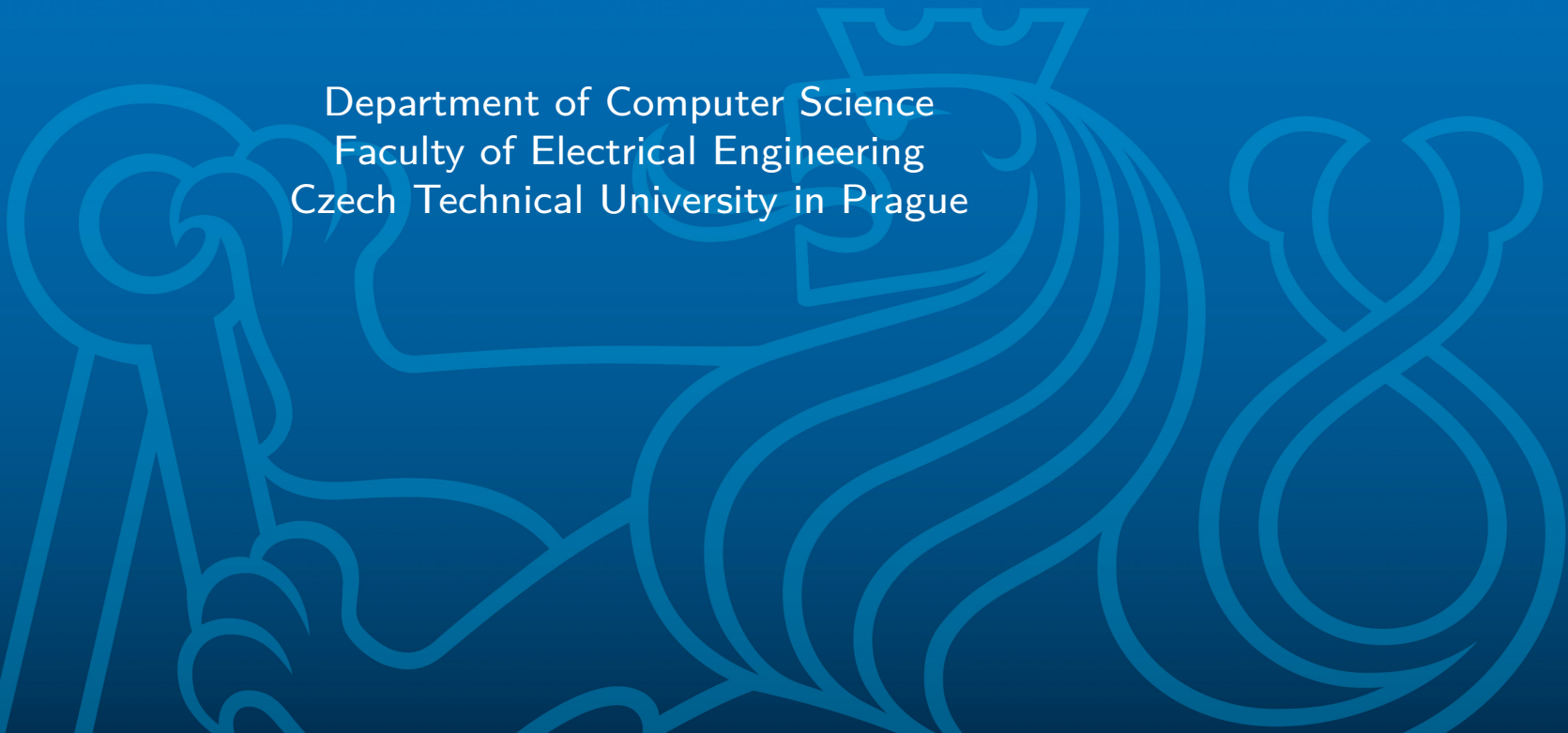
Symbolic Machine Learning

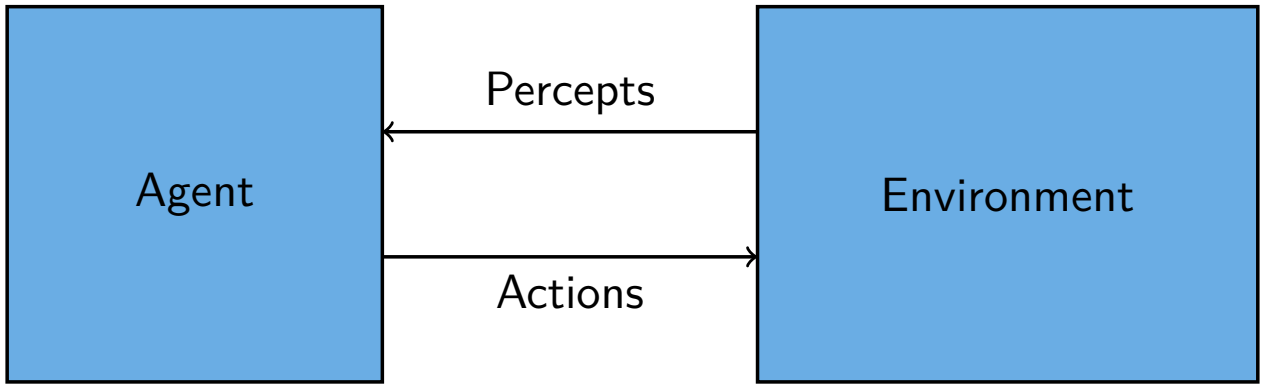
Lecture Slides

Filip Železný

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



Agent Interacts with Environment



- Discrete *time*

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

- *Percepts*

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

- *Actions*

$$\forall k : y_k \in Y$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Histories, Probability, Agent's Policy

History

$$xy_{\leq k} = x_1, y_1, x_2, y_2, \ldots, x_k, y_k$$

has probability

$$\mu(xy_{\leq k}) = \mu(x_1)\mu(y_1|x_1)\mu(x_2|x_1, y_1)\ldots\mu(x_k|xy_{<k})\mu(y_k|x_k, xy_{<k})$$

History influenced by agent's deterministic *policy*

$$y_k = \pi(xy_{<k}, x_k)$$

so

$$\mu(y_k|xy_{<k}, x_k) = \begin{cases} 1 \text{ if } y_k = \pi(xy_{<k}, x_k) \\ 0 \quad \text{otherwise} \end{cases}$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards

Rewards $r_k \in R \subset \mathbf{R}$ are a distinguished part of percepts

$$x_k = (o_k, r_k)$$

Everything else in the percepts are *observations* $o_k \in O$.

Note: r_1 is immaterial.

Define marginals on O and R so that

$$\mu(x_k|xy_{<k}) = \mu(o_k, r_k|xy_{<k}) = \mu_O(o_k|r_k, xy_{<k})\mu_R(r_k|xy_{<k})$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Nonsequential Interaction

Nonsequential: a special, simplified case of interaction.

- Observations are i.i.d.:

$$\mu_O(o_k|r_k, xy_{<k}) = \mu_O(o_k)$$

  Note: very strong assumption!

- Reward given by previous observation and the action taken on it:

$$\mu_R(r_k|o_k, xy_{<k}) = \mu_R(r_k|o_{k-1}, y_{k-1})$$

Note that through y_{k-1}, r_k still depends on the entire preceding history.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Batch Learning

Batch: a non-sequential interaction with two successive phases:

- *learning (training, exploration)* at $k = 1, 2, \ldots K$
- *action (testing, exploitation)* at $k = K + 1, K + 2, \ldots$

Assumption: no change of policy in the action phase, so for $k > K$

$$y_k = \pi(xy_{\leq K}, o_k)$$

A model for most "data-mining" scenarios.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Rewards in Batch Learning

In the action phase ($k > K$)

$$\mu_R(r_k|o_{k-1}, y_{k-1}) = \mu_R(r_k|o_{k-1}, xy_{\leq K})$$

Batch case being non-sequential, o_{k-1} is i.i.d. from $\mu_O(o_{k-1})$ that does not depend on k. So r_k is i.i.d from

$$\mu_R(r_k|xy_{\leq K})$$

where

$$\mu_R(r_k|xy_{\leq K}) = \sum_{o_{k-1} \in O} \mu_O(o_{k-1}) \mu_R(r_k|o_{k-1}, xy_{\leq K})$$

So a learning history $xy_{\leq K}$ determines the reward distribution (and the expected reward) in the action phase.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent's Goal

Given a *finite* time horizon $k' \in N$ of interest, choose $y_1, y_2, \ldots, y_{k'-1}$ to maximize expected cumulative reward

$$\sum_{r_{\leq k'}} \mu_R(r_{\leq k'}|y_{\leq k'})(r_1 + r_2 + \ldots + r_{k'})$$

If horizon *infinite*, use the discount sequence δ_k such that $\sum_{i=1}^{\infty} \delta_i < \infty$ and maximize

$$\lim_{k' \to \infty} \sum_{r_{\leq k'}} \mu_R(r_{\leq k'}|y_{\leq k'}) \sum_{k=1}^{k'} r_k \delta_k$$

In the *batch* scenario, simply maximize the expected reward in the action phase

$$\sum_{r_k \in R} \mu_R(r_k|xy_{\leq K}) r_k$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Environment: State-Based Description

Assumption: environment fully described by its *state* $e_k \in E$ at each k.

Generation of percepts $x_k = (o_k, r_k)$:

- observations: $\mu_o(o_k|e_k, y_{k-1})$
- rewards: $\mu_r(r_k|e_{k-1}, y_{k-1})$

Assume $|E| <$ some constant that does not depend on k. So the environment has a *finite memory* and percepts do not depend on the entire history.

State gets *updated stochastically* at each $k > 1$ according to distribution

$$\mathcal{E}(e_k|e_{k-1}, y_{k-1})$$

Note: x_{k-1} not assumed among the conditions.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent: State-Based Description

Assumption: agent fully described by its *state* $a_k \in A$ at each k.

Generation of actions y_k:

$$y_k = \pi(a_k, o_k)$$

Assume $|A| <$ some constant that does not depend on k. So the agent has a *finite memory* and percepts do not depend on the entire history.

State gets *updated deterministically* at each $k > 1$ according to function

$$a_k = \mathcal{A}(a_{k-1}, x_k)$$

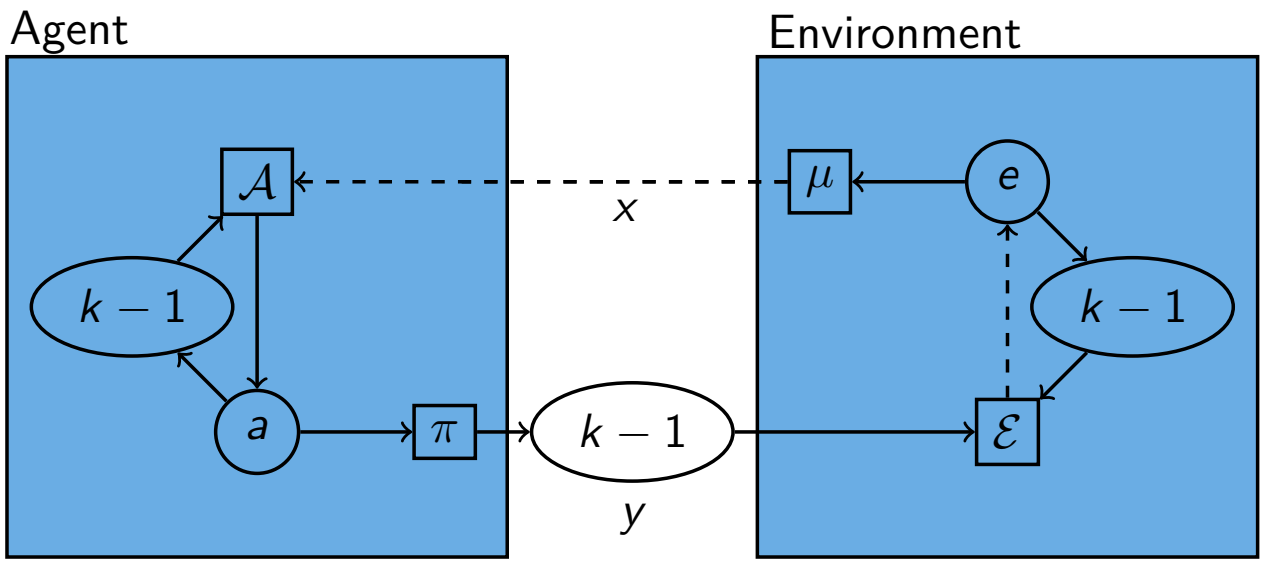
Note: x_k in the argument, not x_{k-1}. Recall the 'clock' ticks after agent's action.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

State-Based Interaction Model

Although we will not do it, o_k (resp. e_{k-1} and y_{k-1}) can be formally remembered as part of a_k (resp. e_k), allowing a simplified notation:

Agent		Environment	
actions:	$y = \pi(a_k)$	percepts:	$x_k \sim \mu(x_k \mid e_k)$
state update:	$a_k = \mathcal{A}(a_{k-1}, x_k)$	state update:	$e_k \sim \mathcal{E}(e_k \mid e_{k-1}, y_{k-1})$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Non-sequential and Batch with States

Non-sequential assumption: e_k sampled i.i.d. from

$$\mathcal{E}(e_k)$$

that does not depend on k, so observations are also i.i.d from

$$\mu_O(o_k) = \sum_{e_k \in E} \mu_O(o_k|e_k)\mathcal{E}(e_k)$$

Rewards in the action phase of the *batch scenario* are also i.i.d. from distribution $\mu_R(r_k|a_K)$ depending only on the last agent's state a_K in the learning phase (see textbook for derivation).

