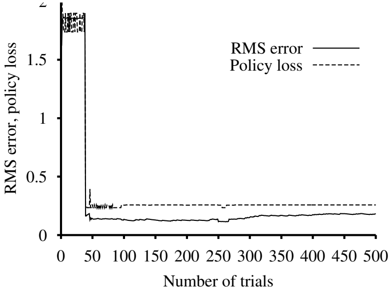
$$f = \begin{cases} \max R \text{ if } N_k(s,a) < m \\ \sum_{s \in S} P_S(s|s_k, a_k) U_e(s_k) \text{ otherwise} \end{cases}$$

If $U_e(s) \geq U(s)$ is preserved during updates for $\forall s$ then utilities converge to $U(s)$.

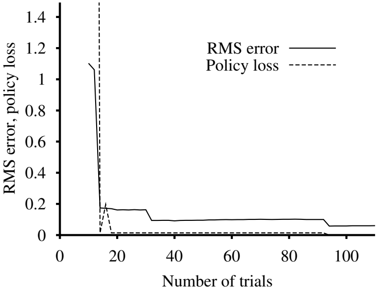
Exploration vs. Exploitation dilemma: it is not feasible to optimize m theoretically. Use ‘rule of the thumb’.

Greedy vs. Exploratory ADP Agent

Exploratory ADP Agent: like greedy ADP but with U replaced by U_e.



Greedy ADP agent



Exploratory ADP agent with $\max R = 2,\ t = 5.$

Unlike greedy, it converges to the optimal policy (loss → 0) and comes close to the true state utilities (small root mean squared error).

Active TD Agent

Recall: unlike ADP, the passive TD agent does not need a model of P_S to estimate U. However, its *active* version would still need such a model to approximate the policy

$$\pi(s) = \arg\max_a \sum_{s'} P_S(s'|s,a)U^\pi(s')$$

This can be prevented: instead of learning state utilities $U^\pi(s)$, learn *state-action utilities $Q^\pi(s,a)$*.

$Q^\pi(s,a)$ is the utility of taking action a in state s under policy π, so

$$U^\pi(s) = \max_a Q^\pi(s,a)$$

(Q means Q^π for an optimal policy π.)

Explorative Q-Learning Agent

Q^π can be computed as the solution to the set of Bellman-like equations

$$Q^\pi(s,a) = r(s) + \gamma \sum_{s' \in S} P_S(s'|s,a) \max_{a' \in A} Q^\pi(s',a')$$

$\forall s \in S, a \in A$. This is achieved *without* P_S by iterating similar to TD:

$$Q^\pi(s_k,a_k) \leftarrow Q^\pi(s_k,a_k) + \alpha \left(r_k + \gamma \max_a Q^\pi(s_{k+1},a) - Q^\pi(s_k,a_k) \right)$$

The exploration incentive is put in the policy:

$$a_k = \begin{cases} \text{none if } s_k \text{ is terminal} \\ \arg\max_a f\left(Q^\pi(s_k,a), N(s_k,a)\right) \text{ otherwise} \end{cases}$$

$N(s,a)$ counts the times a was taken in state s.

Q-Learning Agent: Design

Agent's state is a tuple

$$t_k = \left\langle s_k^{\mathrm{old}}, a_k^{\mathrm{old}}, r_k^{\mathrm{old}}, N_k, \widehat{Q}_k, \alpha \right\rangle$$

- $s_k^{\mathrm{old}}, a_k^{\mathrm{old}}$: last state and action, $a_1^{\mathrm{old}} = s_1^{\mathrm{old}} = \mathrm{none}$
- r_k^{old}: last reward, $r_1^{\mathrm{old}} = 0$
- N_k: state-action pair frequency array addressed by $[s, a]$, N_1 filled with zeros.
- $\widehat{Q}_k$: Q^π estimate array addressed by $[s, a]$. $\widehat{Q}_1$ filled with zeros.
- $\alpha : \mathbb{N} \to \mathbb{R}$: a positive, monotone decreasing function

Note: this agent must store the last action as policy is not fixed: in general $a_{k-1} = \pi(t_{k-1}, s_{k-1}) \neq \pi(t_k, s_{k-1})$.

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Q-Learning Agent: Design (cont'd)

Agent state update:

$$N_{k+1}[s_k, a_k] = N_k[s_k, a_k] + 1$$

Counter incremented for current state/action, rest of array unchanged.

$$s^{\text{old}}_{k+1} = s_k,\ a^{\text{old}}_{k+1} = a_k$$
$$r^{\text{old}}_{k+1} = r_k$$

Current observation, action, and reward are stored for use at next update.

$$\widehat{Q}_{k+1}[s^{\text{old}}_k, a^{\text{old}}_k] = r_k$$
if $a^{\text{old}}_k = \text{none}$

Value of a terminal state (detected by none action) is its reward. For non-terminal states, iterate as below. Rest of $\widehat{Q}$ array unchanged.

$$\widehat{Q}_{k+1}[s^{\text{old}}_k, a^{\text{old}}_k] =$$
$$\widehat{Q}_k[s^{\text{old}}_k, a^{\text{old}}_k] + \alpha(N_k[s^{\text{old}}_k, a^{\text{old}}_k]) \left(r^{\text{old}}_k + \gamma \max_a \widehat{Q}_k[s_k, a] - \widehat{Q}_k[s^{\text{old}}_k, a^{\text{old}}_k] \right)$$
if $s^{\text{old}}_k \neq \text{none} \neq a^{\text{old}}_k$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Greedy Q-Learning Agent

Consider a greedy (non-exploratory) variant of the Q-Learning agent, deciding by

$$a_{k+1} = \arg\max_{a} Q^{\pi}(s_{k+1}, a)$$

Here, the iteration

$$Q^{\pi}(s_k, a_k) \leftarrow Q^{\pi}(s_k, a_k) + \alpha \left(r_k + \gamma \max_{a} Q^{\pi}(s_{k+1}, a) - Q^{\pi}(s_k, a_k) \right)$$

can get rid of the maximization:

$$Q^{\pi}(s_k, a_k) \leftarrow Q^{\pi}(s_k, a_k) + \alpha \left(r_k + \gamma Q^{\pi}(s_{k+1}, a_{k+1}) - Q^{\pi}(s_k, a_k) \right)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

SARSA Agent

SARSA agent is the exporatory Q-Learning agent where even for a non-greedy strategy the iteration is changed to

$$Q^\pi(s_k, a_k) \leftarrow Q^\pi(s_k, a_k) + \alpha \left(r_k + \gamma Q^\pi(s_{k+1}, a_{k+1}) - Q^\pi(s_k, a_k) \right)$$

Name due to *S*tate-*A*ction-*R*eward-*S*tate-*A*ction quintuplet

$$s_k, a_k, r_k, s_{k+1}, a_{k+1}$$

from which Q^π iterated.

Q-Learning is an *off-policy* (as in, less dependent on policy) strategy. Tends to learn Q better event if π is far from optimal.

SARSA is an *on-policy* strategy. Tends to adapt better to partially enforced policies.

Problems with Table Models

So far, $\widehat{U}, \widehat{Q}$ have been look-up tables (arrays) demanding at least $\mathcal{O}(|S|)$ resp. $\mathcal{O}(|S| \cdot |A|)$ memory and time.

Table-based agents would not scale to large ('real-life') state spaces S.

- Backgammon or Chess: $|S|$ somewhere btw. 10^{20} and 10^{45}
- No way to capture in an array, let alone do policy evaluation

A more compact ('generalized') model for

$$U : S \rightarrow \mathbb{R} \text{ or } Q : S \times A \rightarrow \mathbb{R}$$

is needed. Must allow learning (updating) from $[s_k, a_k, r_k, s_{k+1}]$ or $[s_k, a_k, r_k, s_{k+1}, a_{k+1}]$ samples.

Feature-Based Representation of $\widehat{U}$

Consider learning $\widehat{U}$ with the *Direct Utility Estimation* agent.

A simple option is to define a set of relevant features $\phi^i : S \to \mathbb{R}$ and use a *regression model*.

$$\widehat{U}(\mathbf{w}, s) = \sum_{i=1}^{n} w^i \phi^i(s)$$

and adapt the parameters $\mathbf{w} = [w^1, w^2, \dots, w^n]$ at each episode's end to reduce the squared error

$$E_j(\mathbf{w}, s) = \frac{1}{2} \left(\widehat{U}(\mathbf{w}, s) - u_j(s) \right)^2$$

where $u_j(s)$ is the utility sample obtained for s at the end of episode $j = 1, 2, \dots$ (when a terminal state is reached).

Feature-Based Representation of $\widehat{U}$ (cont'd)

Going against the error gradient with learning rate $\alpha \in \mathbb{R}$:

$$w^i \leftarrow w^i - \alpha \frac{\partial E_j(\mathbf{w}, s)}{\partial w^i} = w^i + \alpha \left(u_j(s) - \widehat{U}\left(\mathbf{w}, s\right)\right) \frac{\partial \widehat{U}(\mathbf{w}, s)}{\partial w^i}$$

Example: Let $[\phi^1(s), \phi^2(s)] = [s^1, s^2]$, i.e., the agent's coordinates in the grid environment and $\phi^3 \equiv 1$.

Note: superscript component indexes to disambiguate from subscripted time indexes

Then

$$\widehat{U}(\mathbf{w}, s) = w^1 s^1 + w^2 s^2 + w^3$$

and the iterative update:

$$\begin{aligned} w^1 \leftarrow & w^1 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s))s^1, \\ w^2 \leftarrow & w^2 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s))s^2 \\ w^3 \leftarrow & w^3 + \alpha(u_j(s) - \widehat{U}(\mathbf{w}, s)) \end{aligned}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Feature-Based Representation of $\widehat{U}$: Notes

1. Observe:

$$\frac{\partial \widehat{U}(\mathbf{w}, s)}{\partial w^i} = \phi^i(s)$$

So the derivative is simple even with non-linear features such as

$$\phi^i(s) = \sqrt{(s^1 - 4)^2 + (s^2 - 3)^2}$$

measuring the Euclidean ('air') distance to the terminal state $(4, 3)$.

2. Features allow to deal with a kind of *partial state observability*. If a component of the state is not observable, design features that do not use that component.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Feature-Based Representation of $\widehat{Q}$

A similar strategy can be applied in the TD agent or the Q-Learning agent. For the latter

$$\widehat{Q}(\mathbf{w}, s, a) = \sum_{i=1}^{n} w^i \phi^i(s, a)$$

where ϕ^i are predefined features of state-action pairs.

Follow the gradient descent (again, $\frac{\partial \widehat{Q}(\mathbf{w},s,a)}{\partial w^i} = \phi^i(s, a)$) at each time k

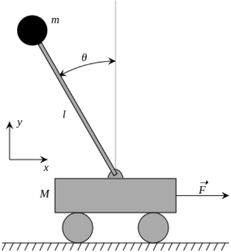
$$w^i_{k+1} = w^i_k + \alpha \left(r(s_k) + \gamma \max_a \widehat{Q}(\mathbf{w}_k, s_{k+1}, a) - \widehat{Q}(\mathbf{w}_k, s_k, a_k) \right) \phi^i(s_k, a_k)$$

The principle is simple, the art is in designing good features ϕ^i.

Inverted Pendulum Demo

Real-valued features especially appropriate where environment is a dynamic physical system. Typical features are *positions* and *accelerations* of objects.

Example: inverted pendulum



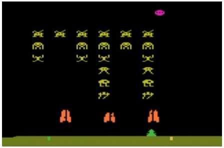
Videos: Single (Experience Replay - see later), Triple (!).

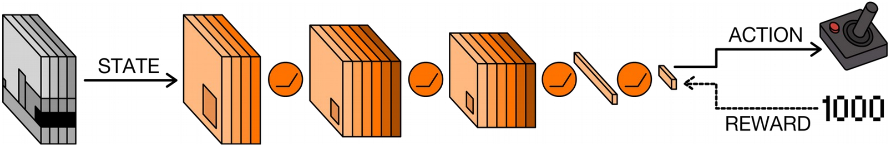
Deep Q-Learning: DQN

Learns to play ATARI 2600 games from screen images and score.
(DeepMind / Nature, 2015)

Deep feed-forward network approximating $Q(s, a)$

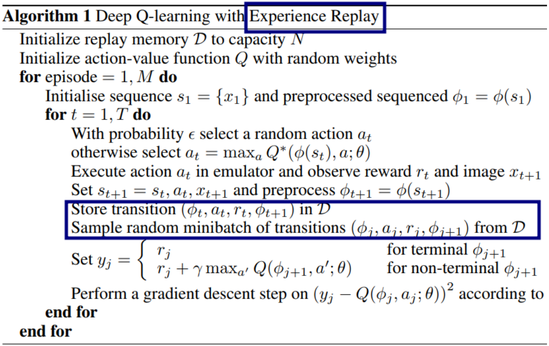
- input = state = 4 time-subsequent 84x84 gray-scale screens
- separate output for each $a \in A$
- 2 convolution + 1 connected hidden layers



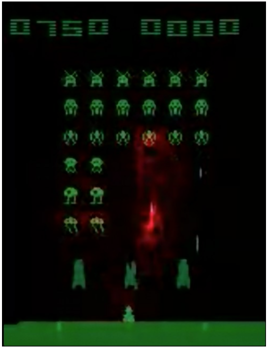


Demo

Deep Q-Learning: DQN (cont'd)



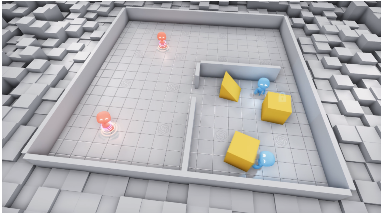
```
Algorithm 1 Deep Q-learning with Experience Replay
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to
    end for
end for
```



Experience replay prevents long chains of correlated training examples by sampling from a buffer of $\vec{\phi}(s_k), a_k, r_k, \vec{\phi}(s_{k+1})$ tuples recorded in the past.

Backpropagation to the original image inputs reveals areas of 'attention'.

OpenAI: Hide and Seek Game



go to web

- Two hiders, two seekers, each learning by reinforcement
- Can move, shift and lock blocks, see others and blocks (if in line of sight), sense distance
- Team-wide rewards to hiders: -1 if any hider seen by a seeker, +1 if all hiders hidden
- Seekers get opposite rewards.

Policy Search

Instead of searching $\widehat{Q}$ (or $\widehat{U}$ or P_S and r) search directly a good policy $\pi : S \rightarrow A$.

If S unmanageably large, use features again: $\phi^i : S \rightarrow Z_i$ where $i = 1, 2, \ldots n$ and Z_i are the feature value ranges.

Then set $\pi(S) = \pi'(\phi_1(S), \ldots \phi_n(S))$ where $\pi' : Z_1 \times \ldots \times Z_n \rightarrow A$

Quality of π' is estimated as mean total rewards over repeated episodes using π. Search may e.g. be greedy adjustments to π' improving its quality.

Since A is finite (discrete), π' is not differentiable so gradient descent not applicable. If Z_i are finite (e.g. discretized), this means combinatorial search.

Differentiable Policy Search

Gradient-based policy search is possible with a *stochastic policy* choosing action a in state s with (softmax) probability

$$\frac{e^{q(\mathbf{w},\boldsymbol{\phi}(s,a))}}{\sum_{a' \in A} e^{q(\mathbf{w},\boldsymbol{\phi}(s,a'))}}$$

where $\mathbf{w} \in \mathbb{R}^n$ are real parameters, $\boldsymbol{\phi} = \langle \phi_1, \ldots, \phi_{n'} \rangle$ are some real-valued features, and $q : \mathbb{R}^{n+n'} \rightarrow \mathbb{R}$. If q differentiable in all ϕ^i as in e.g. $(n = n')$

$$q(\mathbf{w}, \boldsymbol{\phi}(s,a)) = \sum_{i=1}^{n} w^i \phi^i(s,a)$$

then $\mathbf{w}$ can be adapted through (empirical) gradient descent (but gradient estimation not trivial with stochastic environment and policy).

Reminds of $\widehat{Q}$ learning but q optimized w.r.t. policy performance.

Learning a Feature-Based Environment Model

An agent deriving policy from a model of P_S and r can learn such models. In the ADP agent, such models were just relative frequencies.

They can also be feature based. So $P_S(s'|s,a)$ can be modeled e.g. by

$$f(\mathbf{w}, \boldsymbol{\phi}(s', s, a)) = \sum_{i=1}^{n} w^i \phi^i(s', s, a)$$

where ϕ^i are features of the s', s, a triple and w^i real parameters. Gradient method applicable, every transition provides a sample. No need to normalize f (probability!) if only used in $\arg\max$ expressions.

Similarly for the reward model (modeling a function, not a prob.)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Bayesian Learning of an Environment Model

Consider the following *Bayesian* approach which involves

- a countable probability *distribution class* $\mathcal{M}$ ("model class")
- at each time k, a probability distribution B_k on $\mathcal{M}$ where $B_k(P)$ ($P \in \mathcal{M}$) quantifies the belief that $P_S \equiv P$. B_1 is the initial belief.

At each time k, our model $\xi_k(s_{k+1}|s_k, a)$ of $P_S(s_{k+1}|s_k, a_k)$ is

$$\xi_k(s_{k+1}|s_k, a_k) = \sum_{P \in \mathcal{M}} P(s_{k+1}|s_k, a_k) B_k(P)$$

i.e, a probability-weighted sum where each model contributes the stronger the higher its belief. $|\mathcal{M}|$ may be ∞ but the sum obviously converges.

Bayesian Learning of an Environment Model (cont'd)

At each time $k+1$, B_k is updated by the Bayes rule to the posterior

$$B_{k+1}(P) = \alpha P(s_{k+1}|s_k, a_k) B_k(P)$$

for each $P \in \mathcal{M}$, where the normalizer α is such that

$$\sum_{P \in \mathcal{M}} B_{k+1}(P) = 1$$

Note that the s_{k+1} states are sampled mutually independently *given* s_k, a_k from the same distribution $P_S(s_{k+1}|s_k, a_k)$ although s_{k+1} are not independent of s_k or a.

(This can also be posed as learning a separate model $P_{s_k,a_k}(s_{k+1})$ for each possible $s_k \in S, a_k \in A$.)

Bayesian Learning of an Environment Model (notes)

- ξ is a *convex linear combination* of environments (transition distributions). So with simpler notation ($\mathbf{w}_k \in \mathbb{R}^{|\mathcal{M}|}$):

$$\xi_k(s_{k+1}|s_k, a_k) = \sum_{P_i \in \mathcal{M}} w_k^i P_i(s_{k+1}|s_k, a_k) \tag{2}$$

$$w_{k+1}^i = \alpha w_k^i P_i(s_{k+1}|s_k, a_k) \tag{3}$$

- For an uncountable model class $\mathcal{M}_{\mathbf{w}}$ parameterized by $\mathbf{w} \in \mathbb{R}^n$ (different $\mathbf{w}$ from above!), we would have

$$\xi_k(s_{k+1}|s_k, a_k) = \int_{\mathbf{w}} P(s_{k+1}|s_k, a_k, \mathbf{w}) B(\mathbf{w}) \tag{4}$$

$$B_{k+1}(\mathbf{w}) = \alpha P(s_{k+1}|s_k, a_k, \mathbf{w}) B_k(\mathbf{w}) \tag{5}$$

Bayesian Learning of an Environment Model (notes cont'd)

- The Bayesian approach reminds of Belief-updates in POMDP which you (should) know:

  $$B_{k+1}(s_{k+1}) = \alpha P_{o|s}(o_{k+1}|s_{k+1}) P(s_{k+1}|a_k)$$

  where

  $$P(s_{k+1}|a_k) = \sum_{s_k \in S} P_S(s_{k+1}|s_k, a_k) B_k(s_k)$$

  However, POMDP model unknown states with known transition distributions, whereas we model unknown transition distributions with observed states.

- Both unknown states and unknown transition distributions can be modeled simultaneously in the Bayesian approach, giving rise to 'partially observable reinforcement learning'. Of course, very complex computationally.

Bayesian Learning of an Environment Model (notes cont'd)

An agent implementing the Bayesian updates of ξ and following the optimal policy w.r.t. ξ:

$$\pi^*(s_k) = \arg\max_{a \in A} \sum_{s \in S} \xi(s|s_k, a) U(s)$$

where

$$U(s_k) = r(s_k) + \gamma \sum_{s \in S} \xi(s|s_k, \pi^*(s_k)) U(s)$$

maximizes the expected total reward w.r.t. ξ, where $\xi \rightarrow_{k \rightarrow \infty} P_S$ if $\exists i$ s.t. $P_S \equiv P_i \in \mathcal{M}$ and $w^i > 0$, and has

- no parameters except $\mathcal{M}$ and $\mathbf{w}$
- no exploration/exploitation dilemma

Universal Learning

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Non-Markovian Environments

Recall the non-Markov setting – the most general considered in this course:

$$P(xa_{\leq m}) = P(x_1)P(a_1|x_1)P(x_2|x_1, a_1) \ldots P(x_m|xa_{<m})P(a_m|x_m, xa_{<m})$$

Percept $x_k = (o_k, r_k)$ depends probabilistically on the entire history $xa_{<k}$. There is no state observability as there are no states.

Acting (maximizing rewards) clearly not possible without estimating future percepts. This subsumes the general problem of *sequence prediction* which is formulated without actions and rewards simply as:

Universal Sequence Prediction Problem

Given:

$$o_1, o_2, \ldots o_k$$

Predict o_{k+1}.

Sequence Prediction

Some sequences seem obvious to extend. E.g.

$$1, 2, 3, 4, 5$$

because of the pattern $o_{k+1} = o_k + 1$.

But e.g.

$$1, 2, 3, 4, 29$$

could also be argued due to $o_k = k^4 - 10k^3 + 35k^2 - 49k + 24$.

The first pattern seems more plausible because it is *simpler*. Note that this reason is not statistical/probabilistic.

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

Sequence Prediction (cont'd)

Other sequences have no obvious equational pattern

$$3, 1, 4, 1, 5, 9$$

but there is still a simple extension rule: here o_k is the k's digit in the decimal expansion of the number π. So the extension 9 seems plausible here.

We need to formalize these thoughts to answer questions such as

- What exactly is meant by *pattern*?
- How to measure *complexity* of patterns and sequences?
- Are simple patterns more likely to make correct predictions?
- Are there sequences that have no patterns?

Computing a Sequence

The most general intepretation of sequence ‘pattern’ is a *program* that generates the sequence.

For simplicity, let $O = \{ 0, 1 \}$ so O^* denotes the set of all binary strings. Let $T : O^* \to O^*$ be a *partial recursive function*, i.e., there is a Turing machine computing $T(p)$ but for some $p \in O^*$ it need not halt.

The input p to the T.M. may be interpreted as a program computing $T(p)$.

Intuitively, simple strings (even infinite) are those computed by short programs p. So e.g. the decimal expansion of π, however long, is simple because there is a short program computing it.

Denoting the length of p by $|p|$, this gives rise to the Kolmogorov complexity of strings.

Kolmogorov Complexity

The *Kolmogorov complexity* $K_T(q)$ of $q \in O^*$ with respect to T is

$$K_T(q) = \min \{ \, |p| \, ; p \in \{ 0, 1 \}^* \, , T(p) = q \, \}$$

So the complexity of a (possibly infinite) string is the length of the shortest program that generates it, i.e., the shortest binary input to T that makes it produce the string.

Dependence of $K_T(q)$ on T is not a serious problem as there is a *universal* T.M. U which simulates any T.M. T given the (finite!) sequential description $\langle T \rangle \in O^*$ of T as a distinguished part of its input, i.e.

$$U(\langle T \rangle : p) = T(p)$$

The colon is a distinguished symbol delimiting $\langle T \rangle$ and p on U's (input) tape.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity (cont'd)

Consequence: given a T, for *every* $q \in O^*$:

$$K_U(q) \leq K_T(q) + \mathcal{O}(|\langle T \rangle|)$$

where the rightmost term ('translation overhead') does not depend on q and becomes negligible for large q. So we adopt K_U as the universal complexity measure and denote $K(q) = K_U(q)$.

Clearly, for every $q \in O^*$:

$$K(q) \leq |q| + c$$

since the program for computing q can simply contain the $|q|$ symbols of q plus some constant c number of symbols implementing the loop to print them on the output.

Kolmogorov Complexity - Examples

- $q = \underbrace{0, 0, \ldots, 0}_{n \text{ times}}$ has $K(q) = \log n + c$. Need $\log(n)$ symbols to encode the integer n plus a constant-size code to print it.
- $q =$ the first n digits in the binary expansion of π also has $K(q) = \log n + c$: $\log n$ symbols to specify n plus a constant-size code for calculating (and printing) the digits of π.
- Are there any strings q such that $K(q) \geq |q|$?
  Yes, such strings exist for any length k, as there are only $2^k - 1$ programs (binary strings) shorter than k (you do the math), so there must be some string of length k for which there is no shorter program generating it. Such a string is called *incompressible* or *random* (not in the probabilistic sense!).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Kolmogorov Complexity - Computability

The question whether $K(q) \geq n$ ($q \in O^*$, $n \in \mathbb{N}$) is *undecidable*, i.e., *K is not finitely computable*.

Proof: Assume a deciding program p exists. Consider the first (in the lexicographic order) string $q \in O^*$ such that $K(q) \geq n$. Then q can be generated as follows: for each $q \in O^*$ in the lexicographic order, determine if $K(q) \geq n$ using p and print the first such q. This procedure can be encoded in a program using $|p| + \log(n) + c$ symbols, which (for a sufficiently large n) is smaller than n, so $K(q) < n$. Contradiction → p does not exist.

Proof idea intuitively: consider the *shortest string than cannot be specified with fewer than twelve words*. This string has just been specified with eleven words. This paradox implies that the property *can be specified with n words* is not decidable.

Kolmogorov Complexity - Enumerability

Recall definitions from computability courses: A function $f(x) : \mathbb{N} \to \mathbb{Q}$ is *enumerable* if there is a Turing machine finitely computing a function $f(x, k)$ such that $\lim_{k \to \infty} f(x, k) = f(x)$ and $f(x, k) \leq f(x, k + 1)$ for $\forall k$. A function $f(x)$ is *co-enumerable* if $-f(x)$ is enumerable.

K is co-enumerable.

Proof: *omitted for now, the proof shown in the previously published version was incorrect. I will try to come up with a simple yet correct proof.*

Universal Prior M

To predict o_k from $o_{<k}$ without knowing $P(o_k|o_{<k}) = P(o_{\leq k})/P(o_{<k})$, we can surrogate P with a probability distribution M on O^* giving greater probability to sequences simpler in the Kolmogorov sense.

A natural choice would be $M(q) = 2^{-K(q)}$ but since that would not satisfy probability axioms, Solomonoff (1964) instead proposed the *universal prior*

$$M(q) = \sum_{p:U(p)=q*} 2^{-|p|}$$

where the sum is over all programs for which the universal T.M. U outputs a string starting with q, not necessarily halting.

So all programs p generating q contribute to q's probability but short programs contribute exponentially more than long programs.

Universal Prior M: Properties

$M(q)$ is close to $2^{-K(q)}$ since the shortest program generating q contributes exponentially more to $M(q)$ than other programs.

M is enumerable (proof omitted).

M is not a *normalized* probability distribution on $\{0,1\}^*$. Indeed

$$M(q0) + M(q1) \leq M(q)$$

where $q0$ ($q1$) means the extension of string q with 0 (1), since some programs computing q may afterwards halt or loop forever without writing 0 or 1. As opposed to distribution measures, M is a *semi-measure*.

Normalization is not needed when M is used for sequence prediction, as we will see. Normalizing M into a measure is possible at the price of losing enumerability.

Universal Prior M: Properties (cont'd)

Because $(1-a)^2 \leq -\frac{1}{2}\ln a$ (for $0 \leq a \leq 1$):

$$\sum_{k=1}^{\infty}(1 - M(o_k|o_{<k}))^2 \leq -\frac{1}{2}\sum_{k=1}^{\infty}\ln M(o_k|o_{<k}) =$$

Swap the sum with the logarithm and use the chain-rule:

$$= -\frac{1}{2}\ln M(o_1)\cdot M(o_2|o_1)\cdot M(o_3|o_{<3})\cdot\ldots = -\frac{1}{2}\ln M(o_{1:\infty}) =$$

Plug in the definition of $M(q)$, then drop from the sum all p's computing $o_{1:\infty}$ except for the shortest one denoted $p_{\min}$

$$= -\frac{1}{2}\ln \sum_{p:U(p)=o_{1:\infty}} 2^{-|p|} \leq -\frac{1}{2}\ln 2^{-|p_{\min}|} \leq \frac{1}{2}\ln 2\cdot|p_{\min}|$$

If $o_{1:\infty}$ is computable then clearly $|p_{\min}| < \infty$ and so

$$\sum_{k=1}^{\infty}(1 - M(o_k|o_{<k}))^2 < \infty$$

Using M for Sequence Prediction

$\sum_{k=1}^{\infty} (1 - M(o_k|o_{<k}))^2 < \infty$ implies

$$\lim_{k\to\infty} M(o_k|o_{<k}) = 1$$

(otherwise the sum would diverge).

M is a universal sequence predictor.

This means that after “seeing” the beginning $o_{<k}$ of the sequence, M predicts the next element with probability approaching 1 with $k \to \infty$. So M “recognizes” the environment on the only condition that the latter produces a computable sequence, i.e., it is a Turing machine.

The condition above is **not** strong: all physical theories of the world are computable, so any “reasonable” environment is a T.M. But remind the catch: M itself is not computable, only enumerable.

M as a Bayesian Mixture

Levin (1970) showed that M is equivalent to the Bayesian mixture

$$\xi_U(q) = \sum_{P \in \mathcal{M}_U} 2^{-K(P)} P(q)$$

where $\mathcal{M}_U$ is the set of *all enumerable semi-measures* (containing also enumerable proper measures) and $K(P)$ is the size of the shortest program computing the function P. $\mathcal{M}_U$ is the largest known class of probability distributions resulting in an enumerable mixture.

The equivalence is in the sense that

$$M(q) = \mathcal{O}(\xi_U(q)) \text{ and } \xi_U(q) = \mathcal{O}(M(q))$$

So $\xi_U(q)$ has the same properties as M but is more convenient for approximations (e.g. using only some subset of $\mathcal{M}_U$).

Policies and Utilities in the Non-Markov Case

For bare sequence prediction, we disregarded actions and rewards. Let us now get them back in the game. Recall the non-Markov setting:

- Agent's policy: $\pi : (X \times A)^* \times X \to A$, $a_k = \pi(xa_{<k}, x_k)$
- Policy value (for simplicity, we use the finite horizon m version):

$$V^\pi = \mathbb{E}\left(\sum_{k=1}^{m} r_k\right) = \sum_{x_{\leq m}} P(x_{\leq m}|a_{<m})(r_1 + r_2 + \ldots + r_m) \tag{6}$$

The maximum value policy $\pi^* = \arg\max_\pi V^\pi$ prescribes actions

$$a_k = \arg\max_{a_k} \sum_{x_{k+1}} \ldots \max_{a_{m-1}} \sum_{x_m} (r_{k+1} + \ldots + r_m) P(x_{k+1:m}|x_{\leq k}, a_{<m})$$

maximizing the total 'reward to go' ($x_{k+1:m}$ means $x_{k+1}, x_{k+2}, \ldots x_m$).

Policies and Utilities in the Non-Markov Case (Notes)

Note that in

$$a_k = \arg\max_{a_k} \sum_{x_{k+1}} \ldots \max_{a_{m-1}} \sum_{x_m} (r_{k+1} + \ldots + r_m) P(x_{k+1:m} | x_{\leq k}, a_{<m})$$

the sums implement the expectation w.r.t. the probability of percepts remaining after current time k conditioned on all percepts until k and all actions (past, current, future). Using the chain rule and removing dependencies of x_j on all x_i, a_i such that $i \geq j$ (percepts depend only on *past* percepts and actions), we get

$$P(x_{k+1:m} | x_{\leq k}, a_{<m}) = \prod_{j=k+1}^{m} P(x_j | xa_{<j})$$

When modeling the environment, we can thus either seek a model of $P(x_{k+1:m} | x_{\leq k}, a_{<m})$ or of $P(x_j | xa_{<j})$.

The AIξ Agent

If we know $P(x_j|xa_{<j}) \in \mathcal{M}$ for some distribution class $\mathcal{M}$, Bayesian inference can be applied just like we did in the Markovian case, by replacing it with a mixture distribution, which at time k is:

$$\xi_k(x_k|xa_{<k}) = \sum_{P_i \in \mathcal{M}} w_k^i P_i(x_k|xa_{<k})$$

The initial weights $\mathbf{w}_1 = \left\langle w_1^1, w_1^2, \dots w_1^{|\mathcal{M}|} \right\rangle$ $(\sum_{i=1}^{|\mathcal{M}|} w_1^i = 1, \forall i : w_1^i > 0)$ encode the prior belief in model correctness. If $|\mathcal{M}| < \infty$, they may be uniform. At $k+1$, each w_k^i is updated to

$$w_{k+1}^i = \alpha P_i(x_k|xa_{<k}) w_k^i$$

where $\alpha = 1/\sum_{i=1}^{|\mathcal{M}|} w_{k+1}^i$ is a normalizer.

The AIXI Agent

The *AIXI agent* proposed by Hutter (2005) is the most universal AI agent adopting the largest enumerable model class $\mathcal{M}$, i.e. the class $\mathcal{M}_U$ of all enumerable semimeasures, and using their complexity-weighted mixture ξ_U we have already seen.

We know that ξ_U is equivalent to the universal prior M, allowing simpler notation. We substitute $P(x_{k+1:m}|x_{\leq k}, a_{<m})$ with M in the conditional form

$$M(x_{k+1:m}|x_{\leq k}, a_{<m}) = \sum_{U(p:x_{\leq k}:a_{<m})=x_{k+1:m}*} 2^{-|p|}$$

where the sum is over all programs for the universal T.M. which output $x_{k+1:m}$ (followed by any suffix) given the input $x_{\leq k}, a_{<m}$.

The colon under the sum delimits p and its inputs on U's (input) tape.

The AIXI Agent (cont'd)

Note that there are no updates to the weights of (beliefs in) models done at each time k since $M(x_{k+1:m}|x_{\leq k}, a_{<m})$ accounts for the entire history $x_{\leq k}, a_{<m}$.

In summary, with a finite time horizon m the AIXI agent has policy

$$a_k = \pi(xa_{<k}, x_k) =$$
$$= \arg\max_{a_k} \sum_{x_{k+1}} \ldots \max_{a_{m-1}} \sum_{x_m} (r_{k+1} + \ldots + r_m) \sum_{U(p:x_{\leq k}:a_{<m})=x_{k+1:m}*} 2^{-|p|}$$

For an infinite horizon with the discount sequence $\delta_k = \gamma^k$ $(0 < \gamma < 1)$, we would take the limit of the above for $m \to \infty$ and replace $r_{k+1} + \ldots + r_m$ with $\gamma^{k+1} r_{k+1} + \ldots + \gamma^m r_m$.

AIXI is the theoretical solution to the most general problem of agent-environment interaction considered in this course.

Concept Learning

Concept Learning

In concept learning, the agent tries to guess the environment state $s_k \in S$ at each k from the observation $o_k \in O$ and is immediately (at $k+1$) rewarded for a correct guess. With $S = \{0, 1\}$, the agent makes just yes/no decisions. To do that, it learns a representation of a *concept*, which is the set of all observations in O for which the correct answer is yes.

Formally, we return to the Markovian (state-based) setting, which is summarized below as a reminder, and which we will refine for concept learning. At time k:

- states are distributed by $P_S(s_{k+1}|s_k, a_k)$.
- observations are distributed by $P_o(o_k|s_k)$.
- rewards are distributed by $P_r(o_k|s_k)$, alternatively $P_r(r_{k+1}|s_k, a_k)$.

Concepts

In reinforcement learning, we further assumed full state observability, i.e. $\forall k$:

$$s_k = o_k$$

In concept learning, we make this assumption less stringent, in particular

$$s_k = c(o_k)$$

for some function $c : O \to S$ unknown to the agent. For convenience, we further assume $S = \{ 0, 1 \}$, allowing to represent c also as a subset of O:

$$C = \{ o \in O; c(o) = 1 \}$$

Then C is called a *concept* on O (c is called a concept function) and the agent's goal is to learn the concept so that it can make correct predictions of states.

Rewards

In reinforcement learning, we assumed that each reward was a *function* of the current state: $r_k = r(s_k)$. We carry on the function assumption, except we make r_k dependent on the previous state and action

$$r_{k+1} = r(s_k, a_k) \tag{7}$$

Since the agent's actions are guesses of the environment state, we set $A = S$. Then r_{k+1} should be higher when $s_k = a_k$ and lower otherwise. We will consider only the simplest form

$$r(s, a) = \begin{cases} 0 & \text{if} \quad s = a \\ -1 & \text{otherwise} \end{cases} \tag{8}$$

Notes: The choice between $r_k = r(s_k)$ and (7) is not essential for learnability results; some formulations of reinforcement learning also use (7). $L(s, a) = -r(s, a)$ is called a *loss function*; with (8) it is a *unit loss function*.

Example

Let O consist of pairs of binary values and assume the concept

$$C = \{ (o^1, o^2) \in O; o^1 \cdot o^2 = 1 \}$$

Say, the agent decides by the truth-value of a formula with variables $\mathrm{p}_1, \mathrm{p}_2$ assigned values o^1, o^2. On a mistake at time k recognized by $r_{k+1} = -1$, it changes the formula at time $k+1$.

k	s_k	o_k	r_k	agent's formula	a_k
1	0	$(0,0)$	—	$\neg \mathrm{p}_1$	1
2	1	$(1,1)$	-1	p_1	1
3	0	$(1,0)$	0	p_1	1
4	0	$(0,1)$	-1	$\mathrm{p}_1 \wedge \mathrm{p}_2$	0

After four trials and two errors, the agent guessed a formula which will no longer make mistakes.

Hypotheses

The agent's formulas in the previous example were concrete cases of *hypotheses*. A hypothesis h is any finite description, from which π can derive a $0/1$ decision for an observation.

You can think of π as a Turing machine and h as a program for it, but we will be interested in more specific cases, mainly $h \approx$ logical formulas, $\pi \approx$ their interpreters.

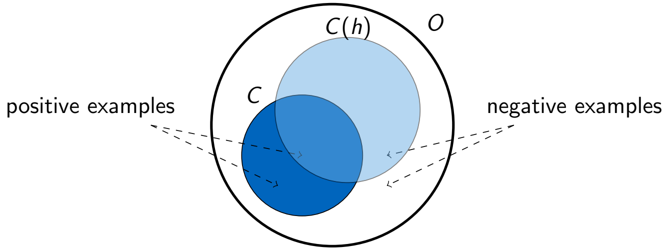
In our general agent model, we have $\pi : T \times O \to A$. Agent's hypothesis h_k at time k is part of its current state $t_k \in T$ but since h_k will be the only part of t_k influencing a_k, we will write $a_k = \pi(h_k, o_k)$ (not $\pi(t_k, o_k)$).

Finally, define the *hypothesized concept*:

$$C(h) = \{\, o \in O; \pi(h, o) = 1 \,\}$$

Concept vs. Hypothesis

A good concept-learning agent will find a h such that $C = C(h)$, which means the same as $c(o) = \pi(h, o)\ \forall o \in O$.



Concept Learning Terminology			
$C \cap \{ o_1, o_2, \dots \}$	..positive examples	$(O \setminus C) \cap \{ o_1, o_2, \dots \}$	..negative examples
$C \cap C(h)$	..true positives	$(O \setminus C) \cap (O \setminus C(h))$	..true negatives
$(O \setminus C) \cap C(h)$	..false positives	$C \cap (O \setminus C(h))$	..false negatives

Generalizing Agent

The agent in our previous example had the policy

$$a_k = \pi(h_k, o_k) = \begin{cases} 1 \text{ if } o_k \models h_k \\ 0 \text{ otherwise} \end{cases}$$

The way it guessed the formulas h_k seemed arbitrary. Can we make it systematic?

Assume $O = \{ 0, 1 \}^n$ and let h_k be *conjunctions* using n propositional variables. Then try this:

- Start with the conjunction of *all literals*, i.e., include both p and $\neg\mathrm{p}$ for each variable p.
- On each error, remove from the conjunction exactly all literals inconsistent with the previous observation (i.e. literals logically false in it).

Generalizing Agent Example

Again, concept $C = \{ (o^1, o^2) \in O; o^1 \cdot o^2 = 1 \}$, same sequence of observations but generalization strategy.

k	s_k	o_k	r_k	h_k	a_k
1	0	$(0,0)$	—	$\mathrm{p}_1 \wedge \neg \mathrm{p}_1 \wedge \mathrm{p}_2 \wedge \neg \mathrm{p}_2$	0
2	1	$(1,1)$	0	$\mathrm{p}_1 \wedge \neg \mathrm{p}_1 \wedge \mathrm{p}_2 \wedge \neg \mathrm{p}_2$	0
3	0	$(1,0)$	-1	$\mathrm{p}_1 \wedge \mathrm{p}_2$	1
4	0	$(0,1)$	0	$\mathrm{p}_1 \wedge \mathrm{p}_2$	0

$C(h_3) = C$. Hurray!

Note that the negative examples o_1, o_3 did not contribute to learning - they did not trigger a change of the hypothesis. Quiz: can negative examples ever make this agent change its hypothesis or are they useless?

Generalizing Agent: Properties

Assume a perfect conjunction h^* exists: $C(h^*) = C$, i.e. the target concept can be expressed through a conjunction. Observe:

1. All literals of h^* are consistent with any positive example (otherwise h^* would not be true for a positive example).
2. If a hypothesis misclassifies a negative example, it has all literals consistent with the example.
3. From 1 and 2: no literal that is in h^* is removed from the agent's hypothesis. Since h_1 contains *all* literals, we have $h_k \supseteq h^*$, $\forall k \in N$.
4. From 2 and 3: if h_k misclassifies a neg. example, then h^* ($\subseteq h_k$) is consistent with the example → contradiction → *h_k never ($\forall k$) misclassifies negative examples* (so yes, they are useless to this agent).

$h_k \supseteq h^*$ means that all literals of h^* are in h_k. We will use set relations for conjunctions and disjunctions this way without further warnings.

Generalizing Agent: Mistake Bound

1. Since h_k makes mistakes only on positive examples, whenever it misclassifies some o_k, it contains some literals inconsistent with o_k.
2. From 1 and the fact that all inconsistent literals are deleted after each mistake: at least one literal is deleted on each mistake.
3. From 2 and the fact that the initial hypothesis h_1 has $2n$ literals: *no more than $2n$ mistakes* are made before $h_k = h^*$.

So the cumulative reward for an arbitrary horizon $m \in N$ is

$$\sum_{k=1}^{m} r_k \geq -2n$$

Mistake-Bound Learning Model

Let $\mathcal{H}$ be a *hypothesis class*, i.e. any set of hypotheses. $\mathcal{H}$ induces the *concept class* $\mathcal{C}(\mathcal{H}) = \{ C(h) \mid h \in \mathcal{H} \}$. Let n be the *size* (complexity) of observations (usually their arity).

On-Line (Mistake-Bound) Learning

Agent *learns class $\mathcal{H}$ on-line (in the mistake-bound model)* if with any target concept $C \in \mathcal{C}(\mathcal{H})$ it will make no more than a polynomial (in n) number of mistakes. Furthermore, it learns $\mathcal{H}$ *efficiently* if it spends at most poly-time btw. each percept and the next action. $\mathcal{H}$ is *(efficiently) learnable on-line* if there is an agent that learns it (efficiently) on-line.

The definition only assumes $C \in \mathcal{C}(\mathcal{H})$ but makes no assumption about the sequence of states s_k from which observations are sampled with $P_O(o_k|s_k)$. The mistake bound must hold for any such sequence.

Verify that the generalization agent learns conjunctions *efficiently*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generality Relation

We call *concept C more general than concept C'* if $C \supseteq C'$. A *hypothesis h is more general than hypothesis h'* if

$$C(h) \supseteq C(h') \tag{9}$$

Assume h, h' are logic formulas and the policy is $\pi(h, o) = 1 \leftrightarrow o \models h$ as in the generalizing agent. Then a *sufficient* condition for (9) is that $o \models h'$ implies $o \models h$ for any model (valuation) o. In logic, this is written as

$$h' \models h \tag{10}$$

If furthermore h, h' are *conjunctions*, a *sufficient* condition for (10) is that

$$h \subseteq h' \tag{11}$$

because for any o, h: $o \models h$ iff *each literal* of h is consistent with o.

Generality vs. Subsumption

When $h \subseteq h'$ for conjunctions h, h', we say that h *subsumes* h'. Although $h \subseteq h'$ implies $h' \models h$, the reverse is not true if h' is *self-resolving*, i.e. it contains a literal as well as its negation. For example

$$\mathrm{p}_1 \wedge \neg \mathrm{p}_1 \models \mathrm{p}_2$$

but

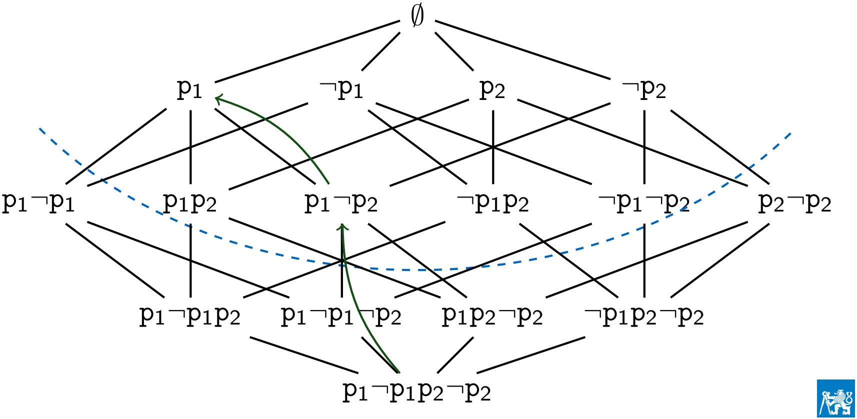
$$\mathrm{p}_2 \not\subseteq \mathrm{p}_1 \wedge \neg \mathrm{p}_1$$

Self-resolving propositional conjunctions are trivial, they are all contradictions. (This will not be the case in in first-order logic.)

We call the last agent *generalizing* because it only *deletes* literals (never adds them), so $\forall k$, $h_{k+1} \subseteq h_k$, thus $h_k \models h_{k+1}$, thus $C(h_{k+1}) \supseteq C(h_k)$.

Subsumption Lattice

The subsumption relation forms a *lattice* (partially ordered set) on conjunctions and the agent traverses it from the bottom (e.g. along the green path). Note that all conjunctions below the dashed line are self-resolving (contradictions). On the first positive example, the agent deletes *n* literals from the initial hypothesis, making the next one satisfiable (non-contradictory).



Least Upper Bound

In a lattice induced by relation $\preceq$, every two elements a, b have exactly one *least upper bound* $c = \mathrm{lup}(a, b)$ such that

- $c \preceq a$ and $c \preceq b$
- there is no "lesser bound" d, i.e. no d such that $c \preceq d$, $d \npreceq c$, $d \preceq a$, $d \preceq b$.

Properties of lup:

- $\mathrm{lup}(a, b) = a$ if $a \preceq b$
- $\mathrm{lup}(a, b) = \mathrm{lup}(b, a)$ (commutativity)
- $\mathrm{lup}(a, \mathrm{lup}(b, c)) = \mathrm{lup}(\mathrm{lup}(a, b), c)$ (associativity)

Least General Generalization

A subsumption lattice is induced by the subset relation $\subseteq$. When the lattice members are conjunctions or disjunctions of literals, we call the lup the *least general generalization* denoted lgg and clearly

$$\mathrm{lgg}(a, b) = a \cap b \tag{12}$$

Due to commutativity and associativity of any lup, lgg is defined uniquely for any finite *set* $\mathcal{H}$ of lattice elements:

$$\mathrm{lgg}(\mathcal{H}) = \mathrm{lgg}\left(h_1, \mathrm{lgg}\left(h_2, \mathrm{lgg}\left(h_3, \ldots\right)\right)\ldots\right) \tag{13}$$

where $h_1, h_2, h_3, \ldots$ are all elements of $\mathcal{H}$ in arbitrary order.

Due to (12) and (13)

$$\mathrm{lgg}(\mathcal{H}) = \bigcap_{h \in \mathcal{H}} h$$

Generalizing Agent: Design

Recall our Markovian agent model with the function $\mathcal{T}$ updating the agent's state given its previous state and the new percept:

$$t_{k+1} = \mathcal{T}(t_k, x_{k+1})$$

Where for each k, the state t_k contains the hypothesis h_k.

When h_k makes a mistake (i.e., $r_{k+1} = -1$), h_{k+1} should exclude all literals of h_k inconsistent with o_k. This is done at time $k+1$ so the agent needs to remember o_k as part of its state. So we will maintain the state in the form of a tuple

$$t_k = \langle h_k, o_k \rangle$$

Generalizing Agent: Design (cont'd)

The hypothesis will then be updated by

$$h_{k+1} = \begin{cases} h_k \text{ if } r_{k+1} = 0 \\ \mathrm{lgg}(h_k, \bar{o}_k) \text{ otherwise} \end{cases} \tag{14}$$

where $\bar{o}$ denotes a conjunction representing observation o:

$$\bar{o} = \bigwedge_{o^i=1} \mathrm{p}_i \bigwedge_{o^j=0} \neg \mathrm{p}_j \tag{15}$$

So e.g. for $n = 3$, $o_k = \langle 1, 0, 1 \rangle$, we have $\bar{o}_k = \mathrm{p}_1 \wedge \neg \mathrm{p}_2 \wedge \mathrm{p}_3$. Assume $h_k = \mathrm{p}_1 \wedge \mathrm{p}_2$ makes a mistake, i.e. $r_{k+1} = -1$. Then $h_{k+1} = \mathrm{lgg}(h_k, \bar{o}_k)$ $= \mathrm{p}_1$, so indeed the inconsistent literal was deleted.