Agent should reach state maximizing the expected reward

$$\sum_{r_k \in R} \mu_R(r_k|a_K)r_k$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

On-line Concept Learning: Example

experiment number k	apoptosis initiated e_k	protein 1 present o_k^1	protein 1 active o_k^2	protein 2 present o_k^3	protein 2 active o_k^4	prediction y_k	reward r_k
1	0	0	0	0	0	0	0
2	0	0	0	1	0	1	0
3	0	1	0	0	0	1	-1
4	1	1	1	0	0	0	-1
5	0	1	0	1	1	0	-1
6	1	1	1	1	0	1	0
7	1	1	1	0	0	1	0
8	0	1	0	1	1	0	0
(etc.)							

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Learning Assumptions

Assumption: "*observation identifies state*", i.e. for any observation o, there is at most one state e such that

$$\mu_O(o|e) > 0$$

Further simplifying assumptions (for convenience):

- Binary environment states: $E = \{0, 1\}$
- Observations are binary tuples $O = \{0, 1\}^n$ $(n \in N)$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Decisions and Rewards in Concept Learning

- Decisions are guesses about states, so $Y = E$
- Agent is punished for the wrong guess

$$r_{k+1} = \begin{cases} 0 & \text{if } e_k = y_k \\ -1 & \text{otherwise} \end{cases}$$

- Note: this is a special case of the *loss function*

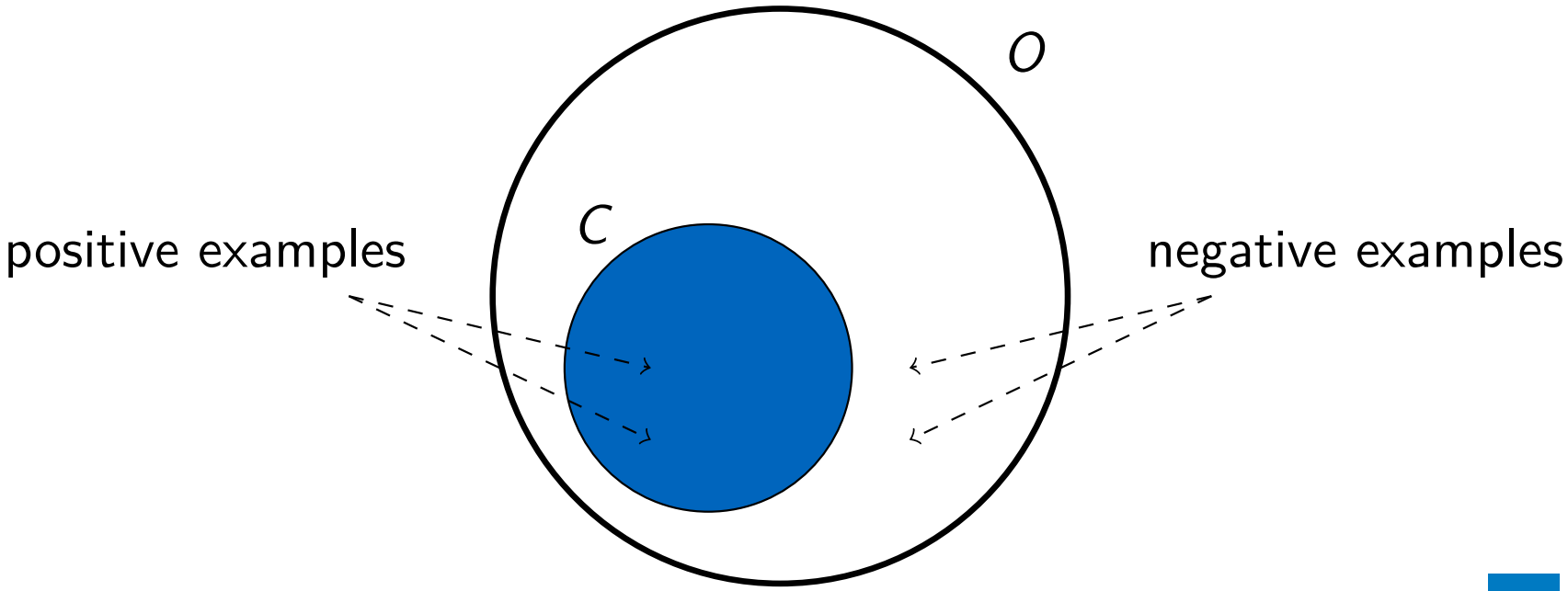
$$r_{k+1} = -L(e_k, y_k)$$

called *unit loss*. We will only work with unit loss.

Concept

- Environment (target) concept:

$$C = \{ o \in O \mid \mu_O(o|1) > 0 \}$$



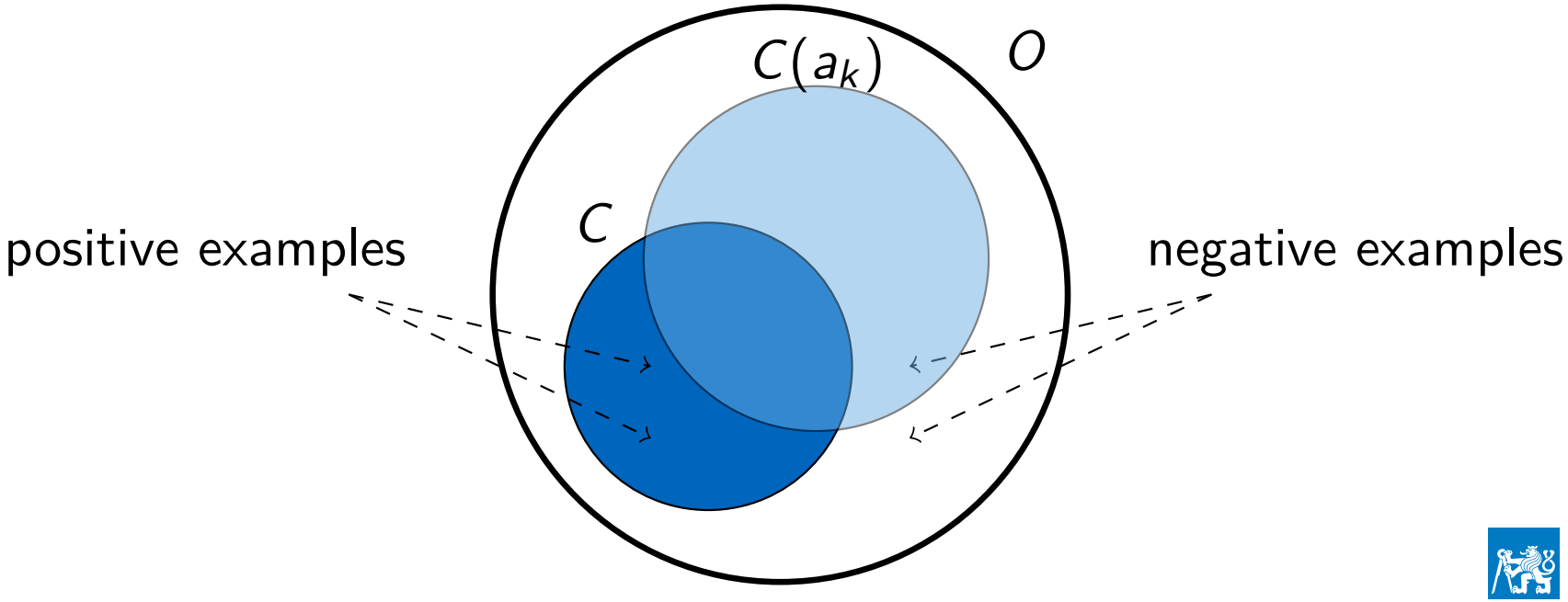
Concept

- Environment (target) concept:

$$C = \{ o \in O \mid \mu_O(o|1) > 0 \}$$

- Agent's (hypothesized) concept at time k:

$$C(a_k) = \{ o \in O \mid \pi(a_k, o_k) = 1 \}$$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Designing a Concept Learning Agent

Note on concept-learning terminology:

- C - *positive class*, $O \setminus C$ - *negative class*
- e_k is the *class label* of o_k.
- "observations" $\sim$ "examples"

First thought:

- At $k+1$, agent knows the class of o_k (it is $|y_k + r_{k+1}|$).
- It can remember a finite set of labeled examples in its state.
- If state size overflown, e.g. forget the oldest example.
- For a unseen observation, give the prevailing class among the most similar (e.g. Hamming distance) observations remembered.
- This is known as nearest-neighbor classification and is barely learning.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent with a Hypothesis

Just like a human, the agent should learn a *hypothesis* by which it will decide.

At time k, agent has hypothesis h_k stored in its state a_k.

As h_k is the only part of a_k influencing the decisions, we may write $\pi(h_k, o_k)$ and $C(h_k)$ instead of $\pi(a_k, o_k)$ and $C(a_k)$.

h_k can be 'anything' (a set of rules, equations, a graph, ...) as long as π can produce a decision out of it.

Updating a Hypothesis

When $r_k = -1$, the agent knows it made a wrong decision at $k-1$ and should change its hypothesis.

Most of the times, it needs to know o_{k-1} to do it. So the last observation will be memorized as part of the state

$$a_k = (o_k, h_k)$$

and the update function $\mathcal{A}(a_{k-1}, x_k)$ will be instantiated to

$$\mathcal{A}\left(\left(o_{k-1}, h_{k-1}\right), \left(o_k, r_k\right)\right) = (o_k, h_k)$$

Agent's intelligence rests in the way it updates towards h_k given h_{k-1} and o_{k-1}, whenever $r_k = -1$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent

Let us design an agent whose hypothesis is a propositional *logic formula*.

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

h_k is made of propositional variables $p_1, p_2, \ldots p_n$ and the binary tuple o_k is a truth-value assignment to them.

More specifically, we assume h_k to be a *conjunction*.

The agent will start with the conjunction of all possible literals and on each error, it will delete all inconsistent literals (i.e. literals logically false for the given example).

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Update Step Example

For $n = 4$:

$$
\begin{aligned}
h_1 = \quad & p_1 \wedge \neg p_1 \quad \wedge p_2 \wedge \neg p_2 \quad \wedge p_3 \wedge \neg p_3 \quad \wedge p_4 \wedge \neg p_4 \\
o_1 = (\quad & 1, \quad 1, \quad 0, \quad 0) \\
y_1 = \quad & 0
\end{aligned}
$$

$$
\begin{aligned}
r_2 &= -1 \\
h_2 = \quad & p_1 \quad \wedge p_2 \quad \wedge \neg p_3 \quad \wedge \neg p_4
\end{aligned}
$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Update Step

Formally:

$$h_k = \begin{cases} h_{k-1} \text{ if } r_k = 0 \\ \text{delete}(h_{k-1}, o_{k-1}) \text{ otherwise} \end{cases}$$

where

$$\text{delete}\left(\bigwedge_{i \in I} p_i \bigwedge_{j \in J} \neg p_j, (o^1, o^2, \ldots, o^n)\right) = \bigwedge_{\substack{i \in I \\ o^i = 1}} p_i \bigwedge_{\substack{j \in J \\ o^j = 0}} \neg p_j$$

That is, retains exactly the consistent literals.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: How Well Does it Learn?

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

Observations (see textbook for more rigor):

1. All literals of h^* are consistent with any 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 3: mistakes are made only on positive examples.
5. From 3: on each mistake, at least one literal in the hypothesis is inconsistent and gets deleted.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent: Mistake Bound

Since at least one literals is deleted on each mistake, and the initial hypothesis has $2n$ literals, *no more than $2n$ mistakes* are made before $h_k = h^*$.