Verify that generally, $lgg(h, \bar{o})$ deletes from h exactly those literals inconsistent with o.

Generalizing Agent in One Line

Since negative examples are irrelevant, assume without loss of generality that all of $o_1, \ldots, o_m$ are *positive*. Since h_1 contains all literals, we have $\bar{o}_1 \subseteq h_1$ and thus

$$\begin{aligned} h_2 &= \mathrm{lgg}(h_1, \bar{o}_1) = \bar{o}_1 \\ h_3 &= \mathrm{lgg}(h_2, \bar{o}_2) = \mathrm{lgg}(\mathrm{lgg}(h_1, \bar{o}_1), \bar{o}_2) = \mathrm{lgg}(\bar{o}_2, \bar{o}_1) \end{aligned}$$

So the agent's output can be written as

$$h_m = \mathrm{lgg}(\bar{o}_{m-1}, \mathrm{lgg}(\bar{o}_{m-2}, \ldots \mathrm{lgg}(\bar{o}_2, \bar{o}_1) \ldots)) = \mathrm{lgg}(\{\, \bar{o}_1, \ldots, \bar{o}_m \,\}) \quad (16)$$

The learned hypothesis is the least general generalization of all positive examples, which gives the intuition why negative examples are not needed: if the *least general* hypothesis was already *too general* (= covering a negative example), it means the target concept cannot be expressed through a conjunction.

Generalizing Agent for Disjunctions

Try the inverse strategy: ignore positive examples and take the lgg of the *negative examples*. Assume this time that $o_1, \ldots, o_m$ are all *negative*.

Say $h'_m = \mathrm{lgg}(\{\, \bar{o}_1, \ldots, \bar{o}_m \,\})$ covers no positives (otherwise there is no conjunction covering all negatives and no positives). Then clearly $\neg h'_m$ covers no negatives and all positives just like h_m from the previous page.

But h_m and $\neg h'_m$ are not the same: h_m is a conjunction while $\neg h'_m$ is the negation of a conjunction, i.e. a *disjunction*. So this inverse approach is suitable when the target concept can be expressed through a disjunction.

The agent can also compute both h_m and $\neg h'_m$ and output whichever one covers all positive and no negative examples. *Such an agent learns $\mathcal{H} =$ Conjunctions $\cup$ Disjunctions on n variables efficiently on-line*.

Sidenote: Non-Binary Observations

We have considered learning examples o to be binary tuples for simplicity. But what if observations are richer, e.g. tuples of rational numbers?

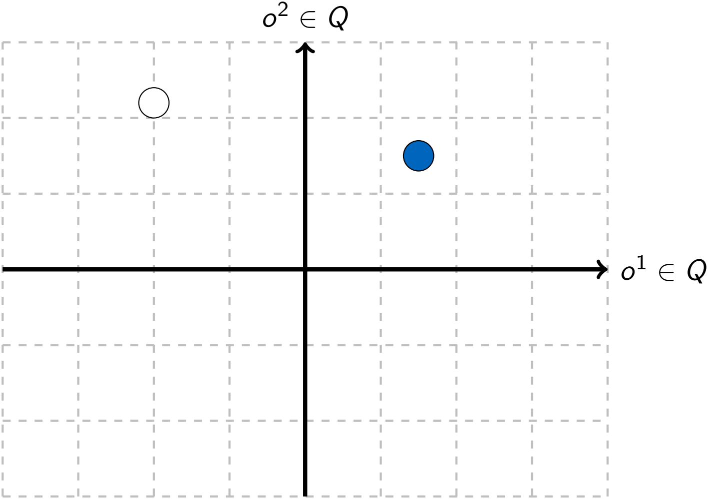
A practical way is to set *thresholds* $\theta^{i,j}$ for o^i along each axis i obtaining a (bigger) set of binary observations tuples o'.

$$\begin{array}{ccc} o'^{1,1} = 1 \text{ iff } o^1 > \theta^{1,1} & o'^{1,2} = 1 \text{ iff } o^1 > \theta^{1,2} & \dots \\ o'^{2,1} = 1 \text{ iff } o^2 > \theta^{2,1} & o'^{2,2} = 1 \text{ iff } o^2 > \theta^{2,2} & \dots \\ \dots & \dots & \dots \end{array}$$

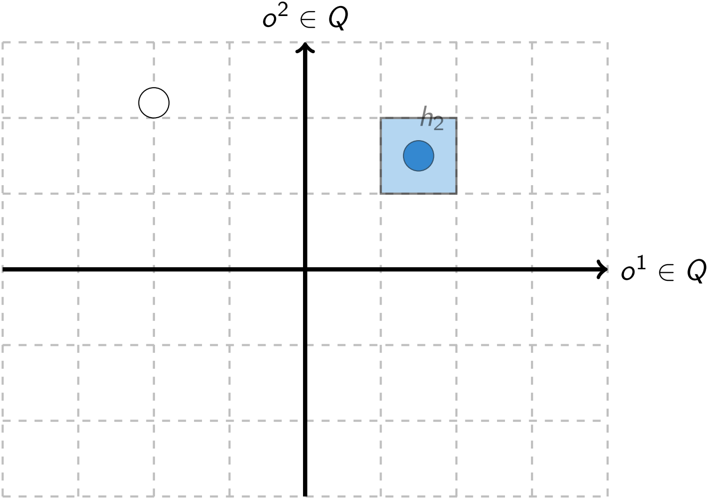
Choosing good threshold is a task for *discretization* techniques which are out of our scope and out of the scope of learnability theory.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

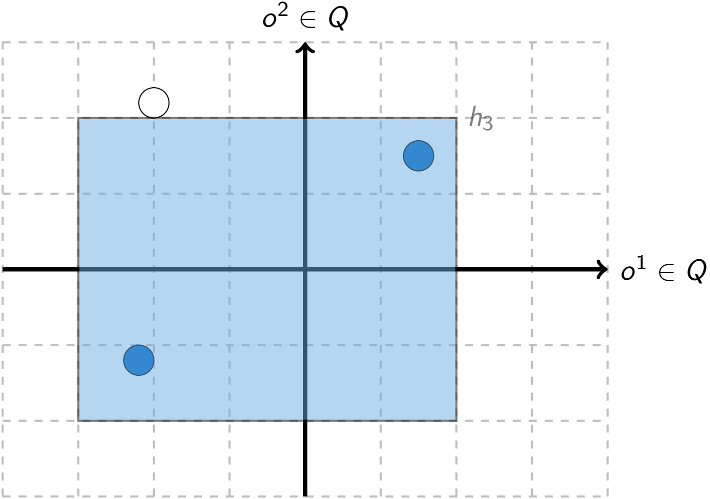
Visualizing Generalization (with Rational Features)



Visualizing Generalization (with Rational Features)



Visualizing Generalization (with Rational Features)



Generalizing Agent for s-CNF, s-DNF

An *s-CNF* is a conjunction of *s-clauses*, which is a disjunction of at most $s \in \mathbb{N}$ literals. s-CNF's can be learned using the generalization strategy. Given n propositional variables:

- Set h_1 to the conjunction of *all* s-clauses on these variables.
- One each positive example, remove from h_k all clauses false for the example.

Reasoning just like for the conjunction-learning agent, the number of mistakes will not be greater than the number of all s-clauses on n variables. This number is $\mathcal{O}\left[\binom{2n}{s}\right] = \mathcal{O}(n^s)$, i.e. *polynomial*. Therefore, *s-CNF's are learnable online*. Check that they are also learned *efficiently* .

Show the same for *s-DNF*, i.e. disjunctions of *s-terms*, which are conjunctions of at most s literals.

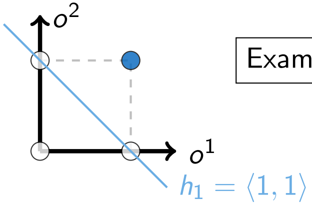
Separating agent

Besides generalization, another prominent learning technique is to *separate* C from its complement with a hyperplane in O. We continue with $O = \{ 0, 1 \}^n$. Agent's hypothesis is a tuple of integers ('weights'), i.e.

$$h_k = \left\langle h_k^1, h_k^2, \ldots, h_k^n \right\rangle$$

and $h_1 = \langle 1, 1, \ldots 1 \rangle$. Decisions are:

$$a_k = \pi(h_k, o_k) = \begin{cases} 1 \text{ if } h_k \cdot o_k > n/2 \text{ (dot product)} \\ 0 \text{ otherwise} \end{cases}$$



Example for $n = 2$

Separating agent: Design

The agent (called Winnow in literature) has a simple learning rule:

- On a false negative o_k, *double* weights h^i for all i, $o^i_k = 1$
- On a false positive, *nullify* weights of these features.

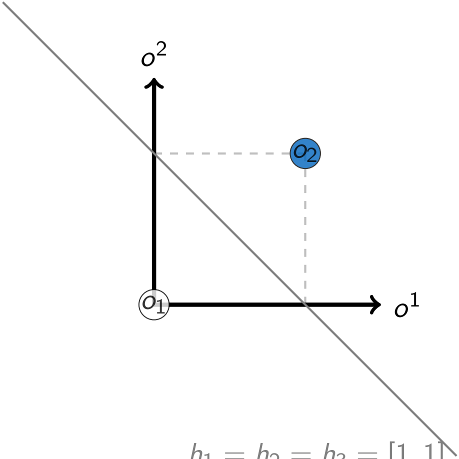
Formally:

$$h_{k+1} = \begin{cases} h_k \text{ if } r_{k+1} = 0 \text{ (Keep } h_k \text{ if it did not make a mistake.)} \\ \mathrm{update}(2, h_k, o_k) \text{ if } r_{k+1} = -1 \text{ and } h_k \cdot o_k \leq n/2 \\ \mathrm{update}(0, h_k, o_k) \text{ if } r_{k+1} = -1 \text{ and } h_k \cdot o_k > n/2 \end{cases}$$

where $\mathrm{update}(\alpha, h_k, o_k)$ multiplies all features with value 1 by α, i.e.

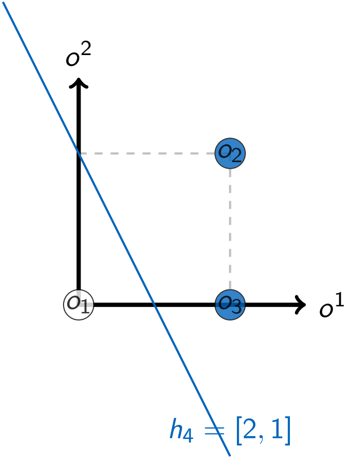
$$h^i_{k+1} = \begin{cases} \alpha \cdot h^i_k \text{ if } o^i_k = 1 \\ h^i_k \text{ otherwise} \end{cases}$$

Visualizing Separation

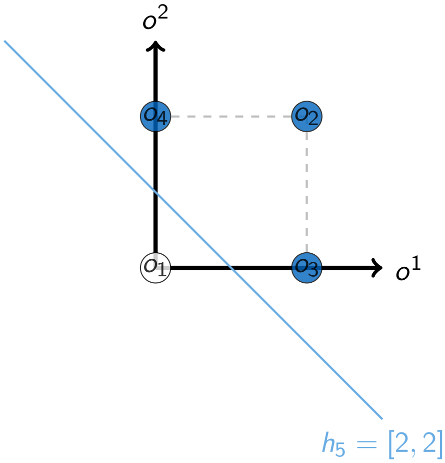


$h_1 = h_2 = h_3 = [1, 1]$

Visualizing Separation



Visualizing Separation



Separating agent: Properties

Agent's properties, omitting proofs:

- Learns efficiently any class $\mathcal{H}$ of linearly separable hypotheses. A linearly separable hypothesis h is such that $C(h)$ is separable from $O \setminus C(h)$ on O. This includes conjunctions and disjunctions.
- Let the target hypothesis h^* $(C(h^*) = C)$ be an *s-conjunction* or an *s-disjunction*, i.e. a conjunction (disjunction) *with at most s literals*. Then the agent makes at most $2 + 2s \log n$ mistakes. For sufficiently small s, this is better than the generalizing agent which has a mistake bound linear in n.

Similarly to the generalizing agent, it can be extended to learning s-CNF's or s-DNF's by using the poly-size set of features corresponding to all possible s-clauses (s-terms).

Sidenotes on the Separating agent

The perceptron algorithm is similar to the separating (Winnow) agent, using real-valued weights and a gradient-based learning rule, allowing non-binary observation vectors. Aritificial neural networks are popular machine-learning models and they are networks of perceptrons.

Numerous machine-learning algorithms including support vector machines follow the separation-by-hyperplane strategy. Non-linear separation (by a hypercurve) can be achieved by a suitable expansion of the observation vectors such as by adding to them cross-products $o^i o^j$ of all feature pairs.

These methods are out of our scope (find them in the Statistical ML class instead) as we are concerned mainly with symbolic, interpretable hypotheses.

General Learning Agent

Consider an agent *general* in the sense that it applies the same strategy using any given hypothesis class $\mathcal{H}$. It keeps *a set* $\mathcal{H}_k$ of hypotheses, rather than a single one starting with $\mathcal{H}_1 = \mathcal{H}$.

At each k, the agent picks an arbitrary h from $\mathcal{H}_k$ and makes a decision by it. If it makes an error, it deletes h from $\mathcal{H}_k$. Formally,

$$\mathcal{H}_{k+1} = \begin{cases} \mathcal{H}_k \text{ if } r_{k+1} = 0 \text{ (Keep } \mathcal{H}_k \text{ if it did not make a mistake.)} \\ \mathcal{H}_k \setminus \{\, h \,\} \text{ if } r_{k+1} = -1 \end{cases}$$

If $C \in \mathcal{C}(\mathcal{H})$, then this agent makes at most $|\mathcal{H}| - 1$ mistakes. (If all hypotheses made a mistake, the last one must be the target.)

General Learning Agent

So the agent learns online (not necessarily efficiently) any $\mathcal{H}$ such that $|\mathcal{H}| \leq poly(n)$, including

- s-conjunctions and s-disjunctions:

$$|\mathcal{H}| = \binom{n}{s} + \binom{n}{s-1} + \ldots + \binom{n}{0} \leq poly(n)$$

but not e.g.

- unconstrained conjunctions or disjunctions:

$$|\mathcal{H}| = 2^{2n} \text{ (including self-resolving)}$$
$$|\mathcal{H}| = 3^{n} \text{ (without self-resolving).}$$

Version Space

The version space agent is similar to the general agent but decides by a majority vote among all hypotheses in $\mathcal{H}_k$ and deletes from $\mathcal{H}_k$ *all* hypotheses inconsistent with the last observation. The agent's state t_k consists of $\mathcal{H}_k$ and the memorized observation o_k.

Specifically, when $\mathcal{H}_k$ contains logical formulas and o_k are valuations:

$$a_k = \pi(\mathcal{H}_k, o_k) = \begin{cases} 1 \text{ if } |\{ h \in \mathcal{H}_k \mid o_k \models h \}| > |\mathcal{H}_k|/2 \\ 0 \text{ otherwise} \end{cases}$$

$$\mathcal{H}_{k+1} = \begin{cases} \{ h \in \mathcal{H}_k \mid o_k \models h \} \text{ if } s_k = 1 \\ \{ h \in \mathcal{H}_k \mid o_k \not\models h \} \text{ if } s_k = 0 \end{cases}$$

Note: $s_{k+1} = |a_k + r_{k+1}|$ where $a_k = \pi(\mathcal{H}_k, o_k)$, so the above can indeed be evaluated at update time.

Version Space (cont'd)

If a mistake is made, at least half of the hypotheses are $\mathcal{H}_k$ are deleted. In the worst case, the last remaining hypothesis is correct.

So the agent makes at most $\log |\mathcal{H}|$ mistakes, i.e. the cumulative reward is

$$\sum_{k=1}^{m} r_k \geq -\log |\mathcal{H}| \tag{17}$$

for any horizon $m \in N$.

Version space *not efficient* even for super-poly (not just super-exp) $\mathcal{H}$. Needs to verify each $h \in \mathcal{H}$ in the update step.

Version Space (cont'd)

If $|\mathcal{H}|$ at most exponential then $\log |\mathcal{H}|$ polynomial and VS agent learns $\mathcal{H}$ online. This includes s-clause CNF's and s-term DNF's.

What about $\mathcal{H}$ covering *all* possible concepts on $O = \{\, 0, 1 \,\}^n$, i.e.

$$\mathcal{C}(\mathcal{H}) = 2^O$$

We would not need to worry whether $C \in \mathcal{C}(\mathcal{H})$.

Since $|O| = 2^n$, we have $|\mathcal{H}| \geq 2^{|O|} = 2^{(2^n)}$, so $|\mathcal{H}|$ is super-exponential. So, nice try but no on-line learning.

Even some hypothesis classes which are more reasonable are super-exponential (we will see later).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

VC Dimension

Concept class $\mathcal{C}$ *shatters* $O' \subseteq O$ if any subset of O' coincides with $C \cap O'$ where $C \in \mathcal{C}$. A hypothesis class $\mathcal{H}$ shatters O' if $\mathcal{C}(\mathcal{H})$ shatters O'.

So O' is shattered by $\mathcal{C}$ (resp. $\mathcal{H}$) if its elements can be classified in all $2^{O'}$ possible ways by concepts from $\mathcal{C}$ (hypotheses from $\mathcal{H}$).

Vapnik-Chervonenkis Dimension

The *VC-dimension* of $\mathcal{C}$ (on O) denoted VC($\mathcal{C}$) is the cardinality of the largest subset of O shattered by $\mathcal{C}$. The VC-dimension of hypothesis class $\mathcal{H}$ is defined as VC($\mathcal{C}(\mathcal{H})$), also denoted VC($\mathcal{H}$) .

Note: the definition does not assume $\mathcal{C}$ or $\mathcal{H}$ finite.

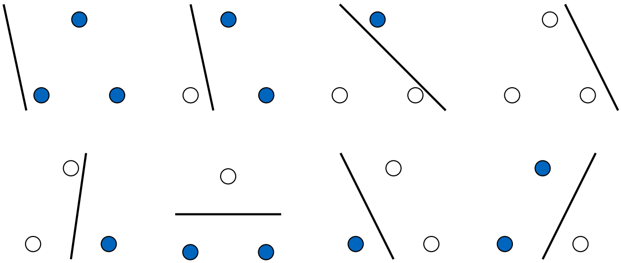
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Determining VC-Dimension

- If *some* $O' \subseteq O$ shattered by $\mathcal{C}$ then $\mathrm{VC}(\mathcal{C}) \geq |O'|$.
- If *none* $O' \subseteq O$ shattered by $\mathcal{C}$ then $\mathrm{VC}(\mathcal{C}) < |O'|$.

Example: $\mathcal{C}$ = half-planes in $\mathbb{R}^2$ (i.e., linear separation)

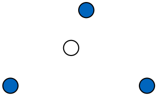
- *Some* 3 points can be shattered



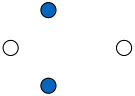
so $\mathrm{VC}(\mathcal{C}) \geq 3$.

Determining VC-Dimension (cont'd)

- *No* 4 points can be shattered. Obvious if 3 in line. Otherwise two cases possible:



One point in the middle



No point in the middle

In both cases, the shown subset cannot be separated by a line. So $\text{VC}(\mathcal{C}) < 4$

We have $\text{VC}(\mathcal{C}) \geq 3$ and $\text{VC}(\mathcal{C}) < 4$, thus $\text{VC}(\mathcal{C}) = 3$.

Lower Bounds on Mistake Bounds

No general lower bound on mistake *counts* as the agent may simply be lucky guessing right each time. But mistake *bounds* can be lower-bounded.

A mistake bound with no special assumptions cannot be lower than $|O|$ as each $o \in O$ may have an arbitrary class.

A mistake bound assuming only $C \in \mathcal{C} \subset 2^O$ *for the target concept* $\mathcal{C}$ *cannot be smaller than* $\mathrm{VC}(\mathcal{C})$ as there is a set $\{ o_1, o_2, \ldots, o_{\mathrm{VC}(\mathcal{C})} \} \subseteq O$ shattered by $\mathcal{C}$. So for the observation sequence $o_1, o_2, \ldots, o_{\mathrm{VC}(\mathcal{C})}$ and any sequence of agent's decisions $a_1, a_2, \ldots, a_{\mathrm{VC}(\mathcal{C})}$ there is a target concept $C \in \mathcal{C}$ by which all these decisions are wrong.

Corollary: an agent using hypothesis class $\mathcal{H}$ cannot be guaranteed to make fewer than $\mathrm{VC}(\mathcal{H})$ mistakes.

I.I.D. Examples

So far we have maintained the general Markovian state transitions, so s_{k+1} was distributed according to

$$P_S(s_{k+1}|s_k, a_k)$$

The mistake bound model did not put any assumption on P_S. Now, we will assume that *s_{k+1} does not depend on s_k or a_k*, so all s_k are sampled from the same distribution

$$P_S(s)$$

so the s_k are *identically and independently distributed*, *i.i.d.* for short. As a consequence, observations o_k are also i.i.d from $P_O(o)$ where

$$P_O(o) = \sum_{s \in \{0,1\}} P_O(o|s) P_S(s)$$

Hypothesis Accuracy and Error

The i.i.d. assumption lets us define the *error of a hypothesis h* as the total probability of observations, which h decides incorrectly

$$\operatorname{err}(h) = P_O\left([C \setminus C(h)] \bigcup [C(h) \setminus C]\right)$$

and the *accuracy of h* as

$$\operatorname{acc}(h) = 1 - \operatorname{err}(h)$$

In this setting, a large V^{π} is achieved when π uses a hypothesis with a small error, so the agent should minimize it. This gives rise to a new learning model.

Both acc and err are with respect to the distribution P_O and the target concept C, which we do not show explicitly in their notation.

The PAC Learning Model

Informally, PAC-learning means finding a low-error hypothesis with high probability using a polynomial number of observations.

Probably Approximately Correct (PAC) Learning

Agent *probably approximately correctly (PAC) learns $\mathcal{H}$* if for any $C \in \mathcal{C}(\mathcal{H})$ and numbers $0 < \epsilon, \delta < 1$, there is a $k < poly(n, 1/\epsilon, 1/\delta)$ such that the agent's hypothesis h_k has $\mathrm{err}(h_k) \leq \epsilon$ with probability at least $1 - \delta$. $\mathcal{H}$ is *PAC-learnable* if there is an agent that PAC-learns it.

n is again the size of observations; with $O = \{ 0, 1 \}^n$, it is simply their arity.

Note that if O is finite and $\epsilon = \min_{o \in O} P_O(o)$, then with prob. at least $1 - \delta$, h_k is correct for *all* observations, i.e. $\mathrm{err}(h_k) = 0$.

Efficient and Proper PAC Learning

Analogically to the mistake-bound model, we say the agent PAC-learns $\mathcal{H}$ *efficiently* if it spends at most $poly(n, 1/\epsilon, 1/\delta)$ time between each percept and the subsequent action; if there is such an agent, $\mathcal{H}$ is *efficiently PAC-learnable*.

Note that in the definition of PAC learning we do not assume that $h_k \in \mathcal{H}$. In general, the agent may work with a hypothesis class larger than $\mathcal{H}$; for example with $\mathcal{H} =$conjunctions, h_k may be a 3-CNF equivalent to the target conjunction. With the additional condition that $h_k \in \mathcal{H}$, we say that the agent *properly* (efficiently) PAC-learns $\mathcal{H}$, and if there is such an agent for $\mathcal{H}$, we say that $\mathcal{H}$ is *properly* (efficiently) PAC-learnable.

Generalizing Agent in the PAC Model

Let us analyze the generalizing agent again, this time with the i.i.d. assumption.

Let $O_l \subseteq O$ denote the set of all observations *inconsistent* with a literal l. We know already that a hypothesis h makes a mistake for an observation only if it has a literal inconsistent with it, so

$$\mathrm{err}(h) \leq \sum_{l \in h} P_O(O_l)$$

With n variables, there are at most $2n$ literals in h so if $P_O(O_l) \leq \epsilon/2n$ for each literal $l \in h$ then $\mathrm{err}(h) \leq \epsilon$. Call literal l *bad* if

$$P_O(O_l) > \frac{\epsilon}{2n} \tag{18}$$

Generalizing Agent in the PAC Model (cont'd)

Let l be bad. At time $k+1$, the probability that $l \in h_{k+1}$ is the probability that l is consistent with k i.i.d. observations (else it would have been removed), i.e. $(1 - P_O(O_l))^k$. Due to (18), we have

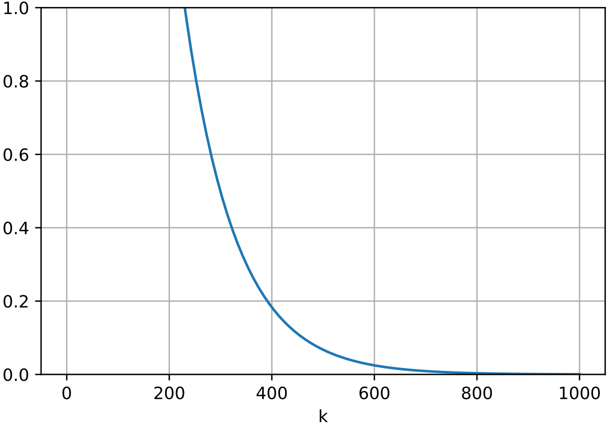
$$(1 - P_O(O_l))^k < \left(1 - \frac{\epsilon}{2n}\right)^k$$

Prob. that *some* bad literal is consistent with the k observations is upper bounded by the above times the number of all literals so it is at most:

$$2n\left(1 - \frac{\epsilon}{2n}\right)^k < 2ne^{-k\frac{\epsilon}{2n}}$$

as there are at most $2n$ bad literals. For the right-hand side, we used $1 - x < e^{-x}$, $x \in (0; 1)$.