So the cumulative reward is

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

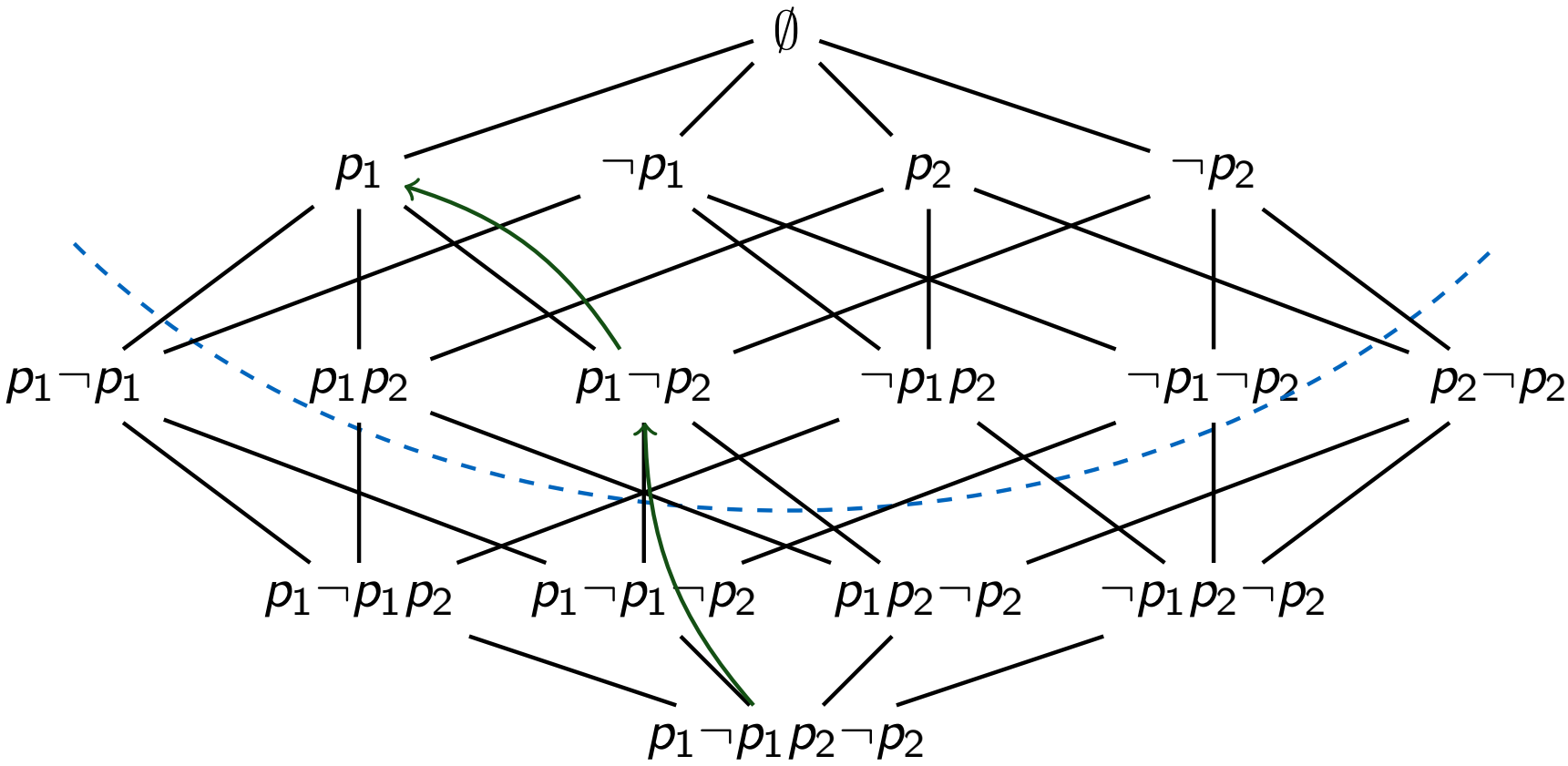
for an arbitrary horizon $k' \in N$.

(Also a lower bound on the expected cumulative reward for any horizon)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption Lattice

We call the agent generalizing because $\forall k,\ h_{k+1} \subseteq h_k$. We say that conjunction h_{k+1} *subsumes* conjunction h_k and learning can be viewed as search in a *subsumption lattice*.



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Subsumption and Entailment

For *conjunctions*, subsumption implies logical entailment:

$$\text{if } h \subseteq h' \text{ then } h' \vdash h$$

But the reverse implication is not true for *self-resolving* conjunctions on the LHS. E.g. $p_1 \wedge \neg p_1 \vdash p_2$. In propositional logic, self-resolving conjunctions are *contradictions*.

For *clauses* (i.e. disjunctions), the implication is different:

$$\text{if } h \subseteq h' \text{ then } h \vdash h'$$

Again, the reverse implication is not true for *self-resolving* clauses on the RHS. E.g. $p_2 \vdash p_1 \vee \neg p_1$. In propositional logic, self-resolving clauses are *tautologies*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent for Disjunctions

Any disjunction can be converted to a conjunction of same size (up to additive/multiplicative factor). E.g.

$$p_1 \vee \neg p_2 \vee p_3 = \neg\left(\neg p_1 \wedge p_2 \wedge \neg p_3\right)$$

So just learn a conjunction (as in the parentheses) with the same algorithm except with inverted actions ($1 - y_k$ instead of y_k).

Optionally, produce a disjunction by going from the RHS to the LHS.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent for s-CNF, s-DNF

An *s-clause* ($s \in N$) is a disjunction of at most s literals. An *s-CNF* is a conjunction of s-clauses.

With n variables, $\mathcal{O}\left[\binom{2n}{s}\right] = \mathcal{O}(n^s)$ different s-clauses can be made. Assign a new propositional variable p'_i to each of them.

Learn a conjunction with the new variables, determining the truth value of each p'_i as that of the assigned clause whose variables are valuated by the examples.

Finally produce the learned s-CNF by replacing the variables in the conjunction with their assigned clauses.

Method efficient if s is a small constant. s-DNF (disjunctions of *s-terms*) learnable analogically.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sidenote: Non-Binary Observations

What if observations are richer, e.g. tuples of rational numbers (Q)?

One way is to discretize o_k^i along each axis i

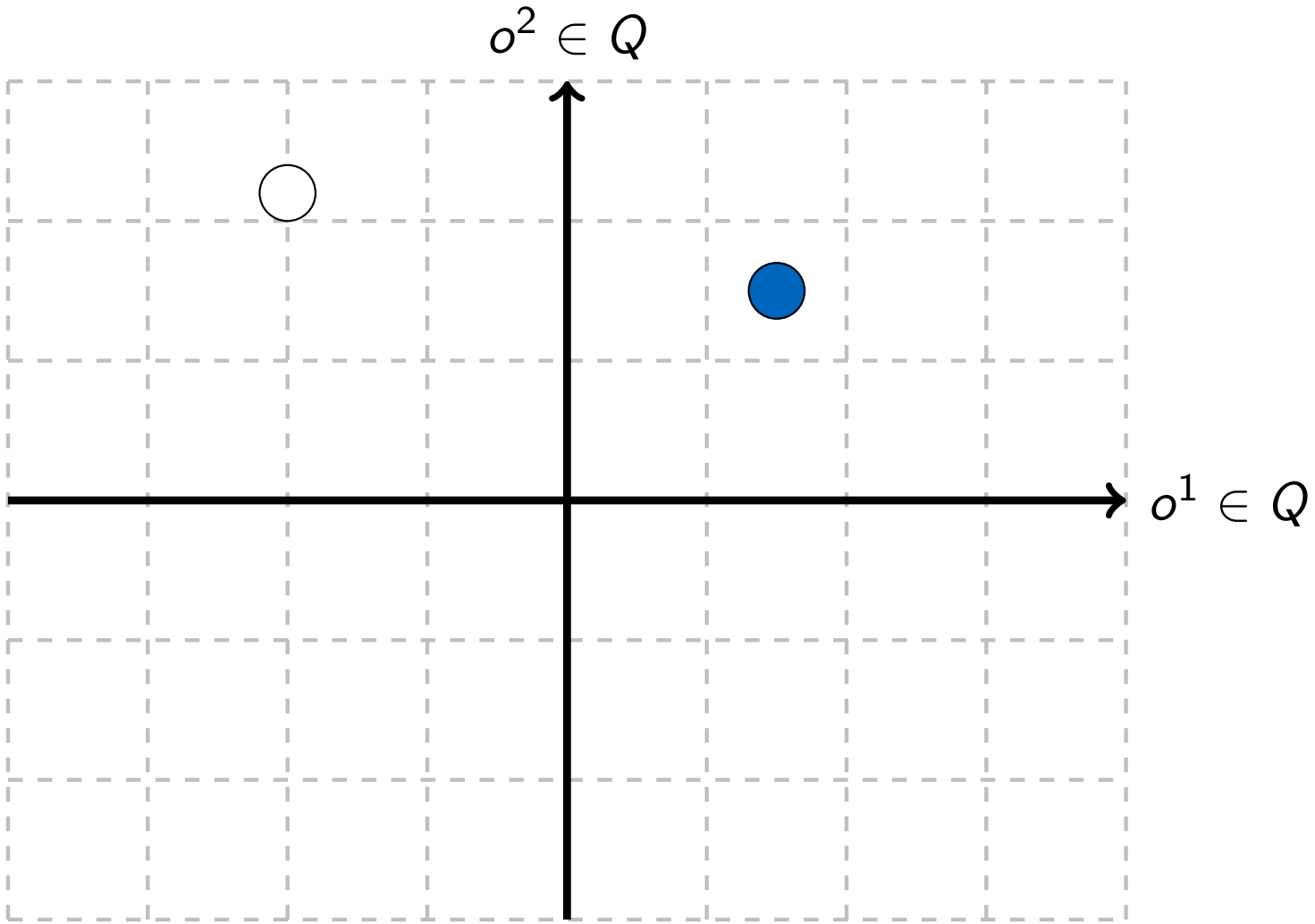
$$o_k^1 > \theta_1^1 \qquad o_k^1 > \theta_2^1 \qquad \ldots$$
$$o_k^2 > \theta_1^2 \qquad o_k^2 > \theta_2^2 \qquad \ldots$$

obtaining a (bigger) set of binary observations tuples we are 'used to'.

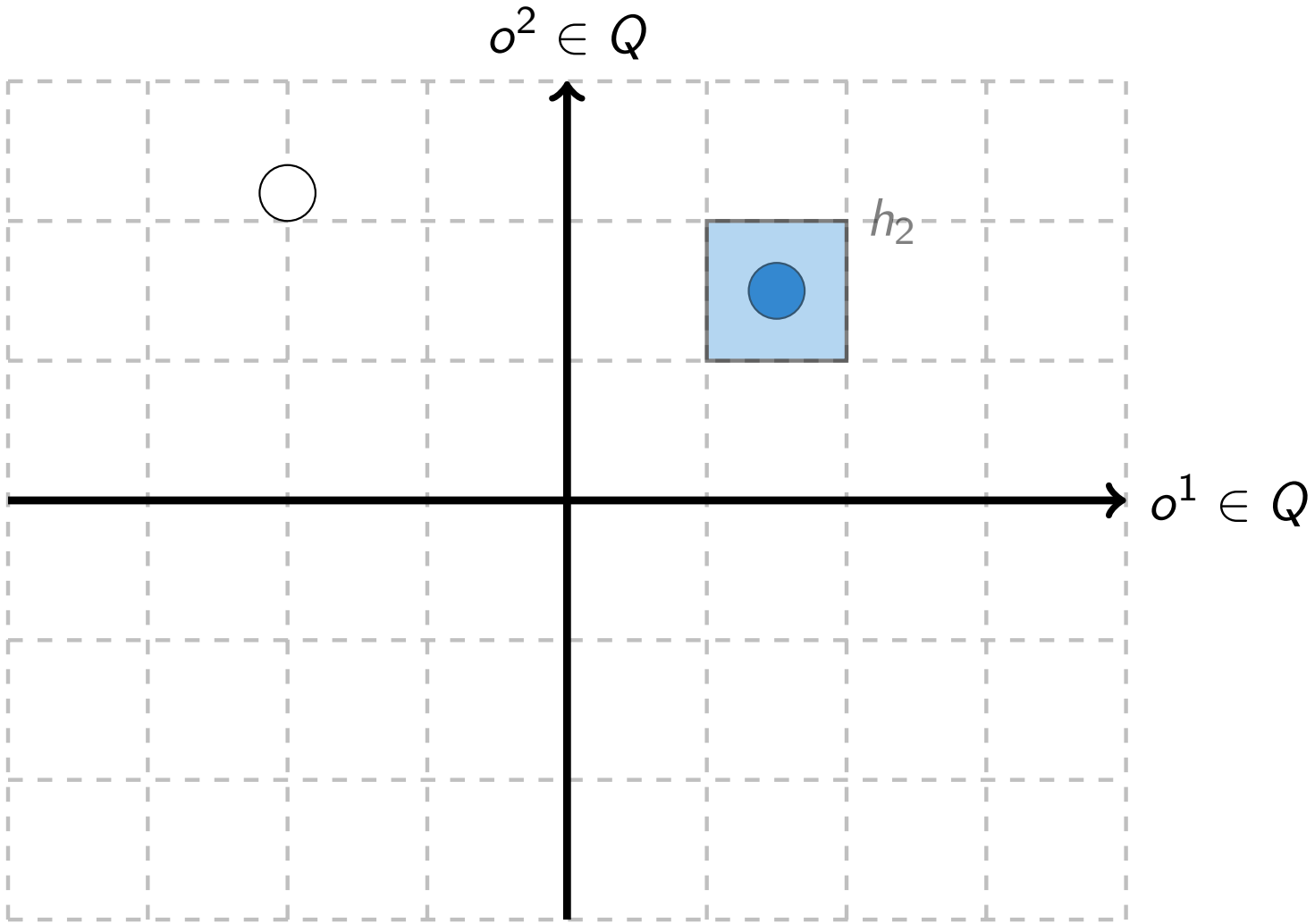
Further note: the components of observation tuples are also called *attributes* or *features*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Visualizing Generalization (with Rational Features)

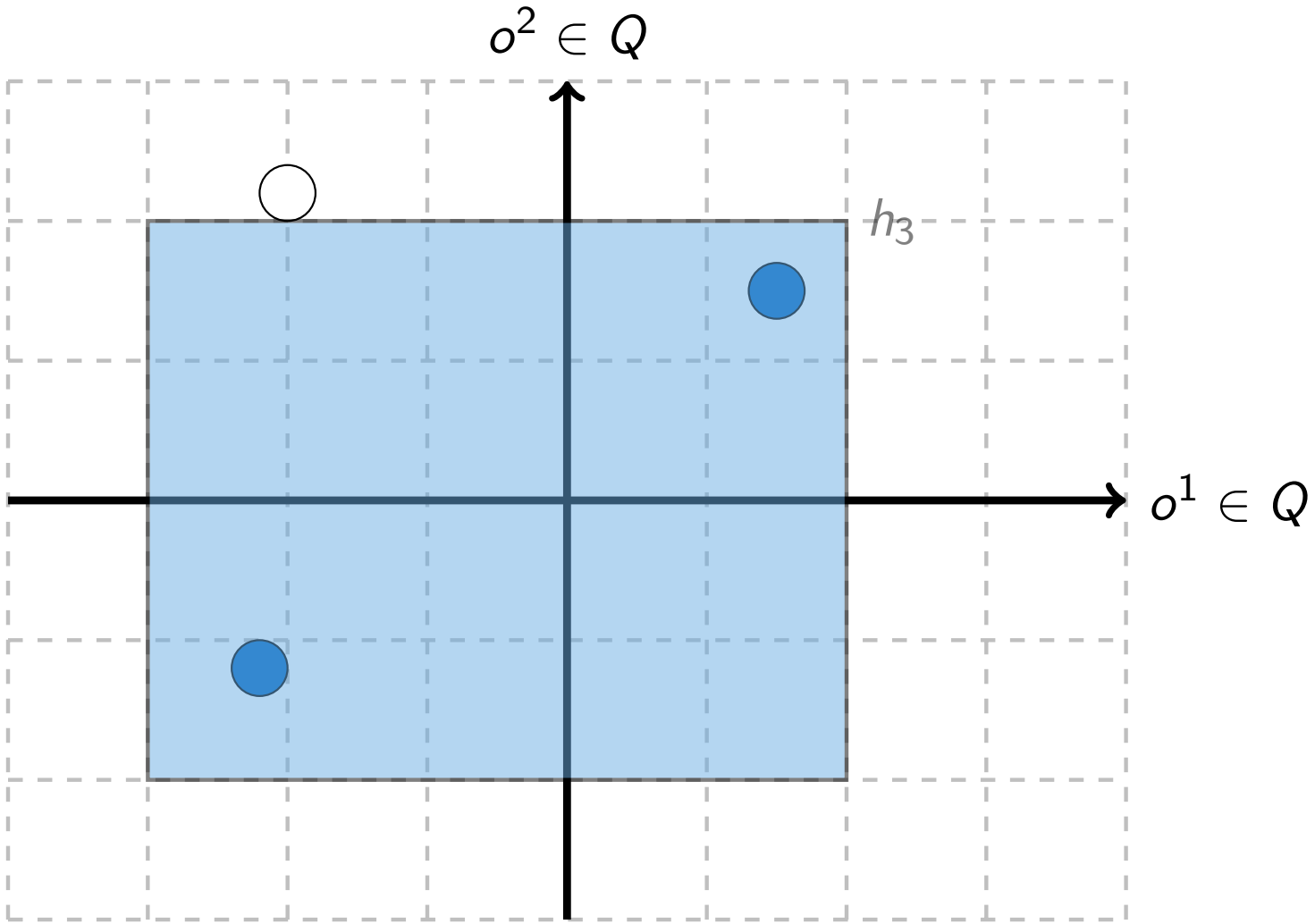


Visualizing Generalization (with Rational Features)



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Visualizing Generalization (with Rational Features)



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Separating agent

Agent's hypothesis is a tuple of integers ('weights') bounded by some fixed $q \in N$, i.e.

$$h_k = [h_k^1, h_k^2, \ldots, h_k^n]$$

where $h_k^i \in \{ 0, 1, \ldots q \}$.

Initial hypothesis:

$$h_1 = (1, 1, \ldots 1)$$

Threshold decisions:

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Separating agent: Update Step

$$h_k = \begin{cases} h_{k-1} \text{ if } r_k = 0 \\ \text{update}(2, h_{k-1}, o_{k-1}) \text{ if } h_{k-1} \cdot o_{k-1} \leq n/2 \\ \text{update}(0, h_{k-1}, o_{k-1}) \text{ if } h_{k-1} \cdot o_{k-1} > n/2 \end{cases}$$

such that for $h_k = \text{update}(\alpha, h_{k-1}, o_{k-1})$

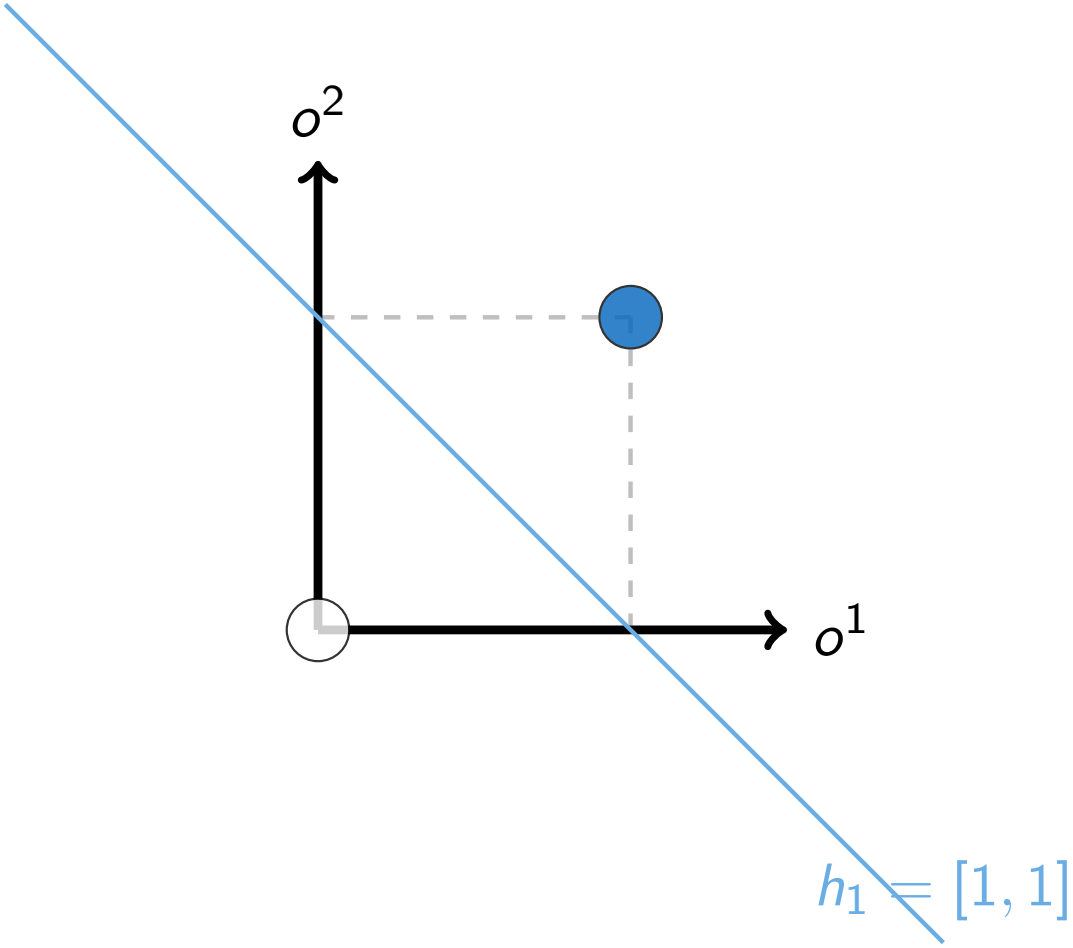
$$h_k^i = \begin{cases} \alpha \cdot h_{k-1}^i \text{ if } o_{k-1}^i = 1 \\ h_{k-1}^i \text{ otherwise} \end{cases}$$

This means

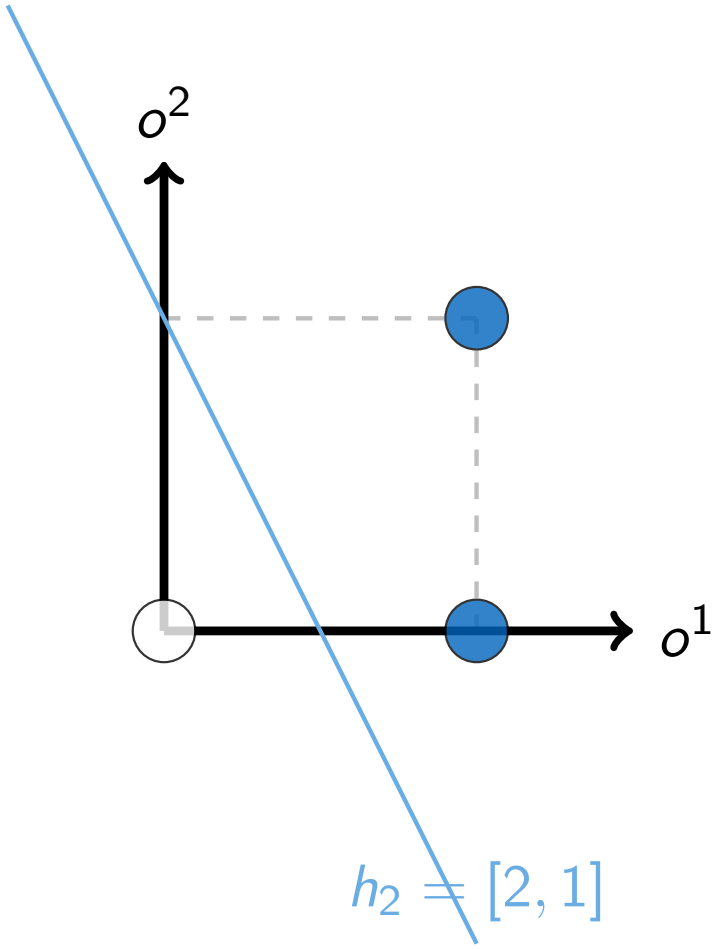
- On a false negative, double weights for 1-features.
- On a false positive, nullify weights for 1-features.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

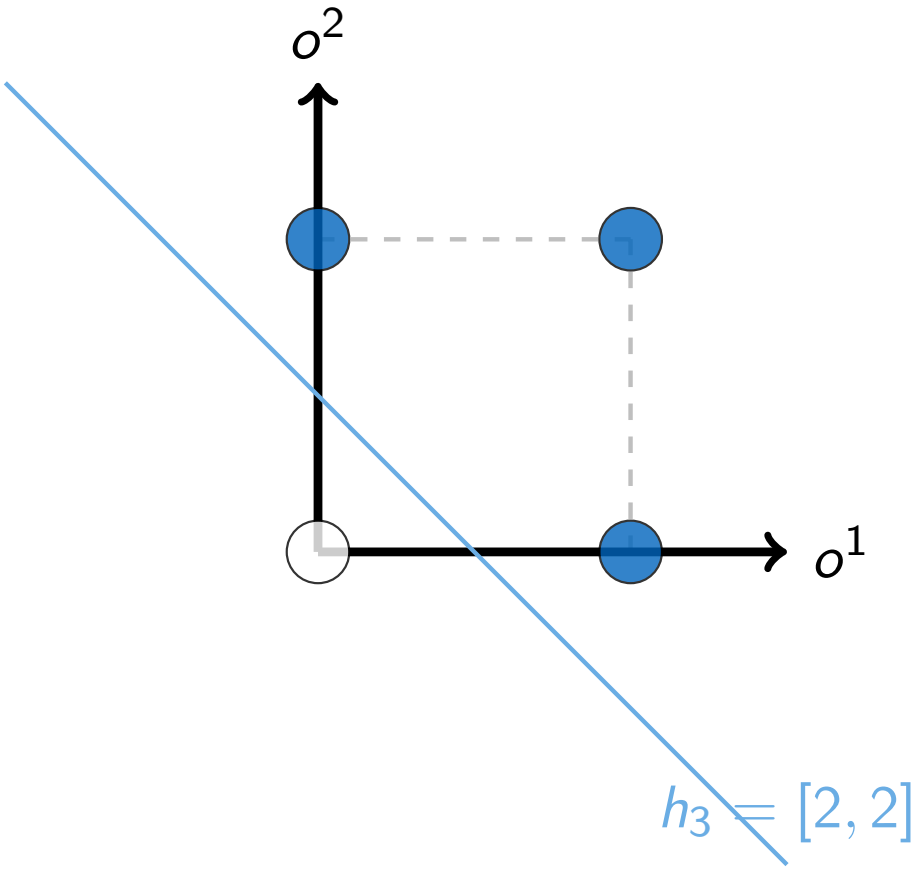
Visualizing Separation



Visualizing Separation



Visualizing Separation



Separating agent: How Well Does It Learn?