$2ne^{-k\frac{\epsilon}{2n}}$, $\epsilon = 0.1, n = 5$



Upper bound for probability that a bad literal remains in h_{k+1} of the generalizing agent with $n = 5$ variables. Note the looseness of this bound: from reasoning in the mistake-bound model, we know that $h_{2n+1} = h_{11}$ is already correct so it has no bad literal.

PAC-Learning with the Generalization Agent

To satisfy PAC-learning conditions, we need to make it

$$2ne^{-k\frac{\epsilon}{2n}} < \delta$$

which is equivalent to

$$k \geq \frac{2n}{\epsilon} \ln \frac{2n}{\delta}$$

So for $k = \frac{2n}{\epsilon} \ln \frac{2n}{\delta}$, $\text{err}(h_{k+1}) \leq \epsilon$ with probability at least δ. Since $k + 1 \leq poly(n, 1/\delta, 1/\epsilon)$, *the agent PAC-learns conjunctions*.

It also learns them efficiently as it spends only $2n$ unit steps (checking each literal's consistency) on each observation.

Through adaptations we have discussed, it also efficiently PAC-learns disjunctions, s-CNF's and s-DNF's.

Standard On-line Agent

An on-line agent deciding by $\pi(h, o)$ is *standard* if it changes its hypothesis ($h_{k+1} \neq h_k$) if and only if h_k makes an error ($r_{k+1} = -1$). This includes the generalizing and separating agent but not e.g. version-space.

Consider any standard on-line agent with mistake bound M. Let $q \in \mathbb{N}$ and $k = Mq$. In the agent's sequence of hypotheses $h_1, h_2, \dots h_{k+1}$, there must be a hypothesis h retained for at least q consecutive steps, i.e. $\exists K \leq k$ such that $h = h_K = h_{K+1} = \dots = h_{K+q}$.

Assume for contradiction that all subsequences of identical hypotheses are shorter than q. Then there are more than M different hypotheses among $h_1, h_2, \dots h_{k+1}$ because $k = Mq$. The agent changes its hypothesis only on a mistake, so this means it has made more than M mistakes up to time k. This is a contradiction because M is the error bound.

PAC-Learning with any Standard On-Line Agent

The probability that a hypothesis h is consistent with q i.i.d. observations is $(1 - \mathrm{err}(h))^q$. Call a hypothesis *bad* if $\mathrm{err}(h) > \epsilon$. So a bad hypothesis is consistent with q i.i.d. observations with probability at most $(1-\epsilon)^q$.

Consider any standard agent learning $\mathcal{H}$ online in the mistake-bound model, i.e., it makes at most $M < poly(n)$ mistakes. Let $k = Mq$. We already know that until time $k+1$ the agent had a hypothesis h consistent with q consecutive observations.

This h is thus bad with probability at most $(1-\epsilon)^q$. With $q = \frac{1}{\epsilon} \ln \frac{1}{\delta}$, this is $(1-\epsilon)^q \leq e^{-q\epsilon} = e^{-\epsilon \frac{1}{\epsilon} \ln \frac{1}{\delta}} = \delta$. Since both M and q are $\leq poly(n, 1/\delta, 1/\epsilon)$, so is $k+1 = Mq+1$. Thus *if a standard agent (efficiently) learns $\mathcal{H}$ online it also (efficiently) PAC-learns it*.

Training Set and Batch Learning

Consider concept learning in an interaction with a *finite* horizon $m+1$. The agent's goal is to minimize the error of its last hypothesis $\mathrm{err}(h_{m+1})$.

Recall that in our interaction scenario, the class $s_k = c(o_k)$ of o_k is determined at $k+1$ as $|a_k + r_{k+1}|$. So at time step $m+1$ the agent has received exactly m observations with determined classes:

$$T = \{\, \langle o_1, s_1 \rangle \,, \langle o_2, s_2 \rangle \,, \ldots, \langle o_m, s_m \rangle \,\} \tag{19}$$

which is called the *training (multi-)set*.

Rather than updating the hypothesis at each $k = 2, 3, \ldots m+1$, the agent can simply store the training set in its state (memory), and only at time $m+1$ compute a hypothesis from T. We will call this *batch learning*.

Consistency with the Training Set

The hypothesis a PAC-learning agent computes from any training set is consistent with it.

Let h be the hypothesis computed from T (19) and assume for contradiction that h misclassifies some $o \in \{ o_1, \ldots, o_m \}$. P_O and $0 < \delta < 1$ can be arbitrary so set them such that $\delta < \prod_{k=1}^{m} P_O(o_k)$, i.e. T is received with probability greater than δ. This implies that $P_O(o_k) > 0$ for $1 \leq k \leq m$, so also $P_O(o) > 0$. Because of that and since $0 < \epsilon < 1$ can be arbitrary, we can set ϵ such that $\epsilon < P_O(o)$. Since h misclassifies o, it follows $\text{err}(h) \geq P_O(o)$ and because $P_O(o) > \epsilon$, we have $\text{err}(h) > \epsilon$. This happens when T is received, i.e. with probability greater than δ. This contradicts PAC-learning conditions.

Consistent Agent

Consider a general *consistent agent* learning $\mathcal{H}'$ and equipped with some hypothesis class $\mathcal{H}$. Given a training set T of size m, it produces an arbitrary $h \in \mathcal{H}$ consistent with T.

Note that producing a consistent hypothesis for any given T is only possible if the target concept C is in $\mathcal{C}(\mathcal{H})$. $C \in \mathcal{C}(\mathcal{H})$ is arbitrary, so

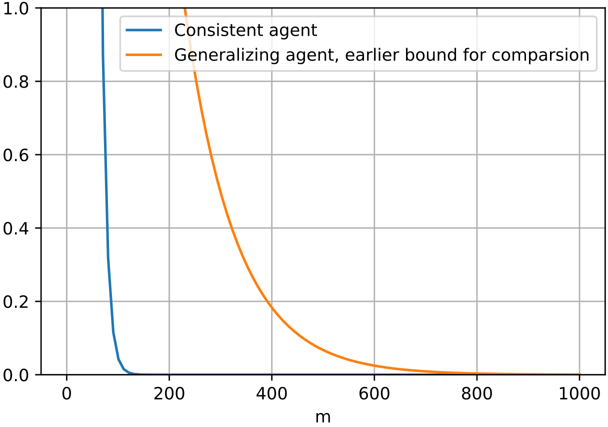
$$\mathcal{C}(\mathcal{H}) \supseteq \mathcal{C}(\mathcal{H}') \tag{20}$$

is a necessary condition.

The probability that a hypothesis $h \in \mathcal{H}$ consistent with T is bad ($\mathrm{err}(h) > \epsilon$) is $(1 - \mathrm{err}(h))^m < (1 - \epsilon)^m$. There are at most $|\mathcal{H}|$ hypotheses so the probability that *some* bad hypothesis is consistent with T is at most

$$|\mathcal{H}|(1 - \epsilon)^m < |\mathcal{H}|e^{-\epsilon m}$$

$|\mathcal{H}|e^{-\epsilon m}$, $\epsilon = 0.1$, $n = 5$



Upper bound on prob. that err(h) $> \epsilon$ with a consistent agent and $\mathcal{H} =$ conjunctions (blue). The bound applies also to the generalizing agent, which is consistent, and is tighter than the previous bound derived specifically for it (orange).

PAC-Learning with any Consistent Agent

$|\mathcal{H}|e^{-\epsilon m}$ is smaller than δ if

$$m \geq \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$$

Since m is polynomial in $1/\epsilon$ and $1/\delta$, *PAC-learns $\mathcal{H}$* if m is further polynomial in n. The only factor on the right-hand side depending on n is $\ln |\mathcal{H}|$, so the condition is

$$\ln |\mathcal{H}| \leq poly(n)$$

i.e., $|\mathcal{H}|$ is at most *exponential* in n.

To learn $\mathcal{H}'$ *properly* we further require $\mathcal{H} \subseteq \mathcal{H}'$, which together with (20) implies $\mathcal{H}' = \mathcal{H}$.

Sizes of Some $\mathcal{H}$ (You do the math)

$\mathcal{H}$	$\lvert\mathcal{H}\rvert$ in increasing size	
s-conjunctions, s-disjunctions	$\mathcal{O}\left[(2n)^s\right]$ (incl. self-resolv.)	poly
s-depth decision trees	$2(2n)^{2^s-1}$	poly
conjunctions, disjunctions	2^{2n} (3^n if no self-resolving)	exp
s-term DNF, s-clause CNF	$\mathcal{O}\left[(3^n)^s\right]$	exp
s-CNF, s-DNF	$\mathcal{O}\left[2^{\binom{2n}{s}}\right] = \mathcal{O}\left[2^{(n^s)}\right]$	exp
dec. trees, DNF, CNF, ...	$\geq 2^{(2^n)}$ (# all concepts)	super-exp

Notes:

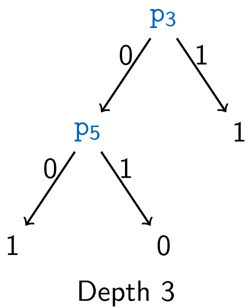
s-term DNF, s-clause CNF. At most s non-self-resolving terms (clauses) of unlimited size.

DNF, CNF. $\lvert\mathcal{H}\rvert = 2^{(3^n)}$. There are only $2^{(2^n)}$ possible concepts on $O = \{0, 1\}^n$, so $\mathcal{H}$ has equivalent pairs in it.

decision trees. Can express any concept so $\lvert\mathcal{H}\rvert \geq 2^{(2^n)}$

s-depth decision trees. See next slide.

Decision Trees with max depth s



Denote $\mathrm{dt}(s) = |\mathcal{H}|$ where $\mathcal{H}$ contains decision trees with maximal depth s on n variables.

$$\mathrm{dt}(1) = 2$$

...just a vertex with 0 or 1. For $s = 2, 3, \dots,$

$$\mathrm{dt}(s) = n \cdot \mathrm{dt}(s-1)^2$$

n choices for root, $\mathrm{dt}(s-1)$ possible subtrees on both left and right. Take a log of both sides and solve the arithmetic series, yielding

$$\mathrm{dt}(s) = 2(2n)^{2^s-1}$$

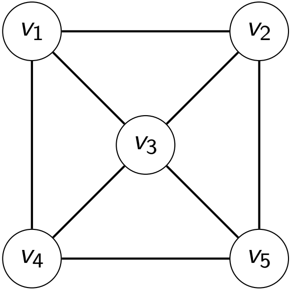
which is poly in n.

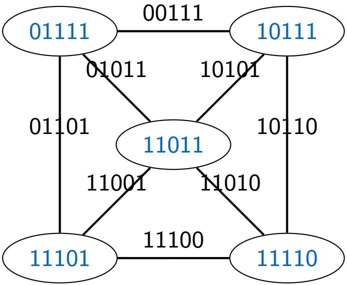
Finding a Consistent 3-term DNF

The NP-complete graph 3-coloring problem can be reduced in poly-time to finding an 3-term DNF consistent with a training set.

$$\text{vertex } v_i \quad \leftrightarrow \quad \text{pos. example } o,\ o^l = \begin{cases} 0 \text{ if } l = i \\ 1 \text{ otherwise} \end{cases}$$

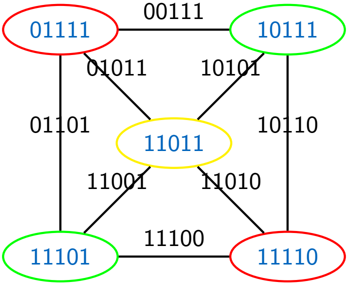
$$\text{edge } e_{ij} \quad \leftrightarrow \quad \text{neg. example } o,\ o^l = \begin{cases} 0 \text{ if } l = i \text{ or } l = j \\ 1 \text{ otherwise} \end{cases}$$





Finding a Consistent *s*-term DNF (cont'd)

Graph 3-colorable iff a 3-term DNF consistent with the training set exists.



$$\bigvee_{\substack{\textbf{color} \in \\ \{R, G, Y\}}} \; \bigwedge_{\substack{v_i \text{ not of} \\ \textbf{color}}} p_i \quad \xrightarrow{\Rightarrow} \quad (p_2 \wedge p_3 \wedge p_4) \vee (p_1 \wedge p_3 \wedge p_5) \vee (p_1 \wedge p_2 \wedge p_4 \wedge p_5)$$

$\Leftarrow$

color of any consistent term

3-colorability NP-hard → finding a consistent 3-term DNF NP-hard. Can be generalized to *s*-term DNF's. So *s-term DNF's are not efficiently properly* PAC-learnable.

PAC-Learning s-term DNF Efficiently using s-CNF

Every k-term DNF formula can be written as a logically equivalent k-CNF formula. Example:

$$(\mathrm{p}_1 \wedge \mathrm{p}_2) \vee (\mathrm{p}_2 \wedge \mathrm{p}_3) \equiv (\mathrm{p}_1 \vee \mathrm{p}_2) \wedge (\mathrm{p}_1 \vee \mathrm{p}_3) \wedge \mathrm{p}_2 \wedge (\mathrm{p}_2 \vee \mathrm{p}_3)$$

Therefore

$$\mathcal{C}(s\text{-term DNF's}) \subseteq \mathcal{C}(s\text{-CNF's}) \tag{21}$$

Thus *by using $\mathcal{H} = s$-CNF's, the agent efficiently PAC-learns $\mathcal{H}' = s$-term DNF's*. Since s-CNF ($\mathcal{O}\left[2^{(n^s)}\right]$) has larger cardinality than s-term DNF ($\mathcal{O}\left[(3^n)^s\right]$), the inclusion (21) is strict, i.e. $\mathcal{C}(\mathcal{H}') \subset \mathcal{C}(\mathcal{H})$. Thus the agent *does not PAC-learn $\mathcal{H}'$ properly* as the final s-CNF hypothesis may not be expressible as an s-term DNF.

PAC-Learnability Due to VC-Dimension

PAC-Learnability Due to VC-Dimension

If h is consistent with

$$m \geq \max\left\{\frac{4}{\epsilon}\log_2\frac{2}{\delta}, \frac{8 \cdot \mathrm{VC}(\mathcal{H})}{\epsilon}\log_2\frac{13}{\epsilon}\right\}$$

i.i.d. training examples, then $\mathrm{err}(h) \leq \epsilon$ with probability at least $1-\delta$.

So a consistent agent using $\mathcal{H}$ PAC-learns any $\mathcal{H}'$ s.t. $\mathcal{C}(\mathcal{H}') \subseteq \mathcal{C}(\mathcal{H})$ as long as $\mathrm{VC}(\mathcal{H})$ is at most polynomial.

For infinite $\mathcal{H}$, $\mathrm{VC}(\mathcal{H})$ plays a role “analogical” to $\ln|\mathcal{H}|$ which is defined only for finite $\mathcal{H}$. The result is useful also for some finite hypothesis classes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learnability Due to VC-Dimension: Example

Consider learning a threshold hypothesis $h \in [0; 1]$ with $O = [0; 1]$.

$$\pi(h, o) = \begin{cases} 1 \text{ if } o > h \\ 0 \text{ otherwise} \end{cases}$$

Assume finite precision. $|\mathcal{H}| = 2^b$ where b is the number of bits to represent $h \in \mathcal{H}$. Values of $m = \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$ for some b and $\epsilon = \delta = 0.1$:

$b = 64$	$b = 128$	$b = 256$
$m = 467$	$m = 911$	$m = 1798$

whereas $\text{VC}(\mathcal{H}) = 1$ independently of precision and the VC bound gives $m = 562$.

Beyond Consistent Learning / Concept Learning

Consistent learning is impossible if $\mathcal{C}(\mathcal{H}') \not\subseteq \mathcal{C}(\mathcal{H})$, or if the very concept assumption $s = c(o)$ is not met (a training set T may then contain the same observation several times with different class labels).

Then settle for the *empirical risk minimization (ERM) principle*:

$$h = \arg\min_{h \in \mathcal{H}} \widehat{\text{err}}(h, T) = \arg\min_{h \in \mathcal{H}} \frac{1}{|T|} |\{ \langle o, s \rangle \in T \mid \pi(h, o) \neq s \}|$$

where $\widehat{\text{err}}(h, T)$ defined by the equation is called the *training error* or *empirical risk*.

We no longer have PAC-learning guarantees but probabilistic bounds on $\text{err}(h)$ can be estimated from $\widehat{\text{err}}(h, T)$ and $\ln |\mathcal{H}|$ or $\text{VC}(\mathcal{H})$. Details in the Statistical Machine Learning class.

Learning from Conjunctions

In (15), we formally converted truth-value assignments o to conjunctions $\bar{o}$. For example, for $o = \langle 1, 0, 1 \rangle$, $\bar{o} = \mathrm{p}_1 \wedge \neg \mathrm{p}_2 \wedge \mathrm{p}_3$. We now explore learning from observations o which are already conjunctions. Unlike $\bar{o}$, they are *arbitrary*, i.e. need not contain a literal for every atom $\mathrm{p}_1, \mathrm{p}_2, \ldots \mathrm{p}_n$.

Denote by O^+ (O^-) is the set of all received positive (negative) conjunctive examples. The agent should find a conjunction h such that

$$\begin{aligned} o^+ &\models h \text{ for all } o^+ \in O^+ \\ o^- &\not\models h \text{ for all } o^- \in O^- \end{aligned} \tag{22}$$

Learning from Conjunctions (cont'd)

If none of $o \in O^+ \cup O^-$ is contradictory, then

$$h = \mathrm{lgg}(O^+) = \bigcap_{o^+ \in O^+} o^+ \tag{23}$$

is also a non-contradictory conjunction and $\models$ ($\not\models$) in (22) is equivalent to $\subseteq$ ($\not\subseteq$), so h is a solution to (22) if a solution exists.

We can compute (23) from all positive examples in the training set (19) when working in the batch mode. In the online setting, the agent follows (14) ($\bar{o}$ denoting the conjunctive observations). Here h_1 would be a conjunction of all $2n$ literals. However, there is no need to build h_1: just make negative predictions ($a_k = 0$) until some example o_k is misclassified ($r_{k+1} = -1$), and then set $h_{k+1} = o_k$.

Learning from Conjunctions: Example

This learning setting is useful: it allows to learn from *incomplete* conjunctive observations. For example, consider learning the *wife* concept from three examples of persons.

$$
\begin{aligned}
o_1^+ &= \texttt{woman} \wedge \texttt{happy} \wedge \texttt{married} \wedge \neg\texttt{rich} \\
o_2^+ &= \texttt{tall} \wedge \texttt{woman} \wedge \texttt{married} \\
o_3^- &= \neg\texttt{tall} \wedge \texttt{woman}
\end{aligned}
$$

The observations are *incomplete* in that e.g. we do not know if first person is tall or if the other two are rich. Still, the solution

$$h = \mathrm{lgg}(o_1^+, o_2^+) = \texttt{woman} \wedge \texttt{married}$$

is as expected, and consistent with the negative example: $o_3^- \not\models h$.

Learning from Disjunctions

Observe (verify, easy) that for conjunctions O^+, (23) is equivalent to

$$\neg h = \mathrm{lgg}(\{\, \neg o^+ ; o^+ \in O^+ \,\})$$

where $\neg h$ and $\neg o^+$ are all *disjunctions*. Since $o^+ \models h$ for $o^+ \in O^+$ due to (22), we have $\neg h \models \neg o^+$ for $o^+ \in O^+$.

Assume now the observations are already *disjunctions*, which are not tautologies. Then (23) is a disjunction satisfying

$$\begin{aligned} h &\models o^+ \text{ for all } o^+ \in O^+ \\ h &\not\models o^- \text{ for all } o^- \in O^- \end{aligned} \tag{24}$$

whenever a disjunction satisfying (24) exists. We will now illustrate why formulation (24) is also useful.

Learning from Disjunctions: Example

Disjunctions are convenient in machine learning because they can be easily rewritten into implications and interpreted as *rules*. E.g. $a \vee b \vee \neg c \vee \neg d$ is tautologically equivalent to $a \wedge b \rightarrow c \vee b$.

For example, from disjunctive examples

$$o_1^+ = \mathtt{man} \wedge \mathtt{adult} \wedge \mathtt{young} \rightarrow \mathtt{married} \vee \mathtt{bachelor}$$
$$o_2^+ = \mathtt{man} \wedge \mathtt{adult} \rightarrow \mathtt{married} \vee \mathtt{bachelor} \vee \mathtt{nerd}$$

the agent can learn $h = \mathrm{lgg}(o_1^+, o_2^+) = o_1^+ \cap o_2^+ =$

$$\mathtt{man} \wedge \mathtt{adult} \rightarrow \mathtt{married} \vee \mathtt{bachelor}$$

So the agent learns the correct rule from observed rules which are true but insufficiently general.

Learning from Conjunctions or Disjunctions: Summary

$h = \mathrm{lgg}(O^+)$ is a solution to (22) when observations are non-contradictory conjunctions and to (24) when observations are non-tautological disjunctions. Using the $\subseteq$ relation, the task can be formulated jointly.

Learning from Subsumption

Given sets O^+, O^- of non-self-resolving conjunctions (disjunctions) find a conjunction (disjunction, respectively) h such that

$$\begin{aligned} h &\subseteq o^+ \text{ for all } o^+ \in O^+ \\ h &\not\subseteq o^- \text{ for all } o^- \in O^- \end{aligned} \tag{25}$$

If a solution exists, then $h = \mathrm{lgg}(O^+)$ is a solution. Furthermore, as we showed here, one can also obtain a conjunctive hypothesis for disjunctive observations (or vice versa) as $h = \neg\mathrm{lgg}(O^-)$.

Example

Solutions of (25) with

$$\begin{aligned} o_1^+ &= \mathrm{p}_1 \wedge \mathrm{p}_3 \wedge \mathrm{p}_4 & o_3^- &= \mathrm{p}_1 \wedge \mathrm{p}_2 \wedge \mathrm{p}_5 \\ o_2^+ &= \mathrm{p}_3 \wedge \mathrm{p}_4 \wedge \mathrm{p}_6 & o_4^- &= \mathrm{p}_2 \wedge \mathrm{p}_5 \wedge \mathrm{p}_6 \end{aligned}$$

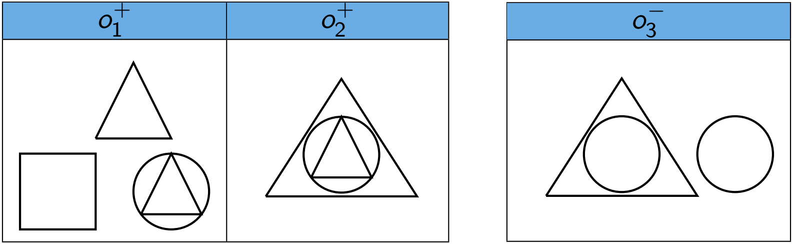
- Conjunctive: $h = \mathrm{lgg}(o_1^+, o_2^+) = \neg\mathrm{lgg}(\neg o_1^+, \neg o_2^+) = \mathrm{p}_3 \wedge \mathrm{p}_4$. (Check consistency with o_3^-, o_4^-!)
- Disjunctive: $h = \neg\mathrm{lgg}(o_3^-, o_4^-) = \mathrm{lgg}(\neg o_3^-, \neg o_4^-) = \mathrm{p}_2 \vee \mathrm{p}_5$. (Check consistency with o_1^+, o_2^+!)

An analogical example can be shown for disjunctive examples: just swap all $\wedge$'s and $\vee$'s above.

While the example set above allowed both a conjunctive and a disjunctive solution, other example sets may allow only one of them or none at all. (lgg of all examples of one class is always defined but the result may not be consistent with the examples of the other class.)

Learning Relational Concepts

Structured Observations



What distinguishes the positive examples from the negative one in terms of shapes and inclusions?

Such examples are abstractions of real-life relational structures (molecules, social networks, GIS objects, ...). There is no obvious way to encode them through propositional-logic formulas. We need a more expressive language.

Relational Logic Language

Relational logic is a subset of first-order logic. The alphabet of a relational logic language consists of two finite sets

Predicates: each with defined arity, e.g. `circle`/1, or `inside`/2

Constants: e.g. $\mathrm{object1}, \mathrm{object2}$

and a countable set of *variables* (e.g. $x, y, z, x_1, \ldots$). Constants and variables are together called *terms*.

Using predicates and terms, we can express simple statements such as

$$\texttt{circle}(\mathrm{object1}) \text{ or } \texttt{inside}(\mathrm{object1}, \mathrm{object2})$$

which are called *atoms* and are analogical to propositional 'variables' such as $\mathrm{p}_1, \mathrm{p}_2$.