Assume the target concept can be expressed through a *disjunction* h^*, i.e. $C(h^*) = C$, and h^* contains no more than s literals.

The agent makes at most $2 + 2s \lg n$ mistakes, i.e. the cumulative reward is

$$\sum_{k=1}^{k'} r_k \geq -2 - 2s \lg n$$

for any horizon $k' \in N$. (proof omitted)

Better than the generalizing agent for sufficiently small s.

Note: adaptation to learning conjunctions, s-CNF, s-DNF, like with the generalizing agent.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Hypothesis and Concept Classes

Denote $\mathcal{H}$ the agent's *hypothesis class*, i.e. set of all hypotheses it can express. For example

- 2^{2n} conjunctions (generalizing, 3^n without contradictions)
- q^n integer tuples (separating)

$\mathcal{H}$ induces the *concept class* $\mathcal{C}(\mathcal{H}) = \{ C(h) \mid h \in \mathcal{H} \}$.

With a finite $\mathcal{H}$, an agent could learn with no more than $|\mathcal{H}| - 1$ errors assuming $C \in \mathcal{C}(\mathcal{H})$:

1. Pick a random $h \in \mathcal{H}$
2. When an error is made using h, delete it from $\mathcal{H}$ and go to 1

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Version Space Agent

Keeps *a set* $\mathcal{H}_k$ of hypotheses (rather than a single one) at each k

$$a_k = (o_k, \mathcal{H}_k)$$

where $\mathcal{H}_1 = \mathcal{H}$ (starting with the full hypothesis class)

At each update, deletes from $\mathcal{H}$ all hypotheses inconsistent with the last observation. For $\mathcal{H}$ consisting of logical formulas:

$$\mathcal{H}_k = \begin{cases} \{\, h \in \mathcal{H}_{k-1} \mid o_{k-1} \models h \,\} & \text{if } e_{k-1} = 1 \\ \{\, h \in \mathcal{H}_{k-1} \mid o_{k-1} \not\models h \,\} & \text{if } e_{k-1} = 0 \end{cases}$$

Note: $e_{k-1} = |y_{k-1} + r_k|$ where $y_{k-1} = \pi(\mathcal{H}_{k-1}, o_{k-1})$, so the above can indeed be evaluated at update time.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Version Space Agent

Decides by a majority vote among all retained hypotheses:

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

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

$$\sum_{k=1}^{k'} r_k \geq -\lg |\mathcal{H}| \tag{1}$$

for any horizon $k' \in N$.

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

On-Line Learning

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 makes at most $poly(n)$ of mistakes. We call $\mathcal{H}$ *learnable on-line* if there is an algorithm (agent) that learns it on-line.

poly is a polynomial and n is the *size* of observations (the number of features in all examples so far.)

So both the generalizing and separating agents learn conjunctions, disjunctions, s-CNF, s-DNF on-line.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

On-Line Learning with Version Space

If $|\mathcal{H}|$ at most exponential then $\lg |\mathcal{H}|$ polynomial and VS agent learns $\mathcal{H}$ online.

What about $\mathcal{H}$ covering all possible concepts on O, i.e.

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

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

Since $|O| = |\{0,1\}^n| = 2^n$, 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 more reasonable hypothesis classes are super-exponential.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Efficient On-Line Learning

Efficient On-Line Learning

An agent that learns hypothesis class $\mathcal{H}$ on-line is said to learn it *efficiently* if it spends at most $poly(n)$ time between the receipt of a percept and the generation of the next action.

Generalizing and separating agents: both take linear time to process a percept, so learn conjunctions, disjunctions, s-CNF and s-DNF efficiently.

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.

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 $\mathrm{VC}(\mathcal{C})$ is the cardinality of the largest subset of O shattered by $\mathcal{C}$.

The VC-dimension of hypothesis class $\mathcal{H}$ is defined as $\mathrm{VC}(\mathcal{C}(\mathcal{H}))$, also denoted $\mathrm{VC}(\mathcal{H})$.

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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 $\left\{ o_1, o_2, \ldots, o_{\mathrm{VC}(\mathcal{C})} \right\} \subseteq O$ 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 $y_1, y_2, \ldots, y_{\mathrm{VC}(\mathcal{C})}$ there is a target concept $C \in \mathcal{C}$ by which all these decisions are wrong.

Corollary: a mistake bound assuming only a hypothesis-based agent with hyp. class $\mathcal{H}$ and $C \in \mathcal{C}(\mathcal{H})$ for the target concept C cannot be smaller than $\mathrm{VC}(\mathcal{H})$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Concept Learning in the Batch Setting

Keeping concept learning assumptions and hypothesis-based agents.

Remind the non-sequential batch setting:

- observations i.i.d. (no adversary selection possible!)
- hypothesis h_K fixed after training phase ending at K
- rewards i.i.d in testing phase, agent maximizes their expectation

$$\sum_{r \in \{-1,0\}} \mu_R(r|h_K) r = -\mu_R(-1|h_K)$$

(note: h_K instead of a_K as only the hypothesis determines decisions)

$\mu_R(-1|h_K)$ is the probability of making a wrong decision in the testing phase and is called the *error* of h_K, denoted $\mathrm{err}(h_K)$, which the agent minimizes.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Training set

In the training phase, the agent receives $m = K - 1$ observations $o_1, o_2, \ldots, o_{K-1}$ whose class e_k can be determined as $e_k = |y_k + r_{k+1}|$. The set

$$\{ (o_1, e_1), (o_2, e_2), \ldots, (o_m, e_m) \}$$

is called the *training set* and its elements are *training examples*.

Since only h_K is relevant for agent's performance in the testing phase, it does not matter if the agent updates h_k at each $k < K$, or computes only h_K from the training set at time K.

Formally: for $k < K$, h_k may be just dummy hypotheses. The training set can be collected as part of the agent's state.

Generalizing Agent in the Batch Setting

Removes all literals inconsistent with training examples from the initially maximal hypothesis (all $2n$ literals).

Denote $\Pr(l)$ the probability that literal l of h is inconsistent with a random observation. h makes a mistake only if it has an inconsistent literal so

$$\text{err}(h) \leq \sum_{l \in h} \Pr(l)$$

There is at most $2n$ literals in h so if $\Pr(l) \leq \epsilon/2n$ ($\epsilon \in \mathbf{R}$) for each literal then $\text{err}(h_k) \leq \epsilon$.

Call a literal *bad* if

$$\Pr(l) > \epsilon/2n$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent in the Batch Setting (cont'd)

Probability that a given bad literal is consistent with all m (i.i.d.!) training examples, and thus present in h_K:

$$(1 - \Pr(l))^m < \left(1 - \frac{\epsilon}{2n}\right)^m$$

Prob. that *some* bad literal consistent with the training set is *at most*:

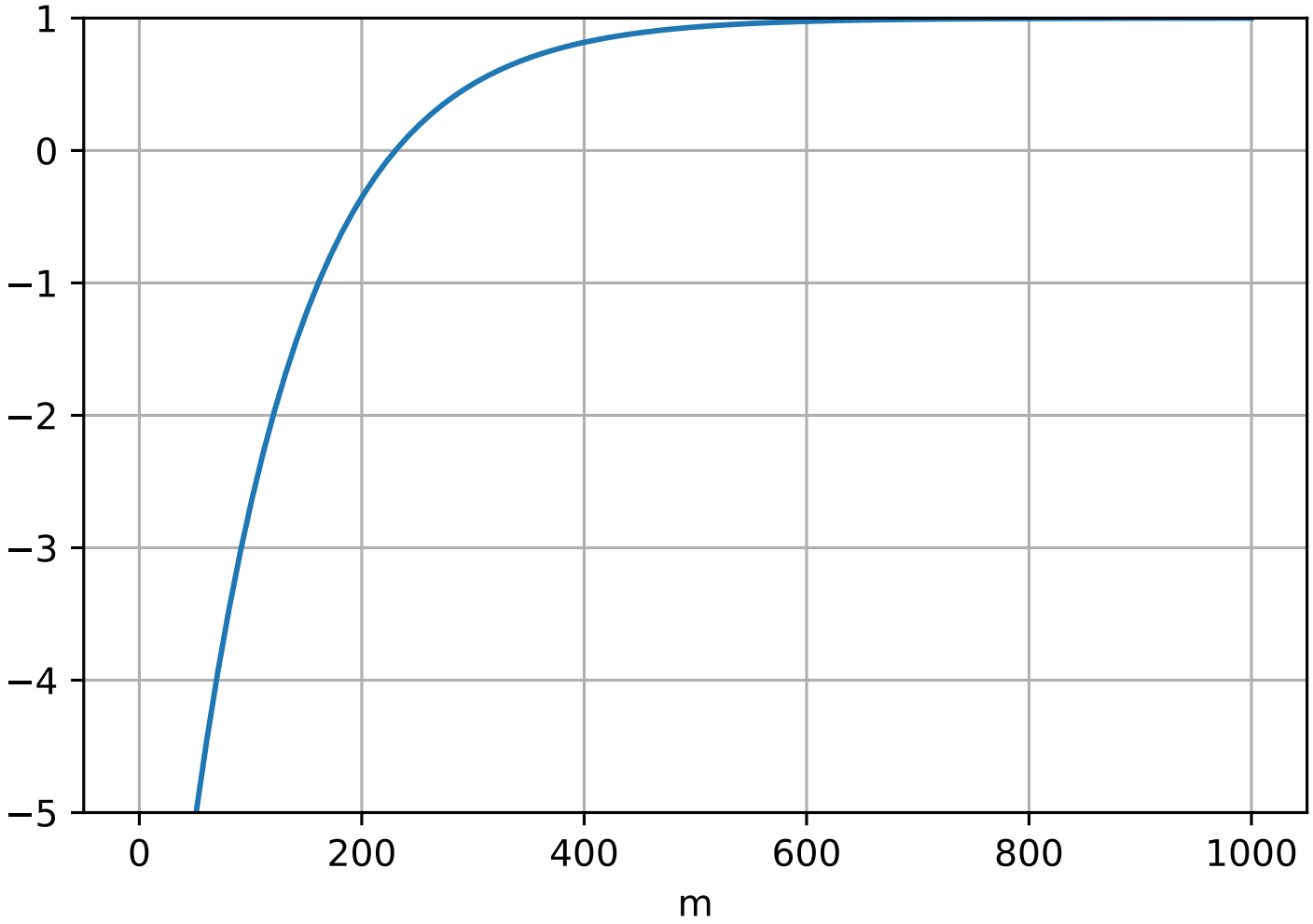
$$2n\left(1 - \frac{\epsilon}{2n}\right)^m \leq 2ne^{-m\frac{\epsilon}{2n}}$$

as there are at most $2n$ bad literals. (We used $1 - x \leq e^{-x}$, $x \in [0; 1]$.)

Otherwise, *with probability at least* $1 - 2ne^{-m\frac{\epsilon}{2n}}$, $err(h_K) \leq \epsilon$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

$1 - 2ne^{-m\frac{\epsilon}{2n}}$, $\epsilon = 0.1, n = 5$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Adapting a Standard On-line Agent for Batch Learning

A *standard* on-line agent uses a single hypothesis for decisions

$$y_k = \pi(h_k, o_k)$$

and $h_{k+1} \neq h_k$ iff $r_{k+1} = -1$, i.e. changes its hypothesis iff it causes a mistake.

Prob. of retaining a *bad* hypothesis ($\mathrm{err}(h_k) > \epsilon$) over q consecutive i.i.d. observations is at most $(1-\epsilon)^q \leq e^{-q\epsilon}$.

If an on-line agent has mistake bound M, set $K = Mq$. Using this agent, some h_k ($k \leq K$) must be retained for q consecutive observations in the training phase. Set h_K to this h_k.