Relational Logic Formulas

As in propositional logic, we can use *connectives* ($\wedge, \vee, \rightarrow$) between *literals*, which are atoms or their negations, to create more complex formulas, e.g.

$$\neg\texttt{circle}(\mathrm{object1}) \wedge \texttt{inside}(\mathrm{object1}, \mathrm{object2})$$

Unlike in propositional logic, we can use quantified *variables* in formulas

$$\exists \mathsf{x}\ \neg\texttt{circle}(\mathsf{x}) \wedge \texttt{inside}(\mathsf{x}, \mathrm{object2})$$
$$\forall \mathsf{x}\ \texttt{circle}(\mathsf{x}) \vee \neg\texttt{inside}(\mathsf{x}, \mathrm{object2})$$

In this course, if quantifiers are not shown, all variables in relational *conjunctions are quantified existentially*, and all variables in relational *disjunctions are quantified universally*.

Formulas without variables are called *ground*.

Substitution

A *substitution* replaces all occurrences of specified variables with specified terms. E.g. with formula

$$\varphi = \texttt{circle}(x) \land \texttt{inside}(x, y)$$

and substitutions

$$\theta_1 = \{\, x \to \text{object1}, y \to \text{object2} \,\}$$
$$\theta_2 = \{\, x \to y \,\}$$

we have

$$\begin{aligned} \varphi\theta_1 = \varphi\theta_1\theta_2 &= \texttt{circle}(\text{object1}) \land \texttt{inside}(\text{object1}, \text{object2}) \\ \varphi\theta_2 &= \texttt{circle}(y) \land \texttt{inside}(y, y) \\ \varphi\theta_2\theta_1 \ &= \texttt{circle}(\text{object2}) \land \texttt{inside}(\text{object2}, \text{object2}) \end{aligned}$$

FACULTY
OF ELECTRICAL
ENGINEERING
CTU IN PRAGUE

θ-Subsumption

Subsumption in relational logic is more involved than in propositional logic.

Let a, b be two relational conjunctions or disjunctions. We say that a *θ-subsumes* b (written $a \subseteq_\theta b$) if there is a substitution θ s.t. $a\theta \subseteq b$.

Example for conjunctions:

$$\begin{aligned} \texttt{inside}(x, y) \subseteq_\theta \texttt{circle}(\mathrm{c}) \wedge \texttt{inside}(x, \mathrm{c}) \\ \theta = \{\, y \mapsto \mathrm{c} \,\} \end{aligned}$$

Example for disjunctions:

$$\begin{aligned} \texttt{inside}(x, y) \wedge \texttt{inside}(y, z) \to \texttt{inside}(x, z) \\ \subseteq_\theta \texttt{inside}(x, \mathrm{object1}) \wedge \texttt{inside}(\mathrm{object1}, z) \to \texttt{inside}(x, z) \\ \theta = \{\, y \mapsto \mathrm{object1} \,\} \end{aligned}$$

θ-Subsumption: Properties

- It is NP-complete to decide whether $a \subseteq_\theta b$ for two conjunctions or disjunctions a, b.
- Note that $a \subseteq_\theta b$ may hold even if a has more literals than b, e.g

$$\mathtt{inside}(x, y) \wedge \mathtt{inside}(w, z) \subseteq_\theta \mathtt{inside}(\mathrm{object1}, \mathrm{object2})$$
$$\theta = \{\, x \mapsto \mathrm{object1}, w \mapsto \mathrm{object1}, y \mapsto \mathrm{object2}, y \mapsto \mathrm{object2} \,\}$$

- Two different conjunctions (disjunctions) a, b may be *θ-equivalent*, meaning *$a \subseteq_\theta b$ and $b \subseteq_\theta a$*. Then we write $a \approx_\theta b$. For example,

$$\mathtt{circle}(x) \approx_\theta \mathtt{circle}(x) \wedge \mathtt{circle}(y)$$

θ-Subsumption: Properties (cont'd)

- As in the propositional case, $a \subseteq_\theta b$ implies $b \models a$ ($a \models b$) for conjunctions (disjunctions) a, b, while the reverse implications do not hold. For example, with disjunctions

  $$a = \neg \mathrm{p}(x, y) \vee \neg \mathrm{p}(y, z) \vee \mathrm{p}(x, z)$$
  $$b = \neg \mathrm{p}(\mathrm{a}, \mathrm{b}) \vee \neg \mathrm{p}(\mathrm{b}, \mathrm{c}) \vee \neg \mathrm{p}(\mathrm{c}, \mathrm{d}) \vee \mathrm{p}(\mathrm{a}, \mathrm{d})$$

  $a \models b$ (verify by resolution) but $a \not\subseteq_\theta b$. This is because a is self-resolving, i.e. contains a positive and a negative literal with the same predicate.

Unlike in propositional logic, relational self-resolving disjunctions need not be tautologies (e.g. a is not a tautology) and relational self-resolving conjunctions need not be contradictions (e.g. $\neg a$ is not a contradiction).

Learning from θ-Subsumption

The learning task is as in (25) except $\subseteq$ is replaced with $\subseteq_\theta$, i.e.:

Learning from $\subseteq_\theta$-Subsumption

Given sets O^+, O^- of non-self-resolving conjunctions (disjunctions) find a conjunction (disjunction, respectively) h such that

$$\begin{aligned} h \subseteq_\theta o^+ &\text{ for all } o^+ \in O^+ \\ h \not\subseteq_\theta o^- &\text{ for all } o^- \in O^- \end{aligned} \tag{26}$$

Clearly, if a solution exists then the least general generalization (i.e. the least upper bound as defined here) of O^+ with respect to $\subseteq_\theta$ is a solution.

So if we can define lgg for the $\subseteq_\theta$ order, the relational agent can work as described in here. (While h_1 is combinatorially large, its computation can be avoided as described here.)

Relational lgg

Unlike in the propositional case, where lgg was with respect to $\subseteq$ and thus simply computable (12), relational lgg is more involved.

We first define lgg for two *literals* which are *compatible*, i.e. have the same sign (plain or negated), predicate symbol, and arity.

The lgg of $\mathrm{p}(t_1, t_2, \ldots t_a)$ and $\mathrm{p}(u_1, u_2, \ldots u_a)$ where t_i, u_i are terms (constants or variables) is

$$\mathrm{p}\left(\mathrm{lgg}(t_1, u_1), \mathrm{lgg}(t_2, u_2), \ldots \mathrm{lgg}(t_a, u_a)\right) \tag{27}$$

Relational lgg (cont'd)

The arguments in (27) are

$$\mathrm{lgg}(t_i, u_i) = \begin{cases} t_i \text{ if } t_i = u_i \\ v_i \text{ otherwise} \end{cases}$$

where v_i is

- either a new variable (different from terms $t_1, u_1, \dots t_{i-1}, t_{i-1}$) if $\langle t_i, u_i \rangle$ have not yet been matched (i.e. the pair is different from all of $\langle t_1, u_1 \rangle, \dots \langle t_{i-1}, t_{i-1} \rangle$)
- same as variable v_j if $\langle t_i, u_i \rangle$ have already been matched ($\langle t_i, u_i \rangle = \langle t_j, u_j \rangle$ for some $j < i$) and assigned variable v_j.

Example: $\mathrm{lgg}\left(\mathrm{p}\left(x, \mathrm{a}, x, \mathrm{b}, \mathrm{a}\right)\right), \mathrm{p}\left(\mathrm{a}, \mathrm{b}, \mathrm{a}, \mathrm{a}, \mathrm{a}\right)) = \mathrm{p}\left(v_1, v_2, v_1, v_3, \mathrm{a}\right)$

Anti-Unification Algorithm

The lgg of two compatible literals is computed by the *Anti-unification* algorithm, which rewrites both of the input literals a, b into $\mathrm{lgg}(a, b)$.

```
Require: Literals $a, b$ compatible with each other and not sharing a variable (if they
    share a variable, just replace all occurrences of it in $a$ with a new variable)
$i = 0$; $\theta := \emptyset$; $\sigma := \emptyset$        ▷ a counter and two substitutions
$v_1, v_2, \dots$ : variables not appearing in $a$ or $b$
while $a \neq b$ do
    Let $p$ be the leftmost position where $a$ and $b$ differ and $s$ and $t$ be the terms at
    this position in $a$ and $b$, respectively.
    if for some $j$ $(1 \leq j \leq i)$, $v_j\theta = s$ and $v_j\sigma = t$ then    ▷ variable already assigned
        put $v_j$ to position $p$ in both $a$ and $b$    ▷ replace the terms with that variable
    else
        $i := i + 1$
        put $v_i$ to position $p$ in both $a$ and $b$    ▷ replace the terms with a new variable
        $\theta := \theta \cup \{ v_i \mapsto s \}$, $\sigma := \sigma \cup \{ v_i \mapsto t \}$    ▷ store assignment of $v_i$
    end if
end while
return $a$
```

Relational lgg (cont'd)

Finally, the lgg of two conjunctions or disjunctions a, b contains the lgg of each pair $l_a \in a, l_b \in b$ of compatible literals. Importantly, when matching some terms t, u that were already assigned a variable v in the lgg of another pair of literals, they have to be assigned v again.

Example:

$$\begin{aligned} a &= \texttt{parent}(\mathrm{jack}, \mathrm{ann}) \wedge \texttt{female}(\mathrm{ann}) \rightarrow \texttt{daughter}(\mathrm{ann}) \\ b &= \texttt{parent}(\mathrm{tracy}, \mathrm{sarah}) \wedge \texttt{female}(\mathrm{sarah}) \rightarrow \texttt{daughter}(\mathrm{sarah}) \\ \mathrm{lgg}(a, b) &= \texttt{parent}(x, y) \wedge \texttt{female}(y) \rightarrow \texttt{daughter}(y) \end{aligned}$$

Gordon Plotkin showed that the lgg operator we just defined indeed computes the least upper bound with respect to the θ-subsumption order.

lgg Example: Non-Ground Disjunctions

lgg of

$$\mathtt{male}(x) \wedge \mathtt{female}(y) \wedge \mathtt{parent}(x, y) \rightarrow \mathtt{daughter}(y, x)$$

and

$$\mathtt{female}(x) \wedge \mathtt{parent}(\mathrm{ann}, x) \rightarrow \mathtt{daughter}(x, \mathrm{ann})$$

	$\neg\mathtt{female}(x)$	$\neg\mathtt{parent}(\mathrm{ann}, x)$	$\mathtt{daughter}(x, \mathrm{ann})$
$\neg\mathtt{male}(x)$			
$\neg\mathtt{female}(y)$	$\neg\mathtt{female}(v_1)$		
$\neg\mathtt{parent}(x, y)$		$\neg\mathtt{parent}(v_2, v_1)$	
$\mathtt{daughter}(y, x)$			$\mathtt{daughter}(v_1, v_2)$

θ	σ	
y	x	v_1
x	ann	v_2

is

$$\mathtt{female}(x) \wedge \mathtt{parent}(y, x) \rightarrow \mathtt{daughter}(x, y)$$

lgg Example: Non-Ground Disjunctions (cont'd)

$\texttt{male}(x) \wedge \texttt{female}(y) \wedge \texttt{parent}(x, y) \rightarrow \texttt{daughter}(y, x)$

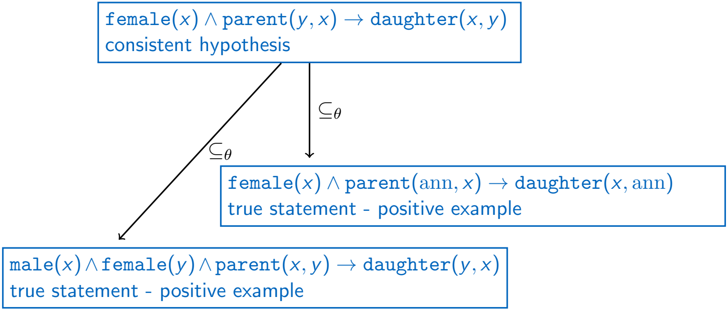
true statement - positive example

lgg Example: Non-Ground Disjunctions (cont'd)

$\texttt{female}(x) \land \texttt{parent}(\mathrm{ann}, x) \rightarrow \texttt{daughter}(x, \mathrm{ann})$
true statement - positive example

$\texttt{male}(x) \land \texttt{female}(y) \land \texttt{parent}(x, y) \rightarrow \texttt{daughter}(y, x)$
true statement - positive example

lgg Example: Non-Ground Disjunctions (cont'd)



$\mathtt{female}(x) \land \mathtt{parent}(y, x) \rightarrow \mathtt{daughter}(x, y)$

consistent hypothesis

$\subseteq_\theta$

$\subseteq_\theta$

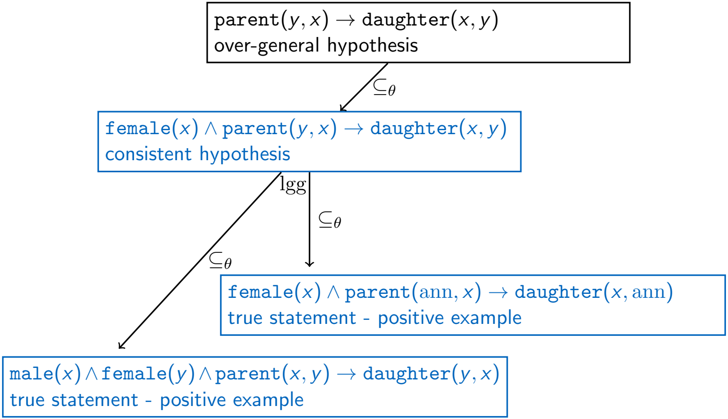
$\mathtt{female}(x) \land \mathtt{parent}(\mathrm{ann}, x) \rightarrow \mathtt{daughter}(x, \mathrm{ann})$

true statement - positive example

$\mathtt{male}(x) \land \mathtt{female}(y) \land \mathtt{parent}(x, y) \rightarrow \mathtt{daughter}(y, x)$

true statement - positive example

lgg Example: Non-Ground Disjunctions (cont'd)



$\texttt{parent}(y, x) \rightarrow \texttt{daughter}(x, y)$
over-general hypothesis

$\subseteq_\theta$

$\texttt{female}(x) \wedge \texttt{parent}(y, x) \rightarrow \texttt{daughter}(x, y)$
consistent hypothesis

lgg

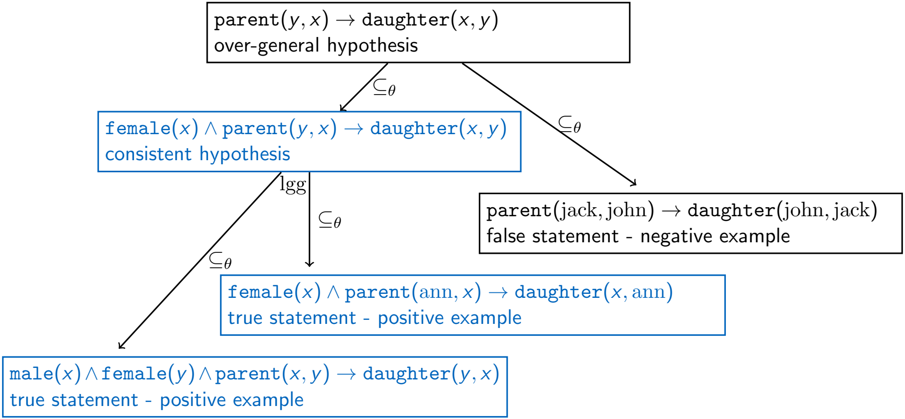
$\subseteq_\theta$

$\subseteq_\theta$

$\texttt{female}(x) \wedge \texttt{parent}(\mathrm{ann}, x) \rightarrow \texttt{daughter}(x, \mathrm{ann})$
true statement - positive example

$\texttt{male}(x) \wedge \texttt{female}(y) \wedge \texttt{parent}(x, y) \rightarrow \texttt{daughter}(y, x)$
true statement - positive example

lgg Example: Non-Ground Disjunctions (cont'd)



$\texttt{parent}(y, x) \rightarrow \texttt{daughter}(x, y)$

over-general hypothesis

$\texttt{female}(x) \wedge \texttt{parent}(y, x) \rightarrow \texttt{daughter}(x, y)$

consistent hypothesis

$\texttt{parent}(\text{jack}, \text{john}) \rightarrow \texttt{daughter}(\text{john}, \text{jack})$

false statement - negative example

$\texttt{female}(x) \wedge \texttt{parent}(\text{ann}, x) \rightarrow \texttt{daughter}(x, \text{ann})$

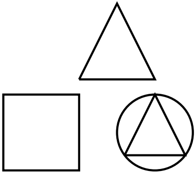
true statement - positive example

$\texttt{male}(x) \wedge \texttt{female}(y) \wedge \texttt{parent}(x, y) \rightarrow \texttt{daughter}(y, x)$

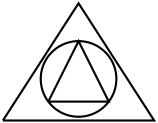
true statement - positive example

lgg Example: Conjunctions

We return to the initial example and encode the observations as relational conjunctions.



$o_1^+ =$
$\mathtt{triangle}(\mathrm{t}_1) \wedge \mathtt{rectangle}(\mathrm{r}_1) \wedge$
$\mathtt{circle}(\mathrm{c}_1) \wedge \mathtt{triangle}(\mathrm{u}_1) \wedge$
$\mathtt{inside}(\mathrm{u}_1, \mathrm{c}_1)$



$o_2^+ =$
$\mathtt{triangle}(\mathrm{t}_2) \wedge \mathtt{circle}(\mathrm{c}_2) \wedge$
$\mathtt{triangle}(\mathrm{u}_2) \wedge \mathtt{inside}(\mathrm{u}_2, \mathrm{c}_2) \wedge$
$\mathtt{inside}(\mathrm{c}_2, \mathrm{t}_2) \wedge \mathtt{inside}(\mathrm{u}_2, \mathrm{t}_2)$

lgg Example: Conjunctions (cont'd)

Abbreviating predicate symbols

$o_2^+ \Downarrow o_1^+ \Rightarrow$	$\mathtt{t}(\mathrm{t}_1)$	$\mathtt{r}(\mathrm{r}_1)$	$\mathtt{c}(\mathrm{c}_1)$	$\mathtt{t}(\mathrm{u}_1)$	$\mathtt{in}(\mathrm{u}_1, \mathrm{c}_1)$
$\mathtt{t}(\mathrm{t}_2)$	$\mathtt{t}(x_1)$			$\mathtt{t}(x_4)$	
$\mathtt{c}(\mathrm{c}_2)$			$\mathtt{c}(x_3)$		
$\mathtt{t}(\mathrm{u}_2)$	$\mathtt{t}(x_2)$			$\mathtt{t}(x_5)$	
$\mathtt{in}(\mathrm{u}_2, \mathrm{c}_2)$					$\mathtt{in}(x_5, x_3)$
$\mathtt{in}(\mathrm{c}_2, \mathrm{t}_2)$					$\mathtt{in}(x_6, x_7)$
$\mathtt{in}(\mathrm{u}_2, \mathrm{t}_2)$					$\mathtt{in}(x_5, x_7)$

θ	σ	
t_1	t_2	x_1
t_1	u_2	x_2
c_1	c_2	x_3
u_1	t_2	x_4
u_1	u_2	x_5
u_1	u_2	x_6
c_1	t_2	x_7

$$\begin{aligned}\mathrm{lgg}(o_1^+, o_2^+) = & \mathtt{t}(x_1) \wedge \mathtt{t}(x_2) \wedge \mathtt{c}(x_3) \wedge \mathtt{t}(x_4) \wedge \mathtt{t}(x_5) \\ & \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)\end{aligned}$$

lgg Example: Conjunctions (cont'd)

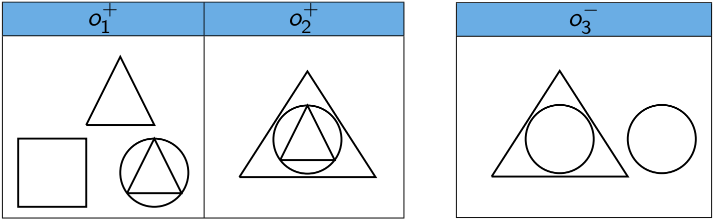
$\text{lgg}(o_1^+, o_2^+)$ is a complex conjunction but it is θ-equivalent to a simpler one under substitution $\theta = \{\, x_1 \mapsto x_5, x_2 \mapsto x_5, x_4 \mapsto x_5, x_6 \mapsto x_5, x_7 \mapsto x_3 \,\}$:

$$\mathtt{t}(x_1) \wedge \mathtt{t}(x_2) \wedge \mathtt{c}(x_3) \wedge \mathtt{t}(x_4) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$$

$$\approx_\theta$$

$$\mathtt{c}(x_3) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3)$$

meaning "a triangle in a circle". This is indeed a correct pattern consistent with the observations. Hurray!



Clause Reduction

For brevity, the following definitions and examples speak only of universally quantified disjunctions, i.e., *clauses*. However, they apply analogically to existentially quantified conjunctions (the next page provides an example).

Clause Reduction

Clause a is *reduced* if for no clause b, $b \subset a$, $b \approx_\theta a$. A reduced clause b is a *reduction* of clause a if $b \approx_\theta a$.

Reduction principle: *Given clause a, find a literal $l \in a$ such that $a \subseteq_\theta a \setminus \{ l \}$. If found, set $a \leftarrow a\theta$ and repeat; else return a.*

Remind that deciding the relation $\subseteq_\theta$ is NP-complete, so a clause reduction cannot be computed efficiently.

Conjunction Reduction: Example

1. $h =$
   $\mathtt{t}(x_1) \wedge \mathtt{t}(x_2) \wedge \mathtt{c}(x_3) \wedge \mathtt{t}(x_4) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$
   $\theta = \{ x_1 \mapsto x_2 \}$, $h \subseteq_\theta h \setminus \{ \mathtt{t}(x_1) \}$, so set $h \leftarrow h\theta$.
2. $h = \mathtt{t}(x_2) \wedge \mathtt{c}(x_3) \wedge \mathtt{t}(x_4) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$
   $\theta = \{ x_2 \mapsto x_4 \}$, $h \subseteq_\theta h \setminus \{ \mathtt{t}(x_2) \}$, so set $h \leftarrow h\theta$.
3. $h = \mathtt{c}(x_3) \wedge \mathtt{t}(x_4) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$
   $\theta = \{ x_4 \mapsto x_5 \}$, $h \subseteq_\theta h \setminus \{ \mathtt{t}(x_4) \}$, so set $h \leftarrow h\theta$.
4. $h = \mathtt{c}(x_3) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$
   $\theta = \{ x_6 \mapsto x_5, x_7 \mapsto x_3 \}$, $h \subseteq_\theta h \setminus \{ \mathtt{in}(x_6, x_7) \}$, so set $h \leftarrow h\theta$
5. $h = \mathtt{c}(x_3) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3) \wedge \mathtt{in}(x_6, x_7) \wedge \mathtt{in}(x_5, x_7)$
   $\theta = \{ x_6 \mapsto x_5, x_7 \mapsto x_3 \}$, $h \subseteq_\theta h \setminus \{ \mathtt{in}(x_6, x_7) \}$, so set $h \leftarrow h\theta$
6. $h = \mathtt{c}(x_3) \wedge \mathtt{t}(x_5) \wedge \mathtt{in}(x_5, x_3)$. h is reduced.

Learnability of Relational Clauses

The proof from here, which applied only to the propositional-logic agent can easily be extended to any lgg based agent, including the relational one. Let h^* be the target clause.

1. $h^* \subseteq_\theta o_k$ for all positive observations o_k, so $h^* \subseteq_\theta h_1 = o_1$.
2. If h_{k-1} misclassifies a negative example o_{k-1}, i.e. $h_{k-1} \subseteq_\theta o_{k-1}$, then $h_k = \mathrm{lgg}(h_{k-1}, o_{k-1}) \approx_\theta h_{k-1}$, so $h^* \subseteq_\theta h_{k-1}$ *implies* $h^* \subseteq_\theta h_k$.
3. If h_{k-1} misclassifies a positive example o_{k-1}, i.e. $h_{k-1} \not\subseteq_\theta o_{k-1}$, then $h^* \subseteq_\theta h_{k-1}$ *implies* $h^* \subseteq_\theta h_k = \mathrm{lgg}(h_{k-1}, o_{k-1})$, because $h^* \subseteq_\theta o_{k-1}$ and h_k is a least generalization
4. By induction from the blue assertions above: $h^* \subseteq_\theta h_k, \ \forall k \in N$.
5. From 4: mistakes are made only on positive examples and after each mistake, $h_k \subset_\theta h_{k-1}$ (because $h_{k-1} \not\subseteq_\theta o_{k-1}$ and $h_k \subseteq_\theta o_{k-1}$).

Learnability of Relational Clauses (cont'd)

We have shown the agent makes a strict generalization after each mistake, and never over-generalizes. Enough for an error bound in the mistake-bound model?

Consider the infinite sequence of clauses a_k $(k = 2, 3, \ldots)$:

$$a_k = \bigvee_{1 \le i,j \le k, i \ne j} \mathrm{p}(x_i, x_j)$$

$$a_2 = \mathrm{p}(x_1, x_2) \vee \mathrm{p}(x_2, x_1)$$
$$a_3 = \mathrm{p}(x_1, x_2) \vee \mathrm{p}(x_2, x_1) \vee \mathrm{p}(x_1, x_3) \vee \mathrm{p}(x_3, x_1) \vee \mathrm{p}(x_2, x_3) \vee \mathrm{p}(x_3, x_2)$$

etc.

Observe that

$$\mathrm{p}(x_1, x_2) \subseteq_\theta a_2 \subset_\theta a_3 \subset_\theta \ldots \subseteq_\theta \mathrm{p}(x_1, x_1)$$

.

Learnability of Relational Clauses (cont'd)

So in the subsumption lattice, two finite clauses such as $\mathrm{p}(x_1, x_1)$ and $\mathrm{p}(x_1, x_2)$ are connected by an *infinite path of strict generalizations*.

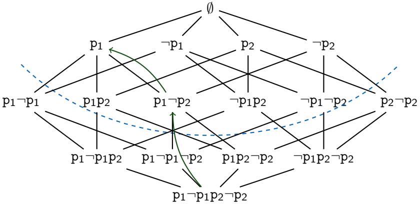
Thus for any $M \in N$, the sequence $o_1 = a_{M+1}, o_2 = a_M, \ldots, o_{M+1} = a_1$ of positive examples causes $M + 1$ errors. Therefore, we *cannot establish a finite error bound in the mistake-bound model* for lgg-based learning of relational clauses ☹.

Observe that in the propositional generalization agent, we had $h_{k+1} \subset h_k$ after each error. Thus infinite chains of generalizations were not possible as eventually the hypothesis will be empty. But in the relational agent, we only have a *weaker* guarantee: $h_{k+1} \subset_\theta h_k$ wherein h_{k+1} may have *more* literals than h_k.

Learnability of Relational Clauses (cont'd)

Lattices with respect to the $\subseteq_\theta$ order.

Propositional



Finite

Relational

Infinite, infinitely dense

"Infinitely dense" $=$ infinite-length paths between finite elements as exemplified on the previous page.

Learnability of Relational Clauses (cont'd)

We have failed to establish a bound in the mistake-bound model. Not surprisingly.

In literature, unconstrained relational disjunctions or conjunctions were proven non-learnable in the PAC model, and thus they are neither learnable in the mistake-bound model.

This is in contrast to propositional disjunctions or conjunctions, which (remind) are learnable in both settings.

However, learnability of relational formulas even beyond conjunctions and disjunctions can be achieved by imposing size bounds.

Learnability of Relational *st*-Clauses

Consider $\mathcal{H} =$ non-self-resolving *st-clauses*, i.e. clauses with no more than s literals and no more than t occurrences of constant and variable symbols in each literal. For example,

$$\texttt{father}(x, y) \rightarrow \texttt{parent}(x, y)$$

is a (non-self-resolving) $2, 2$-clause.

Define the learning task size as $\langle n, |P|, |C| \rangle$, where

- n is the number of literals in the largest observation (analogical to the dimension n in the propositional case)
- $|P|$ ($|C|$)is the number of predicates (constants) in the used language.

We will now calculate $|\mathcal{H}|$.

Learnability of Relational *st*-Clauses (cont'd)

An *st*-clause has no more that *st* different variables. So the maximum number of different terms in an *st*-clause is $|C| + st$.

An atom consists of a single predicate ($|P|$ different choices) and at most t terms (each from from $|C| + st$ choices). So there are $|P|(|C| + st)^t$ different atoms, i.e. $2|P|(|C| + st)^t$ different literals.

An *st*-clause combines at most s literals so there are at most

$$\mathcal{O}\binom{2|P|(|C| + st)^t}{s} = poly(|P|, |C|) = poly(n, |P|, |C|)$$

different *st*-clauses.

Learnability of Relational *st*-Clauses (cont'd)

Since $|\mathcal{H}|$ is polynomial, it is *properly learnable* online in the mistake-bound model e.g. with the version-space agent using $\mathcal{H}$. It is also PAC-learnable e.g. with the consistent agent using $\mathcal{H}$. However, we *cannot prove efficient learnability* in either case, as the consistency check $h \subseteq_\theta o$ is NP-complete, taking time exponential in n.

Alternatively, $\mathcal{H}$ can be learned with the generalization agent producing $h = \mathrm{lgg}\,(O^+)$. Since lgg is a poly-time procedure, $\mathcal{H}$ is *learnable efficiently* (both online and PAC) but *not properly* because h need not be an *st*-clause even though all of $o^+ \in O^+$ are.

Note: The dilemma btw. efficient and proper learning is the same as with $\mathcal{H} =$ propositional s-term DNF's. Efficient proper learning of *st*-clauses can be achieved with a further syntactical restriction, in particular, that observations o are non-self-resolving *ground, range-restricted* *st*-clauses, as then $h \subseteq_\theta o$ can be checked efficiently. A clause is range-restricted if each variable in a positive literal also occurs in some negative literal.

Learnability of *st*-CNF's

Let a_i $(1 \leq i \leq n)$ and o be non-self-resolving clauses and

$$h = \bigwedge_{1 \leq i \leq n} a_i$$

i.e., h is a CNF. Clearly, $h \models o$ iff $a_i \subseteq_\theta o$ for all $1 \leq i \leq n$.

So learning a CNF from relational non-self-resolving clauses is defined as in (26), except the conditions read

$$\forall o^+ \in O^+ \; \forall a_i \in h \; a_i \subseteq_\theta o^+$$
$$\forall o^- \in O^- \; \exists a_i \in h \; a_i \not\subseteq_\theta o^-$$

Learnability of *st*-CNF's (cont'd)

Let $\mathcal{H} = st$-CNF's, i.e. conjunctions of non-self-resolving *st*-clauses.

Since there are only $poly(n, |P|, |C|)$ number of *st*-clauses, $|\mathcal{H}| = 2^{poly(n,|P|,|C|)}$, so $\ln |\mathcal{H}|$ is polynomial.

Thus *st*-CNF's are also learnable both online and PAC. In the online setting, they can be learned for example with this strategy, in which h_1 is the conjunction of all *st*-clauses.

As a trivial consequence, non-self-resolving *st-conjunctions* and *st-DNF's* are also learnable, with the same caveats we discussed for *st*-clauses and *st*-CNF's.

Background Knowledge

lgg of

$$\mathtt{female}(x) \wedge \mathtt{father}(y, x) \rightarrow \mathtt{daughter}(x, y)$$

and

$$\mathtt{female}(x) \wedge \mathtt{mother}(y, x) \rightarrow \mathtt{daughter}(x, y)$$

is

$$\mathtt{female}(x) \rightarrow \mathtt{daughter}(x, y)$$

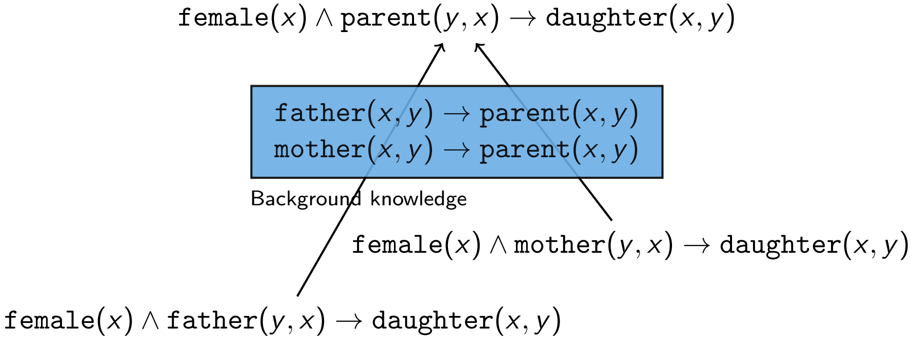
Intuitively, an over-generalization. We really want to generalize towards

$$\mathtt{female}(x) \wedge \mathtt{parent}(y, x) \rightarrow \mathtt{daughter}(x, y)$$

But $\mathtt{father}/2$ and $\mathtt{mother}/2$ are incompatible and the agent does not know about their (semantic) joint generalization $\mathtt{parent}/2$.

Background Knowledge (cont'd)

Perhaps it could?



$\texttt{female}(x) \wedge \texttt{parent}(y, x) \rightarrow \texttt{daughter}(x, y)$

$\texttt{father}(x, y) \rightarrow \texttt{parent}(x, y)$
$\texttt{mother}(x, y) \rightarrow \texttt{parent}(x, y)$

Background knowledge

$\texttt{female}(x) \wedge \texttt{mother}(y, x) \rightarrow \texttt{daughter}(x, y)$

$\texttt{female}(x) \wedge \texttt{father}(y, x) \rightarrow \texttt{daughter}(x, y)$

We aim at generalization *with respect to background knowledge*.

Background Knowledge (cont'd)

An agent equipped with background knowledge B, here

$$B = (\texttt{father}(x, y) \rightarrow \texttt{parent}(x, y)) \land (\texttt{mother}(x, y) \rightarrow \texttt{parent}(x, y))$$

should find a hypothesis h such that $B \land h \models o^+$ for all $o^+ \in O^+$, here

$$B \land h \models \texttt{female}(x) \land \texttt{father}(y, x) \rightarrow \texttt{daughter}(x, y)$$
$$B \land h \models \texttt{female}(x) \land \texttt{mother}(y, x) \rightarrow \texttt{daughter}(x, y)$$

and $B \land h \not\models o^-$ for all $o^- \in O^-$.

To find a solution through lgg, we need a subsumption relation $\subseteq_\theta^B$ such that for (non-self-resolving) clauses a, b, $a \subseteq_\theta^B b$ coincides with $B \land a \models b$.

Relative θ-Subsumption

This is possible only for a limited class of background knowledge, which does *not* include the previous example.

Let a, b be clauses and B a conjunction of *ground* relational *literals*. We say that a *θ-subsumes* b *relative to* B (written $a \subseteq_\theta^B b$) if $a \subseteq_\theta b \vee \neg B$. (The $\subset_\theta^B$ and $\approx_\theta^B$ relations are then defined in the obvious way.)

Verify that $b \vee \neg B$ is a clause so the definition makes sense.

Main property: $a \subseteq_\theta^B b$ implies $B \wedge a \models b$. Furthermore, for non-self-resolving a, b, the two relations are equivalent.

Relative Subsumption: Example

$\mathtt{m}/1, \mathtt{f}/1, \mathtt{p}/2, \mathtt{d}/2 \approx$ male, female, parent, daughter

Assume

$$B = \mathtt{m}(\mathrm{axel}) \wedge \mathtt{p}(\mathrm{axel}, \mathrm{brenda}) \wedge \mathtt{f}(\mathrm{brenda}) \wedge \mathtt{p}(\mathrm{brenda}, \mathrm{clara}) \wedge \mathtt{f}(\mathrm{clara})$$

Show that

$$\boxed{\mathtt{f}(y) \wedge \mathtt{p}(x, y) \rightarrow \mathtt{d}(y, x)} \subseteq_{\theta}^{B} \boxed{\mathtt{d}(\mathrm{brenda}, \mathrm{axel})}$$

$$\begin{aligned} \mathtt{f}(y) \wedge \mathtt{p}(x, y) \rightarrow \mathtt{d}(y, x) \;&\mathrel{|}\!\!\!=\!\!\!\mathrel{|} \neg\mathtt{f}(y) \vee \neg\mathtt{p}(x, y) \vee \mathtt{d}(y, x) \\ \mathtt{d}(\mathrm{brenda}, \mathrm{axel}) \vee \neg B \;&\mathrel{|}\!\!\!=\!\!\!\mathrel{|}\; \mathtt{d}(\mathrm{brenda}, \mathrm{axel}) \vee \neg\mathtt{m}(\mathrm{axel}) \vee \neg\mathtt{p}(\mathrm{axel}, \mathrm{brenda}) \\ &\qquad \vee \neg\mathtt{f}(\mathrm{brenda}) \vee \neg\mathtt{p}(\mathrm{brenda}, \mathrm{clara}) \neg\mathtt{f}(\mathrm{clara}) \end{aligned}$$

Relative Subsumption: Example (cont'd)

With

$$\theta = \{\, x \mapsto \text{axel}, y \mapsto \text{brenda} \,\}$$

we have

$$\begin{gathered} \{\, \neg\mathtt{f}(\text{brenda}), \neg\mathtt{p}(\text{axel}, \text{brenda}), \mathtt{d}(\text{brenda}, \text{axel}) \,\} \\ \subseteq \\ \left\{ \begin{array}{c} \mathtt{d}(\text{brenda}, \text{axel}), \neg\mathtt{m}(\text{axel}), \neg\mathtt{p}(\text{axel}, \text{brenda}), \\ \neg\mathtt{f}(\text{brenda}), \neg\mathtt{p}(\text{brenda}, \text{clara}), \neg\mathtt{f}(\text{clara}) \end{array} \right\} \end{gathered}$$

So indeed

$$\mathtt{f}(y) \wedge \mathtt{p}(x, y) \rightarrow \mathtt{d}(y, x) \subseteq_{\theta}^{B} \mathtt{d}(\text{brenda}, \text{axel})$$

Learning from Relative θ-Subsumption

The learning task is as in (26) except $\subseteq_\theta$ is replaced with $\subseteq_\theta^B$, i.e.:

Learning from Relative $\subseteq_\theta$-Subsumption

Given sets O^+, O^- of non-self-resolving relational clauses and a conjunction B of ground relational literals, find a clause h such that

$$\begin{aligned} h \subseteq_\theta^B o^+ \text{ for all } o^+ \in O^+ \\ h \not\subseteq_\theta^B o^- \text{ for all } o^- \in O^- \end{aligned} \tag{28}$$

Clearly, if a solution exists then $\mathrm{rlgg}_B(O^+) = \mathrm{lgg}(\{\ o^+ \vee \neg B; o^+ \in O^+\ \})$ is a solution. rlgg is called the *relative least general generalization*.

rlgg Example

$$B = \mathtt{m}(\text{axel}) \wedge \mathtt{p}(\text{axel}, \text{brenda}) \wedge \mathtt{f}(\text{brenda}) \wedge \mathtt{p}(\text{brenda}, \text{clara}) \wedge \mathtt{f}(\text{clara})$$

$$\begin{aligned} o_1 &= \mathtt{d}(\text{brenda}, \text{axel}) \\ o_2 &= \mathtt{d}(\text{clara}, \text{brenda}) \end{aligned}$$

$$\text{rlgg}_B(o_1, o_2) = \text{lgg}(o_1 \vee \neg B, o_2 \vee \neg B)$$

$$\begin{aligned} o_1 \vee \neg B = {} & \mathtt{d}(\text{brenda}, \text{axel}) \vee \neg\mathtt{m}(\text{axel}) \vee \neg\mathtt{p}(\text{axel}, \text{brenda}) \\ & \vee \neg\mathtt{f}(\text{brenda}) \vee \neg\mathtt{p}(\text{brenda}, \text{clara}) \vee \neg\mathtt{f}(\text{clara}) \\ o_2 \vee \neg B = {} & \mathtt{d}(\text{clara}, \text{brenda}) \vee \neg\mathtt{m}(\text{axel}) \vee \neg\mathtt{p}(\text{axel}, \text{brenda}) \\ & \vee \neg\mathtt{f}(\text{brenda}) \vee \neg\mathtt{p}(\text{brenda}, \text{clara}) \vee \neg\mathtt{f}(\text{clara}) \end{aligned}$$

rlgg Example (cont’d)

a, b, c ≈ axel, brenda, clara

	$\mathtt{d}(\mathrm{c}, \mathrm{b})$	$\neg\mathtt{m}(\mathrm{a})$	$\neg\mathtt{p}(\mathrm{a}, \mathrm{b})$	$\neg\mathtt{f}(\mathrm{b})$	$\neg\mathtt{p}(\mathrm{b}, \mathrm{c})$	$\neg\mathtt{f}(\mathrm{c})$
$\mathtt{d}(\mathrm{b}, \mathrm{a})$	$\mathtt{d}(v_1, v_2)$					
$\neg\mathtt{m}(\mathrm{a})$		$\neg\mathtt{m}(\mathrm{a})$				
$\neg\mathtt{p}(\mathrm{a}, \mathrm{b})$			$\neg\mathtt{p}(\mathrm{a}, \mathrm{b})$		$\neg\mathtt{p}(v_2, v_1)$	
$\neg\mathtt{f}(\mathrm{b})$				$\neg\mathtt{f}(\mathrm{b})$		$\neg\mathtt{f}(v_1)$
$\neg\mathtt{p}(\mathrm{b}, \mathrm{c})$			$\neg\mathtt{p}(v_3, v_4)$		$\neg\mathtt{p}(\mathrm{b}, \mathrm{c})$	
$\neg\mathtt{f}(\mathrm{c})$				$\neg\mathtt{f}(v_4)$		$\neg\mathtt{f}(\mathrm{c})$

θ	σ	new variable
b	c	v_1
a	b	v_2
b	a	v_3
c	b	v_4

$$h = \mathrm{rlgg}_B(o_1, o_2) = \mathtt{d}(v_1, v_2) \leftarrow \mathtt{m}(\mathrm{a}) \wedge \mathtt{p}(\mathrm{a}, \mathrm{b}) \wedge \mathtt{p}(v_2, v_1) \wedge \mathtt{f}(\mathrm{b}) \wedge$$
$$\mathtt{f}(v_1) \wedge \mathtt{p}(v_3, v_4) \wedge \mathtt{p}(\mathrm{b}, \mathrm{c}) \wedge \mathtt{f}(v_4) \wedge \mathtt{f}(\mathrm{c})$$

This is θ-equivalent (relative to B) to

$$h' = \mathtt{d}(v_1, v_2) \leftarrow \mathtt{p}(v_2, v_1) \wedge \mathtt{f}(v_1)$$

rlgg Example (cont'd)

Show $h \approx^{B}_{\theta} h'$:

- $h' \subseteq^{B}_{\theta} h$ because $h' \subseteq_{\theta} h$ because $h' \subseteq h$. Indeed:

$$\{\, \mathtt{d}(v_1, v_2), \neg\mathtt{p}(v_2, v_1), \neg\mathtt{f}(v_1) \,\} \subseteq \left\{ \begin{array}{l} \mathtt{d}(v_1, v_2), \neg\mathtt{m}(\mathrm{a}), \neg\mathtt{p}(\mathrm{a}, \mathrm{b}), \neg\mathtt{p}(v_2, v_1), \neg\mathtt{f}(\mathrm{b}), \\ \neg\mathtt{f}(v_1), \neg\mathtt{p}(v_3, v_4), \neg\mathtt{p}(\mathrm{b}, \mathrm{c}), \neg\mathtt{f}(v_4), \neg\mathtt{f}(\mathrm{c}) \end{array} \right\}$$

- $h \subseteq^{B}_{\theta} h'$ because with $\theta = \{\, v_3 \mapsto v_2, v_4 \mapsto v_1 \,\}$, $h\theta \subseteq h' \vee \neg B$:

$$\left\{ \begin{array}{l} \mathtt{d}(v_1, v_2), \neg\mathtt{m}(\mathrm{a}), \neg\mathtt{p}(\mathrm{a}, \mathrm{b}), \\ \neg\mathtt{p}(v_2, v_1), \neg\mathtt{f}(\mathrm{b}), \neg\mathtt{f}(v_1), \\ \cancel{\neg\mathtt{p}(v_2, v_1)}, \neg\mathtt{p}(\mathrm{b}, \mathrm{c}), \cancel{\neg\mathtt{f}(v_1)}, \neg\mathtt{f}(\mathrm{c}) \end{array} \right\} \subseteq \left\{ \begin{array}{l} \mathtt{d}(v_1, v_2), \neg\mathtt{p}(v_2, v_1), \\ \neg\mathtt{f}(v_1), \neg\mathtt{m}(\mathrm{a}), \neg\mathtt{p}(\mathrm{a}, \mathrm{b}), \\ \neg\mathtt{f}(\mathrm{b}), \neg\mathtt{p}(\mathrm{b}, \mathrm{c}), \neg\mathtt{f}(\mathrm{c}) \end{array} \right\}$$

Relative Clause Reduction

Relative Clause Reduction

Clause a is *reduced relative to B* if for no clause b, $b \subset a$, $b \approx_\theta^B a$. A clause b which is reduced relative to B is a *reduction* of a *relative to B* if $b \approx_\theta^B a$.

Reduction principle: As in here where $\subseteq_\theta$ is replaced by $\subseteq_\theta^B$. (Literals from $\neg B$ can be safely removed as the first step - verify why).

- $h = \mathtt{d}(v_1, v_2) \leftarrow$ $\mathtt{m}(\mathrm{a}) \wedge \mathtt{p}(\mathrm{a}, \mathrm{b}) \wedge \mathtt{p}(v_2, v_1) \wedge \mathtt{f}(\mathrm{b}) \wedge \mathtt{f}(v_1) \wedge \mathtt{p}(v_3, v_4) \wedge \mathtt{p}(\mathrm{b}, \mathrm{c}) \wedge \mathtt{f}(v_4) \wedge \mathtt{f}(\mathrm{c})$
  Remove from h all literals from $\neg B$
- $h = \mathtt{d}(v_1, v_2) \leftarrow \mathtt{p}(v_2, v_1) \wedge \mathtt{f}(v_1) \wedge \mathtt{p}(v_3, v_4) \wedge \mathtt{f}(v_4)$
  $\theta = \{\, v_3 \mapsto v_2, v_4 \mapsto v_1 \,\}$, $h \subseteq_\theta^B h \setminus \{\, \neg\mathtt{p}(v_3, v_4), \neg\mathtt{f}(v_4) \,\}$, set h to $h\theta$
- $h = \mathtt{d}(v_1, v_2) \leftarrow \mathtt{p}(v_2, v_1) \wedge \mathtt{f}(v_1)$ is reduced relative to B.

First-Order Language and lgg

First-order logic (FOL) is a superset of relational logic. It has one more *term type* in addition to constants and variables: *functions*.

A function has a symbol ('functor'), e.g. f, and arity $a \geq 1$. A function term consists of a functor and a terms in argument places, e.g. with $a = 3$

$$\mathrm{f}(x, \mathrm{a}, y)$$

Substitution and θ-Subsumption in FOL are defined as in relational logic.

For two FOL literals, the anti-unification algorithm yields their lgg as exemplified on the next page. Observe that for functions with same functor and arity, we take the lgg of the terms in corresponding arguments. So for functions $\mathrm{f}(t_1, t_2, t_3)$, $\mathrm{f}(u_1, u_2, u_3)$, we get $\mathrm{f}(\mathrm{lgg}(t_1, u_1), \mathrm{lgg}(t_2, u_2), \mathrm{lgg}(t_3, u_3))$.

Anti-Unification of FOL Literals: Example

i	atom a	θ	atom b	σ
0	$\mathrm{p}(x, \mathrm{f}(\mathrm{a}, \mathrm{b}, \mathrm{g}(\mathrm{b}, \mathrm{a})), \mathrm{h}(\mathrm{a}))$	$\emptyset$	$\mathrm{p}(y, \mathrm{f}(\mathrm{b}, \mathrm{a}, \mathrm{g}(\mathrm{a}, \mathrm{a})), \mathrm{s}(\mathrm{a}))$	$\emptyset$
1	$\mathrm{p}(v_1, \mathrm{f}(\mathrm{a}, \mathrm{b}, \mathrm{g}(\mathrm{b}, \mathrm{a})), \mathrm{h}(\mathrm{a}))$	$\{v_1 \mapsto x\}$	$\mathrm{p}(v_1, \mathrm{f}(\mathrm{b}, \mathrm{a}, \mathrm{g}(\mathrm{a}, \mathrm{a})), \mathrm{s}(\mathrm{a}))$	$\{v_1 \mapsto y\}$
2	$\mathrm{p}(v_1, \mathrm{f}(v_2, \mathrm{b}, \mathrm{g}(\mathrm{b}, \mathrm{a})), \mathrm{h}(\mathrm{a}))$	$\{v_1 \mapsto x,$ $v_2 \mapsto \mathrm{a}\}$	$\mathrm{p}(v_1, \mathrm{f}(v_2, \mathrm{a}, \mathrm{g}(\mathrm{a}, \mathrm{a})), \mathrm{s}(\mathrm{a}))$	$\{v_1 \mapsto y,$ $v_2 \mapsto \mathrm{b}\}$
3	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(\mathrm{b}, \mathrm{a})), \mathrm{h}(\mathrm{a}))$	$\{v_1 \mapsto x,$ $v_2 \mapsto \mathrm{a},$ $v_3 \mapsto \mathrm{b}\}$	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(\mathrm{a}, \mathrm{a})), \mathrm{s}(\mathrm{a}))$	$\{v_1 \mapsto y,$ $v_2 \mapsto \mathrm{b},$ $v_3 \mapsto \mathrm{a}\}$
4	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(v_3, \mathrm{a})), \mathrm{h}(\mathrm{a}))$	$\{v_1 \mapsto x,$ $v_2 \mapsto \mathrm{a},$ $v_3 \mapsto \mathrm{b}\}$	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(v_3, \mathrm{a})), \mathrm{s}(\mathrm{a}))$	$\{v_1 \mapsto y,$ $v_2 \mapsto \mathrm{b},$ $v_3 \mapsto \mathrm{a}\}$
5	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(v_3, \mathrm{a})), v_4)$	$\{v_1 \mapsto x,$ $v_2 \mapsto \mathrm{a},$ $v_3 \mapsto \mathrm{b},$ $v_4 \mapsto \mathrm{h}(\mathrm{a}))\}$	$\mathrm{p}(v_1, \mathrm{f}(v_2, v_3, \mathrm{g}(v_3, \mathrm{a})), v_4)$	$\{v_1 \mapsto y,$ $v_2 \mapsto \mathrm{b},$ $v_3 \mapsto \mathrm{a}$ $v_4 \mapsto \mathrm{s}(\mathrm{a})\}$

First-Order Language and lgg (cont'd)

The lgg of two FOL clauses (or conjunctions) is again defined as in here.