We have an error bound $\mathrm{err}(h_K) \leq \epsilon$ with probability at least $1 - e^{-q\epsilon}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistent Agent

A *consistent agent* sets h_K to any hypothesis $h \in \mathcal{H}$ consistent with the training set.

Can be achieved with the version-space agent, setting h_K to an arbitrary hypothesis from $\mathcal{H}_K$. But that would never be less complex than just collecting the training set and then computing a consistent hypothesis at time K.

Prob. some bad hypothesis $(\text{err}(h) > \epsilon)$ from $\mathcal{H}$ consistent with m training examples at most $|\mathcal{H}|e^{-\epsilon m}$.

So with $m = K - 1$ training samples, we have $err(h_K) \leq \epsilon$ with prob. *at least* $1 - |\mathcal{H}|e^{-\epsilon m}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

The PAC Learning Model

Probably Approximately Correct (PAC) Learning

Agent *probably approximately correctly (PAC) learns* $\mathcal{H}$ if in the batch setting with any target concept $C \in \mathcal{C}(\mathcal{H})$: $K \leq poly(n, 1/\delta, 1/\epsilon)$ and with prob. at least $1 - \delta$: $\text{err}(h_K) \leq \epsilon$. $\mathcal{H}$ is *PAC-learnable* if there is an algorithm (agent) that PAC-learns it.

Recall that $m = K - 1$ is the training set size. Environment tells n, δ, ϵ to agent, agent sets m.

Efficient PAC Learning

Agent *efficiently PAC learns* $\mathcal{H}$ if it PAC-learns it and spends at most $poly(n, 1/\delta, 1/\epsilon)$ time between each percept and successive action in the training phase.

For a consistent agent, this means that a hypothesis consistent with the training set must be computed efficiently.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learning with the Generalization Agent

Prob. that $\text{err}(h_K) > \epsilon$ is at most $2ne^{-m\frac{\epsilon}{2n}}$ as we have estabished. To PAC-learn, we need to make it

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

which is equivalent to

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

So setting $m = \frac{2n}{\epsilon} \ln \frac{2n}{\delta}$ satisfies the inequality. The agent PAC-learns conjunctions since $m \leq poly(n, 1/\delta, 1/\epsilon)$.

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

Through adaptations we have discussed, it also efficiently PAC-learns disjunctions, s-CNF and s-DNF if s is a fixed constant.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learning with Standard On-Line Agents

Consider a standard agent learning $\mathcal{H}$ online. This means it makes at most $M < poly(n)$ mistakes in the on-line setting.

We have shown how to adapt this agent for the batch setting so that $\text{err}(h_K) \leq \epsilon$ with probability at least $1 - e^{-q\epsilon}$ where $K = Mq$.

Set $q = \frac{1}{\epsilon} \ln \frac{1}{\delta}$ to satisfy PAC-learning requirement:

$$1 - e^{-q\epsilon} = 1 - e^{-\epsilon \frac{1}{\epsilon} \ln \frac{1}{\delta}} = 1 - \delta$$

Since both M and q are $\leq poly(n, 1/\delta, 1/\epsilon)$, so is $K = Mq$.

So any standard agent learning (efficiently) $\mathcal{H}$ online has a counter-part (efficiently) PAC-learning $\mathcal{H}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

PAC-Learning with a Consistent Agent

For a consistent agent using $\mathcal{H}$, we have shown $\text{err}(h_K) > \epsilon$ with prob. at most $|\mathcal{H}|e^{-\epsilon m}$. For PAC-learning we need

$$|\mathcal{H}|e^{-\epsilon m} < \delta$$

The smallest m satisfying the inequality is

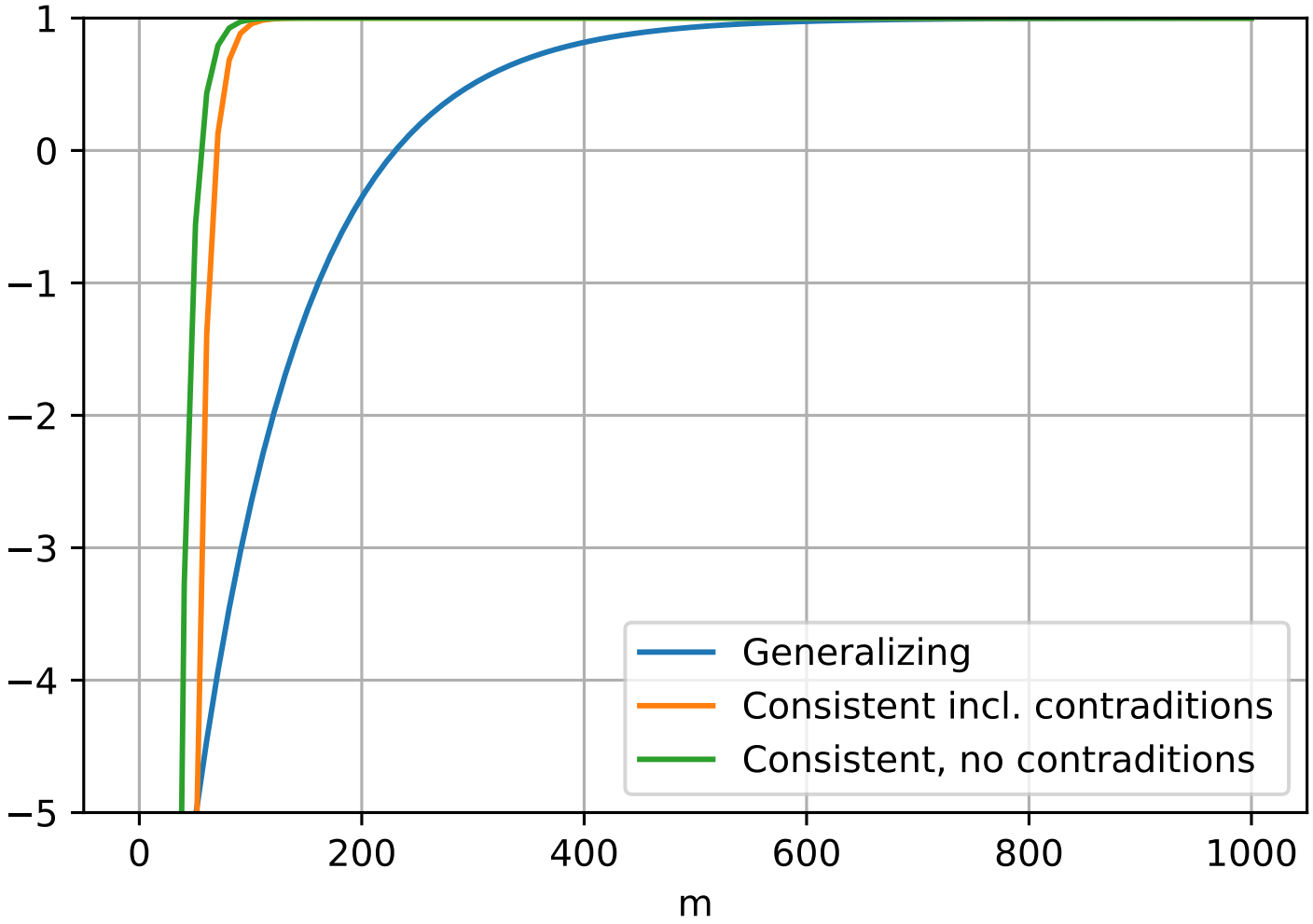
$$m = \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$$

$K = m + 1$ is poly in $1/\delta$ and $1/\epsilon$.

It is also poly in n *if* $|\mathcal{H}|$ *is at most exponential in* n and then the agent PAC-learns $\mathcal{H}$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

L-bound for $\Pr(\text{err}(h_K) \leq 0.1)$ for $\mathcal{H}$=conjunctions, $n = 5$



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Some More $\mathcal{H}$ Classes

s-conjunctions, s-disjunctions. At most s literals. $|\mathcal{H}|$ is of $\mathcal{O}\left[3^s\right]$ (non-self-resolving)

s-term DNF, s-clause CNF. At most s non-self-resolving terms (clauses) of unlimited size. $|\mathcal{H}|$ is of

$$\mathcal{O}\left[\binom{3^n}{s}\right] = \mathcal{O}\left[(3^n)^s\right]$$

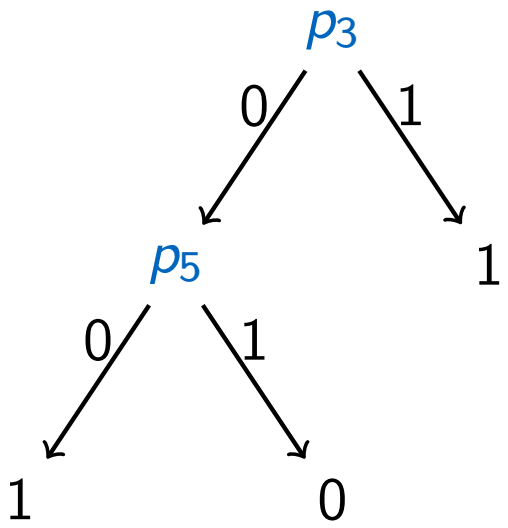
general DNF, CNF. $|\mathcal{H}| = 2^{(3^n)}$. Recall there are only $2^{(2^n)}$ possible concepts on $O = \{0,1\}^n$, so $\mathcal{H}$ has equivalent pairs in it.

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

s-depth decision trees. See next slide.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Decision Trees with max depth s



Depth 3

Denote $c(s) = |\mathcal{H}|$ where $\mathcal{H}$ contains trees with max depth s.

$$c(1) = 2$$

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

$$c(s) = n \cdot c(s-1)^2$$

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

$$c(s) = 2(2n)^{2^s - 1}$$

which is poly in n.

Sizes of Some $\mathcal{H}$

$\mathcal{H}$	$\lvert\mathcal{H}\rvert$ in increasing size	
s-conjunctions, s-disjunctions	$(2n)^s$ (incl. self-resolving)	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

Exp or poly implies PAC-learnability.

Poly implies *efficient* PAC-learnablity if single consistency check efficient.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

To PAC-Learn, Agent **Must** Be Consistent

Consider an agent unable to find a hypothesis consistent with a training set containing m observations $O' \subseteq O$. Environment can be such that

$$\mu_O(o) = \frac{1}{m}$$

for all $o \in O'$ (note: $\mu_O(o) = \mu_O(o|0) + \mu_O(o|1)$) and

$$\epsilon < \frac{1}{m+1}$$

If h_K inconsistent on a single $o \in O'$ then

$$\mathrm{err}(h_K) \geq \frac{1}{m} > \epsilon$$

with probability $1 \geq \delta$. So to PAC-learn efficiently, consistent hypothesis must be found in poly time.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Sidenote: Empirical Risk Minimization

Learning a consistent hypothesis may not be possible or reasonable, e.g.

- $C \notin \mathcal{C}(\mathcal{H})$ (target concept not expressible through a hypothesis)
- $\mu(o|0)\mu(o|1) > 0$ (not concept learning, not crisp classification)
- a consistent h would be too complex

ERM principle: produce $h \in \mathcal{H}$ 'most consistent' with training set T

$$h = \arg\min_{h \in \mathcal{H}} \widehat{\mathrm{err}}(h, T) = \arg\min_{h \in \mathcal{H}} \frac{1}{m} |\{ (o, e) \in T \mid \pi(h, o) \neq e \}|$$

$\widehat{\mathrm{err}}(h, T)$ is the *training error* or *empirical risk*.

No longer PAC-learning, but bounds on $\mathrm{err}(h)$ can be estimated from $\widehat{\mathrm{err}}(h, T)$ and properties of $\mathcal{H}$. See the Statistical Machine Learning class.