In FOL, we can learn e.g. from clausal examples

$$o_1^+ = \mathtt{odd}(x) \rightarrow \mathtt{even}(\mathrm{sum}(x,x))$$
$$o_2^+ = \mathtt{odd}(1) \wedge \mathtt{odd}(3) \rightarrow \mathit{predeven}(\mathrm{sum}(1,3))$$

the hypothesis $h = \mathrm{lgg}(o_1^+, o_2^+) =$ (verify!)

$$\mathtt{odd}(v_1) \wedge \mathtt{odd}(v_2) \rightarrow \mathtt{even}(\mathrm{sum}(v_1, v_2))$$

Verify that *st*-clauses and *st*-CNF's remain learnable even in the FOL setting. Here, t bounds the number of occurrences of constant, variable, and function symbols; and the task size is the tuple $\langle n, |P|, |C|, |F| \rangle$ where F is the set of all functions symbols in the language.

Note that when learning conjunctions or disjunction online as described in here, we *cannot* compute h_1 that contains *all* literals of the FOL language as there is an infinity of them. So the alternative approach from here is necessary.

More on Functions

Note that using functions, we can express statements about an infinite number of objects, e.g.

$$\texttt{natural}(0) \bigwedge (\texttt{natural}(x) \rightarrow \texttt{natural}(\mathrm{successor}(x)))$$

means that all of $0, \mathrm{successor}(x), \mathrm{successor}(\mathrm{successor}(x)), \ldots$ are natural.

Through functions, terms can be aggregated into more complex terms. E.g. the succession of terms

$$[t_1, t_2, \ldots] \tag{29}$$

can be represented as

$$\mathrm{f}(t_1, \mathrm{f}(t_2, \mathrm{f}(\ldots, \mathrm{end}))) \tag{30}$$

where $\mathrm{f}/2$ (end) is an auxiliary function (constant).

Logic Program

We will call terms (30) *lists* and we will use the shorter notation (29) for them. So the empty list [] corresponds to the special constant end.

For any term t_0 and any list $t = [t_1, t_2, \ldots]$, $[t_0|t]$ is an abbreviation for $[t_0, t_1, t_2, \ldots]$.

A *Horn clause* is a FOL clause with at most one positive literal. A *logic program* is a conjunction of Horn clauses (i.e., a CNF). For example

$$\varphi = \begin{array}{l} \mathtt{member}(x, [x|y]) \wedge \\ \mathtt{member}(x, [y|z]) \leftarrow \mathtt{member}(x, z) \end{array} \tag{31}$$

Observe that e.g

$$\varphi \models \mathtt{member}(\mathrm{b}, [\mathrm{a}, \mathrm{b}, \mathrm{c}])$$

as can be shown by the resolution method.

Logic Program (cont'd)

Furthermore, given a logic program φ and a non-ground conjunction a, the resolution method computes the set of substitutions θ such that $\varphi \models a\theta$.

For example, with φ from (31) and $a = \mathtt{member}(x, [\mathrm{a}, \mathrm{b}, \mathrm{c}])$, we get the set of answer substitions

$$x \mapsto \mathrm{a}, x \mapsto \mathrm{b}, x \mapsto \mathrm{c}$$

Therefore, φ can be viewed as a declarative program which answers the input *query* a. In the present example, it outputs all elements of the list given by a.

Hence the name *logic program*.

Learning a Logic Program

Since φ (31) is a CNF of FOL clauses, we should be able to *learn* (i.e., "induce") it from examples, such as

$$o_1^+ = \texttt{member}(\mathrm{a}, [\mathrm{a}, \mathrm{b}, \mathrm{c}]) \qquad o_2^- = \texttt{member}(\mathrm{d}, [\mathrm{a}, \mathrm{b}, \mathrm{c}])$$
$$o_3^+ = \texttt{member}(\mathrm{b}, [\mathrm{a}, \mathrm{b}, \mathrm{c}]) \qquad o_4^- = \texttt{member}(\mathrm{e}, [\mathrm{a}, \mathrm{b}, \mathrm{c}])$$

etc.

i.e., find a logic program h such that

$$h \models o^+ \text{ for all } o^+ \in O^+$$
$$h \not\models o^- \text{ for all } o^- \in O^-$$

Learning a Logic Program with Backround Knowledge

Even more ambitiously, we may want to learn using *background knowledge* B. For example, given

$$B = \begin{array}{l} \texttt{append}(x, [], [x]) \wedge \\ \texttt{append}(x, [y|z], [y|w]) \leftarrow \texttt{append}(x, z, w) \end{array}$$

and

$$\begin{array}{ll} o_1^+ = \texttt{reverse}([\mathrm{a}, \mathrm{b}, \mathrm{c}], [\mathrm{c}, \mathrm{b}, \mathrm{a}]) & o_2^- = \texttt{reverse}([\mathrm{a}, \mathrm{b}, \mathrm{c}], [\mathrm{a}, \mathrm{a}]) \\ o_3^+ = \texttt{reverse}([\mathrm{a}, \mathrm{b}], [\mathrm{b}, \mathrm{a}]) & o_4^- = \texttt{reverse}([\mathrm{b}, \mathrm{c}], [\mathrm{a}, \mathrm{a}, \mathrm{a}]) \end{array}$$

etc.

learn

$$h = \begin{array}{l} \texttt{reverse}([], []) \wedge \\ \texttt{reverse}([x|y], z) \leftarrow \texttt{reverse}(y, w) \wedge \texttt{append}(x, w, z) \end{array} \tag{32}$$

Inductive Logic Programming

Inductive Logic Programming (ILP)

Given sets O^+, O^- of Horn clauses and a logic program B (background knowledge), find a logic program h such that

$$\begin{aligned} h \wedge B &\models o^+ \text{ for all } o^+ \in O^+ \\ h \wedge B &\not\models o^- \text{ for all } o^- \in O^- \end{aligned} \tag{33}$$

Note that the lgg method of learning we have studied assumes the formulation (28) to which (33) *cannot* be reduced:

1. In (28), h is a single clause and B is a set of ground literals. In (33) both h and B are logic programs, i.e. conjunctions of Horn clauses.
2. (28) assumes *non-self-resolving* observations implying h is also non-self-resolving. In logic programs, self-resolving clauses in h are important as they facilitate *recursion*. See e.g. (31), (32).

ILP Algorithms

Inductive operators other than lgg have been proposed that allow learning (from) self-resolving clauses. They do not provide as strong guarantees as lgg, namely producing the lup with respect to the generality order. For details, see e.g. this theoretical ILP book, this newer monograph, or Chapter 19.5 of the AIMA Book. The ILP wiki page provides a brief overview.

(Hyperlinked material not part of exam.)

Alternatively, a first-order DNF can be learned by a fully *heuristic* approach adopting the *covering strategy* from the rule learning subfield of AI. This approach does not provide any theoretical guarantees although it is often effective in practice. We will demonstrate it on a slight restriction of the ILP task: learning a *predicate definition*.

Learning a Predicate Definition

A *predicate definition* is a logic program where all clauses have the same predicate in their positive atom (called *head*), such as member/2 in (31).

Using the equality predicate $=/2$, the terms in the heads can be also unified

$$\begin{aligned}\texttt{member}(x, y) &\leftarrow y = [x|z] \\ \texttt{member}(x, y) &\leftarrow y = [z|w] \wedge \texttt{member}(x, w)\end{aligned}$$

and the definition can be expressed as

$$\texttt{member}(x, y) \leftarrow y = [x|z] \vee (y = [z|w] \wedge \texttt{member}\,(x, w))$$

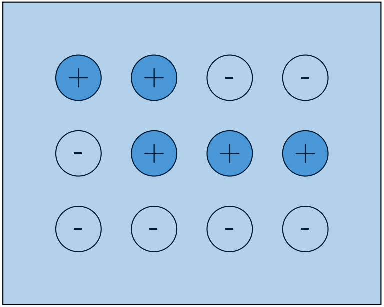
and thus learning the definition reduces to learning the right-hand side, which is a *monotone DNF*. It can be learned heuristically using the *covering algorithm*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Covering Algorithm for First-Order Logic DNF

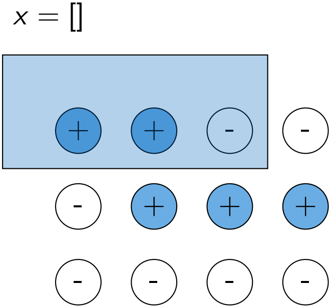
```
procedure LearnDNF($O^+$, $O^-$, $B$)
    $h$ = empty DNF.
    while $O^+$ not empty do
        $h = h \vee$ FindConjunction($O^+$, $O^-$, $B$, $h$)
        Remove from $O^+$ all $o^+$ such that $h \wedge B \models o^+$.
    end while
return $h$
end procedure

procedure FindConjunction($O^+$, $O^-$, $B$, $h$)
    $a$ = empty conjunction.
    while $(h \vee a) \wedge B \models o^-$ for some $o^- \in O^-$ do      ▷ negative example covered
        $a = a \wedge$ FindLiteral($O^+$, $O^-$, $B$, $h$)
    end while
return $a$
end procedure

procedure FindLiteral($O^+$, $O^-$, $B$, $h$, $a$)
    Finds a literal such that when added to $a$, $(h \vee a) \wedge B \models o$ for as many as possible
(at least one) $o \in O^+$, and as few as possible $o \in O^-$.
end procedure
```

Covering Algorithm: Example



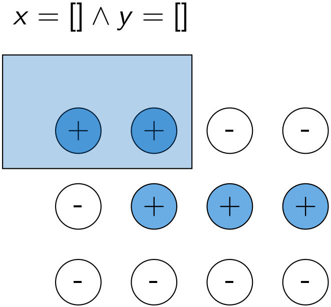
$\texttt{reverse}(x, y)$

Covering Algorithm: Example



$\texttt{reverse}(x, y) \leftarrow x = []$

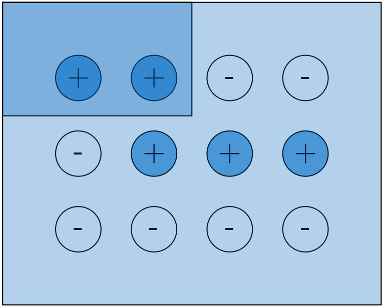
Covering Algorithm: Example



$\texttt{reverse}(x, y) \leftarrow x = [] \wedge y = []$

Covering Algorithm: Example

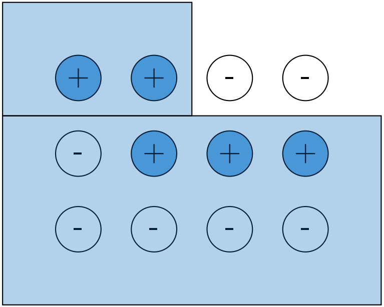
$x = [] \wedge y = []$



$\texttt{reverse}(x, y) \leftarrow x = [] \wedge y = [] \bigvee$

Covering Algorithm: Example

$x = [] \wedge y = []$

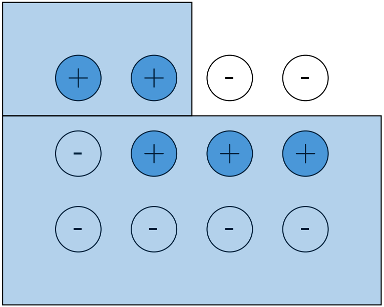


$x = [z|w]$

$$\mathtt{reverse}(x, y) \leftarrow x = [] \wedge y = [] \bigvee x = [z|w]$$

Covering Algorithm: Example

$x = [] \wedge y = []$

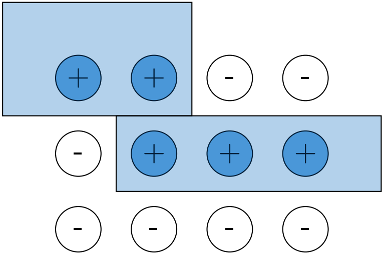


$x = [z|w] \wedge \texttt{reverse}(w, v)$

$$\texttt{reverse}(x, y) \leftarrow x = [] \wedge y = [] \bigvee x = [z|w] \wedge \texttt{reverse}(w, v)$$

Covering Algorithm: Example

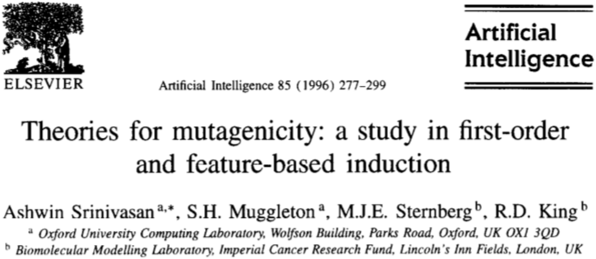
$x = [] \wedge y = []$



$x = [z|w] \wedge \texttt{reverse}(w, v) \wedge \texttt{append}(x, v, y)$

$\texttt{reverse}(x, y) \leftarrow x = [] \wedge y = [] \bigvee x = [z|w] \wedge \texttt{reverse}(w, v) \wedge \texttt{append}(x, v, y)$

Application Example: Predicting Mutagenicity



ELSEVIER

Artificial Intelligence 85 (1996) 277–299

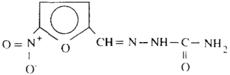
Artificial Intelligence

Theories for mutagenicity: a study in first-order and feature-based induction

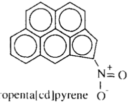
Ashwin Srinivasan [a,*], S.H. Muggleton [a], M.J.E. Sternberg [b], R.D. King [b]

[a] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, UK OX1 3QD

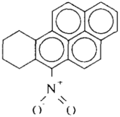
[b] Biomolecular Modelling Laboratory, Imperial Cancer Research Fund, Lincoln's Inn Fields, London, UK



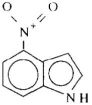
nitrofurazone



4-nitropenta[cd]pyrene



6-nitro-7,8,9,10-tetrahydrobenzo[a]pyrene

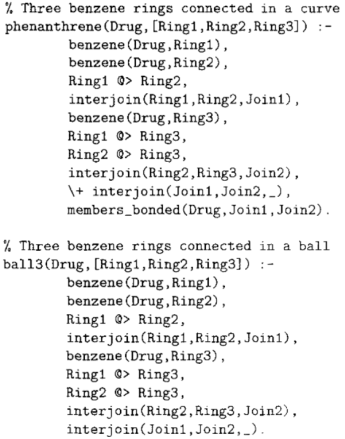


4-nitroindole

positive (mutagenic) examples o^+ negative (inactive) examples o^-
encoded as Horn clauses

Application Example: Predicting Mutagenicity (cont'd)

B defines known biochemical concepts.

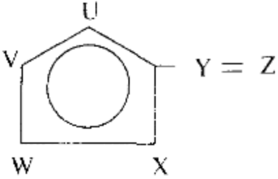


```
% Three benzene rings connected in a curve
phenanthrene(Drug,[Ring1,Ring2,Ring3]) :-
        benzene(Drug,Ring1),
        benzene(Drug,Ring2),
        Ring1 @> Ring2,
        interjoin(Ring1,Ring2,Join1),
        benzene(Drug,Ring3),
        Ring1 @> Ring3,
        Ring2 @> Ring3,
        interjoin(Ring2,Ring3,Join2),
        \+ interjoin(Join1,Join2,_),
        members_bonded(Drug,Join1,Join2).

% Three benzene rings connected in a ball
ball3(Drug,[Ring1,Ring2,Ring3]) :-
        benzene(Drug,Ring1),
        benzene(Drug,Ring2),
        Ring1 @> Ring2,
        interjoin(Ring1,Ring2,Join1),
        benzene(Drug,Ring3),
        Ring1 @> Ring3,
        Ring2 @> Ring3,
        interjoin(Ring2,Ring3,Join2),
        interjoin(Join1,Join2,_).
```

(etc.)

The ILP system Progol learned a logic program *h* satisfying (33).

One of its clauses encodes a previously unknown motif strongly predictive of mutagenicity.



Structured Output Prediction

Consider an ILP agent learning or having learned the definition for $\texttt{reverse}/2$.

In the concept-learning setting, the agent responds with binary actions, so the interaction is e.g. as in the left column:

o_k	a_k
$\texttt{reverse}([\mathrm{a},\mathrm{b},\mathrm{c}],[\mathrm{c},\mathrm{b},\mathrm{a}])$	1
$\texttt{reverse}([\mathrm{a},\mathrm{b}],[\mathrm{c}])$	0

o_k	a_k
$[\mathrm{a},\mathrm{b},\mathrm{c}]$	$[\mathrm{c},\mathrm{b},\mathrm{a}]$
$[\mathrm{a},\mathrm{b}]$	$[\mathrm{b},\mathrm{a}]$

The logic program is *generative*, i.e., for any list given in the first argument of query $\texttt{reverse}([\mathrm{a},\mathrm{b},\mathrm{c}],x)$, it computes the answer $x \mapsto [\mathrm{c},\mathrm{b},\mathrm{a}]$. Thus the agent can also interact with the environment as exemplified in the right column. This scenario is known as *structured output prediction*.

Bayesian Networks

Structured Output Prediction in the Binary Setting

We have seen a logic-programming method which facilitates structured output prediction. We will now study the latter in the more basic, binary setting.

The environment's state s_k at each k is a binary tuple. We assume s_k are sampled i.i.d. from the distribution P_S on $S = \{ 0, 1 \}^n$. (That does *not* mean the components of s_k are independent of each other!)

The agent receives a subset of the s_k components through observations o_k. Formally, $O = \{ 0, 1, ? \}^n$ and $P_O(o_k|s_k)$ generates o_k by replacing a random subset of s_k components with the ? symbol.

The components of o_k with the ? value are called *unobserved variables*, the other components are *observed variables*.

Structured Output Prediction: Rewards and Policy

As in concept learning, we have $A = S$ and we keep (8). So the policy value (6) is maximized by policy

$$a_k = \pi(o_k) = \arg\max_{s_k \in S} P_S(s_k | o_k)$$

Let $I_k = \{1 \leq i \leq n | o_k \neq ?\}$, i.e. I_k are indexes of the observed variables in o_k. Clearly, $a_k^i = o_k^i$ for $i \in I_k$, while the tuple $\left\langle a_k^i \right\rangle_{i \notin I_k}$ of unobserved variables maximizes P_S conditioned on the observed variables:

$$\left\langle a_k^i \right\rangle_{i \notin I_k} = \arg\max_{s^i \in S} P_S(\left\langle s^i \right\rangle_{i \notin I_k} \,|\, \left\langle o_k^i \right\rangle_{i \in I_k}) \tag{34}$$

Since the agent does not know P_S, it has to learn a model $\widehat{P}_S$ of it. *Graphical probabilistic models* (GPM) are an important class of methods to learn distribution models. We will study a particular GPM method, called *Bayesian Networks*.

Tractability due to Independence

Recall a similar task earlier in reinforcement learning, where we learned a model of $P_S(s'|s,a)$ computed from an array $N[s',s,a]$ of frequencies. This was feasible as the number of states was low.

Now we have 2^n states so an analogical array $N[s']$ would take memory exponential in n, unless the components s^i were pair-wise *independent*. Then only n probability estimates $P^i \in [0;1]$ (one of each s^i) would be needed to maintain as

$$\widehat{P}_S(s) = \prod_{s^i=1} P^i \prod_{s^j=0} (1 - P^j)$$

But the assumption is too strong. More realistically, the P_S may exhibit patterns of *conditional independence* between *some* pairs of variables.

Conditional Independence

Conditional Independence

Let $I \subseteq \{ 1, 2, \dots n \}$, $i, j \in \{ 1, 2, \dots n \} \setminus I$. We say that s^i, s^j are *conditionally independent given* $s^I = \{ s^i \mid i \in I \}$ if $P_S(s^i, s^j | s^I) = P_S(s^i | s^I) P_S(s^j | s^I)$.

Example with 3 binary variables:	
$s^1 = T$	outdoor temperature high
$s^2 = I$	ice-cream sales high
$s^3 = H$	heart-attack rates high

We will abbreviate $T = 1$ ($T = 0$) with t ($\neg t$, respectively.) So $\neg t$ means outdoor temperature is not high. Same convention for other binary random variables.

I and H are not independent:

$$P(I, H) \neq P(I)P(H)$$

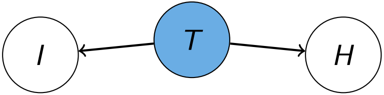
but they are *conditionally independent*:

$$P(I, H | \{ T \}) = P(I | \{ T \}) P(H | \{ T \})$$

Conditional Independence in Cause-Effect Graphs

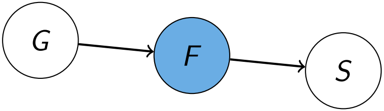
Note: dropping set delimiters for singletons in the conditional parts.

Heart attack and ice-cream eating independent if temperature known:



$$P(I, H|T) = P(I|T)P(H|T)$$

Son and grandfather's high IQ independent if same known for father:



$$P(S, G|F) = P(S|F)P(G|F)$$

In both cases: *any vertex is conditionally independent of all of its non-descendants given all its parents.*

Bayes Graph

Denote $\mathrm{par}_G(v)$ the set of all parents of vertex v in an oriented graph G.

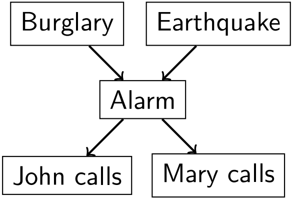
Bayes Graph

A *Bayes Graph* for a distribution P on variables $s^1, s^2, \dots s^n$ is an acyclic directed graph G with vertices $\{ s^1, s^2, \dots s^n \}$ such that any s^i is *conditionally independent (w.r.t. P) of all its non-descendants* in G *given $\mathrm{par}_G(s^i)$*.

Note: "given $\mathrm{par}_G(s^i)$" means that all the variables in $\mathrm{par}_G(s^i)$, and no other variables are in the conditional part of the probability query.

So a Bayes Graph is similar to cause-effect graphs but edges *need not correspond to cause-effect directions*. A Bayes graph for P indicates pairs of variables conditionally independent under P. There may by multiple BG's for one P. The fewer edges a BG has, the more independent pairs can be inferred (verify!).

Bayes Graph: Example (from AIMA)



From this Bayes graph, we can infer:

- $P(B,E) = P(B)P(E)$
- $P(J|X,A) = P(J|A)$ for all of $X \in \{ B, E, M \}$
- $P(M|X,A) = P(M|A)$ for all of $X \in \{ B, E, J \}$

By the 'chain rule' of probability

$$P(B,E,A,M,J) = P(J|B,E,A,M)P(M|B,E,A)P(A|B,E)P(B,E)$$

but this simplifies using the inferred equalities:

$$P(B,E,A,J,M) = P(J|A)P(M|A)P(A|B,E)P(B)P(E)$$

Multiplying right-to-left corresponds to going down the graph.

Conditional Probability Table

To specify $P(B, E, A, J, M)$ with a probability table using no independence assumption, we need $2^5 - 1 = 31$ parameters.

To specify $P(J|A)P(M|A)P(A|B, E)P(B)P(E)$, we need a *conditional probability table (CPT)* for each of the factors. E.g. for $P(A|B, E)$

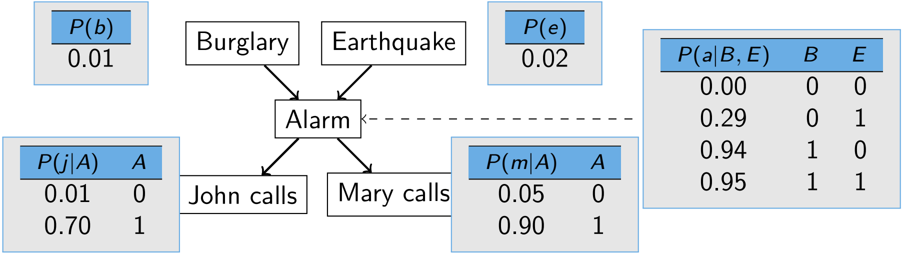
$P(a\|B, E)$	E	B
0.001	0	0
0.940	0	1
0.290	1	0
0.950	1	1

Of course, $P(\neg a|B, E) = 1 - P(a|B, E)$. Altogether, the BN's CPT's have $2 + 2 + 4 + 1 + 1 = 10$ parameters.

Bayes Network

Bayes Network

A *Bayes Network* for a distribution P consists of a Bayes graph G for P, and a conditional probability table for each vertex v of G, specifying $P(v|\mathrm{par}_G(v))$.



$P(b)$
0.01

$P(e)$
0.02

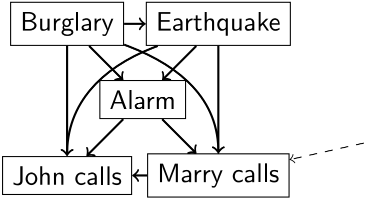
$P(a\|B,E)$	B	E
0.00	0	0
0.29	0	1
0.94	1	0
0.95	1	1

$P(j\|A)$	A
0.01	0
0.70	1

$P(m\|A)$	A
0.05	0
0.90	1

A BN fully specifies P. There are in general multiple BN's specifying the same P. More edges → more parameters.

A Different Graph for the Same Example



$P(m\|B,E,A)$	B	E	A
0.00	0	0	0
0.30	0	0	1
0.05	0	1	0
0.25	0	1	1

(. . . 4 more rows)

This Bayes graph does not imply any conditional independence. For each vertex, all non-descendants are parents. Joint distribution calculated as

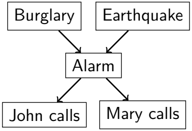
$$P(B,E,A,M,J) = P(J|B,E,A,M)P(M|B,E,A)P(A|B,E)P(E|B)P(B)$$

CPT's for the BN with this BG have $2^4 + 2^3 + 2^2 + 2 + 1 = 31$ parameters, same as the table for $P(B,E,A,M,J)$.

Computing Marginal Probabilities

So far we know how to compute the full joint distribution from CPT's:

$$P(B, E, A, J, M) = P(J|A)P(M|A)P(A|B, E)P(B)P(E)$$



A straightforward way to compute marginals, e.g. $P(A, J)$ is to *sum out* the remaining variables

$$\begin{aligned} P(A, J) &= \sum_B \sum_E \sum_M P(J|A)P(M|A)P(A|B, E)P(B)P(E) \\ &= P(J|A) \sum_M P(M|A) \sum_B \sum_E P(A|B, E)P(B)P(E) \end{aligned}$$

B under summation symbol means summing over b and $\neg b$. Same for other variables.

Computing Conditional Probabilities

Conditional probabilities are just fractions of marginals, e.g.

$$P(A, J|B, E) = \frac{P(A, J, B, E)}{P(B, E)}$$

Instead of calculating the denominator, we can evaluate the numerator for all assignments to A, J and normalize, since $\sum_A \sum_J P(A, J|B, E) = 1$. So

$$\alpha \left[P(\neg a, \neg j, B, E) + P(\neg a, j, B, E) + P(a, \neg j, B, E) + P(a, j, B, E)\right] = 1$$

After computing the summands, we compute $\alpha = 1/P(B, E)$ from the equation above. Then we can get the conditional probability for any $\langle A, J \rangle$; e.g. for $\langle \neg a, j \rangle$

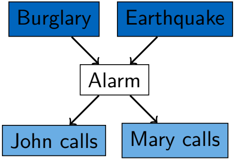
$$P(\neg a, j|B, E) = \alpha \cdot P(\neg a, j, B, E)$$

Evidence and Query Variables

In BN terminology, the variables ('vars', for short) whose joint conditional probability is computed are called *query* vars; the vars in the condition part are *evidence* vars.

Example query: *probability that neither John nor Mary will call during a burglary and no earthquake*:

$$P(\underbrace{\neg j, \neg m}_{\text{query}} \mid \underbrace{b, \neg e}_{\text{evidence}})$$

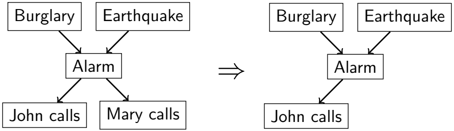


In (34), the query vars include *all* unobserved vars and evidence includes *all* observed vars. But BN's enable more general queries: query vars and evidence vars can be arbitrary subsets of all vars.

Removing Irrelevant Variables

Consider $P(J|b) = \alpha P(b) \sum_E P(E) \sum_A P(A|b, E) P(J, A) \sum_M P(M|A)$

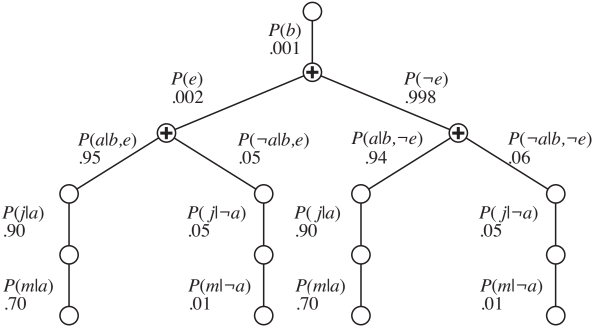
$\sum_M P(M|A) = 1$ so it can be left out, i.e. remove the corresponding vertex from the BN.



In general, *any vertex that is not an ancestor to a query or evidence variable of a query can be removed from the graph when computing the query*.

Redundant Computation

Consider $P(b|j,m) = \alpha P(b) \sum_E P(E) \sum_A P(A|b,e) P(j|A) P(m|A)$.
Observe the repeated sub-summing (branches) in its computation:



The *variable elimination* method removes this redundancy by using *factors*, which are arrays storing intermediate sums.

Computing Probabilities with Factors

Factors are arrays computed from CPT's and other factors. Example:

$$P(B|j,m) = \alpha \underbrace{P(B)}_{f_1(B)} \sum_E \underbrace{P(E)}_{f_2(E)} \sum_A \underbrace{P(A|B,E)}_{f_3(A,B,E)} \underbrace{P(j|A)}_{f_4(A)} \underbrace{P(m|A)}_{f_5(A)}$$

$f_1 \ldots f_5$ are the initial factors derived from CPT's. E.g. f_4 is just a CPT

$$f_4(A) = \begin{bmatrix} P(j|\neg a) \\ P(j|a) \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0.90 \end{bmatrix}$$

f_4 has no dimension for j since the latter is fixed $j = 1$. Same for f_5 and m.

Factors have an entry for each value of their arguments, which are the non-evidence (upper-case) vars in the corresponding probability expression. So e.g. f_3 has 8 entries unlike the corresponding CPT which has 4 entries.

Computing Probabilities with Factors (cont'd)

Same equation rewritten with factors and *point-wise* multiplication ×:

$$P(B|j,m) = \alpha f_1(B) \times \sum_E f_2(E) \times \sum_A f_3(A,B,E) \times f_4(A) \times f_5(A)$$

Example of point-wise multiplication $f_3(A,B,E) \times f_4(A) \times f_5(A) =$

$$\left[\begin{array}{c|l} 1.00 & (\neg a,\neg b,\neg e) \\ 0.71 & (\neg a,\neg b,e) \\ 0.06 & (\neg a,b,\neg e) \\ 0.05 & (\neg a,b,e) \\ 0.00 & (a,\neg b,\neg e) \\ 0.29 & (a,\neg b,e) \\ 0.94 & (a,b,\neg e) \\ 0.95 & (a,b,e) \end{array}\right] \times \left[\begin{array}{c|l} 0.05 & (\neg a) \\ 0.90 & (a) \end{array}\right] \times \left[\begin{array}{c|l} 0.01 & (\neg a) \\ 0.70 & (a) \end{array}\right] = \left[\begin{array}{c} 1.00 \cdot 0.05 \cdot 0.01 \\ 0.71 \cdot 0.05 \cdot 0.01 \\ 0.06 \cdot 0.05 \cdot 0.01 \\ 0.05 \cdot 0.05 \cdot 0.01 \\ 0.00 \cdot 0.90 \cdot 0.70 \\ 0.29 \cdot 0.90 \cdot 0.70 \\ 0.94 \cdot 0.90 \cdot 0.70 \\ 0.95 \cdot 0.90 \cdot 0.70 \end{array}\right]$$

Blue text = our labels, not part of the arrays!

Computing Probabilities with Factors (cont'd)

The multiplication is computed *only when a var is summed out* and involves only factors depending on the var. The operation yields a new factor:

$$f_6(B,E) = \sum_A f_3(A,B,E) \times f_4(A) \times f_5(A)$$

so f_6 is computed as

$$f_6(B,E) = f_3(a,B,E) \times f_4(a) \times f_5(a) + f_3(\neg a,B,E) \times f_4(\neg a) \times f_5(\neg a) =$$

$$= \left[\begin{array}{c|c} 1.00 \cdot 0.05 \cdot 0.01 + 0.00 \cdot 0.90 \cdot 0.70 & (\neg b, \neg e) \\ 0.71 \cdot 0.05 \cdot 0.01 + 0.29 \cdot 0.90 \cdot 0.70 & (\neg b, e) \\ 0.06 \cdot 0.05 \cdot 0.01 + 0.94 \cdot 0.90 \cdot 0.70 & (b, \neg e) \\ 0.05 \cdot 0.05 \cdot 0.01 + 0.95 \cdot 0.90 \cdot 0.70 & (b, e) \end{array}\right]$$

Now the equation is

$$P(B|j,m) = \alpha f_1(B) \times \sum_E f_2(E) \times f_6(B,E)$$

Computing Probabilities with Factors (cont'd)

Continue analogically:

$$f_7(B) = \sum_E f_2(E) \times f_6(B,E) = f_2(e) \times f_6(B,e) + f_2(\neg e) \times f_6(B,\neg e)$$

so finally we have

$$P(B|j,m) = \alpha f_1(B) \times f_7(B)$$

where $\alpha = 1/(f_1(b) \times f_7(b) + f_1(\neg b) \times f_7(\neg b))$.

Inferring probabilities from a BN takes exponential time (in n, the number of vars) in the worst case even with variable elimination.

Efficiency depends on the network structure as well as the order of variables in the product of probabilities made out of the BN.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

MAP-inference

Remind the agent's goal (34) to find the most probable values of the unobserved variables given all observed values.

In the present example, this can e.g. mean: given no earthquake and no-one calls, what is the most probable joint state of alarm and burglary?

So we want the joint state with *maximum aposteriori probability:*

$$\arg\max_{A,B} P(A, B|\neg e, \neg j, \neg m)$$

This is called the *MAP*-state and the task is called *MAP-inference*.

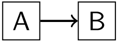
We want to find the MAP state of n (binary) variables without computing the probabilities of all $2^{n'}$ possible states with n' unobserved variables.

MAP-inference (cont'd)

Note that the joint MAP state need not consist of the states maximizing the marginals!

Consider e.g.

$P(a)$
0.6



$P(b\|A)$	A
0.90	0
0.55	1

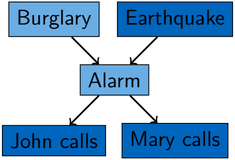
$$\arg\max_{A} P(A) = 1$$

but

$$\arg\max_{A,B} P(A,B) = (0,1)$$

MAP-inference: Example

Principle similar to probability inference. Example:



First compute the maximum value:

$$\max_{A,B} P(A, B|\neg e, \neg j, \neg m) =$$

$$\alpha \max_{A,B} P(B)P(\neg e)P(A|B, \neg e)P(\neg j|A)P(\neg m|A) =$$

$$\alpha P(\neg e) \max_{B} P(B) \max_{A} P(A|B, \neg e)P(\neg j|A)P(\neg m|A) = \text{(next page)}$$

We won't need to evaluate the normalizing constant α to compute the arg max.

MAP-inference by Variable Elimination

$$= \alpha P(\neg e) \max_B \overbrace{P(B) \underbrace{\max_A \underbrace{P(A|B,\neg e)P(\neg j|A)P(\neg m|A)}_{f_1'(A,B)}}_{f_1(B)}}^{f_2'(B)}$$

(Initial factors not shown).

$f_1'(A, B)$ and $f_2'(B)$ are *intermediate* factors (before *maximizing out* A resp. B).

Intermediate factors were not needed in probability inference, but in MAP inference they are stored for later retrieval of the maximizing argument.

MAP-inference by Variable Elimination (cont'd)

Using CPT's from the running example:

$$f_1'(A,B) = \left[\begin{array}{lll|l} 1 & \cdot(1-0.01) & \cdot(1-0.05) & (\neg a,\neg b) \\ (1-0.94) & \cdot(1-0.01) & \cdot(1-0.05) & (\neg a,b) \\ 0.00 & \cdot(1-0.7) & \cdot(1-0.9) & (a,\neg b) \\ 0.94 & \cdot(1-0.7) & \cdot(1-0.9) & (a,b) \end{array}\right] \approx \left[\begin{array}{c} 0.94 \\ 0.06 \\ 0.00 \\ 0.09 \end{array}\right]$$

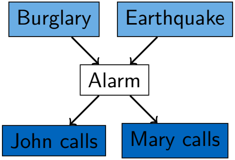
$$f_1(B) = \max_A f_1'(A,B) \approx \left[\begin{array}{c|c} 0.94 & (\neg b) \\ 0.09 & (b) \end{array}\right]$$

$$f_2'(B) \approx \left[\begin{array}{c|c} 0.99 \cdot 0.94 & (\neg b) \\ 0.01 \cdot 0.09 & (b) \end{array}\right] \approx \left[\begin{array}{c} 0.93 \\ 0.00 \end{array}\right]$$

First: $\arg\max_B f_2'(B) = \neg b$. Then: $\arg\max_A f_1'(A,\neg b) = \neg a$. So if no earthquake and no-one calls, the most probable state is no burglary, no alarm.

MAP-inference with Don't-Care Variables

Consider a case where query vars are not the full complement to evidence vars (as in here):



We want to find the most probable state of B, E when both John and Mary call; alarm is irrelevant. Now we have both maximization and summation:

$$\max_{B,E} P(B, E|j, m) = \alpha \max_{B,E} \sum_{A} P(B)P(E)P(A|B, E)P(j|A)P(m|A) \tag{35}$$

MAP-inference with Don't-Care Variables (cont'd)

Still can push operators before the first occurrence of their arguments *but* never swap max with $\sum$ since $\sum_Y \max_X P(X,Y) \neq \max_X \sum_Y P(X,Y)$.

So (35) rewrites to

$$\alpha \max_B \underbrace{P(B) \underbrace{\max_E \overbrace{P(E) \overbrace{\sum_A P(A|B,E)P(j|A)P(m|A)}^{f_1(B,E)}}^{f_2'(B,E)}}_{f_2(B)}}_{f_3'(B)}$$

MAP-inference with Don’t-Care Variables (cont’d)

$$f_1(B,E) = \left[\begin{array}{c|c} 1.00\cdot 0.01\cdot 0.95 + 0.00\cdot 0.70\cdot 0.09 & (\neg b,\neg e) \\ 0.79\cdot 0.01\cdot 0.95 + 0.29\cdot 0.70\cdot 0.09 & (\neg b,e) \\ 0.06\cdot 0.01\cdot 0.95 + 0.94\cdot 0.70\cdot 0.09 & (b,\neg e) \\ \underbrace{0.05\cdot 0.01\cdot 0.95}_{a} + \underbrace{0.95\cdot 0.70\cdot 0.09}_{\neg a} & (b,e) \end{array}\right] \approx \begin{bmatrix} 0.0095 \\ 0.0258 \\ 0.0598 \\ 0.0603 \end{bmatrix}$$

$$f_2'(B,E) \approx \begin{bmatrix} 0.98\cdot 0.0095 \\ 0.02\cdot 0.0258 \\ 0.98\cdot 0.0598 \\ 0.02\cdot 0.0603 \end{bmatrix} \approx \left[\begin{array}{c|c} 0.0093 & (\neg b,\neg e) \\ 0.0005 & (\neg b,e) \\ 0.0059 & (b,\neg e) \\ 0.0012 & (b,e) \end{array}\right]$$

$$f_2(B) = \max_E f_2(B,E) \approx \left[\begin{array}{c|c} 0.0093 & (\neg b) \\ 0.0059 & (b) \end{array}\right] \qquad f_3'(B) \approx \left[\begin{array}{c|c} 0.99\cdot 0.0093 & (\neg b) \\ 0.01\cdot 0.0059 & (b) \end{array}\right]$$

First: $\arg\max_B f_3'(B) = \neg b$. Then: $\arg\max_E f_2'(E,\neg b) = \neg e$. So even if J and M call, the most probable state is no burglary, no earthquake.

Learning a Bayes Network

A Bayes-Network model $\widehat{P}_{G,\mathbf{w}}$ of distribution P consists of a Bayes graph G and the entries of all its CPT's, jointly denoted $\mathbf{w}$. Two kinds of $\widehat{P}$ learning:

1. Estimating G from data,
2. Given G, estimating $\mathbf{w}$ from data

In both cases, learning is based on maximizing the *likelihood* $L(G, \mathbf{w})$, which is the probability of the m received observations according to the model:

$$L(G, \mathbf{w}) = \widehat{P}_{G,\mathbf{w}}(o_1, o_2, \ldots o_m)$$

but 1 is computationally much harder.

We will study only 2, i.e. assume that G is agent's background knowledge and write $L_G(\mathbf{w})$.

Likelihood Maximization

Given m observations and G with n vertices denoted $1, 2, \ldots n$, find $\mathbf{w}$ maximizing

$$L_G(\mathbf{w}) = \widehat{P}_{G,\mathbf{w}}(o_1, o_2, \ldots o_m) \overset{\text{i.i.d.}}{=} \prod_{j=1}^{m} \widehat{P}_{G,\mathbf{w}}(o_j) = \prod_{j=1}^{m} \prod_{i=1}^{n} \widehat{P}_{G,\mathbf{w}}(o_j^i | \mathrm{par}_G(i))$$

Parameters $\mathbf{w}$ consist of CPT tables $\mathbf{w} = \langle w^1, w^2, \ldots w^n \rangle$. Let $w^i(o)$ be the probability stored in the row of CPT w^i for which the values of o apply. E.g. if w^i is the right-most CPT here and $o = (B, E, A, J, M)$ is such that $B = 1$ and $E = 0$ then $w^i(o) = 0.94$. With this notation:

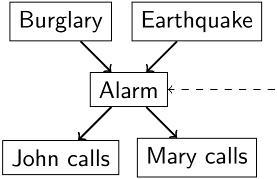
$$\widehat{P}_{G,\mathbf{w}}(o^i | \mathrm{par}_G(i)) = \begin{cases} 1 & \text{if } o^i = ? \text{ or } o^j = ? \text{ for some } j \in \mathrm{par}_G(i) \\ w^i(o) & \text{if } o^i = 1 \\ 1 - w^i(o) & \text{if } o^i = 0 \end{cases}$$

Value 1 in the last case drops from the product above a term whose value cannot be determined due to missing values in data.

Likelihood Maximization (cont'd)

Denote $m_i^+[j]$ the number of observations o from $o_1, o_2, \ldots o_m$ that satisfy conditions of line j in CPT w^i and $o^i = 1$. Similarly define $m_i^-[j]$, except with $o^i = 0$.

Example:



$P(a\|B,E)$	B	E
$w_A[1]$	0	0
$w_A[2]$	0	1
$w_A[3]$	1	0
$w_A[4]$	1	1

	B	E	A	J	M
o_1	0	1	1	0	0
o_2	?	1	1	1	1
o_3	0	1	1	1	0
o_4	0	1	0	?	0
o_5	1	0	0	0	?
o_6	?	?	1	1	1

$m_A^+[2] = 2,\ m_A^-[2] = 1.$

Likelihood Maximization (cont'd)

Reformulate the likelihood using the counts $m_i^+[j]$, $m_i^-[j]$:

$$L(\mathbf{w}) = \prod_{i=1}^{n} \prod_{j=1}^{2^{|\mathrm{par}_G(i)|}} \underbrace{w^i[j]^{m_i^+[j]}(1 - w^i[j])^{m_i^-[j]}}_{\overset{\text{def.}}{=} L(w^i[j])}$$

$L(\mathbf{w})$ is maximized by maximizing separately the likelihood $L(w^i[j])$ of each parameter $w^i[j]$. From

$$\frac{\partial L(w^i[j])}{\partial w^i[j]} = 0$$

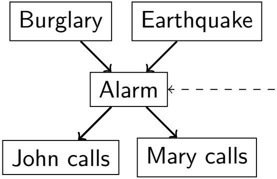
we get for $m_i^+[j] + m_i^-[j] > 0$

$$w^i[j] = \frac{m_i^+[j]}{m_i^+[j] + m_i^-[j]} \tag{36}$$

which is the *relative frequency* estimate.

Likelihood Maximization (cont'd)

Example:



$P(a \mid B, E)$	B	E
$w_A[1]$	0	0
$w_A[2]$	0	1
$w_A[3]$	1	0
$w_A[4]$	1	1

	B	E	A	J	M
o_1	0	1	1	0	0
o_2	?	1	1	1	1
o_3	0	1	1	1	0
o_4	0	1	0	?	0
o_5	1	0	0	0	?
o_6	?	?	1	1	1

$$\arg\max L(w_A[2]) = \frac{2}{2+1} \qquad \arg\max L(w_A[3]) = \frac{0}{0+1}$$

$w_A[1], w_A[4]$ cannot be determined due to zero-denominator. Need to wait for more observations.

BN Agent: Design

Design principles for and agent learning a BN model in the online setting, given Bayes graph G as background knowledge:

- Keep the counts $m_i^+[j], m_i^-[j]$ for $i = 1 \dots n$, $j = 1 \dots 2^{\mathrm{par}_G(i)}$ in the agent's state. Initially all zero.
- On each observation, increment the appropriate counts
- Set probabilities in all CPT's w^i by (36). If denominator zero, default to e.g. $1/2$.
- Decide by (34) replacing P_S with its model $\widehat{P}_{G,\mathbf{w}}$.

BN Agent: Design (con't)

An improvement is possible in the *batch* learning setting analogical to that we defined for concept learning.

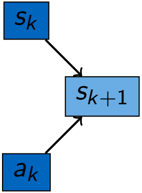
- i.e., agent first collects a *training set* and then calculates its model $\mathbf{w}$ from it.

This allows to use the *EM algorithm* to handle missing observation values in parameter estimation:

1. Replace missing values in observations with random initial values
2. E step: Compute $\mathbf{w}$ with the resulting data by (36).
3. M step: Replace the initially missing values with their MAP states according to $P_{G,\mathbf{w}}$.
4. Repeat from 2 until convergence.

Non-Boolean Variables: Example

Extension from $S = \{\,0,1\,\}^n$ to arbitrary finite S is straightforward. Example: learning a model of $P_S(s_{k+1}|s_k, a_k)$ in the grid world from reinforcement learning. $|S| = 3 \cdot 4 - 1 = 11$.

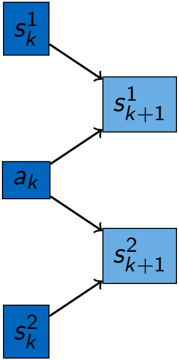


$P(\langle 1,1\rangle)$	$P(\langle 1,2\rangle)$ . . .	$P(\langle 4,2\rangle)$	s_k	a_k
Probabilities of $3 \cdot 4 - 1 = 10$ states			$\langle 1,1\rangle$	left
for each of $11 \cdot 4 = 44$ values of $\langle s_k, a_k\rangle$			$\langle 1,1\rangle$	right
. . .			. . .	

CPT not needed for the observed variables s_k and a_k to evaluate $\widehat{P}_S(s_{k+1}|s_k, a_k)$. Model has $10 \cdot 44 = 440$ parameters. Same as the model we used with the ADP agent.

Non-Boolean Variables: Example

Now assume that actions left, right never cause vertical movement and action up, down never cause horizontal movement. This makes s^1_{k+1} (horizontal coordinate) independent of s^2_k and s^2_{k+1} (vertical coordinate) independent of s^1_k allowing for a more BN with only $3 \cdot 16 + 2 \cdot 12 = 72$ parameters.

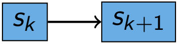


$P(1)$	$P(2)$	$P(3)$	s^1_k	a_k
Probs of 3 vals of s^1_{k+1}			1	left
for each of $4 \cdot 4 = 16$			1	right
values of $\langle s^1_k, a_k \rangle$			1	up
...			...	

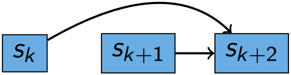
$P(1)$	$P(2)$	s^2_k	a_k
Probs of 2 vals of s^2_{k+1}		1	left
for each of $3 \cdot 4 = 12$		1	right
values of $\langle s^2_k, a_k \rangle$		1	up
...		...	

Temporal Bayesian Networks

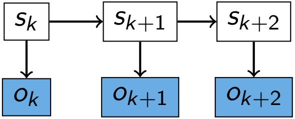
Bayesian networks generalize some well known temporal models.



Markov process (1^{st} order)



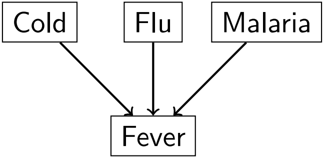
Markov process (2^{nd} order)



Markov hidden process (1^{st} order, 3 observations)

BN Encoding Logical Rules

Bayesian networks generalize propositional logical rules.



$P(\texttt{fever} \mid C, F, M)$	C	F	M
1	1	1	1
0	all other cases		

$\texttt{fever} \leftarrow \texttt{cold} \wedge \texttt{flu} \wedge \texttt{malaria}$

$P(\texttt{fever} \mid C, F, M)$	C	F	M
0	0	0	0
1	all other cases		

$\texttt{fever} \leftarrow \texttt{cold} \vee \texttt{flu} \vee \texttt{malaria}$