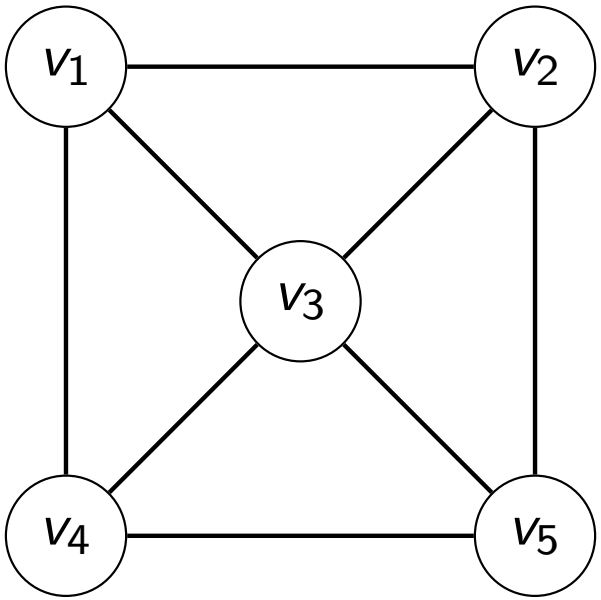
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

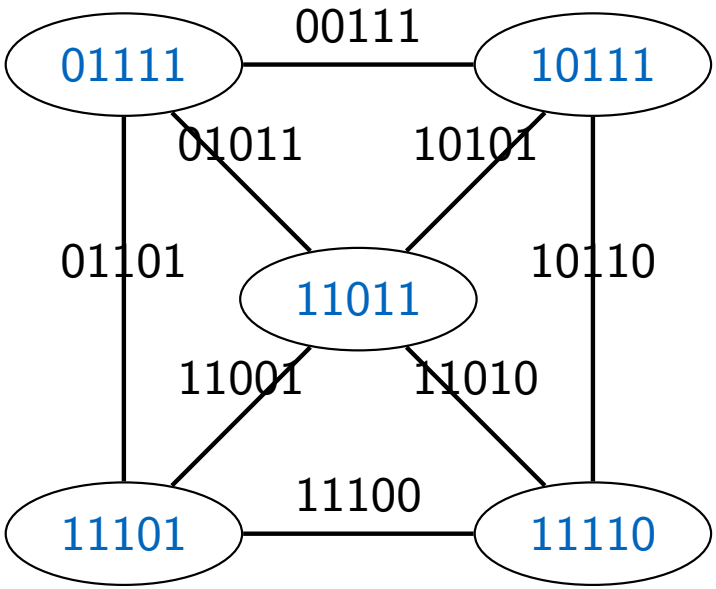
Finding a Consistent s-term DNF

Graph 3-colorability can be reduced in poly time to a training set.

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

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

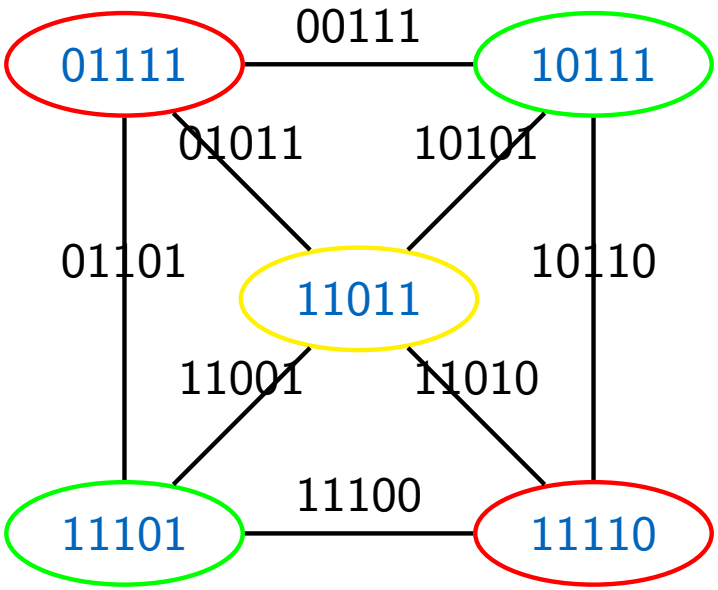




FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

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



$$p_2 \wedge p_3 \wedge p_4 \quad \text{(all non-red)}$$
$$\vee$$
$$p_1 \wedge p_3 \wedge p_5 \quad \text{(all non-green)}$$
$$\vee$$
$$p_1 \wedge p_2 \wedge p_4 \wedge p_5 \quad \text{(all non-yellow)}$$

3-colorability NP-hard $\rightarrow$ finding a consistent 3-term DNF NP-hard. Can be generalized to s-term DNF. They are not *efficiently* PAC-learnable.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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

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

$$(p_1 \wedge p_2) \vee (p_2 \wedge p_3) \equiv (p_1 \vee p_2) \wedge (p_1 \vee p_3) \wedge p_2 \wedge (p_2 \vee p_3)$$

So s-term DNF $\subseteq$ s-CNF. Thus by using $\mathcal{H} = s$-CNF, the agent can PAC-learn even though $\mathcal{C} = s$-term DNF.

s-CNF ($\mathcal{O}\left[2^{(n^s)}\right]$) is larger than s-term DNF ($\mathcal{O}\left[(3^n)^s\right]$).

So s-term DNF $\subset$ s-CNF. Thus the learned s-CNF may not be expressible as an s-term DNF.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

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 PAC-learns $\mathcal{H}$ if $\mathrm{VC}(\mathcal{H})$ is at most polynomial.

$\mathrm{VC}(\mathcal{H})$ is "analogical" to $\ln|\mathcal{H}|$.

Very useful for infinite or large finite $\mathcal{H}$ with limited "shattering potential."

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. Values of $m = \frac{1}{\epsilon} \ln \frac{|\mathcal{H}|}{\delta}$ for some b and $\epsilon = \delta = 0.1$:

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

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

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Determining VC-Dimension

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

Example: $\mathcal{H}$ half-planes in (a finite subset of) $\mathbf{R}^2$.

- *Some* 3 points can be shattered (obvious) so $\mathrm{VC}(\mathcal{H}) \geq 3$.
- *No* 4 points can be shattered. Obvious if 3 in line. Otherwise two cases possible and impossible labeling exists in each. So $\mathrm{VC}(\mathcal{H}) < 4$.

Thus $\mathrm{VC}(\mathcal{H}) = 3$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Structured Observations

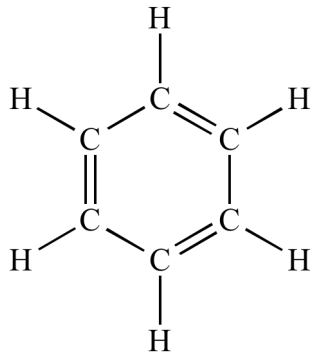
Our observations so far had a simple form

o_1	o_2	...	o_n
0	1	...	0

We considered extensions beyond binary

o_1	o_2	...	o_n
120	5	...	14

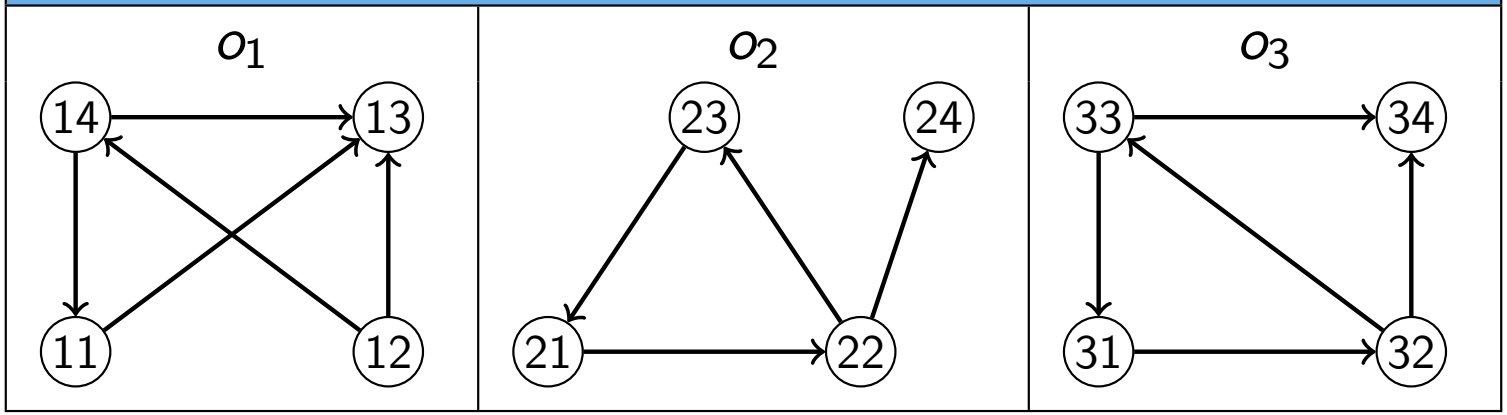
but still flat. What if observations are structures such as



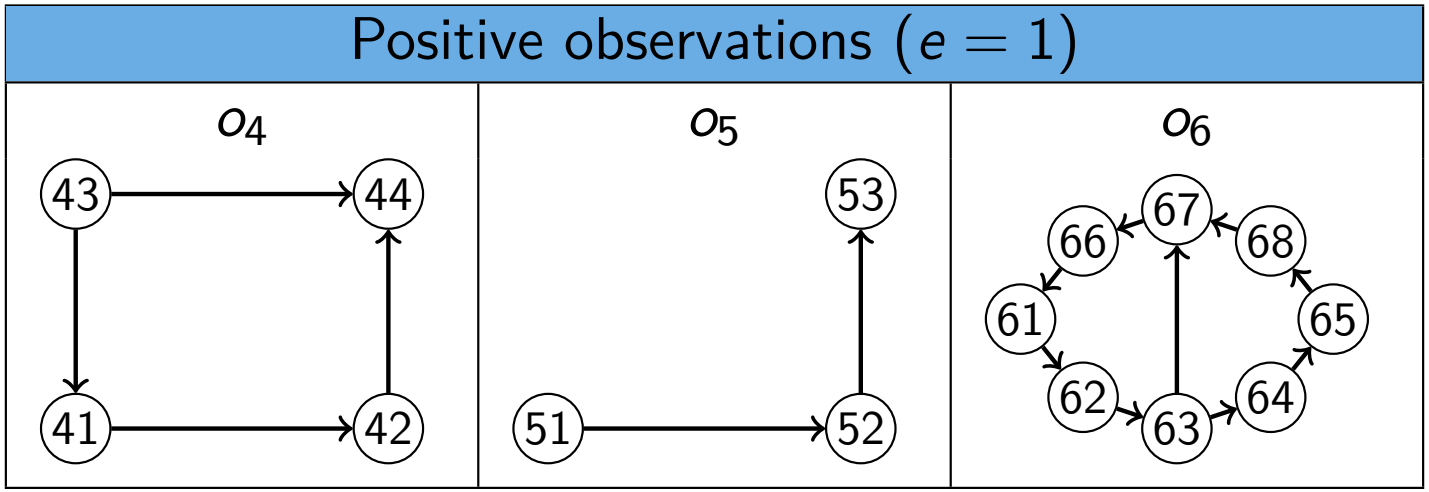
Here we need to learn from *graphs*, not tuples.

Observations as Oriented Graphs: Example

Negative observations ($e = 0$)

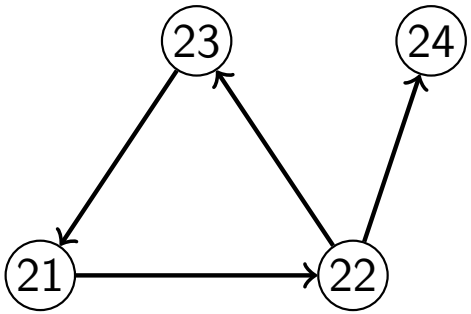


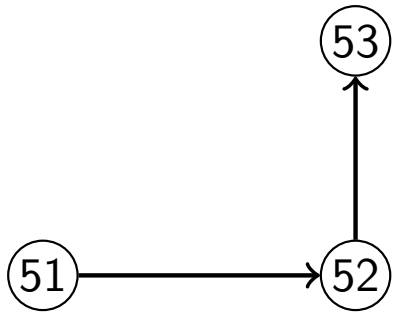
Positive observations ($e = 1$)



FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Graphs as Herbrand Interpretations





$$\left\{ \begin{array}{l} \texttt{edge}(21,22), \texttt{edge}(22,23), \\ \texttt{edge}(23,21), \texttt{edge}(23,24) \end{array} \right\} \qquad \{ \texttt{edge}(51,52), \texttt{edge}(52,53) \}$$

A *Herbrand interpretation* gives truth values to all possible ground atoms by listing exactly the true ones.

E.g., $\texttt{edge}(23,24)$ is *false* in the interpretations above.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Logic Language (Reminder)

A first-order logic language given by

- P: set of *predicate* symbols with associated arity. E.g. edge/2.
- F: set of *function* symbols with associated arity. Zero-arity functions, e.g. 51, are *constants*.

A *function* consists of a function symbol and a_f terms (a_f = functions's arity). E.g. 51 or label(51) or label(x).

A *term* is either a variable ($x, y, \ldots$) or a function.

A first-order *atom* consists of a predicate symbol and a_p terms (a_p = predicate's arity). A first-order *literal* is a first-order atom or its negation.

A *ground* term / atom / literal / clause: contains *no variables*.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order Clauses (Reminder)

A first-order clause γ is a *universally quantified* disjunction of first-order literals, e.g.

$$\gamma = \forall x, y : \text{edge}(x, y) \vee \text{edge}(y, x)$$

The default quantifiers (here $\forall x, y :$) may be omitted.

A *substitution* θ is an assignment of terms to variables. E.g. for $\theta = \{ x \to 51 \}$.

$$\gamma\theta = \text{edge}(51, y) \vee \text{edge}(y, 51)$$

$\gamma\theta$ is a *ground instance* of clause γ if θ maps *all* variables of γ to ground terms. E.g. $\theta = \{ x \to 51, y \to 52 \}$

$$\gamma\theta = \text{edge}(51, 52) \vee \text{edge}(52, 51)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Herbrand Models

A Herbrand interpretation may be a *model* of a ground clause, e.g.

$$\{ \text{edge}(51, 52), \text{edge}(52, 53) \} \models \text{edge}(51, 52) \vee \text{edge}(52, 51)$$

which is analogical to propositional logic: e.g. valuation $[1, 1, 0]$ for vars p_1, p_2, p_3 can be written $\{ p_1, p_2 \}$, and

$$\{ p_1, p_2 \} \models p_1 \vee p_3$$

A H.I. is a model of a (non-ground) clause iff it is a model of *all its ground instances*, so for $F = \{ 51, 52, 53 \}$:

$$\{ \text{edge}(51, 52), \text{edge}(52, 53), \text{edge}(51, 53) \} \models \text{edge}(x, y) \vee \text{edge}(y, x)$$

but

$$\{ \text{edge}(51, 52), \text{edge}(52, 53) \} \not\models \text{edge}(x, y) \vee \text{edge}(y, x)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Agent Using a First-Order Hypothesis

Consider $\mathcal{H}$ = first-order logic formulas and O = Herbrand interpretations. (Both using language P, F)

The agent decides analogically to its propositional-logic counterpart:

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

We will assume h to be a clause or a conjunction of clauses (CNF).

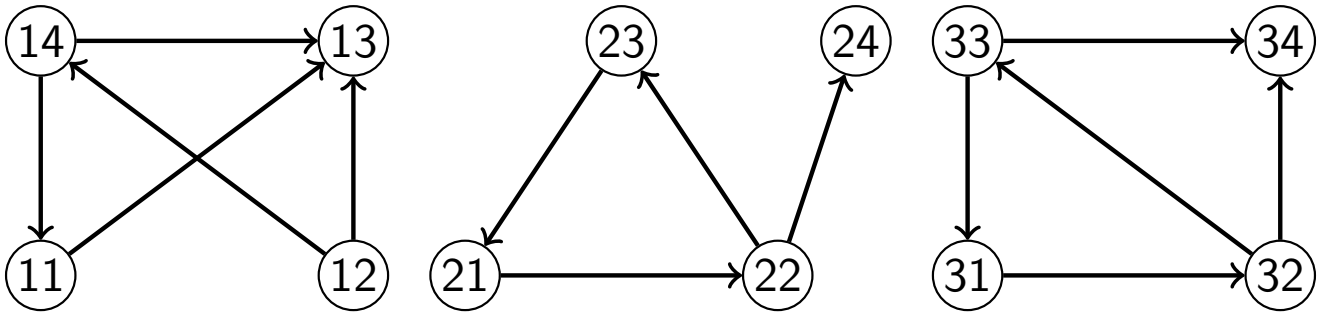
Interpretation o is a model of a CNF if it is a model of all its clauses.

How to learn h in the current example?

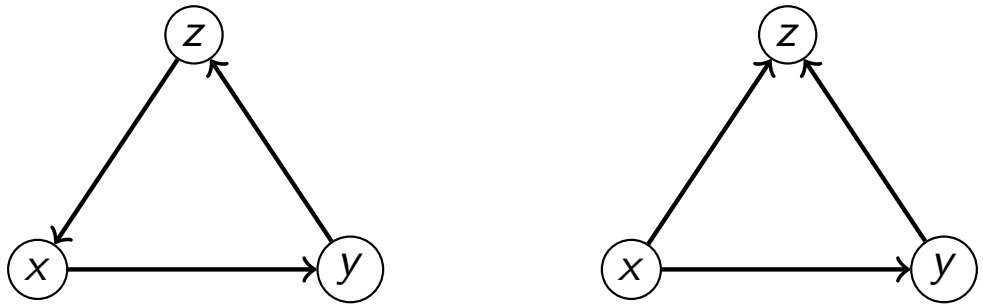
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

A Discriminating Pattern

Agent should 'notice' that all the negative observations (o_1, o_2, o_3)



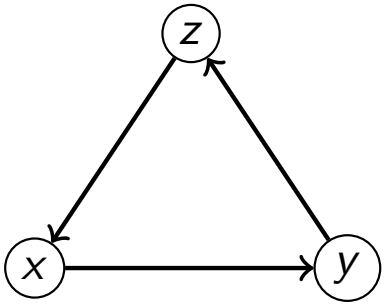
contain one of two kinds of triangle



and none of the positive observations does.

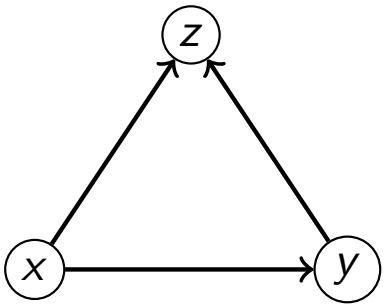
A Discriminating Hypothesis

Hypothesis should say: "none of the two triangle kinds included."



Conjunction $\exists x, y, z : \mathrm{edge}(x, y) \wedge \mathrm{edge}(y, z) \wedge \mathrm{edge}(z, x)$ is true when the triangle is included. Its negation is the clause

$$\gamma_1 = \forall x, y, z : \neg\mathrm{edge}(x, y) \vee \neg\mathrm{edge}(y, z) \vee \neg\mathrm{edge}(z, x)$$



Similarly for the second triangle

$$\gamma_2 = \forall x, y, z : \neg\mathrm{edge}(x, y) \vee \neg\mathrm{edge}(y, z) \vee \neg\mathrm{edge}(x, z)$$

We want to learn CNF $\gamma_1 \wedge \gamma_2$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent for First-Order CNF

We design an agent learning conjunctions of *st-clauses*. An *st*-clause has

- no more than $s \in N$ literals
- no more than $t \in N$ occurrences of predicate, function, and variable symbols.

Fix s, t and denote Γ the set of all s, t-clauses made with language P, F.

The size of the learning task is defined as $n, |P|, |F|$, where n is the cardinality of the largest observation.

Recall the propositional agent where the task size was just the dimension of the observation tuples.

Where we assumed $poly(n)$, now we assume $poly(n, |P|, |F|)$.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Generalizing Agent for First-Order CNF (cont'd)

Agent follows the usual generalization strategy. Starts with the conjunctions of all *st*-clauses

$$h_1 = \bigwedge_{\gamma \in \Gamma} \gamma$$

and on each error, it deletes from h_k all clauses inconsistent with the misclassified observation o_k:

$$h_{k+1} = \text{delete}(h_k, o_k) = \text{delete}\left(\bigwedge_{i \in I} \gamma_i, o_k\right) = \bigwedge_{\substack{i \in I \\ o_k \models \gamma_i}} \gamma_i$$

What is the mistake bound in the on-line setting?

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order CNF Generalizing: Mistake Bound

No more than $|\Gamma|$ mistakes. (Apply same reasoning as with the propositional generalizing agent.) Is $|\Gamma| = poly(n, |P|, |F|)$?

At most $|P|(|F| + st)^{t-1}$ different atoms in the language because

- $|P|$ choices of predicate
- at most $t - 1$ other symbols chosen out of $|F| + st$ symbols:
  - $|F|$ function symbols (incl. constants)
  - at most st different variables

So at most $2|P|(|F| + st)^{t-1}$ different literals. An st-clause combines at most s literals so

$$|\Gamma| = \mathcal{O}\binom{2|P|(|F| + st)^{t-1}}{i} = poly(n, |P|, |F|)$$

So the agent learns CNF's of st-clauses online (implies PAC-learning!)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

First-Order CNF Generalizing: Efficiency

Efficiency depends on the complexity of the consistency check $o_k \models \gamma_i$ in the delete function

Remind that $o \not\models \gamma$ iff there is a substitution θ such that the atoms of

1. all negative literals of $\gamma\theta$ are in o
2. no positive literal of $\gamma\theta$ is in o

Finding such a θ has exponential worst-case complexity. But it can be made polynomial by another assumption on the clauses.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Range Restriction

A clause is *range-restricted* if all variables in its positive literals are also in some of its negative literals.

Example:

$$\gamma = \neg\mathtt{edge}(x, y) \vee \neg\mathtt{edge}(y, z) \vee \mathtt{path}(x, z)$$

or equivalently

$$\gamma = \mathtt{edge}(x, y) \wedge \mathtt{edge}(y, z) \rightarrow \mathtt{path}(x, z)$$

is a range-restricted $3, 3$-clause.

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistency Check for Range-Restricted Clauses

Consequence of range restriction: if θ grounds all negative literals it also grounds all positive literals of the same clause.

So checking $o \not\models \gamma$ has two steps

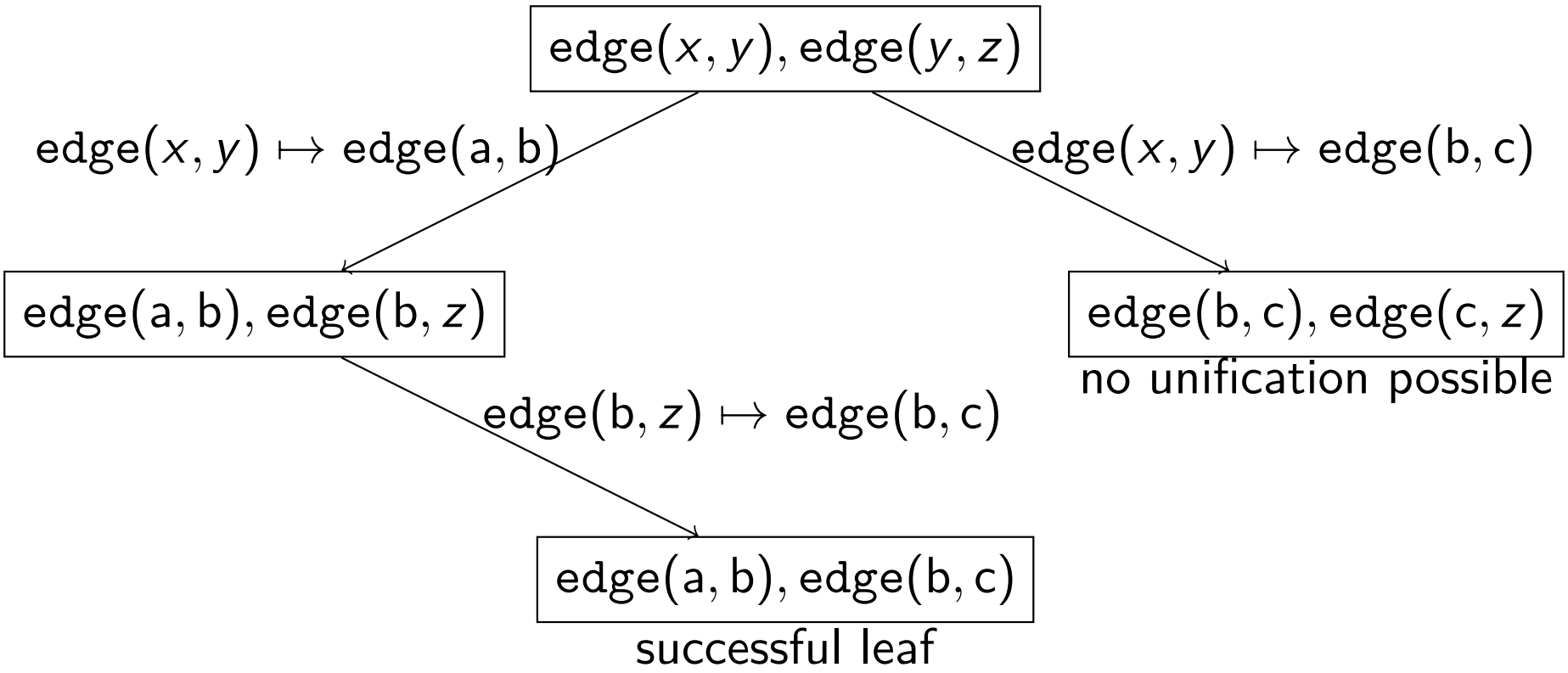
1. Find θ such that all negative literals of $\gamma\theta$ are in o
2. Check that no positive literal of $\gamma\theta$ (they are all ground!) is in o.

Step 2 is linear-time (no search). Step 2 can be arranged as a tree-search. Consider checking

$$\{ \texttt{edge}(\text{a}, \text{b}), \texttt{edge}(\text{b}, \text{c}), \texttt{path}(\text{a}, \text{c}) \}$$
$$\not\models$$
$$\texttt{edge}(x, y) \wedge \texttt{edge}(y, z) \rightarrow \texttt{path}(x, z)$$

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE

Consistency Check for Range-Restricted Clauses (cont'd)



At most n^s vertices, at most t arguments in each. So search takes at most tn^s time, which is $poly(n, |P|, |F|)$. The agent *learns range-restricted st-clauses efficiently* on-line. (Implies efficient PAC-learning!)

FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE